University of Sussex

Computer Science Report

Proof methodologies for behavioural equivalence in Distributed PICALCULUS

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 PICALCULUS

ALBERTO CIAFFAGLIONE, MATTHEW HENNESSY and JULIAN RATHKE

ABSTRACT. We focus on techniques for proving behavioural equivalence between systems in DPI, a distributed version of the PICALCULUS in which processes may migrate between dynamically created locations, and where resource access policies are implemented by means of capability types.

We devise a tractable collection of auxiliary proof methods, relying mainly on the use of *bisimulations up-to β-reductions*, which considerably relieve the burden of exhibiting witness bisimulations. Using such methods we model simple distributed protocols, such as crossing a firewall, the interaction between a server and its clients, metaservers installing memory services, and address their correctness in a relatively simple manner.

1 Introduction

Bisimulations [Mil89], and the related bisimulation equivalence, have been proved to be of central importance in the elaboration of semantic theories of processes, and in developing verification techniques for them. The purpose of this work is to demonstrate that may also be employed for the verification of distributed systems, even when the correctness depends on access control policies.

We focus on an abstract system description language called DPI [HR02b], an extension of the well-known PICALCULUS [MPW92, SW01]. In this language a system consists of a collection of *processes*, or *agents*, distributed among different *sites*, where they can use *local resources*; these resources are modelled using *local* versions of PICALCULUS *communication channels*. Agents may migrate from site to site, generate new local resources, or indeed new sites.

Following ideas originally formulated in [PS00], DPI can be endowed with a system of *capability types*, with which access policies to both resources and sites can be expressed. Since the behaviour of systems is dependent on the access policy in force, a new theory of semantic equivalence is required to take this dependency into account. This was developed in [HR02a, HMR04], where the equivalence is expressed in the form of triples

$$\mathcal{I} \models M \approx_{bis} N$$

Intuitively this means that the systems M and N exhibit the same behaviour, from the point of view of a user constrained by the access policy $\mathcal{I}$; formally, $\mathcal{I}$ is simply a type environment, giving, for each resource and location, the capabilities which may be exercised by the user.

In this paper we show that this relativised notion of system behaviour can be effectively employed to demonstrate the correctness of access protocols for

M, N ::=	*Systems*
$l[\![P]\!]$	Located agents
$M \mid N$	Composition
$(\mathsf{new}\, e : \mathsf{E})\, M$	Name Scoping
$\mathbf{0}$	Termination

R, U ::=	*Processes, or Agents*
$u!\langle V\rangle.R$	Output
$u?(X).R$	Input
$\mathsf{goto}\, v.R$	Migration
$(\mathsf{newc}\, c : \mathsf{C})\ R$	Local channel creation
$(\mathsf{newloc}\, k : \mathsf{K})\ R$	Location creation
$\mathsf{if}\ v_1 = v_2\ \mathsf{then}\ R\ \mathsf{else}\ U$	Matching
$R \mid U$	Parallelism
$*R$	Iteration
stop	Termination

FIGURE 1. Syntax for DPI

distributed systems. All the examples considered are very simple; nevertheless, we feel that they at least demonstrate the feasibility of this approach to system verification.

In the next section we review the language DPI, its type system, and the relativised notion of bisimulation equivalence. This is followed by an exposition of some useful proof techniques, relying mainly on the use of bisimulations up-to in the spirit of [SM92], which actually alleviate the burden of exhibiting witness bisimulations. This is then followed by three sections, each considering a particular verification example. The final section is about related and future work.

2 DPI: a synopsis

In this section we recall the essential features of the language DPI; readers are referred to [HMR04, HR02b] for a more detailed description.

2.1 Syntax

The syntax of the language is given in Figure 1, and presupposes a set of *identifiers*; these consist either of names n, m, a, b, l, k, taken from some predefined set NAMES, or variables x, y, z, taken from a set VARS. There are two syntactic

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

$$(\mathsf{new}\, e : \mathsf{E})(l[\![P]\!] \mid k[\![Q]\!]) \mid l[\![R]\!]$$

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 PICALCULUS [SW01]. There are input and output on local channels, parallelism, matching of values, iteration, and a migration construct. For example, in the system

$$l[\![P \mid \mathsf{goto}\, k.Q]\!] \;\mid\; k[\![R]\!]$$

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

$$l[\![P]\!] \;\mid\; k[\![Q \mid R]\!]$$

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.

The values, V, communicated along channels consist of tuples of *simple values*, v. These, in turn, may be *identifiers*, u, or structured values, of the form $u_{1@}u_2$; the latter are used to represent channels which are not local to the site at which the communication takes place. In turn, the input construct $u?(X)\,R$ uses patterns, X, to deconstruct incoming values; these may be taken to be values constructed from variables, in which each variable has at most one occurrence.

For example, consider the following definition of a server

$$S \Leftarrow s[\![\mathsf{req}?(x, y_{@}z)\mathsf{goto}\, z.y!\langle \mathit{isprime}(x)\rangle\, \mathsf{stop}]\!]$$

which expects to receive a structured value of the form $(i, c_{@}l)$. This is a pair, consisting of an integer i, and a return address $c_{@}l$, that is the name of a reply channel, c, together with the location of that channel, l. The server then executes the procedure $\mathit{isprime}(-)$ on the incoming value, i, sends a process to the return site, and delivers the result on the return channel there. The procedure *isprime* is not directly part of the language, but one can easily imagine an extension of it. Such an extension could also support let expressions, in which case the body of the server would be better represented as

$$\mathsf{req}?(x, y_{@}z)\mathsf{let}\; b = \mathit{isprime}(x)\; \mathsf{in}\; \mathsf{goto}\, z.y!\langle b\rangle\, \mathsf{stop}$$

thereby emphasising that the procedure is executed at the server's site.

A typical client of such a server takes the form

$$\begin{aligned} C \Leftarrow c[\![(\mathsf{newc}\, r : \mathsf{R})\;\; & r?(x)\, \mathsf{print}!\langle x\rangle\, \mathsf{stop} \\ & \mid \mathsf{goto}\, s.\mathsf{req}!\langle v_c, r_{@}c\rangle\, \mathsf{stop}]\!] \end{aligned}$$

Base Types:	$\mathbf{base} ::= \mathbf{int} \mid \mathbf{bool} \mid \mathbf{unit} \mid \top \mid \ldots$
Value Types:	$\mathsf{A} ::= \mathbf{base} \mid \mathsf{C} \mid \mathsf{C}_{@}\mathsf{loc} \mid \mathsf{K}$
Local Channel types:	$\mathsf{C} ::= \mathsf{r}\langle\mathsf{T}\rangle \mid \mathsf{w}\langle\mathsf{T}\rangle \mid \mathsf{rw}\langle\mathsf{T}\rangle$
Location Types:	$\mathsf{K} ::= \mathsf{loc}[c_1 : \mathsf{C}_1, \ldots, c_n : \mathsf{C}_n],\ n \geq 0$ (provided $c_i = c_j$ implies $i = j$)
Transmission Types:	$\mathsf{T} ::= (\mathsf{A}_1, \ldots, \mathsf{A}_n), n \geq 0$

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 $\top$, the main ones are

LOCAL CHANNEL TYPES: these are ranged over by C and can take the form $\mathsf{rw}\langle\mathsf{T}\rangle$, giving the ability to both read and write values of type T, or the restricted supertypes $\mathsf{r}\langle\mathsf{T}\rangle$ and $\mathsf{w}\langle\mathsf{T}\rangle$;

NON-LOCAL CHANNEL TYPES: these take the form $\mathsf{C}_{@}\mathsf{loc}$, and a value of this type is a structured value, $c_{@}l$;

LOCATION TYPES: these take the form $\mathsf{loc}[c_1 : \mathsf{C}_1, \ldots, c_n : \mathsf{C}_n]$; 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

types of resources at multiple locations.

The types come equipped with a *subtyping* relation, which is defined inductively, from the standard requirements on channel types, and *record subtyping* on location types

$$\mathsf{loc}[c_1 : \mathsf{C}_1, \ldots c_n : \mathsf{C}_n] <: \mathsf{loc}[c_1 : \mathsf{C}_1, \ldots c_k : \mathsf{C}_k], \quad \text{whenever } k \leq n$$

Viewing types (intuitively) as sets of capabilities, $\mathsf{T}_1 <: \mathsf{T}_2$ means that the capabilities of T_2 are a subset of those of T_1.

The static typing of a system M is with respect to a *type environment* Γ, giving the type of all the free names in M; for example, to type (1) we need to specify the type of req at site s and the type of print at c. Formally, a type environment Γ consists of a consistent list of entries, which must take one of the following forms

- $u : \mathsf{loc}$, indicating u is to be used as a location;
- $u_{@}w : \mathsf{C}$, indicating that w is already known to Γ as a location, and u is a local channel at w with type C.

So, for example, we would expect the system (1) to be well-typed with respect to the environment

$$\Delta \;=\; s : \mathsf{loc},\; c : \mathsf{loc},\; \mathsf{req}_{@}s : \mathsf{S},\; \mathsf{print}_{@}c : \mathsf{w}\langle \mathbf{bool} \rangle,$$

where, for the moment, we leave the type S unspecified.

The main typing judgement

$$\Gamma \vdash M,$$

indicating that M uses all its identifiers in accordance with the types designated in Γ, is defined by induction on the structure of M. The only interesting rule is

$$\frac{\Gamma \vdash_k P}{\Gamma \vdash k[\![P]\!]} \;(\text{TY-AGENT})$$

which says that $k[\![P]\!]$ is well-typed (relative to Γ) provided P is well-typed to run at the location k, $\Gamma \vdash_k P$.

This auxiliary typing judgement for agents needs to be parameterised relative to the current location, because resources are located: they may be available at one site and not another. For example, we would expect

$$\Delta,\; r_{@}c : \mathsf{R} \not\vdash_c \mathsf{req}!\langle v_c, r_{@}c \rangle\, \mathsf{stop}$$

because the channel req exists at site s but not at c, whereas we could hope for

$$\Delta,\; r_{@}c : \mathsf{R} \vdash_s \mathsf{req}!\langle v_c, r_{@}c \rangle\, \mathsf{stop}$$

if the type of req at s is properly chosen.

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

$$\frac{\Gamma \vdash_w V : \mathsf{T} \quad \Gamma \vdash_w P \quad \Gamma \vdash_w u : \mathsf{w}\langle \mathsf{T} \rangle}{\Gamma \vdash_w u!\langle V \rangle\, P}\ (\text{TY-OUT}) \qquad \frac{\Gamma,\ \langle X : \mathsf{T} \rangle_{@w} \vdash_w R \quad \Gamma \vdash_w u : \mathsf{r}\langle \mathsf{T} \rangle}{\Gamma \vdash_w u?(X)\, R}\ (\text{TY-IN})$$

while that for migration is

$$\frac{\Gamma \vdash u : \mathsf{loc} \quad \Gamma \vdash_u R}{\Gamma \vdash_w \mathsf{goto}\, u.R}\ (\text{TY-GO})$$

These rules, in turn, require the ability to assign types to identifiers, and more generally values. For example, in order for $u!\langle V \rangle\, P$ to be well-typed to run at w, (TY-OUT) dictates that u must be known at site w to be a channel with an output capability at some transmission type T, which can also be assigned to V. Similarly, according to (TY-IN), to run $u?(X : \mathsf{T})\, R$ at w, u must be known there with a read capability, and the residual R must be typeable with respect to the environment augmented assuming the variables in the pattern bound to values whose types are determined by the transmission type T. We forgo the exact explanation of how this augmented environment is constructed, that is the notation $\langle X : \mathsf{T} \rangle_{@w}$.

Referring back to the system (1), let us now see (informally) why it can be typed with respect to Δ. First notice that the channel generated by the client, r, will be used by the server to send a boolean, and by the client itself to read a boolean: so the declaration type R should be set to $\mathsf{rw}\langle \mathbf{bool} \rangle$. Then

$$\Delta,\ r_{@c} : \mathsf{R} \vdash_s \mathsf{req}!\langle v_c, r_{@c} \rangle\, \mathsf{stop}$$

follows from (TY-OUT) provided the type associated with req at s supports the inference $\Delta \vdash_s \mathsf{req} : \mathsf{w}\langle \mathbf{int}, \mathsf{R}_{w@}\mathsf{loc} \rangle$, for some type R_w which can be assigned to r; since we allow subtyping, R_w can be any supertype of R. That is, informally it can consist of any subset of the capabilities in the declaration type R. This judgement leads to

$$\Delta,\ r_{@c} : \mathsf{R} \vdash_c \mathsf{goto}\, s.\mathsf{req}!\langle v_c, r_{@c} \rangle\, \mathsf{stop}$$

via the rule (TY-GO), and eventually to that of $\Delta \vdash C$.

On the server side, the non-local channel input to req, which is bound to y, must allow the sending of boolean values. So, establishing well-typing relies on the inference $\Delta \vdash_s \mathsf{req} : \mathsf{r}\langle \mathbf{int}, \mathsf{R}_{w@}\mathsf{loc} \rangle$, with R_w set to $\mathsf{w}\langle \mathbf{bool} \rangle$. Therefore, all

that is required of Δ in order to type both the server and the client is to let S, the type associated with the request channel, to be $\mathsf{rw}\langle\, \mathbf{int}, \mathsf{w}\langle \mathbf{bool}\rangle_{@}\mathsf{loc}\,\rangle$.

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 $\mathcal{I} \triangleright M$, where

- $\mathcal{I}$ is a type environment which associates some type to every free name in M
- there is a type environment Γ such that $\Gamma \vdash M$ and $\Gamma <: \mathcal{I}$

This latter requirement means that if $\mathcal{I}$ can assign a type $\mathsf{T}_{\mathcal{I}}$ to a name n, then Γ can assign a type T_Γ such that $\mathsf{T}_\Gamma <: \mathsf{T}_{\mathcal{I}}$. Again, viewing types as sets of capabilities, this means that $\mathsf{T}_{\mathcal{I}}$, representing the knowledge of the external user, is a subset of T_Γ, 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

$$\mathcal{I} \triangleright M \xrightarrow{\mu} \mathcal{I}' \triangleright M' \tag{2}$$

where the label μ can take any of the following forms

- τ: an internal action, requiring no participation by the user;
- $(\tilde{e} : \tilde{\mathsf{E}})k.a?V$: 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;
- $(\tilde{e} : \tilde{\mathsf{E}})k.a!V$: 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 μ, and the user must have, in $\mathcal{I}$, 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 $\mathcal{I}^r(k, a)$ and $\mathcal{I}^w(k, a)$ return the read, respectively write, type associated with the channel a at the location k in $\mathcal{I}$ (of course these

(M-IN)
$$\frac{\mathcal{I}^w(k,a)\downarrow \qquad \mathcal{I} \vdash_k V : \mathcal{I}^w(k,a)}{\mathcal{I} \rhd k[\![a?(X)\,R]\!] \xrightarrow{k.a?V} \mathcal{I} \rhd k[\![R\{^V/_X\}]\!]}$$

(M-WEAK)
$$\frac{\mathcal{I}, \langle e : \mathsf{E}\rangle \rhd M \xrightarrow{(\tilde{d}:\tilde{\mathsf{D}})k.a?V} \mathcal{I}' \rhd M'}{\mathcal{I} \rhd M \xrightarrow{(e:\mathsf{E}\ \tilde{d}:\tilde{\mathsf{D}})k.a?V} \mathcal{I}' \rhd M'}\ \mathsf{bn}(e) \notin \mathcal{I}$$

(M-OUT)
$$\frac{\mathcal{I}^r(k,a)\downarrow}{\mathcal{I} \rhd k[\![a!\langle V\rangle\, P]\!] \xrightarrow{k.a!V} \mathcal{I},\ \langle V : \mathcal{I}^r(k,a)\rangle_{@}k \rhd k[\![P]\!]}$$

(M-OPEN)
$$\frac{\mathcal{I}, \langle e : \top\rangle \rhd M \xrightarrow{(\tilde{d}:\tilde{\mathsf{D}})k.a!V} \mathcal{I}' \rhd M'}{\mathcal{I} \rhd (\mathsf{new}\, e : \mathsf{E})\, M \xrightarrow{(e:\mathsf{E}\tilde{d}:\tilde{\mathsf{D}})k.a!V} \mathcal{I}' \rhd M'}$$

(M-CTXT)
$$\frac{\mathcal{I} \rhd M \xrightarrow{\mu} \mathcal{I}' \rhd M'}{\begin{array}{l}\mathcal{I} \rhd M \mid N \xrightarrow{\mu} \mathcal{I}' \rhd M' \mid N \\ \mathcal{I} \rhd N \mid M \xrightarrow{\mu} \mathcal{I}' \rhd N \mid M'\end{array}}\ \mathsf{bn}(\mu) \notin \mathsf{fn}(N)$$

(M-NEW)
$$\frac{\mathcal{I}, \langle e : \top\rangle \rhd M \xrightarrow{\mu} \mathcal{I}', \langle e : \top\rangle \rhd M'}{\mathcal{I} \rhd (\mathsf{new}\, e : \mathsf{E})\, M \xrightarrow{\mu} \mathcal{I}' \rhd (\mathsf{new}\, e : \mathsf{E})\, M'}\ \mathsf{bn}(e) \notin \mu$$

FIGURE 3. External actions-in-context for DPI

may not exist, and $\mathcal{I}^w(k,a)\downarrow$, for example, indicates that the write type is indeed defined). We extract names from entries in environments with the function $\mathsf{bn}(-)$, defined by $\mathsf{bn}(u) = u$ and $\mathsf{bn}(u_{@}w) = u$; this is extended to actions μ in the obvious manner. Finally, we use the notation $\mathsf{fn}(-)$ for free variables.

In Figure 3, the rule (M-IN) says that $\mathcal{I} \rhd k[\![a?(X)\,R]\!]$ can perform the input action $\xrightarrow{k.a?V}$ provided the user can *write* the value; that is, $\mathcal{I}$ has a write capability on a at k, and has the knowledge to actually produce V. This, in conjunction with

(M-WEAK), allows us to derive the following action from the server S

$$\Delta \rhd S \xrightarrow{(r@c:\mathsf{R})\alpha} \Delta, r_{@}c : \mathsf{R} \rhd s[\![\mathsf{goto}\, c.r!\langle \mathit{isprime}(v_c)\rangle\, \mathsf{stop}]\!] \qquad (3)$$

where α is the input action $s.\mathsf{req}?(v_c, r_{@}c)$, because

$$\Delta, r_{@}c : \mathsf{R} \rhd S \xrightarrow{\alpha} \Delta, r_{@}c : \mathsf{R} \rhd s[\![\mathsf{goto}\, c.r!\langle \mathit{isprime}(v_c)\rangle\, \mathsf{stop}]\!]$$

Similarly, (M-OUT) requires $\mathcal{I}$ to have a *read* capability on a at k, in order for $k[\![a!\langle V\rangle\, P]\!]$ to be able to perform the obvious output; note that here the current knowledge of the user, $\mathcal{I}$, is augmented by whatever new knowledge which can be gleaned from the received value V. Intuitively, $\langle V : \mathsf{T}\rangle_{@}k$ 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 $\mathcal{I}$ is $\mathcal{I}^r(k, a)$, the reception type that the user currently has on a at k. Thus (M-OPEN) allows us to deduce

$$\Delta \rhd (\mathsf{new}\, r_{@}c : \mathsf{R})\, s[\![\mathsf{req}!\langle v_c, r_{@}c\rangle\, \mathsf{stop}]\!] \xrightarrow{(r@c:\mathsf{R})\gamma} \Delta, r_{@}c : \mathsf{R}_w \rhd s[\![\mathsf{stop}]\!] \qquad (4)$$

where γ is the output action $s.\mathsf{req}!\langle v_c, r_{@}c\rangle$, because with (M-OUT) we can derive

$$\Delta, r_{@}c : \top \rhd s[\![\mathsf{req}!\langle v_c, r_{@}c\rangle\, \mathsf{stop}]\!] \xrightarrow{\gamma} \Delta, r_{@}c : \mathsf{R}_w \rhd s[\![\mathsf{stop}]\!]$$

The use of $\top$ 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 β-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, $\mathcal{I}_1$ and $\mathcal{I}_2$, are allowed. This may be surprising at first, but intuitively τ-actions should be independent of all external knowledge. For example, we can use (3) and (4) above to derive

$$\begin{array}{l}\mathcal{I} \rhd S \mid (\mathsf{new}\, r_{@}c : \mathsf{R})\, s[\![\mathsf{req}!\langle v_c, r_{@}c\rangle\, \mathsf{stop}]\!] \xrightarrow{\tau} \\ \mathcal{I} \rhd (\mathsf{new}\, r_{@}c : \mathsf{R})\, s[\![\mathsf{goto}\, c.r!\langle \mathit{isprime}(v_c)\rangle\, \mathsf{stop}]\!] \mid s[\![\mathsf{stop}]\!]\end{array}$$

for an arbitrary $\mathcal{I}$.

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

(M-COMM)
$$\frac{\mathcal{I}_1 \rhd M \xrightarrow{(\tilde{e}:\tilde{\mathsf{E}})k.a?V} \mathcal{I}'_1 \rhd M' \qquad \mathcal{I}_2 \rhd N \xrightarrow{(\tilde{e}:\tilde{\mathsf{E}})k.a!V} \mathcal{I}'_2 \rhd N'}{\mathcal{I} \rhd M \mid N \xrightarrow{\tau} \mathcal{I} \rhd (\mathsf{new}\,\widetilde{e} : \widetilde{\mathsf{E}})(M' \mid N')}$$

(M-COMM)
$$\frac{\mathcal{I}_1 \rhd M \xrightarrow{(\tilde{e}:\tilde{\mathsf{E}})k.a!V} \mathcal{I}'_1 \rhd M' \qquad \mathcal{I}_2 \rhd N \xrightarrow{(\tilde{e}:\tilde{\mathsf{E}})k.a?V} \mathcal{I}'_2 \rhd N'}{\mathcal{I} \rhd M \mid N \xrightarrow{\tau} \mathcal{I} \rhd (\mathsf{new}\,\widetilde{e} : \widetilde{\mathsf{E}})(M' \mid N')}$$

(M-MOVE)
$$\mathcal{I} \rhd k[\![\mathsf{goto}\, l.P]\!] \xrightarrow{\tau}_\beta \mathcal{I} \rhd l[\![P]\!]$$

(M-C.CREATE)
$$\mathcal{I} \rhd k[\![(\mathsf{newc}\, c : \mathsf{C})\ P]\!] \xrightarrow{\tau}_\beta \mathcal{I} \rhd (\mathsf{new}\, c_@k : \mathsf{C})\, k[\![P]\!]$$

(M-L.CREATE)
$$\mathcal{I} \rhd k[\![(\mathsf{newloc}\, l : \mathsf{L})\ P]\!] \xrightarrow{\tau}_\beta \mathcal{I} \rhd (\mathsf{new}\, l : \mathsf{L})\, k[\![P]\!]$$

(M-EQ)
$$\mathcal{I} \rhd k[\![\mathsf{if}\ v = v\ \mathsf{then}\ P\ \mathsf{else}\ Q]\!] \xrightarrow{\tau}_\beta \mathcal{I} \rhd k[\![P]\!]$$

(M-NEQ)
$$\mathcal{I} \rhd k[\![\mathsf{if}\ v_1 = v_2\ \mathsf{then}\ P\ \mathsf{else}\ Q]\!] \xrightarrow{\tau}_\beta \mathcal{I} \rhd k[\![Q]\!] \quad (v_1 \neq v_2)$$

(M-SPLIT)
$$\mathcal{I} \rhd k[\![P \mid Q]\!] \xrightarrow{\tau}_\beta \mathcal{I} \rhd k[\![P]\!] \mid k[\![Q]\!]$$

(M-UNWIND)
$$\mathcal{I} \rhd k[\![*P]\!] \xrightarrow{\tau}_\beta \mathcal{I} \rhd k[\![*P \mid P]\!]$$

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

$$\begin{array}{ccc} (\mathcal{I}_M \rhd M) \;\; \mathcal{R} \;\; (\mathcal{I}_N \rhd N) & & (\mathcal{I}_M \rhd M) \;\; \mathcal{R} \;\; (\mathcal{I}_N \rhd N) \\ \Big\downarrow \mu & \text{implies} & \Big\Downarrow \hat{\mu} \\ (\mathcal{I}_{M'} \rhd M') & & (\mathcal{I}_{M'} \rhd M') \;\; \mathcal{R} \;\; (\mathcal{I}_{N'} \rhd N') \end{array}$$

Here we use standard notation, see [MPW92], with $\stackrel{\mu}{\Longrightarrow}$ representing $\stackrel{\tau}{\longrightarrow}{}^* \circ \stackrel{\mu}{\longrightarrow} \circ \stackrel{\tau}{\longrightarrow}{}^*$, and $\stackrel{\hat{\mu}}{\Longrightarrow}$ meaning $\stackrel{\tau}{\longrightarrow}{}^*$, if μ is τ, and $\stackrel{\mu}{\Longrightarrow}$ otherwise. This allows a single internal move to be matched by zero or more internal moves.

We let $\approx_{bis}$ denote the largest bisimulation between configurations. ■

Rather than writing $(\mathcal{I} \rhd M) \approx_{bis} (\mathcal{I} \rhd N)$, we use the more suggestive notation

$$\mathcal{I} \models M \approx_{bis} N$$

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 $\approx_{bis}$ 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, in some sense, the largest reasonable typed equivalence between DPI systems.

3 Proof techniques

The basic method for showing that two systems M and N are equivalent, relative to an environment $\mathcal{I}$, is to exhibit a parameterised relation $\mathcal{R}$ such that $\mathcal{I} \models M \mathcal{R} N$, and demonstrate that it satisfies the requirements of being a bisimulation. In this section we give a number of auxiliary methods, which can considerably relieve the burden of exhibiting such relations.

The following Theorem is proved in [HMR04], and justifies a form of contextual reasoning.

THEOREM 3.1 (CONTEXTUALITY).

- $\mathcal{I} \models M \approx_{bis} N$ *and* $\mathcal{I} \vdash O$ *imply* $\mathcal{I} \models M \mid O \approx_{bis} N \mid O$
- $\mathcal{I}, \langle e : \mathsf{E} \rangle \models M \approx_{bis} N$ *implies* $\mathcal{I} \models (\mathsf{new}\, e : \mathsf{E})\, M \approx_{bis} (\mathsf{new}\, e : \mathsf{E})\, N$ ■

We can also manipulate system descriptions. Let $\equiv$ be the least equivalence relation which satisfies the rules in Figure 5, and is preserved by the constructs $- \mid -$ and $(\mathsf{new}\, e : \mathsf{E})(-)$; this is referred to as *structural equivalence*.

(S-EXTR) $(\mathsf{new}\, e : \mathsf{E})(M \mid N) \equiv M \mid (\mathsf{new}\, e : \mathsf{E})\, N \quad \text{if } \mathsf{bn}(e) \notin \mathsf{fn}(M)$

(S-COM) $M \mid N \equiv N \mid M$

(S-ASSOC) $(M \mid N) \mid O \equiv M \mid (N \mid O)$

(S-ZERO) $M \mid \mathbf{0} \equiv M$
$k[\![\mathsf{stop}]\!] \equiv \mathbf{0}$

(S-FLIP) $(\mathsf{new}\, e : \mathsf{E})\,(\mathsf{new}\, e' : \mathsf{E}')\, M \equiv (\mathsf{new}\, e' : \mathsf{E}')\,(\mathsf{new}\, e : \mathsf{E})\, M$
$\text{if } \mathsf{bn}(e) \notin (e' : \mathsf{E}'),\ \mathsf{bn}(e') \notin (e : \mathsf{E})$

FIGURE 5. Structural equivalence for DPI

PROPOSITION 3.2. *$M \equiv N$ implies $M \approx_{bis} N$.* ■

This means that we can employ the axioms in Figure 5 as equations for semantics preserving manipulations of systems. For example, from now on, we will omit the termination process stop, because $k[\![\mathsf{stop}]\!] \equiv \mathbf{0}$ and $M \mid \mathbf{0} \equiv M$.

Further equations can be obtained by considering the internal actions in Figure 4. First recall that these actions do not change the environment of a configuration, and therefore, for convenience, let us abbreviate from now on $\mathcal{I} \rhd M \xrightarrow{\tau} \mathcal{I} \rhd M'$ to the simpler $\mathcal{I} \rhd M \xrightarrow{\tau} M'$. When internal actions are β-annotated, they have the following very special *commutative property*.

LEMMA 3.3. *Suppose $\mathcal{I} \rhd M \xrightarrow{\tau}_{\beta}^{*} M'$. Then, for every action $\mathcal{I} \rhd M \xrightarrow{\mu} \mathcal{I}' \rhd N$, there is a system N' such that $\mathcal{I}' \rhd N \xrightarrow{\tau}_{\beta}^{*} N'$ and $\mathcal{I} \rhd M' \xrightarrow{\mu} \mathcal{I}' \rhd N'$.*

Proof: It is sufficient to prove this for the single labelled action $\mathcal{I} \rhd M \xrightarrow{\tau}_{\beta} M'$, for which a simple, but tedious induction on the derivation of the action can be carried out. ■

PROPOSITION 3.4. *Suppose $\mathcal{I} \rhd M \xrightarrow{\tau}_{\beta}^{*} N$. Then $\mathcal{I} \models M \approx_{bis} N$.*

Proof: Let the parameterised relation $\mathcal{R}$ be defined by letting $\mathcal{I} \rhd M \mathcal{R} N$ whenever

- $\mathcal{I} \rhd M$ is a configuration
- either $M = N$ or $M \xrightarrow{\tau}_{\beta}^{*} N$

The previous Lemma provides sufficient information to show that $\mathcal{R}$ is a bisimulation. The result therefore follows. ■

This Proposition gives more valid equations for reasoning about systems. Typical examples, obtained just by examining the axioms in Figure 4 describing

β-actions, include

$$k[\![P \mid Q]\!] \approx_{bis} k[\![P]\!] \mid k[\![Q]\!]$$
$$k[\![\mathsf{goto}\, l.P]\!] \approx_{bis} l[\![P]\!]$$
$$k[\![(\mathsf{newc}\, c : \mathsf{C})\ P]\!] \approx_{bis} (\mathsf{new}\, c_{@}k : \mathsf{C})\, k[\![P]\!]$$

But these β-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-β). A binary relation between configurations is said to be a *bisimulation up-to-β* if it satisfies the following transfer properties

$$\begin{array}{ccc} (\mathcal{I}_M \rhd M) & \mathcal{R} & (\mathcal{I}_N \rhd N) \\ \mu \downarrow & & \\ (\mathcal{I}_{M'} \rhd M') & & \end{array} \quad \text{implies} \quad \begin{array}{ccc} (\mathcal{I}_M \rhd M) & \mathcal{R} & (\mathcal{I}_N \rhd N) \\ & & \hat{\mu} \Downarrow \\ (\mathcal{I}_{M'} \rhd M') & \mathcal{A}_l \circ \mathcal{R} \circ \mathcal{A}_r & (\mathcal{I}_{N'} \rhd N') \end{array}$$

$$\begin{array}{ccc} (\mathcal{I}_N \rhd N) & \mathcal{R} & (\mathcal{I}_M \rhd M) \\ \mu \downarrow & & \\ (\mathcal{I}_{N'} \rhd N') & & \end{array} \quad \text{implies} \quad \begin{array}{ccc} (\mathcal{I}_N \rhd N) & \mathcal{R} & (\mathcal{I}_M \rhd M) \\ & & \hat{\mu} \Downarrow \\ (\mathcal{I}_{N'} \rhd N') & \mathcal{A}_l \circ \mathcal{R} \circ \mathcal{A}_r & (\mathcal{I}_{M'} \rhd M') \end{array}$$

where $\mathcal{A}_l$ is the relation $(\xrightarrow{\tau}_{\beta}^{*} \circ \equiv)$, and $\mathcal{A}_r$ is $\approx_{bis}$; strictly speaking, these relations are over systems, but they are lifted in the obvious manner to configurations. ■

The idea of these approximate bisimulations is that to match an action $(\mathcal{I}_M \rhd M) \xrightarrow{\mu} (\mathcal{I}_{M'} \rhd M')$ it is sufficient to find a β-derivative of the residual $(\mathcal{I}_{M'} \rhd M') \xrightarrow{\tau}_{\beta}^{*} (M'')$ and a matching action $(\mathcal{I}_N \rhd N) \stackrel{\hat{\mu}}{\Longrightarrow} (\mathcal{I}_{N'} \rhd N')$ such that, up-to structural equivalence and bisimilarity, respectively, the pairs $(\mathcal{I}_{M'} \rhd M'')$ and $(\mathcal{I}_{N'} \rhd N')$ are once more related. Intuitively, a configuration can represent all configurations to which it can evolve using β-moves.

LEMMA 3.6. *If $P \equiv Q$ and $\mathcal{I} \rhd P \xrightarrow{\tau}_{\beta} P'$, then there exists Q' such that $\mathcal{I} \rhd Q \xrightarrow{\tau}_{\beta} Q'$ and $P' \equiv Q'$.*

Proof: By induction on the derivation of $P \equiv Q$. ■

PROPOSITION 3.7. *If $(\mathcal{I} \rhd M)\, \mathcal{R}\, (\mathcal{I} \rhd N)$, where $\mathcal{R}$ is a bisimulation up-to-β, then $\mathcal{I} \models M \approx_{bis} N$.*

Proof: We leave to the reader to check that the relation $(\approx_{bis} \circ \mathcal{R} \circ \approx_{bis})$ is a bisimulation over configurations. The key properties for establishing this are the two inclusions $\xrightarrow{\tau}_{\beta} \subseteq \approx_{bis}$ (Proposition 3.4) and $\equiv \subseteq \approx_{bis}$ (Proposition 3.2), Lemma 3.3 and transitivity, in Definition 3.5, of both $\mathcal{A}_l$ (due to Lemma 3.6) and $\mathcal{A}_r$. The result then follows, since $(\approx_{bis} \circ \mathcal{R} \circ \approx_{bis})$ trivially contains $\mathcal{R}$. ■

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

$$F \Leftarrow (\mathsf{new}\ f : \mathsf{F})\ f[\![P \mid *\mathsf{goto}\, a.\mathsf{tell}!\langle f\rangle]\!]$$

Here f is the name of the firewall, which is created with the capabilities described in the location type F, and P is some code which maintains the internal business of the firewall. A typical example of the capabilities could be given by

$$\mathsf{F} = \mathsf{loc}[\mathsf{info} : \mathsf{rw}\langle \mathsf{I}\rangle,\ \mathsf{req} : \mathsf{rw}\langle \mathsf{R}\rangle]$$

which allow reading to and writing from two resources info and req in f. Then P could, for example, maintain appropriate services at the resources; of course, it would also be able to use non-local resources it knows about in its current environment.

The existence of the firewall is made known only to another domain, a, via the information channel tell located there. An example is the following

$$A \Leftarrow a[\![R \mid \mathsf{tell}?(x)\,\mathsf{goto}\, x.Q]\!]$$

where a is informed of f by inputing on the local channel tell. If we consider an arbitrary type environment Γ, we have the execution

$$\Gamma \rhd F \mid A \xrightarrow{\tau}{}^{*} (\mathsf{new}\ f : \mathsf{F})(f[\![P \mid *\mathsf{goto}\, a.\mathsf{tell}!\langle f\rangle \mid Q]\!]) \ \mid\ a[\![R]\!] \qquad (5)$$

so the code Q is allowed to execute locally within the firewall.

Notice that the resources to which Q has access within the firewall are controlled by the capability type associated with the information channel tell. For example, suppose in Γ the type associated with this channel is

$$\mathsf{rw}\langle \mathsf{F}_r\rangle, \qquad \mathsf{F}_r = \mathsf{loc}[\mathsf{info} : \mathsf{w}\langle \mathsf{I}\rangle,\ \mathsf{req} : \mathsf{r}\langle \mathsf{R}\rangle]$$

Then F_r is a supertype of the declaration type F: hence in (5), Q, having gained entry into the firewall, can only write to resource info and read from req.

Let us now consider the correctness of this simple protocol, which allows access of one agent, Q, to the firewall. Let Γ be any type environment such that

$$\Gamma \vdash F \mid A \qquad (6)$$

Then one might expect to be able to derive

$$\Gamma \models F \mid A \approx_{bis} (\mathsf{new}\ f : \mathsf{F})(f[\![P \mid *\mathsf{goto}\, a.\mathsf{tell}!\langle f\rangle \mid Q]\!]) \mid a[\![R]\!] \qquad (7)$$

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 $\Gamma \vdash_a \mathsf{tell} : \mathsf{rw}\langle \mathsf{F}_r\rangle$, and this allows other agents in the environment access to tell. For example, consider

$$\mathsf{Rogue} \Leftarrow b[\![\mathsf{goto}\, a.\mathsf{tell}!\langle b\rangle]\!]$$

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

$$\Gamma \models F \mid A \mid \mathsf{Rogue} \approx_{bis} (\mathsf{new}\ f : \mathsf{F})(f[\![P \mid *\mathsf{goto}\, a.\mathsf{tell}!\langle f\rangle \mid Q]\!]) \mid a[\![R]\!] \mid \mathsf{Rogue}$$

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

$$\Gamma \rhd F \mid a[\![R]\!] \mid b[\![Q]\!]$$

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 τ-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 $\Gamma \vdash_k^{max} V : \mathsf{T}$ to mean

- $\Gamma \vdash_k V : \mathsf{T}$
- $\Gamma \vdash_k V : \mathsf{T}'$ implies $\mathsf{T} <: \mathsf{T}'$

In other words, T is the *largest* type which can be assigned to V. Now suppose $\mathcal{I}$ is a type environment which satisfies

(i) $\mathcal{I} \vdash_a^{max} \mathsf{tell} : \mathsf{r}\langle \mathsf{F}\rangle$

(ii) $\mathcal{I} \vdash a[\![R]\!]$

(iii) $\mathcal{I} \vdash (\mathsf{new}\ f : \mathsf{F})\, f[\![P]\!]$

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

$$\mathcal{I} \models F \mid A \approx_{bis} (\mathsf{new}\ f : \mathsf{F})(f[\![P \mid *\mathsf{goto}\, a.\mathsf{tell}!\langle f\rangle \mid Q]\!]) \mid a[\![R]\!] \tag{8}$$

First note that (up-to structural equivalence)

$$\mathcal{I} \rhd F \mid A \xrightarrow{\tau}_{\beta} F \mid A_t \mid a[\![R]\!]$$

via (M-SPLIT) and (M-CTXT), where A_t is a shorthand for $a[\![\mathsf{tell}?(x)\,\mathsf{goto}\, x.Q]\!]$. So, by Propositions 3.2 and 3.4, it is sufficient to prove

$$\mathcal{I} \models F \mid A_t \mid a[\![R]\!] \approx_{bis} (\mathsf{new}\ f : \mathsf{F})(f[\![P \mid *\mathsf{goto}\, a.\mathsf{tell}!\langle f\rangle \mid Q]\!]) \mid a[\![R]\!]$$

Here assumption (ii) comes in useful, as by *Contextuality* it is now sufficient to prove

$$\mathcal{I} \models F \mid A_t \approx_{bis} (\mathsf{new}\ f : \mathsf{F})(f[\![P \mid *\mathsf{goto}\, a.\mathsf{tell}!\langle f\rangle \mid Q]\!])$$

Then the left-hand side can be manipulated using the structural equivalence rule (S-EXTR), thereby reducing the proof burden to

$$\begin{array}{c}\mathcal{I} \models (\mathsf{new}\ f : \mathsf{F})(f[\![P \mid *\mathsf{goto}\, a.\mathsf{tell}!\langle f\rangle]\!] \mid A_t) \approx_{bis} \\ (\mathsf{new}\ f : \mathsf{F})(f[\![P \mid *\mathsf{goto}\, a.\mathsf{tell}!\langle f\rangle \mid Q]\!])\end{array}$$

and another application of *Contextuality* reduces this further to

$$\mathcal{I}_f \models f[\![P \mid *\mathsf{goto}\, a.\mathsf{tell}!\langle f\rangle]\!] \mid A_t \approx_{bis} f[\![P \mid *\mathsf{goto}\, a.\mathsf{tell}!\langle f\rangle \mid Q]\!]$$

where $\mathcal{I}_f$ is a shorthand for $\mathcal{I}, \langle f : \mathsf{F}\rangle$.

Now let F_g represent the system $f[\![*\mathsf{goto}\, a.\mathsf{tell}!\langle f\rangle]\!]$. Then we have

- $\mathcal{I}_f \rhd f[\![P \mid *\mathsf{goto}\, a.\mathsf{tell}!\langle f\rangle]\!] \mid A_t \xrightarrow{\tau}_{\beta} f[\![P]\!] \mid F_g \mid A_t$
- $\mathcal{I}_f \rhd f[\![P \mid *\mathsf{goto}\, a.\mathsf{tell}!\langle f\rangle \mid Q]\!] \xrightarrow{\tau}_{\beta}^{*} f[\![P]\!] \mid F_g \mid f[\![Q]\!]$

So, further applications of Proposition 3.4, *Contextuality* and assumption (iii), give the requirement

$$\mathcal{I}_f \models F_g \mid A_t \approx_{bis} F_g \mid f[\![Q]\!] \tag{9}$$

This we establish directly by exhibiting a particular bisimulation.

We define the parameterised relation $\mathcal{R}$ by letting

$$\mathcal{J} \models M\, \mathcal{R}\, N$$

whenever

(a) $\mathcal{J} \rhd M$ is a configuration and N is the same as M

(b) or $\mathcal{J}$ is $\mathcal{I}_f$ and

- M has the form $F_g \mid A_t \mid \Pi_n\ (a[\![\mathsf{tell}!\langle f\rangle]\!])^n$
- N has the form $F_g \mid f[\![Q]\!] \mid \Pi_n\ (a[\![\mathsf{tell}!\langle f\rangle]\!])^n$

where $\Pi_n\ (a[\![\mathsf{tell}!\langle f\rangle]\!])^n$, for some $n \geq 0$, means n copies of $a[\![\mathsf{tell}!\langle f\rangle]\!]$ running in parallel.

PROPOSITION 4.1. *The parameterised relation $\mathcal{R}$ defined above is a bisimulation up-to-β.*

Proof: Suppose $\mathcal{J} \models M\mathcal{R}N$. Let us consider all possible actions from $\mathcal{J} \rhd M$. In fact, it is sufficient to consider the case (b) above, when $\mathcal{J}$ and M and N are of the prescribed form. The actions fall into one of three categories (for convenience we shorten $\Pi_n\ (a[\![\mathsf{tell}!\langle f\rangle]\!])^n$ with Π_n).

- Here F_g is responsible, so the action takes the form

  $$\mathcal{I}_f \rhd M \xrightarrow{\tau}_\beta f[\![*\mathsf{goto}\, a.\mathsf{tell}!\langle f\rangle \mid \mathsf{goto}\, a.\mathsf{tell}!\langle f\rangle]\!] \mid A_t \mid \Pi_n$$

  But

  $$\mathcal{I}_f \rhd f[\![*\mathsf{goto}\, a.\mathsf{tell}!\langle f\rangle \mid \mathsf{goto}\, a.\mathsf{tell}!\langle f\rangle]\!] \mid A_t \mid \Pi_n \xrightarrow{\tau}_\beta F_g \mid a[\![\mathsf{tell}!\langle f\rangle]\!] \mid A_t \mid \Pi_n$$

  and this can be matched, via clause (b), by

  $$\mathcal{I}_f \rhd N \xrightarrow{\tau}{}^*_\beta F_g \mid a[\![\mathsf{tell}!\langle f\rangle]\!] \mid f[\![Q]\!] \mid \Pi_n$$

  because $F_g|a[\![\mathsf{tell}!\langle f\rangle]\!]|A_t|\Pi_n \equiv F_g|A_t|\Pi_{n+1}$, and $F_g|a[\![\mathsf{tell}!\langle f\rangle]\!]|f[\![Q]\!]|\Pi_n \equiv F_g \mid f[\![Q]\!] \mid \Pi_{n+1}$, and $\equiv \subseteq \approx_{bis}$ (Proposition 3.2).

- The second possibility is that the third component, $\Pi_n\ (a[\![\mathsf{tell}!\langle f\rangle]\!])^n$, is responsible for the action, which must be $a.\mathsf{tell}!f$. It is easy to see that $\mathcal{I}_f \rhd N$ can perform exactly the same action, to a related configuration in clause (b).

- Finally, the middle component, A_t, might be involved in the action. Note that the action can not be external, as the action $a.\mathsf{tell}?V$ (for some value V) is not allowed by the environment. So it must be a communication, of the form

  $$\mathcal{I}_f \rhd M \xrightarrow{\tau} F_g \mid a[\![\mathsf{goto}\ f.Q]\!] \mid \Pi_{n-1}$$

  But the following β-steps can be carried out starting from this configuration

  $$\mathcal{I}_f \rhd F_g \mid a[\![\mathsf{goto}\ f.Q]\!] \mid \Pi_{n-1} \xrightarrow{\tau}{}^*_\beta F_g \mid a[\![\mathsf{tell}!\langle f\rangle]\!] \mid f[\![Q]\!] \mid \Pi_{n-1} \equiv F_g \mid f[\![Q]\!] \mid \Pi_n$$

  and this can be matched in clause (a) by the empty sequence of internal actions from $\mathcal{I}_f \rhd N$.

Symmetrically, it is easy to see that every action from $\mathcal{J} \rhd N$ can be matched by one from $\mathcal{J} \rhd M$, possibly preceded by a number of τ-actions: these latter are required when $f[\![Q]\!]$ is responsible for the action to be matched. ■

This, by using Proposition 3.7, completes our proof of (8) above.

Note that the firewall F allows, in principle, multiple entries of agents from a. So, for example, if R, in (8), had the form $R' \mid \mathsf{tell?}(x)\,\mathsf{goto}\,x.Q'$, then the reasoning we have just completed could be repeated, to prove

$$\mathcal{I} \models F \mid a[\![R]\!] \approx_{bis} (\mathsf{new}\ f : \mathsf{F})(f[\![P \mid *\mathsf{goto}\,a.\mathsf{tell!}\langle f\rangle \mid Q']\!]) \mid a[\![R']\!]$$

Then, under the assumption $\mathcal{I} \vdash (\mathsf{new}\ f : \mathsf{F})\, f[\![Q]\!]$ and by transitivity of $\approx_{bis}$, this can be combined with (8), to prove

$$\begin{aligned}\mathcal{I} \models F \mid a[\![R' \mid \mathsf{tell?}(x)\,\mathsf{goto}\,x.Q' \mid \mathsf{tell?}(x)\,\mathsf{goto}\,x.Q]\!] \approx_{bis}\\ (\mathsf{new}\ f : \mathsf{F})(f[\![P \mid *\mathsf{goto}\,a.\mathsf{tell!}\langle f\rangle \mid Q' \mid Q]\!]) \mid a[\![R']\!]\end{aligned}$$

where the domain a has managed to send two separate agents into the firewall.

5 A server and its clients

We consider in this section the canonical example of a *server* and its *clients*. A server is a domain providing services to potentially arbitrary clients, as for example the following, a generalisation of the one used as working example in section 2

$$S \Leftarrow s[\![*\mathsf{req?}(x, y_{@}z)\mathsf{goto}\,z.y!\langle isprime(x)\rangle \mid S']\!]$$

The service at resource req is here iterated, and S' provides internal code to setup and administrate the site. The channel req awaits indefinitely an integer and a located channel, checks whether the integer is a prime (according to the convention, stated in section 2, that the procedure *isprime* is executed at the server's site), and returns the answer at the proffered address.

Typical clients of the server are domains taking the form

$$C_i \Leftarrow c_i[\![(\mathsf{newc}\ r : \mathsf{R})\ \ \mathsf{goto}\ s.\mathsf{req!}\langle v_i, r_{@}c_i\rangle \mid C_i']\!]$$

These generate a private channel r at the declaration type $\mathsf{R} = \mathsf{rw}\langle\mathbf{bool}\rangle$, and send a process to the server (whose address they need to know) asking for the primality of an integer; concurrently, the agent C_i' executes at the site.

As in the case of the firewall, the correctness of the protocol between the server S and its clients C_i depends on the proper management of the access to the request channel req: clients should only have write access, while the server only needs read access. So the correctness of the protocol can be expressed as an equivalence between two systems, relative to a restricted environment.

Let $\mathcal{I}$ be a type environment satisfying

(i) $\mathcal{I} \vdash_s^{max} \mathsf{req} : \mathsf{w}\langle\, \mathbf{int}, \mathsf{w}\langle\mathbf{bool}\rangle_{@}\mathsf{loc}\,\rangle$

(ii) $\mathcal{I} \vdash s[\![S']\!]$

(iii) $\mathcal{I} \vdash C_i$

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' nor C_i' can read on req.

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

$$\mathcal{I} \models S \mid C_1 \approx_{bis} S \mid c_1[\![(\mathsf{newc}\, r : \mathsf{R})\ r!\langle isprime(v_1)\rangle \mid C_1']\!] \tag{10}$$

Note that (up-to-structural equivalence)

$$\mathcal{I} \rhd S \mid C_1 \xrightarrow{\tau}_{\beta}^{*} (\mathsf{new}\, r_{@}c_1 : \mathsf{R})\, S_r \mid s[\![S']\!] \mid s[\![\mathsf{req}!\langle v_1, r_{@}c_1\rangle]\!] \mid c_1[\![C_1']\!]$$

where we use S_r as a shorthand for $s[\![*\mathsf{req}?(x, y_{@}z)\mathsf{goto}\, z.y!\langle isprime(x)\rangle]\!]$, and

$$\begin{aligned}\mathcal{I} \rhd S \mid c_1[\![(\mathsf{newc}\, r : \mathsf{R})\ r!\langle isprime(v_1)\rangle \mid C_1']\!] &\xrightarrow{\tau}_{\beta}^{*} \\ (\mathsf{new}\, r_{@}c_1 : \mathsf{R})\, S_r \mid s[\![S']\!] \mid c_1[\![r!\langle isprime(v_1)\rangle]\!] &\mid c_1[\![C_1']\!]\end{aligned}$$

By Propositions 3.2, 3.4, *Contextuality*, and requirements (ii), (iii), it is therefore sufficient to prove

$$\mathcal{I}_r \models S_r \mid s[\![\mathsf{req}!\langle v_1, r_{@}c_1\rangle]\!] \approx_{bis} S_r \mid c_1[\![r!\langle isprime(v_1)\rangle]\!]$$

where $\mathcal{I}_r$ is a shorthand for $\mathcal{I}, \langle r_{@}c_1 : \mathsf{R}\rangle$. We establish this equivalence by exhibiting a particular parameterised relation, and showing that it satisfies the requirements to be a bisimulation. Let $\mathcal{R}$ be the parameterised relation defined by letting

$$\mathcal{J} \models M \mathcal{R} N$$

whenever

(a) $\mathcal{J} \rhd M$ is a configuration and N is the same as M

(b) or $\mathcal{J}$ is $\mathcal{I}_r$ and

- M has the form $S_r \mid s[\![\mathsf{req}!\langle v_1, r_{@}c_1\rangle]\!] \mid \Pi_n$
- N has the form $S_r \mid c_1[\![r!\langle isprime(v_1)\rangle]\!] \mid \Pi_n$

where Π_n is a shorthand for $\Pi_n\ (s[\![\mathsf{req}?(x, y_{@}z)\mathsf{goto}\, z.y!\langle isprime(x)\rangle]\!])^n$

(c) or $\mathcal{J}$ is $\mathcal{I}_r'$, where the domain of $\mathcal{I}_r'$ is a superset of that of $\mathcal{I}_r$, and

- M has the form $S_r \mid s[\![\mathsf{req}!\langle v_1, r_{@}c_1\rangle]\!] \mid \Pi_n \mid \Pi_{j\in J}\ (k_j[\![d_j!\langle isprime(i_j)\rangle]\!])$
- N has the form $S_r \mid c_1[\![r!\langle isprime(v_1)\rangle]\!] \mid \Pi_n \mid \Pi_{j\in J}\ (k_j[\![d_j!\langle isprime(i_j)\rangle]\!])$

such that $\mathcal{I}_r' \vdash_s^{max} \mathsf{req} : \mathsf{w}\langle\, \mathbf{int}, \mathsf{w}\langle \mathbf{bool}\rangle_{@}\mathsf{loc}\,\rangle$ and $\mathcal{I}_r' \vdash_{k_j} d_j : \mathsf{w}\langle \mathbf{bool}\rangle$ for every $j \in J$. Here, the notation $\Pi_{j\in J}\ (k_j[\![d_j!\langle isprime(i_j)\rangle]\!])$ means (different) instances of systems running in parallel.

PROPOSITION 5.1. *The parameterised relation $\mathcal{R}$ defined above is a bisimulation up-to-β.*

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

- First S_r is responsible

  $$\mathcal{I}_r \rhd M \xrightarrow{\tau}_\beta s[\![*\mathsf{req}?(x, y_{@}z)\mathsf{goto}\, z.y!\langle isprime(x)\rangle \mid R']\!] \mid s[\![\mathsf{req}!\langle v_1, r_{@}c_1\rangle]\!] \mid \Pi_n$$

  where R' is a shorthand for $\mathsf{req}?(x, y_{@}z)\mathsf{goto}\, z.y!\langle isprime(x)\rangle$. But

  $$\mathcal{I}_r \rhd s[\![*\mathsf{req}?(x, y_{@}z)\mathsf{goto}\, z.y!\langle isprime(x)\rangle \mid R']\!] \mid s[\![\mathsf{req}!\langle v_1, r_{@}c_1\rangle]\!] \mid \Pi_n \xrightarrow{\tau}_\beta S_r \mid \Pi_1 \mid s[\![\mathsf{req}!\langle v_1, r_{@}c_1\rangle]\!] \mid \Pi_n$$

  and this can be matched by

  $$\mathcal{I}_r \rhd N \xrightarrow{\tau}_\beta^* S_r \mid \Pi_1 \mid c_1[\![r!\langle isprime(v_1)\rangle]\!] \mid \Pi_n$$

  because both configurations belong to $\mathcal{R}$, clause (b), up-to structural equivalence.

- The third component, Π_n $(s[\![\mathsf{req}?(x, y_{@}z)\mathsf{goto}\, z.y!\langle isprime(x)\rangle]\!])^n$, is responsible for the action, which is either $s.\mathsf{req}?\langle i_j, d_j@k_j\rangle$ or $(e{:}E)s.\mathsf{req}?\langle i_j, d_j@k_j\rangle$. 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 $\mathcal{I}_r \rhd N$ can perform exactly the same actions, to a related configuration in clause (c).

- Finally, the middle component, $s[\![\mathsf{req}!\langle v_1, r_{@}c_1\rangle]\!]$, might be involved in the action, which must be a communication, of the form

  $$\mathcal{I}_r \rhd M \xrightarrow{\tau} S_r \mid s[\![\mathsf{goto}\, c_1.r!\langle isprime(v_1)\rangle]\!] \mid \Pi_{n-1}$$

  Then the following β-steps can be carried out

  $$\mathcal{I}_r \rhd S_r \mid s[\![\mathsf{goto}\, c_1.r!\langle isprime(v_1)\rangle]\!] \mid \Pi_{n-1} \xrightarrow{\tau}_\beta^* S_r \mid \Pi_1 \mid c_1[\![r!\langle isprime(v_1)\rangle]\!] \mid \Pi_{n-1}$$

  and this configuration can be matched, in clause (a), by the empty sequence of actions from $\mathcal{I}_r \rhd N$.

Symmetrically, every action performed by $\mathcal{I}_r \rhd N$ can be matched by $\mathcal{I}_r \rhd M$; for example, consider the output action by the second component of N

$$\mathcal{I}_r \rhd S_r \mid c_1[\![r!\langle isprime(v_1)\rangle]\!] \mid \Pi_n \xrightarrow{c_1.r!\langle isprime(v_1)\rangle} \mathcal{I}_r \rhd S_r \mid \Pi_n$$

This can be easily matched by $\mathcal{I}_r \rhd M$, via clause (a), using τ-steps followed by the same action.

Finally, it is not problematic to check that all configurations in $\mathcal{R}$ by virtue of clause (c) can have their respective actions properly matched. ■

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

$$\mathcal{I} \models S \mid C_1 \mid C_2 \;\approx_{bis}\; S \mid \Pi_{i\in\{1,2\}}\, c_i[\![(\mathsf{newc}\, r : \mathsf{R})\;\; r!\langle isprime(v_i)\rangle \mid C_i']\!] \qquad (11)$$

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

$$\mathcal{I} \models S \mid C_1 \mid C_2 \;\approx_{bis}\; S \mid c_1[\![(\mathsf{newc}\, r : \mathsf{R})\;\; r!\langle isprime(v_1)\rangle \mid C_1']\!] \mid C_2 \qquad (12)$$

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

$$\mathcal{I} \models S \mid C_2 \;\approx_{bis}\; S \mid c_2[\![(\mathsf{newc}\, r : \mathsf{R})\;\; r!\langle isprime(v_2)\rangle \mid C_2']\!]$$

But $\mathcal{I} \vdash C_1$ (again (iii)) also implies $\mathcal{I} \vdash c_1[\![(\mathsf{newc}\, r : \mathsf{R})\;\; r!\langle isprime(v_1)\rangle \mid C_1']\!]$, and therefore, by *Contextuality*

$$\begin{array}{c}\mathcal{I} \models S \mid C_2 \mid c_1[\![(\mathsf{newc}\, r : \mathsf{R})\;\; r!\langle isprime(v_1)\rangle \mid C_1']\!] \approx_{bis} \\ S \mid \Pi_{i\in\{1,2\}}\, c_i[\![(\mathsf{newc}\, r : \mathsf{R})\;\; r!\langle isprime(v_i)\rangle \mid C_i']\!]\end{array}$$

So we conclude (11) from (12), Proposition 3.2, and transitivity of $\approx_{bis}$.

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

$$\mathcal{I} \models S \mid \Pi_{i\in\{1,\ldots,n\}}\, C_i \;\approx_{bis}\; S \mid \Pi_{i\in\{1,\ldots,n\}}\, c_i[\![(\mathsf{newc}\, r : \mathsf{R})\;\; r!\langle isprime(v_i)\rangle \mid C_i']\!]$$

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' and C_i': we set S' to stop and C_i' to $r?(x)\,\mathsf{print}_i!\langle x\rangle$, where print_i 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

$$\begin{array}{l} S'' \Leftarrow s[\![*\mathsf{req}?(x, y_{@}z)\mathsf{goto}\, z.y!\langle isprime(x)\rangle]\!] \\ C_1'' \Leftarrow c_1[\![(\mathsf{newc}\, r : \mathsf{R})\;\; \mathsf{goto}\, s.\mathsf{req}!\langle 4, r_{@}c_1\rangle \mid r?(x)\,\mathsf{print}_1!\langle x\rangle]\!] \\ C_2'' \Leftarrow c_2[\![(\mathsf{newc}\, r : \mathsf{R})\;\; \mathsf{goto}\, s.\mathsf{req}!\langle 3, r_{@}c_2\rangle \mid r?(x)\,\mathsf{print}_2!\langle x\rangle]\!] \end{array}$$

and we want to prove the following

$$\mathcal{I} \models S'' \mid C_1'' \mid C_2'' \;\approx_{bis}\; S'' \mid c_1[\![\mathsf{print}_1!\langle false\rangle]\!] \mid c_2[\![\mathsf{print}_2!\langle true\rangle]\!]$$

This follows from (11) (the requirement (iii) holds for every residual C_i', so $\mathcal{I}$ is supposed to take into account the local channels print_1 and print_2, in the case) if we can establish

$$\begin{array}{c}\mathcal{I} \models S'' \mid \Pi_{i\in\{1,2\}}\, c_i[\![(\mathsf{newc}\, r : \mathsf{R})\;\; r!\langle isprime(v_i)\rangle \mid r?(x)\,\mathsf{print}_i!\langle x\rangle]\!] \;\approx_{bis} \\ S'' \mid c_1[\![\mathsf{print}_1!\langle false\rangle]\!] \mid c_2[\![\mathsf{print}_2!\langle true\rangle]\!]\end{array}$$

But *Contextuality* with requirement (iii), as usual, allows us to simplify this fur-

ther, to the tasks

$$\mathcal{I} \models S'' \mid c_1[\![(\mathsf{newc}\, r : \mathsf{R})\ r!\langle isprime(4)\rangle \mid r?(x)\, \mathsf{print}_1!\langle x\rangle]\!] \approx_{bis} S'' \mid c_1[\![\mathsf{print}_1!\langle false\rangle]\!]$$

$$\mathcal{I} \models S'' \mid c_2[\![(\mathsf{newc}\, r : \mathsf{R})\ r!\langle isprime(3)\rangle \mid r?(x)\, \mathsf{print}_2!\langle x\rangle]\!] \approx_{bis} S'' \mid c_2[\![\mathsf{print}_2!\langle true\rangle]\!]$$

Note that *Contextuality* does not allow us to eliminate S'' from these judgements, since $\mathcal{I} \vdash S''$ 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 location name at the reply address

$$S \Leftarrow s[\![*\mathsf{setup}?(y_{@}z)\,(\mathsf{newloc}\, m : \mathsf{M})\ \mathsf{goto}\, m.\mathsf{Mem} \mid \mathsf{goto}\, z.y!\langle m\rangle]\!]$$

where Mem is the code running at the location m, and for instance can take the form

$$\begin{aligned}\mathsf{Mem} \Leftarrow (\mathsf{newc}\, v : \mathsf{V})\ & v!\langle 0\rangle \mid *\mathsf{get}?(y_{@}z)\ v?(w)\ (\mathsf{goto}\, z.y!\langle w\rangle \mid v!\langle w\rangle) \\ & \mid *\mathsf{put}?(x, y_{@}z)\ v?(w)\ (\mathsf{goto}\, z.y! \mid v!\langle x\rangle)\end{aligned}$$

Here we are using the channel v as a restricted form of memory cell: the value it contains (whose initial value is set to 0) disappears once it is read, therefore it has to be reinstated. The methods get and put can be seen as the canonical ways to access the cell, therefore the declaration type of the new site can be set to

$$\mathsf{M} = \mathsf{loc}[\mathsf{get} : \mathsf{T}_g, \mathsf{put} : \mathsf{T}_p]$$

Notice that we have chosen this particular instantiation for the running code Mem just for reasons of simplicity, as the proofs we are going to develop are, in principle, independent of it.

Clients of the memory service generate a new reply channel, send a request to the server, and wait for the server to deliver the new memory cell

$$C_i \Leftarrow c_i[\![(\mathsf{newc}\, r : \mathsf{R})\ \mathsf{goto}\, s.\mathsf{setup}!\langle r_{@}c_i\rangle \mid r?(x)\, P_i(x)]\!]$$

where $P_i(x)$ is parametric code which depends on (the name of) the new site, x, and $\mathsf{R} = \mathsf{rw}\langle\mathsf{M}\rangle$.

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 proffered site

$$S' \Leftarrow s'[\![*\mathsf{setup}'?(x, y_{@}z)\ \mathsf{goto}\ x.\mathsf{Mem} \mid \mathsf{goto}\ z.y!]\!]$$

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

$$C_i' \Leftarrow c_i[\![(\mathsf{newc}\ t : \mathsf{T})\ \ (\mathsf{newloc}\ m_i : \mathsf{M})\ \ \mathsf{goto}\ s'.\mathsf{setup}'!\langle m_i, t_{@}c_i\rangle \mid t?P_i(m_i)]\!]$$

where $\mathsf{T} = \mathsf{rw}\langle\mathbf{unit}\rangle$.

We want now to relate the two different approaches, therefore connecting the behaviour of the two following systems, relative to a typing environment $\mathcal{I}$

$$\mathcal{I} \models S \mid C_1 \mid C_2 \tag{13}$$

$$\mathcal{I} \models S' \mid C_1' \mid C_2' \tag{14}$$

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' lead to equivalent behaviour. This means finding a suitable type environment $\mathcal{I}$ such that

$$\mathcal{I} \models\ S \mid C_1 \mid C_2\ \approx_{bis}\ S' \mid C_1' \mid C_2' \tag{15}$$

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* setup′ *channels*. Thus, the equivalence can be proved relative to a restricted environment $\mathcal{I}$, satisfying

$$\mathcal{I} \vdash_{s}^{max} \mathsf{setup} : \top \qquad\qquad \mathcal{I} \vdash_{s'}^{max} \mathsf{setup}' : \top$$

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

$$\mathcal{I} \models S \mid \Pi_{i\in\{1,2\}}\ (\mathsf{new}\ m_i : \mathsf{M})(m_i[\![\mathsf{Mem}]\!] \mid c_i[\![P_i(m_i)]\!]) \tag{16}$$

$$\mathcal{I} \models S' \mid \Pi_{i\in\{1,2\}}\ (\mathsf{new}\ m_i : \mathsf{M})(m_i[\![\mathsf{Mem}]\!] \mid c_i[\![P_i(m_i)]\!]) \tag{17}$$

Therefore we address (15) in three steps: first we prove that the two pairs of systems (13),(16) and (14),(17) are equivalent; then we connect the systems (16) and (17) by a technical lemma. That is

(i) $\mathcal{I} \models\ S \mid \Pi_{i\in\{1,2\}}\ C_i\ \approx_{bis}\ S \mid \Pi_{i\in\{1,2\}}\ (\mathsf{new}\ m_i : \mathsf{M})(m_i[\![\mathsf{Mem}]\!] \mid c_i[\![P_i(m_i)]\!])$

(ii) $\mathcal{I} \models\ S' \mid \Pi_{i\in\{1,2\}}\ C_i'\ \approx_{bis}\ S' \mid \Pi_{i\in\{1,2\}}\ (\mathsf{new}\ m_i : \mathsf{M})(m_i[\![\mathsf{Mem}]\!] \mid c_i[\![P_i(m_i)]\!])$

(iii) $\mathcal{I} \models\ l[\![*a?(x)P(x)]\!] \mid Q\ \approx_{bis}\ Q$, for any $\mathcal{I}, Q, l, a, P$ such that $\mathcal{I} \vdash_{l}^{max} a : \top$ and $a \notin \mathsf{fn}(Q)$

The proof of the point (iii) is straightforward, as a witness bisimulation $\mathcal{R}$ can be promptly defined by letting $\mathcal{J} \models\ M\,\mathcal{R}\,N$ whenever

(a) $\mathcal{J} \rhd M$ and $\mathcal{J} \rhd N$ are configurations

(b) $\mathcal{J}$ is $\mathcal{I}$ and M has the form $l[\![*a?(x)P(x)]\!] \mid \Pi_n\ (l[\![a?(x)P(x)]\!])^n \mid N$

which can be easily proved to be a bisimulation up-to-β.

We argue below both the proof of (i) (leaving the one of (ii), which is completely similar, to the reader) and how to get the proof of (15) from those of (i), (ii), (iii). Let us start from the latter.

Using the equations (i) and (ii), the equivalence (15) can be reduced to

$$\mathcal{I} \models\ S \mid \Pi_{i\in\{1,2\}}\ Q_i\ \approx_{bis}\ S' \mid \Pi_{i\in\{1,2\}}\ Q_i \tag{18}$$

where Q_i denotes $(\mathsf{new}\ m_i : \mathsf{M})(m_i[\![\mathsf{Mem}]\!] \mid c_i[\![P_i(m_i)]\!])$. It is natural now to assume that the conditions required by the lemma (iii) are satisfied by the code Q_i ($\mathsf{setup}, \mathsf{setup}' \notin \mathsf{fn}(Q_i)$, in the case). Hence, it is possible to apply that lemma to both the sides of the equation (18), thus obtaining an identity.

Finally, we address the proof of the point (i).
First notice that (up-to structural equivalence)

$$\mathcal{I} \rhd S \mid \Pi_{i\in\{1,2\}}\ C_i \xrightarrow{\tau}{}^{*}_{\beta}\ (\mathsf{new}\, r_{1@}c_1 : \mathsf{R}, r_{2@}c_2 : \mathsf{R})\ S \mid \Pi_{i\in\{1,2\}}\ (s[\![\mathsf{setup}!\langle r_{i@}c_i\rangle]\!] \mid c_i[\![r_i?(x)\, P_i(x)]\!])$$

and $S \mid \Pi_{i\in\{1,2\}}\ Q_i \ \equiv\ (\mathsf{new}\, m_1 : \mathsf{M}, m_2 : \mathsf{M})\ S \mid \Pi_{i\in\{1,2\}}\ (m_i[\![\mathsf{Mem}]\!] \mid c_i[\![P_i(m_i)]\!])$. Therefore, by Propositions 3.2, 3.4, we reduce (i) to the following

$$\mathcal{I} \models (\mathsf{new}\, r_{1@}c_1{:}\mathsf{R}, r_{2@}c_2{:}\mathsf{R})\ S \mid \Pi_{i\in\{1,2\}}\ (s[\![\mathsf{setup}!\langle r_{i@}c_i\rangle]\!] \mid c_i[\![r_i?(x)\, P_i(x)]\!]) \approx_{bis} (\mathsf{new}\, m_1 : \mathsf{M}, m_2 : \mathsf{M})\ S \mid \Pi_{i\in\{1,2\}}\ (m_i[\![\mathsf{Mem}]\!] \mid c_i[\![P_i(m_i)]\!])$$

which we prove by exhibiting a particular bisimulation. Let us fix before some shorthand notation

$$\begin{aligned}
S_{!i} &\triangleq s[\![\mathsf{setup}!\langle r_{i@}c_i\rangle]\!]\\
C_{?i} &\triangleq c_i[\![r_i?(x)\, P_i(x)]\!]\\
M_i &\triangleq m_i[\![\mathsf{Mem}]\!]\\
C_{!i} &\triangleq c_i[\![r_i!\langle m_i\rangle]\!]\\
C_{P_i} &\triangleq c_i[\![P_i(m_i)]\!]\\
B &\triangleq (\mathsf{new}\ m_1, m_2 : \mathsf{M})\ S \mid \Pi_n \mid M_1 \mid C_{P_1} \mid M_2 \mid C_{P_2}\\
\Pi_n &\triangleq \Pi_n\ (s[\![\mathsf{setup}?(y_{@}z)\,(\mathsf{newloc}\, m : \mathsf{M})\ \mathsf{goto}\, m.\mathsf{Mem} \mid \mathsf{goto}\, z.y!\langle m\rangle]\!])^n
\end{aligned}$$

We define the parameterised relation $\mathcal{R}$ by letting

$$\mathcal{J} \models\ P\, \mathcal{R}\, Q$$

whenever $\mathcal{J} \rhd P$ is a configuration and Q is the same as P, or $\mathcal{J}$ is $\mathcal{I}$ and Q has the form B and P has the form

(a) $(\mathsf{new}\, r_{1@}c_1 : \mathsf{R}, r_{2@}c_2 : \mathsf{R})\ S \mid \Pi_n \mid S_{!1} \mid S_{!2} \mid C_{?1} \mid C_{?2}$

(b) or $(\mathsf{new}\, r_{1@}c_1 : \mathsf{R}, r_{2@}c_2 : \mathsf{R}, m_1 : \mathsf{M})\ S \mid \Pi_n \mid S_{!2} \mid C_{?2} \mid M_1 \mid C_{!1} \mid C_{?1}$

(c) or $(\mathsf{new}\, r_{1@}c_1 : \mathsf{R}, r_{2@}c_2 : \mathsf{R}, m_2 : \mathsf{M})\ S \mid \Pi_n \mid S_{!1} \mid C_{?1} \mid M_2 \mid C_{!2} \mid C_{?2}$

(d) or $(\mathsf{new}\, r_{2@}c_2 : \mathsf{R}, m_1 : \mathsf{M})\ S \mid \Pi_n \mid S_{!2} \mid C_{?2} \mid M_1 \mid C_{P_1}$

(e) or $(\mathsf{new}\, r_{1@}c_1 : \mathsf{R}, m_2 : \mathsf{M})\ S \mid \Pi_n \mid S_{!1} \mid C_{?1} \mid M_2 \mid C_{P_2}$

(f) or $(\mathsf{new}\, r_{1@}c_1 : \mathsf{R}, r_{2@}c_2 : \mathsf{R}, m_1 : \mathsf{M}, m_2 : \mathsf{M})\ S \mid \Pi_n \mid M_1 \mid C_{!1} \mid C_{?1} \mid M_2 \mid C_{!2} \mid C_{?2}$

(g) or $(\mathsf{new}\, r_{2@}c_2 : \mathsf{R}, m_1 : \mathsf{M}, m_2 : \mathsf{M})\ S \mid \Pi_n \mid M_1 \mid C_{P_1} \mid M_2 \mid C_{!2} \mid C_{?2}$

(h) or $(\mathsf{new}\, r_{1@}c_1 : \mathsf{R}, m_1 : \mathsf{M}, m_2 : \mathsf{M})\ S \mid \Pi_n \mid M_1 \mid C_{!1} \mid C_{?1} \mid M_2 \mid C_{P_2}$

The relation $\mathcal{R}$ draws a directed graph, whose edges can be labelled by communications.

PROPOSITION 6.1. *The parameterised relation $\mathcal{R}$ defined above is a bisimulation up-to-β.* ■

We leave to the reader the task of proving the Proposition, as no extra critical aspects arise with respect to the proofs detailed in the previous two sections. Summing up, we have shown that

$$\mathcal{I} \models\ S \mid C_1 \mid C_2\ \approx_{bis}\ S' \mid C_1' \mid C_2'$$

under the following assumptions

- $\mathcal{I} \vdash_s^{max} \mathsf{setup} : \top$ and $\mathcal{I} \vdash_{s'}^{max} \mathsf{setup}' : \top$
- $\mathsf{setup} \notin \mathsf{fn}(\mathsf{Mem})$ and $\mathsf{setup}' \notin \mathsf{fn}(\mathsf{Mem})$
- $\mathsf{setup} \notin \mathsf{fn}(P_i)$ and $\mathsf{setup}' \notin \mathsf{fn}(P_i)$

It is then possible to consider an arbitrary number of clients. The correctness of these can once more be addressed using the techniques, such as *Contextuality*, discussed in the previous sections.

7 Related and future work

Proofs of correctness of protocols or language translations are often carried out with respect to *contextual* equivalences [GC99, LS00]. Nevertheless, the use of bisimulation-based notions of equivalences enables such proofs to be considerably simplified. For instance, in [MN03], two up-to proof techniques (up-to expansion and up-to context) are borrowed from the PICALCULUS and adapted to develop an algebraic theory and prove the correctness of the perfect firewall protocol [CG98]. Our paper tries to contribute to this second approach, using bisimulations, extending their application to situations in which the environment plays a significant role in system behaviour.

In this document, we have defined and illustrated a collection of methods for proving bisimulation equivalences for distributed, mobile systems, modelled

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 β-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*, for addressing the verification of sample access protocols, such as crossing a firewall, the interaction between a server and its clients, and metaservers providing memory services.

In the future, we plan to test further with the up-to β-reduction and the auxiliary techniques we have devised, by dealing with more involved protocols, possibly in the spirit of [US01]. That work uses a novel notion of coupled simulation that, despite not coinciding with any contextual equivalence, allows the proof of correctness of a simple central-forwarding-server algorithm.

We would like also to extend the results and techniques stated for Dpi to the more involved SafeDpi [HRY04], which takes into account extra safety aspects of distributed systems.

Acknowledgements. The authors would like to acknowledge the financial support of the two EU Global Computing projects, *Mikado* and *Myths*.

References

[CG98] Luca Cardelli and Andrew D. Gordon. Mobile ambients. In Proc. of *FoSSaCS*, *Lecture Notes in Computer Science* 1378, Springer, 1998.

[GC99] Andrew D. Gordon and Luca Cardelli. Equational properties of mobile ambients. In Proc. of *FoSSaCS*, *Lecture Notes in Computer Science* 1578, Springer, 1999.

[HMR04] Matthew Hennessy, Massimo Merro, and Julian Rathke. Towards a behavioural theory of access and mobility control in distributed systems. *Theoretical Computer Science* 322(3), 2004.

[HR02a] Matthew Hennessy and Julian Rathke. Typed behavioural equivalences for processes in the presence of subtyping. *Electronic Notes in Theoretical Computer Science* 61, 2002.

[HR02b] Matthew Hennessy and James Riely. Resource access control in systems of mobile agents. *Information and Computation* 173(1), 2002.

[HRY04] Matthew Hennessy, Julian Rathke, and Nobuko Yoshida. SafeDpi: a language for controlling mobile code. In Proc. of *FoSSaCS*, *Lecture Notes in Computer Science* 2987, Springer, 2004.

[JR04] Alan Jeffrey and Julian Rathke. A theory of bisimulation for a fragment of concurrent ML with local names. *Theoretical Computer Science* 323(1-3), 2004.

[LS00] Francesca Levi and Davide Sangiorgi. Controlling interference in ambients. In Proc. of *POPL*, 2000.

[Mil89] Robin Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[MN03] Massimo Merro and Francesco Zappa Nardelli. Bisimulation proof methods for mobile ambients. In Proc. of *ICALP*, *Lecture Notes in Computer Science* 2719, Springer, 2003.

[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 PICALCULUS. *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* PICALCULUS*: a Theory of Mobile Processes*. Cambridge University Press, 2001.

[US01] Asis Unyapoth and Peter Sewell. Nomadic pict: correct communication infrastructure for mobile computation. In Proc. of *POPL*, 2001.