Full Abstraction for Polymorphic Pi-Calculus

Alan Jeffrey*[1] and Julian Rathke[2]

[1] Bell Labs, Lucent Technologies, Lisle, IL, USA
[2] University of Sussex, Brighton, UK

August 2006, To appear in *Theoretical Computer Science*

Abstract. The problem of finding a fully abstract model for the polymorphic π-calculus was stated in Pierce and Sangiorgi's work in 1997 and has remained open since then. In this paper, we show that a variant of their language has a fully abstract model, which does not depend on type unification or logical relations. This is the first fully abstract model for a polymorphic concurrent language. In addition, we discuss the relationship between our work and Pierce and Sangiorgi's, and show that their model based on type unification is sound but not complete.

1 Introduction

Finding sound and complete models for languages with polymorphic types is notoriously difficult. Consider the following implementation of a polymorphic 'or' function in Java 5.0 [16]:

```
static<X> X or (X t, X a, X b) {
  if (a == t) { return a; } else { return b; }
}
```

This implementation of `or` takes a type parameter `X`, which will be instantiated with the representation chosen for the booleans, together with three parameters of type `X`: a constant for 'true', and the values to be 'or'ed. This function can be called in many different ways, for example[3]:

```
or.<int> (1, 0, 1); or.<bool> (true, false, true);
```

In each case, there is no way for the callee to determine the exact type the caller instantiated for `X`, and so *no matter what implementation for* `or` *is used*, there is no observable difference between the above program and the following:

```
or.<int> (1, 0, 1); or.<string> ("true", "false", "true");
```

or the following:

* This material is based upon work supported by the National Science Foundation under Grant No. 0430175

[3] Java purists should note that this discussion assumes for simplicity that downcasting and reflection are not being used, and a particular implementation of autoboxing, for example the code `or.<int> (1, 0, 1)` is implemented as `Integer x = new Integer(1); Integer y = new Integer(0); or.<Integer> (x, y, x)`.

```
or.<int> (1, 0, 1); or.<int> (2, 3, 2);
```

However, there *is* an observable difference between the above programs and:

```
or.<int> (1, 0, 1); or.<int> (1, 0, 1);
```

since we can use the following implementation of `or` to distinguish them:

```
static Object x=null;
static<X> X or (X t, X a, X b) {
  if (a == x) { System.out.println ("hello"); } else { x=a; }
  if (a == t) { return a; } else { return b; }
}
```

This example demonstrates some subtleties with polymorphic languages: the presence of impure features (such as mutable fields in this case) and equality testing (such as `a == x` in this case) can significantly impact the distinguishing power of tests. In the case of pure languages such as System F [10], the technique of *logical relations* [26, 23] can be used to establish equivalence of all of the above calls to `or`, which is evidently broken by the addition of impurity and equality testing.

Much of the work in finding models of pure polymorphic languages comes in finding appropriate techniques for modelling *parametricity* [25, 26] to show that programs are completely independent of the instantiations for their type parameters. Such parametricity results are surprisingly strong, and can be used to establish 'theorems for free' [30] such as the functoriality of the list type constructor. The strength of the resulting theorems, however, comes at a cost: the proof techniques required to establish them are quite difficult. In particular, even proving the existence of logical relations is problematic in the presence of recursive types [23].

In this paper, we show that providing models for impure polymorphic languages with equality testing can be surprisingly straightforward, (albeit with some subtlety of choice of language features, as discussed in Section 4). We believe that the techniques discussed here will extend to the polymorphic features of languages such as Java 5.0 [16], and C# 2.0 [7]: F-bounded polymorphism [5], subtyping, recursive types and object features. In this paper, we will investigate a minimal impure polymorphic language with equality testing and mismatch, based on Pierce and Sangiorgi's work [22] on a polymorphic extension of Milner *et al.*'s [20, 19] π-calculus.

Pierce and Sangiorgi have established a sound model for a polymorphic π-calculus, but they only conjectured the existence of a complete model [22, Sec. 12.2]. In this paper, we develop a sound and complete model for a polymorphic π-calculus: the resulting model and proof techniques are quite simple. In particular, our model makes no use of type unification, which is an important feature of Pierce and Sangiorgi's model. We then compare our model to theirs, and show that ours is strictly finer: hence we have resolved their outstanding conjecture, by demonstrating their model to be sound but not complete.

This is the first sound and complete model for a polymorphic π-calculus: Pierce and Sangiorgi [22] and Honda *et al.* [3] have established soundness results, but not completeness.

We would like to thank the anonymous referees for their hard work and detailed comments: this paper is significantly improved by their effort.

$$
\begin{array}{rll}
a,b,c,d & & \text{(Names)} \\
x,y,z & & \text{(Variables)} \\
n,m & ::= a \mid x & \text{(Values)} \\
P,Q,R & ::= n(\vec{X};\vec{x}:\vec{T})\,.\,P \mid \overline{n}\langle\vec{T};\vec{n}\rangle \mid \mathbf{0} \mid P \mid Q & \text{(Processes)} \\
 & \quad \mid \nu(a:T)P \mid\, !P \mid \mathsf{if}\, n = m\, \mathsf{then}\, P\, \mathsf{else}\, Q &
\end{array}
$$

Fig. 1. Syntax

2 An Asynchronous Polymorphic Pi-Calculus

The language we investigate in this paper is an asynchronous variant of Pierce and Sangiorgi's polymorphic π-calculus. This is an extension of the π-calculus with type-passing in addition to value-passing.

2.1 Syntax

The syntax of the asynchronous polymorphic π-calculus is given in Figure 1. The syntax makes use of types (ranged over by T,U,V,W) and type variables (ranged over by X,Y,Z), which are defined in Section 2.3.

Definition 1 (Free identifiers). *Write* $\mathsf{fn}(P)$ *for the free names of P,* $\mathsf{fn}(n)$ *for the free names of n,* $\mathsf{fv}(P)$ *for the free variables of P,* $\mathsf{fv}(n)$ *for the free variables of n,* $\mathsf{ftv}(P)$ *for the free type variables of P and* $\mathsf{ftv}(T)$ *for the free type variables of T.*

Definition 2 (Substitution). *Let* σ *be a substitution of the form* $(\vec{V}/\vec{X};\vec{n}/\vec{x})$*, and let* $n[\sigma]$*,* $T[\sigma]$ *and* $P[\sigma]$ *be defined to be the result of applying the capture-free substitution of type variables* $\vec{X}$ *by types* $\vec{V}$ *and variables* $\vec{x}$ *by values* $\vec{n}$*, defined in the normal fashion. Let the domain of a substitution* $\mathsf{dom}(\sigma)$ *be defined as* $\mathsf{dom}(\vec{V}/\vec{X};\vec{n}/\vec{x}) = \{\vec{X},\vec{x}\}$.

Definition 3 (Process contexts). *A process context* $\mathcal{C}[\cdot]$ *is a process containing one occurrence of a 'hole'* $(\cdot)$*. Write* $\mathcal{C}[P]$ *for the process given by replacing the hole by P.*

We present an example process, following [22], in the untyped π-calculus, in which we implement a boolean abstract datatype as:

$$\nu(t)\nu(f)\nu(test)(\overline{getBools}\langle t,f,test\rangle \mid !t(x,y)\,.\,\overline{x}\langle\rangle \mid !f(x,y)\,.\,\overline{y}\langle\rangle \mid !test(b,x,y)\,.\,\overline{b}\langle x,y\rangle)$$

This process generates new channels t, f and $test$, which it publishes on a public channel $getBools$. It then waits for input on channel t: when it receives a pair (x,y) of channels, it sends a signal on x. The same is true for channel f except that it sends the signal on y. Finally, on a test channel we wait to be sent a boolean b (which should either be t or f) together with a pair $(x.y)$ of channels, and just forwards the pair on to b, which chooses whether to signal x or signal y as appropriate. This can be typed as:

$$
\begin{array}{rl}
B_1 \stackrel{\mathsf{def}}{=} & \nu(t:Bool)\nu(f:Bool)\nu(test:Test(Bool))(\\
 & \quad \overline{getBools}\langle Bool;t,f,test\rangle \mid \\
 & \quad !t(x:Signal,y:Signal)\,.\,\overline{x}\langle\rangle \mid \\
 & \quad !f(x:Signal,y:Signal)\,.\,\overline{y}\langle\rangle \mid \\
 & \quad !test(b:Bool,x:Signal,y:Signal)\,.\,\overline{b}\langle x,y\rangle \\
 &)
\end{array}
$$

where we define:

$$Signal \stackrel{\mathsf{def}}{=} \updownarrow[] \quad Bool \stackrel{\mathsf{def}}{=} \updownarrow[Signal, Signal] \quad Test(T) \stackrel{\mathsf{def}}{=} \updownarrow[T, Signal, Signal]$$

The interesting typing is for the channel *getBools* where the implementation of booleans is published:

$$getBools : \updownarrow[X; X, X, Test(X)]$$

that is, the implementation type *Bool* is never published: instead we just publish an abstract type X together with the values $t : X$, $f : X$ and $test : Test(X)$. Since the implementing type is kept abstract, we should be entitled to change the implementation without impact on the observable behaviour of the system, for example by uniformly swapping the positions of x and y in outputs:

$$\begin{array}{rl} B_2 \stackrel{\mathsf{def}}{=} & \nu(t : Bool)\nu(f : Bool)\nu(test : Test(Bool))(\\ & \overline{getBools}\langle Bool; t, f, test\rangle \mid \\ & !t(x : Signal, y : Signal) . \bar{y}\langle\rangle \mid \\ & !f(x : Signal, y : Signal) . \bar{x}\langle\rangle \mid \\ & !test(b : Bool, x : Signal, y : Signal) . \bar{b}\langle y, x\rangle \\ ) & \end{array}$$

As Pierce and Sangiorgi observe, as untyped processes B_1 and B_2 are easily distinguished, for example by the testing context:

$$T \stackrel{\mathsf{def}}{=} \cdot \mid \nu(a)\nu(b)(getBools(t, f, test) . \bar{t}\langle a, b\rangle \mid a() . \bar{c}\langle\rangle \mid b() . \bar{d}\langle\rangle)$$

However, this process does not typecheck, since when we come to typecheck T, the channel t has abstract type X, not the implementation type *Bool*. We expect any sound and complete model to consider B_1 and B_2 equivalent.

An illustrative example of a contextual inequivalence is given below. For some generative type T (that is, T is not a type variable) consider the following processes:

$$\begin{array}{l} L \;= \nu(b : \updownarrow[T], c : \updownarrow[T], d : T)(\bar{a}\langle T, T; b, b, c, d\rangle \mid c(y : T) . \overline{\mathsf{fail}}\langle\rangle) \\ L' = \nu(b : \updownarrow[T], c : \updownarrow[T], d : T)(\bar{a}\langle T, T; b, b, c, d\rangle \mid c(y : T) . \mathbf{0}) \end{array}$$

and a type environment Γ which contains only $a : \updownarrow[X, Y; \updownarrow[X], \updownarrow[Y], \updownarrow[Y], X]$ and a suitable type for fail. Now it may at first appear that L and L' should be considered equivalent with respect to the type information in Γ as the private name d is only released along channel a at some abstract type represented by X, say. And the private name c is only released as a channel which carries values of abstract type Y, say. In order to distinguish these processes a test term would need to obtain a value of type Y to send on c. However, there is a testing context which allows the name d to be cast to type Y:

$$R = a(X, Y; z : \updownarrow[X], z' : \updownarrow[Y], z'' : \updownarrow[Y], x : X) . (\bar{z}\langle x\rangle \mid z'(y : Y) . \overline{z''}\langle y\rangle)$$

It is easy to check that this process is well-typed with respect to Γ. Here, when R communicates with L and L', the vector of fresh names is received along a and the variables z and z' are aliased so that a further internal communication within R sends d as if it were of type X but receives it as if it were of type Y. It can then be sent along c to interact with the remainder of L and L' to distinguish them.

$$\mu ::= \tau \mid c(\vec{U};\vec{b}) \mid \nu(\vec{a}:\vec{T})\overline{c}\langle\vec{U};\vec{b}\rangle \quad \text{(Untyped Labels)}$$

$$\frac{}{c(\vec{X};\vec{x}:\vec{T}).P \xrightarrow{c(\vec{U};\vec{b})} P[\vec{U}/\vec{X};\vec{b}/\vec{x}]}\ (\text{R-IN}) \qquad \frac{}{\overline{c}\langle\vec{U};\vec{b}\rangle \xrightarrow{\overline{c}\langle\vec{U};\vec{b}\rangle} \mathbf{0}}\ (\text{R-OUT})$$

$$\frac{P \xrightarrow{\mu} P' \quad \mathsf{bn}(\mu)\cap\mathsf{fn}(Q)=\emptyset}{P\mid Q \xrightarrow{\mu} P'\mid Q}\ (\text{R-PAR})$$

$$\frac{P \xrightarrow{c(\vec{U};\vec{b})} P' \quad Q \xrightarrow{\nu(\vec{a}:\vec{T})\overline{c}\langle\vec{U};\vec{b}\rangle} Q' \quad \{\vec{a}\}\cap\mathsf{fn}(P)=\emptyset}{P\mid Q \xrightarrow{\tau} \nu(\vec{a}:\vec{T})(P'\mid Q')}\ (\text{R-COM})$$

$$\frac{P \xrightarrow{\mu} P' \quad a\notin\mathsf{fn}(\mu)\cup\mathsf{bn}(\mu)}{\nu(a:T)P \xrightarrow{\mu} \nu(a:T)P'}\ (\text{R-NEW}) \qquad \frac{P \xrightarrow{\nu(\vec{a}:\vec{T})\overline{c}\langle\vec{U};\vec{b}\rangle} P' \quad a\in\{\vec{b}\}\setminus\{c,\vec{a}\}}{\nu(a:T)P \xrightarrow{\nu(\vec{a}:\vec{T},a:T)\overline{c}\langle\vec{U};\vec{b}\rangle} P'}\ (\text{R-OPEN})$$

$$\frac{!P\mid P \xrightarrow{\mu} P'}{!P \xrightarrow{\mu} P'}\ (\text{R-REPL})$$

$$\frac{P \xrightarrow{\mu} P'}{\mathsf{if}\ a=a\ \mathsf{then}\ P\ \mathsf{else}\ Q \xrightarrow{\mu} P'}\ (\text{R-TEST-T}) \qquad \frac{a\neq b \quad Q \xrightarrow{\mu} Q'}{\mathsf{if}\ a=b\ \mathsf{then}\ P\ \mathsf{else}\ Q \xrightarrow{\mu} Q'}\ (\text{R-TEST-F})$$

Fig. 2. Untyped Labelled Transitions $P \xrightarrow{\mu} P'$ (eliding symmetric rules for $P\mid Q$)

2.2 Dynamic Semantics

The untyped transition semantics for the asynchronous polymorphic π-calculus is given in Figure 2, and is the same as Pierce and Sangiorgi's.

We define the free names of a label $\mathsf{fn}(\mu)$ as $\mathsf{fn}(\tau)=\emptyset$, $\mathsf{fn}(c(\vec{U};\vec{b}))=\{c,\vec{b}\}$ and $\mathsf{fn}(\nu(\vec{a}:\vec{T})\overline{c}\langle\vec{U};\vec{b}\rangle)=\{c,\vec{b}\}\setminus\{\vec{a}\}$. We also define the bound names of a label $\mathsf{bn}(\mu)$ as $\mathsf{bn}(\tau)=\mathsf{bn}(c(\vec{U};\vec{b}))=\emptyset$ and $\mathsf{bn}(\nu(\vec{a}:\vec{T})\overline{c}\langle\vec{U};\vec{b}\rangle)=\{\vec{a}\}$. We define *weak transitions* as follows:

$$P \Longrightarrow P' \text{ whenever } P \longrightarrow \cdots \longrightarrow P'$$
$$P \overset{\mu}{\Longrightarrow} P' \text{ whenever } P \Longrightarrow \cdot \xrightarrow{\mu} \cdot \Longrightarrow P'$$
$$P \overset{\hat{\tau}}{\Longrightarrow} P' \text{ whenever } P \Longrightarrow P'$$
$$P \overset{\hat{\mu}}{\Longrightarrow} P' \text{ whenever } P \overset{\mu}{\Longrightarrow} P' \text{ (in the case } \mu\neq\tau)$$

The untyped semantics is useful for defining the run-time behaviour of processes, but is not immediately appropriate for defining a notion of equivalence, as it distinguishes terms such as B_1 and B_2 which cannot be distinguished by any well-typed environment:

$$B_1 \xrightarrow{\nu(t:Bool,f:Bool,test:Test(Bool))\overline{getBools}\langle Bool;t,f,test\rangle} \xrightarrow{t(a,b)} \xrightarrow{\overline{a}\langle\rangle}$$
$$B_2 \xrightarrow{\nu(t:Bool,f:Bool,test:Test(Bool))\overline{getBools}\langle Bool;t,f,test\rangle} \xrightarrow{t(a,b)} \xrightarrow{\overline{b}\langle\rangle}$$

$$
\begin{array}{rll}
X,Y,Z & & \text{(Type Variables)} \\
T,U,V,W & ::= X \mid \updownarrow[\vec{X};\vec{T}] & \text{(Types: } X \text{ is non-generative, } \updownarrow[\vec{X};\vec{T}] \text{ is generative)} \\
\Gamma,\Delta & ::= \vec{X};\vec{n}:\vec{T} & \text{(Typing Contexts)}
\end{array}
$$

$$\frac{X \in \Gamma}{\Gamma \vdash X}\ \text{(T-TVAR)} \qquad \frac{\vec{X},\Gamma \vdash \vec{T} \quad \{\vec{X}\} \cap \mathsf{dom}(\Gamma) = \emptyset \quad \vec{X} \text{ disjoint}}{\Gamma \vdash \updownarrow[\vec{X};\vec{T}]}\ \text{(T-CHAN)}$$

$$\frac{\vec{X} \vdash \vec{T}}{\vec{X};\vec{n}:\vec{T} \vdash \diamond}\ \text{(T-ENV)} \qquad \frac{\Gamma \vdash \diamond \quad (n:T) \in \Gamma}{\Gamma \vdash n:T}\ \text{(T-VAL)}$$

$$\frac{\Gamma \vdash n:\updownarrow[\vec{X};\vec{T}] \quad \vec{X},\Gamma,\vec{x}:\vec{T} \vdash P \quad \{\vec{X},\vec{x}\} \cap \mathsf{dom}(\Gamma) = \emptyset \quad \vec{x} \text{ disjoint}}{\Gamma \vdash n(\vec{X};\vec{x}:\vec{T}).P}\ \text{(T-IN)}$$

$$\frac{\Gamma \vdash n:\updownarrow[\vec{X};\vec{U}] \quad \Gamma \vdash \vec{n}:\vec{U}[\vec{T}/\vec{X}]}{\Gamma \vdash \overline{n}\langle\vec{T};\vec{n}\rangle}\ \text{(T-OUT)}$$

$$\frac{\Gamma \vdash \diamond}{\Gamma \vdash \mathbf{0}}\ \text{(T-NIL)} \qquad \frac{\Gamma \vdash P \quad \Gamma \vdash Q}{\Gamma \vdash P \mid Q}\ \text{(T-PAR)}$$

$$\frac{\Gamma,a:T \vdash P \quad a \notin \mathsf{dom}(\Gamma) \quad \mathsf{ftv}(T) \subseteq \mathsf{dom}(\Gamma) \quad T \text{ is generative}}{\Gamma \vdash \nu(a:T)P}\ \text{(T-NEW)}$$

$$\frac{\Gamma \vdash P}{\Gamma \vdash !P}\ \text{(T-REPL)} \qquad \frac{\Gamma \vdash n:T \quad \Gamma \vdash m:U \quad \Gamma \vdash P \quad \Gamma \vdash Q}{\Gamma \vdash \mathsf{if}\, n = m\, \mathsf{then}\, P\, \mathsf{else}\, Q}\ \text{(T-TEST-W)}$$

Fig. 3. Type System, with judgements $\Gamma \vdash T$, $\Gamma \vdash \diamond$, $\Gamma \vdash n:T$ and $\Gamma \vdash P$

These behaviours correspond to the untyped test T, but do not correspond to any well-typed test, which only has access to the abstract type X and not to the concrete type *Bool*. As a result, no well-typed test can cause the action $\xrightarrow{t(a,b)}$ to be performed. We will come back to this point in Section 3.2.

2.3 Static Semantics

The static semantics for the asynchronous polymorphic π-calculus is given in Figure 3 where the domain of a typing context $\mathsf{dom}(\Gamma)$ is $\mathsf{dom}(\vec{X};\vec{n}:\vec{T}) = \{\vec{X},\vec{n}\}$, the free names of a typing context $\mathsf{fn}(\Gamma)$ are $\mathsf{fn}(\vec{X};\vec{n}:\vec{T}) = \mathsf{fn}(\vec{n})$, the free variables of a typing context $\mathsf{fv}(\Gamma)$ are $\mathsf{fv}(\vec{X};\vec{n}:\vec{T}) = \mathsf{fv}(\vec{n})$, and the free type variables of a typing context $\mathsf{ftv}(\Gamma)$ are $\mathsf{ftv}(\vec{X};\vec{n}:\vec{T}) = \{\vec{X}\} \cup \mathsf{ftv}(\vec{T})$. We say that a typing context Δ is closed if $\mathsf{fv}(\Delta) = \mathsf{ftv}(\Delta) = \emptyset$ and moreover for any $a:T \in \Delta$ and $a:U \in \Delta$ then $T = U$. We write $\Gamma[\sigma]$ as the typing context given by $(\vec{X};\vec{n}:\vec{T})[\vec{W}/\vec{Y};\vec{m}/\vec{y}] = (\vec{X} \setminus \vec{Y};\vec{n}[\vec{m}/\vec{y}] : \vec{T}[\vec{W}/\vec{Y}])$.

This is quite a simple type system, as it does not include subtyping, bounded polymorphism, or recursive types, although we expect that such features could be added with no essential difficulty.

In Section 4, we will discuss the relationship between this type system and that of Pierce and Sangiorgi. For the moment, we will just highlight one crucial non-standard point about our typing judgement: we are allowing identifiers to have more than one type in a typing context. For example:

$$X,Y;a:\updownarrow[\updownarrow[X],\updownarrow[Y]],b:\updownarrow[X],b:\updownarrow[Y]\vdash\overline{a}\langle b,b\rangle$$

To motivate the use of these multicontexts consider the processes

$$\begin{aligned}P &\stackrel{\text{def}}{=} c(X,Y;x:\updownarrow[\updownarrow[X],\updownarrow[Y]]).x(y:\updownarrow[X],z:\updownarrow[Y]).\overline{x}\langle y,z\rangle\\ Q &\stackrel{\text{def}}{=} \nu(a:\updownarrow[\updownarrow[\mathsf{int}],\updownarrow[\mathsf{int}]])\nu(b:\updownarrow[\mathsf{int}])\overline{c}\langle\mathsf{int},\mathsf{int};a\rangle\mid\overline{a}\langle b,b\rangle\end{aligned}$$

which can interact as follows:

$$\begin{aligned}P\mid Q &\xrightarrow{\tau} \nu(a:\updownarrow[\updownarrow[\mathsf{int}],\updownarrow[\mathsf{int}]])(a(y:\updownarrow[\mathsf{int}],z:\updownarrow[\mathsf{int}]).\overline{a}\langle y,z\rangle\mid\nu(b:\updownarrow[\mathsf{int}])(\overline{a}\langle b,b\rangle))\\ &\xrightarrow{\tau} \nu(a:\updownarrow[\updownarrow[\mathsf{int}],\updownarrow[\mathsf{int}]])\nu(b:\updownarrow[\mathsf{int}])\overline{a}\langle b,b\rangle\end{aligned}$$

This interaction comes about due to the following labelled transitions from P (with appropriate matching transitions from Q):

$$\begin{aligned}P &\xrightarrow{c(\mathsf{int},\mathsf{int};a)} a(y:\updownarrow[\mathsf{int}],z:\updownarrow[\mathsf{int}]).\overline{a}\langle y,z\rangle\\ &\xrightarrow{a(b,b)} \overline{a}\langle b,b\rangle\end{aligned}$$

Now, P typechecks as:

$$c:\updownarrow[X,Y;\updownarrow[\updownarrow[X],\updownarrow[Y]]]\vdash P$$

and we would like to find an appropriate typing for $\overline{a}\langle b,b\rangle$. The obvious typing would be to use Q's choice of concrete implementation of X and Y as int; however in order to reason about P independently of Q we must choose a typing which preserves type abstraction and is independent of any choice provided by Q. To do this we use a typing which more closely resembles P's view of the interaction:

$$X,Y;c:\updownarrow[X,Y;\updownarrow[\updownarrow[X],\updownarrow[Y]]],a:\updownarrow[\updownarrow[X],\updownarrow[Y]],b:\updownarrow[X],b:\updownarrow[Y]\vdash\overline{a}\langle b,b\rangle$$

which makes a use of two different types for b in the type environment.

Note that multiple typings for the same identifier are only required in giving the labelled transition system semantics in Section 3.2. In particular, type-checking closed terms can be performed without ever using an environment with multiple bindings for the same variable.

Pierce and Sangiorgi do not allow multiple typings for the same identifier: instead, they use *type unification* for the same purpose. In their model, the types X and Y above would be unified, and so b would just have one type $b:\updownarrow[X]$. This produces a model which is sound, but not complete, as we discuss in Section 4.

An alternative strategy to either multiple typings for variables or type unification would be subtyping with intersection types [6, 27], which ensure that meets exist in

the subtype relation. Subtyping with meets are used, for example, by Hennessy and Riely [12] to ensure subject reduction. Intersection types would provide this language with pleasant properties such as principal typing, which it currently lacks, but at the cost of complexity.

3 Equivalences for Asynchronous Polymorphic Pi-Calculus

Process equivalence has a long history, including Milner's [18] bisimulation, Brookes, Hoare and Roscoe's [4] failures-divergences equivalence, and Hennessy's [11] testing equivalence. In this paper, we will follow Pierce and Sangiorgi [22] and investigate *contextual equivalence* on processes [13, 21], and prove that it coincides with an appropriate bisimulation. We conjecture that our results would carry over to other equivalences such as relating failures-divergences and testing: the shift from a labelled transition system to a failures set should not be affected by the polymorphic nature of the labels.

Contextual equivalence has a very natural definition: it is the most generous equivalence satisfying three natural properties: *reduction closure* (that is, respecting the operational semantics), *contextuality* (that is, respecting the syntax of the language), and *barb preservation* (that is, respecting output on visible channels).

Unfortunately, although contextual equivalence has a very natural definition, it is difficult to reason about directly, due to the requirement of contextuality. Since contextuality requires processes to be equivalent in all contexts, to show contextual equivalence of P and Q, we have to show contextual equivalence of $\mathcal{C}[P]$ and $\mathcal{C}[Q]$ for any appropriately typed context $\mathcal{C}$: moreover, attempts to show this by induction on $\mathcal{C}$ break down due to reduction closure.

The problem of showing processes to be contextually equivalent is not restricted to polymorphic π-calculi, for example this problem comes up in treatments of the λ-calculus [2], monomorphic π-calculus [19] and object languages [1]. The standard solution is to ask for a *fully abstract* model, which coincides with contextual equivalence, but is hopefully more tractable.

The problem of finding fully abstract models of programming languages originates with Milner [17], and was investigated in depth by Plotkin [24] for the functional language PCF. For polymorphic functional languages, logical relations [26] allow for the construction of fully abstract models [23] but require an induction on type, and so break down in the presence of recursive types. Sumii and Pierce have recently shown that bisimulation based on sets of relations [29] yields a fully abstract model in the presence of recursive types.

To date the only known models for polymorphic process languages have been sound but not complete [22, 3]. We will now show that a very direct treatment of type-respecting labelled transitions generates a fully abstract bisimulation equivalence which makes no use of logical relations or type unification.

3.1 Contextual Equivalence

Process contexts are typed as follows: $\Delta \vdash \mathcal{C}[\Gamma]$ whenever $\forall(\Gamma \vdash P)\,.\,(\Delta \vdash \mathcal{C}[P])$. A typed relation on closed processes $\mathcal{R}$ is a set of triples (Γ, P, Q) such that $\Gamma \vdash P$ and $\Gamma \vdash Q$ and Γ is closed. We will typically write $\Gamma \vDash P\ \mathcal{R}\ Q$ whenever $(\Gamma, P, Q) \in \mathcal{R}$. Given any typed relation on closed processes $\mathcal{R}$, we can define its open extension $\mathcal{R}^\circ$ to be the

$$\alpha ::= \tau \mid \nu(\vec{a}:\vec{T})c[\vec{U};\vec{b}] \mid \nu(\vec{a})\overline{c}\langle\vec{X};\vec{b}:\vec{V}\rangle \quad \text{(Typed Labels)}$$
$$C ::= (\Gamma \vdash [\sigma]P) \quad \text{(Configurations)}$$

$$\frac{P \xrightarrow{\tau} P'}{(\Gamma \vdash [\sigma]P) \xrightarrow{\tau} (\Gamma \vdash [\sigma]P')} \text{(TR-SILENT)}$$

$$\frac{\Gamma, \vec{a}:\vec{T} \vdash \overline{c}\langle\vec{U};\vec{b}\rangle \quad \{\vec{a}\} \cap \mathsf{dom}(\Gamma) = \emptyset \quad \vec{T} \text{ are generative}}{(\Gamma \vdash [\sigma]P) \xrightarrow{\nu(\vec{a}:\vec{T})c[\vec{U};\vec{b}]} (\Gamma, \vec{a}:\vec{T} \vdash [\sigma]P \mid (\overline{c}\langle\vec{U};\vec{b}\rangle[\sigma]))} \text{(TR-RECEP)}$$

$$\frac{P \xrightarrow{\nu(\vec{a}:\vec{T})\overline{c}\langle\vec{U};\vec{b}\rangle} P' \quad \Gamma \vdash c(\vec{X};\vec{x}:\vec{V}).\mathbf{0} \quad \{\vec{a},\vec{X}\} \cap \mathsf{dom}(\Gamma) = \emptyset}{(\Gamma \vdash [\sigma]P) \xrightarrow{\nu(\vec{a})\overline{c}\langle\vec{X};\vec{b}:\vec{V}\rangle} (\vec{X}, \Gamma, \vec{b}:\vec{V} \vdash [\vec{U}/\vec{X}, \sigma]P')} \text{(TR-OUT-W)}$$

Fig. 4. Typed Labelled Transitions $C \xrightarrow{\alpha} C'$

typed relation on processes given by $\Gamma \vDash P \; \mathcal{R}^{\circ} \; Q$ whenever $\Gamma[\sigma], \Delta \vDash P[\sigma] \; \mathcal{R} \; Q[\sigma]$ for any closed typing context of the form $(\Gamma[\sigma], \Delta)$.

Definition 4 (Reduction closure). *A typed relation $\mathcal{R}$ on closed processes is reduction-closed whenever $\Delta \vDash P \; \mathcal{R} \; Q$ and $P \xrightarrow{\tau} P'$ implies there exists some Q' such that $Q \Longrightarrow Q'$ and $\Delta \vDash P' \; \mathcal{R} \; Q'$.*

Definition 5 (Contextuality). *A typed relation $\mathcal{R}$ on closed processes is contextual whenever $\Gamma \vDash P \; \mathcal{R}^{\circ} \; Q$ and $\Delta \vdash \mathcal{C}[\Gamma]$ implies $\Delta \vDash \mathcal{C}[P] \; \mathcal{R}^{\circ} \; \mathcal{C}[Q]$.*

Definition 6 (Barb preservation). *A typed relation $\mathcal{R}$ on closed processes is barb-preserving whenever $\Delta \vDash P \; \mathcal{R} \; Q$ and $P \xrightarrow{\overline{a}\langle\rangle}$ implies $Q \xRightarrow{\overline{a}\langle\rangle}$.*

We can now define contextual equivalence $\cong$ as the open extension of the largest symmetric typed relation on closed processes which is reduction-closed, contextual and barb-preserving. The requirement of contextuality makes it very difficult to prove properties about contextual equivalence, and so we investigate bisimulation as a more tractable proof technique for establishing contextual equivalence.

3.2 Bisimulation

As a first attempt to find a more tractable presentation of contextual equivalence, we could use *bisimulation*. Unfortunately, as we discussed in Section 2.2, our untyped labelled transition system does not respect the type system, and so gives rise to too fine an equivalence. We therefore investigate a restricted labelled transition system which respects types: this is defined in Figure 4. The transition system is given by a relation:

$$(\Gamma \vdash [\sigma]P) \xrightarrow{\alpha} (\Gamma' \vdash [\sigma']P')$$

between configurations of the form $(\Gamma \vdash [\sigma]P)$. These comprise three constituent parts:

– P is the process being observed: after the transition, it becomes process P'.
– Γ is the *external* view of the typing context P operates in. This external view may not have complete information about the types, for example P may have exported the concrete type int as an abstract type X. Only X will be recorded in the typing context. As P exports more type information, Γ may grow to become Γ'. It is here that we make use of the multiple entries in type environments.
– σ is a type substitution, mapping the external view to the internal view. This mapping provides complete information about the types exported by P, for example int/X records that external type X is internal type int. Note that this substitution is **not** applied to P, we represent unapplied substitutions as $[\sigma]P$, and applied substitutions as $P[\sigma]$. We will elide the type substitution when it is empty.

A configuration $(\Gamma \vdash [\sigma]P)$ is closed whenever $\Gamma[\sigma] \vdash P$ and $\Gamma[\sigma]$ is closed. There are three kinds of transitions:

– *Silent transitions* $(\Gamma \vdash [\sigma]P) \xrightarrow{\tau} (\Gamma \vdash [\sigma]P')$ which are inherited from the untyped transition system.
– *Receptivity transitions* $(\Gamma \vdash [\sigma]P) \xrightarrow{\nu(\vec{a}:\vec{T})c[\vec{U};\vec{b}]} (\Gamma, \vec{a}:\vec{T} \vdash [\sigma]P \mid (\overline{c}\langle\vec{U};\vec{b}\rangle[\sigma]))$ which allow the environment to send data to the process. We require the message to typecheck, and we allow the environment to generate new names, which are recorded in the type environment. We are modelling an asynchronous language, and so processes are always input-enabled. Note that the process is sending no information to the environment, so the type substitution σ does not grow. Note also that the message is typed using the external view Γ but must have the type mapping σ applied to it for it to be mapped to the internal type consistent with P.
– *Output transitions* $(\Gamma \vdash [\sigma]P) \xrightarrow{\nu(\vec{a})\overline{c}\langle\vec{X};\vec{b}:\vec{V}\rangle} (\vec{X}, \Gamma, \vec{b}:\vec{V} \vdash [\vec{U}/\vec{X}, \sigma]P')$ which allow the process to send data to the environment. The channel being used to communicate with the environment must be typed $\updownarrow[\vec{X};\vec{V}]$, so the typing context is extended with abstract types $\vec{X}$ and the new type information $\vec{b}:\vec{V}$. This may result in more than one type being given to the same name, which is why we allow duplicate entries in typing contexts. The process P must have provided concrete implementations $\vec{U}$ of the abstract types $\vec{X}$: these are recorded in the type substitution.

To demonstrate how our typed labelled transitions can be used we return to the example above of processes L and L' and type environment Γ. We show a sequence of typed transitions from $(\Gamma \vdash L)$ which cannot be matched by $(\Gamma \vdash L')$:

$$(\Gamma \vdash L) \xrightarrow{\nu(b,c,d)\overline{a}\langle X,Y;b:\updownarrow[X],b:\updownarrow[Y],c:\updownarrow[Y],d:Y\rangle} (\Gamma' \vdash [\sigma]c(y:\updownarrow[T]).\,\overline{\mathsf{fail}}\langle\rangle)$$

where σ is $[T, T/X, Y]$ and Γ' is $X, Y, \Gamma, b:\updownarrow[X], b:\updownarrow[Y], c:\updownarrow[Y], d:X$. At this point we would like to use Rule TR-RECEP to provide a message on channel c to facilitate a communication, however, there is no name of the appropriate type listed in Γ' and the restriction to generative types for the fresh names means that this cannot yet be done.

However, note the following transitions:

$$\begin{array}{rl}(\Gamma' \vdash [\sigma]c(y:\updownarrow[T]).\overline{\mathsf{fail}}\langle\rangle) & \xrightarrow{b[d]} \quad (\Gamma' \vdash [\sigma]c(y:\updownarrow[T]).\overline{\mathsf{fail}}\langle\rangle \mid \overline{b}\langle d\rangle) \\ & \xrightarrow{\overline{b}\langle d\rangle} \quad (\Gamma', d:Y \vdash [\sigma]c(y:\updownarrow[T]).\overline{\mathsf{fail}}\langle\rangle) \\ & \xrightarrow{c[d]} \quad (\Gamma', d:Y \vdash [\sigma]c(y:\updownarrow[T]).\overline{\mathsf{fail}}\langle\rangle \mid \overline{c}\langle d\rangle) \\ & \xRightarrow{\overline{\mathsf{fail}}\langle\rangle}\end{array}$$

in which the second type listed for b in Γ' is used to justify the $\overline{b}\langle d\rangle$ transition. These transitions serve to mimic the typecasting and subsequent use of the extruded name d by a testing context which are crucial to distinguishing L and L'.

We now formalise our notion of bisimulation equivalence. A typed relation on closed configurations $\mathcal{R}$ is a set of 5-tuples $(\Gamma, \sigma, P, \rho, Q)$ such that $\Gamma[\sigma] \vdash P$ and $\Gamma[\rho] \vdash Q$ and both $\Gamma[\sigma]$ and $\Gamma[\rho]$ are closed. For convenience we will write $\Gamma \vDash [\sigma]P \mathrel{\mathcal{R}} [\rho]Q$ whenever $(\Gamma, \sigma, P, \rho, Q) \in \mathcal{R}$.

Definition 7 (Bisimulation). *A simulation $\mathcal{R}$ is a typed relation on closed configurations such that if $\Gamma \vDash [\sigma]P \mathrel{\mathcal{R}} [\rho]Q$ and $(\Gamma \vdash [\sigma]P) \xrightarrow{\alpha} (\Gamma' \vdash [\sigma']P')$ then we can show $(\Gamma \vdash [\rho]Q) \xRightarrow{\widehat{\alpha}} (\Gamma' \vdash [\rho']Q')$ for some $\Gamma' \vDash [\sigma']P' \mathrel{\mathcal{R}} [\rho']Q'$. A bisimulation is a simulation whose inverse is also a simulation. Let $\approx$ be the largest bisimulation.*

As an example, a possible execution of B_1 is as follows:

$$(\Gamma \vdash B_1) \xrightarrow{\nu(t,f,test)\overline{getBools}\langle X;t:X,f:X,test:Test(X)\rangle} (\Gamma' \vdash [Bool/X]B_1')$$

$$\begin{array}{rcl}\Gamma & \stackrel{\mathsf{def}}{=} & getBools : \updownarrow[X;X,X,Test(X)] \\ \Gamma' & \stackrel{\mathsf{def}}{=} & X, \Gamma, t:X, f:X, test:Test(X) \\ B_1' & \stackrel{\mathsf{def}}{=} & !t(x:Signal, y:Signal).\overline{x}\langle\rangle \mid \\ & & !f(x:Signal, y:Signal).\overline{y}\langle\rangle \mid \\ & & !test(b:Bool, x:Signal, y:Signal).\overline{b}\langle x,y\rangle\end{array}$$

Note that in this output action, only the abstract type X is revealed: the concrete type *Bool* is kept hidden, and is recorded in the type substitution $Bool/X$. At this point, the type requirements of Rule TR-RECEP block any communication on the channels t and f, since they are only known at the abstract type X. As a result, the only productive action at this point is to use Rule TR-RECEP to communicate on channel *test*, for example:

$$\begin{array}{rl}(\Gamma' \vdash [Bool/X]B_1') & \xrightarrow{\nu(a:Signal,b:Signal)test[t,a,b]} (\Gamma'' \vdash [Bool/X]B_1' \mid \overline{test}\langle t,a,b\rangle) \\ & \xrightarrow{\tau} (\Gamma'' \vdash [Bool/X]B_1' \mid \overline{t}\langle a,b\rangle)\end{array}$$

where the environment has generated two new names a and b, which are recorded in the type environment:

$$\Gamma'' \stackrel{\mathsf{def}}{=} \Gamma', a:Signal, b:Signal$$

Again, the environment only knows about t at abstract type X, not at channel type, and so cannot observe the output on channel t, so the only output which is observable is the one on channel a given by:

$$\begin{array}{rl}(\Gamma'' \vdash [Bool/X]B_1' \mid \overline{t}\langle a,b\rangle) & \xrightarrow{\tau} (\Gamma'' \vdash [Bool/X]B_1' \mid \overline{a}\langle\rangle) \\ & \xrightarrow{\overline{a}\langle\rangle} (\Gamma'' \vdash [Bool/X]B_1')\end{array}$$

Note that B_2 has a matching behaviour:

$$\begin{array}{rl}(\Gamma \vdash B_2) & \xrightarrow{\nu(t,f,test)\overline{getBools}\langle X;t{:}X,f{:}X,test{:}Test(X)\rangle} (\Gamma' \vdash [Bool/X]B_2') \\ & \xrightarrow{\nu(a{:}Signal,b{:}Signal)test[t,a,b]} (\Gamma'' \vdash [Bool/X]B_2' \mid \overline{test}\langle t,a,b\rangle) \\ & \xrightarrow{\tau} (\Gamma'' \vdash [Bool/X]B_2' \mid \overline{t}\langle b,a\rangle) \\ & \xrightarrow{\tau} (\Gamma'' \vdash [Bool/X]B_2' \mid \overline{a}\langle\rangle) \\ & \xrightarrow{\overline{a}\langle\rangle} (\Gamma'' \vdash [Bool/X]B_2')\end{array}$$

$$\begin{array}{rl}B_2' \stackrel{\mathsf{def}}{=} & !t(x:Signal,y:Signal).\overline{y}\langle\rangle \mid \\ & !f(x:Signal,y:Signal).\overline{x}\langle\rangle \mid \\ & !test(b:Bool,x:Signal,y:Signal).\overline{b}\langle y,x\rangle\end{array}$$

We can now show that $\Gamma \vDash B_1 \approx B_2$ by defining an appropriate relation and showing that it is a bisimulation. The details of this are routine, apart from a slight complication caused by Rule TR-RECEP being allowed at any point, even on channels (such as *getBools*) which are intended for use only as output channels. We avoid this complication by looking at *bisimulation with* t, f, *test inputs*:

Definition 8 (Configuration with $\vec{c}$ inputs). *A process P with $\vec{c}$ inputs is one where any subprocess of the form $n(\vec{X};\vec{x}:\vec{T}).Q$ has $n \in \{\vec{c}\}$. A configuration $(\Gamma \vdash [\sigma]P)$ with $\vec{c}$ inputs is one where P is a process with $\vec{c}$ inputs.*

Definition 9 (Bisimulation with $\vec{c}$ inputs). *A simulation with $\vec{c}$ inputs $\mathcal{R}$ is a typed relation on closed configurations with c inputs such that if we have $\Gamma \vDash [\sigma]P \mathcal{R} [\rho]Q$ and $(\Gamma \vdash [\sigma]P) \xrightarrow{\alpha} (\Gamma' \vdash [\sigma']P')$, for α not a receptivity transition on a channel outside $\vec{c}$, then we can show $(\Gamma \vdash [\rho]Q) \stackrel{\widehat{\alpha}}{\Longrightarrow} (\Gamma' \vdash [\rho']Q')$ for some $\Gamma' \vDash [\sigma']P' \mathcal{R} [\rho']Q'$. A bisimulation with $\vec{c}$ inputs is a simulation with $\vec{c}$ inputs whose inverse is also a simulation with $\vec{c}$ inputs.*

Proposition 1. *If $\mathcal{R}$ is a bisimulation with $\vec{c}$ inputs then $\mathcal{R} \subseteq \approx$.*

Proof. Let $\mathcal{R}'$ be defined as $\Gamma,\Delta \vDash [\sigma]P \mid R \mathcal{R}' [\rho]Q \mid R$ whenever $\mathsf{dom}(\Gamma) \cap \mathsf{dom}(\Delta) = \emptyset$ and $\Gamma \vDash [\sigma]P \mathcal{R} [\rho]Q$ and R is of the form $\prod_i \overline{a_i}\langle\vec{T_i};\vec{b_i}\rangle$ where $a_i \notin \{\vec{c}\}$. It is routine to show that $\mathcal{R}'$ is a bisimulation, and hence $\mathcal{R} \subseteq \mathcal{R}' \subseteq \approx$. □

If we define $\mathcal{R}$ such that:

$$\Gamma \vDash B_1 \mathcal{R} B_2$$

and:

$$\Gamma', \vec{a} : \mathit{Signal} \vDash [\mathit{Bool}/X](B_1' \mid P_1) \; \mathcal{R} \; [\mathit{Bool}/X](B_2' \mid P_2)$$

where P_1 is of the form:

$$P_1 = \prod_i \overline{\mathit{test}}\langle v_i, b_i, c_i\rangle \mid \prod_j \overline{w_j}\langle d_j, e_j\rangle \mid \prod_k \overline{f_k}\langle\rangle$$

with P_2 of the form:

$$P_2 = \prod_i \overline{\mathit{test}}\langle v_i, b_i, c_i\rangle \mid \prod_j \overline{w_j}\langle e_j, d_j\rangle \mid \prod_k \overline{f_k}\langle\rangle$$

and:

$$v_i, w_j \in \{t, f\} \quad b_i, c_i, d_j, e_j, f_k \in \{\vec{a}\}$$

then it is direct to show that $\mathcal{R}$ is a bisimulation with t, f, test inputs, and hence B_1 and B_2 are bisimilar.

We are now in position to show full abstraction of bisimulation for contextual equivalence, and so provide a tractable model of polymorphic π-calculus.

3.3 Soundness of Bisimulation for Contextual Equivalence

The difficult property to show is that bisimulation is a congruence: from this it is routine to establish that bisimulation implies contextual equivalence. Showing congruence for bisimulation is a well-established problem for process languages, going back to Milner [18]. In the case of polymorphic π, the problem is in showing that bisimulation is preserved by parallel composition. We do this by constructing a candidate bisimulation:

$$\begin{array}{ll} \Gamma \vDash [\sigma]P \mid R[\sigma] \; \mathcal{R} \; [\rho]Q \mid R[\rho] & \text{whenever } \Gamma \vDash [\sigma]P \approx [\rho]Q \\ & \text{and } \Gamma \vdash R \\ & \text{and } \sigma \text{ and } \rho \text{ are type substitutions} \end{array}$$

and then showing that this is a bisimulation (up to some technicalities which we shall elide for the moment). This has a routine proof, except for one case, which is when $R[\sigma] \longrightarrow R'[\sigma]$. It is straightforward to establish that type substitutions do not influence reduction, and so we have $R[\rho] \longrightarrow R'[\rho]$, and all that remains is to show that $\Gamma \vDash [\sigma]P \mid R'[\sigma] \; \mathcal{R} \; [\rho]Q \mid R'[\rho]$. Unfortunately, this is not directly possible, due to the requirement that $\Gamma \vdash R'$. If we had a subject reduction result for open processes, then this would be routine, but this result is not true due to channels with multiple types:

$$\begin{array}{rcl} \overline{a}\langle c\rangle \mid a(x:Y).\,\overline{b}\langle x\rangle & \longrightarrow & \mathbf{0} \mid \overline{b}\langle c\rangle \\ X, Y; a : \updownarrow[X], a : \updownarrow[Y], b : \updownarrow[Y], c : X & \vdash & \overline{a}\langle c\rangle \mid a(x:Y).\,\overline{b}\langle x\rangle \\ X, Y; a : \updownarrow[X], a : \updownarrow[Y], b : \updownarrow[Y], c : X & \not\vdash & \mathbf{0} \mid \overline{b}\langle c\rangle \end{array}$$

Pierce and Sangiorgi's technique for dealing with this problem is to introduce type unification to ensure that every channel has a unique type. Unfortunately, as we will discuss in Section 4, the resulting semantics is incomplete. Instead of using such unifications, we observe that in any case where subject reduction fails, it does so because of communication on a visible channel: if the channel was hidden by a ν-binder, then it would have only one type, and so subject reduction holds. We therefore observe that in the cases where subject reduction fails to hold, there must be a pair of matching visible reductions which caused the communication.

Proposition 2 (Open subject reduction). *If* $\Gamma \vdash P$ *and* $P \xrightarrow{\tau} P''$ *then either:*

1. $\Gamma \vdash P''$, *or*
2. $P \xrightarrow{\nu(\vec{a}:\vec{T})\overline{c}\langle\vec{U};\vec{b}\rangle} \xrightarrow{c(\vec{X};\vec{b})} P'$ *where* $P'' \equiv (\nu(\vec{a}:\vec{T})P')[\vec{U}/\vec{X}]$.

Proof. Given in Appendix A.

Here, we are working up to structural equivalence, which has its usual definition [19].

Definition 10 (Structural equivalence). *Let* $\equiv$ *be the equivalence generated by treating* $|$ *as a commutative monoid with unit* $\mathbf{0}$*, satisfying scope extrusion,* $!P \equiv !P \,|\, P$*, and closed under* $|$ *and* $\nu(a:T)$.

In the example (up to structural equivalence):

$$\begin{array}{rcl}
\overline{a}\langle c\rangle \,|\, a(x:Y)\,.\,\overline{b}\langle x\rangle & \xrightarrow{\overline{a}\langle c\rangle} & \mathbf{0} \,|\, a(x:Y)\,.\,\overline{b}\langle x\rangle \\
 & \xrightarrow{a(c)} & \mathbf{0} \,|\, \overline{b}\langle c\rangle \\
X,Y;a:\updownarrow[X],a:\updownarrow[Y],b:\updownarrow[Y],c:X & \vdash & \overline{a}\langle c\rangle \,|\, a(x:Y)\,.\,\overline{b}\langle x\rangle \\
X,Y;a:\updownarrow[X],a:\updownarrow[Y],b:\updownarrow[Y],c:X,c:Y & \vdash & \mathbf{0} \,|\, a(x:Y)\,.\,\overline{b}\langle x\rangle \\
X,Y;a:\updownarrow[X],a:\updownarrow[Y],b:\updownarrow[Y],c:X,c:Y & \vdash & \mathbf{0} \,|\, \overline{b}\langle c\rangle
\end{array}$$

The crucial point is that these extra transitions by the testing context correspond to complementary typed transitions by the process such that, after the visible $\overline{a}\langle c\rangle$ output action, the typing context Γ is extended with $c:Y$. The problematic residual of the test term R' ($\mathbf{0} \,|\, \overline{b}\langle c\rangle$ in the example) can now be typed in this extended Γ and the bisimulation argument can be completed.

We can now show that bisimulation is a congruence, from which soundness follows directly. We recall that $\approx^\circ$ is the open extension of $\approx$.

Theorem 1 (Bisimulation is a congruence). *If* $\Gamma \vDash P \approx^\circ Q$ *then* $\Delta \vDash \mathcal{C}[P] \approx^\circ \mathcal{C}[Q]$ *for any* $\Delta \vdash \mathcal{C}[\Gamma]$.

Proof. Given in Appendix A.

Theorem 2 (Soundness of bisimulation for contextual equivalence). *If* $\Gamma \vDash P \approx^\circ Q$ *then* $\Gamma \vDash P \cong Q$.

Proof. It suffices to prove the result for closed processes, for which we need to show that $\approx$ is symmetric, reduction-closed,contextual and barb-preserving. All of these are direct, except for contextuality, which follows from Theorem 1.

3.4 Completeness of Bisimulation for Contextual Equivalence

The proof of soundness for bisimulation required some non-standard techniques. In comparison, the proof of completeness is quite straightforward, and follows the usual *definability* argument [11, 9, 15] of showing that for every visible action α, we can find a process R which exactly tests for the ability to perform α. Once we have established definability, completeness follows in a straightforward fashion.

Theorem 3 (Completeness of bisimulation for contextual equivalence). *If* $\Gamma \vDash P \cong Q$ *then* $\Gamma \vDash P \approx^\circ Q$.

Proof. Given in Appendix B.

4 Comparison with Pierce and Sangiorgi

In this paper, we have shown that weak bisimulation is fully abstract for observational equivalence for an asynchronous polymorphic π-calculus. This is almost enough to settle the open problem set by Pierce and Sangiorgi [22] of finding a fully abstract semantics for their polymorphic π-calculus. There are, however, some differences between their setting and ours, most of which we believe to be routine, with one important exception: the type rule for if-then-else.

4.1 Minor differences

The minor differences between our polymorphic π-calculus and theirs are:

1. We are considering weak bisimulation rather than strong bisimulation.
2. Since we are considering weak bisimulation, we have not included $P+Q$ in our language of processes. We speculate that this could be handled in the usual fashion, by defining observational equivalence on processes in the style of Milner [18].
3. We have treated an asynchronous rather than a synchronous language, since the soundness result follows more naturally for the resulting asynchronous transition system. We speculate that a fully abstract bisimulation for a synchronous language can be given by adding transitions for synchronous input as well as receptivity:

$$\frac{P \xrightarrow{c(\vec{U};\vec{b})} P' \quad \Gamma,\vec{a}:\vec{T} \vdash \overline{c}\langle\vec{U};\vec{b}\rangle \quad \{\vec{a}\} \cap \mathsf{dom}(\Gamma) = \emptyset \quad \vec{T} \text{ are generative}}{(\Gamma \vdash [\sigma]P) \xrightarrow{\nu(\vec{a}:\vec{T})c(\vec{U};\vec{b})} (\Gamma,\vec{a}:\vec{T} \vdash [\sigma]P')} \text{(TR-IN)}$$

   Note that the label used here for synchronous input is distinct from the label used for receptivity.
4. We have used Honda and Yoshida's definition of observational equivalence [13] as our touchstone equivalence. Although this has been proposed as *too strong* in the literature [28], where it has been shown not to coincide with Milner and Sangiorgi's notion of observational equivalence, [21], it is important to note that in the presence of a variable-name distinction, the discrepancy between these two definitions disappears as variable-open terms are more clearly identified and congruence with respect to input prefixing is guaranteed. See [8] for further information on the relationship between Honda and Yoshida's work and that of Milner and Sangiorgi [21].
5. Our type system keeps track explicitly of free type variables, rather than treating them implicitly: this makes some of the book-keeping easier, at the cost of some additional syntactic overhead.

We do not consider these issues any further in this paper.

4.2 Major difference: typing if-then-else

There is, however, one important difference between our language and Pierce and Sangiorgi's, even though it may appear at first sight to be a minor point: the type rule for if-then-else. In their paper, a strong type rule is given:

$$\frac{\Gamma \vdash n:T \quad \Gamma \vdash m:T \quad \Gamma \vdash P \quad \Gamma \vdash Q}{\Gamma \vdash \mathsf{if}\, n = m \,\mathsf{then}\, P \,\mathsf{else}\, Q} \text{(T-TEST-S)}$$

In our work, the weaker type rule T-TEST-W is used, which allows n and m to have different types: name equality testing is essentially untyped in this setting. Note that in a language with subtyping and a top type (such as Java or C#), these rules are equivalent, since we can always choose T to be the top type, and use subsumption to derive T-TEST-W from T-TEST-S. In the absence of subtyping, however, the rule T-TEST-W allows more processes to typecheck, so raises the expressive power of tests, and hence makes observational equivalence finer. For example:

$$P \stackrel{\text{def}}{=} \nu(b : \updownarrow[\mathsf{int}])\nu(c : \updownarrow[\mathsf{string}])\overline{a}\langle \mathsf{int}, \mathsf{string}; b, c\rangle$$
$$Q \stackrel{\text{def}}{=} \nu(b : \updownarrow[\mathsf{int}])\overline{a}\langle \mathsf{int}, \mathsf{int}; b, b\rangle$$

As long as $a : \updownarrow[X, Y; \updownarrow[X], \updownarrow[Y]]$ these processes cannot be distinguished by any test which uses the type rule T-TEST-S, but they can be distinguished by:

$$R \stackrel{\text{def}}{=} a(X, Y; x : \updownarrow[X], y : \updownarrow[Y]) \,.\, \mathsf{if}\, x = y \,\mathsf{then}\, \overline{d}\langle\rangle \,\mathsf{else}\, \mathbf{0}$$

which typechecks using type rule T-TEST-W. In fact, there is a third possible type rule for if-then-else, which makes use of type unification:

$$\frac{\Gamma \vdash n : T \quad \Gamma \vdash m : U \qquad \mathsf{mgu}(T, U) = \sigma \Rightarrow \Gamma[\sigma] \vdash P[\sigma] \quad \Gamma \vdash Q}{\Gamma \vdash \mathsf{if}\, n = m \,\mathsf{then}\, P \,\mathsf{else}\, Q} \text{ (T-TEST-U)}$$

where $\mathsf{mgu}(T, U)$ builds the most general type substitution σ such that $T[\sigma] = U[\sigma]$. This type rule is strictly weaker than T-TEST-W, and raises the expressive power of tests even further, and hence makes observational equivalence even finer. For example:

$$P \stackrel{\text{def}}{=} \nu(c : \updownarrow[\mathsf{int}, \mathsf{string}])\nu(d : \updownarrow[\mathsf{int}])\overline{a}\langle \mathsf{int}, \mathsf{string}; c, d\rangle \,.\, \overline{b}\langle \mathsf{string}; c\rangle \,.\, d(x : \mathsf{int}) \,.\, \overline{e}\langle x\rangle$$
$$Q \stackrel{\text{def}}{=} \nu(c : \updownarrow[\mathsf{int}, \mathsf{string}])\nu(d : \updownarrow[\mathsf{int}])\overline{a}\langle \mathsf{int}, \mathsf{string}; c, d\rangle \,.\, \overline{b}\langle \mathsf{string}; c\rangle$$

As long as $a : \updownarrow[X, Y; \updownarrow[X, Y], \updownarrow[X]]$, $b : \updownarrow[Z; \updownarrow[\mathsf{int}, Z]]$ and $e : \updownarrow[\mathsf{int}]$, these processes cannot be distinguished by any test which uses T-TEST-W, but they can be distinguished by:

$$R \stackrel{\text{def}}{=} a(X, Y; x : \updownarrow[X, Y], y : \updownarrow[X]) \,.\, b(Z; z : \updownarrow[\mathsf{int}, Z]) \,.\, \mathsf{if}\, x = z \,\mathsf{then}\, \overline{y}\langle 5\rangle \,\mathsf{else}\, \mathbf{0}$$

which typechecks using type rule T-TEST-U. We have that:

- The type rule T-TEST-W has a matching fully abstract bisimulation equivalence $\approx$, which for purpose of this discussion we shall refer to as $\approx_{\mathsf{w}}$ (shown in Theorems 2 and 3).
- The type rule T-TEST-S has a matching fully abstract bisimulation equivalence $\approx_{\mathsf{s}}$ (shown in Appendix D).
- The type rule T-TEST-U has a matching fully abstract bisimulation equivalence $\approx_{\mathsf{u}}$ (shown in Appendix E).

Moreover:

– We have inclusions on these equivalences: if $\Gamma \vDash P \approx_{\mathsf{w}} Q$ then $\Gamma \vDash P \approx_{\mathsf{s}} Q$ for any $\Gamma \vdash_{\mathsf{s}} P$ and $\Gamma \vdash_{\mathsf{s}} Q$ (and similarly for $\approx_{\mathsf{u}}$ and $\approx_{\mathsf{w}}$).
– The above examples show that the inclusions are strict: we have $\Gamma \vDash P \not\approx_{\mathsf{w}} Q$ and $\Gamma \vDash P \approx_{\mathsf{s}} Q$ for some $\Gamma \vdash_{\mathsf{s}} P$ and $\Gamma \vdash_{\mathsf{s}} Q$ (and similarly for $\approx_{\mathsf{u}}$ and $\approx_{\mathsf{w}}$).
– The type rule for if-then-else used by Pierce and Sangiorgi is T-TEST-S.
– Pierce and Sangiorgi's bisimulation is the strong, synchronous version of $\approx_{\mathsf{u}}$ (shown in Appendix C).

Hence, since synchrony and weak bisimulation play no role in the above examples, we have a resolution of Pierce and Sangiorgi's conjecture:

– Pierce and Sangiorgi's polymorphic bisimulation is sound, but not complete, for their polymorphic π-calculus.

These arguments are formalised in Appendices C, D and E. We note that the only parts of the proof which significantly change are the proofs of two propositions: Propositions 9 (Output Contextuality) and 10 (Extrusion).

5 Conclusions

This paper gives the first fully abstract semantics for a polymorphic process language. Moreover, due to careful choice of language features (in particular the typing rule for if-then-else), the semantics is straightforward: the only nonstandard part of the presentation is that names are given more than one type in a type environment. This corresponds to the ability for a polymorphic program to be sent the same channel at multiple different types. In contrast to polymorphic λ-calculi, polymorphic π-calculi have the ability to compare names for syntactic equality, and so there is an internal test which can detect when the same name has been given multiple different types.

We believe that the techniques given in this paper are quite robust (for example there are no uses of type induction) and could be scaled with little difficulty to larger type systems with features such as subtyping, F-bounded polymorphism, and recursive types. Moreover, object languages such as the ς-calculus support object equality, and so we believe that adapting our previous fully abstract semantics [14] for objects [1] to deal with generic objects would also be possible.

References

1. M. Abadi and L. Cardelli. *A Theory of Objects*. Springer-Verlag, 1996.
2. H. P. Barendregt. *The Lambda Calculus, Its Syntax and Semantics*. North Holland, 1984.
3. M. Berger, K. Honda, and N. Yoshida. Genericity and the pi-calculus. In *Proc. Int. Conf. Foundations of Software Science and Computer Structures (FoSSaCs)*, Lecture Notes in Computer Science. Springer-Verlag, 2003.
4. S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *J. ACM*, 31(3):560–599, 1984.
5. P. Canning, W. Cook, W. Hill, W. Olthoff, and J. C. Mitchell. F-bounded polymorphism for object-oriented programming. In *Proc. Int. Conf. Functional Programming Languages and Computer Architecture (FPCA)*, pages 273–280. ACM Press, 1989.
6. M. Coppo and M. Dezani-Ciancaglini. A new type-assignment for λ-terms. *Archiv Math. Logik*, 19:139–156, 1978.

7. Microsoft Corporation. ECMA and ISO/IEC C# and common language infrastructure standards, 2004. http://msdn.microsoft.com/net/ecma/.
8. C. Fournet and G. Gonthier. A hierarchy of equivalences for asynchronous calculi. In *Proc. Int. Conf. Automata, Languages and Programming (ICALP)*, volume 1443 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.
9. C. Fournet, G. Gonthier, J-J. Levy, L. Maranget, and D. Remy. A calculus of mobile agents. In *Proc. Int. Conf. Concurrency Theory (CONCUR)*, volume 1119 of *Lecture notes in computer science*. Springer-Verlag, 1996.
10. J-Y. Girard, P. Taylor, and Y. Lafont. *Proofs and Types*. Cambridge University Press, 1989.
11. M. Hennessy. *Algebraic Theory of Processes*. MIT Press, 1988.
12. M. Hennessy and J. Riely. Resource access control in systems of mobile agents. *Information and Computation*, 173(1):82–120, 2002.
13. K. Honda and N. Yoshida. On reduction-based process semantics. *Theoretical Computer Science*, 152(2):437–486, 1995.
14. A. S. A. Jeffrey and J. Rathke. A fully abstract may testing semantics for concurrent objects. In *Proc. IEEE Logic In Computer Science*, pages 101–112. IEEE Press, 2002. Full version to appear in *Theoretical Computer Science*.
15. A. S. A. Jeffrey and J. Rathke. Contextual equivalence for higher-order pi-calculus revisited. In *Proc. Mathematical Foundations of Programming Semantics*, Electronic Notes in Computer Science. Elsevier, 2003.
16. Sun Microsystems. Release notes Java 2 platform standard edition development kit 5.0, 2004. http://java.sun.com/j2se/1.5.0/relnotes.html.
17. R. Milner. Fully abstract models of typed lambda-calculi. *Theoretical Computer Science*, 4:1–22, 1977.
18. R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.
19. R. Milner. *Communication and mobile systems: the π-calculus*. Cambridge University Press, 1999.
20. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, Part I + II. *Information and Computation*, 100(1):1–77, 1992.
21. R. Milner and D. Sangiorgi. Barbed bisimulation. In *Proc. Int. Conf. Automata, Languages and Programming (ICALP)*, volume 623 of *Lecture Notes in Computer Science*. Springer-Verlag, 1992.
22. B. C. Pierce and D. Sangiorgi. Behavioral equivalence in the polymorphic pi-calculus. *J. ACM*, 47(3):531–584, 2000.
23. A. M. Pitts. Parametric polymorphism and operational equivalence. *Mathematical Structures in Computer Science*, 10:321–359, 2000.
24. G. D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5:223–255, 1977.
25. J. C. Reynolds. Types, abstraction and parametric polymorphism. *Information Processing*, 83:513–523, 1983.
26. J. C. Reynolds. An introduction to logical relations and parametric polymorphism (abstract). In *Proc. ACM Symp. Principles of Programming Languages*, pages 155–156. ACM Press, 1993.
27. J. C. Reynolds. *Theories of Programming Languages*. Cambridge University Press, 1998.
28. D. Sangiorgi and D. Walker. *The Pi-Calculus: A Theory Of Mobile Processes*. Cambridge University Press, 2001.
29. E. Sumii and B. C. Pierce. A bisimulation for type abstraction and recursion. In *Proc. ACM Symp. Principles of Programming Languages*, pages 63–74, 2005.
30. P. Wadler. Theorems for free! In *Proc. Int. Conf. Functional Programming Languages and Computer Architecture (FPCA)*, pages 347–359. ACM Press, New York, 1989.

A Bisimulation is a congruence

Definition 11 (Free and bound names of typed labels). *Define the free names of a typed label as:*

$$\mathsf{fn}(\tau) = \emptyset \quad \mathsf{fn}(\nu(\vec{a} : \vec{T})c[\vec{U};\vec{b}]) = \mathsf{fn}(\nu(\vec{a} : \vec{T})\overline{c}\langle\vec{U};\vec{b}\rangle) = \{c,\vec{b}\} \setminus \{\vec{a}\}$$

Define the bound names of a typed label as:

$$\mathsf{bn}(\tau) = \emptyset \quad \mathsf{bn}(\nu(\vec{a} : \vec{T})c[\vec{U};\vec{b}]) = \mathsf{bn}(\nu(\vec{a} : \vec{T})\overline{c}\langle\vec{U};\vec{b}\rangle) = \{\vec{a}\}$$

Definition 12 (ν-extension of a relation). *For any typed relation on closed configurations $\mathcal{R}$, define its ν-extension $\mathcal{R}^\nu$ to be the typed relation on closed configurations generated by:*

$$\begin{array}{l}\Gamma \vDash [\sigma]P\ \mathcal{R}^\nu\ [\rho]Q \text{ whenever } \Gamma' \vDash [\sigma']P'\ \mathcal{R}\ [\rho']Q' \\ \qquad\qquad \text{for some } P \equiv \nu(\vec{a} : \vec{T})P', Q \equiv \nu(\vec{a} : \vec{U})Q' \\ \qquad\qquad \text{and } \sigma \subseteq \sigma', \rho \subseteq \rho', \Gamma \subseteq \Gamma'\end{array}$$

Definition 13 (Bisimulation up to ν). *A simulation up to ν is a typed relation on closed configurations $\mathcal{R}$ such that if $\Gamma \vDash [\sigma]P\ \mathcal{R}\ [\rho]Q$ and $(\Gamma \vdash [\sigma]P) \xrightarrow{\alpha} (\Gamma' \vdash [\sigma']P')$ then we have $(\Gamma \vdash [\rho]Q) \stackrel{\widehat{\alpha}}{\Longrightarrow} (\Gamma' \vdash [\rho']Q')$ for some $\Gamma' \vDash [\sigma']P'\ \mathcal{R}^\nu\ [\rho']Q'$. A bisimulation up to ν is a simulation up to ν whose inverse is also a simulation up to ν.*

Proposition 3 (Soundness of bisimulation up to ν). *If $\mathcal{R}$ is a bisimulation up to ν then $\mathcal{R} \subseteq \approx$.*

Proof. To show this we take $\mathcal{R}^\nu$ itself as the bisimulation witness and demonstrate that this relation is indeed a bisimulation. This is straightforward from the following two, easily verifiable properties:

- If $(\Gamma \vdash [\sigma]\nu(\vec{a} : \vec{T})P) \xrightarrow{\alpha} (\Gamma' \vdash [\sigma']P')$ and $\Gamma \subseteq \Gamma''$, $\sigma \subseteq \sigma''$ and $\Gamma''[\sigma''] \vdash P$ then we have that $(\Gamma'' \vdash [\sigma'']P) \xrightarrow{\alpha'} (\Gamma''' \vdash [\sigma''']P''')$ where $\sigma' \subseteq \sigma'''$, $P' \equiv \nu(\vec{a} : \vec{T} \setminus \mathsf{bn}(\alpha))P'''$ and α' is $\nu(\vec{c} \setminus \vec{a})\overline{c}\langle\vec{U};\vec{c'}\rangle$ if α is $\nu(\vec{c})\overline{c}\langle U;\vec{c'}\rangle$ and α otherwise.
- If $(\Gamma'' \vdash [\sigma'']P) \xrightarrow{\alpha'} (\Gamma''' \vdash [\sigma''']P''')$ and $\Gamma \subseteq \Gamma''$, $\sigma \subseteq \sigma''$, $\{\vec{a}\}, \Gamma$ distinct and $\Gamma[\sigma] \vdash \nu(\vec{a} : \vec{T})P$ then $(\Gamma \vdash [\sigma]\nu(\vec{a} : \vec{T})P) \xrightarrow{\alpha} (\Gamma' \vdash [\sigma']P')$ where $\sigma' \subseteq \sigma'''$, $P' \equiv \nu(\vec{a} : \vec{T} \setminus \mathsf{bn}(\alpha))P'''$ and α' is as above.

Proposition 4 (Reduction under type substitution). *For any process P and type substitution σ, $P[\sigma] \xrightarrow{\mu} Q$ if and only if we can find μ' and Q' such that $P \xrightarrow{\mu'} Q'$, $\mu = \mu'[\sigma]$ and $Q = Q'[\sigma]$.*

Proof. Follows by an easy induction on the derivation of $P[\sigma] \xrightarrow{\mu} Q$ and conversely on the derivation of $P \xrightarrow{\mu'} Q'$.

Proposition 5 (Output reduction). *If $\Gamma \vdash P$ and $P \xrightarrow{\nu(\vec{a}:\vec{T})\overline{c}\langle\vec{U};\vec{b}\rangle} P'$ then $\vec{T}$ are generative, $\Gamma, \vec{a} : \vec{T} \vdash \overline{c}\langle\vec{U};\vec{b}\rangle$ and $\Gamma, \vec{a} : \vec{T} \vdash P'$.*

Proof. It is not difficult to check that $P \xrightarrow{\nu(\vec{a}:\vec{T})\overline{c}\langle\vec{U};\vec{b}\rangle} P'$ tells us that $P \equiv \nu(\vec{a} : \vec{T})(\overline{c}\langle\vec{U};\vec{b}\rangle \mid P')$. We know that $\Gamma \vdash P$ also so we know from rule T-NEW that $\vec{T}$ is generative and moreover, from rule T-PAR, that $\Gamma, \vec{a} : \vec{T} \vdash \overline{c}\langle\vec{U};\vec{b}\rangle$ and $\Gamma, \vec{a} : \vec{T} \vdash P'$ as required.

Proposition 6 (Input reduction). *If $\Gamma \vdash P$ and $P \xrightarrow{c(\vec{X};\vec{b})} P'$ and $\{\vec{X}\} \cap \mathsf{dom}(\Gamma) = \emptyset$ then $\Gamma \vdash c(\vec{X};\vec{x} : \vec{V})$ and $\vec{X}, \Gamma, \vec{b} : \vec{V} \vdash P'$.*

Proof. This is proved similarly to the previous lemma, however in this case we must also make use of the substitution lemma below.

Lemma 1 (Substitution). *Suppose* $\vec{Y},\Gamma,\Delta \vdash \vec{n} : \vec{U}[\vec{T}/\vec{X}]$, *then*

1. $\vec{Y},\vec{X},\Gamma,\vec{x}:\vec{U},\Delta \vdash P$ *implies* $\vec{Y},\Gamma,\Delta[\vec{T}/\vec{X}] \vdash P[\vec{T}/\vec{X};\vec{n}/\vec{x}]$ *and*
2. $\vec{Y},\vec{X},\Gamma,\vec{x}:\vec{U},\Delta \vdash \vec{m}:\vec{V}$ *implies* $\vec{Y},\Gamma,\Delta[\vec{T}/\vec{X}] \vdash \vec{m}[\vec{n}/\vec{x}] : \vec{V}[\vec{T}/\vec{X}]$

Proof. Standard induction on the derivations of the type judgements.

Proposition 7 (Closed subject reduction). *If* Γ *is a closed typing context,* $\Gamma \vdash P$ *and* $P \xrightarrow{\tau} P'$ *then* $\Gamma \vdash P'$.

Proof. This is entirely analogous to the proof in [22].

We recall the property of *open subject reduction* as stated in Proposition 2.

(Open subject reduction) If $\Gamma \vdash P$ and $P \xrightarrow{\tau} P''$ then either:

1. $\Gamma \vdash P''$, or
2. $P \xrightarrow{\nu(\vec{a}:\vec{T})\overline{c}\langle\vec{U};\vec{b}\rangle} \xrightarrow{c(\vec{X};\vec{b})} P'$ where $P'' \equiv (\nu(\vec{a}:\vec{T})P')[\vec{U}/\vec{X}]$.

Proof. We first of all introduce an annotation to the τ label. We will write τ_c for τ actions from P which have been derived as a communication on channel c which is free in P. We will write τ_ν for communications on a private channel.

There are essentially two parts to this proof, first we show that communications on free channel names always guarantee commuting visible input and output actions. This case leads to conclusion (2) above:

if $\Gamma \vdash P$ and $P \xrightarrow{\tau_c} P''$ then $P \xrightarrow{\nu(\vec{a}:\vec{T})\overline{c}\langle\vec{U};\vec{b}\rangle} \xrightarrow{c(\vec{X};\vec{b})} P'$ where $P'' \equiv \nu(\vec{a}:\vec{T})P'[\vec{U}/\vec{X}]$.

The only interesting case here occurs as an instance of the R-COM rule. Here we have $P \equiv P_0 \mid Q_0$ for some $P_0 \xrightarrow{c(\vec{U};\vec{b})} P_0'$, $Q_0 \xrightarrow{\nu(\vec{a}:\vec{T})\overline{c}\langle\vec{U};\vec{b}\rangle} Q_0'$ and $P'' \equiv \nu(\vec{a}:\vec{T})(P_0' \mid Q_0')$. Given this, we immediately have $P \xrightarrow{\nu(\vec{a}:\vec{T})\overline{c}\langle\vec{U};\vec{b}\rangle} (P_0 \mid Q_0')$ and, by the receptivity of types on rule R-IN we can always choose $\vec{X}$ such that $P_0 \xrightarrow{c(\vec{X};\vec{b})} P_0''$ and $P_0''[\vec{U}/\vec{X}] \equiv P_0'$. Therefore,

$$P \xrightarrow{\nu(\vec{a}:\vec{T})\overline{c}\langle\vec{U};\vec{b}\rangle} (P_0 \mid Q_0') \xrightarrow{c(\vec{X};\vec{b})} (P_0'' \mid Q_0')$$

with $\nu(\vec{a}:\vec{T})(P_0'' \mid Q_0')[\vec{U}/\vec{X}] \equiv \nu(\vec{a}:\vec{T})(P_0' \mid Q_0') \equiv P''$ as required.

Secondly, we show by induction over the derivation of the silent transition the following property:

If $\Gamma \vdash P$ and either ($P \xrightarrow{\tau_c} P'$ such that there is a unique type T with $\Gamma \vdash c : T$) or ($P \xrightarrow{\tau_\nu} P'$) then $\Gamma \vdash P'$.

We show the two interesting cases in which the last derivation rule used is R-NEW and R-COM, respectively. If the last rule used is R-NEW then we have three sub-cases:

1. If $P \xrightarrow{\tau_\nu} P'$, then it is easy to check that $P \equiv \nu(\vec{a}:\vec{T})P_0$ for some P_0 such that $P_0 \xrightarrow{\tau} P_0'$ with $P' \equiv \nu(\vec{a}:\vec{T})P_0'$. We now have two sub-sub-cases:
   (a) If $P_0 \xrightarrow{\tau_\nu} P_0'$ then the inductive hypothesis will guarantee $\Gamma,\vec{a}:\vec{T} \vdash P_0'$ and we use type rule T-NEW to finish.

(b) If $P_0 \xrightarrow{\tau_c} P_0'$ with $c \in \vec{a}$, $P \equiv \nu(\vec{a}:\vec{T})P_0$ then we know that $\Gamma, \vec{a}:\vec{T} \vdash P_0$ and moreover, there is a unique type T_0 such that $\Gamma, \vec{a}:\vec{T} \vdash c : T_0$, therefore the inductive hypothesis can be applied to obtain $\Gamma, \vec{a}:\vec{T} \vdash P_0'$ and again, T-NEW can be used to finish.

2. If $P \xrightarrow{\tau_c} P'$ then we proceed similarly to the first sub-sub-case above.

Suppose then that the last rule used is R-COM so that

$$\frac{P \xrightarrow{c(\vec{U};\vec{b})} P' \quad Q \xrightarrow{\nu(\vec{a}:\vec{T})\overline{c}\langle\vec{U};\vec{b}\rangle} Q' \quad \{\vec{a}\} \cap \mathsf{fn}(P) = \emptyset}{P \mid Q \xrightarrow{\tau_c} \nu(\vec{a}:\vec{T})(P' \mid Q')}$$

We know that $\Gamma \vdash P \mid Q$ and by analysis of the transition rules we must have

$$\begin{aligned} P &\equiv \nu(\vec{a}':\vec{T}')(c(\vec{X};\vec{x}:\vec{T}_0).P'' \mid P''') \\ Q &\equiv \nu(\vec{a}:\vec{T})(\overline{c}\langle\vec{U};\vec{b}\rangle \mid Q') \\ P' &\equiv \nu(\vec{a}':\vec{T}')(P''[\vec{U}/\vec{X};\vec{b}/\vec{x}] \mid P''') \end{aligned}$$

so we are required to show that $\Gamma, \vec{a}:\vec{T} \vdash P' \mid Q'$.

We know that $\Gamma \vdash Q$ so this easily implies that $\Gamma, \vec{a}:\vec{T} \vdash Q'$. It remains to show that

$$\Gamma, \vec{a}:\vec{T} \vdash \nu(\vec{a}':\vec{T}')(P''[\vec{U}/\vec{X};\vec{b}/\vec{x}] \mid P''')$$

Given that $\Gamma \vdash P$ we know (using weakening) that $\Gamma, \vec{a}:\vec{T}, \vec{a}':\vec{T}' \vdash P'''$ so we reduce our obligation to proving

$$\Gamma, \vec{a}:\vec{T}, \vec{a}':\vec{T}' \vdash P''[\vec{U}/\vec{X};\vec{b}/\vec{x}]$$

Note that $\Gamma \vdash Q$ also implies that $\Gamma, \vec{a}:\vec{T} \vdash \overline{c}\langle\vec{U};\vec{b}\rangle$. This implies further that

$$\begin{aligned} &\Gamma \vdash c : \updownarrow[\vec{X}';\vec{T}_0'] \\ &\Gamma, \vec{a}:\vec{T} \vdash \vec{b} : \vec{T}_0'[\vec{U}/\vec{X}'] \end{aligned}$$

for some $\vec{X}'$ and $\vec{T}_0'$. Note now that $\Gamma \vdash P$ implies $\Gamma, \vec{a}':\vec{T}' \vdash c(\vec{X};\vec{x}:\vec{T}_0).P''$ and that this (together with $c \in \mathsf{dom}(\Gamma)$) further implies $\Gamma \vdash c : \updownarrow[\vec{X};\vec{T}_0]$. The hypothesis states that there is a unique type for c in Γ, so we know $\vec{X} = \vec{X}'$ and $\vec{T}_0 = \vec{T}_0'$. This (with appropriate weakening) allows us to derive $\vec{X}, \Gamma, \vec{a}':\vec{T}', \vec{x}:\vec{T}_0 \vdash P''$. We can use this and the typing for $\vec{b}$ above (also with appropriate weakening) as hypotheses to the Substitution lemma to obtain our goal.

Proposition 8 (Labelled Subject Reduction). *If* $(\Gamma \vdash [\sigma]P) \xrightarrow{\alpha} (\Gamma' \vdash [\sigma']P')$ *and* $(\Gamma \vdash [\sigma]P)$ *is a closed configuration then* $(\Gamma' \vdash [\sigma']P')$ *is also a closed configuration.*

Proof. For α a τ action, we simply note that Γ' and σ' are unchanged and that (using Proposition 4) $P \xrightarrow{\tau} P'$ implies $P[\sigma] \xrightarrow{\tau} P'[\sigma]$. Closed subject reduction then gives the result. For α a receptivity action, we need only observe that $\Gamma, \vec{a}:\vec{T} \vdash \overline{c}\langle\vec{U};\vec{b}\rangle$ implies $(\Gamma, \vec{a}:\vec{T})[\sigma] \vdash \overline{c}\langle\vec{U};\vec{b}\rangle[\sigma]$.

For α a send action the difficulty lies in demonstrating that $(\vec{X}, \Gamma, \vec{b}:\vec{V})[\vec{U}/\vec{X}, \sigma]$ is a closed environment. Indeed, we know that $\Gamma[\vec{U}/\vec{X}, \sigma] = \Gamma[\sigma]$ is closed so we need to check that any $b \in \vec{b}$ has a unique typing in $(\vec{X}, \Gamma, \vec{b}:\vec{V})[\vec{U}/\vec{X}, \sigma]$. We know $\Gamma[\sigma] \vdash P$. By unfolding this derivation we can obtain

$$\begin{aligned} &\Gamma[\sigma], \vec{a}:\vec{T} \vdash \vec{b} : \vec{W}[U/X] \\ &\Gamma[\sigma], \vec{a}:\vec{T} \vdash c : \updownarrow[X;\vec{W}] \end{aligned}$$

for some $\vec{W}$. We also have that $\Gamma \vdash c(\vec{X};\vec{x}:\vec{V})$, and it is easy to check $\Gamma[\sigma] \vdash c(\vec{X};\vec{x}:\vec{V}[\sigma])$, which yields $\Gamma[\sigma] \vdash c : \updownarrow[\vec{X};\vec{V}[\sigma]]$. As $\Gamma[\sigma]$ is a closed environment we must have a unique type for c. Therefore, $\vec{V}[\sigma]$ must be $\vec{W}$. Now, take any $b \in \vec{b}$. If $b \notin \mathsf{dom}(\Gamma)$ then we must have $b \in \vec{a}$, uniquely as a_i say. Thus, b must have a unique type $(T_i = V_i[\vec{U}/\vec{X}, \sigma])$ in $(\vec{X}, \Gamma, \vec{b}:\vec{V})[\vec{U}/\vec{X}, \sigma]$. Otherwise, we have $b \in \mathsf{dom}(\Gamma)$. In this case though, we know that $b : V[\vec{U}/\vec{X}, \sigma]$ already appears in $\Gamma[\sigma]$ because $\Gamma[\sigma], \vec{a}:\vec{T} \vdash \vec{b} : \vec{V}[\vec{U}/\vec{X}, \sigma]$.

We can now prove Theorem 1: if $\Gamma \vDash P \mathrel{\dot{\approx}} Q$ then $\Delta \vDash \mathcal{C}[P] \mathrel{\dot{\approx}} \mathcal{C}[Q]$ for any $\Delta \vdash \mathcal{C}[\Gamma]$.

Proof. We show that $\mathrel{\dot{\approx}}$ is preserved by each of the process operators, from which the result follows by induction on $\mathcal{C}$. The difficult case is to show that $\mathrel{\dot{\approx}}$ is preserved by $|$, which follows if we can establish that the following relation is a bisimulation up to ν:

$$\Gamma \vDash [\sigma]P \,|\, R[\sigma] \;\mathcal{R}\; [\rho]Q \,|\, R[\rho] \text{ whenever } \Gamma \vDash [\sigma]P \approx [\rho]Q \text{ and } \Gamma \vdash R \text{ and } \sigma \text{ and } \rho \text{ are type substitutions}$$

Since $\mathcal{R}$ is symmetric, it suffices from Proposition 3 to show that $\mathcal{R}$ is a simulation up to ν. Consider any transition of the form:

$$(\Gamma \vdash [\sigma]P \,|\, R[\sigma]) \xrightarrow{\alpha} (\Gamma' \vdash [\sigma']P'')$$

where:

$$\Gamma \vDash [\sigma]P \approx [\rho]Q \quad \Gamma \vdash R$$

We are required to establish a matching weak transition for $(\Gamma \vdash [\rho]Q \,|\, R[\rho])$, for which we proceed by case analysis on α. The interesting case is when $\alpha = \tau$, so from Rule TR-SILENT we have:

$$P \,|\, R[\sigma] \xrightarrow{\tau} P''$$

and we proceed by case analysis on the derivation of this transition. The interesting case is when the symmetric form of Rule R-PAR was used, and we have:

$$R[\sigma] \xrightarrow{\tau} R''' \quad P'' = P \,|\, R'''$$

for which we use Proposition 4 to get that:

$$R \xrightarrow{\tau} R'' \quad R''' = R''[\sigma]$$

We then use Proposition 2 to get two cases, of which the interesting one is 2, where we have:

$$R \xrightarrow{\nu(\vec{a}:\vec{T})\overline{c}\langle \vec{U};\vec{b}\rangle} \xrightarrow{c(\vec{X};\vec{b})} R' \quad R'' \equiv (\nu(\vec{a}:\vec{T})R')[\vec{U}[\sigma]/\vec{X},\sigma]$$

so we can use Propositions 5 and 6 to get that $\vec{T}$ are generative and:

$$\Gamma, \vec{a}:\vec{T} \vdash \overline{c}\langle \vec{U};\vec{b}\rangle \quad \Gamma, \vec{a}:\vec{T} \vdash c(\vec{X};\vec{x}:\vec{V}) \quad \vec{X}, \Gamma, \vec{a}:\vec{T}, \vec{b}:\vec{V} \vdash R'$$

Hence we can use Rules TR-RECEP and TR-OUT-W to establish:

$$\begin{array}{rl} (\Gamma \vdash [\sigma]P) & \xrightarrow{\nu(\vec{a}:\vec{T})c[\vec{U};\vec{b}]} \; (\Gamma, \vec{a}:\vec{T} \vdash [\sigma]P \,|\, \overline{c}\langle \vec{U}[\sigma];\vec{b}\rangle) \\ & \xrightarrow{\overline{c}\langle \vec{X};\vec{b}:\vec{V}\rangle} \; (\vec{X}, \Gamma, \vec{a}:\vec{T}, \vec{b}:\vec{V} \vdash [\vec{U}[\sigma]/X,\sigma]P \,|\, \mathbf{0}) \end{array}$$

Note that this step makes use of input receptivity, hence our use of asynchronous rather than synchronous π-calculus. Since $\Gamma \vDash [\sigma]P \approx [\rho]Q$ we have:

$$\begin{array}{rl} (\Gamma \vdash [\rho]Q) & \xRightarrow{\nu(\vec{a}:\vec{T})c[\vec{U};\vec{b}]} \; (\Gamma, \vec{a}:\vec{T} \vdash [\rho]Q'') \\ & \xRightarrow{\overline{c}\langle \vec{X};\vec{b}:\vec{V}\rangle} \; (\vec{X}, \Gamma, \vec{a}:\vec{T}, \vec{b}:\vec{V} \vdash [\vec{W}/\vec{X},\rho]Q') \end{array}$$

where:

$$\vec{X}, \Gamma, \vec{a}:\vec{T}, \vec{b}:\vec{V} \vDash [\vec{U}[\sigma]/X,\sigma]P \approx [\vec{W}/\vec{X},\rho]Q'$$

From Proposition 4 we have:

$$R[\rho] \xrightarrow{\nu(\vec{a}:\vec{T}[\rho])\overline{c}\langle \vec{U}[\rho];\vec{b}\rangle} \xrightarrow{c(\vec{W};\vec{b})} R'[\vec{W}/\vec{X},\rho]$$

and so it is routine to establish using Rules R-PAR and R-COM:

$$Q \,|\, R[\rho] \implies \nu(\vec{a} : \vec{T}[\rho])Q' \,|\, R'[\vec{W}/\vec{X}, \rho]$$

and hence using TR-SILENT:

$$(\Gamma \vdash [\rho]Q \,|\, R[\rho]) \implies (\Gamma \vdash [\rho]\nu(\vec{a} : \vec{T}[\rho])(Q' \,|\, R'[\vec{W}/\vec{X}, \rho]))$$

Finally, since $\vec{X}, \Gamma, \vec{a} : \vec{T}, \vec{b} : \vec{V} \vDash [\vec{U}[\sigma]/X, \sigma]P \approx [\vec{W}/\vec{X}, \rho]Q'$ we have by definition of $\mathcal{R}$:

$$\vec{X}, \Gamma, \vec{a} : \vec{T}, \vec{b} : \vec{V} \vDash [\vec{U}[\sigma]/X, \sigma]P \,|\, R'[\vec{U}[\sigma]/X, \sigma] \;\mathcal{R}\; [\vec{W}/\vec{X}, \rho]Q' \,|\, R'[\vec{W}/\vec{X}, \rho]$$

and hence by definition of $\mathcal{R}^\nu$:

$$\Gamma \vDash [\sigma]\nu(\vec{a} : \vec{T}[\sigma])(P \,|\, R'[\vec{U}[\sigma]/X, \sigma]) \;\mathcal{R}^\nu\; [\rho]\nu(\vec{a} : \vec{T}[\rho])(Q' \,|\, R'[\vec{W}/\vec{X}, \rho])$$

which is as required.

B Completeness of bisimulation for contextual equivalence

Definition 14. *We define a typed relation $\cong^p$ on closed configurations by asking that $\cong^p$ be the largest relation which is symmetric, reduction closed, barb preserving (with these concepts lifted to configurations in the obvious way), and is closed with respect to the following condition:*

$$\Gamma, \Gamma' \vDash [\sigma]P \,|\, R[\sigma] \;\mathcal{R}\; [\rho]Q \,|\, R[\rho] \quad \begin{array}{l} \textit{whenever } \Gamma \vDash [\sigma]P \approx [\rho]Q \\ \textit{and } \Gamma, \Gamma' \vdash R \\ \textit{and } \sigma \textit{ and } \rho \textit{ are type substitutions} \end{array}$$

Note immediately, that $\Gamma \vDash P \cong Q$ implies $\Gamma \vDash P \cong^p Q$. Therefore it is sufficient to prove completeness of $\approx$ with respect to $\cong^p$. Before we can do this we show two Propositions which will be used to execute the proof. We omit the proofs of these as they follow the lines of similar propositions for the (higher-order) π-calculus [15].

Proposition 9 (Output Contextuality). *For any closed configuration $(\Gamma \vdash [\sigma]P)$, if*

$$(\Gamma \vdash [\sigma]P) \xrightarrow{\nu(\vec{a})\overline{c}\langle \vec{X}; \vec{b} : \vec{V} \rangle} (\Gamma' \vdash [\sigma']P')$$

where $P \xrightarrow{\nu(\vec{a} : \vec{T})\overline{c}\langle \vec{U}; \vec{b} \rangle} P'$ then there exists some process R, and $\mathsf{ext}, \mathsf{fail} \notin \mathsf{dom}(\Gamma)$ such that

$$\Gamma, \mathsf{ext} : \updownarrow[\vec{X}; \vec{V}], \mathsf{fail} : \updownarrow[] \vdash R$$

and

$$P \,|\, R[\sigma] \implies \nu(\vec{a} : \vec{T})(P' \,|\, \overline{\mathsf{ext}}\langle \vec{U}; \vec{b} \rangle) \qquad \nu(\vec{a} : \vec{T})(P' \,|\, \overline{\mathsf{ext}}\langle \vec{U}; \vec{b} \rangle) \not\downarrow_{\mathsf{fail}}.$$

Moreover, for any closed configuration $(\Gamma \vdash [\rho]Q)$ such that $Q \,|\, R[\rho] \implies Q''$ with $Q'' \not\downarrow_{\mathsf{fail}}$ we have

$$Q'' \equiv \nu(\vec{a} : \vec{T}')(Q' \,|\, \overline{\mathsf{ext}}\langle \vec{W}; \vec{b} \rangle)$$

and $Q \xrightarrow{\nu(\vec{a} : \vec{T}')\overline{c}\langle \vec{W}; \vec{b} \rangle} Q'$ for some $\vec{W}$.

Proof. (Outline) We show how to define the process R in two simplified cases, from which the proof for the general case can be inductively derived.

Firstly, suppose the typed output transition is labelled $\overline{c}\langle \vec{X}; b : V \rangle$ and is derived from an underlying untyped action labelled $\overline{c}\langle \vec{U}; b \rangle$. We know then, that $\Gamma \vdash c(\vec{X}; x : V) \,.\, \mathbf{0}$ and $\sigma' = (\vec{U}/\vec{X}, \sigma)$ and $\Gamma' = (\vec{X}, \Gamma, b : V)$. From Proposition 5 we have $\Gamma[\sigma] \vdash \overline{c}\langle \vec{U}; b \rangle$, and so $b \in \mathsf{dom}(\Gamma)$. Thus, we have that the process

$$R = \overline{\mathsf{fail}}\langle\rangle \,|\, c(\vec{X}; x : V) \,.\, \mathsf{if}\, x = b \,\mathsf{then}\, \mathsf{fail}() \,.\, \overline{\mathsf{ext}}\langle \vec{X}; x \rangle \,\mathsf{else}\, \mathbf{0}$$

is well-typed with respect to $\Gamma, \mathsf{ext} : \updownarrow[\vec{X};V], \mathsf{fail} : \updownarrow[]$, and $P \mid R[\sigma]$ will reduce as required. Moreover, for any Q, ρ such that $Q \mid R[\rho] \Longrightarrow Q'' \not\downarrow_{\mathsf{fail}}$ we must have a communication on fail, which demands an earlier communication on c of the form $\overline{c}\langle \vec{W}; b\rangle$. These facts tell us that $Q'' \equiv (Q' \mid \overline{\mathsf{ext}}\langle \vec{W}; b\rangle)$ such that $Q \xRightarrow{\overline{c}\langle \vec{W}; b\rangle} Q'$ as required.

Secondly, suppose that the typed output transition is labelled $\nu(b)\overline{c}\langle \vec{X}; b : V\rangle$ and is dervied from an underlying untyped action labelled $\nu(b : T)\overline{c}\langle \vec{U}; b\rangle$. We let R be defined as

$$\begin{aligned} R = \overline{\mathsf{fail}}\langle\rangle \mid c(\vec{X}; x : V).\ (&\mathsf{if}\ x = a_1\ \mathsf{then}\ \mathbf{0} \\ &\mathsf{else\ if}\ x = a_2\ \mathsf{then}\ \mathbf{0} \\ &\mathsf{else\ if}\ \ldots \\ &\vdots \\ &\mathsf{else}\ \mathsf{fail}().\,\overline{\mathsf{ext}}\langle \vec{X}; x\rangle) \end{aligned}$$

where $\vec{a}_i$ are all of the names such that $a_i \in \mathsf{dom}(\Gamma)$. It is easy to check that R is well-typed with respect to $\Gamma, \mathsf{ext} : \updownarrow[\vec{X};V], \mathsf{fail} : \updownarrow[]$. It is also easy to check that $P \mid R[\sigma]$ reduces as expected. This is because we know that b is fresh to Γ, and so $a_i \neq b$ for any of the a_i. For any other closed configuration $(\Gamma \vdash [\rho]Q)$ such that $Q \mid R[\rho] \Longrightarrow Q'' \not\downarrow_{\mathsf{fail}}$, we know similarly that there must have been an communication on c of the form $\nu(b' : T')\overline{c}\langle \vec{W}; b'\rangle$. This name must be different to every a_i, and so must be fresh, and hence can be α-converted to b, hence $Q'' \equiv \nu(b : T')(Q' \mid \overline{\mathsf{ext}}\langle \vec{W}; b\rangle)$ and $Q \xRightarrow{\nu(b:T')\overline{c}\langle \vec{W}; b\rangle} Q'$ as required.

Proposition 10 (Extrusion). *If*

$$\Gamma, \mathsf{ext} : \updownarrow[\vec{X};\vec{V}] \vDash [\sigma]\nu(\vec{a} : \vec{T})(P \mid \overline{\mathsf{ext}}\langle \vec{U}; \vec{b}\rangle) \cong^p [\rho]\nu(\vec{a} : \vec{T}')(Q \mid \overline{\mathsf{ext}}\langle \vec{W}; \vec{b}\rangle)$$

with $\vec{a} \subseteq \vec{b}$ *and* $\mathsf{ext} \notin \mathsf{fn}(P, Q)$ *then*

$$\vec{X}; \Gamma, \vec{b} : \vec{V} \vDash [\vec{U}/\vec{X}, \sigma]P \cong^p [\vec{W}/\vec{X}, \rho]Q.$$

Proof. The proof of this is again similar to the proof of the analogous property in [15].

We can now prove Theorem 3: if $\Gamma \vDash P \cong Q$ then $\Gamma \vDash P \approx^{\circ} Q$.

Proof. If suffices to prove the result for closed processes and for $\cong^p$ in place of $\cong$. We proceed by coinduction by defining $\mathcal{R}$ to be

$$\Gamma \vDash [\sigma]P\ \mathcal{R}\ [\rho]Q \text{ whenever } \Gamma \vDash [\sigma]P \cong^p [\rho]Q$$

and showing that $\mathcal{R}$ forms a bisimulation up to $\equiv$. Suppose that $\Gamma \vDash [\sigma]P\ \mathcal{R}\ [\rho]Q$ and further suppose that $(\Gamma \vdash [\sigma]P) \xrightarrow{\alpha} (\Gamma' \vdash [\sigma']P')$. We must show that $(\Gamma \vdash [\rho]Q)$ has a matching transition. This is straightforward in the cases in which α is generated by rules (TR-SILENT) or (TR-RECEP). Otherwise, α is generated by rule (TR-OUT-W), that is

- α is of the form $\nu(\vec{a})\overline{c}\langle \vec{X}; \vec{b} : \vec{V}\rangle$,
- Γ' is $\vec{X}; \Gamma, \vec{b} : \vec{V}$
- σ' is $[\vec{U}/\vec{X}, \sigma]$
- and $P \xrightarrow{\nu(\vec{a}:\vec{T})\overline{c}\langle \vec{U}; \vec{b}\rangle} P'$

We can now appeal to Proposition 9 to find a process R such that $\Gamma, \mathsf{ext} : \updownarrow[\vec{X};\vec{V}], \mathsf{fail} : \updownarrow[] \vdash R$ and

$$P \mid R[\sigma] \Longrightarrow \nu(\vec{a} : \vec{T})(P' \mid \overline{\mathsf{ext}}\langle \vec{U}; \vec{b}\rangle) \qquad \nu(\vec{a} : \vec{T})(P' \mid \overline{\mathsf{ext}}\langle \vec{U}; \vec{b}\rangle) \not\downarrow_{\mathsf{fail}}.$$

We know that $\Gamma \vDash [\sigma]P \cong^p [\rho]Q$ and, by definition, this gives us

$$\Gamma, \mathsf{ext} : \updownarrow[\vec{X};\vec{V}], \mathsf{fail} : \updownarrow[] \vDash [\sigma]P \mid R[\sigma] \cong^p Q \mid R[\rho]$$

also. As $\cong^p$ is reduction-closed and barb-preserving, we must have $Q \mid R[\sigma] \Longrightarrow Q''$ for some $Q'' \not\downarrow_{\mathsf{fail}}$ such that (strengthening to remove fail from the environment)

$$\Gamma, \mathsf{ext} : \updownarrow[\vec{X};\vec{V}] \vDash \nu(\vec{a} : \vec{T})(P' \mid \overline{\mathsf{ext}}\langle\vec{U};\vec{b}\rangle) \cong^p Q''. \tag{1}$$

By Proposition 9 we have $Q'' \equiv \nu(\vec{a} : \vec{T}')(Q' \mid \overline{\mathsf{ext}}\langle\vec{W};\vec{b}\rangle)$ for some Q' and $\vec{W}$ and $Q \xrightarrow{\nu(\vec{a}:\vec{T}')\bar{c}\langle\vec{W};\vec{b}\rangle} Q'$. This tells us that

$$(\Gamma \vdash [\rho]Q) \xrightarrow{\alpha} (\Gamma' \vdash [\vec{W}/\vec{X}, \rho]Q')$$

and moreover, by applying Proposition 10 to (1), we see that

$$\vec{X};\Gamma, \vec{b} : \vec{V} \vDash [\vec{U}/\vec{X}, \sigma]P' \cong^p [\vec{W}/\vec{X}, \rho]Q'.$$

which is to say

$$\Gamma' \vDash [\sigma']P' \mathrel{\mathcal{R}} [\vec{W}/\vec{X}, \rho]Q'.$$

as required.

C Pierce and Sangiorgi's polymorphic bisimulation is our unifying bisimulation

Pierce and Sangiorgi's definition of polymorphic bisimulation relies on an 'allow relation' [22, Defn 12.1.1] which, rewritten to fit our notation, is almost the same as in Definition 15. The 'almost' is the addition of the condition '$\vec{T}$ are generative' to Rule A-INP which is missing in their formulation: this appears to be a slight error in their definition.

Definition 15 (Allow Relation). *The allow relation* $(\Gamma \parallel \sigma) \xrightarrow{\mu} (\Gamma' \parallel \sigma')$*, where* $\Gamma[\sigma]$ *and* $\Gamma'[\sigma']$ *are closed, is defined by:*

$$\frac{}{(\Gamma \parallel \sigma) \xrightarrow{\tau} (\Gamma \parallel \sigma)} \text{(A-TAU)}$$

$$\frac{\Gamma, \vec{a} : \vec{T} \vdash \bar{c}\langle\vec{U};\vec{b}\rangle \quad \{\vec{a}\} \cap \mathsf{dom}(\Gamma) = \emptyset \quad \vec{T} \textit{ are generative}}{(\Gamma \parallel \sigma) \xrightarrow{c[\vec{U}[\sigma];\vec{b}]} (\Gamma, \vec{a} : \vec{T} \parallel \sigma)} \text{(A-INP)}$$

$$\frac{\begin{array}{c}\Gamma \vdash c(\vec{X};\vec{x} : \vec{V}) \quad \vec{Y}, \Gamma, \vec{a} : \vec{Y} \vdash \vec{b} : \vec{W} \\ \{\vec{a}, \vec{X}, \vec{Y}\} \cap \mathsf{dom}(\Gamma) = \emptyset \quad (\mathsf{mgu}(\vec{V}, \vec{W});\sigma') = (\vec{T}/\vec{Y}, \sigma)\end{array}}{(\Gamma \parallel \sigma) \xrightarrow{\nu(\vec{a}:\vec{T})\bar{c}\langle\vec{U};\vec{b}\rangle} ((\vec{X}, \vec{Y}, \Gamma, \vec{a} : \vec{Y})[\mathsf{mgu}(\vec{V}, \vec{W})] \parallel \sigma')} \text{(A-OUT)}$$

The weak, asynchronous formulation of Pierce and Sangiorgi's definition of polymorphic bisimulation [22, Defn 12.2.2] is then as in Definition 16. Readers familiar with their paper will note that this is the definition without clause 3(a), which is their conjectured fully abstract model.

Definition 16 (Polymorphic bisimulation). *A polymorphic (asynchronous weak) simulation* $\mathcal{R}$ *is a typed relation on closed configurations such that if* $\Gamma \vDash [\sigma]P \mathrel{\mathcal{R}} [\rho]Q$ *then:*

1. *if* $P \xrightarrow{\tau} P'$ *then we have* $Q \Longrightarrow Q'$ *for some* $\Gamma \vDash [\sigma]P' \mathrel{\mathcal{R}} [\rho]Q'$*;*
2. *if* $(\Gamma \parallel \sigma) \xrightarrow{c[\vec{U};\vec{b}]} (\Gamma' \parallel \sigma')$ *then* $(\Gamma \parallel \rho) \xrightarrow{c[\vec{W};\vec{b}]} (\Gamma' \parallel \rho')$ *and* $(Q \mid \bar{c}\langle\vec{W};\vec{b}\rangle) \Longrightarrow Q'$ *for some* $\Gamma' \vDash [\sigma'](P \mid \bar{c}\langle\vec{U};\vec{b}\rangle) \mathrel{\mathcal{R}} [\rho']Q'$*;*

3. *if* $(\Gamma \parallel \sigma) \xrightarrow{\nu(\vec{a}:\vec{T})\overline{c}\langle\vec{U};\vec{b}\rangle} (\Gamma' \parallel \sigma')$ *and* $P \xrightarrow{\nu(\vec{a}:\vec{T})\overline{c}\langle\vec{U};\vec{b}\rangle} P'$ *then* $(\Gamma \parallel \rho) \xrightarrow{\nu(\vec{a}:\vec{V})\overline{c}\langle\vec{W};\vec{b}\rangle} (\Gamma' \parallel \rho')$ *and* $Q \xRightarrow{\nu(\vec{a}:\vec{V})\overline{c}\langle\vec{W};\vec{b}\rangle} Q'$ *for some* $\Gamma' \vDash [\sigma']P' \mathcal{R} [\rho']Q'$.

A polymorphic bisimulation is a simulation whose inverse is also a simulation. Let $\simeq$ *be the largest polymorphic bisimulation.*

Proposition 11. $\simeq$ *and* $\approx_{\mathsf{u}}$ *coincide.*

Proof. We have to show two properties: $\approx_{\mathsf{u}}$ is a polymorphic simulation, and $\simeq$ is a simulation. We consider each of these in turn. For convenience we will drop the subscript on $\approx_{\mathsf{u}}$ for the remainder of this proof.

$\approx$ is a polymorphic simulation. Consider any $\Gamma \vDash [\sigma]P \approx [\rho]Q$.

1. If $P \xrightarrow{\tau} P'$ then by Rule TR-SILENT and the definition of bisimulation, we have $Q \Longrightarrow Q'$ for some $\Gamma \vDash [\sigma]P' \approx [\rho]Q'$ as required.
2. If $(\Gamma \parallel \sigma) \xrightarrow{c[\vec{U}[\sigma];\vec{b}]} (\Gamma, \vec{a} : \vec{T} \parallel \sigma)$ then by Rule A-INP we have:
$$\Gamma, \vec{a} : \vec{T} \vdash \overline{c}\langle\vec{U};\vec{b}\rangle \quad \{\vec{a}\} \cap \mathsf{dom}(\Gamma) = \emptyset \quad \vec{T} \text{ are generative}$$
and so we also have:
$$(\Gamma \parallel \rho) \xrightarrow{c[\vec{U}[\rho];\vec{b}]} (\Gamma, \vec{a} : \vec{T} \parallel \rho)$$
Moreover, we have:
$$(\Gamma \vdash [\sigma]P) \xrightarrow{\nu(\vec{a}:\vec{T})c[\vec{U};\vec{b}]} (\Gamma, \vec{a} : \vec{T} \vdash [\sigma]P \mid \overline{c}\langle\vec{U}[\sigma];\vec{b}\rangle)$$
and so by definition of bisimulation:
$$(\Gamma \vdash [\sigma]Q) \xRightarrow{\nu(\vec{a}:\vec{T})c[\vec{U};\vec{b}]} (\Gamma, \vec{a} : \vec{T} \vdash [\rho]Q') \quad \Gamma, \vec{a} : \vec{T} \vDash [\sigma]P \mid \overline{c}\langle\vec{U};\vec{b}\rangle \approx Q'$$
which must come from Rules TR-SILENT and TR-RECEP where:
$$Q \Longrightarrow Q'' \quad Q'' \mid \overline{c}\langle\vec{U}[\rho];\vec{b}\rangle \Longrightarrow Q'$$
and so from R-PAR we have:
$$Q \mid \overline{c}\langle\vec{U}[\rho];\vec{b}\rangle \Longrightarrow Q'$$
as required.
3. If $(\Gamma \parallel \sigma) \xrightarrow{\nu(\vec{a}:\vec{T})\overline{c}\langle\vec{U};\vec{b}\rangle} (\Gamma' \parallel \sigma')$ and $P \xrightarrow{\nu(\vec{a}:\vec{T})\overline{c}\langle\vec{U};\vec{b}\rangle} P'$ then by Rule A-OUT we have:
$$\Gamma' = (\vec{X}, \vec{Y}, \Gamma, \vec{a} : \vec{Y})[\mathsf{mgu}(\vec{V}, \vec{W})] \quad \Gamma \vdash c(\vec{X};\vec{x} : \vec{V}) \quad \vec{Y}, \Gamma, \vec{a} : \vec{Y} \vdash \vec{b} : \vec{W}$$
$$\{\vec{a}, \vec{X}, \vec{Y}\} \cap \mathsf{dom}(\Gamma) = \emptyset \quad (\mathsf{mgu}(\vec{V}, \vec{W}); \sigma') = (\vec{T}/\vec{Y}, \sigma)$$
and by Rule TR-OUT-U we have:
$$(\Gamma \vdash [\sigma]P) \xrightarrow{\nu(\vec{a}:\vec{Y})\overline{c}\langle\vec{X};\vec{b}:\vec{V}\rangle} (\Gamma' \vdash [\sigma']P')$$
and so, by definition of $\approx$ we have:
$$(\Gamma \vdash [\rho]Q) \xRightarrow{\nu(\vec{a}:\vec{Y})\overline{c}\langle\vec{X};\vec{b}:\vec{V}\rangle} (\Gamma' \vdash [\rho']Q') \quad \Gamma' \vDash [\sigma']P' \approx [\rho']Q'$$
which must come from Rules TR-SILENT and TR-OUT-U where:
$$Q \xRightarrow{\nu(\vec{a}:\vec{T}')\overline{c}\langle\vec{U}';\vec{b}\rangle} Q'$$
as required.

$\simeq$ **is a simulation.** Consider any $\Gamma \vDash [\sigma]P \simeq [\rho]Q$.

1. If $(\Gamma \vdash [\sigma]P) \xrightarrow{\tau} (\Gamma \vdash [\sigma]P')$ then by Rule TR-SILENT and the definition of polymorphic bisimulation, we have $(\Gamma \vdash [\rho]Q) \Longrightarrow (\Gamma \vdash [\rho]Q')$ for some $\Gamma \vDash [\sigma]P' \simeq [\rho]Q'$ as required.
2. If $(\Gamma \vdash [\sigma]P) \xrightarrow{\nu(\vec{a}:\vec{T})c[\vec{U};\vec{b}]} (\Gamma, \vec{a} : \vec{T} \vdash [\sigma]P')$ then by Rule TR-RECEP we have:

   $$P' = P \mid \overline{c}\langle \vec{U}; \vec{b} \rangle[\sigma] \quad \Gamma, \vec{a} : \vec{T} \vdash \overline{c}\langle \vec{U}; \vec{b} \rangle \quad \{\vec{a}\} \cap \mathsf{dom}(\Gamma) = \emptyset \quad \vec{T} \text{ are generative}$$

   and so by Rule A-INP we have:

   $$(\Gamma \parallel \sigma) \xrightarrow{c[\vec{U}[\sigma];\vec{b}]} (\Gamma, \vec{a} : \vec{T} \parallel \sigma)$$

   which means by definition of polymorphic bisimulation we have:

   $$Q \mid \overline{c}\langle \vec{U}[\rho]; \vec{b} \rangle \Longrightarrow Q' \quad \Gamma, \vec{a} : \vec{T} \vDash [\sigma]P' \simeq [\rho]Q'$$

   so by Rules TR-RECEP and TR-SILENT we have:

   $$(\Gamma \vdash [\rho]Q) \xRightarrow{\nu(\vec{a}:\vec{T})c[\vec{U};\vec{b}]} (\Gamma, \vec{a} : \vec{T} \vdash [\rho]Q')$$

   as required.
3. If $(\Gamma \vdash [\sigma]P) \xrightarrow{\nu(\vec{a}:\vec{Y})\overline{c}\langle \vec{X}; \vec{b}:\vec{V} \rangle} (\Gamma' \vdash [\sigma']P')$ then by TR-OUT-U we have:

   $$\begin{gathered} \Gamma' = (\vec{X}, \vec{Y}, \Gamma, \vec{b} : \vec{Y})[\mathsf{mgu}(\vec{V}, \vec{W})] \quad P \xrightarrow{\nu(\vec{a}:\vec{T})\overline{c}\langle \vec{U}; \vec{b} \rangle} P' \\ \Gamma \vdash c(\vec{X}; \vec{x} : \vec{V}) \quad \vec{Y}, \Gamma, \vec{a} : \vec{Y} \vdash \vec{b} : \vec{W} \\ \{\vec{a}, \vec{X}, \vec{Y}\} \cap \mathsf{dom}(\Gamma) = \emptyset \quad (\mathsf{mgu}(\vec{V}, \vec{W}); \sigma') = (\vec{T}/\vec{Y}, \sigma) \end{gathered}$$

   and so by Rule A-OUT we have:

   $$(\Gamma \parallel \sigma) \xrightarrow{\nu(\vec{a}:\vec{T})\overline{c}\langle \vec{U}; \vec{b} \rangle} ((\vec{X}, \vec{Y}, \Gamma, \vec{a} : \vec{Y})[\mathsf{mgu}(\vec{V}, \vec{W})] \parallel \sigma')$$

   which means by the definition of polymorphic bisimulation we have:

   $$(\Gamma \parallel \rho) \xrightarrow{\nu(\vec{a}:\vec{T}')\overline{c}\langle \vec{U}'; \vec{b} \rangle} (\Gamma' \parallel \sigma') \quad Q \xRightarrow{\nu(\vec{a}:\vec{T}')\overline{c}\langle \vec{U}'; \vec{b} \rangle} Q' \quad \Gamma' \vDash [\sigma']P' \simeq [\rho']Q'$$

   so by Rule A-OUT we have:

   $$(\mathsf{mgu}(\vec{V}, \vec{W}); \rho') = (\vec{T}'/\vec{Y}, \rho)$$

   and hence by Rules TR-OUT-U and TR-SILENT we have:

   $$(\Gamma \vdash [\rho]Q) \xRightarrow{\nu(\vec{a}:\vec{Y})\overline{c}\langle \vec{X}; \vec{b} \rangle} (\Gamma' \vdash [\rho']Q')$$

   as required.

Thus, $\approx$ and $\simeq$ coincide.

D Strong typing for if-then-else

Definition 17 (Strong typing). *Write $\Gamma \vdash_{\mathsf{s}} P$ when the process typing $\Gamma \vdash P$ can be derived using Rule* T-TEST-S *in place of* T-TEST-W.

Definition 18 (Strong typed contextual equivalence). *Let $\cong_{\mathsf{s}}$ be the contextual equivalence generated by type system $\Gamma \vdash_{\mathsf{s}} P$.*

Definition 19 (Strong closing substitution). *A substitution σ strongly closes Γ if:*

1. $\mathsf{dom}(\sigma) \subseteq \mathsf{dom}(\Gamma)$,
2. $\Gamma[\sigma]$ *is closed,*
3. *for any* $n : T \in \Gamma$ *and* $m : U \in \Gamma$, *if* $n = m$ *then* $T = U$, *and*
4. *for any* $n : T \in \Gamma$ *and* $m : U \in \Gamma$, *if* $n[\sigma] = m[\sigma]$ *and* $T = U$ *then* $n = m$.

Note that the empty substitution strongly closes any closed Γ. A configuration $(\Gamma \vdash [\sigma]P)$ is strongly closed whenever $\Gamma[\sigma] \vdash P$ and σ strongly closes Γ: for the remainder of this section all configurations are considered to be strongly closing.

Definition 20 (Strong typed labelled transitions). *Write* $C \xrightarrow{\alpha}_{\mathsf{s}} C'$ *when the labelled transition* $C \xrightarrow{\alpha} C'$ *can be derived using Rule* TR-OUT-S *in place of* TR-OUT-W.

$$\frac{P \xrightarrow{\nu(\vec{a}:\vec{T})\overline{c}\langle\vec{U};\vec{b}\rangle} P' \quad \Gamma \vdash n(\vec{X};\vec{x}:\vec{V}).\mathbf{0} \quad (n,\vec{m})[\sigma'] = (c,\vec{b}) \quad \sigma' = (\vec{U}/\vec{X},\sigma,\vec{c}/\vec{z}) \text{ strongly closes } \Gamma' = (\vec{X},\Gamma,\vec{m}:\vec{V})}{(\Gamma \vdash [\sigma]P) \xrightarrow{\nu(\vec{z})\overline{n}\langle\vec{X};\vec{m}:\vec{V}\rangle} (\Gamma' \vdash [\sigma']P')} \text{(TR-OUT-S)}$$

Note, we must also modify Rule TR-RECEP slightly, to use variables, $\vec{x}$ in place of names $\vec{a}$, and to use values $\vec{n}, m$ in place of names $\vec{b}, c$.

Definition 21 (Strong typed bisimulation). *Write* $\approx_{\mathsf{s}}$ *for the bisimulation generated by the labelled transition system* $C \xrightarrow{\alpha}_{\mathsf{s}} C'$.

Theorem 4 (Full abstraction of strong typed bisimulation for strong typed contextual equivalence). $\Gamma \vDash P \approx_{\mathsf{s}}^{\circ} Q$ *if and only if* $\Gamma \vDash P \cong_{\mathsf{s}} Q$.

Proof. The proof of this follows along the same lines as the proof of Theorems 2 and 3. The significant differences occur in the Output Contextuality and Extrusion Propositions. We will outline the changes to these below.

Proposition 12 (Strong Typing Output Contextuality). *For any strongly closed configuration* $(\Gamma \vdash [\sigma]P)$, *if*

$$(\Gamma \vdash [\sigma]P) \xrightarrow{\nu(\vec{z})\overline{n}\langle\vec{X};\vec{m}:\vec{V}\rangle} (\Gamma' \vdash [\sigma']P')$$

where $P \xrightarrow{\nu(\vec{a}:\vec{T})\overline{c}\langle\vec{U};\vec{b}\rangle} P'$ *then there exists some process* R, *and* $\mathsf{ext}, \mathsf{fail} \notin \mathsf{dom}(\Gamma)$ *such that*

$$\Gamma, \mathsf{ext} : \updownarrow[\vec{X};\vec{V}], \mathsf{fail} : \updownarrow[] \vdash R$$

and

$$P \,|\, R[\sigma] \Longrightarrow \nu(\vec{a}:\vec{T})(P' \,|\, \overline{\mathsf{ext}}\langle\vec{U};\vec{b}\rangle) \qquad \nu(\vec{a}:\vec{T})(P' \,|\, \overline{\mathsf{ext}}\langle\vec{U};\vec{b}\rangle) \not\downarrow_{\mathsf{fail}}.$$

Moreover, for any strongly closed $(\Gamma \vdash [\rho]Q)$ *such that* $Q \,|\, R[\rho] \Longrightarrow Q''$ *with* $Q'' \not\downarrow_{\mathsf{fail}}$ *we have*

$$Q'' \equiv \nu(\vec{a}':\vec{T}')(Q' \,|\, \overline{\mathsf{ext}}\langle\vec{W};\vec{b}'\rangle)$$

and $Q \xRightarrow{\nu(\vec{a}':\vec{T}')\overline{c}'\langle\vec{W};\vec{b}'\rangle} Q'$ *where* $(n,\vec{m})[\rho'] = (c',\vec{b}')$ *and* $\rho' = (\vec{W}/\vec{X},\rho,\vec{c}'/\vec{z})$ *strongly closes* Γ'.

Proof. (Outline) Again, we show how to define the process R in two simplified cases, from which the proof for the general case can be inductively derived.

Firstly, suppose the typed output transition is labelled $\overline{n}\langle\vec{X};m:V\rangle$ and is derived from an underlying untyped action labelled $\overline{c}\langle\vec{U};b\rangle$. We know then, that $\sigma' = (\vec{U}/\vec{X},\sigma)$ and $\Gamma' = (\vec{X},\Gamma)$,

so by definition of strong closure, $m : V \in \Gamma$, and moreover, $(n,m)[\sigma] = (c,b)$. In particular, we notice that the process

$$R = \overline{\mathsf{fail}}\langle\rangle \mid n(\vec{X}; x : V)\,.\,\mathsf{if}\, x = m\,\mathsf{then}\,\mathsf{fail}()\,.\,\overline{\mathsf{ext}}\langle\vec{X}; x\rangle\,\mathsf{else}\,\mathbf{0}$$

is well-typed with respect to $\Gamma, \mathsf{ext} : \updownarrow[\vec{X}; V], \mathsf{fail} : \updownarrow[]$, and $P \mid R[\sigma]$ will reduce as required. Moreover, for any Q, ρ such that $Q \mid R[\rho] \Longrightarrow Q'' \not\downarrow_{\mathsf{fail}}$ we must have a communication on fail, which demands an earlier communication on $c' = n[\rho]$ in which a name b', say, is sent from Q. Furthermore, we know that $b' = m[\rho]$ as the conditional test is passed successfully. These facts tell us that $Q'' \equiv (Q' \mid \overline{\mathsf{ext}}\langle\vec{W}; b'\rangle)$ such that $Q \xRightarrow{\overline{c'}\langle\vec{W};b'\rangle} Q'$ as required. Since $\rho' = (\vec{W}/\vec{X}, \rho)$ and ρ strongly closes Γ, we have that ρ' strongly closes Γ' as required.

Secondly, suppose that the typed output transition is labelled $\nu(y)\overline{n}\langle\vec{X}; y : V\rangle$. We let R be defined as

$$\begin{aligned} R = \overline{\mathsf{fail}}\langle\rangle \mid n(\vec{X}; x : V).\ (&\mathsf{if}\ x = n_1\ \mathsf{then}\ \mathbf{0} \\ &\mathsf{else\ if}\ x = n_2\ \mathsf{then}\ \mathbf{0} \\ &\mathsf{else\ if}\ \ldots \\ &\vdots \\ &\mathsf{else}\ \mathsf{fail}()\,.\,\overline{\mathsf{ext}}\langle\vec{X}; x\rangle) \end{aligned}$$

where $\vec{n}_i$ are all of the values such that $n : V \in \Gamma$. It is easy to check that R is well-typed with respect to $\Gamma, \mathsf{ext} : \updownarrow[\vec{X}; V], \mathsf{fail} : \updownarrow[]$. It is also easy to check that $P \mid R[\sigma]$ reduces as expected. This is because we know that y is fresh to Γ, and σ' strongly closes $n_i[\sigma] \neq b$ for any of the n_i. For any other process and mapping, Q, ρ such that $Q \mid R[\rho] \Longrightarrow Q'' \not\downarrow_{\mathsf{fail}}$, we know similarly that there must have been an output from Q along channel $c' = n[\rho]$ of a name b', say. This name must be different to every $n_i[\rho]$ so it must be the case that, $\rho' = (\vec{W}/\vec{X}, \rho, b'/y)$ and since ρ strongly closes Γ, we have that ρ' strongly closes Γ' as required.

Proposition 13 (Strong typing extrusion). *If*

$$\Gamma, \mathsf{ext} : \updownarrow[\vec{X}; \vec{V}] \vDash [\sigma]\nu(\vec{a} : \vec{T})(P \mid \overline{\mathsf{ext}}\langle\vec{U}; \vec{b}\rangle) \cong^p [\rho]\nu(\vec{a}' : \vec{T}')(Q \mid \overline{\mathsf{ext}}\langle\vec{W}; \vec{b}'\rangle)$$

with $\vec{a} \subseteq \vec{b}$, $\vec{a}' \subseteq \vec{b}'$, $\mathsf{ext} \notin \mathsf{fn}(P, Q)$ *and*

$$\sigma' = (\vec{U}/\vec{X}, \sigma, \vec{b}/\vec{y}) \text{ and } \rho' = (\vec{W}/\vec{X}, \sigma, \vec{b}'/\vec{y}) \text{ both strongly close } \Gamma' = (\vec{X}, \Gamma, \vec{y} : \vec{V})$$

then $\Gamma' \vDash [\sigma']P \cong^p [\rho']Q$.

Proof. The proof of this is again similar to the proof of the analogous property in [15].

E Unifying typing for if-then-else

Definition 22 (Unifying typing). *Write* $\Gamma \vdash_{\mathsf{u}} P$ *when the process typing* $\Gamma \vdash P$ *can be derived using Rule* T-TEST-U *in place of* T-TEST-W.

Definition 23 (Unifying typed contextual equivalence). *Let* $\cong_{\mathsf{u}}$ *be the contextual equivalence generated by type system* $\Gamma \vdash_{\mathsf{u}} P$.

Definition 24 (Unifying typed labelled transitions). *Write* $C \xrightarrow{\alpha}_{\mathsf{u}} C'$ *when the labelled transition* $C \xrightarrow{\alpha} C'$ *can be derived using Rule* TR-OUT-U *in place of* TR-OUT-W.

$$\frac{P \xrightarrow{\nu(\vec{a}:\vec{T})\overline{c}\langle\vec{U};\vec{b}\rangle} P' \quad \Gamma \vdash c(\vec{X}; \vec{x} : \vec{V}) \quad \vec{Y}, \Gamma, \vec{a} : \vec{Y} \vdash \vec{b} : \vec{W} \quad \{\vec{a}, \vec{X}, \vec{Y}\} \cap \mathsf{dom}(\Gamma) = \emptyset \quad (\mathsf{mgu}(\vec{V}, \vec{W}); \sigma') = (\vec{T}, \vec{U}/\vec{Y}, \vec{X}, \sigma)}{(\Gamma \vdash [\sigma]P) \xrightarrow{\nu(\vec{a}:\vec{Y})\overline{c}\langle\vec{X};\vec{b}:\vec{V}\rangle} ((\vec{X}, \vec{Y}, \Gamma, \vec{a} : \vec{Y})[\mathsf{mgu}(\vec{V}, \vec{W})] \vdash [\sigma']P')} \text{(TR-OUT-U)}$$

Definition 25 (Unifying typed bisimulation). *Write* $\approx_{\mathsf{u}}$ *for the bisimulation generated by the labelled transition system* $C \xrightarrow{\alpha}_{\mathsf{u}} C'$.

Theorem 5 (Full abstraction of unifying typed bisimulation for unifying typed contextual equivalence). $\Gamma \vDash P \approx_{\mathsf{u}}^{\circ} Q$ *if and only if* $\Gamma \vDash P \cong_{\mathsf{u}} Q$.

Proof. The proof of this also follows similar lines to Theorems 2 and 3 so we do not repeat the details here. As in the previous section though, the significant changes to the proof lie in the Output Contextuality and Extrusion Propositions. We show these below.

The main change that needs to occur is that after testing for each output action, the process which receives this output must re-emit on the ext channel, not just the values communicated but also representative values for the entire environment. This is because the type unification allows us to update the types of previously emitted values. The following notation is useful: write $\vec{a}_\Gamma$ to mean $\mathsf{fn}(\Gamma)$ rendered as a value and write (Γ) to mean the collection of types T such that $a : T \in \Gamma$ for some a.

Proposition 14 (Unifying Output Contextuality). *For any closed configuration* $(\Gamma \vdash [\sigma]P)$*, if*

$$(\Gamma \vdash [\sigma]P) \xrightarrow{\nu(\vec{a}:\vec{Y})\bar{c}\langle\vec{X};\vec{b}:\vec{V}\rangle} (\Gamma' \vdash [\sigma']P')$$

where $P \xrightarrow{\nu(\vec{a}:\vec{T})\bar{c}\langle\vec{U};\vec{b}\rangle} P'$ *then there exists some process* R*, and* $\mathsf{ext}, \mathsf{fail} \notin \mathsf{dom}(\Gamma)$ *such that*

$$\Gamma, \mathsf{ext} : \updownarrow[\vec{X}; (\Gamma')], \mathsf{fail} : \updownarrow[] \vdash R$$

and

$$P \,|\, R[\sigma] \Longrightarrow \nu(\vec{a} : \vec{T})(P' \,|\, \overline{\mathsf{ext}}\langle\vec{U}; \vec{a}_{\Gamma'}\rangle) \qquad \nu(\vec{a} : \vec{T})(P' \,|\, \overline{\mathsf{ext}}\langle\vec{U}; \vec{a}_{\Gamma'}\rangle) \not\downarrow_{\mathsf{fail}}.$$

Moreover, for any closed configuration $(\Gamma \vdash [\rho]Q)$ *such that* $Q \,|\, R[\rho] \Longrightarrow Q''$ *with* $Q'' \not\downarrow_{\mathsf{fail}}$ *we have*

$$Q'' \equiv \nu(\vec{a} : \vec{T}')(Q' \,|\, \overline{\mathsf{ext}}\langle\vec{W}; \vec{a}_{\Gamma'}\rangle)$$

and $Q \xRightarrow{\nu(\vec{a}:\vec{T}')\bar{c}\langle\vec{W};\vec{b}\rangle} Q'$ *for some* $\vec{W}$.

Proof. (Outline) Again, we only show how to define the process R in the same two simplified cases.

Firstly, suppose the typed output transition is labelled $\bar{z}\langle\vec{X}; b : V\rangle$ and is derived from an underlying untyped action labelled $\bar{c}\langle\vec{U}; b\rangle$. We let

$$R = \overline{\mathsf{fail}}\langle\rangle \,|\, z(\vec{X}; x : V)\,.\,\mathsf{if}\, x = b\,\mathsf{then}\,\mathsf{fail}()\,.\,\overline{\mathsf{ext}}\langle\vec{X}; \vec{a}_\Gamma\rangle\,\mathsf{else}\,\mathbf{0}$$

and check that R is well-typed with respect to the environment $\Gamma, \mathsf{ext} : \updownarrow[\vec{X}; (\Gamma')], \mathsf{fail} : \updownarrow[]$ where $\Gamma' = (\vec{X}, \Gamma)[\mathsf{mgu}(V, W)]$ and $\Gamma \vdash b : W$. This results in checking that

$$\Gamma, \mathsf{ext} : \updownarrow[\vec{X}; (\Gamma')], \mathsf{fail} : \updownarrow[], x : V \vdash \mathsf{if}\, x = b\,\mathsf{then}\,\mathsf{fail}()\,.\,\overline{\mathsf{ext}}\langle\vec{X}; \vec{a}_\Gamma\rangle\,\mathsf{else}\,\mathbf{0}$$

We can see that this holds true though as the type rule T-TEST-U allows us to reduce this to checking that

$$(\vec{X}, \Gamma)[\mathsf{mgu}(V, W)] \vdash \vec{a}_\Gamma : (\Gamma')$$

This follows easily because $(\Gamma') = ((\vec{X}, \Gamma)[\mathsf{mgu}(V, W)])$ and $\mathsf{fn}(\Gamma) = \mathsf{fn}(\Gamma')$. It is easy to see that $P \,|\, R[\sigma]$ reduces as required.

Take, any Q, ρ such that $Q \,|\, R[\rho] \Longrightarrow Q'' \not\downarrow_{\mathsf{fail}}$ we must have a communication on fail, which demands an earlier communication on c in which a name b', say, is sent from Q. Furthermore, we

know that $b' = b$ as the conditional test is passed successfully. Therefore $Q'' \equiv (Q' \mid \overline{\mathsf{ext}}\langle \vec{W}; b \rangle)$ such that $Q \xrightarrow{\overline{c}\langle \vec{W}; b \rangle} Q'$ as required.

Secondly, suppose that the typed output transition is labelled $\nu(b : Y)\overline{c}\langle \vec{X}; b : V \rangle$. We let R be defined as

$$\begin{aligned} R = \overline{\mathsf{fail}}\langle\rangle \mid z(\vec{X}; x : V).\ (&\mathsf{if}\ x = a_1\ \mathsf{then}\ \mathbf{0} \\ &\mathsf{else\ if}\ x = a_2\ \mathsf{then}\ \mathbf{0} \\ &\mathsf{else\ if}\ \ldots \\ &\vdots \\ &\mathsf{else}\ \mathsf{fail}().\,\overline{\mathsf{ext}}\langle \vec{X}; \vec{a}_\Gamma, x \rangle) \end{aligned}$$

where $\vec{a}_n$ are all of the a in $\mathsf{dom}(\Gamma)$.

Note, that, in this case we use $\mathsf{mgu}(V, Y)$ as $\Gamma, b : Y \vdash b : Y$. Therefore,

$$\Gamma' = (\vec{X}, Y, \Gamma, b : Y)[\mathsf{mgu}(V, Y)] = \vec{X}, \Gamma, b : V$$

Again, we need to check that R is well-typed with respect to $\Gamma, \mathsf{ext} : \updownarrow[\vec{X}; (\Gamma')], \mathsf{fail} : \updownarrow[]$. This time we cannot make any use of unification for the output on ext as we are using the else branch of a conditional. Note though that

$$\vec{X}, \Gamma, x : V \vdash \vec{a}_\Gamma, x : (\Gamma')$$

because $(\Gamma') = (\Gamma), V$.

Clearly, $P \mid R[\sigma]$ reduces as expected because we know that b is fresh to Γ. For any Q, ρ such that $Q \mid R[\rho] \Longrightarrow Q'' \not\downarrow_{\mathsf{fail}}$, we know that there must have been an output from Q along channel c of a name b', say. This name must be different to every a_i so it must be fresh to Γ and bound in Q. By alpha-conversion in Q, we can therefore choose b' to be b to obtain the required properties of Q.

Proposition 15 (Unifying Extrusion). *If*

$$\Gamma, \mathsf{ext} : \updownarrow[\vec{X}; (\Gamma')] \vDash [\sigma]\nu(\vec{a} : \vec{T})(P \mid \overline{\mathsf{ext}}\langle \vec{U}; \vec{a}_{\Gamma'} \rangle) \cong^p [\rho]\nu(\vec{a} : \vec{T}')(Q \mid \overline{\mathsf{ext}}\langle \vec{W}; \vec{a}_{\Gamma'} \rangle)$$

with $\vec{a} \subseteq \vec{a}_{\Gamma'}$ *and* $\mathsf{ext} \notin \mathsf{fn}(P, Q)$ *then*

$$\Gamma' \vDash [\vec{U}/\vec{X}, \sigma]P \cong^p [\vec{W}/\vec{X}, \rho]Q.$$

whenever these are configurations.

Proof. The proof of this is straightforward and similar to that found in [15].