Modelling Dynamic Web Data ⋆

Philippa Gardner, Sergio Maffeis *

Department of Computing
180 Queen's Gate, South Kensington Campus,
Imperial College London SW7 2AZ, UK.

Abstract

We introduce the X$d\pi$ calculus, a peer-to-peer model for reasoning about *dynamic* web data. Web data is not just stored statically. Rather it is referenced indirectly, for example using hyperlinks, service calls, or scripts for dynamically accessing data, which require the complex coordination of data and processes between sites. The X$d\pi$ calculus models this coordination, by integrating the XML data structure with process orchestration techniques associated with the distributed pi-calculus. We study behavioural equivalences for X$d\pi$, to analyze the various possible patterns of data and process interaction.

Key words: Dynamic web data, process orchestration, π-calculus, XML, web services, semi-structured data

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.

⋆ This paper is an extended and revised version of [16,17].

* Corresponding author is Maffeis, who acknowledges support from Microsoft Research Cambridge and an EPSRC e-Science Grant.

Email addresses: pg@doc.ic.ac.uk (Philippa Gardner), maffeis@doc.ic.ac.uk (Sergio Maffeis).

Peer-to-peer data management systems are decentralized 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 of running processes. 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 [32], hyperlinks, forms, and scripting.

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 specialized orchestration tools. In contrast, distributed process interaction—describing both communication and coordination—is central to our model, and is inspired by the current research on distributed process calculi. 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 data model, and π-like processes [28,35,22] for modelling process interaction, process migration, and interaction with data. The data model is a basic model of semi-structured data—unordered labelled trees with explicit pointers (URLs) for referring to other parts of the network—with embedded processes for querying and updating such data: for example, a hyperlink with an external pointer referring to another site, and a light-weight trusted process for retrieving information associated with the link.

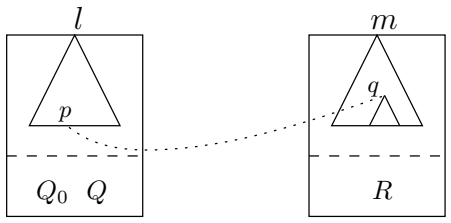
The embedding of processes in data provides many alternative patterns for exchanging information on the web. We study behavioural equivalences for X$d\pi$. In particular, we define when two processes are equivalent in such a way that, when the processes are put in the same position in a network, the 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 translate X$d\pi$ into a simpler calculus (Core X$d\pi$), where the location structure has been pushed inside the data and processes. This translation technique, also found in [10], enables us to separate reasoning about processes from reasoning about data and networks. Finally, we define process equivalence and study examples. In a companion paper [26], we analyze alternative notions of network and process equivalence, and give a proof method for process equivalence based on

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

labelled-bisimulation.

1.1 A Simple Example

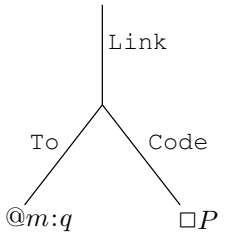
As an illustrative example, we use a *hyperlink* which consists of an external pointer to a remote location, and a process which 'follows' the pointer and gets the target subtree. Consider the picture



This picture represents two distinct locations l and m. Each location contains a data tree (the triangles) and a working space of active processes (the processes Q_0, Q and R). Location l contains a hyperlink at the node identified by path p, with an external pointer to data identified by path q at location m. In the working space of location l, a process can activate the hyperlink (click on the hyperlink), firing a request to m to copy the tree identified by path q and write the result to p at l. Below we give specific processes Q_0, Q and R, to show how activating a hyperlink and firing a request might be modelled.

The hyperlink, written in both XML notation (LHS) and the notation used in this paper (RHS), has the form:

```
<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 activates the mechanism to fetch the subtree at q from m. The process Q_0 has the form $\mathsf{run}_{p/\texttt{Link/Code}}$ which triggers the execution of P in the working space. Process P has the form

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

The read command reads the external pointer at $p/\texttt{Link/To}$ in the tree, substitutes the values m and q for the parameters x and y in the continuation,

and evolves to the output process $\overline{load}\langle m, q, p\rangle$. This output process records the target location m, the target path q, and the position p where the result tree will go. This output process will call a corresponding input process inside Q using π-calculus interaction.

One possible candidate for Q is

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

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 unsubtle, and does not mimic what happens in practice. Instead, we use 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: a process $\mathsf{go}\, m.\overline{get}\langle q, l, c\rangle$ 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 $\overline{get}$, copies the appropriate subtree, and returns it to the unique channel c at l. Using our definition of process equivalence, we show that the processes Q_s and Q are interchangeable.

1.2 Related Work

Models of distributed systems and distributed data access have, until quite recently, been entirely separate research topics. Data models tend to describe data manipulation without modelling the underlying distribution layer. In contrast, process models describe exchanges of data and processes between peers, but tend to abstract from the actual data. Our work on the X$d\pi$ calculus is the first attempt to integrate the study of mobile processes and semi-structured data for Web-based data-sharing applications, and is characterized by its emphasis on dynamic data.

The X$d\pi$ calculus is probably most related to the Active XML model for data integration developed independently by Abiteboul *et al.* [4]. They introduce

a peer-to-peer model where each peer contains a data model similar to ours except that only service calls can be scripted in documents, only documents (and hence service calls) can migrate between locations, only service definitions can be in the working space, the data resulting from a service call must return to the position of that service call, and the underlying distribution layer is not modelled. In contrast, our process 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 (a service definition) to the university to make the assessment.

The database and process algebra communities have studied several data models for the web. Our tree model extends those found in [3] and [11] with dynamic behaviour and with a stronger emphasis on links. Several distributed query languages, such as [30,25,8,32], extend traditional query languages with facilities for distribution awareness. Our approach is closest to the one of Sahuguet and Tannen [32], who introduce the ubQL query language for streaming large amounts of distributed data, partly motivated by ideas from the π-calculus.

The X$d\pi$ processes are based on an extension of the asynchronous π-processes of [22,23] with explicit locations, along the lines of $d\pi$ [21]. Core X$d\pi$ uses the polyadic synchronization of ${}^e\pi$ [10]. In [18] we propose a proof technique for the equivalence of X$d\pi$ processes, based on higher-order bisimulation techniques studied for example in [36,14,33]. In [26], we study alternative notions of network and process equivalence, and give a proof method based on first-order bisimulation which exploits the technique of [33,24] for translating higher-order to first-order labels. Process calculi have been used to study several web features, such as security properties of web services [20], mobile resources [19], and a sketch of a distributed query language [31]. Bierman and Sewell [7] have extended a small functional language for XML with π-primitives in order to program Home Area Network devices. We believe that X$d\pi$ will provide a useful, but challenging, framework in which to study such applications.

2 A Model of Dynamic Web data

We model a peer-to-peer system as a set of interconnected locations (*networks*), where the content of each location consists of a term (the *tree*) representing the local data and a term (the *process*) representing both the services provided by the peer, the agents executing on behalf of other peers, and local agents waiting for answers from remote queries. Processes can query and update the local data, communicate with each other through named *channels*, and migrate to other peers. Migration should not be interpreted as a

prescription on possible implementations, but rather as a conceptual device to distinguish clearly local from non-local interaction, useful to express both remote communication and more advanced protocols involving code mobility.

2.1 Trees

Our data model is an unordered edge-labelled rooted tree, with leaves containing internal and external pointers, and with embedded (static) processes which may be activated by a process in the working space. Semi-structured data models are often unordered [3], in contrast with the ordered trees of XML documents. The choice of using edge-labelled rooted trees, compared with node-labelled forests, is merely a matter of style. Another choice was to embed processes and pointers throughout the tree, rather than just at the leaves. The ideas in this paper do not depend on these choices. The set of trees is generated from a set of *edge labels* $\mathcal{A}$ with elements denoted by $\mathtt{a}, \mathtt{b}, \mathtt{c}$, a set of *path expressions* $\mathcal{E}$ with elements p, q used to identify subdata within a data tree, and a set of *locations* $\mathcal{L}$ with generic elements denoted by l, m and a special element $\circlearrowleft$ which denotes the enclosing location. The set of data trees $\mathcal{T}$ is given by

$$
\begin{array}{rcll}
T & ::= & 0 & \text{empty rooted tree} \\
 & | & T \,|\, T & \text{composition of trees, joining the roots} \\
 & | & \mathtt{a}[\,T\,] & \text{edge labelled } a \text{ with subtree } T \\
 & | & \mathtt{a}[\,\Box P\,] & \text{edge labelled } a \text{ with scripted process } \Box P \\
 & | & \mathtt{a}[\,@l{:}p\,] & \text{edge labelled } a \text{ with pointer } @l{:}p
\end{array}
$$

The composition of trees is total, analogous to tag labels not being unique in XML. A *scripted process* $\Box P$ is a static process awaiting a command to run. A pointer $@l{:}p$ refers to data identified by path expression p at location l. Processes and path expressions are described below. We have a structural congruence on trees, which states that trees are unordered and scripted processes are identified up to the structural congruence for processes (see Table 1).

2.2 Processes

We use the process orchestration techniques associated with the π-calculus to coordinate the movement of data and processes between sites. We model the exchange of data and channels between processes using the π-calculus communication, model private channel creation using the restriction operator, and

extend the π-processes with an explicit migration primitive for moving processes between locations. We model the local interaction between the data tree and processes in the working space, by a simple update operation for rewriting data and a simple run command for activating the processes embedded in the data.

Let $a, b, c \in \mathcal{C}$ denote channel names or channels variables. Channel names are partitioned into *public* and *session* channels, given by the disjoint sets $\mathcal{C}_P$ 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 therefore be restricted. Session channels are used for process interaction, and are not free in the scripted processes occurring in data. Values consist of channel names or all forms of data: trees, scripted processes, locations and path expressions. Formally, the set $\mathcal{V}$ of values (ranged over by $u, v, \ldots$) is given by $u, v ::= a \mid T \mid \Box P \mid l \mid p$. Let $\tilde{v}$ denote a tuple of values, and let $\tilde{x}$ denote a tuple of variables over such values. The set of processes $\mathcal{P}$ is given by

$$
\begin{array}{rcll}
P & ::= & 0 & \text{the nil process} \\
& \mid & P \,|\, P & \text{composition of processes} \\
& \mid & (\nu a)P & \text{declare new channel name } a \\
& \mid & \overline{a}\langle\tilde{v}\rangle & \text{output values } \tilde{v} \text{ on channel } a \\
& \mid & a(\tilde{x}).P & \text{input parameterized by distinct variables } \tilde{x} \\
& \mid & !a(\tilde{x}).P & \text{replication of an input process} \\
& \mid & \mathsf{go}\, l.P & \text{migrate to } l, \text{ continue as } P \\
& \mid & \mathsf{run}_p & \text{run the processes identified by path expression } p \\
& \mid & \mathsf{update}_p(\chi, V).P & \text{update command, described below}
\end{array}
$$

The channel name a is bound in $(\nu a)P$, and the distinct variables $\tilde{x}$ are bound in $a(\tilde{x}).P$ and $!a(\tilde{x}).P$. We assume a simple sorting discipline on channels, to ensure that the number of values sent along a channel matches the number of variables expected to receive those values. We write $\overline{a}$ and $a.P$ when the tuples of values and variables are empty.

The *migration* primitive $\mathsf{go}\, l.P$ is common in calculi for describing distributed systems; see for example [21]. 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 $\overline{l{\cdot}c}\langle\tilde{v}\rangle$ to denote the output of a message addressed to channel c at location l. We will in fact use such terms in our study of process equivalence in Section 7.

The command run_p activates the scripted processes identified by the path expression p in the local tree.

The update command $\mathsf{update}_p(\chi, V).P$ is a simple rewriting command used to locally update the data tree, by pattern-matching the data identified by path expression p to the pattern χ. Patterns have the form

$$\begin{array}{rcll} \chi & ::= & X & \text{matches tree } T\text{; assigns } T \text{ to variable } X \\ & | & @x{:}y & \text{matches pointer } @l{:}p\text{; assigns } l \text{ and } p \text{ to } x \text{ and } y \\ & | & \Box X & \text{matches embedded process } \Box P\text{; assigns } P \text{ to } X \end{array}$$

The data term V ranges over trees, pointers and scripted processes, extended with variables associated with the patterns. The variables in χ are bound in V and P. The update command finds all the values V_i given by the path expression p, and pattern-matches these values with χ to obtain the substitution σ_i when it exists. For each successful pattern-matching, it replaces the V_i with $V\sigma_i$ and starts $P\sigma_i$ in parallel [2].

We may derive simple commands from this general update command, such as the standard read_p, copy_p, cut_p and paste_p commands:

$$\begin{array}{rcll} \mathsf{read}_p(@x{:}y).P & \stackrel{\text{def}}{=} & \mathsf{update}_p(@x{:}y, @x{:}y).P & \left\{\begin{array}{l}\text{read the pointer at } p,\\ \text{use its location and path in } P\end{array}\right. \\ \mathsf{copy}_p(X).P & \stackrel{\text{def}}{=} & \mathsf{update}_p(X, X).P & \text{copy the tree at } p\text{; use it in } P \\ \mathsf{cut}_p(X).P & \stackrel{\text{def}}{=} & \mathsf{update}_p(X, 0).P & \text{cut the tree at } p\text{; use it in } p \\ \mathsf{paste}_p\langle T\rangle.P & \stackrel{\text{def}}{=} & \mathsf{update}_p(X, X \,|\, T).P & \left\{\begin{array}{l}\text{where } X \text{ is not free in } T \text{ or } P,\\ \text{paste tree } T \text{ at } p\text{; evolve to } P\end{array}\right. \end{array}$$

The structural congruence on processes is similar to that 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.

2.3 Networks

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

$$N ::= 0 \;\mid\; N \,|\, N \;\mid\; l\,[\,T \parallel P\,] \;\mid\; (\nu c)N$$

[2] Instead of continuing with process $P\sigma_i$ for each i, an alternative choice would have been to continue with $P\sigma$, where σ is the substitution obtained by joining all the values obtained during the update pattern-matching. Our techniques are robust with respect to this choice. We have chosen a simple update rewriting command for the interaction between processes and data. In future, we envisage combining our work with an imperative language for updating trees [9].

(Trees)	$V \equiv V' \Rightarrow \mathsf{a}[\,V\,] \equiv \mathsf{a}[\,V'\,]$	
(Values)	$v' \equiv w' \wedge \tilde{v} \equiv \tilde{w} \Rightarrow v', \tilde{v} \equiv w', \tilde{w}$	$P \equiv Q \Rightarrow \Box P \equiv \Box Q$
(Processes)	$(\nu c)(\nu d)P \equiv (\nu d)(\nu c)P$	$(\nu c)0 \equiv 0$
	$c \notin fn(P) \Rightarrow P \,|\, (\nu c)Q \equiv (\nu c)(P \,|\, Q)$	$\tilde{v} \equiv \tilde{w} \Rightarrow \overline{c}\langle\tilde{v}\rangle \equiv \overline{c}\langle\tilde{w}\rangle$
	$V \equiv V' \Rightarrow \mathsf{update}_p(\chi, V).P \equiv \mathsf{update}_p(\chi, V').P$	
(Networks)	$(\nu c)(\nu d)N \equiv (\nu d)(\nu c)N$	$(\nu c)0 \equiv 0$
	$c \notin fn(N) \Rightarrow N \,|\, (\nu c)M \equiv (\nu c)(N \,|\, M)$	
	$T \equiv S \wedge P \equiv Q \Rightarrow l\,[\,T \parallel P\,] \equiv l\,[\,S \parallel Q\,]$	
	$l\,[\,T \parallel (\nu c)P\,] \equiv (\nu c)l\,[\,T \parallel P\,]$	

Table 1
Structural congruence for X$d\pi$ is the least congruence satisfying alpha-conversion, the commutative monoidal laws for $(0, |)$ on trees, processes and networks, and the axioms reported above.

The network composition $N_1 \,|\, N_2$ is *partial* in that the location names associated with N_1 and N_2 must be disjoint. The *location* $l\,[\,T \parallel P\,]$ denotes location l containing a tree T and a process P. It is well-formed when the tree and process are closed, and the tree contains no free session channels. In our hyperlink example, 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 $(\nu c)N$. The structural congruence for networks is defined in Table 1, and is analogous to that given for processes.

2.4 Path Expressions

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. We regard a path p as a function from trees to sets of nodes (up to structural congruence): $p(T)$ denotes the tree T where the nodes identified by p are selected. For simplicity we do not show node identifiers explicitly, but we underline the selected nodes.

In this paper, we use a very simple subset of XPath expressions [27], where "a" denotes a step along an edge labelled a, "/" denotes path composition, ".." a step back, "//" any node, and ".", which can appear only inside trees, denotes the path from the root of the tree to the current node. For example, in $\mathsf{a}[\,\underline{\mathsf{a}[\,S\,]}\,|\,\mathsf{b}[\,S'\,]\,|\,\mathsf{c}[\,T'\,]\,]$ we have underlined the nodes selected by path $//\mathsf{a}$.

Pattern matching of XML-like values is an active research topic. In this paper, we will just consider a very basic form of pattern matching, which merely matches trees, pointers and scripted processes. Pattern matching is carried out by the partial function $\mathsf{match}(-,-)$ which takes two arguments: the first is either a tree, or a pointer, or a scripted process, and the second is a pattern. If the first argument matches the pattern in the second, match returns a substitution binding the variables occurring in the pattern. The definition of match is:

$$\begin{aligned} \mathsf{match}(T, X) &= \{T/X\} \\ \mathsf{match}(@l{:}p, @x{:}y) &= \{l, p/x, y\} \\ \mathsf{match}(\Box P, \Box X) &= \{\Box P/\Box X\} \end{aligned}$$

2.6 Reduction and Update 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. The reduction axioms are reported in Table 2. Reduction is closed under structural congruence and reduction contexts, which are given by

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

There are two rules for process movement between locations: rule (Stay) describes the case where the process is already at the target location, and rule (Go) allows a process $\mathsf{go}\, l.P$ at m to leave m and reach l. This rule depends on the existence of location l. In contrast, the migration rule for $d\pi$ [21] always assume that migration is possible. Our choice has a profound effect on the behavioural equivalences studied in Section 7. In future work, we will associate some security check to this operation.

The rules (Com) and (!Com) describe standard π-calculus interaction.

The generic (Update) rule provides interaction between processes and data, and depends on the auxiliary updating function $\rightsquigarrow\!\!\!\!\mid$ also given in Table 2. Using path expression p, the rule selects for update some data in T, denoted by $p(T)$, it applies the updating function $\rightsquigarrow\!\!\!\!\mid$ to $p(T)$ in order to obtain the new data T' and the set of bindings Σ, and finally spawns a copy of the continuation $P\sigma$ for each $\sigma \in \Sigma$. This function $\rightsquigarrow\!\!\!\!\mid$ is parameterized by p, l, χ, V. It essentially matches the underlined data with pattern χ to obtain substitution σ (when it exists), updates the data with $V\sigma$ and records σ. Rule (Up) is subtle. It matches any selected (underlined) U in $p(T)$ with χ, to obtain substitution σ; when

Reduction axioms:

$$\text{(Stay)} \quad m\,[\,T \parallel Q \mid \mathsf{go}\ m.P\,] \searrow m\,[\,T \parallel Q \mid P\,]$$

$$\text{(Go)} \quad m\,[\,T \parallel Q \mid \mathsf{go}\, l.P\,] \mid l\,[\,S \parallel R\,] \searrow m\,[\,T \parallel Q\,] \mid l\,[\,S \parallel R \mid P\,]$$

$$\text{(Com)} \quad l\,[\,T \parallel \overline{c}\langle\tilde{v}\rangle \mid c(\tilde{z}).P \mid Q\,] \searrow l\,[\,T \parallel P\{\tilde{v}/\tilde{z}\} \mid Q\,]$$

$$\text{(Com!)} \quad l\,[\,T \parallel \overline{c}\langle\tilde{v}\rangle \mid\ !c(\tilde{z}).P \mid Q\,] \searrow l\,[\,T \parallel\ !c(\tilde{z}).P \mid P\{\tilde{v}/\tilde{x}\} \mid Q\,]$$

$$\text{(Update)} \quad \frac{p(T) \rightsquigarrow_{p,l,\chi,V} T', \{\sigma_1, \cdots, \sigma_n\}}{l\,[\,T \parallel \mathsf{update}_p(\chi, V).P \mid Q\,] \searrow l\,[\,T' \parallel P\sigma_1 \mid \ldots \mid P\sigma_n \mid Q\,]}$$

$$\text{(Run)} \quad \frac{p(T) \rightsquigarrow_{p,l,\Box X,\Box X} T, \{\{\Box P_1/\Box X\}, \cdots, \{\Box P_n/\Box X\}\}}{l\,[\,T \parallel \mathsf{run}_p \mid Q\,] \searrow l\,[\,T \parallel P_1 \mid \ldots \mid P_n \mid Q\,]}$$

Updating function:

$$\text{(Zero)} \quad 0 \rightsquigarrow_\Theta 0, \emptyset \qquad \text{(Link)} \quad \mathsf{a}[\,@m{:}q\,] \rightsquigarrow_\Theta \mathsf{a}[\,@m{:}q\,], \emptyset$$

$$\text{(Script)} \quad \mathsf{a}[\,\Box Q\,] \rightsquigarrow_\Theta \mathsf{a}[\,\Box Q\,], \emptyset \qquad \text{(Node)} \quad \frac{T \rightsquigarrow_\Theta T', \Sigma}{\mathsf{a}[\,T\,] \rightsquigarrow_\Theta \mathsf{a}[\,T'\,], \Sigma}$$

$$\text{(Par)} \quad \frac{T \rightsquigarrow_\Theta T', \Sigma_1 \qquad S \rightsquigarrow_\Theta S', \Sigma_2}{T \mid S \rightsquigarrow_\Theta T' \mid S', \Sigma_1 \oplus \Sigma_2}$$

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

Table 2
Reduction axioms and updating function. The reduction relation on networks is the smallest relation closed with respect to reduction contexts, structural congruence and the reduction axioms. In the updating function, Σ is a multiset of substitutions, $\oplus$ denotes multiset union, and $\Theta = p, l, \chi, V$ are the parameters of an update or run command.

σ exists, it continues updating $V\sigma$, which might also contain other selected (underlined) nodes, until we obtain the data V' and the set of substitutions Σ; it then replaces U with V' and adds to Σ the substitution $\sigma\{l/\circlearrowleft, p/.\}$, where any reference to the current location and path is substituted with the actual values l and p.

We give an example illustrating the derived cut command:

$$l\,[\,\mathsf{c}[\,\mathsf{a}[\,T\,] \mid \mathsf{a}[\,T'\,] \mid \mathsf{b}[\,S\,]\,] \parallel \mathsf{cut}_{\mathsf{c}/\mathsf{a}}(X).P\,] \searrow$$

$$l\,[\,\mathsf{c}[\,\mathsf{a}[\,0\,] \mid \mathsf{a}[\,0\,] \mid \mathsf{b}[\,S\,]\,] \parallel P\{T/X\} \mid P\{T'/X\}\,]$$

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. The next example illustrates run and the substitution of local references:

$$S = \mathsf{a}[\,\mathsf{b}[\,\Box\mathsf{go}\, m.\mathsf{go}\ \circlearrowleft.Q\,] \mid \mathsf{b}[\,\Box\mathsf{cut}_{./../\mathsf{c}}(X).P\,]\,]$$

$$l\,[\,S \parallel \mathsf{run}_{\mathtt{a/b}}\,] \searrow l\,[\,S \parallel \mathsf{go}\,m.\mathsf{go}\,l.Q \mid \mathsf{cut}_{\mathtt{a/b/../c}}(X).P\,]$$

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.

Remark 2.1 *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 it 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, if $Q = \mathsf{update}_{//}(X, 0).P$*

$$l\,[\,\mathtt{a}[\,\mathtt{b}[\,T\,]\,] \parallel Q\,] \searrow l\,[\,\mathtt{a}[\,0\,] \parallel P\{\mathtt{b}[\,T\,]/X\}\,] \quad \textit{top-down}$$
$$l\,[\,\mathtt{a}[\,\mathtt{b}[\,T\,]\,] \parallel Q\,] \searrow l\,[\,\mathtt{a}[\,0\,] \parallel P\{\mathtt{b}[\,0\,]/X\} \mid P\{T/X\}\,] \quad \textit{bottom-up}$$

because first $\mathtt{a}[\,\mathtt{b}[\,T\,]\,]$ *becomes* $\mathtt{a}[\,\mathtt{b}[\,0\,]\,]$ *giving* $P\{T/X\}$*, and then* $\mathtt{a}[\,\mathtt{b}[\,0\,]\,]$ *becomes* $\mathtt{a}[\,0\,]$*, giving* $P\{\mathtt{b}[\,0\,]/X\}$*.*

3 Dynamic Web Data at Work

3.1 Web Services

In the introduction, we described the hyperlink example. Here we generalize this example to arbitrary web services. Web services have sometimes been defined as "web sites for computers" or, in other words, language-independent interfaces to web sites, which function similarly to remote procedure calls. We define a web service c with parameters $\tilde{z}$, body B (a concatenation of prefixes), and type of result specified by the distinct variables $\tilde{w}$ bound by B:

$$\mathsf{Def}\ c(\tilde{z})\ \mathsf{as}\ B\ \mathsf{out}\ \langle\tilde{w}\rangle \stackrel{\text{def}}{=} !c(\tilde{z}, l, x).\, B.\, \mathsf{go}\, l.\, \overline{x}\langle\tilde{w}\rangle$$

where l and x are fixed parameters (not in $B, \tilde{w}$) which are used to specify the return location and channel. For example, process R described in the introduction can be written $\mathsf{Def}\ get(q)\ \mathsf{as}\ \mathsf{copy}_q(X)\ \mathsf{out}\ \langle X\rangle$.

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 $\tilde{w}$:

$$l{\cdot}\mathsf{Call}\ m{\cdot}c\langle\tilde{v}\rangle\ \mathsf{ret}\ (\tilde{w}).Q \stackrel{\text{def}}{=} (\nu b)(\mathsf{go}\, m.\overline{c}\langle\tilde{v}, l, b\rangle \mid b(\tilde{w}).Q)$$

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 running at l can be given by

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

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

$$l{\cdot}\mathsf{Sub}\ m{\cdot}c\langle\tilde{v}\rangle\ \mathsf{ret}\ (\tilde{w}).Q \stackrel{\text{def}}{=} (\nu b)(\mathsf{go}\, m.\overline{c}\langle\tilde{v}, l, b\rangle \mid !b(\tilde{w}).Q)$$

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.

3.2 XLink Base

We look at a refined example of the use of linking, along the lines of XLink. XLink is an XML standard for describing hyperlinks, intended in a very general sense. Simplifying, we can say that an XLink is an entity which denotes a set of "sources" and a set of "destinations", and which describes a relation between those two sets. An HTML tag $<\texttt{A href} = \texttt{URL}>$ ... can be interpreted as an XLink with a single source (the associated URL), a single implicit destination (the current browser window) and a implicit relation which is "get a document from the source and display it at the destination".

As we have mentioned, links specify both of their endpoints, and therefore can be stored in some external repository, for example

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

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

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:

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

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 $@l{:}p$.

$$
\begin{array}{l}
\mathsf{Def}\ xls(l,p)\ \mathsf{as}\ B\ \mathsf{out}\ \langle x, y, \Box X\rangle \\
B = \ \mathsf{copy}_{p_1}(@x{:}y).\mathsf{copy}_{p_2}(\Box X) \\
p_1 = \ \texttt{XLinkBase/XLink[From[}\,@l{:}p\,\texttt{]]/To} \\
p_2 = \ \texttt{XLinkBase/XLink[From[}\,@l{:}p\,\texttt{]|To[}\,@x{:}y\,\texttt{]]/Code}
\end{array}
$$

In p_1 we use the XPath syntax `XLink[From[`$@l{:}p$`]]/To` to identify the node `To` which is a son of node `XLink` and a sibling of `From[`$@l{:}p$`]`; similarly for p_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. The simplest and more common example of a form is the interface of your favourite search engine: it consist of a text field where the user can write the search criteria and a button that, when clicked, causes the input text to be sent to a server to execute the search.

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

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

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

$$s\,[\,S \parallel !handler(X).P|...\,]$$

This example suggests 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.

In the hyperlink example of the introduction, we have stated that the processes Q and Q_s basically have the same intended behaviour in any context containing R. 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. Then, we indicate how to translate X$d\pi$ into another (equivalent) calculus, called Core X$d\pi$, where it is easier to separate reasoning about processes from reasoning about data. Finally, we define process equivalence on Core X$d\pi$ terms.

4.1 Network Equivalence

We apply a standard technique for reasoning about processes distributed between locations to our non-standard setting. We define a *barbed congruence* relation between networks which is reduction-closed, closed with respect to reduction contexts, and which satisfies an additional *observation relation* described using *barbs*. In our case, a barb describes the path where an update command can affect the data.

Definition 4.1 *A* barb *has the form $l{\cdot}p$, where l is a location name and p is a path. The* observation relation*, denoted by $N \downarrow_{l\cdot p}$, is a binary relation between Xdπ-networks and barbs defined by*

$$N \downarrow_{l\cdot p} \quad \textit{iff} \quad \exists_{C,T,\chi,U,P,Q}.\ N \equiv C[l\,[\,T \parallel \mathsf{update}_p(\chi, U).P \mid Q)\,]]$$

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

$$N \Downarrow_{l\cdot p} \quad \textit{iff} \quad \exists N'.\, N \searrow^* N' \wedge N' \downarrow_{l\cdot p}$$

Observing a barb corresponds to observing at what points in some data tree a process has the capability to read or write data. Notice that a barb $l{\cdot}p$ gives no information on *how* the data is modified, and ignores run commands. This additional information can be observed indirectly using contexts.

Definition 4.2 *Barbed congruence ($\cong$) is the largest symmetric relation $\mathcal{R}$ on Xdπ-networks such that $N\,\mathcal{R}\,M$ implies*

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

In the companion paper [26], we have studied alternative notions of network equivalence based on different observation relations, looking for example at the shape of tree data, at the presence of outputs, and at the existence of locations. These alternative notions coincide with each other, and yield an equivalence relation which is more liberal than the one presented here. We study this specific equivalence in order to distinguish processes based on their attempts to interact with the data tree: that is, equivalent processes have equivalent permissions to access data.

4.2 Examples

Our first example illustrates that network equivalence does not imply that the initial data trees need be equivalent.

Example 4.1 *Consider the networks N and M given by*

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

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

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

Similarly, network equivalence does not imply that the initial processes need be structurally congruent.

Example 4.2 *Consider the process $\mathtt{xch}(T_1, T_2)$ defined below, which continuously replaces T_1 with T_2 and vice versa.*

$$\mathtt{xch}(T_1, T_2) = (\nu c)(\overline{c} \mid !c.\mathsf{update}_p(X, T_1).\mathsf{update}_p(X, T_2).\overline{c})$$

For all T_1, T_2 and T we have $l\,[\,T \parallel \mathtt{xch}(T_1, T_2)\,] \cong l\,[\,T \parallel \mathtt{xch}(T_2, T_1)\,]$ because the processes have the same barbs, and if T contains a subtree at p, they can simulate each other. Yet, $\mathtt{xch}(T_1, T_2) \not\equiv \mathtt{xch}(T_2, T_1)$.

Our next example shows that equivalence is sensitive to the interactions with data also when they have no effect.

Example 4.3 *A minimal example of non-equivalence is given by*

$$l\,[\,T \parallel \mathsf{update}_T(X, X).0\,] \ncong l\,[\,T \parallel 0\,]$$

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

above would be equivalent if we chose instead the structure of trees as the observable (see [26]).

We conclude this section with a web service related example.

Example 4.4 *We now place our definition of web service given in Section 3.1 into a generic context in order to show it equivalent to its specification. Consider the simple networks*

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

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

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

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

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

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 introduce a linear type system, studied for example in [6], to ensure service uniqueness.

5 Core X$d\pi$

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 be able to analyze processes directly, we introduce the Core X$d\pi$-calculus, in which the location structure is pushed locally to the data and processes. We translate X$d\pi$ in Core X$d\pi$, and equate X$d\pi$-equivalence with Core X$d\pi$-equivalence.

Core X$d\pi$ trees, paths and values are defined as for X$d\pi$, with the exception that scripted processes are now Core X$d\pi$ processes.

5.1 Located Processes

Core X$d\pi$ processes are based on asynchronous π_2-processes [10], extended with the update and run operations for interacting with trees. The set of pro-

(Trees)	$V \equiv V' \Rightarrow \mathsf{a}[\,V\,] \equiv \mathsf{a}[\,V'\,]$
(Values)	$v' \equiv w' \wedge \tilde{v} \equiv \tilde{w} \Rightarrow v', \tilde{v} \equiv w', \tilde{w} \qquad P \equiv Q \Rightarrow \Box P \equiv \Box Q$
(Processes)	$(\nu c)(\nu c')P \equiv (\nu c')(\nu c)P \qquad (\nu c)0 \equiv 0$
	$c \notin fn(P) \Rightarrow P \,|\, (\nu c)Q \equiv (\nu c)(P \,|\, Q)$
	$V \equiv V' \wedge P \equiv Q \Rightarrow l{\cdot}\mathsf{update}_p(\chi, V).P \equiv l{\cdot}\mathsf{update}_p(\chi, V').Q$
(Stores)	$\forall\, l.\, D(l) \equiv B(l) \Rightarrow D \equiv B$
(Networks)	$D \equiv B \wedge P \equiv Q \Rightarrow (D, P) \equiv (B, Q)$

Table 3
Structural congruence for Core X$d\pi$ is the least congruence satisfying alpha-conversion, the commutative monoidal laws for $(0, |)$ on trees and processes, and the axioms reported above.

cesses, denoted by $\mathcal{P}$, is given by

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

The constructs in the first line of the grammar correspond to those found in the π_2-calculus: *nil*, *composition* and *restriction* are standard, the *output* process $\overline{l{\cdot}b}\langle\tilde{v}\rangle$ denotes a vector of values $\tilde{v}$ waiting to be sent via channel b at location l, the *input* process $l{\cdot}b(\tilde{z}).P$ waits to receive values from an output process via channel b at l, and the *replicated input* is standard. The run and update commands are located versions of the same commands for X$d\pi$. Structural congruence is analogous to the one for X$d\pi$, and is reported in Table 3. We use the notation $l{\cdot}P$ if P is 0 or a parallel composition of processes (output, input, replicated input, run, update) explicitly located at l. If c does not appear free in P, we will also use the abbreviation

$$\text{(Tau Action)} \quad l{\cdot}\tau.P \stackrel{\text{def}}{=} (\nu c)(\overline{l{\cdot}c} \,|\, l{\cdot}c.P)$$

5.2 Networks and Stores

A network is represented by a pair (D, P) where the first component (the *store*) is a finite partial function from location names to trees, and the second component is a process. Interaction between processes and data is always local, as shown by rules (Update) and (Run) in Table 4. We write $dom(D)$ to denote the domain of store D. We write $D_1 \uplus D_2$ for the union of stores D_1 and D_2

(Com)	$(\{l \mapsto T\}, \overline{l{\cdot}c}\langle\tilde{v}\rangle \mid l{\cdot}c(\tilde{x}).P) \to (\{l \mapsto T\}, P\{\tilde{v}/\tilde{x}\})$
(!Com)	$(\{l \mapsto T\}, \overline{l{\cdot}c}\langle\tilde{v}\rangle \mid !l{\cdot}c(\tilde{x}).P) \to (\{l \mapsto T\}, !l{\cdot}c(\tilde{x}).P \mid P\{\tilde{v}/\tilde{x}\})$
(Update)	$\dfrac{p(T) \rightsquigarrow_{p,l,\chi,V} T', \{\sigma_1, \cdots, \sigma_n\}}{(\{l \mapsto T\}, l{\cdot}\mathsf{update}_p(\chi, V).P) \to (\{l \mapsto T'\}, P\sigma_1 \mid \cdots \mid P\sigma_n)}$
(Run)	$\dfrac{p(T) \rightsquigarrow_{p,l,\Box X,\Box X} T, \{\{\Box P_1/\Box X\}, \cdots, \{\Box P_n/\Box X\}\}}{(\{l \mapsto T\}, l{\cdot}\mathsf{run}_p) \to (\{l \mapsto T\}, P_1 \mid \cdots \mid P_n)}$

Table 4
The reduction relation for Core X$d\pi$ is the smallest relation closed with respect to reduction contexts, structural congruence and the axioms reported above.

with disjoint domains. The network (D, P) is well-formed if D and P contain no free variables, and all the scripted processes have no free session names. Our reduction semantics on networks will be closed with respect to *network contexts* $(C_{\mathcal{S}}, C_{\mathcal{P}})$:

(Store Contexts)	$C_{\mathcal{S}} ::= - \mid C_{\mathcal{S}} \uplus D$
(Process Contexts)	$C_{\mathcal{P}} ::= - \mid C_{\mathcal{P}} \mid P \mid (\nu c)\, C_{\mathcal{P}}$

Given a network (D, P) and a context $C = (C_{\mathcal{S}}, C_{\mathcal{P}})$, we write $C[(D, P)]$ for their composition: for example, if $C_{\mathcal{S}} = - \uplus B$, $C_{\mathcal{P}} = (\nu c) -$ then $C[(D, P)] = (D \uplus B, (\nu c)P)$. A composition involving stores is defined only for stores with disjoint domains. We will omit the subscripts from contexts when no ambiguity can arise.

5.3 Reduction Semantics

The reduction relation $\to$ (Table 4) depends on the same updating function $\rightsquigarrow$ given in Table 2, and describes process interaction, the interaction between processes and data, and (implicitly) the movement of processes across locations. Rules (Com) and (!Com) are basically the standard communication rules for the π-calculus, except that processes only communicate if they are at the same location l, and l is present in the store. Rule (Update) provides interaction between processes and data, and is analogous to that for X$d\pi$.

5.4 Barbed congruence for Core X$d\pi$

The definition of barbed congruence for X$d\pi$ can be simply adapted to Core X$d\pi$.

Definition 5.1 *We define the observation relation $N \downarrow_{l \cdot p}$ on networks and*

barbs by

$$N \downarrow_{l \cdot p} \quad \textit{iff} \quad \exists C, T, \chi, U, P.\ N \equiv C[(\{l \mapsto T\}, l{\cdot}\mathsf{update}_p(\chi, U).P)]$$

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

$$N \Downarrow_{l \cdot p} \quad \textit{iff} \quad \exists N'.\ N \to N' \wedge N' \downarrow_{l \cdot p}$$

Definition 5.2 *Barbed congruence ($\simeq$) is the largest symmetric relation $\mathcal{R}$ on Core Xdπ-networks such that $N \,\mathcal{R}\, M$ implies*

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

We will see in the next section how the positive and negative examples of barbed congruence for Xdπ given in Section 4 can be translated to examples in Core Xdπ.

6 Separation of Data and Processes

The encoding of Xdπ into Core Xdπ is described in Table 5. We explain it using the hyperlink example:

$$\begin{aligned} N &= l\,[\,\mathtt{Link}[\,\mathtt{To}[\,@m{:}q\,]\,|\,\mathtt{Code}[\,\Box P\,]\,]\,] \parallel Q\,] \,|\, m\,[\,T \parallel R\,] \\ Q &= !load(m,q,p).(\nu c)(\mathsf{go}\, m.\, \overline{get}\langle q,l,c\rangle \mid c(X).\mathsf{paste}_p\langle X\rangle) \\ R &= !get(y,x,w).\mathsf{copy}_y(X).\mathsf{go}\, x.\overline{w}\langle X\rangle \end{aligned}$$

The translation to Core Xdπ involves pushing the location structure –in this case the l and m– inside the data and processes. We use $(\!|N|\!)$ to denote the translation of a network, $\llcorner T \lrcorner$ to denote the translation of a tree T, and $(\!|P|\!)_l$ for the translation of a process P which depends on a location l. Our hyperlink example becomes:

$$\begin{aligned} (\!|N|\!) &= (\{l \mapsto \mathtt{Link}[\,\mathtt{To}[\,@m{:}q\,]\,|\,\mathtt{Code}[\,\Box (\!|P|\!)_{\circlearrowleft}\,]\,], m \mapsto \llcorner T \lrcorner\}, (\!|Q|\!)_l \,|\, (\!|R|\!)_m) \\ (\!|Q|\!)_l &= !l{\cdot}load(m,q,p).(\nu c)(l{\cdot}\tau.m{\cdot}\tau.\, \overline{m{\cdot}get}\langle q,l,c\rangle \mid l{\cdot}c(X).l{\cdot}\mathsf{paste}_p\langle X\rangle) \\ (\!|R|\!)_m &= !m{\cdot}get(y,x,w).m{\cdot}\mathsf{copy}_y(X).m{\cdot}\tau.x{\cdot}\tau.\overline{x{\cdot}w}\langle X\rangle \end{aligned}$$

There are several points to notice. The network translation $(\!|-|\!)$ assigns locations to translated trees, which remain the same except that the scripted processes are translated using the self location $\circlearrowleft$. The use of $\circlearrowleft$ is necessary since the location where the scripted process will run is not pre-determined. In our hyperlink example, it runs at l. With an HTML form for example, it is

Network translation:

$(\!|0|\!) = (\emptyset, 0)$

$(\!|N \mid M|\!) = (D \uplus B, P \mid Q)$ where $(\!|N|\!) = (D, P)$ and $(\!|M|\!) = (B, Q)$

$(\!|(\nu c)N|\!) = (D, (\nu c)P)$ where $(\!|N|\!) = (D, P)$

$(\!|l\,[\,T \parallel P\,]|\!) = (\{l \mapsto \llcorner T \lrcorner\}, \langle\!|P|\!\rangle_l)$

Value translation:

$\llcorner 0 \lrcorner = 0$

$\llcorner T \mid T' \lrcorner = \llcorner T \lrcorner \mid \llcorner T' \lrcorner$

$\llcorner \mathsf{a}[\,T\,] \lrcorner = \mathsf{a}[\,\llcorner T \lrcorner\,]$

$\llcorner @l{:}p \lrcorner = @l{:}p$

$\llcorner c \lrcorner = c$, $\llcorner l \lrcorner = l$, $\llcorner p \lrcorner = p$

$\llcorner x \lrcorner = x$, $\llcorner \chi \lrcorner = \chi$

$\llcorner v', \tilde{v} \lrcorner = \llcorner v' \lrcorner, \llcorner \tilde{v} \lrcorner$

$\llcorner \Box P \lrcorner = \Box \langle\!|P|\!\rangle_{\circlearrowleft}$

Process translation:

$\langle\!|0|\!\rangle_l = 0$

$\langle\!|P \mid Q|\!\rangle_l = \langle\!|P|\!\rangle_l \mid \langle\!|Q|\!\rangle_l$

$\langle\!|(\nu c)P|\!\rangle_l = (\nu c)\langle\!|P|\!\rangle_l$

$\langle\!|\mathsf{go}\ m.P|\!\rangle_l = l{\cdot}\tau.m{\cdot}\tau.\langle\!|P|\!\rangle_m$

$\langle\!|\overline{a}\langle\tilde{v}\rangle|\!\rangle_l = \overline{l{\cdot}a}\langle\llcorner \tilde{v} \lrcorner\rangle$

$\langle\!|a(\tilde{x}).P|\!\rangle_l = l{\cdot}a(\tilde{x}).\langle\!|P|\!\rangle_l$

$\langle\!|!a(\tilde{x}).P|\!\rangle_l = !l{\cdot}a(\tilde{x}).\langle\!|P|\!\rangle_l$

$\langle\!|\mathsf{update}_p(\chi, U).P|\!\rangle_l = l{\cdot}\mathsf{update}_p(\chi, \llcorner U \lrcorner).\langle\!|P|\!\rangle_l$

$\langle\!|\mathsf{run}_p|\!\rangle_l = l{\cdot}\mathsf{run}_p$

Table 5
Encodings from X$d\pi$ to Core X$d\pi$.

not known where a form with an embedded scripted process will be required. The process translation $\langle\!|-|\!\rangle_-$ embeds locations in processes. In our example, it embeds location l in Q and location m in R. The only non-trivial case is the migration command. For example, the process $\mathsf{go}\,x.\overline{w}\langle X\rangle$ translates to $m{\cdot}\tau.x{\cdot}\tau.\overline{x{\cdot}w}\langle X\rangle$. The two located tau actions test the existence of the source location m and of the destination location x, and the final located output is obtained by translating the continuation $\overline{w}\langle X\rangle$ at x.

Network equivalence is preserved by the translation from X$d\pi$ to Core X$d\pi$.

Theorem 6.1 (Full Abstraction) *$N \cong M$ if and only if $(\!|N|\!) \simeq (\!|M|\!)$.*

Proof. The proof is long and is reported in the Appendix. The idea is that we split the translation in two steps: a translation from X$d\pi$ to Core X$d\pi^+$, a hybrid calculus still retaining explicit migration, with the reductions in tight correspondence with those for X$d\pi$, and then a translation from Core X$d\pi^+$ to Core X$d\pi$ (the latter is actually a proper subset of the former), where the migration is translated in terms of tau actions. □

7 Process Equivalence

We now analyze process equivalence for Core X$d\pi$. This equivalence depends on the locations present in the network (*network connectivity*). Consider replacing the definition of a service at location l, which uses only local data, with an equivalent one depending on data from another location m. If m is always connected, then the behaviour of the services is the same. On the other hand, if location m should fail, the behaviour of the services is different. With network equivalences, the reliable locations are those in the domain of the store. With process equivalences, it is necessary to state explicitly the minimum set of reliable locations. For example, consider $\texttt{oldS} \stackrel{\text{def}}{=} l{\cdot}\mathsf{cut}_{/}(X)$ and $m{\cdot}\texttt{newS} \stackrel{\text{def}}{=} (\nu c)(\overline{m{\cdot}c}\langle/\rangle \,|\, m{\cdot}c(x).l{\cdot}\mathsf{cut}_x(X))$. The two processes are equivalent if m is reliable, otherwise they are not: in the context $(\{l \mapsto T\}, -)$ the first process can delete T, but the second one cannot move to m. As a consequence, process equivalence is indexed by a given domain of locations.

Definition 7.1 *Given a set of location names Λ, we define the induced* domain barbed congruence *on closed processes by*

$$\sim_\Lambda = \{(P,Q) | \forall\, D\,.\, \Lambda \subseteq dom(D) \Rightarrow (D,P) \simeq (D,Q)\}$$

For example, consider the process `xch` of Example 4.2. For any Λ and l, we have that $\langle\!|\texttt{xch}(T_1,T_2)|\!\rangle_l \sim_\Lambda \langle\!|\texttt{xch}(T_2,T_1)|\!\rangle_l$.

In order to be able to replace a process sub-term by an equivalent one, we extend process equivalence to *open* terms (terms with free variables).

Definition 7.2 Full process contexts *are defined by*

$$C ::= - \;\mid\; C \,|\, P \;\mid\; (\nu c)\, C \;\mid\; l{\cdot}a(\tilde{x}).C \;\mid\; !l{\cdot}a(\tilde{x}).C \;\mid\; l{\cdot}\mathsf{update}_p(\chi, V).C$$

Definition 7.3 *A substitution σ is a* closing substitution *for P iff $P\sigma$ is closed. Given an equivalence $\sim$ for closed processes, and two open processes P and Q, we say that $P \sim Q$ iff $P\sigma \sim Q\sigma$ for all closing substitutions σ.*

From now on our results are implicitly stated for open processes, and therefore hold trivially also for closed processes, which are a special case.

Theorem 7.1 ([26]) *For all Λ, (i) if $\Lambda \subset \Lambda'$ then $\sim_\Lambda \subset \sim_{\Lambda'}$; (ii) $\sim_\Lambda$ is a congruence over full process contexts.*

As an example for the strict inclusion of (i), consider the processes `oldS` and $m{\cdot}\texttt{newS}$ given above. We have $\texttt{oldS} \sim_{l,m} m{\cdot}\texttt{newS}$ but $\texttt{oldS} \not\sim_l m{\cdot}\texttt{newS}$.

If two equivalent processes are both located in the same location l, then we can remove l from the assumptions on the location domain.

Lemma 7.2 ([26]) *If $l{\cdot}P \sim_{\Lambda\cup\{l\}} l{\cdot}Q$ then $l{\cdot}P \sim_{\Lambda} l{\cdot}Q$.*

The congruence result stated in point (ii) of Theorem 7.1 can be strengthened to an even larger class of contexts, where the hole can occur inside values in processes.

Lemma 7.3 ([26]) *For any $l, \tilde{v}$ and V, if $P \sim_{\emptyset} Q$ then*

$$\overline{l{\cdot}a}\langle\tilde{v}\rangle \sim_{\emptyset} \overline{l{\cdot}a}\langle\tilde{v}\{\Box Q/\Box P\}\rangle$$

$$l{\cdot}\mathsf{update}_{/}(X, V) \sim_{\emptyset} l{\cdot}\mathsf{update}_{/}(X, V\{\Box Q/\Box P\})$$

This reinforced congruence property on processes allows us to derive an important property of network equivalence: equivalent scripted processes can be substituted for each other inside the store.

Proposition 7.4 *For all D, R, l, T, if $P \sim_{\emptyset} Q$ then*

$$(D \uplus \{l \mapsto T\}, R) \simeq (D \uplus \{l \mapsto T\{\Box Q/\Box P\}\}, R)$$

Proof. Let $P_1 = l{\cdot}\mathsf{update}_{/}(X, T)$ and $P_2 = l{\cdot}\mathsf{update}_{/}(X, T\{\Box Q/\Box P\})$. From Lemma 7.3 we have that $P_1 \sim_{\emptyset} P_2$. By definition of $\sim_{\emptyset}$, we have that in particular $(\{l \mapsto T\}, P_1) \simeq (\{l \mapsto T\}, P_2)$. Since $(\{l \mapsto T\}, P_1) \to (\{l \mapsto T\}, 0)$, by reduction closure of $\simeq$ it must be the case that $(\{l \mapsto T\}, P_2) \to (\{l \mapsto T\{\Box Q/\Box P\}\}, 0) \simeq (\{l \mapsto T\}, 0)$. By contextuality of $\simeq$ we conclude with $\forall D, R.\ (D \uplus \{l \mapsto T\}, R) \simeq (D \uplus \{l \mapsto T\{\Box Q/\Box P\}\}, R)$. $\Box$

Remark 7.1 *Core Xdπ is an extension of the asynchronous π-calculus, and accordingly the* asynchrony law *– stating that the presence of a communication buffer cannot be observed – holds also in Core Xdπ: $!l{\cdot}a(x).\overline{l{\cdot}a}\langle x\rangle \sim_{\Lambda} 0$. On the other hand, the law for* equators *does not hold: let*

$$l{\cdot}\mathtt{E}(a, b) \stackrel{\text{def}}{=} !l{\cdot}a(x).\overline{l{\cdot}b}\langle x\rangle \mid !l{\cdot}b(x).\overline{l{\cdot}a}\langle x\rangle.$$

We have that

$$l{\cdot}\mathtt{E}(a, b) \mid \overline{l{\cdot}c}\langle a\rangle \not\sim_{\Lambda} l{\cdot}\mathtt{E}(a, b) \mid \overline{l{\cdot}c}\langle b\rangle,$$

since a context can read b from c at l, and use it at some fresh location m where no equator is defined. In the next section we will show how using distributed equators *it is possible regard different names interchangeably only on some designated locations.*

We conclude this subsection with two bigger examples of the equivalence of web services.

Example 7.1 *Recall the web service example in Example 4.4.*

$$\begin{aligned} Q_1 &= \langle\!\langle l{\cdot}\mathsf{Call}\ m{\cdot}c\langle\tilde{v}\rangle\ \mathsf{ret}\ (\tilde{w}).Q\rangle\!\rangle_l \\ Q_2 &= \langle\!\langle \mathsf{go}\ m.P\{\tilde{v}/\tilde{z}\}.\mathsf{go}\ l.Q\rangle\!\rangle_l \\ P_0 &= \langle\!\langle \mathsf{Def}\ c(\tilde{z})\ \mathsf{as}\ P\ \mathsf{out}\ \langle\tilde{w}\rangle\rangle\!\rangle_m \end{aligned}$$

We have that the specification is equivalent to the refinement:

$$(\nu c)(Q_1 \mid P_0) \sim_{\{l,m\}} (\nu c)(Q_2 \mid P_0)$$

Example 7.2 *We give now an example of how it is possible to replicate a web service transparently to the users. Let internal nondeterminism be represented as* $P \oplus Q \stackrel{\text{def}}{=} (\nu a)(\overline{a} \mid a.P \mid a.Q)$*, where* a *does not occur free in* P, Q*. We define two service calls to two interchangeable services, service* R_1 *on channel* c *and* R_2 *on channel* d*:*

$$\begin{aligned} Q_1 &= \langle\!\langle l{\cdot}\mathsf{Call}\ m{\cdot}c\langle\tilde{v}\rangle\ \mathsf{ret}\ (\tilde{w}).Q\rangle\!\rangle_l && \textit{call } c \textit{ on } m \\ Q_2 &= \langle\!\langle l{\cdot}\mathsf{Call}\ n{\cdot}d\langle\tilde{v}\rangle\ \mathsf{ret}\ (\tilde{w}).Q'\rangle\!\rangle_l && \textit{call } d \textit{ on } n \\ P_m &= \langle\!\langle \mathsf{Def}\ c(\tilde{z})\ \mathsf{as}\ P_1\ \mathsf{out}\ \langle\tilde{w}\rangle\rangle\!\rangle_m && \textit{define } c \textit{ at } m \textit{ as } P_1 \\ P_1 &= \langle\!\langle \mathsf{go}\ n.R_1 \oplus \overline{d}\langle\tilde{z}, l, x\rangle\rangle\!\rangle_m && \textit{provide } R_1 \textit{ or call } d \textit{ on } n \\ P_n &= \langle\!\langle \mathsf{Def}\ d(\tilde{z})\ \mathsf{as}\ P_2\ \mathsf{out}\ \langle\tilde{w}\rangle\rangle\!\rangle_n && \textit{define } d \textit{ at } n \textit{ as } P_2 \\ P_2 &= \langle\!\langle \mathsf{go}\ m.R_2 \oplus \overline{c}\langle\tilde{z}, l, x\rangle\rangle\!\rangle_n && \textit{provide } R_1 \textit{ or call } c \textit{ on } m \end{aligned}$$

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

$$Q_1 \mid P_m \mid P_n \sim_{\{m,n\}} Q_2 \mid P_m \mid P_n$$

We can also show a result analogous to the single web service given in Example 7.1. The nondeterministic specification process

$$Q_s = \langle\!\langle \mathsf{go}\ m.R_1\{\tilde{v}/\tilde{z}\}.\mathsf{go}\ l.Q \oplus \mathsf{go}\ n.R_2\{\tilde{v}/\tilde{z}\}.\mathsf{go}\ l.Q'\rangle\!\rangle_l$$

is equivalent to any of the two service calls Q_1 *or* Q_2*. For example*

$$(\nu c, d)(Q_1 \mid P_m \mid P_n) \sim_{\{m,n\}} (\nu c, d)(Q_s \mid P_m \mid P_n)$$

where the restriction of c *and* d *avoids competing services on the same channel. Now consider the case when* $R_1 = R_2$*, and* R_1 *does not have any barb at* m*. Let* $Q'_s = \langle\!\langle \mathsf{go}\ m.R_1\{\tilde{v}/\tilde{z}\}.\mathsf{go}\ l.Q\rangle\!\rangle_l$ *and let* $Q = Q'$*. We have the equivalence*

$$(\nu c, d)(Q_1 \mid P_m \mid P_n) \ \sim_{\{m,n\}} \ (\nu c, d)(Q'_s \mid P_m \mid P_n)$$

which shows that a client cannot be aware that the service has been replicated.

The type of equivalences defined in the previous section are known to be difficult to use. In particular, the condition of closure under contexts involves a universal quantification on processes which complicates the proofs. In [18,26], we have studied proof methods based on labelled bisimulation relations where congruence is a derived property. In particular, in [26] we have defined a bisimulation equivalence $\approx_\Lambda$ with the property that, given two processes P, Q, if $P \approx_\Lambda Q$ then $P \sim_\Lambda Q$. Although the proof technique is not complete, it is strong enough to prove all the process equivalences shown in this paper. The main difficulties involved in defining such equivalences for Core X$d\pi$ are caused by having scripted processes among values, and by barbed equivalence being sensitive to the presence of locations.

The technique defined in [18] is based on higher-order bisimulation for concurrent processes, which has been studied for example in [36,14]. As noted in [33], there is a problem inherently connected with the higher-order technique that we have chosen there: requiring bisimilarity for $\Box P$ and $\Box Q$ can be too restrictive. In fact, it could be the case that $P \not\approx_\Lambda Q$, but $\Box P$ and $\Box Q$ are only run inside some context C such that $C[P] \approx_\Lambda C[Q]$.

In [26] we define an alternative notion of bisimulation, where we solve the problem just mentioned by translating messages containing scripts into ones where each script is replaced by a uniquely named *trigger* (a placeholder), and placing in parallel some *definitions* associating each trigger with the code of the scripted process. Using this approach, it is possible to analyze the interaction between scripts and their contexts. For a discussion of this technique, see [24,33] (where it is applied to the higher-order π-calculus). We solve the problem of location-sensitivity using an adaptation of the bisimulation approach to families of relations indexed by sets of locations, which we call *domain-dependent bisimilarity*.

8 Conclusions and Future Work

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 links to other parts of the network, and π-processes extended with an explicit migration primitive and an update command for interacting with data. An alternative approach would have been to encode data as π-processes. Instead, 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 inter-

action.

We have studied behavioural equivalences for X$d\pi$. In future work we plan to use them to reason formally about many alternative patterns for exchanging information on the web, for example the ones discussed in [4,32].

Alex Ahern has developed a prototype implementation, adapting the ideas presented here to XML standards [5]. 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 [29]. We aim to continue this implementation work, perhaps incorporating ideas from other recent work on languages based on the π-calculus [12,15].

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 many similarities between our model and features of the Active XML [4,1] implementation, and we are in the process of doing an in-depth comparison between the two projects.

Security is a central concern for systems sharing dynamic data on the Web. We are currently studying fine-grained access control for web services and documents, and type systems for statically guaranteeing the structure of X$d\pi$ data, extending techniques studied for the distributed π-calculus [21] and semi-structured data [13].

In summary, this paper provides a first step towards the adaptation of techniques associated with process calculi and semi-structured data to reason about the dynamic evolution of data on the Web.

8.1 Acknowledgements

We would like to thank Serge Abiteboul, Alex Ahern, Omar Benjelloun, Luca Cardelli, Giorgio Ghelli, Tova Milo, Val Tannen, and Nobuko Yoshida for many stimulating discussions. We thank the anonymous referees for their suggestions, which have helped us to improve the presentation of the paper.

References

[1] Serge Abiteboul, Omar Benjelloun, Tova Milo, Ioana Manolescu, and Roger Weber. Active XML: A data-centric perspective on Web services. Verso Report number 213, 2002.

[2] Serge Abiteboul, Angela Bonifati, Grégory Cobena, Ioana Manolescu, and Tova Milo. Dynamic XML documents with distribution and replication. In *Proceedings of ACM SIGMOD*, 2003.

[3] Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the Web: from relations to semistructured data and XML.* Morgan Kaufmann, 2000.

[4] Serge Abiteboul, et al. Active XML primer. INRIA Futurs, GEMO Report number 275, 2003.

[5] Alexander Ahern. A Language for Updating Web Data. Master's thesis, Imperial College London, 2003.

[6] Martin Berger, Kohei Honda, and Nobuko Yoshida. Linearity and bisimulation. In *Proceedings of FoSSaCS'02*, pages 290–301. LNCS, 2002.

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

[8] Reinhard Braumandl, Markus Keidl, Alfons Kemper, Donald Kossmann, Alexander Kreutz, Stefan Seltzsam, and Konrad Stocker. Objectglobe: Ubiquitous query processing on the internet. To appear in the VLDB Journal: Special Issue on E-Services, 2002.

[9] Cristiano Calcagno, Philippa Gardner, and Uri Zarfaty. Context logic and tree update. POPL'05, to appear, 2004.

[10] Marco Carbone and Sergio Maffeis. On the expressive power of polyadic synchronisation in π-calculus. *Nordic Journal of Computing*, 10(2):70–98, 2003.

[11] Luca Cardelli and Giorgio Ghelli. A query language based on the ambient logic. In *Proceedings of ESOP'01)*, volume 2028 of *LNCS*, pages 1–22. Springer, 2001.

[12] Sylvain Conchon and Fabrice Le Fessant. Jocaml: Mobile agents for Objective-Caml. In *Proceedings of ASA'99/MA'99*, Palm Springs, CA, USA, 1999.

[13] Ernesto Damiani, Sabrina De Capitani di Vimercati, Stefano Paraboschi, and Pierangela Samarati. A fine-grained access control system for xml documents. *ACM Trans. Inf. Syst. Secur.*, 5(2):169–202, 2002.

[14] William Ferreira, Matthew Hennessy, and Alan Jeffrey. A theory of weak bisimulation for core cml. *Journal of Functional Programming*, 8(5):447–491, 1998.

[15] Philippa Gardner, Cosimo Laneve, and Lucian Wischik. Linear forwarders. In *Proceedings of CONCUR 2003*, volume 2761 of *LNCS*, pages 415–430. Springer, 2003.

[16] Philippa Gardner and Sergio Maffeis. Modelling dynamic Web data. Proc. of Appsem'04, Tallinn, 2004.

[17] Philippa Gardner and Sergio Maffeis. Modelling dynamic Web data. In Georg Lausen and Dan Suciu, editors, *Proc. of DBPL'03*, volume 2921 of *LNCS*, September 2003.

[18] Philippa Gardner and Sergio Maffeis. Modelling dynamic Web data. Imperial College London Technical Report, Oct 2003.

[19] Jens Godskesen, Thomas Hildebrandt, and Vladimiro Sassone. A calculus of mobile resources. In *Proceedings of CONCUR'02*, LNCS, 2002.

[20] Andrew Gordon and Riccardo Pucella. Validating a web service security abstraction by typing. In *Proceedings of the 2002 ACM Workshop on XML Security*, pages 18–29, 2002.

[21] Matthew Hennessy and James Riely. Resource access control in systems of mobile agents. *Inf. Comput.*, 173(1):82–120, 2002.

[22] Kohei Honda and Mario Tokoro. An object calculus for asynchronous communication. In *Proceedings of ECOOP*, volume 512 of *LNCS*, pages 133–147, Berlin, Heidelberg, New York, Tokyo, 1991. Springer-Verlag.

[23] Kohei Honda and Nobuko Yoshida. On reduction-based process semantics. *Theoretical Computer Science*, 151(2):437–486, 1995.

[24] Alan Jeffrey and Julian Rathke. Contextual equivalence for higher-order pi-calculus revisited. Computer Science Report 04/2002, University of Sussex, 2002.

[25] Alfons Kemper and Christian Wiesner. Hyperqueries: Dynamic distributed query processing on the internet. In *Proceedings of VLDB'01*, pages 551–560, 2001.

[26] Sergio Maffeis and Philippa Gardner. Behavioural equivalences for dynamic Web data. In *Proc. of Ifip TCS'04*, Kluwier, August 2004. Draft long version available from `http://www.doc.ic.ac.uk/~maffeis/`.

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

[28] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, I and II. *Information and Computation*, 100(1):1–40,41–77, September 1992.

[29] Benjamin C. Pierce and David N. Turner. Pict: A programming language based on the pi-calculus. In *Proof, Language and Interaction: Essays in Honour of Robin Milner*. MIT Press, 2000.

[30] Arnaud Sahuguet. *ubQL: A Distributed Query Language to Program Distributed Query Systems*. PhD thesis, University of Pennsylvania, 2002.

[31] Arnaud Sahuguet, Benjamin Pierce, and Val Tannen. Distributed Query Optimization: Can Mobile Agents Help? Unpublished draft.

[32] Arnaud Sahuguet and Val Tannen. Resource Sharing Through Query Process Migration. University of Pennsylvania Technical Report MS-CIS-01-10, 2001.

[33] Davide Sangiorgi. Expressing mobility in process algebras: First-order and higher-order paradigms. PhD thesis, University of Edinburgh, 1992.

[34] Davide Sangiorgi and Robin Milner. Techniques of weak bisimulation up to. Revised version of an article appeared in the Proc. CONCUR'92, LNCS 630, 1992.

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

[36] Bent Thomsen. A theory of higher order communicating systems. *Inf. Comput.*, 116(1):38–57, 1995.

A Full Abstraction

In order to study the formal relation between X$d\pi$ and Core X$d\pi$ we find it helpful to consider a slightly bigger language (Core X$d\pi^+$) which still retains explicit migration. In this way it is possible to have a tighter correspondence between reductions in X$d\pi$ and in Core X$d\pi^+$ than it would be possible directly for Core X$d\pi$. After relating network equivalence for the first two languages, we can relate the one for the latter two by exploiting the fact that Core X$d\pi$ is a proper subset of Core X$d\pi^+$.

A.1 From Xdπ to Core Xdπ⁺

Core X$d\pi^+$ is obtained from Core X$d\pi$ by extending the grammar of processes with the production

$$P ::= \ldots \;\mid\; l{\cdot}\mathsf{go}\ m.P$$

and extending the reduction rules with the axioms

$$(\textsc{Stay})\ (\{l \mapsto T\}, l{\cdot}\mathsf{go}\ l.P) \rightarrow\ (\{l \mapsto T\}, P)$$

$$(\textsc{Go})\ (\{m \mapsto T, l \mapsto S\}, l{\cdot}\mathsf{go}\ m.P) \rightarrow\ (\{m \mapsto T, l \mapsto S\}, P)$$

The only effect of rule (Go) is to verify the existence of locations l and m, and rule (Stay) can be seen as the special case where $l = m$. Structural congruence

for Core X$d\pi^+$ is analogous to the one for Core X$d\pi$. All the definitions and results of Section 5 and Section 7 hold analogously for Core X$d\pi^+$. The most interesting additional laws holding for Core X$d\pi^+$ state that an initial migration step from a location in the domain of the store cannot be observed, and that independently from the store domain, a migration step is equivalent to two subsequent tau actions located at the source and target of the migration. This justifies informally the rule for encoding process migration from X$d\pi$ to Core X$d\pi$.

Lemma A.1 ([26]) *(i) Migration from an existing location cannot be observed:* $l{\cdot}\mathsf{go}\ m.m{\cdot}P \sim_{\{l\}} m{\cdot}P$. *(ii) A migration step is equivalent to checking the existence of the source and target locations:* $l{\cdot}\mathsf{go}\ m.P \sim_{\emptyset} l{\cdot}\tau.m{\cdot}\tau.P$.

The encoding of X$d\pi$ is the same as that for Core X$d\pi$, except in the case for migration, which becomes $(\!|\mathsf{go}\ m.P|\!)_l = l{\cdot}\mathsf{go}\ m.(\!|P|\!)_m$. With a slight abuse of notation, in this section we reserve $(\!|-|\!), (\!|-|\!)_-, \llcorner - \lrcorner$ for the encodings of networks, processes and trees from X$d\pi$ into Core X$d\pi^+$, and we will denote by $[\![(\!|-|\!)]\!]$ the encoding of networks from X$d\pi$ into Core X$d\pi$, denoted in precedence simply with $(\!|-|\!)$.

In order to establish an operational correspondence between X$d\pi$ and Core X$d\pi^+$, we need some auxiliary lemmas. The first lemma shows that the encodings respect structural congruence.

Lemma A.2 $N \equiv M$ *if and only if* $(\!|N|\!) \equiv (\!|M|\!)$.

Proof. Both cases follow by structural induction on N. □

In order to show that the encodings preserve substitutions, we need to extend the notion of data translation to substitutions and sets of substitutions, in the obvious way: $\llcorner\{\sigma_1, \cdots, \sigma_n\}\lrcorner = \{\llcorner\sigma_1\lrcorner, \cdots, \llcorner\sigma_n\lrcorner\}$, $\quad \llcorner\{\tilde{v}/\tilde{x}\}\lrcorner = \{\llcorner\tilde{v}\lrcorner/\tilde{x}\}$.

Lemma A.3 $(\!|P\{\tilde{v}/\tilde{x}\}|\!)_l = (\!|P|\!)_l\{\llcorner\tilde{v}\lrcorner/\tilde{x}\}$.

Proof. Follows by structural induction on P. □

The updating and querying of trees is preserved by the encodings.

Lemma A.4 *(i) If* $p(T) \rightsquigarrow_{p,l,\chi,V} T', \Sigma$ *then* $p(\llcorner T\lrcorner) \rightsquigarrow_{p,l,\chi,\llcorner V\lrcorner} \llcorner T'\lrcorner, \llcorner\Sigma\lrcorner$ *(ii) If* $p(\llcorner T\lrcorner) \rightsquigarrow_{p,l,\chi,\llcorner V\lrcorner} \llcorner T'\lrcorner, \llcorner\Sigma\lrcorner$ *then* $p(T) \rightsquigarrow_{p,l,\chi,V} T', \Sigma$

Proof. Both cases follow by well-founded induction on the nesting depth of underlinings in $p(T)$, where the nesting depth $d(T)$ is defined inductively by

$$d(0) = d(@l{:}p) = d(\Box P) = 0 \qquad d(T \mid S) = max(d(T), d(S))$$
$$d(\mathsf{a}[\,T\,]) = d(T) \qquad d(\mathsf{a}[\underline{\,U\,}]) = 1 + d(U)$$

□

For clarity, we state explicitly the reduction rules dealing with structural congruence and reduction contexts for X$d\pi$ (the ones for Core X$d\pi^+$ are analogous).

$$(\textsc{Ctx})\ \frac{N \searrow N'}{C[N] \searrow C[N']} \qquad (\textsc{Struct})\ \frac{N \equiv M \searrow M' \equiv N'}{N \searrow N'}$$

Lemma A.5 (Strong Operational Correspondence) *For any X$d\pi$ network N, (i) if $N \searrow M$ then $(\!|N|\!) \to (\!|M|\!)$; (ii) if $(\!|N|\!) \to M'$ then there exists M such that $(\!|M|\!) \equiv M'$ and $N \searrow M$.*

Proof. (i) By rule induction on the derivation of $N \searrow M$. The base cases are ((Stay),(Go),(Com),(Com!),(Update),(Run)), and the inductive cases are ((Struct),(Ctx)).

- (Stay): If $N \searrow M$ by rule (Stay), then $N = m\,[\,T \parallel Q \mid \mathsf{go}\, m.P\,]$ and $M = m\,[\,T \parallel Q \mid P\,]$, and by definition of encoding

  $$(\!|N|\!) = (\{m \mapsto \llcorner T \lrcorner\}, \langle\!|Q|\!\rangle_m \mid m{\cdot}\mathsf{go}\ m.\langle\!|P|\!\rangle_m) = N'$$

  We derive $N' \to (\{m \mapsto \llcorner T \lrcorner\}, \langle\!|Q|\!\rangle_m \mid \langle\!|P|\!\rangle_m) = M'$ by using rule (Ctx), with the instance of axiom (Stay) $(\{m \mapsto \llcorner T \lrcorner\}, m{\cdot}\mathsf{go}\ m.\langle\!|P|\!\rangle_m) \to (\{m \mapsto \llcorner T \lrcorner\}, \langle\!|P|\!\rangle_m)$ as a premise. We conclude since by definition of encoding, $(\!|M|\!) = M'$.
- (Go): Similar to the previous case.
- (Com): Like the previous case, using point (i) of Lemma A.3 to justify that the results of the communication in the translation equals the encoding of M: $(\{l \mapsto \llcorner T \lrcorner\}, \langle\!|P|\!\rangle_l\{\llcorner \tilde{v} \lrcorner/\tilde{x}\} \mid \langle\!|Q|\!\rangle_l) = (\{l \mapsto \llcorner T \lrcorner\}, \langle\!|P\{\tilde{v}/\tilde{x}\}|\!\rangle_l \mid \langle\!|Q|\!\rangle_l)$.
- (Com!): Similar to the previous case.
- (Update): Similar to the previous cases, using point (i) of Lemma A.4 to relate the premises $p(T) \rightsquigarrow\!\!\!\!\shortmid_{p,l,\chi,V} T', \Sigma$ and $p(\llcorner T \lrcorner) \rightsquigarrow\!\!\!\!\shortmid_{p,l,\chi,\llcorner V \lrcorner} \llcorner T' \lrcorner, \llcorner \Sigma \lrcorner$ of the axiom (Update) in the two calculi.
- (Run): Similar to the previous case.
- (Ctx): Suppose that $N \searrow M$ and $(\!|N|\!) \to (\!|M|\!)$. We want to show that $C[N] \searrow C[M]$ implies $(\!|C[N]|\!) \to (\!|C[M]|\!)$. By a simple induction on the structure of $C[-]$, it follows that there exists a Core X$d\pi^+$ context $K[-]$ such that $(\!|C[N]|\!) = K[(\!|N|\!)]$ and $(\!|C[M]|\!) = K[(\!|M|\!)]$. This allows us to conclude, by rule (Ctx), that $(\!|C[N]|\!) \to (\!|C[M]|\!)$.
- (Struct): Follows using point (i) of Lemma A.2 and rule (Struct) on the Core X$d\pi$ term.

(ii) By cases on the axioms used to derive $(\!|N|\!) \to M'$. We show in detail the case for (Stay). If the axiom used to derive $(\!|N|\!) \to M'$ is (Stay) then it must be the case that $(\!|N|\!) \equiv (\nu\tilde{c})(\{m \mapsto \llcorner T \lrcorner\} \uplus B, m{\cdot}\mathsf{go}\ m.\langle\!|P|\!\rangle_m \mid \langle\!|R|\!\rangle_m \mid Q^{dom(B)}) = N_1$, and $M' \equiv (\nu\tilde{c})(\{m \mapsto \llcorner T \lrcorner\} \uplus B, \langle\!|P|\!\rangle_m \mid \langle\!|R|\!\rangle_m \mid Q^{dom(B)}) = M_1$, where for some N', $(\!|N'|\!) = (B, Q)$. For $N_2 = (\nu\tilde{c})(m\,[\,T \parallel \mathsf{go}\ m.P \mid R\,] \mid N')$ we have that $(\!|N_2|\!) = N_1$, and by definition of $\searrow$, $N_2 \searrow (\nu\tilde{c})(m\,[\,T \parallel P \mid R\,] \mid N_1) = M_2$, with $(\!|M_2|\!) = M_1$. By point (ii) of Lemma A.2 we have that $(\!|N|\!) \equiv N_1$ implies

$N \equiv N_2$. We conclude using $N_2 \searrow M_2$ and (Struct) to derive $N \searrow M$ where $M = M_2$, and therefore $(\!|M|\!) \equiv M'$. The cases for the other axioms are similar, and use point (ii) of Lemma A.3 and point (ii) of Lemma A.4. □

Lemma A.6 (Weak Operational Correspondence) *Given any Xdπ network N, (i) if $N \searrow^* M$ then $(\!|N|\!) \rightarrow^* (\!|M|\!)$; (ii) if $(\!|N|\!) \rightarrow^* M'$ then there exists M such that $(\!|M|\!) \equiv M'$ and $N \searrow^* M$.*

Proof. Both cases follow by induction on the number of reduction steps and Lemma A.5. □

Lemma A.7 (Observational Correspondence) $N \Downarrow_{l\cdot\beta} \Leftrightarrow (\!|N|\!) \Downarrow_{l\cdot\beta}$.

Proof. ($\Rightarrow$) By definition of $\Downarrow$ and Lemma A.6 it is enough to show that if $N \downarrow_{l\cdot\beta}$ then $(\!|N|\!) \downarrow_{l\cdot\beta}$, which follows easily from the definition of the encodings. ($\Leftarrow$) Analogous to the previous case. □

Note that there are Core X$d\pi^+$ processes which cannot be obtained from the translation of X$d\pi$ processes, such as for example $l\cdot a(x).\overline{m\cdot b}\langle x\rangle$. Consequently Core X$d\pi^+$ has in some sense *more* reduction contexts than X$d\pi$, and in order to relate contextual equivalence in the two languages we need to relate the contexts first.

Lemma A.8 *For any Core Xdπ$^+$ process P and any l, there exists an Xdπ process Q such that $\langle\!|Q|\!\rangle_l \sim_{\{l\}} P$.*

Proof. By structural induction on P. We show the inductive case for the input process. Given a Core X$d\pi^+$ process $m\cdot a(x).P'$, we want to show that for the X$d\pi$ process $Q = go\ m.a(x).R$, where R is such that, by hypothesis, $\langle\!|R|\!\rangle_m \sim_{\{m\}} P'$, we have $m\cdot a(x).P' \sim_{\{l\}} \langle\!|Q|\!\rangle_l = l\cdot\mathsf{go}\ m.m\cdot a(x).\langle\!|R|\!\rangle_m$. By the congruence property of process equivalence we have that $m\cdot a(x).P' \sim_{\{m\}} m\cdot a(x).\langle\!|R|\!\rangle_m$, by the analogous of Lemma 7.2 for Core X$d\pi^+$ we have that this holds also for $\sim_\emptyset$, and by point (ii) of Theorem 7.1 we get $m\cdot a(x).P' \sim_{\{l\}} m\cdot a(x).\langle\!|R|\!\rangle_m$. By point (i) of Lemma A.1 we obtain

$$m\cdot a(x).\langle\!|R|\!\rangle_m \sim_{\{l\}} l\cdot\mathsf{go}\ m.m\cdot a(x).\langle\!|R|\!\rangle_m$$

and we conclude by transitivity of $\sim_{\{l\}}$. The other cases are similar. □

Lemma A.9 (Contextual Correspondence) *Given any Xdπ network N, and any Core Xdπ$^+$ context $C[-]$, if $N \not\equiv 0$ then there exists an Xdπ context*

$K[-]$ such that $(\!|K[N]|\!) \simeq C[(\!|N|\!)]$.

Proof. (Sketch). By induction on the structure of $C[-]$. The inductive case where $C[-] = (C_1[-], C_2[-] \mid P)$ uses the hypothesis of non-emptiness of N to build a context $K[-]$ where an X$d\pi$ process R such that $\langle\!|R|\!\rangle_l$ equivalent to P (which exists by Lemma A.8) is placed inside a location l in the domain of N. □

We now have all the building blocks to show that the encoding preserves barbed congruence.

Theorem A.10 (Full Abstraction) *$N \cong M$ if and only if $(\!|N|\!) \cong (\!|M|\!)$.*

Proof. ($\Rightarrow$) By showing that the relation $\mathcal{R} = \{(N', M') \mid N' \simeq (\!|N|\!),\ M' \simeq (\!|M|\!),\ N \cong M\}$ on Core X$d\pi^+$ networks is included in $\simeq$[3]. Relation $\mathcal{R}$ preserves the observables by Lemma A.7, and is reduction closed by Lemma A.6. We now show that $\mathcal{R}$ preserves contexts. Suppose that $(N', M') \in \mathcal{R}$, and therefore $N' \simeq (\!|N|\!)$, $M' \simeq (\!|M|\!)$, and $N \cong M$. We need to show that for an arbitrary $C[-]$, also $(C[N'], C[M']) \in \mathcal{R}$. By $N' \simeq (\!|N|\!)$ and contextuality of $\simeq$, we get $C[N'] \simeq C[(\!|N|\!)]$. By Lemma A.9 and transitivity of $\simeq$, we get $C[N'] \simeq (\!|K[N]|\!)$ for a suitable $K[-]$. Similarly, we derive $C[M'] \simeq (\!|K[M]|\!)$, and we conclude deriving $K[N] \cong K[M]$, by contextuality of $\cong$, from $N \cong M$. ($\Leftarrow$) By showing that the relation $\mathcal{S} = \{(N, M) \mid (\!|N|\!) \simeq (\!|M|\!)\}$ on X$d\pi$ networks is included in $\cong$. The first two cases are similar to the previous point. Relation $\mathcal{S}$ preserves contexts by definition of encoding. □

A.2 From Core X$d\pi^+$ to Core X$d\pi$

We have formally related X$d\pi$ and Core X$d\pi^+$. In this subsection we relate Core X$d\pi^+$ and Core X$d\pi$, in order to justify using Core X$d\pi$ as a core calculus for the study of behavioural equivalences.

Consider the encoding $[\![-]\!]$ from Core X$d\pi^+$ to Core X$d\pi$ networks which is homomorphic on all terms except for the case of process migration, where it is specified as $[\![l{\cdot}\mathsf{go}\ m.P]\!] = l{\cdot}\tau.m{\cdot}\tau.[\![P]\!]$, as informally suggested by point (ii) of Lemma A.1. The encoding of processes respects process equivalence.

Lemma A.11 *For all Core X$d\pi^+$ processes P, $P \sim_\emptyset [\![P]\!]$.*

Proof. By structural induction on P. All the homomorphic cases are trivial.

[3] Weak bisimulation up-to weak bisimulation is a valid proof technique because of the use of the weak observation relation on the first clause of barbed congruence (see [34]).

We show the inductive case for the migration process. Given $P = l{\cdot}\mathsf{go}\ m.R$ we have that $[\![P]\!] = l{\cdot}\tau.m{\cdot}\tau.[\![R]\!]$, where by hypothesis $[\![R]\!] \sim_\emptyset R$, and we want to show that $l{\cdot}\mathsf{go}\ m.R \sim_\emptyset l{\cdot}\tau.m{\cdot}\tau.[\![R]\!]$. By applying the congruence property of process equivalence on $[\![R]\!] \sim_\emptyset R$ we get $l{\cdot}\tau.m{\cdot}\tau.[\![R]\!] \sim_\emptyset l{\cdot}\tau.m{\cdot}\tau.R = P'$. By point (ii) of Lemma A.1 we get $P \sim_\emptyset P'$ and by transitivity of $\sim_\emptyset$ we conclude with $P \sim_\emptyset [\![P]\!]$. □

The substitutivity of process equivalence and the previous lemma tell us that the encoding of trees preserves equivalence.

Corollary A.12 *From the analogous of Proposition 7.4 on Core Xdπ⁺, and Lemma A.11, it follows that for all D, R, l, T, $(D \uplus \{l \mapsto T\}, R) \simeq (D \uplus \{l \mapsto [\![T]\!]\}, R)$.*

Lemma A.13 *For any Core Xdπ⁺ network N, $N \simeq [\![N]\!]$.*

Proof. By structural induction on N. The base case for $N = (\emptyset, 0)$ is trivial. There are two inductive cases, one for stores and one for processes. For the inductive case for stores, suppose $N_1 = (D, P) \simeq [\![(D, P)]\!] = ([\![D]\!], [\![P]\!]) = M_1$. We want to show that $N = (\{l \mapsto T\} \uplus D, P) \simeq [\![(\{l \mapsto T\} \uplus D, P)]\!] = (\{l \mapsto [\![T]\!]\} \uplus [\![D]\!], [\![P]\!]) = M$. By contextuality of $\simeq$ on $N_1 \simeq M_1$ we have that $(\{l \mapsto [\![T]\!]\} \uplus D, P) \simeq M$, and by Corollary A.12 on $(\{l \mapsto [\![T]\!]\} \uplus D, P)$ and N, and by transitivity of $\simeq$ we conclude with $N \simeq M$. The case for processes is similar, and goes by induction on the structure of the process context $C[-]$, using Lemma A.11 instead of Corollary A.12. □

For clarity, below we denote by $\simeq^+$ the network barbed equivalence for Core Xdπ⁺ networks, and by $\simeq$ the one for Core Xdπ.

Observation A.14 *By definition of encoding, we have that for any Core Xdπ⁺ network N, and any Core Xdπ⁺ context $C[-]$, there exists a Core Xdπ context $K[-]$ such that $[\![C[N]]\!] = K[[\![N]\!]]$.*

Theorem A.15 *For any Core Xdπ⁺ networks N, M, $N \simeq^+ N$ if and only if $[\![N]\!] \simeq [\![M]\!]$.*

Proof. ($\Rightarrow$) Follows easily from Lemma A.13 and the fact that each Core Xdπ context is also a Core Xdπ⁺ context. ($\Leftarrow$) Follows again from Lemma A.13, using Observation A.14 for relating the contexts. □

Corollary A.16 (Theorem 6.1) *From Theorem A.10 and Theorem A.15 follows that, for any Xdπ networks N, M, $N \simeq M$ if and only if $[\![[\![N]\!]]\!] \simeq [\![[\![M]\!]]\!]$.*