

US

University of Sussex

Computer Science Report

**Proof methodologies for behavioural
equivalence in Distributed π -calculus**

Alberto Ciaffaglione
Matthew Hennessy
Julian Rathke

Report 2005:03

April 2005

Department of Informatics
University of Sussex
Brighton BN1 9QH

ISSN 1350–3170

Proof methodologies for behavioural equivalence in Distributed π -calculus

Alberto Ciaffaglione, Matthew Hennessy and Julian Rathke

Abstract. We focus on techniques for proving behavioural equivalence between systems in D

categories, for systems, and agents. A typical system takes the form

$$(\text{new } e : E)(l[[P]] \text{ j } k[[Q]]) \text{ j } l[[R]]$$

This represents a system with two sites, l and k , with the agents P and R running at the former and Q at the latter; moreover P and Q , although executing at different sites, share some private information, e , of type E . The syntax for agents, or processes, is an extension of that of the π -calculus [SW01]. There are input and output on local channels, parallelism, matching of values, iteration, and a migration construct. For example, in the system

$$l[[P \text{ j } \text{goto } k:Q]] \text{ j } k[[R]]$$

the process Q can migrate from l to k , leading to the resulting system

$$l[[P]] \text{ j } k[[Q \text{ j } R]]$$

Finally, processes have the ability to create new instances of names (channels, `newc`, and sites, `newloc`); their declaration types dictate the use to which these will be put. Finally, `crer(l)` creates a new site l .

The values, V , communicated along channels consist of tuples of *simple values*, v , of type τ . The values v consist of tuples of *simple values*, v , of type τ .

Base Types:	$\mathbf{base} ::= \mathbf{int} \mid \mathbf{bool} \mid \mathbf{unit} \mid > \mid j \quad \dots$
Value Types:	$A ::= \mathbf{base} \mid C \mid C_{@loc} \mid K$
Local Channel types:	$C ::= \mathbf{rhTi} \mid \mathbf{whTi} \mid \mathbf{rwhTi}$
Location Types:	$K ::= \mathbf{loc}[c_1 : C_1; \dots; c_n : C_n]; n \geq 0$ (provided $c_i = c_j$ implies $i = j$)
Transmission Types:	$T ::= (A_1; \dots; A_n); n \geq 0$

Figure 2. Types for Dpi - informal

This generates a new reply channel, r , at the declaration type R , and awaits input on this channel to be printed. Concurrently, it sends to the server site an agent, which sends to the request channel the tuple consisting of some value, v_c , hopefully an integer, and the reply address, $r@c$. Then, running the combined system

$$S \mid C \tag{1}$$

should result in a boolean being printed at the client's site, the value of which is determined by the primality of v_c .

2.2 Typing

Dpi is a capability based language, in the sense that the behaviour of processes depends on the capabilities the various entities have received in their environment. Formally, these capabilities are represented as types, and the various categories of types we use are given in Figure 2. Apart from the standard base types, and the special *top* type $>$, the main ones are

local channel types: these are ranged over by C and can take the form \mathbf{rwhTi} , giving the ability to both read and write values of type T , or the restricted supertypes \mathbf{rhTi} and \mathbf{whTi} ;

non-local channel types: these take the form $C_{@loc}$, and a value of this type is a structured value, $c@l$;

location types: these take the form $\mathbf{loc}[c_1 : C_1; \dots; c_n : C_n]$; receiving a value l of this type gives access to the channels, or resources, c_i at type C_i , for $1 \leq i \leq n$.

In this overview we omit one further category of types, that of *registered names*, as they play no part in the current paper; as usual, the reader is referred to [HMR04] for an explanation of their role in ensuring consistency between the

The rules for typing agents are more or less borrowed from the π -calculus [PS00], with the addition of a rule for migration. For example, (local) input and output are handled by the rules

$$\begin{array}{c}
 \text{(ty-out)} \\
 \frac{\begin{array}{c} \text{\textbackslash}_w V : T \\ \text{\textbackslash}_w P \\ \text{\textbackslash}_w u : whTi \end{array}}{\text{\textbackslash}_w u!hVi P}
 \end{array}
 \qquad
 \text{(ty-in)}$$

that is required of Γ in order to type both the server and the client is to let \mathbf{S} , the type associated with the request channel, to be $\text{rw} \int \mathbf{int}; \text{wh} \mathbf{bool} \mathbf{i} @ \text{loc } i$.

There is an interesting point to be made here. The client generates the reply channel r with both read and write capabilities; only the latter is sent to the server, via req , and the former is retained for internal use. This use of restricted capabilities provides a certain level of protection to the client, as it knows that the reply from the server can not be usurped by any other client.

2.3 Behaviour

The behaviour of a system, that is the ability of its agents to interact with other agents, depends on the knowledge these agents have of each others capabilities. In the example just discussed we have seen the client generating a reply channel with two capabilities, but only making one of these externally available; indeed, the proper functioning of the client/server interaction depends on such decisions.

Definition 2.1 (Configurations). A *configuration* consists of a pair $\Gamma \vdash M$, where

Γ is a type environment which associates some type to every free name in M
 there is a type environment Γ_1 such that $\Gamma_1 \vdash M$ and $\Gamma_1 \leq \Gamma$

This latter requirement means that if Γ_1 can assign a type T_1 to a name n , then Γ can assign a type T such that $T \leq T_1$. Again, viewing types as sets of capabilities, this means that T_1 , representing the knowledge of the external user, is a subset of T , the actual set of capabilities used to type the system M .

So we define the behaviour in terms of actions over configurations; these are of the form

$$\Gamma \vdash M \xrightarrow{\alpha} \Gamma^0 \vdash M^0 \quad (2)$$

where the label α can take any of the following forms

- τ : an internal action, requiring no participation by the user;
- $(\tilde{e} : \tilde{E})k:a?V$: the input of value V along the channel a , located at the site k . The bound names in (\tilde{e}) are freshly generated by the user;
- $(\tilde{e} : \tilde{E})k:a!V$: the output of value V along the channel a , located at the site k . The bound names in (\tilde{e}) are freshly generated by the environment.

The rules for defining these actions are given in Figure 3 and Figure 4, a slightly different but equivalent formulation to that given in [HMR04]. The guiding principle for (2) to happen, is that M must be able to perform the action α , and the user must have, in Γ , the capability to participate in the action. The rules use some new notation for looking up the types associated with channels in environments: the partial functions $\Gamma^r(k; a)$ and $\Gamma^w(k; a)$ return the read, respectively write, type associated with the channel a at the location k in Γ (of course these

(m-in)

$$\frac{\Gamma^w(k; a) \# \quad \Gamma^w_k V : \Gamma^w(k; a)}{\Gamma B k[[a?(X) R]] \quad \Gamma^k a^V \quad \Gamma B k[[Rf^V \rightarrow X]]}$$

(m-weak)

$$\frac{\Gamma ; he : Ei B M \quad \Gamma^{\tilde{d}:\tilde{D}} k : a^V \quad \Gamma^0 B M^0}{\Gamma B M \quad \Gamma^{\tilde{d}:\tilde{D}} k : a^V \quad \Gamma^0 B M^0} \quad \text{bn}(e) \notin \Gamma$$

(m-out)

$$\frac{\Gamma^r(k; a) \#}{\Gamma B k[[a!hVi P]] \quad \Gamma^k a^V \quad \Gamma ; hV : \Gamma^r(k; a) \quad \Gamma B k[[P]]}$$

(m-open)

$$\Gamma ; he : >i B M \quad \Gamma^{\tilde{d}:\tilde{D}} k : a^V \quad \Gamma^0 B M$$

(m-weak), allows us to derive the following action from the server S

$$B S \stackrel{(r@c:R)}{!} ; r@c : R B s[[\text{goto } c:r!\text{hisprime}(v_c)\text{i stop}]] \quad (3)$$

where $!$ is the input action $s:\text{req?}(v_c; r@c)$, because

$$; r@c : R B S \stackrel{!}{!} ; r@c : R B s[[\text{goto } c:r!\text{hisprime}(v_c)\text{i stop}]]$$

Similarly, (m-OUT) requires l to have a *read* capability on a at k , in order for $k[[a!hVi P]]$ to be able to perform the obvious output; note that here the current knowledge of the user, l , is augmented by whatever new knowledge which can be gleaned from the received value V . Intuitively, $hV : T\text{i}@k$ decomposes the value V , relative to the type T , from the standpoint of k ; this last only comes into play when V contains instances of local channels, which are then interpreted as channels at k . But the important point in (m-OUT) is that the type at which V is added to l is $l^r(k; a)$, the reception type that the user currently has on a at k . Thus (m-open) allows us to deduce

$$B (\text{new } r@c : R) s[[\text{req!}hv_c; r@c\text{i stop}]] \stackrel{(r@c:R)}{!} ; r@c : R_w B s[[\text{stop}]] \quad (4)$$

where $!$ is the output action $s:\text{req!}hv_c; r@c\text{i}$, because with (m-OUT) we can derive

$$; r@c : > B s[[\text{req!}hv_c; r@c\text{i stop}]] \stackrel{!}{!} ; r@c : R_w B s[[\text{stop}]]$$

The use of $>$ is simply to ensure that we have a valid configuration; but note that the user has gained only the restricted capability R_w on the new channel r , rather than the more liberal declaration capability R , because the former is the type at which the user can receive values along req .

The rules for the internal actions are given in Figure 4, and most are straightforward. We have labelled some as -actions , which will be useful in the next section; but for the moment these labels can be ignored. The only interesting rule is (m-COMM), which formalises *communication*. Note that, in the hypotheses of both variations, arbitrary user environments, l_1 and l_2 , are allowed. This may be surprising at first, but intuitively -actions should be independent of all external knowledge. For example, we can use (3) and (4) above to derive

$$\begin{aligned} l B S j (\text{new } r@c : R) s[[\text{req!}hv_c; r@c\text{i stop}]] \stackrel{!}{!} \\ l B (\text{new } r@c : R) s[[\text{goto } c:r!\text{hisprime}(v_c)\text{i stop}]] j s[[\text{stop}]] \end{aligned}$$

for an arbitrary l .

We now have a labelled transition system in which the states are configurations, and we can apply the standard definition of (weak) bisimulation.

$$\frac{\text{(m-comm)} \quad \begin{array}{c} \text{I}_1 \text{ B } M \quad (\tilde{z}:\tilde{\mathbb{E}})k:a!V \quad \text{I}_1^0 \text{ B } M^0 \quad \text{I}_2 \text{ B } N \quad (\tilde{z}:\tilde{\mathbb{E}})k:a!V \quad \text{I}_2^0 \text{ B } N^0 \end{array}}{\text{I B } M \text{ j } N \quad ! \quad \text{I B } (\text{new } e : \mathbb{E})(M^0 \text{ j } N^0)}$$

$$\frac{\text{(m-comm)} \quad \begin{array}{c} \text{I}_1 \text{ B } M \quad (\tilde{z}:\tilde{\mathbb{E}})k:a!V \quad \text{I}_1^0 \text{ B } M^0 \quad \text{I}_2 \text{ B } N \quad (\tilde{z}:\tilde{\mathbb{E}})k:a!V \quad \text{I}_2^0 \text{ B } N^0 \end{array}}{\text{I B } M \text{ j } N \quad ! \quad \text{I B } (\text{new } e : \mathbb{E})(M^0 \text{ j } N^0)}$$

$$\frac{\text{(m-move)}}{\text{I B } k[\text{goto } l:P] \quad ! \quad \text{I B } l[P]}$$

$$\frac{\text{(m-c:create)}}{\text{I B } k[(\text{new } c : \mathbb{C}) P] \quad ! \quad \text{I B } (\text{new } c@k : \mathbb{C}) k[P]}$$

$$\frac{\text{(m-l:create)}}{\text{I B } k[(\text{new } loc l : \mathbb{L}) P] \quad ! \quad \text{I B } (\text{new } l : \mathbb{L}) k[P]}$$

$$\frac{\text{(m-eq)}}{\text{I B } k[\text{if } v = v \text{ then } P \text{ else } Q] \quad ! \quad \text{I B } k[P]}$$

$$\frac{\text{(m-neq)}}{\text{I B } k[\text{if } v_1 = v_2 \text{ then } P \text{ else } Q] \quad ! \quad \text{I B } k[Q] \quad (v_1 \neq v_2)}$$

$$\frac{\text{(m-split)}}{\text{I B } k[P \text{ j } Q] \quad ! \quad \text{I B } k[P] \text{ j } k[Q]}$$

$$\frac{\text{(m-unwind)}}{\text{I B } k[P] \quad ! \quad \text{I B } k[P \text{ j } P]}$$

Figure 4. Internal actions-in-context for Dpi

Definition 2.2 (Bisimulations). We say a binary relation over configurations is a *bisimulation* if both it, and its inverse, satisfy the following transfer property

$$\begin{array}{ccc}
 (I_M \text{ B } M) \text{ R } (I_N \text{ B } N) & & (I_M \text{ B } M) \text{ R } (I_N \text{ B } N) \\
 \downarrow \text{ ? } & \text{implies} & \downarrow \text{ ? } \\
 (I_{M^0} \text{ B } M^0) & & (I_{M^0} \text{ B } M^0) \text{ R } (I_{N^0} \text{ B } N^0)
 \end{array}$$

Here we use standard notation, see [MPW92], with \Rightarrow representing \rightarrow , $\hat{\Rightarrow}$ meaning \Rightarrow , if \rightarrow is \rightarrow , and \Rightarrow otherwise. This allows a single internal move to be matched by zero or more internal moves.

We let bis denote the largest bisimulation between configurations.

Rather than writing $(I \text{ B } M) \text{ bis } (I \text{ B } N)$; we use the more suggestive notation

$$I \text{ j= } M \text{ bis } N$$

This can be viewed as a relation between systems, parameterised over type environments which represent user’s knowledge of the systems’ capabilities.

It is this bisimilarity bis which is the object of our study: we aim to show that, despite the complexity of its definition, tractable proof techniques can be developed for it.

Finally, we should remark this is not an arbitrarily chosen version of bisimulation equivalence. In [HMR04] its definition is justified in detail: it is shown to

be the largest bisimulation that is preserved by internal moves.

(s-extr) $(\text{new } e : E)(M \text{ j } N) \quad M \text{ j } (\text{new } e : E)N \quad \text{if } \text{bn}(e) \notin \text{fn}(M)$

(s-com) $M \text{ j } N$

-actions, include

$$\begin{aligned}
 k[[P \text{ j } Q]] & \text{ bis } k[[P]] \text{ j } k[[Q]] \\
 k[[\text{goto } l:P]] & \text{ bis } l[[P]] \\
 k[[\text{new } c : C \text{ } P]] & \text{ bis } (\text{new } c@k : C) k[[P]]
 \end{aligned}$$

But these τ -labelled internal actions also provide us with a very powerful method for approximating bisimulations, in the spirit of [JR04].

Definition 3.5 (Bisimulations up-to- τ). A binary relation between configurations is said to be a *bisimulation up-to- τ* if it satisfies the following transfer properties

$$\begin{array}{ccc}
 (I_M \text{ B } M) \text{ R } (I_N \text{ B } N) & & (I_M \text{ B } M) \text{ R } (I_N \text{ B } N) \\
 \downarrow \text{?} & \text{implies} & \\
 (I_{M^0} \text{ B } M^0) & & (I_{M^0} \text{ B } M^0) \text{ A}_l
 \end{array}$$

Proof: We leave to the reader to check that the relation $(\sim_{bis} \ R \ \sim_{bis})$ is a bisimulation over configurations. The key properties for establishing this are the two inclusions $\sim_{bis} \subseteq \sim_{bis}$ (Proposition 3.4) and $\sim_{bis} \subseteq \sim_{bis}$ (Proposition 3.2), Lemma 3.3 and transitivity, in Definition 3.5, of both A_l (due to Lemma 3.6) and A_r . The result then follows, since $(\sim_{bis} \ R \ \sim_{bis})$ trivially contains R .

4 Crossing a firewall

Let us consider the *firewall* example, first proposed in [CG98] and studied at length in [GC99, LS00, MN03] within versions of Mobile Ambients. Intuitively, a firewall is a domain to which access is restricted: only agents which are permitted, in some sense, by the firewall are allowed in. A simple example takes the form

$$F \ (\text{new } f : F) f$$

Then one might expect to be able to derive

$$\models F \text{ j } A \text{ j } \text{bis} \text{ (new } f : F \text{)(} f \llbracket P \text{ j goto } a \text{:tell!hf i j } Q \rrbracket \text{) j } a \llbracket R \rrbracket \quad (7)$$

But this happens not to be true, because of the implicit assumption that the information channel `tell` in `a` can only be accessed by partners in the entry protocol, `f` and `a`. But, in order for (6) to be true, we must have $\vdash_a \text{tell} : \text{rhF}_r i$; and this allows other agents in the environment access to `tell`. For example, consider

$$\text{Rogue} \text{ (} b \llbracket \text{goto } a \text{:tell!hbi} \rrbracket$$

and suppose that the only type inference from \vdash involving `b` is $\vdash b : \text{loc}$; so \vdash is not aware of any resources at `b`. Nevertheless \vdash `Rogue`, and therefore *Contextuality* (Theorem 3.1) applied to (7) would give

$$\models F \text{ j } A \text{ j } \text{Rogue} \text{ j } \text{bis} \text{ (new } f : F \text{)(} f \llbracket P \text{ j goto } a \text{:tell!hf i j } Q \rrbracket \text{) j } a \llbracket R \rrbracket \text{ j } \text{Rogue}$$

But this is obviously not the case, as the left-hand system can reduce via a series of \rightarrow -steps (representing the interaction between `A` and `Rogue`) to the state

$$B \text{ F j } a \llbracket R \rrbracket \text{ j } b \llbracket Q \rrbracket$$

Under reasonable assumptions about the code `Q`, the right-hand system has no corresponding reduction to a similar state. On the left-hand side the code `Q`, now located at `b`, can not run, while on the right-hand side, no matter what \rightarrow -steps are made, `Q` will be able to execute at `f`.

Thus (7) can not be true.

However, our framework allows us to amend the correctness statement (7) above, taking into account the implicit assumption about the information channel `tell`. The essential point is that the protocol works provided that *only the firewall can write on tell*. This can be formalised by proving the equivalence between the two systems relative to a restricted environment, one which does not allow write access to `tell`.

First some notation. Let us write $\vdash_k^{max} V : T$ to mean

$$\vdash_k V : T$$

$$\vdash_k V : T^0 \text{ implies } T <: T^0$$

In other words, T is the *largest* type which can be assigned to `V`. Now suppose Γ is a type environment which satisfies

$$(i) \Gamma \vdash_a^{max} \text{tell} : \text{rhFi}$$

$$(ii) \Gamma \vdash a \llbracket R \rrbracket$$

$$(iii) \Gamma \vdash (\text{new } f : F) f \llbracket P \rrbracket$$

The import of the first requirement, which is the most important, is that systems in the computational context can not write on tell. The other requirements, which are mainly for convenience, ensure that the residual behaviour at a and f is well-behaved, although a side-effect is that they also can not write on tell. Under these assumptions, we prove

$$\vdash \text{F} \text{ j } A \text{ }_{bis} \text{ (new } f : \text{F})(f \llbracket P \text{ j goto } a \text{:tell!hf i j } Q \rrbracket) \text{ j } a \llbracket R \rrbracket \quad (8)$$

First note that (up-to structural equivalence)

$$\vdash \text{B} \text{ F} \text{ j } A \text{ } ! \text{ F} \text{ j } A_t \text{ j } a \llbracket R \rrbracket$$

via (m-split) and (m-ctxt), where A_t is a shorthand for $a \llbracket \text{tell?}(x) \text{ goto } x \text{:} Q \rrbracket$. So, by Propositions 3.2 and 3.4, it is su

M has the form $F_g \mid A_t \mid _n (a[\text{tell!hfi}])^n$

N has the form $F_g \mid f[Q] \mid _n (a[\text{tell!hfi}])^n$

where $_n (a[\text{tell!hfi}])^n$, for some $n \geq 0$, means n copies of $a[\text{tell!hfi}]$ running in parallel.

Proposition 4.1. *The parameterised relation R defined above is a bisimulation up-to- .*

Proof: Suppose $J \vDash MRN$. Let us consider all possible actions from $J \in BM$. In fact, it is sufficient to consider the case (b) above, when J and M and N are of the prescribed form. The actions fall into one of three categories (for convenience we shorten $_n (a[\text{tell!hfi}])^n$ with $_n$).

Note that the firewall F allows, in principle, multiple entries of agents from a . So, for example, if R

The first requirement establishes that *the computational context can not read on req*, while the following points ensure that the residual behaviour at the server and the clients is well-behaved, with the side-effect that neither S^0 nor C_i^0 can read on req.

First, let us show that one client interacts correctly with the server

$$\vdash S \text{ j } C_1 \quad \text{bis} \quad S \text{ j } c_1 \llbracket (\text{new } r : \mathbf{R}) \text{ } r! \text{hisprime}(v_1) \text{ i j } C_1^0 \rrbracket \quad (10)$$

Note that (up-to-structural equivalence)

$$\vdash B S \text{ j } C_1 \quad ! \quad (\text{new } r@c_1 : \mathbf{R}) S_r \text{ j } s \llbracket S^0 \rrbracket \text{ j } s \llbracket \text{req}!hv_1; r@c_1 \text{ i} \rrbracket \text{ j } c_1 \llbracket C_1^0 \rrbracket$$

where we use S_r as a shorthand for $s \llbracket \text{req}?(x; y@z) \text{ goto } z:y! \text{hisprime}(x) \text{ i} \rrbracket$, and

$$\vdash B S \text{ j } c_1 \llbracket (\text{new } r : \mathbf{R}) \text{ } r! \text{hisprime}(v_1) \text{ i j } C_1^0 \rrbracket \quad ! \quad (\text{new } r@c_1 : \mathbf{R}) S_r \text{ j } s \llbracket S^0 \rrbracket \text{ j } c_1 \llbracket r! \text{hisprime}(v_1) \text{ i} \rrbracket \text{ j } c_1 \llbracket C_1^0 \rrbracket$$

By Propositions 3.2, 3.4,

Proposition 5.1. *The parameterised relation R defined above is a bisimulation up-to- .*

Proof: Suppose $J \models M R N$. The actions from $J \rightarrow B M$ in the case (b) above fall into one of three categories.

First S_r is responsible

$$\Gamma_r \rightarrow B M \quad ! \quad s \llbracket \text{req?}(x; y@z) \text{goto } z:y!\text{hisprime}(x) \rrbracket j R^0 \rrbracket j s \llbracket \text{req!}hv_1; r@c_1 \rrbracket j \quad n$$

where R^0 is a shorthand for $\text{req?}(x; y@z) \text{goto } z:y!\text{hisprime}(x) \rrbracket j$. But

$$\Gamma_r \rightarrow B s \llbracket \text{req?}(x; y@z) \text{goto } z:y!\text{hisprime}(x) \rrbracket j R^0 \rrbracket j s \llbracket \text{req!}hv_1; r@c_1 \rrbracket j \quad n \quad ! \\ S_r j \quad 1 j s \llbracket \text{req!}hv_1; r@c_1 \rrbracket j \quad n$$

and this can be matched by

$$\Gamma_r \rightarrow B N \quad ! \quad S_r j \quad 1 j c_1 \llbracket r!\text{hisprime}(v_1) \rrbracket j \quad n$$

because both configurations belong to R , clause (b), up-to structural equivalence.

The third component, $\quad n (s \llbracket \text{req?}(x; y@z) \rrbracket j \text{goto } z:y!\text{hisprime}(x) \rrbracket j)^n$, is responsible for the action, which is either $s:\text{req? } i_j; d_j@k_j$ or $(e:E)s:\text{req? } i_j; d_j@k_j$. These actions correspond to the delivery of (new) data by the environment (from which the system is allowed to learn infinitely new names), and are followed by the action (m-move). However, it is easy to see that $\Gamma_r \rightarrow B N$ can

This completes our proof of (10), that one client can interact correctly with the server. Contextual reasoning can now be employed to generalise this result to an arbitrary number of clients. For example, let us show

$$\vdash S \mid C_1 \mid C_2 \text{ bis } S \mid_{i \in \{1,2\}} c_i \llbracket (\text{new } r : R) \text{ } r!\text{hisprime}(v_i) \mid C_i^0 \rrbracket \quad (11)$$

Because of $\vdash C_2$ (requirement (iii) above), *Contextuality* applied to (10) gives

$$\vdash S \mid C_1 \mid C_2 \text{ bis } S \mid c_1 \llbracket (\text{new } r : R) \text{ } r!\text{hisprime}(v_1) \mid C_1^0 \rrbracket \mid C_2 \quad (12)$$

On the other hand, repeating the analysis of C_1 on C_2 , we obtain

$$\vdash S \mid C_2 \text{ bis } S \mid c_2 \llbracket (\text{new } r : R) \text{ } r!\text{hisprime}(v_2) \mid C_2^0 \rrbracket$$

But $\vdash C_1$ (again (iii)) also implies $\vdash c_1 \llbracket (\text{new } r : R) \text{ } r!\text{hisprime}(v_1) \mid C_1^0 \rrbracket$, and therefore, by *Contextuality*

$$\vdash S \mid C_2 \mid c_1 \llbracket (\text{new } r : R) \text{ } r!\text{hisprime}(v_1) \mid C_1^0 \rrbracket \text{ bis } S \mid_{i \in \{1,2\}} c_i \llbracket (\text{new } r : R) \text{ } r!\text{hisprime}(v_i) \mid C_i^0 \rrbracket$$

So we conclude (11) from (12), Proposition 3.2, and transitivity of *bis*.

It is then a simple matter to extend this reasoning, using induction, to show that an arbitrary number of clients can be handled

$$\vdash S \mid_{i \in \{1, \dots, n\}} C_i \text{ bis } S \mid_{i \in \{1, \dots, n\}} c_i \llbracket (\text{new } r : R) \text{ } r!\text{hisprime}(v_i) \mid C_i^0 \rrbracket$$

This we leave to the reader.

As a further example of the modularity of our proofs, let us consider a particular instantiation of the residual processes, S^0 and C_i^0 : we set S^0 to `stop` and C_i^0 to `r?(x) printi!hx`, where `printi` are local channels. For convenience we restrict attention to two clients, and let us assume that they send the integer values $v_1 = 4$ and $v_2 = 3$, respectively, to the server. So we have

$$\begin{aligned} S^{00} & (\text{ } s \llbracket \text{req?}(x; y@z) \text{ goto } z; y!\text{hisprime}(x) \mid \rrbracket \\ C_1^{00} & (\text{ } c_1 \llbracket (\text{new } r : R) \text{ goto } s; \text{req!}h4; r@c_1 \mid r?(x) \text{ prC} \rrbracket \end{aligned}$$

ther, to the tasks

$$\begin{array}{l} I \models S^{00} j c_1 \llbracket (\text{new } c \ r : R) \ r! \text{hisprime}(4) \rrbracket i \ j \ r?(x) \ \text{print}_1! \langle x \rangle \rrbracket \quad \text{bis} \\ S^{00} j c_1 \llbracket \text{print}_1! \langle \text{false} \rangle \rrbracket \end{array}$$

$$\begin{array}{l} I \models S^{00} j c_2 \llbracket (\text{new } c \ r : R) \ r! \text{hisprime}(3) \rrbracket i \ j \ r?(x) \ \text{print}_2! \langle x \rangle \rrbracket \quad \text{bis} \\ S^{00} j c_2 \llbracket \text{print}_2! \langle \text{true} \rangle \rrbracket \end{array}$$

Note that *Contextuality* does not allow us to eliminate S^{00} from these judgements, since $I \not\models S^{00}$ is not true. Nevertheless, it is a simple matter to construct a witnessing bisimulation to demonstrate directly these two equivalences, as the reader can check.

6 Metaservers

In this section we describe a *memory service* by involving the `newloc` operator of `Dpi`, which allows the creation of new instances of sites. A (meta)server contains a resource `setup`, where requests are received, and installs the service at a new site, thus providing personalised treatment to its clients.

A first version of the server receives a return address, generates a new located memory cell, and installs some code there, meanwhile delivering the new

An alternative, slightly different version of the server leaves to the clients the responsibility to create the memory cells, just installing the servicing code at the preferred site

$$S^0 \text{ (} s^0 \llbracket \text{setup}^0(x; y@z) \text{ goto } x:\text{Mem} \text{ j goto } z:y! \rrbracket \text{)}$$

Correspondingly, clients generate an acknowledgement channel and a new location, send a request to the server, and await the server to acknowledge the service has been installed

$$C_i^0 \text{ (} c_i \llbracket (\text{new } t : T) (\text{new } loc\ m_i : M) \text{ goto } s^0:\text{setup}^0!h_{m_i} \text{ } t@c_i \text{ i j } t?P_i(m_i) \rrbracket \text{)}$$

where $T = \text{rw}h\text{uniti}$.

We want now to relate the two different approaches, therefore connecting the behaviour of the two following systems, relative to a typing environment Γ

$$\Gamma \models S \text{ j } C_1 \text{ j } C_2 \quad (13)$$

$$\Gamma \models S^0 \text{ j } C_1^0 \text{ j } C_2^0 \quad (14)$$

Our goal is to establish that, from the point of view of the clients, under certain hypotheses the two kinds of servers S and S^0 lead to equivalent behaviour. This means finding a suitable type environment Γ such that

$$\Gamma \models S \text{ j } C_1 \text{ j } C_2 \quad \text{bis} \quad S^0 \text{ j } C_1^0 \text{ j } C_2^0 \quad (15)$$

It is immediate to notice that the correctness of this protocol requires that *the computational context should have neither write nor read access to the setup and setup⁰ channels*. Thus, the equivalence can be proved relative to a restricted environment Γ , satisfying

$$\Gamma \stackrel{\text{max}}{s} \text{setup} : > \quad \Gamma \stackrel{\text{max}}{s^0} \text{setup}^0 : >$$

Now, the internal actions can be used to deduce a derivation from (13) and (14) to the systems

$$\Gamma \models S \text{ j }_{i2f1;2g} (\text{new } m_i : M)(m_i \llbracket \text{Mem} \rrbracket \text{ j } c_i \llbracket$$

- (b) or $(\text{new } r_1@c_1 : R; r_2@c_2 : R; m_1 : M) S \mid_n S_{12} \mid C_{?2} \mid M_1 \mid C_{!1} \mid C_{?1}$
- (c) or $(\text{new } r_1@c_1 : R; r_2@c_2 : R; m_2 : M) S \mid_n S_{!1} \mid C_{?1} \mid M_2 \mid C_{!2} \mid C_{?2}$
- (d) or $(\text{new } r_2@c_2 : R; m_1 : M) S \mid_n S_{!2} \mid C_{?2} \mid M_1 \mid C_{P_1}$
- (e) or $(\text{new } r_1@c_1 : R; m_2 : M) S \mid_n S$

with the Dpi calculus [HR02b]. In order to cope with bisimulation equivalence in Dpi [HMR04], it is natural to look for bisimulations up-to in the spirit of [SM92]. More precisely, we have introduced in our work *bisimulations up-to -reductions*, which have been inspired by a similar approach to concurrent ML [JR04]. This technique actually relieves the burden of exhibiting witness bisimulations, and its feasibility has been proved to be successful, combined mainly with *Contextuality*

- [MPW92] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes (I and II). *Information and Computation*, 100(1,2), 1992.
- [PS00] Benjamin C. Pierce and Davide Sangiorgi. Behavioral equivalence in the polymorphic π -calculus. *Journal of ACM* 47(3), 2000.
- [SM92] Davide Sangiorgi and Robin Milner. The problem of “weak bisimulation up to”. In Proc. of *CONCUR, Lecture Notes in Computer Science* 630, Springer, 1992.
- [SW01] Davide Sangiorgi and David Walker. *The π -calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.
- [US01] Asis Unyapoth and Peter Sewell. Nomadic π -calculus: correct communication infrastructure for mobile computation. In Proc. of *POPL*, 2001.