Modelling Dynamic Web Data

Philippa Gardner Sergio Maffeis*

Department of Computing
Imperial College London
{pg,maffeis}@doc.ic.ac.uk

October 2003

Abstract

We introduce X$d\pi$, a peer-to-peer model for reasoning about the dynamic behaviour of web data. It is based on an idealised model of semi-structured data, and an extension of the π-calculus with process mobility and with operations for interacting with data. Our model can be used to reason about behaviour found in, for example, dynamic web page programming, applet interaction, and service orchestration. We study behavioural equivalences for X$d\pi$, motivated by examples.

* Supported by a grant from Microsoft Research, Cambridge.

Contents

1 Introduction

Web data, such as XML, plays a fundamental rôle in the exchange of information between globally distributed applications. Applications naturally fall into some sort of mediator approach: systems are divided into peers, with mechanisms based on XML for interaction between peers. The development of analysis techniques, languages and tools for web data is by no means straightforward. In particular, although web services allow for interaction between processes and data, direct interaction between processes is not well-supported.

Peer-to-peer data management systems are decentralised distributed systems where each component offers the same set of basic functionalities and acts both as a producer and as a consumer of information. We model systems where each peer consists of an XML data repository and a working space where processes are allowed to run. Our processes can be regarded as agents with a simple set of functionalities; they communicate with each other, query and update the local repository, and migrate to other peers to continue execution. Process definitions can be included in documents[1], and can be selected for execution by other processes. These functionalities are enough to express most of the dynamic behaviour found in web data, such as web services, distributed (and replicated) documents [2], distributed query patterns [26], hyperlinks, forms, and scripting.

In this paper we introduce the X$d\pi$-calculus, which provides a formal semantics for the systems described above. It is based on a network of locations (peers) containing a (semi-structured) data model, and π-like processes [22, 27, 17] for modelling process interaction, process migration, and interaction with data. The data model consists of unordered labelled trees, with embedded processes for querying and updating such data, and explicit pointers for referring to other parts of the network: for example, a document with a hyperlink referring to another site and a light-weight trusted process for retrieving information associated with the link. The idea of embedding processes (scripts) in web data is not new: examples include Javascript, SmartTags and calls to web services. However, web applications do not in general provide direct communication between active processes, and process coordination therefore requires specialised orchestration tools. In contrast, distributed process interaction (communication and co-ordination) is central to our model, and is inspired by the current research on distributed process calculi.

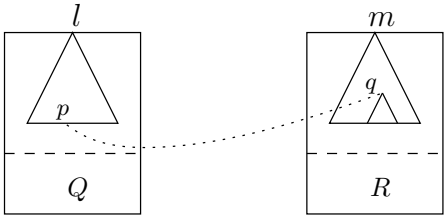
We study behavioural equivalences for X$d\pi$. In particular, we define when two processes are equivalent in such a way that when the processes are put in the same position in a network, the resulting networks are equivalent. We do this in several stages. First, we define what it means for two X$d\pi$-networks to be equivalent. Second, we give a translation of X$d\pi$ into a simpler calculus (Xπ_2), where the location structure has been pushed inside the data and processes. This translation technique, first proposed in [9], enables us to separate reasoning about processes from reasoning about data and networks. Third, we define

[1] We regard process definitions in documents as an atomic piece of data, and we do not consider queries which modify such definitions.

process equivalence for Xπ_2, and provide a proof method based on a labelled bisimulation. Finally, we study examples: in particular, we prove how services can be transparently replicated.

1.1 A Simple Example

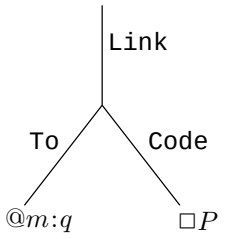
We use the *hyperlink example* as a simple example to illustrate our ideas. Consider two locations (machines identified by their IP addresses) l and m. Location l contains a hyperlink at p referring to data identified by path q at location m:



In the working space of location l, the process Q activates the process embedded in the hyperlink, which then fires a request to m to copy the tree identified by path q and write the result to p at l.

The hyperlink at l, written in both XML notation (LHS) and ambient notation used in this paper (RHS), is:

```
<Link>
    <To>
      m:q
    </To>
    <Code>
      P
    </Code>
</Link>
```



$\texttt{Link}[\,\texttt{To}[\,@m{:}q\,]\,|\,\texttt{Code}[\,\Box P\,]\,]$

This hyperlink consists of two parts: an external pointer $@m{:}q$, and a scripted process $\Box P$ which provides the mechanism to fetch the subtree q from m. Process P has the form

$$P = \mathsf{read}_{p/\texttt{Link/To}}(@x{:}y).\overline{load}\langle x, y, p\rangle$$

The read command reads the pointer reference from position $p/\texttt{Link/To}$ in the tree, substitutes the values m and q for the parameters x and y in the continuation and evolves to the output process $\overline{load}\langle m, q, p\rangle$, which records the target location m, the target path q and the position p where the result tree will go. This output process calls a corresponding input process inside Q, using π-calculus interaction. The specification of Q has the form:

$$Q_s \;= !load(x, y, z).\mathsf{go}\, x.\, \mathsf{copy}_y(X).\mathsf{go}\, l.\mathsf{paste}_z\langle X\rangle$$

The replication ! denotes that the input process can be used as many times as requested. The interaction between the $\overline{load}$ and $load$ replaces the parameters x, y, z with the values m, q, p in the continuation. The process then goes to m, copies the tree at q, comes back to l, and pastes the tree to p.

The process Q_s is just a specification. We refine this specification by having a process Q (acting as a service call), which sends a request to location m for the tree at q, and a process R (the service definition) which grants the request. Processes Q and R are defined by

$$Q = !load(x, y, z).(\nu c)(\mathsf{go}\, x.\overline{get}\langle y, l, c\rangle \mid c(X).\mathsf{paste}_z\langle X\rangle)$$

$$R = !get(y, x, w).\mathsf{copy}_y(X).\mathsf{go}\, x.\overline{w}\langle X\rangle$$

Once process Q receives parameters from $\overline{load}$, it splits into two parts: the process that sends the output message $\overline{get}\langle q, l, c\rangle$ to m, with information about the particular target path q, the return location l and a private channel name c (created using the π-calculus restriction operator ν), and the process $c(X).\mathsf{paste}_p\langle X\rangle$ waiting to paste the result delivered via the unique channel c. Process R receives the parameters from $\overline{get}$ and returns the tree to the unique channel c at l. Using our definition of process equivalence, we show that Q does indeed have the same intended behaviour as its specification Q_s, and that the processes are interchangeable independently from the context where they are installed.

1.2 Related Work

Our work is related to the Active XML approach to data integration developed independently by Abiteboul et al. [4]. They introduce an architecture which is a peer-to-peer system where each peer contains a data model very similar to ours (but where only service calls can be scripted in documents), and a working space where only web service definitions are allowed. Moreover Active XML service definitions cannot be moved around. In this respect our approach is more flexible: for example, we can define an auditing process for assessing a university course—it goes to a government site, selects the assessment criteria appropriate for the particular course under consideration, then moves this information (web service definition) to the university to make the assessment.

Several distributed query languages, such as [24, 19, 8], extend traditional query languages with facilities for distribution awareness. Our approach is closest to the one of Sahuguet and Tannen [26], who introduce the ubQL query language based on streams for exchanging large amounts of distributed data, partly motivated by ideas from the π-calculus. There has been much study of data models for the web in the XML, database and process algebra communities. Our ideas have evolved from those found in [3] and [10]. Our process-algebra techniques have most in common with [18, 9, 16]. Process calculi have also been used for example to study security properties of web services [15], reason about mobile resources [14], and in [25] to sketch a distributed query language. Bierman and Sewell [7] have recently extended a small functional language for XML

with π-calculus-based concurrency in order to program Home Area Networks devices.

Our work is the first attempt to integrate the study of mobile processes and semi-structured data for Web-based data-sharing applications, and is characterised by its emphasis on dynamic data.

2 A model for dynamic web data

We model a peer-to-peer system as a sets of interconnected locations (*networks*), where the content of each location consists of an abstraction of a XML data repository (the *tree*) and a term representing both the services provided by the peer and the agents in execution on behalf of other peers (the *process*). Processes can query and update the local data, communicate with each other through named *channels* (public or private), and migrate to other peers. Process definitions can be included in documents and can be selected for execution by other processes.

2.1 Syntax

2.1.1 Trees

Our data model extends the unordered labelled rooted trees of [10], with leaves which can either be scripted processes or pointers to data. We use the following constructs: *edge labels* denoted by $\mathtt{a}, \mathtt{b}, \mathtt{c} \in \mathcal{A}$, *path expressions* denoted by $p, q \in \mathcal{E}$ used to identify specific subtrees, and *locations* of the form $\circlearrowleft, l, m \in \mathcal{L}$, where the 'self' location $\circlearrowleft$ refers to the enclosing location. The set of data trees, denoted $\mathcal{T}$, is given by

$$T ::= 0 \mid T \mid T \mid \mathtt{a}[\,T\,] \mid \mathtt{a}[\,\Box P\,] \mid \mathtt{a}[\,@l{:}p\,]$$

Tree 0 denotes a rooted tree with no content. Tree $T_1 \mid T_2$ denotes the composition of T_1 and T_2, which simply joins the roots. A tree of the form $\mathtt{a}[\,...\,]$ denotes a tree with a single branch labelled $\mathtt{a}$ which can have three types of content: a subtree T; a *scripted process* $\Box P$ which is a static process awaiting a command to run; a *pointer* $@l{:}p$ which denotes a pointer to a set of subtrees identified by path expression p in the tree at location l. Processes and path expressions are described below. The structural congruence for trees states that trees are unordered, and scripted processes are identified up to the structural congruence for processes (see Table 1).

2.1.2 Processes

Our processes are based on π-processes extended with an explicit migration primitive between locations, and an operation for interacting directly with data. The π-processes describe the movement of values between channels. Generic

Structural congruence is the least congruence satisfying alpha-conversion, the commutative monoidal laws for $(0, |)$ on trees, processes and networks, and the axioms reported below:

(Trees)

$U \equiv V \Rightarrow \mathsf{a}[\,U\,] \equiv \mathsf{a}[\,V\,]$

(Values)

$P \equiv Q \Rightarrow \Box P \equiv \Box Q$ $\qquad$ $v' \equiv w' \wedge \tilde{v} \equiv \tilde{w} \Rightarrow v', \tilde{v} \equiv w', \tilde{w}$

(Processes)

$(\nu c)0 \equiv 0$ $\qquad$ $c \notin fn(P) \Rightarrow P \,|\, (\nu c)Q \equiv (\nu c)(P \,|\, Q)$

$(\nu c)(\nu c')P \equiv (\nu c')(\nu c)P$ $\qquad$ $\tilde{v} \equiv \tilde{w} \Rightarrow \bar{c}\langle\tilde{v}\rangle \equiv \bar{c}\langle\tilde{w}\rangle$

$V \equiv V' \wedge P \equiv Q \Rightarrow \mathsf{update}_p(\chi, V).P \equiv \mathsf{update}_p(\chi, V').Q$

(Networks)

$(\nu c)0 \equiv 0$ $\qquad$ $c \notin fn(N) \Rightarrow N \,|\, (\nu c)M \equiv (\nu c)(N \,|\, M)$

$(\nu c)(\nu c')N \equiv (\nu c')(\nu c)N$ $\qquad$ $P \equiv Q \Rightarrow l\langle P\rangle \Rightarrow l\langle Q\rangle$

$l\left[\,T \,\|\, (\nu c)P\,\right] \equiv (\nu c)l\left[\,T \,\|\, P\,\right]$ $\qquad$ $T \equiv S \wedge P \equiv Q \Rightarrow l\left[\,T \,\|\, P\,\right] \equiv l\left[\,S \,\|\, Q\,\right]$

Table 1: Structural congruence for X$d\pi$.

variables are x, y, z, channel names or channel variables are a, b, c and values, ranged over by u, v, w, are

$$u ::= T \mid c \mid l \mid p \mid \Box P \mid x$$

We use the notation $\tilde{z}$ and $\tilde{v}$ to denote vectors of variables and values respectively. Identifiers U, V range over scripted processes, pointers and trees. The set of processes, denoted by $\mathcal{P}$, is given by

$$\begin{aligned} P ::= \quad & 0 \mid P \,|\, P \mid (\nu c)P \mid \overline{b}\langle\tilde{v}\rangle \mid b(\tilde{z}).P \mid !b(\tilde{z}).P \\ & \mid \mathsf{go}\, l.P \mid \mathsf{update}_p(\chi, V).P \mid \mathsf{run}_p \end{aligned}$$

The processes in the first line of the grammar are constructs arising from the π-calculus: the *nil* process 0, the *composition* of processes $P_1 \,|\, P_2$, the *restriction* $(\nu c)P$ which restricts (binds) channel name c in process P, the *output* process $\overline{b}\langle\tilde{v}\rangle$ which denotes a vector of values $\tilde{v}$ waiting to be sent via channel b, the *input* process $b(\tilde{z}).P$ which is waiting to receive values from an output process via channel b to replace the vector of distinct, bound variables $\tilde{z}$ in P, and *replicated input* $!b(\tilde{z}).P$ which spawns off a copy of $b(\tilde{z}).P$ each time one is requested. We assume a simple sorting discipline, to ensure that the number of values sent along a channel matches the number of variables expected to receive those values. Channel names are partitioned into *public* and *session* channels, denoted $\mathcal{C}_P$ and $\mathcal{C}_S$ respectively. Public channels denote those channels that

are intended to have the same meaning at each location, such as *finger*, and cannot be restricted. Session channels are used for process interaction, can be restricted, and cannot be free in the scripted processes occurring in data. We assume upon the latter the usual notions of *free* and *bound* names (fn, bn).

The *migration* primitive $\mathsf{go}\, l.P$ is common in calculi for describing distributed systems; see for example [16]. It enables a process to go to l and become P. An alternative choice would have been to incorporate the location information inside the other process commands: for example using $\overline{l \cdot b}\langle \tilde{v} \rangle$ to denote a process which goes to l and interacts via channel b.

The generic *update* command $\mathsf{update}_p(\chi, U).P$ is used to interact with the data trees. *Patterns* χ, ξ have the form

$$\chi ::= X \mid @x{:}y \mid \Box X,$$

where X denotes a tree or process variable. Here U can contain variables and must have the same sort as χ. The variables free in χ are bound in U and P. The update command finds all the values U_i given by the path p, pattern-matches these values with χ to obtain the substitution σ_i when it exists. For each successful pattern-matching, it replaces the U_i with $U\sigma_i$ and evolves to $P\sigma_i$. Simple commands can be derived from this update command, including standard copy_p, cut_p and paste_p commands. Command run_p prescribes, for all scripted processes $\Box P_i$ found at the end of path p, to run P_i in the workspace.

The structural congruence on processes is similar to that given for the π-calculus, and is given in Table 1. Notice that it depends on the structural congruence for trees, since trees can be passed as values.

2.1.3 Networks

We model networks as a composition of *unique* locations, where each location contains a tree and a set of processes. The set of networks, denoted $\mathcal{N}$, is given by

$$N ::= 0 \mid N \mid N \mid l\left[\, T \,\middle\|\, P \,\right] \mid (\nu c)N \mid l\langle P \rangle$$

The *location* $l\left[\, T \,\middle\|\, P \,\right]$ denotes location l containing tree T and process P. It is well-formed when the tree and process are closed (have no free variables), and the tree contains no free session channels. The network composition $N_1 \mid N_2$ is *partial* in that the location names associated with N_1 and N_2 must be disjoint. Communication between locations is modelled by process migration, which we represent explicitly: the process $l\langle P \rangle$ represents a (higher-order) migration message addressed to l and containing process P, which has left its originating location. In the introduction, we saw that a session channel can be shared between processes at different locations. We must therefore lift the restriction to the network level using $(\nu c)N$. Structural congruence for networks is defined in Table 1, and is analogous to that given for processes.

2.2 Semantics

2.2.1 Path Expressions

In the examples of this paper, we use a very simple subset of XPath expressions [20]. In our examples, "/" denotes path composition and ".", which can appear only inside trees, denotes the path to its enclosing node.

Our semantic model is robust with respect to any choice of mechanism which, given some expression p, identifies a set of nodes in a tree T (modulo structural congruence). We let $p(T)$ denote the tree T where the nodes identified by p are selected, and we represent a selected node by underlining its contents. For example the selected subtrees below are S' and T':

$$T = \mathtt{a}[\,\mathtt{a}[\,S\,] \,|\, \mathtt{b}[\underline{\,S'\,}] \,|\, \mathtt{c}[\underline{\,T'\,}]\,]$$

A path expression such as $*/\mathtt{a}$ might select nested subtrees. We give an example:

$$*/\mathtt{a}(T) = \underline{\mathtt{a}[\,\underline{\mathtt{a}[\,S\,]} \,|\, \mathtt{b}[\,S'\,] \,|\, \mathtt{c}[\,T'\,]\,]}$$

2.2.2 Reduction Semantics

The reduction relation $\searrow$ describes the movement of processes across locations, the interaction between processes and processes, and the interaction between processes and data. Reduction is closed under network composition, restriction and structural congruence, and it relies on the updating function $\rightsquigarrow$ given in Table 2, which we will explain later. The reduction relation is given in Table 3.

The rules for process movement between locations are inspired by [5]. Rule (Exit) always allows a process $\mathtt{go}\, l.P$ to leave its enclosing location. At the moment, rule (Enter) permits the code of P to be installed at l provided that location l exists[2]. In future work, we intend to associate some security check to this operation. Process interaction (rules (Com) and (!Com)) is inspired by π-calculus interaction. If one process wishes to output on a channel, and another wishes to input on the same channel, then they can react together and transmit some values as part of that reaction.

The generic (Update) rule provides interaction between processes and data. Using path p it selects for update some subtrees in T, denoted by $p(T)$, and then applies the updating function $\rightsquigarrow$ to $p(T)$ in order to obtain the new tree T' and a set of substitutions Σ, used to generate the continuation processes. Given a subtree selected by p, the function $\rightsquigarrow$ pattern matches the subtree with pattern χ to obtain substitution σ (when it exists) and updates the subtree with $V\sigma$. The formal definition[3] of $\rightsquigarrow$, parameterised by $\Theta = p, l, \chi, V$, is given in Table 2.

Rule (Up) in Table 3 deserves some explanation. First of all it substitutes in U any reference to the current location and path with the actual values l and p, denoted $U\{l/\circlearrowleft, p/.\}$. It then matches the result with χ, to obtain substitution

[2]This feature is peculiar to our calculus, as opposed to e.g. [16], where the existence of the location to be entered is not a precondition to migration. Our choice makes the study of equivalences non-trivial and models an important aspect of *location failure*.

[3]Where Σ is a multiset of substitutions and $\oplus$ is multiset union.

$$
\begin{array}{lll}
\text{(Zero)} & 0 \rightsquigarrow_\Theta 0, \emptyset & \text{(Link)} \quad @m{:}q \rightsquigarrow_\Theta @m{:}q, \emptyset \\
\text{(Proc)} & \Box Q \rightsquigarrow_\Theta \Box Q, \emptyset & \text{(Node)} \quad \dfrac{U \rightsquigarrow_\Theta U', \Sigma}{\mathsf{a}[\,U\,] \rightsquigarrow_\Theta \mathsf{a}[\,U'\,], \Sigma} \\
\text{(Par)} & \dfrac{T \rightsquigarrow_\Theta T', \Sigma_1 \qquad S \rightsquigarrow_\Theta S', \Sigma_2}{T \,|\, S \rightsquigarrow_\Theta T' \,|\, S', \Sigma_1 \oplus \Sigma_2} & \\
\text{(Up)} & \dfrac{\mathsf{match}(U\{l/\circlearrowleft, p/.\}, \chi) = \sigma \qquad V\sigma \rightsquigarrow_\Theta V', \Sigma \qquad \Theta = p, l, \chi, V}{\mathsf{a}[\underline{U}\,] \rightsquigarrow_\Theta \mathsf{a}[\,V'\,], \{\sigma\} \oplus \Sigma} &
\end{array}
$$

Table 2: Update Semantics

Reduction Axioms

$$
\begin{array}{ll}
\text{(Exit)} & m\left[\,T \,\middle\|\, Q \mid \mathsf{go}\, l.P\,\right] \;\searrow\; m\left[\,T \,\middle\|\, Q\,\right] \mid l\langle P\rangle \\
\text{(Enter)} & l\left[\,T \,\middle\|\, Q\,\right] \mid l\langle P\rangle \;\searrow\; l\left[\,T \,\middle\|\, Q \mid P\,\right] \\
\text{(Com)} & l\left[\,T \,\middle\|\, \overline{c}\langle\tilde{v}\rangle \mid c(\tilde{z}).P \mid Q\,\right] \;\searrow\; l\left[\,T \,\middle\|\, P\{\tilde{v}/\tilde{z}\} \mid Q\,\right] \\
\text{(Com!)} & l\left[\,T \,\middle\|\, \overline{c}\langle\tilde{v}\rangle \mid\, !c(\tilde{z}).P \mid Q\,\right] \;\searrow\; l\left[\,T \,\middle\|\, !c(\tilde{z}).P \mid P\{\tilde{v}/\tilde{x}\} \mid Q\,\right] \\
\text{(Update)} & \dfrac{p(T) \rightsquigarrow_{p,l,\chi,V} T', \{\sigma_1, \cdots, \sigma_n\}}{l\left[\,T \,\middle\|\, \mathsf{update}_p(\chi, V).P \mid Q\,\right] \searrow l\left[\,T' \,\middle\|\, P\sigma_1 \mid \cdots \mid P\sigma_n \mid Q\,\right]} \\
\text{(Run)} & \dfrac{p(T) \rightsquigarrow_{p,l,\Box X,\Box X} T, \{\{\Box P_1/\Box X\}, \cdots, \{\Box P_n/\Box X\}\}}{l\left[\,T \,\middle\|\, \mathsf{run}_p \mid Q\,\right] \searrow l\left[\,T \,\middle\|\, P_1 \mid \cdots \mid P_n \mid Q\,\right]}
\end{array}
$$

Contextual Rules

$$
\begin{array}{ll}
\text{(Res)} & \dfrac{N \searrow N'}{(\nu\tilde{c})N \searrow (\nu\tilde{c})N'} \\
\text{(Par)} & \dfrac{N \searrow N'}{N \mid M \searrow N' \mid M} \\
\text{(Struct)} & \dfrac{N \equiv M \searrow M' \equiv N'}{N \searrow N'}
\end{array}
$$

Table 3: Reduction Semantics

σ; when σ exists, it continues updating $V\sigma$, and when we obtain some subtree V' along with a process R, it replaces U with V' and it returns $P\sigma$ in parallel

with R.

The ability to select nested nodes introduces a difference between updating the tree in a top-down rather than bottom-up order. In particular the resulting tree is the same, but a different set of processes P is collected. We chose the top-down approach because is bears a closer correspondence with intuition: a copy of P will be created for each update still visible in the final tree outcome. For example,

$$l\left[\,\mathsf{a}[\,\mathsf{b}[\,T\,]\,]\;\middle\|\;\mathsf{update}_*(X,0).P\,\right]\searrow l\left[\,\mathsf{a}[\,0\,]\;\middle\|\;P\{\mathsf{b}[\,T\,]/X\}\,\right]$$

whereas, following a bottom-up approach we would have

$$l\left[\,\mathsf{a}[\,\mathsf{b}[\,T\,]\,]\;\middle\|\;\mathsf{update}_*(X,0).P\,\right]$$
$$\searrow l\left[\,\mathsf{a}[\,0\,]\;\middle\|\;P\{\mathsf{b}[\,0\,]/X\}\,|\,P\{T/X\}\,\right]$$

because first we would transform $\mathsf{a}[\,\mathsf{b}[\,T\,]\,]$ in $\mathsf{a}[\,\mathsf{b}[\,0\,]\,]$ generating $P\{T/X\}$, and then transform $\mathsf{a}[\,\mathsf{b}[\,0\,]\,]$ in $\mathsf{a}[\,0\,]$, generating $P\{\mathsf{b}[\,0\,]/X\}$.

2.2.3 Derived Commands

Throughout our examples, we use the derived commands given in Table 4. When concerned with access control issues, it may become desirable to include each of these derived forms as primitives, in order to use fine-grained security policies. Nonetheless it is important to be aware that semantically are all expressible through a single construct.

$\mathsf{copy}_p(X).P$	$\triangleq$	$\mathsf{update}_p(X,X).P$	copy the tree at p and use it in P
$\mathsf{read}_p(@x{:}y).P$	$\triangleq$	$\mathsf{update}_p(@x{:}y,@x{:}y).P$	read the pointer at p, use its location and path in P
$\mathsf{cut}_p(X).P$	$\triangleq$	$\mathsf{update}_p(X,0).P$	cut the tree at p and use it in p
$\mathsf{paste}_p\langle T\rangle.P$	$\triangleq$	$\mathsf{update}_p(X,X\,\vert\,T).P$	where X is not free in T or P, paste tree T at p and evolve to P

Table 4: Derived Commands

Example 2.1 *The following reaction illustrates the* cut *command:*

$$l\left[\,\mathsf{c}[\,\mathsf{a}[\,T\,]\,|\,\mathsf{a}[\,T'\,]\,|\,\mathsf{b}[\,S\,]\,]\;\middle\|\;\mathsf{cut}_{\mathsf{c}/\mathsf{a}}(X).P\,\right]$$
$$\searrow l\left[\,\mathsf{c}[\,\mathsf{a}[\,0\,]\,|\,\mathsf{a}[\,0\,]\,|\,\mathsf{b}[\,S\,]\,]\;\middle\|\;P\{T/X\}\,|\,P\{T'/X\}\,\right]$$

The cut *operation cuts the two subtrees T and T' identified by the path expression* c/a *and spawns one copy of P for each subtree.*

Now we give an example to illustrate run *and the substitution of local references:*

$$S=\mathsf{a}[\,\mathsf{b}[\,\square\mathsf{go}\,m.\mathsf{go}\;\circlearrowleft.Q\,]\,|\,\mathsf{b}[\,\square\mathsf{update}_{./../\mathsf{c}}(\chi,T).P\,]\,]$$

$$l\left[\, S \,\|\, \mathsf{run}_{\mathtt{a/b}} \,\right]$$

$$\searrow l\left[\, S \,\|\, \mathsf{go}\, m.\mathsf{go}\, l.Q \mid \mathsf{update}_{\mathtt{a/b/../c}}(\chi, T).P \,\right]$$

The data S is not affected by the run *operation, which has the effect of spawning the two processes found by path* a/b*. Note how the local path ./../*c *has been resolved into the completed path* a/b/../c*, and $\circlearrowleft$ has been substituted by l.*

2.3 Dynamic web data at work

We give some examples of dynamic web data modelled in X$d\pi$.

2.3.1 Web Services

In the introduction, we described the hyperlink example. Here we generalise this example to arbitrary web services. We define a web service c with parameters $\tilde{z}$, body P, and type of result specified by the distinct variables $\tilde{w}$ bound by P:

$$\mathsf{Define}\ c(\tilde{z})\ \mathsf{as}\ P\ \mathsf{output}\ \langle \tilde{w} \rangle \triangleq !c(\tilde{z}, l, x).\, P.\, \mathsf{go}\, l.\, \overline{x}\langle \tilde{w} \rangle$$

where l and x are fresh variables not occurring in P or $\tilde{w}$, which are used to specify the return location and channel. For example, process R described in the introduction can be written $\mathsf{Define}\ get(q)\ \mathsf{as}\ \mathsf{copy}_q(X)\ \mathsf{output}\ \langle X \rangle$.

We specify a service call at l to the service c at m, sending actual parameters $\tilde{v}$ and expecting in return the result specified by distinct bound variables $\tilde{w}$:

$$l{\cdot}\mathsf{Call}\ m{\cdot}c\langle \tilde{v} \rangle\ \mathsf{return}\ (\tilde{w}).Q \triangleq (\nu b)(\mathsf{go}\, m.\overline{c}\langle \tilde{v}, l, b \rangle \mid b(\tilde{w}).Q)$$

This process establishes a private session channel b, which it passes to the web service as the unique return channel. Returning to the hyperlink example, the process Q can be given by

$$!load(m, q, p).\, l{\cdot}\mathsf{Call}\ m{\cdot}get\langle q \rangle\ \mathsf{return}\ (X).\mathsf{paste}_q\langle X \rangle$$

Notice that it is easy to model subscription to continuous services in our model, by simply replicating the input on the session channel:

$$l{\cdot}\mathsf{Subscribe}\ m{\cdot}c\langle \tilde{v} \rangle\ \mathsf{return}\ (\tilde{w}).P \triangleq (\nu b)(\mathsf{go}\, m.\overline{c}\langle \tilde{v}, l, b \rangle \mid !b(\tilde{w}).P)$$

Note that some web services may take as a parameter or return as a result some data containing another service call (for example, see the *intensional parameters* of [1]). In our system the choice of when to invoke such nested services is completely open, and is left to the service designer.

2.3.2 XLink Base

We look at a refined example of the use of linking, along the lines of XLink. Links specify both of their endpoints, and therefore can be stored in some external repository, for example

$$\mathtt{XLink}[\,\mathtt{To}[\,@n{:}q\,] \mid \mathtt{From}[\,@l{:}p\,] \mid \mathtt{Code}[\,\Box P\,]\,]$$

$$\texttt{XLinkBase}[\,\texttt{XLink}[\,...\,]\,|\,...\,|\,\texttt{XLink}[\,...\,]\,]$$

Suppose that we want to download from an XLink server the links associated with node p in the local repository at l.

We can define a function $xload$ which takes a parameter p and requests from the XLink service xls at m all the XLinks originating from $@l{:}p$, in order to paste them under p at location l:

$$!xload(p).l{\cdot}\mathsf{Subscribe}\ m{\cdot}xls\langle l,p\rangle\ \mathsf{return}\ (x,y,\Box\chi)$$
$$.\mathsf{paste}_p\langle\texttt{Link}[\,\texttt{To}[\,@x{:}y\,]\,|\,\texttt{Code}[\,\Box\chi\,]\,]\rangle$$

Service xls defined below is the XLink server. It takes as parameters the two components l, p making up the `From` endpoint of a link, and returns all the pairs `To`, `Code` defined in the database for $@l{:}p$.

$$\mathsf{Define}\ xls(l,p)\ \mathsf{as}\ P\ \mathsf{output}\ \langle x,y,\Box\chi\rangle$$

$$P = \mathsf{copy}_{p_1}(@x{:}y).\mathsf{copy}_{p_2}(\Box\chi)$$
$$p_1 = \texttt{XLinkBase/XLink}[\,\texttt{From}[\,@l{:}p\,]\,]\texttt{/To}$$
$$p_2 = \texttt{XLinkBase/XLink}[\,\texttt{From}[\,@l{:}p\,]\,|\,\texttt{To}[\,@x{:}y\,]\,]\texttt{/Code}$$

In p_1 we use the XPath syntax $\texttt{XLink}[\,\texttt{From}[\,@l{:}p\,]\,]\texttt{/To}$ to identify the node `To` which is a son of node `XLink` and a sibling of $\texttt{From}[\,@l{:}p\,]$; similarly for p_2.

2.3.3 Forms

Forms enhance documents with the ability to input data from a user and then send it to a server for processing. For example, assuming that the server is at location s, that the form is at path p, and that the code to process the form result is called *handler*, we have

$$\begin{aligned}&\texttt{form}[\texttt{input}[\,0\,]\\&|\,\texttt{submit}[\,\Box\mathsf{copy}_{\texttt{./../input}}(X).\mathsf{go}\ s.\overline{handler}\langle X\rangle\,]\\&|\,\texttt{reset}[\,\Box\mathsf{cut}_{\texttt{./../input}}(X)\,]]\end{aligned}$$

where $\mathsf{run}_{p/\texttt{form/submit}}$ (or $\mathsf{run}_{p/\texttt{form/reset}}$) is the event generated by clicking on the submit (or reset) button. Some user input T can be provided by a process

$$\mathsf{paste}_{p/\texttt{form/input}}\langle T\rangle$$

and on the server there will be a handler ready to deal with the received data

$$s\left[\,S\,\middle\|\,!handler(X).P|...\,\right]$$

This example is suggestive of the usefulness of embedding processes rather than just service calls in a document: the code to handle submission may vary from form to form, and for example some input validation could be performed on the client side.

3 Behaviour of dynamic web data

In the hyperlink example of the introduction, we have stated that process Q and its specification Q_s have the same intended behaviour. In this section we provide the formal analysis to justify this claim. We do this in several stages. First, we define what it means for two X$d\pi$-networks to be equivalent. Second, we give a translation of X$d\pi$ into a simpler calculus (Xπ_2), where the location structure has been pushed inside the data and processes. This translation technique, first proposed in [9], enables us to separate reasoning about processes from reasoning about data and networks. Third, we define process equivalence for Xπ_2, and provide a proof method based on a labelled bisimulation. Finally, we study the examples: we show that Q behaves like Q_s and we prove that services can be transparently replicated.

3.1 Network Equivalence

We apply a standard technique for reasoning about processes distributed between locations to our non-standard setting. The network contexts are

$$C ::= - \mid C \,|\, N \mid (\nu c)\, C$$

We define a *barbed congruence* between networks which is reduction-closed, closed with respect to network contexts, and which satisfies an additional *observation relation* described using *barbs*. In our case, the barbs describe the update commands, the commands which directly affect the data.

Definition 3.1 *A* barb *has the form $l{\cdot}\beta$, where l is a location name and $\beta \in \{\mathsf{update}_p\}_{p\in\mathcal{E}}$. The* observation relation*, denoted by $N \downarrow_{l\cdot\beta}$, is a binary relation between X$d\pi$-networks and barbs defined by*

$$N \downarrow_{l\cdot \mathsf{update}_p} \triangleq \exists C, S, \chi, U, P, Q.\ N \equiv C[l\,[\, S \,\|\, \mathsf{update}_p(\chi, U).P \,|\, Q\,]]$$

that is, N contains a location l with an update_p command. The weak observation relation*, denoted $N \Downarrow_{l\cdot\beta}$, is defined by*

$$N \Downarrow_{l\cdot\beta} \quad \triangleq \quad \exists N'.\, N \searrow N' \wedge N' \downarrow_{l\cdot\beta}$$

Observing a barb corresponds to observe at what points in some data-tree a process has the capability to read or write data.

Definition 3.2 *Barbed congruence ($\simeq_b$) is the largest symmetric relation $\mathcal{R}$ on X$d\pi$-networks such that $N\,\mathcal{R}\,M$ implies*

- *N and M have the same barbs: $N \downarrow_{l\cdot\beta} \Rightarrow M \Downarrow_{l\cdot\beta}$;*
- *R is reduction-closed: $N \searrow N' \Rightarrow (\exists M'.M \searrow^* M' \ \wedge \ N'\,\mathcal{R}\ M')$;*
- *R is closed under network contexts: $\forall C.C[N]\,\mathcal{R}\,C[M]$.*

Example 3.1 *Our first example illustrates that network equivalence does not imply that the initial data trees need to be equivalent:*

$$N = l\left[\,\mathsf{b}[\,0\,] \parallel !\mathsf{paste}_{\mathsf{b}}\langle \mathsf{a}[\,0\,]\rangle \mid !\mathsf{cut}_{\mathsf{b}}(X)\,\right]$$

$$M = l\left[\,\mathsf{b}[\,\mathsf{a}[\,0\,] \mid \mathsf{a}[\,0\,]\,] \parallel !\mathsf{paste}_{\mathsf{b}}\langle \mathsf{a}[\,0\,]\rangle \mid !\mathsf{cut}_{\mathsf{b}}(X)\,\right]$$

We have $N \simeq_b M$ since each state reachable by one network is also reachable by the other, and vice versa.

An interesting example of non-equivalence is

$$l\left[\,T \parallel \mathit{update}_p(X,X).0\,\right] \not\simeq_b l\left[\,T \parallel 0\,\right]$$

Despite this particular update (copy) command having no effect on the data and continuation, we currently regard it as observable since it has the capability to modify the data at p, even if it does not use it.

Example 3.2 *Our definition of web service is equivalent to its specification. Consider the simple networks*

$$N = l\left[\,T \parallel l{\cdot}\mathsf{Call}\ m{\cdot}c\langle\tilde{v}\rangle\ \mathsf{return}\ (\tilde{w}).Q|R\,\right]$$

$$N_s = l\left[\,T \parallel \mathsf{go}\ m.P\{\tilde{v}/\tilde{z}\}.\mathsf{go}\ l.Q|R\,\right]$$

$$M = m\left[\,S \parallel \mathsf{Define}\ c(\tilde{z})\ \mathsf{as}\ P\ \mathsf{output}\ \langle\tilde{w}\rangle \mid R'\,\right]$$

If c does not appear free in R and R', then

$$(\nu c)(N \mid M) \simeq_b (\nu c)(N_s \mid M)$$

A special case of this example is the hyperlink example discussed in the introduction. The restriction c is used to prevent the context providing any competing service on c. It is clearly not always appropriate however to make a service name private. An alternative approach is to ensure service uniqueness, introducing a linear type system as for example in [6], or requiring unique receptors for each channel, as for example in [12].

3.2 Separation of Data and Processes

Our aim is to define when two processes are equivalent in such a way that, when the processes are put in the same position in a network, the resulting networks are equivalent. In order to do this, we introduce the $X\pi_2$-calculus, in which the location structure is pushed locally to the data and processes, in order to be able to talk directly about processes. We translate the $Xd\pi$-calculus in the $X\pi_2$-calculus, and equate $Xd\pi$-equivalence with Xπ_2-equivalence.

(Trees)	$T ::= 0 \mid T \vert T \mid \mathsf{a}[\,T\,] \mid \mathsf{a}[\,\Box^{\circlearrowleft}\!\!\frown P\,] \mid \mathsf{a}[\,@l{:}p\,]$
(Processes)	$P ::= \quad l{\cdot}0 \mid P \,\vert\, P \mid \overline{l{\cdot}b}\langle\tilde{v}\rangle \mid l{\cdot}b(\tilde{z}).P \mid !l{\cdot}b(\tilde{z}).P \mid (\nu c)P$ $\mid l{\cdot}\mathsf{go}\, m.P \mid l{\cdot}\mathsf{update}_p(\chi, V).P \mid l{\cdot}\mathsf{run}_p \mid l\langle P\rangle \mid 0$
(Stores)	$D ::= \emptyset \mid \{l \mapsto T\} \uplus D$
(Networks)	(D, P)

Table 5: The calculus Xπ_2.

3.2.1 The Xπ_2 calculus

In Table 5 we give the syntax of Xπ_2. Trees are defined as in X$d\pi$, apart from scripted processes being located at $\circlearrowleft$. Networks are represented by pairs where the first component (the *store*) contains a map from location names to data trees, and the second component is a parallel composition of located processes. We use notation $^{l}\!\frown P$ to express that P is a parallel composition of processes located at l. The set $loc(P)$ denotes all l_i such that $P \equiv (\nu\tilde{b})(^{l_1}\!\frown P_1 \mid \cdots \mid {}^{l_n}\!\frown P_n)$.

Processes deserve more explanation: parallel composition, restriction and the migration message are as usual; $l{\cdot}0$ and $\overline{l{\cdot}b}\langle\tilde{v}\rangle$ represent respectively a null process and an output process located at l — the input, replicated input, migration and update are similar, and 0 stands for the null location. Well-formedness, defined formally in Table 9, requires that $loc(P) = dom(D)$ for (D, P), that the migration message cannot be prefixed by any other process, that the continuation of an explicitly located process be located at the same location, except for the case of $\mathsf{go}\ m.P$, where P must be located at m. Additionally, well-formedness requires the same conditions given for X$d\pi$[4].

Structural congruence for trees, stores, networks and processes is defined in Table 6. Note how $l{\cdot}0 \,|\, ^{l}\!\frown P \equiv ^{l}\!\frown P$, whereas for example $l{\cdot}0 \not\equiv 0$: this choice guarantees that starting from a well-formed network (D, P) there is always at least one process (possibly null) located at l, for each $l \in dom(D)$, and there is never a process located at m for $m \notin dom(D)$. In Table 7 below, we give the semantic rules for Xπ_2. The rules correspond closely with those for X$d\pi$. Reduction is closed under restriction, parallel composition and structural congruence (rules (Res), (Par) and (Struct)).

3.2.2 Barbed congruence for Xπ_2

Definition 3.3 *Reduction contexts for processes are* $C' ::= - \mid C' \,|\, P \mid (\nu c)C'$, *and reduction contexts for networks are* $C ::= (B \uplus -, C')$, *where* $C[(D, P)]$ *is* $(B \uplus D, C'[P])$.

[4] In practice we are going to encode X$d\pi$ terms in Xπ_2 terms which will be well-formed by construction.

Structural congruence is the least congruence satisfying alpha-conversion, the commutative monoidal laws for $(0, |)$ on trees, processes and networks, and the axioms reported below:

(Trees)	$U \equiv U' \Rightarrow \mathsf{a}[\,U\,] \equiv \mathsf{a}[\,U'\,]$
(Values)	$v' \equiv w' \wedge \tilde{v} \equiv \tilde{w} \Rightarrow v', \tilde{v} \equiv w', \tilde{w} \quad P \equiv Q \Rightarrow \Box P \equiv \Box Q$
(Processes)	$(\nu c)0 \equiv 0 \quad l{\cdot}0 \mid^{l\curvearrowright} P \equiv^{l\curvearrowright} P \qquad (\nu c) l{\cdot}0 \equiv l{\cdot}0$
	$c \notin fn(P) \Rightarrow P \mid (\nu c)Q \equiv (\nu c)(P \mid Q) \quad (\nu c)(\nu c')P \equiv (\nu c')(\nu c)P$
	$V \equiv V' \wedge P \equiv Q \Rightarrow l{\cdot}\mathsf{update}_p(\chi, V).P \equiv l{\cdot}\mathsf{update}_p(\chi, V').Q$
(Stores)	$dom(D) = dom(B) \wedge (\forall l \in dom(D).D(l) \equiv B(l)) \Rightarrow D \equiv B$
(Networks)	$D \equiv B \wedge P \equiv Q \Rightarrow (D, P) \equiv (B, Q)$
(Abstractions)	$V \equiv V' \Longrightarrow (\chi)V \equiv (\chi)V'$

Table 6: Structural congruence for Xπ_2.

Note that $X\pi_2$ has in some sense *more* reduction contexts than $Xd\pi$. In particular, according to the definition above, we can insert a new process in a location which is already defined, whereas in $Xd\pi$ that is not allowed. Nonetheless when reasoning about weak equivalences this difference is no longer a limitation, because we can add in $Xd\pi$ a process message to install a new process in an existing location, preserving weak bisimilarity.

Barbed congruence for Xπ_2 is analogous to that for X$d\pi$.

Definition 3.4 *Barbs for networks are defined as*

$$\begin{array}{rcl} (D, P) \downarrow_{l\cdot\mathsf{update}_p} & \triangleq & \exists\, C', \chi, U, P'.\ P \equiv C'[l{\cdot}\mathsf{update}_p(\chi, U).P'] \\ (D, P) \Downarrow_{l\cdot\beta} & \triangleq & (D, P) \rightarrow^* (D', P') \wedge (D', P') \downarrow_{l\cdot\beta} \end{array}$$

Definition 3.5 *Barbed congruence ($\cong_b$) is the largest symmetric relation $\mathcal{R}$ on $Xa\pi_2$ networks such that $(D, P)\,\mathcal{R}\,(B, Q)$ implies:*

- $(D, P) \downarrow_{l\cdot\beta} \Rightarrow (B, Q) \Downarrow_{l\cdot\beta}$;
- $(D, P) \rightarrow (D', P') \Rightarrow (\exists B', Q'.(B, Q) \rightarrow^* (B', Q') \wedge (D', P')\,\mathcal{R}\,(B', Q'))$;
- $\forall\ C.C[(D, P)]\,\mathcal{R}\,C[(B, Q)]$.

3.2.3 From X$d\pi$ to Xπ_2

Recall the hyperlink example:

$$\begin{array}{rcl} N & = & l\left[\,\mathtt{Link}[\,\mathtt{To}[\,@m{:}q\,] \mid \mathtt{Code}[\,\Box P\,]\,]\,\middle\|\, Q\,\right] \mid m\left[\,S\,\middle\|\, R\,\right], \text{ where} \\ Q & = & !load(m, q, p).(\nu c)(\mathsf{go}\, m.\,\overline{get}\langle q, l, c\rangle \mid c(X).\mathsf{paste}_p\langle X\rangle) \\ R & = & !get(q, l, c).\mathsf{copy}_q(X).\mathsf{go}\, l.\bar{c}\langle X\rangle \end{array}$$

Reduction Axioms

(Exit) $(\emptyset, m{\cdot}\mathsf{go}\ l.P) \ \rightarrow (\emptyset, m{\cdot}0 \mid l\langle P\rangle)$

(Enter) $(\{l \mapsto T\}, l\langle P\rangle) \ \rightarrow (\{l \mapsto T\}, P)$

(Com) $(\emptyset, \overline{l{\cdot}c}\langle\tilde{v}\rangle \mid l{\cdot}c(\tilde{x}).P) \rightarrow \ (\emptyset, P\{\tilde{v}/\tilde{x}\})$

(!Com) $(\emptyset, \overline{l{\cdot}c}\langle\tilde{v}\rangle \mid !l{\cdot}c(\tilde{x}).P) \rightarrow \ (\emptyset, !l{\cdot}c(\tilde{x}).P | P\{\tilde{v}/\tilde{x}\})$

(Update) $$\frac{p(T) \rightsquigarrow_{p,l,\chi,V} T', \{\sigma_1, \cdots, \sigma_n\}}{(\{l \mapsto T\}, l{\cdot}\mathsf{update}_p(\chi, V).P) \rightarrow \ (\{l \mapsto T'\}, P\sigma_1 \mid \cdots \mid P\sigma_n)}$$

(Run) $$\frac{p(T) \rightsquigarrow_{p,l,\Box X,\Box X} T, \{\{\Box P_1/\Box X\}, \cdots, \{\Box P_n/\Box X\}\}}{(\{l \mapsto T\}, l{\cdot}\mathsf{run}_p) \rightarrow \ (\{l \mapsto T\}, P_1 \mid \cdots \mid P_n)}$$

(The function $\rightsquigarrow$ is analogous to the one defined in Table 3.)

Structural Rules

(Res) $$\frac{(D, P) \rightarrow (D', P')}{(D, (\nu c)P) \rightarrow (D', (\nu c)P')}$$

(Par) $$\frac{(D, P) \rightarrow (D', P')}{(D \uplus E, P|R) \rightarrow (D' \uplus E, P'|R)}$$

(Struct) $$\frac{(D, P) \equiv (B, Q) \rightarrow (B', Q') \equiv (D', P')}{(D, P) \rightarrow (D', P')}$$

Table 7: Reduction rules for Xπ_2.

The translation to Xπ_2 involves pushing the location structure, in this case the l and m, inside the data and processes. We use $(\!|N|\!)$ to denote the translated data and $\llbracket S \rrbracket$ to denote the translation of tree S; also $[\![N]\!]$ for the translated processes and $\langle\!|P|\!\rangle_l$ for the translation of process P which depends on location l. Our hyperlink example becomes

$$\begin{aligned}(\!|N|\!) &= \{l \mapsto \mathtt{Link}[\,\mathtt{To}[\,@m{:}q\,] \mid \mathtt{Code}[\,\Box\langle\!|P|\!\rangle_{\circlearrowleft}\,]\,], m \mapsto (\!|S|\!)\} \\ [\![N]\!] &= !l{\cdot}load(m,q,p).(\nu c)(l{\cdot}\mathsf{go}\, m.\, \overline{m{\cdot}get}\langle q,l,c\rangle \mid l{\cdot}c(X).\mathsf{paste}_p\langle X\rangle \mid \\ &\quad\ !m{\cdot}get(q,l,c).m{\cdot}\mathsf{copy}_q(X).m{\cdot}\mathsf{go}\, l.\overline{l{\cdot}c}\langle X\rangle\end{aligned}$$

There are several points to notice. The data translation $\llbracket\_\rrbracket$ assigns locations to translated trees, which remain the same except that the scripted processes are translated using the self location $\circlearrowleft$: in our example $\Box P$ is translated to $\Box\langle\!|P|\!\rangle_{\circlearrowleft}$. The use of $\circlearrowleft$ is necessary since it is not pre-determined where the scripted process will run. In our hyperlink example, it runs at l. With an HTML form, for example, it is not known where a form with an embedded scripted process will be required. The process translation $[\![\_]\!]$ embeds locations

in processes. In our example, it embeds location l in Q and location m in R. After a migration command, for example the $\mathsf{go}\, m.\_$ in Q, the location information changes to m, following the intuition that the continuation will be active at location m. The encodings are formally defined in Table 10.

3.2.4 Properties of the translation

The crucial properties of the encoding are that it preserves the observation relation and is fully abstract with respect to barbed congruence. The proofs are in Appendix A.1

Lemma 3.1 *$N \downarrow_{l\cdot\beta}$ if and only if $(([\![N]\!]), [\![N]\!]) \downarrow_{l\cdot\beta}$.*

Theorem 3.1 *$N \simeq_b M$ if and only if $(([\![N]\!]), [\![N]\!]) \simeq_b (([\![M]\!]), [\![M]\!])$.*

3.3 Process Equivalence

We now have the machinery to define process equivalence. We use the notation (D, P) to denote a network in $X\pi_2$, where D stands for located trees and P for located processes. A network (D, P) is well formed if and only if $(D, P) = (([\![N]\!]), [\![N]\!])$ for some X$d\pi$ network N.

3.3.1 Process Equivalence

Definition 3.6 *Processes P and Q are* barbed equivalent*, denoted $P \sim_b Q$, if and only if, for all D such that (D, P) is well formed, $(D, P) \simeq_b (D, Q)$.*

Example 3.3 *Recall the web service example in Example 3.2. The processes below are barbed equivalent:*

$$Q_1 = \langle\!|l\cdot\mathsf{Call}\ m\cdot c\langle\tilde{v}\rangle\ \mathsf{return}\ (\tilde{w}).Q|\!\rangle_l \quad Q_2 = \langle\!|\mathsf{go}\ m.P\{\tilde{v}/\tilde{z}\}.\mathsf{go}\ l.Q|\!\rangle_l$$

$$P_0 = \langle\!|\mathsf{Define}\ c(\tilde{z})\ \mathsf{as}\ P\ \mathsf{output}\ \langle\tilde{w}\rangle|\!\rangle_m$$

$$(\nu c)(Q_1 \mid P_0) \sim_b (\nu c)(Q_2 \mid P_0)$$

These equivalences are known to be difficult to prove. Below we provide a method for proving such equivalences, by extending the labelled transition system of the π-calculus and defining a location-sensitive bisimulation relation on processes. In particular, we develop a bisimulation equivalence $\approx$ with the property that, given two Xπ_2 processes P, Q, then $P \approx Q$ implies $P \sim_b Q$.

3.3.2 Labelled transition system

We define in Table 8 an *early* labelled transition system for $X\pi_2$ processes. Labels α are:

$$\alpha ::= \overline{l\cdot c}\langle\tilde{v}\rangle \mid (\tilde{b})\overline{l\cdot c}\langle\tilde{v}\rangle \mid l\cdot c(\tilde{v}) \mid l\cdot\mathsf{update}_p(\mathcal{U}, (\chi)V) \mid l\cdot\mathsf{run}_p \mid \tau$$

(Exit) $m{\cdot}\mathsf{go}\ l.P \xrightarrow{\tau} m{\cdot}0 \mid l\langle P\rangle$ (Enter) $l\langle P\rangle \xrightarrow{\tau} P$

(Res) $\dfrac{P \xrightarrow{\alpha} P'}{(\nu c)P \xrightarrow{\alpha} (\nu c)P'}\ c\notin n(\alpha)$ (Par) $\dfrac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}\ bn(\alpha)\cap fn(Q)=\emptyset$

(Struct) $\dfrac{P \equiv Q \xrightarrow{\alpha} Q' \equiv P'}{P \xrightarrow{\alpha} P'}$ (Open) $\dfrac{P \xrightarrow{(\tilde{a})\overline{l{\cdot}c}\langle\tilde{v}\rangle} P'}{(\nu b)P \xrightarrow{(b,\tilde{a})\overline{l{\cdot}c}\langle\tilde{v}\rangle} P'}\ b\neq c, b\in fn(\tilde{v})$

(In!) $!l{\cdot}c(\tilde{x}).P \xrightarrow{l{\cdot}c(\tilde{v})} !l{\cdot}c(\tilde{x}).P \mid P\{\tilde{v}/\tilde{x}\}$ (In) $l{\cdot}c(\tilde{x}).P \xrightarrow{l{\cdot}c(\tilde{v})} P\{\tilde{v}/\tilde{x}\}$

(Com) $\overline{l{\cdot}c}\langle\tilde{v}\rangle \mid l{\cdot}c(\tilde{x}).P \xrightarrow{\tau} P\{\tilde{v}/\tilde{x}\}$ (Out) $\overline{l{\cdot}c}\langle\tilde{v}\rangle \xrightarrow{\overline{l{\cdot}c}\langle\tilde{v}\rangle} l{\cdot}0$

(Com!) $\overline{l{\cdot}c}\langle\tilde{v}\rangle \mid !l{\cdot}c(\tilde{x}).P \xrightarrow{\tau} !l{\cdot}c(\tilde{x}).P \mid P\{\tilde{v}/\tilde{x}\}$

(Update) $l{\cdot}\mathsf{update}_p(\chi,V).P \xrightarrow{l{\cdot}\mathsf{update}_p(\mathcal{U},(\chi)V)} P\{U_1/\chi\} \mid \cdots \mid P\{U_n/\chi\}$ where $\mathcal{U}=\{U_1,\cdots,U_n\}$

(Run) $l{\cdot}\mathsf{run}_p \xrightarrow{l{\cdot}\mathsf{run}_p} l{\cdot}0$ (Zero) $0 \xrightarrow{\tau} l{\cdot}0$

Table 8: Labelled Transition System

Note that the label for update contains an *abstraction* $(\chi)V$ meaning that the variables in χ are bound in V. We use the standard notation $\overset{\alpha}{\twoheadrightarrow} \triangleq \xrightarrow{\tau}{}^{*} \circ \xrightarrow{\alpha} \circ \xrightarrow{\tau}{}^{*}$ if $\alpha \neq \tau$, and $\overset{\tau}{\twoheadrightarrow} \triangleq \xrightarrow{\tau}{}^{*}$. We say that an action α is relevant to a process P, abbreviated by $rel(\alpha, P)$, if α has the form $(\tilde{b})\overline{l{\cdot}c}\langle\tilde{v}\rangle$ and $\{\tilde{b}\} \cap fn(P) = \emptyset$.

3.3.3 Bisimilarity

Definition 3.7 *Given a process P and a set of locations Λ, such that $\Delta = \Lambda \setminus loc(P)$, for $\Delta = \{l_1, \cdots, l_n\}$, $n \geq 0$, we call $P \mid l_1{\cdot}0 \mid \cdots \mid l_n{\cdot}0$ the* extension of P to Λ*, and we denote it by P_Λ.*

Definition 3.8 *(Data equivalence) Given a set of locations Λ and a relation on processes $\mathcal{R}$, we denote by $\equiv_{(\mathcal{R},\Lambda)}$ the relation generated by the same axioms and rules as $\equiv$ but with axiom $P \equiv Q \Longrightarrow \Box P \equiv \Box Q$ replaced by $(\forall l \in \mathcal{L}.(P\{p/.\}\{l/\circlearrowleft\})_\Lambda \mathcal{R} (Q\{p/.\}\{l/\circlearrowleft\})_\Lambda) \Longrightarrow \Box P \equiv_{(\mathcal{R},\Lambda)} \Box Q$.*

From the previous definition, we have for example that $\equiv_{(\equiv,\Lambda)}$ coincides with $\equiv$ for any $\Lambda \neq \emptyset$. We now extend data equivalence to actions of the lts.

Definition 3.9 *(Action Equivalence) An equivalence relation $\mathcal{R}$ on values induces a corresponding relation on lts actions according to the following rules:*

$$\frac{\tilde{v}\mathcal{R}\tilde{u}}{\overline{\overline{l{\cdot}c}\langle\tilde{v}\rangle\mathcal{R}\overline{l{\cdot}c}\langle\tilde{u}\rangle}} \qquad \frac{\tilde{v}\mathcal{R}\tilde{u}}{(\tilde{b})\overline{l{\cdot}c}\langle\tilde{v}\rangle\mathcal{R}(\tilde{b})\overline{l{\cdot}c}\langle\tilde{u}\rangle}$$

$$\frac{\tilde{v}\mathcal{R}\tilde{u}}{l{\cdot}c(\tilde{v})\mathcal{R}l{\cdot}c(\tilde{u})} \qquad \frac{\mathcal{U}\mathcal{R}\mathcal{U}' \wedge (\chi)V\mathcal{R}(\xi)V'}{l{\cdot}\mathsf{update}_p(\mathcal{U},V)\mathcal{R}l{\cdot}\mathsf{update}_p(\mathcal{U}',V')}$$

$$\frac{u\mathcal{R}u' \wedge \mathcal{U}\mathcal{R}\mathcal{U}'}{\{u\}\oplus\mathcal{U}\mathcal{R}\{u'\}\oplus\mathcal{U}'} \qquad \frac{\Box P\mathcal{R}\Box P' \wedge \mathcal{U}\mathcal{R}\mathcal{U}'}{\{P\}\oplus\mathcal{U}\mathcal{R}\{P'\}\oplus\mathcal{U}'}$$

Definition 3.10 *(Store Equivalence) An equivalence relation $\mathcal{R}$ on values induces a corresponding relation on stores according to the following rule*

$$T\mathcal{R}T' \wedge D\mathcal{R}D' \Longrightarrow \{l \mapsto T\} \uplus D\mathcal{R}\{l \mapsto T'\} \uplus D'$$

Definition 3.11 *Bisimilarity ($\approx$) is the largest symmetric relation $\dot{\approx}$ such that $P\dot{\approx}Q$ implies $loc(P) = loc(Q)$ and if $P \xrightarrow{\alpha} P'$ with $rel(\alpha, Q)$ then $Q \overset{\alpha'}{\twoheadrightarrow} Q'$ where $\alpha' \equiv_{(\dot{\approx}, loc(P))} \alpha$ and $P'\dot{\approx}Q'$.*

Theorem 3.2 *$\approx$ is a congruence.*

Theorem 3.3 *Process equivalence is a sound approximation of barbed equivalence. In particular, $\approx \subseteq \sim_b$.*

Remark 3.1 *For simplicity, we do not consider the interesting issue related to completeness of bisimulation for the asynchronous π-calculus, which have been thoroughly studied in [21]. Our issues are orthogonal, and we believe that our bisimulation can be adapted analogously. Nonetheless it remains an open problem whether a bisimulation informed of [21] would be complete for $X\pi_2$. We are not aware of counterexamples besides the ones holding for π-calculus.*

Example 3.4 *We show that we can replicate a web service in such a way that the behaviour of the system is the same as the non-replicated case. Let internal nondeterminism be represented as $P \oplus Q \triangleq (\nu a)(\overline{a} \,|\, a.P \,|\, a.Q)$, where $a \notin fn(P|Q)$. We define two service calls to two interchangeable services, service R_1 on channel c and R_2 on channel d:*

$$Q_1 = [\![l{\cdot}\mathsf{Call}\ m{\cdot}c\langle\tilde{v}\rangle\ \mathsf{return}\ (\tilde{w}).Q]\!]_l$$

$$Q_2 = [\![l{\cdot}\mathsf{Call}\ n{\cdot}d\langle\tilde{v}\rangle\ \mathsf{return}\ (\tilde{w}).Q']\!]_l$$

$$P_m = [\![\mathsf{Define}\ c(\tilde{z})\ \mathsf{as}\ P_1\ \mathsf{output}\ \langle\tilde{w}\rangle]\!]_m$$

$$P_1 = [\![\mathsf{go}\ n.\overline{d}\langle\tilde{z}, l, x\rangle \oplus R_1]\!]_m$$

$$P_n = [\![\mathsf{Define}\ d(\tilde{z})\ \mathsf{as}\ P_2\ \mathsf{output}\ \langle\tilde{w}\rangle]\!]_n$$

$$P_2 = [\![\mathsf{go}\ m.\overline{c}\langle\tilde{z}, l, x\rangle \oplus R_2]\!]_n$$

We can show that, regardless of which service is invoked, a system built out of these processes behaves in the same way:

$$Q_1 \,|\, P_m \,|\, P_n \sim_b Q_2 \,|\, P_m \,|\, P_n$$

We can also show a result analogous to the single web-service given in Example 3.3. Given the specification process

$$Q_s = [\![\mathsf{go}\ m.R_1\{\tilde{v}/\tilde{z}\}.\mathsf{go}\ l.Q \oplus \mathsf{go}\ n.R_2\{\tilde{v}/\tilde{z}\}.\mathsf{go}\ l.Q']\!]_l$$

we can prove the equivalence

$$(\nu c, d)(Q_1 \mid P_m \mid P_n) \sim_b (\nu c, d)(Q_s \mid P_m \mid P_n)$$

where the restriction of c and d avoids competing services on the same channel. In particular, if $Q = Q'$, $R_1 = R_2$, and R_1 does not have any barb at m, then

$$(\nu c, d)(Q_1 \mid P_m \mid P_n) \sim_b (\nu c, d)([\![\mathsf{go}\ m.R_1\{\tilde{v}/\tilde{z}\}.\mathsf{go}\ l.Q]\!]_l \mid P_m \mid P_n).$$

This equivalence illustrates that we can replicate a web service without a client's knowledge.

Theorem 3.4 *We prove the equivalence anticipated in Example 3.3. Similar reasoning applies for Example 3.4.*

$$Q_1 = \quad (\!|l{\cdot}\mathsf{Call}\ m{\cdot}c\langle\tilde{v}\rangle\ \mathsf{return}\ (\tilde{w}).Q|\!)_l \tag{1}$$
$$Q_1' = \quad (\!|l{\cdot}\mathsf{Subscribe}\ m{\cdot}c\langle\tilde{v}\rangle\ \mathsf{return}\ (\tilde{w}).Q|\!)_l \tag{2}$$
$$Q_2 = \quad (\!|\mathsf{go}\ m.P\{\tilde{v}/\tilde{z}\}.\mathsf{go}\ l.Q|\!)_l \tag{3}$$
$$P_0 = \quad (\!|\mathsf{Define}\ c(\tilde{z})\ \mathsf{as}\ P\ \mathsf{output}\ \langle\tilde{w}\rangle|\!)_m \tag{4}$$

Then

1. $(\nu c)(Q_1' \mid P_0) \sim_b (\nu c)(Q_2 \mid P_0)$
2. *If P returns its results only once,* $(\nu c)(Q_1 \mid P_0) \sim_b (\nu c)(Q_2 \mid P_0)$

Proof.

1. We show that
$$(\nu c)(Q_1' \mid P_0) \approx (\nu c)(Q_2 \mid P_0)$$
We consider always the case when the LHS moves first. The symmetric case is analogous.

Note that both processes have the same domain. The only action that $(\nu c)(Q_1' \mid P_0)$ can perform is a τ by rule (Exit), to become
$$(\nu c)((\nu b)(m\langle\overline{m{\cdot}c}\langle\tilde{v}, l, b\rangle\rangle \mid !l{\cdot}b(\tilde{w}).Q) \mid P_0)$$
We let Q_2 perform its (Exit) operation and we must show that
$$(\nu c)((\nu b)(m\langle\overline{m{\cdot}c}\langle\tilde{v}, l, b\rangle\rangle \mid !l{\cdot}b(\tilde{w}).Q) \mid P_0) \approx (\nu c)(m\langle{}^{m}\!\frown P\{\tilde{v}/\tilde{z}\}.h{\cdot}\mathsf{go}l.Q\rangle \mid P_0)$$
for some h depending on the translation for obtaining Q_2, in particular by the translation of P at m. The LHS can only perform a τ action by rule

(Enter) and the RHS can do the same (both domains remain the same), giving us

$$(\nu c)((\nu b)(\overline{m{\cdot}c}\langle \tilde{v}, l, b\rangle \mid !l{\cdot}b(\tilde{w}).Q) \mid P_0) \approx (\nu c)(^{m\curvearrowright}P\{\tilde{v}/\tilde{z}\}.h{\cdot}\mathsf{go}\ l.Q \mid P_0)$$

Now the LHS can do a τ by (Com!), (Par) and (Struct), and the RHS stays the same:

$$(\nu c, b)(!l{\cdot}b(\tilde{w}).Q \mid P_0 \mid {}^{m\curvearrowright}P\{\tilde{v}/\tilde{z}\}.h{\cdot}\mathsf{go}\ l.\overline{l{\cdot}b}\langle\tilde{w}\rangle) \approx (\nu c)(^{m\curvearrowright}P\{\tilde{v}/\tilde{z}\}.h{\cdot}\mathsf{go}\ l.Q \mid P_0)$$

where also the LHS contains the $h{\cdot}\mathsf{go}\ l.\_$ prefix determined again by the translation of P at m. By hypothesis, $b \notin fn(P\{\tilde{v}/\tilde{z}\})$, P binds $\tilde{w}$ and the free variables of Q are contained in $\tilde{w}$. Therefore we can continue adding pairs to the bisimulation relation by performing the same action on both sides as far as $P\{\tilde{v}/\tilde{z}\}$ is concerned. In particular P can split in more than one thread of activity, and any of these threads can eventually terminate firing the continuation $h{\cdot}\mathsf{go}\ l.l{\cdot}b(\tilde{u})$ for some $\tilde{u}$ computed by P from $\tilde{v}$. We represent such a generic case by the pair

$$(\nu c, b)(!l{\cdot}b(\tilde{w}).Q \mid P_0 \mid {}^{m\curvearrowright}P'\{\tilde{v}/\tilde{z}\}.h{\cdot}\mathsf{go}\ l.l{\cdot}b(\tilde{w}) \mid h{\cdot}\mathsf{go}\ l.\overline{l{\cdot}b}\langle\tilde{u}\rangle)$$

$$\approx (\nu c)(^{m\curvearrowright}P'\{\tilde{v}/\tilde{z}\}.h{\cdot}\mathsf{go}l.Q \mid h{\cdot}\mathsf{go}l.Q\{\tilde{u}/\tilde{w}\} \mid P_0)$$

After moving with rules (Exit) and (Enter) on both sides, we reach

$$(\nu c, b)(!l{\cdot}b(\tilde{w}).Q \mid P_0 \mid {}^{m\curvearrowright}P'\{\tilde{v}/\tilde{z}\}.h{\cdot}\mathsf{go}\ l.l{\cdot}b(\tilde{w}) \mid \overline{l{\cdot}b}\langle\tilde{u}\rangle)$$

$$\approx (\nu c)(^{m\curvearrowright}P'\{\tilde{v}/\tilde{z}\}.Q \mid h{\cdot}\mathsf{go}l.Q\{\tilde{u}/\tilde{w}\} \mid P_0)$$

and by (Com!) on the LHS only

$$(\nu c, b)(!l{\cdot}b(\tilde{w}).Q \mid Q\{\tilde{u}/\tilde{w}\} \mid P_0 \mid {}^{m\curvearrowright}P'\{\tilde{v}/\tilde{z}\}.h{\cdot}\mathsf{go}\ l.l{\cdot}b(\tilde{w}))$$

$$\approx (\nu c)(^{m\curvearrowright}P'\{\tilde{v}/\tilde{z}\}.Q \mid Q\{\tilde{u}/\tilde{w}\} \mid P_0)$$

The same procedure can be carried on for the other derivatives of $^{m\curvearrowright}P'\{\tilde{v}/\tilde{z}\}$.

2. The proof is similar to the previous case, but relies on the assumption that there will be exactly one reply.

□

4 Concluding Remarks

This paper introduces X$d\pi$, a simple calculus for describing the interaction between data and processes across distributed locations. We use a simple data model consisting of unordered labelled trees, with embedded processes and pointers to other parts of the network, and π-processes extended with an explicit migration primitive and an update command for interacting with data.

Unlike the π-calculus and its extensions, where typically data is encoded as processes, the X$d\pi$-calculus models data and processes at the same level of abstraction, enabling us to study how properties of data can be affected by process interaction.

Alex Ahern has developed a prototype implementation, adapting the ideas presented here to XML standards. The implementation embeds processes in XML documents and uses XPath as a query language. Communication between peers is provided through SOAP-based web services and the working space of each location is endowed with a process scheduler based on ideas from PICT [23]. We aim to continue this implementation work, perhaps incorporating ideas from other recent work on languages based on the π-calculus [11, 13].

Active XML [4] is probably the closest system to our X$d\pi$-calculus. It is based on web services and service calls embedded in data, rather than π-processes. There is however a key difference in approach: Active XML focusses on modelling data transformation and delegates the role of distributed process interaction to the implementation; in contrast, process interaction is fundamental to our model. There are in fact many similarities between our model and features of the Active XML implementation, and we are in the process of doing an in-depth comparison between the two projects.

Developing process equivalences for X$d\pi$ is non-trivial. We have defined a notion of barbed equivalence between processes, based on the update operations that processes can perform on data, and have briefly described a proof method for proving such equivalences between processes. There are other possible definitions of observational equivalence, and a comprehensive study of these choices will be essential in future. We also plan to adapt type theories and reasoning techniques studied for distributed process calculi to analyse properties such as boundary access control, message and host failure, and data integrity. This paper has provided a first step towards the adaptation of techniques associated with process calculi to reason about the dynamic evolution of data on the Web.

4.1 Acknowledgements

We would like to thank Serge Abiteboul, Tova Milo, Val Tannen, Luca Cardelli, Giorgio Ghelli and Nobuko Yoshida for many stimulating discussions.

References

[1] Abiteboul, S., O. Benjelloun, T. Milo, I. Manolescu and R. Weber, *Active XML: A data-centric perspective on Web services*, Verso Report number 213 (2002).

[2] Abiteboul, S., A. Bonifati, G. Cobena, I. Manolescu and T. Milo, *Dynamic XML documents with distribution and replication*, in: *Proceedings of ACM SIGMOD Conference*, 2003.

[3] Abiteboul, S., P. Buneman and D. Suciu, "Data on the Web: from relations to semistructured data and XML," Morgan Kaufmann, 2000.

[4] Abiteboul, S. et al., *Active XML primer*, INRIA Futurs, GEMO Report number 275 (2003).

[5] Berger, M., "Towards Abstractions for Distributed Systems," Ph.D. thesis, Imperial College London (2002).

[6] Berger, M., K. Honda and N. Yoshida, *Linearity and bisimulation.*, in: *Proceedings of FoSSaCS'02* (2002), pp. 290–301.

[7] Bierman, G. and P. Sewell, *Iota: a concurrent XML scripting language with application to Home Area Networks*, University of Cambridge Technical Report UCAM-CL-TR-557 (2003).

[8] Braumandl, R., M. Keidl, A. Kemper, D. Kossmann, A. Kreutz, S. Seltzsam and K. Stocker, *Objectglobe: Ubiquitous query processing on the internet*, To appear in the VLDB Journal: Special Issue on E-Services (2002).

[9] Carbone, M. and S. Maffeis, *On the expressive power of polyadic synchronisation in π-calculus*, Nordic Journal of Computing **10** (2003), pp. 70–98.

[10] Cardelli, L. and G. Ghelli, *A query language based on the ambient logic*, in: *Proceedings of ESOP'01)*, LNCS **2028** (2001), pp. 1–22.

[11] Conchon, S. and F. L. Fessant, *JOCaml: Mobile agents for Objective-Caml*, in: *Proceedings of ASA'99/MA'99*, Palm Springs, CA, USA, 1999.

[12] Fournet, C., "The Join-Calculus: a Calculus for Distributed Mobile Programming," Ph.D. thesis, Ecole Polytechnique (1998).

[13] Gardner, P., C. Laneve and L. Wischik, *Linear forwarders*, in: *Proceedings of CONCUR 2003*, LNCS **2761** (2003), pp. 415–430.

[14] Godskesen, J., T. Hildebrandt and V. Sassone, *A calculus of mobile resources*, in: *Proceedings of CONCUR'02*, LNCS, 2002.

[15] Gordon, A. and R. Pucella, *Validating a web service security abstraction by typing*, in: *Proceedings of the 2002 ACM Workshop on XML Security*, 2002, pp. 18–29.

[16] Hennessy, M. and J. Riely, *Resource access control in systems of mobile agents*, in: *Proceedings of HLCL '98*, ENTCS **16.3** (1998), pp. 3–17.

[17] Honda, K. and M. Tokoro, *An object calculus for asynchronous communication*, in: *Proceedings of ECOOP*, LNCS **512** (1991), pp. 133–147.

[18] Honda, K. and N. Yoshida, *On reduction-based process semantics*, Theoretical Computer Science **151** (1995), pp. 437–486.

[19] Kemper, A. and C. Wiesner, *Hyperqueries: Dynamic distributed query processing on the internet*, in: *Proceedings of VLDB'01*, 2001, pp. 551–560.

[20] World Wide Web Consortium, *XML Path Language (XPath) Version 1.0*, available at `http://w3.org/TR/xpath`.

[21] Merro, M., *On equators in asynchronous name-passing calculi without matching (Extended abstract)*, in: *EXPRESS '99*, ENTCS **27** (1999).

[22] Milner, R., J. Parrow and J. Walker, *A calculus of mobile processes, I and II*, Information and Computation **100** (1992), pp. 1–40,41–77.

[23] Pierce, B. C. and D. N. Turner, *Pict: A programming language based on the pi-calculus*, in: *Proof, Language and Interaction: Essays in Honour of Robin Milner* (2000).

[24] Sahuguet, A., "ubQL: A Distributed Query Language to Program Distributed Query Systems," Ph.D. thesis, University of Pennsylvania (2002).

[25] Sahuguet, A., B. Pierce and V. Tannen, *Distributed Query Optimization: Can Mobile Agents Help?*, Unpublished draft.

[26] Sahuguet, A. and V. Tannen, *Resource Sharing Through Query Process Migration*, University of Pennsylvania Technical Report MS-CIS-01-10 (2001).

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

$wf(c), wf(l), wf(@l{:}p), wf(\square^{\circlearrowleft}\!\!\frown P)$, similarly for variables.

$$\frac{wf(P) \quad (loc(D) = loc(P)) \quad (f_{\mathcal{V}}(D) \cup f_{\mathcal{C}_s}(D) \cup f_{\mathcal{V}}(P) = \emptyset)}{wf(D, P)}$$

$$wf(0) \qquad wf(l{\cdot}0) \qquad \frac{wf(P) \quad wf(Q)}{wf(P \mid Q)}$$

$$\frac{wf(P) \quad P \overset{l}{\frown}}{wf(l\langle P\rangle)} \qquad \frac{wf(\tilde{v})}{wf(\overline{l{\cdot}c}\langle\tilde{v}\rangle)} \qquad \frac{wf(v') \quad wf(\tilde{v})}{wf(v', \tilde{v})}$$

$$\frac{wf(P) \quad P \overset{l}{\frown}}{wf(l{\cdot}c(\tilde{z}).P)} \qquad \frac{wf(P) \quad P \overset{l}{\frown}}{wf(!l{\cdot}c(\tilde{z}).P)} \qquad \frac{wf(P)}{wf((\nu c)P)}$$

$$\frac{wf(P) \quad P \overset{m}{\frown}}{wf(l{\cdot}\mathsf{go}\ m.P)} \qquad \frac{wf(P) \quad wf(U) \quad P \overset{l}{\frown}}{wf(l{\cdot}\mathsf{update}_p(\chi, U).P)}$$

Table 9: Well-formedness for Xπ_2 networks.

A Proofs

A.1 Proof of Full Abstraction

First we define non-unique normal forms for both X$d\pi$ and Xπ_2 processes. Then we define the encoding from X$d\pi$ to Xπ_2 and an inverse encoding exploiting Xπ_2 normal forms. Finally we establish a close correspondence between the notions of observation, reduction and context in the two calculi. We use Notation $P \overset{l}{\not\frown}$ to mean that there are no P', Q, R such that $P \equiv Q \mid P' \mid R$. In the rest of this section, we consider only well-formed networks, for both languages.

Lemma A.1 (Normal forms)

1. *For every X$d\pi$ network N with $loc(N) = \{l_1, ..., l_k\}$*

   $$N \equiv (\nu\tilde{c})(m_1\langle Q_1\rangle \mid \ldots \mid m_h\langle Q_h\rangle \mid l_1 \left[\, T_1 \parallel P_1 \,\right] \mid \ldots \mid l_k \left[\, T_1 \parallel P_k \,\right] \mid 0)$$

   where each P_i is non-restricted.

2. *For every Xπ_2 process P with $loc(P) = \{l_1, ..., l_k\}$*

   $$P \equiv (\nu\tilde{c})(m_1\langle Q_1\rangle \mid \ldots \mid m_h\langle Q_h\rangle \mid P_1 \mid \ldots \mid P_k \mid 0)$$

   where each P_i is a non-restricted (possibly null) process located at l_i.

Proof. Both cases follow by using the structural congruence rules for α-conversion, scope extrusion and commutativity of the parallel operator. □

We define in Table 10 the encoding from X$d\pi$ to Xπ_2 terms, and in Table 11 the inverse encoding. The second encoding is defined on Xπ_2 processes in normal form only.

$$
\begin{array}{ll}
(\!|0|\!) = \emptyset & [\![0]\!] = 0 \\
(\!|N \mid M|\!) = (\!|N|\!) \cup (\!|M|\!) & [\![N \mid M]\!] = [\![N]\!] \mid [\![M]\!] \\
(\!|(\nu c)N|\!) = (\!|N|\!) & [\![(\nu c)N]\!] = (\nu c)[\![N]\!] \\
(\!|l\,[\,T \,||\, P\,]|\!) = \{l \mapsto [\![T]\!]\} & [\![l\,[\,T \,||\, P\,]]\!] = \langle\!|P|\!\rangle_l \\
(\!|l\langle P\rangle|\!) = \emptyset & [\![l\langle P\rangle]\!] = l\langle\langle\!|P|\!\rangle_l\rangle
\end{array}
$$

$$
\begin{array}{ll}
[\![0]\!] = 0 & \langle\!|0|\!\rangle_l = l{\cdot}0 \\
[\![T \mid T']\!] = [\![T]\!] \mid [\![T']\!] & \langle\!|P \mid Q|\!\rangle_l = \langle\!|P|\!\rangle_l \mid \langle\!|Q|\!\rangle_l \\
[\![\mathsf{a}[\,T\,]]\!] = \mathsf{a}[\,[\![T]\!]\,] & \langle\!|(\nu c)P|\!\rangle_l = (\nu c)\langle\!|P|\!\rangle_l \\
[\![@l{:}p]\!] = @l{:}p & \langle\!|\mathsf{go}\ m.P|\!\rangle_l = l{\cdot}\mathsf{go}\ m.\langle\!|P|\!\rangle_m \\
[\![c]\!] = c,\ [\![l]\!] = l,\ [\![p]\!] = p & \langle\!|\overline{a}\langle\tilde{v}\rangle|\!\rangle_l = \overline{l{\cdot}a}\langle\tilde{v}\rangle \\
[\![x]\!] = x,\ [\![\chi]\!] = \chi & \langle\!|a(\tilde{x}).P|\!\rangle_l = l{\cdot}a(\tilde{x}).\langle\!|P|\!\rangle_l \\
[\![v', \tilde{v}]\!] = [\![v']\!], [\![\tilde{v}]\!] & \langle\!|!a(\tilde{x}).P|\!\rangle_l = !l{\cdot}a(\tilde{x}).\langle\!|P|\!\rangle_l \\
[\![\Box P]\!] = \Box\langle\!|P|\!\rangle_{\circlearrowleft} & \langle\!|\mathsf{update}_p(\chi, U).P|\!\rangle_l = l{\cdot}\mathsf{update}_p(\chi, [\![U]\!]).\langle\!|P|\!\rangle_l \\
 & \langle\!|\mathsf{run}_p|\!\rangle_l = l{\cdot}\mathsf{run}_p
\end{array}
$$

Table 10: Encodings from X$d\pi$ to Xπ_2.

$$
\begin{array}{l}
\mathcal{D}((\nu c)P)_D = (\nu c)\mathcal{D}(P)_D \\
\mathcal{D}(l\langle P\rangle \mid Q)_D = l\langle\mathcal{D}_{\mathcal{P}}(P)\rangle \mid \mathcal{D}(Q)_D \\
\mathcal{D}(l\frown P \mid l\curvearrowright Q)_D = l\,[\,\mathcal{D}_{\mathcal{V}}(D(l)) \,||\, \mathcal{D}_{\mathcal{P}}(P)\,] \mid \mathcal{D}(Q)_D \\
\mathcal{D}(0)_D = 0
\end{array}
$$

$$
\begin{array}{ll}
\mathcal{D}_{\mathcal{V}}(0) = 0 & \mathcal{D}_{\mathcal{P}}(l{\cdot}0) = 0 \\
\mathcal{D}_{\mathcal{V}}(T \mid T') = \mathcal{D}_{\mathcal{V}}(T) \mid \mathcal{D}_{\mathcal{V}}(T') & \mathcal{D}_{\mathcal{P}}(P \mid Q) = \mathcal{D}_{\mathcal{P}}(P) \mid \mathcal{D}_{\mathcal{P}}(Q) \\
\mathcal{D}_{\mathcal{V}}(\mathsf{a}[\,T\,]) = \mathsf{a}[\,\mathcal{D}_{\mathcal{V}}(T)\,] & \mathcal{D}_{\mathcal{P}}((\nu c)P) = (\nu c)\mathcal{D}_{\mathcal{P}}(P) \\
\mathcal{D}_{\mathcal{V}}(@l{:}p) = @l{:}p & \mathcal{D}_{\mathcal{P}}(l{\cdot}\mathsf{go}\ m.P) = \mathsf{go}\ m.\mathcal{D}_{\mathcal{P}}(P) \\
\mathcal{D}_{\mathcal{V}}(c) = c,\ \mathcal{D}_{\mathcal{V}}(l) = l,\ \mathcal{D}_{\mathcal{V}}(p) = p & \mathcal{D}_{\mathcal{P}}(\overline{l{\cdot}a}\langle\tilde{v}\rangle) = \overline{a}\langle\mathcal{D}_{\mathcal{V}}(\tilde{v})\rangle \\
\mathcal{D}_{\mathcal{V}}(x) = x,\ \mathcal{D}_{\mathcal{V}}(\chi) = \chi & \mathcal{D}_{\mathcal{P}}(l{\cdot}a(\tilde{x}).P) = a(\tilde{x}).\mathcal{D}_{\mathcal{P}}(P) \\
\mathcal{D}_{\mathcal{V}}(v', \tilde{v}) = \mathcal{D}_{\mathcal{V}}(v'), \mathcal{D}_{\mathcal{V}}(\tilde{v}) & \mathcal{D}_{\mathcal{P}}(!l{\cdot}a(\tilde{x}).P) = !a(\tilde{x}).\mathcal{D}_{\mathcal{P}}(P) \\
\mathcal{D}_{\mathcal{V}}(\Box P) = \Box\mathcal{D}_{\mathcal{P}}(P) & \mathcal{D}_{\mathcal{P}}(l{\cdot}\mathsf{update}_p(\chi, U).P) = \mathsf{update}_p(\chi, \mathcal{D}_{\mathcal{V}}(U)).\mathcal{D}_{\mathcal{P}}(P) \\
 & \mathcal{D}_{\mathcal{P}}(l{\cdot}\mathsf{run}_p) = \mathsf{run}_p
\end{array}
$$

Table 11: Inverse Encodings

Observation 1 *Both encodings and both reduction relations preserve well-formedness:*

- *if N is well-formed then $((\!|N|\!), [\![N]\!])$ is well-formed, and if $N \searrow N'$ also N' is well-formed;*
- *if (D, P) is well-formed and P is in normal form then $\mathcal{D}(P)_D$ is well*

formed, and if $(D, P) \to (D', P')$ *also* (D', P') *is well-formed.*

Lemma A.2 (Inverse encoding) *For all networks* N *and for all networks* (D, P) *the encodings given in Table 10 and Table 11 are complementary up to structural congruence:*

1. $N \equiv N' = \mathcal{D}([\![N']\!])_{(\!|N'|\!)}$;
2. $(D, P) \equiv (D, P') = ((\!|\mathcal{D}(P')_D|\!), [\![\mathcal{D}(P')_D]\!])$.

Proof. Both cases follow by inspection of the rules of the two encodings, using points (i) and (ii) of Lemma A.1 and Observation 1. □

The following lemma is the crucial step towards the full abstraction result.

Lemma A.3 (Operational Correspondece) *For all Xd*π *networks* N, $N \searrow N'$ *if and only if* $((\!|N|\!), [\![N]\!]) \searrow ((\!|N'|\!), [\![N']\!])$.

Proof. We split the proof in two, showing by induction on the derivations that for all Xdπ networks N and for all Xaπ_2 networks (D, P)

1. if $N \searrow N'$ then $((\!|N|\!), [\![N]\!]) \to ((\!|N'|\!), [\![N']\!])$;
2. if $(D, P) \to (D', P')$ then $\mathcal{D}(P)_D \searrow \mathcal{D}(P')_{D'}$.

We show the two most interesting cases for each point.

1. - (Par) We consider directly the case when rule (Par) is applied, as (Res) and (Struct) are trivial. We want to show that

     $$N|M \searrow N'|M \Rightarrow ((\!|N|M|\!), [\![N|M]\!]) \to ((\!|N'|M|\!), [\![N'|M]\!])$$

     By the premise of rule (Par) we have $N \searrow N'$ and, by definition of encoding, we need to show that

     $$((\!|N|\!) \cup (\!|M|\!), [\![N]\!]|[\![M]\!]) \to ((\!|N'|\!) \cup (\!|M|\!), [\![N']\!]|[\![M]\!])$$

     By induction we have that

     $$N \searrow N' \Rightarrow ((\!|N|\!), [\![N]\!]) \to ((\!|N'|\!), [\![N']\!])$$

     Since N and M are disjoint, we have that $dom((\!|N|\!)) \cap dom((\!|M|\!)) = \emptyset$ and therefore we can use the (Par) rule of Xπ_2 to conclude, deriving

     $$((\!|N|\!) \uplus (\!|M|\!), [\![N]\!]|[\![M]\!]) \to ((\!|N'|\!) \uplus (\!|M|\!), [\![N']\!]|[\![M]\!])$$

   - (Com) Again we skip the cases of (Par), (Res) and (Struct), and we consider directly the application of (Com). Let $R = \overline{c}\langle\tilde{v}\rangle \mid c(\tilde{x}).P|Q$ and $R' = P\{\tilde{v}/\tilde{x}\}|Q$. We want to show that

     $$l\left[T \parallel R\right] \searrow l\left[T \parallel R'\right] \Rightarrow ((\!|l\left[T \parallel R\right]|\!), [\![l\left[T \parallel R\right]]\!]) \to ((\!|l\left[T \parallel R'\right]|\!), [\![l\left[T \parallel R'\right]]\!])$$

By definition of encoding and R

$$(\!(l\left[\,T \,\middle\|\, R\,\right]\!), [\![l\left[\,T \,\middle\|\, R\,\right]\!]) = (\{l \mapsto [\![T]\!]\}, \overline{l{\cdot}c}\langle[\![\tilde{v}]\!]\rangle \mid l{\cdot}c(\tilde{x}).(\!|P|\!)_l | (\!|Q|\!)_l)$$

and by (Par) and (Com) we derive

$$(\{l \mapsto [\![T]\!]\}, \overline{l{\cdot}c}\langle[\![\tilde{v}]\!]\rangle \mid l{\cdot}c(\tilde{x}).(\!|P|\!)_l | (\!|Q|\!)_l) \to (\{l \mapsto [\![T]\!]\}, ((\!|P|\!)_l)\{[\![\tilde{v}]\!]/\tilde{x}\} | (\!|Q|\!)_l)$$

Again by definition of encoding and of R' we conclude with

$$(\!(l\left[\,T \,\middle\|\, R'\,\right]\!), [\![l\left[\,T \,\middle\|\, R'\,\right]\!]) = (\{l \mapsto [\![T]\!]\}, (\!|P|\!)_l\{[\![\tilde{v}]\!]/\tilde{x}\} | (\!|Q|\!)_l)$$

2.
- (Par) The reasoning is analogous to the corresponding case of point (i), with the additional complication that, since the premise $(D, P) \to (D', P')$ of (Par) might involve a network (D, P) not well-formed, in order to apply the inductive hypothesis it may be necessary to use (Struct) to reorganise the original process, followed by an *ad-hoc* instance of rule (Par).
- (Com) In order to derive a transition for a well-formed network (D, P) from this rule, we must start from a sub-network of the form $(\emptyset, R)$ where $R = \overline{l{\cdot}c}\langle\tilde{v}\rangle | l{\cdot}c(\tilde{x}).P'$, obtaining the transition

$$(\emptyset, \overline{l{\cdot}c}\langle\tilde{v}\rangle | l{\cdot}c(\tilde{x}).P') \to (\emptyset, P'\{\tilde{v}/\tilde{x}\})$$

Then, we can always apply rules (Par) and (Struct) with $B = \{l \mapsto D(l)\}$ to group together all the other sub-processes of P located at l (process ${}^{l}\!\frown Q$), obtaining

$$(B, R |^{l}\!\frown Q) \searrow (B, P'\{\tilde{v}/\tilde{x}\} |^{l}\!\frown Q)$$

which is a reduction on well-formed networks because also R is located at l by well-formedness of P' and by Observation 1. We can use (Struct) to put the term in normal form and we can now use the inverse encoding

$$\mathcal{D}({}^{l}\!\frown (R|Q))_B \equiv l\left[\,\mathcal{D}_{\mathcal{V}}(B) \,\middle\|\, \mathcal{D}_{\mathcal{P}}(R) | \mathcal{D}_{\mathcal{P}}(Q)\,\right]$$

where $\mathcal{D}_{\mathcal{P}}(R) = \overline{c}\langle\mathcal{D}_{\mathcal{V}}(\tilde{v})\rangle | c(\tilde{x}).\mathcal{D}_{\mathcal{P}}(P')$, to derive the corresponding reduction with rule (Com) in X$d\pi$

$$l\left[\,\mathcal{D}_{\mathcal{V}}(B) \,\middle\|\, \mathcal{D}_{\mathcal{P}}(R) | \mathcal{D}_{\mathcal{P}}(Q)\,\right] \searrow l\left[\,\mathcal{D}_{\mathcal{V}}(B) \,\middle\|\, \mathcal{D}_{\mathcal{P}}(P')\{\mathcal{D}_{\mathcal{V}}(\tilde{v})/\tilde{x}\} | \mathcal{D}_{\mathcal{P}}(Q)\,\right]$$

At this point standard reasoning using rules (Par), (Res) and (Struct) closes the proof.

□

Lemma A.4 (Observational Correspondence) *For all X$d\pi$ networks*

1. *$N \downarrow_{l\cdot\beta}$ if and only if $(([\![N]\!]), [\![N]\!]) \downarrow_{l\cdot\beta}$;*
2. *$N \Downarrow_{l\cdot\beta}$ if and only if $(([\![N]\!]), [\![N]\!]) \Downarrow_{l\cdot\beta}$.*

Proof.

1. Follows from the definitions of $\downarrow$ for the respective calculi and by Lemma A.2.
2. Follows from (i) and from Lemma A.3.

□

Lemma A.5 (Contextual Correspondence) *For all Xdπ networks N*

1. *for all Xdπ contexts $C_1[-]$ there is an Xπ$_2$ context $C_2[-]$ such that*
$$(([\![C_1[N]]\!]), [\![C_1[N]]\!]) \equiv C_2[((([\![N]\!]), [\![N]\!])];$$
2. *for all Xπ$_2$ contexts $C_2[-]$ there is an Xdπ context $C_1[-]$ such that*
$$\mathcal{D}(C_2[((([\![N]\!]), [\![N]\!])])_{C_2[((([\![N]\!]), [\![N]\!])]} \cong C_1[N].$$

Proof. Case (1) follows directly by the definition of encoding. Case (2) is non-trivial because a context $C_1[-]$ in Xπ$_2$ can place a process into an existing location, whereas contexts for Xdπ do not have this power. Therefore it is necessary to simulate the insertion of a process P at l by placing in parallel to N the message $l\langle P\rangle$, which after a reduction produces the desired effect. □

A.1.1 Proof of Theorem 3.1

Theorem A.1 *$N \cong M$ if and only if $(([\![N]\!]), [\![N]\!]) \cong (([\![M]\!]), [\![M]\!])$.*

Proof. The definition of $\cong$ is the same for both calculi, Lemma A.4 guarantees that observables are preserved, Lemma A.3 guarantees that reduction is preserved, and Lemma A.5 guarantees that contexts are preserved. □

A.2 Proof of Congruence

Lemma A.6 *(Variant Lemma) $P \approx Q \Longrightarrow P\{b/a\} \approx Q\{b/a\}$ for any $a, b \in \mathcal{C},\ b \notin fn(P, Q)$.*

Proof. By a straightforward adaptation of the corresponding proof in [18]. □

Lemma A.7 *(Substitution lemma) Let P be a term with free variables $\tilde{x}$:*
$$\tilde{v} \equiv_{(\approx, loc(P))} \tilde{v'} \Longrightarrow P\{\tilde{v}/\tilde{x}\} \approx P\{\tilde{v'}/\tilde{x}\}$$

Proof. The relation $\{(P\{\tilde{v}/\tilde{x}\}, P\{\tilde{v'}/\tilde{x}\}) | \tilde{v} \equiv_{(\approx, loc(P))} \tilde{v'}\}$ is a bisimulation. Note that $\tilde{v}$ and $\tilde{v'}$ can differ only if they contain trees that are structurally equivalent, or processes which are bisimilar. The idea is that after a binding action (such as (Com), etc.) it is possible to write explicitly the substitution, as in $\overline{l{\cdot}a}\langle v\rangle \,|\, l{\cdot}a(x).\overline{l{\cdot}b}\langle x\rangle \xrightarrow{\tau} \overline{l{\cdot}b}\langle x\rangle\{v/x\}$, whereas in comparing actions such as $\overline{l{\cdot}b}\langle v\rangle, \overline{l{\cdot}b}\langle v'\rangle$ we reason using the hypothesis. □

Observation 2 *By having rule (Struct) in the lts, we automatically have that bisimulation up to structural congruence is a bisimulation.*

Lemma A.8 *If $P \approx Q$ then $P \,|\, l{\cdot}0 \approx Q \,|\, l{\cdot}0$.*

Proof. Suppose $l \in loc(P)$. Then, by definition of $\approx$, $l \in loc(Q)$, and by $\equiv$, $P \,|\, l{\cdot}0 \equiv P \approx Q \equiv Q \,|\, l{\cdot}0$. Suppose $l \notin loc(P)$. Then, by definition of $\approx$, $l \notin loc(Q)$. Since $P \xrightarrow{\tau} P \,|\, l{\cdot}0$, then $Q \stackrel{\tau}{\twoheadrightarrow} Q' \approx P \,|\, l{\cdot}0$, and $l \in loc(Q')$. By definition of lts, $Q \,|\, l{\cdot}0 \stackrel{\tau}{\twoheadrightarrow} Q' \,|\, l{\cdot}0$ and by $\equiv$, $Q' \,|\, l{\cdot}0 \equiv Q' \approx P \,|\, l{\cdot}0$. □

A.2.1 Proof of Theorem 3.2

Proof. The proof is based on an analogous proof for the asynchronous π-calculus found in [18]. We need to show the following three points:

1. $\approx$ is an equivalence relation;
2. $P \approx Q \Longrightarrow (\nu\tilde{c})P \approx (\nu\tilde{c})Q$;
3. $P \approx Q \Longrightarrow P \,|\, R \approx Q \,|\, R$.

It is important to keep in mind, during the proof, the distinction between session names and public names, that a scripted processes is well-formed only if it has no free session names, and that processes can be used as values only when they are scripted.

1. Reflexivity and symmetry are immediate. We show transitivity: the relation $\mathcal{R} = \{(P, Q) | P \approx R \wedge R \approx Q\}$ is included in $\approx$. Suppose $P \xrightarrow{\alpha} P'$ and $rel(\alpha, Q)$. There are two cases, based on the relevance of α to R. If $rel(\alpha, R)$ the proof is straightforward. If α is not relevant to R then the action must necessarily be a bound output, in order to have bound names. Suppose therefore $\alpha = (\tilde{b})\overline{l{\cdot}a}\langle\tilde{v}\rangle$ and there exists $\tilde{c}$ such that $\{\tilde{c}\} \subseteq \{\tilde{b}\}$, and $\{\tilde{c}\} \subseteq fn(R)$. Now choose a $\tilde{c'}$ with the same length as $\tilde{c}$, in such a way that $\{\tilde{c'}\} \cap fn(P, Q, R) = \emptyset$. By the side condition on the lts rule (Par), $\{\tilde{c}\} \cap fn(P) = \emptyset$ and by Lemma A.6, $P = P\{\tilde{c'}/\tilde{c}\} \approx R\{\tilde{c'}/\tilde{c}\} = R'$. Similarly we get $R' \approx Q$. But now $rel(\tilde{b}, R')$, and reasoning as in the previous case, we can conclude.

2. We show that $\mathcal{R} = \{((\nu\tilde{c})P, (\nu\tilde{c})Q) | P \approx Q\}$ is included in $\approx$. Tau transitions, output and update commands are trivial, whereas input requires some attention. Suppose $(\nu\tilde{c})P \xrightarrow{l\cdot a(\tilde{v})} P'$ where we use rule (STRUCT) to alpha-convert $\tilde{c}$ to $\tilde{c'}$ in order to avoid the capture of some free session name in $\tilde{v}$. Then it is the case that $(\nu\tilde{c})P \equiv (\nu\tilde{c'})(P\{\tilde{c'}/\tilde{c}\}) \xrightarrow{l\cdot a(\tilde{v})} (\nu\tilde{c'})P'' \equiv P'$, and since $\{a, \tilde{v}\} \cap \{\tilde{c'}\} = \emptyset$, we have $P\{\tilde{c'}/\tilde{c}\} \xrightarrow{l\cdot a(\tilde{v})} P''$. By Lemma A.6 we get $Q\{\tilde{c'}/\tilde{c}\} \overset{l\cdot a(\tilde{v'})}{\twoheadrightarrow} Q'' \approx P''$ with $\tilde{v} \equiv_{(\approx, loc(P))} \tilde{v'}$. Again since $\{a, \tilde{v}\} \cap \{\tilde{c'}\} = \emptyset$ we have $(\nu\tilde{c})Q \equiv ((\nu\tilde{c'})Q\{\tilde{c'}/\tilde{c}\}) \overset{l\cdot a(\tilde{v'})}{\twoheadrightarrow} (\nu\tilde{c'})Q''$ and we are done.

3. We show that $\{(\nu\tilde{c})(P \mid R), (\nu\tilde{c})(Q \mid R')) | P \approx Q \wedge R_{loc(P)} \approx R'_{loc(P)}\}$ is included in $\approx$. There are three interesting cases, depending whether $(\nu\tilde{c})(P \mid R) \xrightarrow{\alpha} P'$ is caused by an action of P alone, of R alone, or by a communication between P and R.

   (a) Suppose $(\nu\tilde{c})(P \mid R) \xrightarrow{\alpha} (\nu\tilde{c})(P' \mid R)$ with $rel(\alpha, (\nu\tilde{c})(Q \mid R'))$ is derived from $P \mid R \xrightarrow{\alpha} P' \mid R$ which comes from $P \xrightarrow{\alpha} P'$. By $P \approx Q$ follows $Q \overset{\alpha'}{\twoheadrightarrow} Q' \approx P'$, by rule (PAR) follows $Q \mid R' \overset{\alpha'}{\twoheadrightarrow} Q' \mid R'$ and by (RES) $(\nu\tilde{c})(Q \mid R') \overset{\alpha'}{\twoheadrightarrow} (\nu\tilde{c})(Q' \mid R')$. Adding R (respectively R') in parallel can make the domain of the processes bigger, but we derive from Lemma A.2 that

$$\alpha \equiv_{(\tilde{\approx}, loc(P))} \alpha' \Longrightarrow \alpha \equiv_{(\tilde{\approx}, loc(P|R))} \alpha'$$

   (b) The case when R moves is similar to the previous case.

   (c) Suppose that for some fresh $\tilde{b}, \tilde{d}$, we have $P \equiv (\nu\tilde{b})(\overline{l\cdot a}\langle\tilde{v}\rangle \mid P_1)$ where $a \notin \{\tilde{b}\}$, and $R \equiv (\nu\tilde{d})(l\cdot a(\tilde{x}).R_1 \mid R_2)$. Then

$$(\nu\tilde{c})(P \mid R) \xrightarrow{\tau} (\nu\tilde{c}, \tilde{b})(P_1 \mid (\nu\tilde{d})(R_1\{\tilde{v}/\tilde{x}\} \mid R_2))$$

   and $P \xrightarrow{(\tilde{b})\overline{l\cdot a}\langle\tilde{v}\rangle} P_1$.
   By $P \approx Q$ we have that

$$Q \overset{\tau}{\twoheadrightarrow} Q_3 \xrightarrow{(\tilde{b})\overline{l\cdot a}\langle\tilde{v'}\rangle} Q_2 \overset{\tau}{\twoheadrightarrow} Q_1 \approx P_1$$

   and $(\tilde{b})\overline{l\cdot a}\langle\tilde{v}\rangle \equiv_{(\tilde{\approx}, loc(P))} (\tilde{b})\overline{l\cdot a}\langle\tilde{v'}\rangle$. By syntactical reasoning $Q_3 \equiv (\nu\tilde{b})(\overline{l\cdot a}\langle\tilde{v'}\rangle \mid Q_2)$, again with $a \notin \{\tilde{b}\}$.
   By $R_{loc(P)} \approx R'_{loc(P)}$ we have that $R \xrightarrow{l\cdot a(\tilde{v})} (\nu\tilde{d})(R_1\{\tilde{v}/\tilde{x}\} \mid R_2) = R_3$ implies (by syntactical reasoning)

$$R' \overset{\tau}{\twoheadrightarrow} (\nu\tilde{e})(l\cdot a(\tilde{y}).R'_1 \mid R'_2) \xrightarrow{l\cdot a(\tilde{v''})} (\nu\tilde{e})(R'_1\{\tilde{v''}/\tilde{y}\} \mid R'_2) \overset{\tau}{\twoheadrightarrow} R'_3 \approx R_3$$

   for some $v'' \equiv_{(\tilde{\approx}, loc(P|R))} v$, and $\tilde{e}$ fresh.

We can now derive

$$(\nu\tilde{c})(Q \mid R') \stackrel{\tau}{\twoheadrightarrow} (\nu\tilde{c})(Q_3 \mid (\nu\tilde{e})(l{\cdot}a(\tilde{y}).R'_1 \mid R'_2))$$

$$\stackrel{\tau}{\longrightarrow} (\nu\tilde{c},\tilde{b})(Q_2 \mid (\nu\tilde{e})(R'_1\{\tilde{v'}/\tilde{y}\} \mid R'_2)) \stackrel{\tau}{\twoheadrightarrow} (\nu\tilde{c},\tilde{b})(Q_1 \mid R_4)$$

and by Lemma A.7, $R_4 \approx R'_3$.
The cases where R performs a replicated input or an output are similar.

□

A.3 Proof of Soundness

We start stating some properties of the lts.

Lemma A.9 *(i)* $P \stackrel{l{\cdot}\mathsf{update}_p(\{U_1,\cdots,U_n\},(\chi)V)}{\longrightarrow} Q \iff P \equiv (\nu\tilde{c})(l{\cdot}\mathsf{update}_p(\chi,V).R' \mid R) \wedge Q \equiv (\nu\tilde{c})(R'\{U_1/\chi\} \mid \cdots \mid R'\{U_n/\chi\} \mid R)$. *(ii)* $P \stackrel{l{\cdot}\mathsf{run}_p}{\longrightarrow} Q \iff P \equiv l{\cdot}\mathsf{run}_p \mid R \wedge Q \equiv l{\cdot}0 \mid R$.

Proof. In both cases ($\Longrightarrow$) follows by induction on the derivation in the lts, ($\Longleftarrow$) follows by structural induction on P. □

Lemma A.10 *(i)* $P \stackrel{\tau}{\longrightarrow} P' \wedge loc(P) = loc(P') \implies (\forall D.\ wf(D,P) \implies (D,P) \to (D,P'))$. *(ii)* $P \stackrel{\tau}{\twoheadrightarrow} P' \wedge loc(P) = loc(P') \implies (\forall D.\ wf(D,P) \implies (D,P) \to^* (D,P'))$.

Proof. (i) Tau transitions in the lts are independent from D. The only case that needs attention is (Enter): in could be the case that in the lts is possible to perform such a transition, whereas in the reduction relation it is not (if $l \notin dom(D)$). The condition $loc(P) = loc(P')$ prevents this situation from happening. (ii) It is the transitive closure of the previous case. □

Lemma A.11 *If* $(D,P) \to (D',P')$ *then one of the following holds:*

1. $P \stackrel{\tau}{\longrightarrow} P'$ *and* $D' = D$.
2. $P \stackrel{l{\cdot}\mathsf{update}_p(\mathcal{U},(\chi)V)}{\longrightarrow} P'$ *where* $\begin{cases} D = \{l \mapsto T\} \uplus E \\ p(T) \rightsquigarrow\!\!\!\!\!\mapsto_{p,l,\chi,V} T', \Sigma \\ range(\Sigma) = \mathcal{U} \\ D' = \{l \mapsto T'\} \uplus E \end{cases}$
3. $P \stackrel{l{\cdot}\mathsf{run}_p}{\longrightarrow} P''$ *where* $\begin{cases} D = \{l \mapsto T\} \uplus E,\ D' = D \\ p(T) \rightsquigarrow\!\!\!\!\!\mapsto_{p,l,\Box X,\Box X} T, \Sigma \\ range(\Sigma) = \{\Box P_1, \cdots, \Box P_n\} \\ P' \equiv P'' \mid P_1 \mid \cdots \mid P_n \end{cases}$

Proof. By induction on the derivation of $(D,P) \to (D',P')$. There are six base cases, we show the case where the axiom used is (Update), which is the most interesting. Suppose that we have $p(T) \rightsquigarrow_{p,l,\chi,V} T', \{\sigma_1, \cdots, \sigma_n\}$ and therefore $(\{l \mapsto T\}, l{\cdot}\mathsf{update}_p(\chi,V).Q) \to (\{l \mapsto T'\}, Q\sigma_1 \mid \cdots \mid Q\sigma_n)$. By rule (Update) in the lts we have $Q \xrightarrow{l\cdot\mathsf{update}_p(\mathcal{U},(\chi)V)} Q'$ where $Q' \equiv Q\{U_1/\chi\} \mid \cdots \mid Q\{U_n/\chi\}$ for any $\mathcal{U} = \{U_1, \cdots, U_n\}$. By choosing a specific $\mathcal{U} = range(\Sigma)$ we conclude. □

Lemma A.12 *Let D, E be two disjoint stores, where $\Lambda = dom(D)$, $\Lambda' = dom(E)$ (and $\Lambda \cap \Lambda' = \emptyset$). We have*

$$D \equiv_{(\approx,\Lambda)} B \Longrightarrow D \uplus E \equiv_{(\approx,\Lambda\cup\Lambda')} B \uplus E$$

Proof. Follows by structural induction on D, applying every time Definition 3.10. The base case follows by induction on the derivation of $D \equiv_{(\approx,\Lambda)} B$ noticing that by Lemma A.2, $\Box P \equiv_{(\approx,\Lambda)} \Box Q \Longrightarrow \Box P \equiv_{(\approx,\Lambda\cup\Lambda')} \Box Q$. □

Lemma A.13 *Let $T \equiv_{(\approx,\Lambda)} S$ and $(\chi)V \equiv_{(\approx,\Lambda)} (\xi)V_1$. If $p(T) \rightsquigarrow_{p,l,\chi,V} T', \Sigma$ then $p(S) \rightsquigarrow_{p,l,\xi,V_1} S', \Sigma'$ where $T' \equiv_{(\approx,\Lambda)} S'$ and $range(\Sigma) \equiv_{(\approx,\Lambda)} range(\Sigma')$.*

Proof. By well-founded induction on the depth of nesting of selected nodes inside of T. The base case (when $p(T) = T$) requires a simple induction on the structure of T. The inductive step (where the nesting of selected nodes inside of T is $n+1$) follows again by structural induction. The only interesting case is the one for rule (Up), reported below for convenience:

$$\frac{\mathsf{match}(U\{l/\circlearrowleft, p/.\}, \chi) = \sigma \qquad V\sigma \rightsquigarrow_{p,l,\chi,V} V', \Sigma}{\mathsf{a}[\underline{U}] \rightsquigarrow_{p,l,\chi,V} \mathsf{a}[V'], \{\sigma\} \oplus \Sigma}$$

By definition of $\approx \Lambda$, the trees T and S have the same structure (up-to structural congruence), and can only differ in the definition of the scripted processes. Therefore $p(T)$ selects the same nodes as $p(S)$, up-to equivalence. If T has a selected node $\mathsf{a}[\underline{U}]$, then S has a selected node $\mathsf{a}[\underline{U'}]$, with $U \equiv_{(\approx,\Lambda)} U'$, where the nesting depth can be at most n. Since $(\chi)V \equiv_{(\approx,\Lambda)} (\xi)V_1$ and V, V_1 cannot contain selected nodes, if $\mathsf{match}(U\{l/\circlearrowleft, p/.\}, \chi) = \sigma$ then $\mathsf{match}(U'\{l/\circlearrowleft, p/.\}, \xi) = \sigma'$, and $V\sigma \equiv_{(\approx,\Lambda)} V_1\sigma'$, where the nesting depth in $V\sigma$ and $V_1\sigma'$ is at most n, which permits to use the inductive hypothesis to derive the conclusion. □

A.3.1 Proof of Theorem 3.3

Proof. We have to show that for all D such that (D,P) is well formed, $P \approx Q \Longrightarrow (D,P) \simeq_b (D,Q)$. In fact, we show a weaker property: the relation

$$\mathcal{S} = \{((D,P),(B,Q)) \mid P \approx Q \ \wedge \ D \equiv_{(\approx,\Lambda)} B\}$$

where $\Lambda = dom(D,B) = loc(P,Q)$, is included in $\simeq_b$.

1. Assume $(D,P)\downarrow_{l\cdot\mathsf{update}_p}$. By definition of barbs and by Lemma A.9 we have $P \xrightarrow{l\cdot\mathsf{update}_p(\mathcal{U},(\chi)V)} P'$, and since $loc(P') = loc(P)$, by $P \approx Q$ we have $Q \overset{l\cdot\mathsf{update}_p(\mathcal{U}',(\xi)V')}{\twoheadrightarrow} Q'$, with $loc(Q') = loc(Q)$. By Lemma A.10 and again Lemma A.9 and definition of barbs we conclude with $(B,Q)\Downarrow_{l\cdot\mathsf{update}_p}$.

2. Let $(D,P)\mathcal{S}(B,Q)$. We show under the hypothesis $P \approx Q$ and $D \equiv_{(\approx,\Lambda)} B$, that $(D,P) \to (D',P') \Longrightarrow (B,Q) \to^* (B',Q')$ and $(D',P')\mathcal{S}(B',Q')$. If $(D,P) \to (D',P')$, by Lemma A.11 we have three cases:

   (a) $P \xrightarrow{\tau} P'$ and $D' = D$. By the hypothesis $P \approx Q$, we have $Q \overset{\tau}{\twoheadrightarrow} Q' \approx P'$, where $loc(Q') = loc(Q)$. By Lemma A.10 we get $(B,Q) \to^* (B,Q')$, and therefore $(D,P')\mathcal{S}(B,Q')$.

   (b) $P \xrightarrow{l\cdot\mathsf{update}_p(\mathcal{U},(\chi)V)} P'$ where $D = \{l \mapsto T\} \uplus E$, $p(T) \rightsquigarrow_{p,l,\chi,V} T', \Sigma$ and $D' = \{l \mapsto T'\} \uplus E$. Noting that $loc(P) = loc(P')$, by hypothesis $Q \overset{\tau}{\twoheadrightarrow} Q_1 \xrightarrow{l\cdot\mathsf{update}_p(\mathcal{U}',(\xi)V')} Q_2 \overset{\tau}{\twoheadrightarrow} Q' \approx P'$, for $\Lambda = loc(Q') = loc(P')$ and $(\chi)V \equiv_{(\approx,\Lambda)} (\xi)V'$, $\mathcal{U} \equiv_{(\approx,\Lambda)} \mathcal{U}'$. By Lemma A.10 we get $(B,Q) \to^* (B,Q_1)$, and by Lemma A.9 we get $Q_1 \equiv (\nu\tilde{c})(l\cdot\mathsf{update}_p(\xi,V').R' \mid R)$ where $Q_2 \equiv (\nu\tilde{c})(R'\{U'_1/\xi\} \mid \cdots \mid R'\{U'_n/\xi\} \mid R)$ for $\mathcal{U}' = \{U'_1,\cdots,U'_n\}$. By definition of $\to$, $(B,Q_1) \to (B',Q'_2)$, where by hypothesis $B = \{l \mapsto S\} \uplus E'$ and $S \equiv_{(\approx,\Lambda)} T$, and $B' = \{l \mapsto S'\}$ and $Q'_2 \equiv (\nu\tilde{c})(R'\{U''_1/\xi\} \mid \cdots \mid R'\{U''_n/\xi\} \mid R)$ for $p(B) \rightsquigarrow_{p,l,\xi,V'} S', \Sigma'$ and $\mathcal{U}'' = \{U''_1,\cdots,U''_n\} = range(\Sigma')$. By Lemma A.13, $B' \equiv_{(\approx,\Lambda)} D'$ and $range(\Sigma) \equiv_{(\approx,\Lambda)} \mathcal{U}''$. By repeatedly using closure under parallel composition of $\approx$, we get $Q'_2 \approx Q_2$. By $Q_2 \overset{\tau}{\twoheadrightarrow} Q' \approx P'$, and by transitivity of $\approx$, we get $Q'_2 \overset{\tau}{\twoheadrightarrow} Q'' \approx P'$, which together with $B' \equiv_{(\approx,\Lambda)} D'$ gives us $(D',P')\mathcal{S}(B',Q')$.

   (c) $P \xrightarrow{l\cdot\mathsf{run}_p} P''$ where $D = \{l \mapsto T\} \uplus E$, $P' \equiv P'' \mid R$ and $R \equiv P_1 \mid \cdots \mid P_n$ for $p(T) \rightsquigarrow_{p,l,\Box X,\Box X} T, \Sigma$ and $range(\Sigma) = \{P_1,\cdots,P_n\}$, and $D' = D$. By $P \approx Q$ we get $Q \overset{\tau}{\twoheadrightarrow} Q_1 \xrightarrow{l\cdot\mathsf{run}_p} Q_2 \overset{\tau}{\twoheadrightarrow} Q'' \approx P''$. By Lemma A.10 we get $(B,Q) \to^* (B,Q_1)$. By Lemma A.9, $Q_1 \equiv l\cdot\mathsf{run}_p \mid Q''_1$ and $Q_2 \equiv l\cdot 0 \mid Q''_1$. By definition of $\to$ and by Lemma A.13 follows that there is R' such that $(B,Q_1) \to (B,Q_2 \mid R')$, where by repeatedly exploiting the closure of $\approx$ under parallel composition we have $R \approx R'$. Again by Lemma A.10 we get $(B,Q_2 \mid R') \to^* (B,Q'' \mid R')$. From $P'' \approx Q''$ and $R \approx R'$ we get $P'' \mid R \approx Q'' \mathcal{R}'$, and since $loc(R) = loc(R') = \{l\} \in loc(P)$ we get $(D,P')\mathcal{S}(B,Q')$.

3. Suppose $(D,P)\,\mathcal{S}\,(B,Q)$. Given an arbitrary $C[-] = (E, C'[-])$, by the hypothesis $P \approx Q$ and by Theorem 3.2 (congruence of $\approx$) we have $C'[P] \approx C'[Q]$. In order for $C[(D,P)]$ to be well-formed it must be the case that for some Λ', $\Lambda' = loc(C'[P]) = dom(D \uplus E)$. By the hypothesis $D \equiv_{(\approx,\Lambda)} B$ and by Lemma A.12 we have $D \uplus E \equiv_{(\approx,\Lambda')} B \uplus E$, which together with $C'[P] \approx C'[Q]$ implies $C[(D,P)]\,\mathcal{S}\,C[(B,Q)]$. $\square$