Behavioural Equivalences for Dynamic Web Data

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

Abstract

Peer-to-peer systems, exchanging dynamic documents through Web services, are a simple and effective platform for data integration on the internet. Dynamic documents can contain both data and references to external sources in the form of links, calls to web services, or coordination scripts. XML standards, and industrial platforms for web services, provide the technological basis for building such systems, and process algebras are a promising tool for studying and understanding their formal properties.

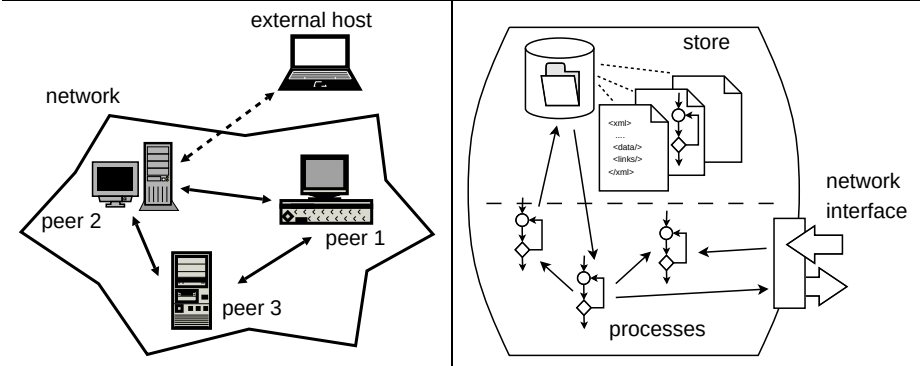
Core X$d\pi$ is the explicitly located version of X$d\pi$, a process calculus designed for reasoning about dynamic Web data, based on explicit repositories of higher-order semistructured data and π-calculus-like processes which can communicate with each other, query and update the local repository, or migrate to other peers to continue execution.

We study behavioural equivalences for Core X$d\pi$ processes. To help with the proofs, which require a costly property of closure under contexts, we define a coinductive relation (called *domain bisimilarity*) which does not quantify over contexts and which entails process equivalence. Its definition is non-standard, because scripts are part of the values, and process equivalences are sensitive to the set of locations constituting the network. We apply our process equivalence to study some communication patterns used by servers in distributed query systems, and we propose a new pattern involving mobile code.

1 Introduction

The World Wide Web is a global network, used in daily activities to find information, communicate ideas, conduct business and carry out distributed computations. In order to fully exploit the potential of this massive network, there is a need for scalable mechanisms to organize and manipulate the available information. Peer-to-peer architectures help to deal with the issue of scalability, and technologies such as XML and Web services facilitate the development of distributed applications. XML [38] is a standardized data model, used to represent uniformly documents containing tagged information not adhering to a fixed structure. Web services [40] are Web sites which are designed to be used by applications rather than humans. Web service inter-operability is facilitated

Figure 1: Reference architecture



(A) Network. (B) Peer.

by the use of XML for data representation and of related standards for service invocation, description and discovery (SOAP, WSDL, UDDI [41, 39, 36]).

Data integration on the Web constitutes a challenging application for these technologies, because of the extreme heterogeneity of data sources involved, and the complexity of communication patterns which can arise. For example, translating a declarative request for networked data into a low-level execution plan may involve recursively invoking other declarative requests on different Web sites. Inspired by this problem, the X$d\pi$ calculus [11] studies peer-to-peer architectures for exchanging Web data, schematically represented in Figure 1. Each *network* is composed by a variable number of interconnected peers, all sharing a similar internal structure, and each one identified by a unique name (Figure 1(A)). Peers share a common messaging protocol where the name of a peer is assumed to coincide with its network address: at this level of abstraction there are no restrictions to connectivity due to network domains or firewalls. Networks are *open* in the sense that it is always possible to add new peers or learn dynamically about their existence, and external hosts may participate in the data exchange too, typically playing a limited role. Each peer (schematized in Figure 1(B)) acts both as a provider and consumer of information. It contains a data repository, an internal working space where processes carry out local computations, and a network interface providing remote communication and services to other peers. Processes can communicate locally with each other, query and update the local repository and, when the architecture supports mobility, can migrate to other peers to continue execution. Repositories present to the processes a semi-structured view of their data. Data contains enough meta-information about its own structure to make it possible writing expressive queries.

Typically, data is not completely static. It may contain references to other data and services, in the form of URLs and queries, or *scripts*. A script is some code describing a process which can be interpreted by the working space to add dynamic content to documents. We refer to such data as *dynamic Web*

data. The World Wide Web itself is a very general example of architecture for dynamic Web data. Servers use the HTTP protocol to interact with each other, either requesting or providing information. HTML pages can contain hyperlinks, forms and client-side scripts, which provide dynamic behaviour. Web clients running a browser can be considered as the "external hosts" which participate to a smaller degree in the exchange of information, by mostly consuming rather than providing data.

A more specific example comes from the database world. The Active XML [33, 3] system for data integration (AXML for short) is based on networks of peers each containing a repository of documents and a set of service definitions. AXML service definitions typically consist of queries and updates on the local repository, but in general can consist of arbitrary Web services, providing an interface to hosts external to the AXML system. AXML documents are XML documents which can include special tags representing calls to services on other peers. The parameters to these service calls can be local queries (path expressions) or AXML data, hence service calls can be nested. Documents containing service calls are called *intensional documents*, and *materialization* is the process of invoking a service call and pasting its results in the original document. One interesting source of flexibility in AXML is the choice of when to materialize service calls. It can be done periodically, or when the data containing the call is fetched from the repository, or when it is returned to the client. Similarly, if a service call appears as a parameter to another service call, it can be materialized before calling the service or it can be passed on to it as an intensional parameter.

Besides Web browsers and AXML, a large class of other Web applications (such as file-sharing programs, personal Web portals, online bibliographic databases, etc.) can be seen as instances of the reference architecture given above, each with its own particular features and restrictions. The problems that these architectures have to address, in order to be practically useful, are varied. Firstly, it is well-known that interaction between concurrent processes is difficult to regulate. In the case of Web services, this problem is complicated by the difficulty in maintaining state across different Web service invocations, and requires the study of orchestration techniques.[1] Secondly, a major concern for systems dealing with dynamic Web data is security. Depending on the application domain, it may be crucial to have control for example over data integrity, confidentiality, or access control. The formal study of security properties needs to be grounded on a rigourous model of these architectures, and process algebraic techniques are particularly suited for the task, as they have already been successfully used to study concurrent, distributed and mobile systems, and analyze their formal properties.

[1] By Web service orchestration, we mean a coordination infrastructure which allows modular applications to invoke different Web services and combine their results.

1.1 The X$d\pi$ calculus

The X$d\pi$-calculus was defined with the aim of reasoning about dynamic Web data. X$d\pi$ terms represent networks of peers where each peer consists of an XML data repository and a working space where π-like processes are allowed to run. We regard processes 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 descriptions, in the form of scripts, can be included in documents and can be executed by other processes. The definition of X$d\pi$ is parametric with respect to the choice of a specific language of query and update expressions.

Consider again the diagram of our reference architecture for dynamic Web data given in Figure 1. X$d\pi$ models each peer as a *location* with a unique name corresponding to the peer identity (for example its IP address). A whole peer-to-peer system is modelled by the parallel composition of the locations corresponding to its peers, which we call a *network*. For example,the network in (A) could be represented by the term

$$peer_1 \left[tree_1 \,\|\, processes_1 \right] | \ldots | peer_n \left[tree_n \,\|\, processes_n \right]$$

The XML data stored at each peer is represented by an ordered, edge-labelled tree.[2] The choice of using edge-labelled rather than node-labelled trees is merely a matter of style. Following a common practice, we do not represent attributes explicitly, but we model them as edges labelled with the attribute name followed by a leaf containing the attribute value. We also embed pointers and scripts as leaves. In a concrete document, we would also expect them to be represented as attributes. Our results do not depend on these particular representation choices. To keep the model simple, we do not represent data values and XML-specific details such as name-spaces, ids and idrefs. The tree structure, along with scripts and pointers, provides a sufficiently accurate model for our purposes. Figure 2(A) shows a fragment of an XML document containing both a hyperlink and a service call, and Figure 2(B) shows its representation in X$d\pi$ (the translation of the hyperlink and the service call are explained below).

Hyperlinks have been one of the main features responsible for the success of the Web. We abstract the concept of hyperlink into that of *pointer*, a pair consisting of a location name and a query to identify some data in the tree of the named location. For example, in Figure 2 we have translated the destination of the hyperlink "http://xdpi.net/papers/xdpi.pdf" into a pointer of the form *query@location* using the host name "xdpi.net" as the location name, and the path relative to the host "papers/xdpi.pdf" as the query. Pointers are declarative references which can be interpreted uniformly across locations. A pointer does not specify what to do with the data denoted by the query, but typically a process will read the location name and the query from a pointer in order to retrieve some data necessary to continue its execution. Clicking on

[2] Semi-structured data models are often unordered [1], in contrast with the ordered trees of XML documents. In previous work [11], we considered unordered trees, but here we prefer the ordered model, which has a straightforward correspondence to the textual syntax.

Figure 2: Representing XML in Core X$d\pi$

```
⟨data⟩
  ⟨a href = "http://xdpi.net/papers/xdpi.pdf"⟩  Download  ⟨/a⟩
  ⟨call⟩  xdpi.net/getRefs(bibtex, l)  ⟨/call⟩
⟨/data⟩
```

(A) A hyperlink and a service call in XHTML.

```
data[
  a[ href[ papers/xdpi.pdf@xdpi.net ] | Download[ ] ]
  call[ ⟨go xdpi.net . getRefs⟨bibtex[ ], l⟩⟩ ]
]
```

(B) The translation to X$d\pi$.

an HTML hyperlink is a simple example, where the browser process reads the contents of the `href` attribute, retrieves the referenced data, and displays it in the browser window. We assume that the same query makes sense on different locations because we are assuming that all the peers export their data in the same semi-structured format.

Current Web technology is familiar with the use of scripts to provide Web pages with dynamic behaviour. Similarly, we propose to use scripts as a generalization of embedded service calls in the context of Web data integration. Since our scripts are used also for coordination, they are written in the same process language used to describe processes in the working space. A script is a static piece of code with some parameters. Scripts do not reference the global state except for names of locations and services, which are constants with a uniform meaning across the network. For example, the service call "xdpi.net/getRefs(bibtex, l)" embedded in (A) in Figure 2 could be (naively) translated to the script shown in (B), which specifies that a process should go from the local host to the host "xdpi.net" and there invoke the service "getRefs" with a data parameter "bibtex" and a return parameter l. We shall see a more realistic representation of service calls in Section 2.6. Scripts are atomic and cannot be combined together to form other scripts (for example, the parallel composition of two scripts is not defined). [3]

The working space of each peer is modelled by a parallel composition of processes inside the corresponding location. The interface between the working space and the data store is modelled by a single operation for updating or

[3] Some languages such as MetaOCaml [35] and TemplateHaskell [34] provide constructs for multi-stage programming, where pieces of code (possibly containing free variables) can be combined together to form bigger programs, which can then executed. If desired, it is possible to support multi-stage programming in X$d\pi$, defining an XML-like meta-syntax for scripts and interpreting it explicitly using parsing processes in the working space.

querying the local tree. Communication between locations is modelled through process migration, providing a flexible abstraction to model complex coordination protocols, and communication between processes is modelled by π-calculus communication. For example,

$$l\left[\, D_l \,\|\, \mathtt{go}\,\mathrm{xdpi}\,.\,\overline{\mathrm{getRefs}}\langle \mathtt{bibtex}[\,], l\rangle \,\right] \mid xdpi\left[\, D_x \,\|\, !\mathrm{getRefs}(x,y).[\ldots] \,\right]$$

represents a network where a script similar to the one described above is run on location l. In parallel, on location $xdpi$, there is a service (represented by the replicated input $!\mathrm{getRefs}(x,y).[\ldots]$) ready to accept the request with parameters x and y. After migration, the output $\overline{\mathrm{getRefs}}\langle \mathtt{bibtex}[\,], l\rangle$ will be at location $xdpi$, ready to be received by the service input

$$l\left[\, D_l \,\|\, \mathbf{0} \,\right] \mid xdpi\left[\, D_x \,\|\, \overline{\mathrm{getRefs}}\langle \mathtt{bibtex}[\,], l\rangle \mid !\mathrm{getRefs}(x,y).[\ldots] \,\right]$$

The service may perform some computation and return some data, in the form of another output process which migrates to the location determined by the second parameter of getRefs (in this case the original location l).

The definition of X$d\pi$ is minimal, including only the basic operations for asynchronous local communication based on pattern matching, execution of a query-update expression on the local repository, migration, spawning of scripted code and creation of fresh channels. From these, one can derive conditional statements, nondeterministic choices, constructs for parsing and iterating on list-like structures and remote communication in the style of Web services. Using these derived constructs in [18], we used X$d\pi$ to give a precise semantics to AXML-like behaviour, and propose possible extensions.

1.2 Equivalences

The combination of Web services and scripted processes provides the data engineer with many alternative patterns for exchanging information. A theory of semantic equivalence for processes is therefore useful to show, for example, that some complex data-exchange protocol corresponds to some intuitive behaviour. Motivated by this consideration, we have defined network equivalences for X$d\pi$ [11], which dictate when two networks can be considered indistinguishable with respect to the properties represented by a specific set of observations, in our case the attempts to interact with the local store.[4] Network equivalences are parametric with respect to the language used for querying and updating documents (so the generic results are not tied to a particular choice), and can be instantiated to specific cases. Our objective is to define equivalence relations on processes such that, when we place equivalent processes in the same context, we obtain equivalent networks. Since we want to use the equivalences for example

[4] In [18] we considered as observations the shape of the data tree of a location, the presence of output actions in a process, and the attempts to interact with the local store, and we studied the formal relationship between the corresponding equivalences. In this paper, we focus on the latter kind of observation because the resulting equivalence coincides or implies the ones resulting from the other observations.

to optimize the interaction between different locations, we must compare several located processes at the same time, which possibly share some private channel names. Moreover, we want to make sure that the behaviour of the processes is robust with respect to changes in the data stored in each location and to the behaviour of other processes running in parallel. It is not straightforward to carry on the kind of reasoning mentioned above directly on X$d\pi$ terms, because locations, processes and data are closely inter-twined. Instead, based on the ideas presented in [19], we propose a calculus called Core X$d\pi$ which serves as an alternative representation of X$d\pi$, where we locate processes explicitly and separate data from processes.

From X$d\pi$ to Core X$d\pi$. Core X$d\pi$ is tailored to be exactly as expressive as X$d\pi$ (a proof can be found in [18]), and is suitable for expressing a partial specification of a network by means of located processes running in parallel, possibly sharing private names. A X$d\pi$ network consists of locations containing trees and processes. In contrast, a Core X$d\pi$ network consists of a store, containing all the trees indexed by their location information, plus all the processes, augmented with explicit location information associated to every action they perform. For example, the X$d\pi$ network described earlier is described in Core X$d\pi$ as

$$\begin{pmatrix} (\{l \mapsto D_l\}, \{xdpi \mapsto D_x\}), \\ l{\cdot}\texttt{go}\,\mathrm{xdpi}\,.\,\overline{xdpi{\cdot}\mathrm{getRefs}}\langle \texttt{bibtex}[\,], l\rangle \mid !xdpi{\cdot}\mathrm{getRefs}(x,y).[\ldots] \end{pmatrix}$$

The store $(\{l \mapsto D_l\}, \{xdpi \mapsto D_x\})$ maps each location to its data. Each process action is prefixed by the location where the action takes place: for example, $l{\cdot}\texttt{go}\,\mathrm{xdpi}\,.[\ldots]$ shows that the migration step towards location $xdpi$ originates from location l; after migration the process is located at $xdpi$. Although Core X$d\pi$ is explicitly located, and does not necessarily require a migration operation, we left that as part of the syntax to have a closer correspondence with X$d\pi$, and to serve as a hook for future security-sensitive checks.

In Section 3.1, we define the correspondent of X$d\pi$ network equivalence for Core X$d\pi$, and we define process equivalence as the closure of network equivalence under composition with different stores. This relation is hard to use directly because it requires a costly property of closure under all contexts. Instead, we define a coinductive equivalence relation (called *domain bisimilarity*) which does not quantify over contexts and which entails process equivalence.

The definition of domain bisimilarity is non-standard, due to the fact that scripts (which can appear in data) are part of the values, and process equivalences are sensitive to the set of locations constituting the network. We address these two problems by adapting existing techniques for translating messages containing scripts into ones where each script is replaced by a first-order value [30, 17], and by generalizing the notion of bisimulation to families of relations indexed by sets of locations.

As an application of our techniques, we use bisimilarity to study some communication patterns used by servers in distributed query systems to answer queries from clients. In Core X$d\pi$, distributed queries take the form of processes which retrieve and combine data from different locations by using remote

communication and local requests. We show that some existing patterns [29] can be combined together obtaining a flexible infrastructure which is provably equivalent to an intuitive specification of the intended behaviour. By exploiting process migration, we also propose a new communication pattern, and we show that it is behaviourally equivalent to a naive, less efficient one.

Our work is one of the first attempts to integrate the study of mobile processes and semi-structured data, and is characterized by its emphasis on dynamic data. It is the first investigation of equivalence properties for (higher-order) data-centric applications based on the Web.

1.3 Related work

X$d\pi$ was developed independently from the AXML system of Abiteboul *et al.* [33], which we have described above. The ubQL distributed query language of Sahuguet and Tannen [27] instead, constituted a source of inspiration for the design of X$d\pi$. ubQL is built by adding process manipulation primitives to any "host" query language. These primitives, inspired by the π-calculus[5], are used in a *deployment phase* to set up a network of processes which, in a successive *execution phase*, will query local repositories and forward their results to other sites, thus implementing a global query execution plan. ubQL processes can deal with streaming data, but there is no support for concurrent execution of query-processes on the same site (so in principle the system may not be able to execute more than one global query at a time). The main influences of ubQL on the design of X$d\pi$ were on the choices of separating the queries from the process primitives, and maintaining independence from a specific query language. Also our examples on distributed query patterns of Section 4 are inspired by ubQL. Overall, X$d\pi$ and ubQL have a significantly different focus and are studied using different methodologies. For example, an important part of the work on ubQL is the study of algorithms for query installation based on cost estimates, which we do not address, whereas behavioural equivalences are not studied in ubQL. Both AXML and ubQL are studied from a data-management viewpoint, which our process algebraic techniques could complement nicely. There are many specific issues which are important in databases, such as the use of meta-data to guide the optimization of queries, which we do not study. Instead we give a formal semantics to the distributed interaction between query-processes, arguing about their equivalence and providing a framework on which to base formal studies of security properties.

We now consider work related to the process algebraic approach. To the best of our knowledge, the only work relating the π-calculus with XML which pre-dates ours is the Iota concurrent XML scripting language of Bierman and Sewell [4], used to program Home Area Networks. Iota is a strongly typed functional language with concurrency primitives inspired by the π-calculus. Although the language has a formal semantics, its behavioural theory has not been

[5]The influence born by the π-calculus on ubQL can be better appreciated considering the preliminary joint work with Pierce [28].

studied. The programs for Home Area devices written in Iota are designed to run on the same Home Area server, and the communication with physical devices is modelled through input and output on special channels: distribution is not represented explicitly. Moreover, as opposed to X$d\pi$, the application domain of Home Area Network programming is more control-oriented than data-oriented: there is no explicit representation of stores, which are central to our approach. Brown, Laneve and Meredith [6] have recently defined an (untyped) extension of the π-calculus with native XML datatypes called πDuce. They compare its expressivity to that of the functional language XDuce [16], and also consider a higher-order extension which enables dynamic content in documents. An interesting idea underlying the design of πDuce is that processes and data share a similar tree-like structure, and can inhabit the same semantic universe. The authors show a very simple encoding of an evaluator for the subset of the language without new name generation, into the language itself: the execution of processes represented as nested document elements can be simulated in the language. A similar approach could be taken in X$d\pi$ to represent scripts as semi-structured data. In the case of dynamic Web data though, it is better to hide the internal structure of processes from the queries, so that one can replace a process by an equivalent one whilst preserving the observable behaviour of the system as a whole. Castagna, De Nicola and Varacca [8] propose $\mathbb{C}\pi$, a π-calculus extended with pattern matching and tuples of values (XML values can be represented through an encoding). The language comes with a very expressive type system featuring intersection and input-output types. The language itself is not distributed and does not include a concept of store. Acciai and Boreale [2] have recently proposed XPi, an extension of the asynchronous π-calculus with code mobility and ML-like pattern matching of structured values. A combination of static and dynamic typing ensures that each channel always exchanges values of the same types, which describe the partial structure of documents. Pattern matching plays a lesser role in X$d\pi$, although it could be easily extended to the more expressive form adopted in XPi. Query expressions instead, which are separate entities from processes, are the primary means to extract information from XML trees.

2 Core X$d\pi$

In this Section we introduce the formal definition of the syntax and semantics of Core X$d\pi$. We begin with trees, data and queries, and then we pass to networks and processes.

Figure 3: Syntax: trees and data

$\boldsymbol{T}, \boldsymbol{S}$ ::=		tree terms
	$\boldsymbol{E} \shortmid \boldsymbol{T}$	branch $\boldsymbol{E}$ composed with tree $\boldsymbol{T}$
	$\varnothing$	empty tree
	x	tree variable
$\boldsymbol{E}, \boldsymbol{F}$ ::=	$\mathtt{a}[\,\boldsymbol{V}\,] \mid x$	branch with edge label $\mathtt{a}$ and data $\boldsymbol{V}$, or variable
$\boldsymbol{U}, \boldsymbol{V}$ ::=		data terms
	$\boldsymbol{T}$	tree $\boldsymbol{T}$
	$\boldsymbol{p}@\boldsymbol{l}$	pointer to location $\boldsymbol{l}$ with query $\boldsymbol{p}$
	$\langle \boldsymbol{A} \rangle$	script $\boldsymbol{A}$
$\boldsymbol{l}$ ::=	$l \mid x$	location name or variable
$\boldsymbol{p}$ ::=	$p \mid x$	query (see Definition 2.1) or variable
$\boldsymbol{A}$ ::=	$A \mid x$	script (see Figure 5) or variable

$\mathtt{a}, \mathtt{b}, \mathtt{c} \in \mathcal{E}$	(Edge Labels)
$l, m \in \mathcal{L}$ (countably infinite)	(Locations Names)
$p, q \in \mathcal{Q}$	(Queries)
$x, y, z \in \mathcal{V}$	(Variables)
$E, F \in \mathcal{B} \stackrel{\text{def}}{=} \{\boldsymbol{E} \;:\; \mathit{fv}(\boldsymbol{E}) = \emptyset\}$	(Branches)
$U, V \in \mathcal{D} \stackrel{\text{def}}{=} \{\boldsymbol{V} \;:\; \mathit{fv}(\boldsymbol{V}) = \emptyset\}$	(Data)
$T, S \in \mathcal{T} \stackrel{\text{def}}{=} \{\boldsymbol{T} \;:\; \mathit{fv}(\boldsymbol{T}) = \emptyset\}$	(Trees)

Function fv is defined in Figure 23.
Notation: $\mathtt{a}[\,] \stackrel{\text{def}}{=} \mathtt{a}[\,\varnothing\,] \quad E_1 \shortmid \ldots \shortmid E_n \stackrel{\text{def}}{=} E_1 \shortmid \ldots \shortmid E_n \shortmid \varnothing$.

2.1 Trees, data and queries

We represent semi-structured data using ordered labelled trees.[6] The formal definition is given in Figure 3 (we use italic bold letters for arbitrary terms, which can contain variables (such as $\boldsymbol{T}$), and plain italic letters for closed terms (such as T)).

Trees and data. We represent a tree as a $\varnothing$-terminated list of branches $E_1 \shortmid \ldots \shortmid E_n \shortmid \varnothing$ (abbreviated with $E_1 \shortmid \ldots \shortmid E_n$) which start from the root. Each branch E_i has the form $\mathtt{a}[\,V\,]$ and denotes an edge labelled $\mathtt{a}$ leading to a node containing the data V. A data item can be a subtree T, a pointer $p@l$ referencing the data selected at location l by query p (described below), or a script

[6] A hypothetical encoding of trees into processes, which could be interesting in itself, would introduce unnecessary complexity, making it hard to reason directly on trees like we can do, thanks to our direct representation, in the equivalences of Chapter 3.

Figure 4: Syntax: Core X$d\pi$ networks and contexts

$D, B \in \mathcal{S} \stackrel{\text{def}}{=} \mathcal{L} \rightharpoonup \mathcal{T}$ (Stores)

$N, M \in \mathcal{N} \stackrel{\text{def}}{=} \{(D, P) : D \in \mathcal{S},\ P \in \mathcal{P},\ \mathit{dom}(P) \subseteq \mathit{dom}(D)\}$ (Networks)

Function dom is defined in Figure 22, processes are defined in Figure 5.

$\langle A \rangle$ (described in Section 2.3) which can be executed to collect data or perform coordination tasks. We show an example of a tree containing a script and a pointer:

$$\mathtt{a}[\,\mathtt{b}[\,\mathtt{c}[\,\langle A \rangle\,]\,|\,\mathtt{d}[\,p@l\,]\,]\,|\,\mathtt{e}[\,]\,]$$

We use the same identifiers $x, y, z, \ldots$ to range over all variables. When necessary, the kind of each variable can be understood by the place where the variable occurs.

Queries. First of all, it is important to clarify that, in this paper, we use the word "queries" to mean expressions used to query *or update* a tree. X$d\pi$ is parametric on the choice of a particular query-update language chosen, as long as it is a language of expressions which can be evaluated against a tree to obtain a new tree (the result of updating the tree) and some data (the list of trees resulting from querying the tree). The only conditions that we need to impose on such a language are that the application of a substitution to a query must be well-defined and yield a query. The reasons why we need to define substitutions on queries will be clear after describing the semantics of processes in Section 2.3. In Section 3, we will impose additional conditions required to ensure that a query language is also compatible with our definitions of semantic equivalences.

Definition 2.1 (Query Language) *A* query language *consists of a triple* $(\mathcal{Q}, \mathit{fv}, \mathfrak{E})$ *where* $\mathcal{Q}$ *is a set of queries ranged over by* $p, q, \ldots$*, together with a function* $\mathit{fv} : \mathcal{Q} \to \wp(\mathcal{V})$ *giving the free variables of each query, and an evaluation function* $\mathfrak{E} : (\mathcal{Q} \times \mathcal{T}) \rightharpoonup \mathcal{T} \times \mathit{lists}(\mathcal{D})$*, which, given a query and a tree, returns an updated tree and a finite list of results. Additionally,* $\mathcal{Q}$ *must be closed under substitution.*

Note that in the definition above the evaluation of queries is a partial function. This generality accounts for both the cases of ill-formed queries, which may not have a precise semantics, and Turing-equivalent query languages, which may not terminate. In Section 2.5, we give a concrete query language which will be used in the examples.

2.2 Networks

A Core X$d\pi$ network represents a peer-to-peer system, where each location corresponds to a peer. Each peer can communicate with any other peer, and has

a unique name. 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. The formal definition is given in Figure 4.[7] For example, in the network $(\{l \mapsto T\}, P)$, the term $\{l \mapsto T\}$ says that the store of the peer at location l is the tree T, and the term P represents the processes running on the peer, which contain explicit location information. Interaction between processes and data is always local, as we shall see later from rule (CRed Request) in Figure 7. In Figure 22, we define the function *dom* giving the domain of both networks, stores and processes. By definition, the domain of a network is the domain of the store, and a network is well-formed if the domain of the process is contained in the domain of the store.

2.3 Processes

The formal definition for Core X$d\pi$ processes is given in Figure 5. We now describe the technical features of each construct.

Communication. The *output* process $\overline{l{\cdot}c}\langle\tilde{v}\rangle$ denotes a vector of values $\tilde{v}$ waiting to be sent via channel c at location l, the *input* process $l{\cdot}c(\widetilde{\pi}).P$ waits to receive values matching the patterns $\widetilde{\pi}$ from an output process via channel c at l, and the *replicated input* is standard.[8] The well-formedness condition $wf(P)$ requires that the continuation on an input (or replicated input) process must be located at the same location where the input is defined. Channel names are partitioned into *private* and *service* channel names. The private channels denote "usual" π-calculus channels, which are typically used for coordination, and which can be kept secret in order to protect a protocol from external interferences. The service channels denote those channels which are used to implement the services which a peer offers to other peers, and which therefore are not meant to be restricted and can be referenced inside scripts.

Pattern matching. Both trees and pointers are data terms which processes need to parse. For this reason, we have added to π-calculus communication a very simple form of pattern matching. Patterns $\pi_1, \ldots, \pi_n$ are terms containing distinct variables which are instantiated, if pattern matching succeeds, with the values found in the corresponding position in the term to be matched. Our patterns do not include regular or recursive expressions, and we will avoid algorithmic issues by simply requiring the guessing of an appropriate substitution in order for pattern matching to take place. Pattern matching for XML-like data is an active research topic, which is orthogonal to our concerns. We believe that the specialized techniques studied elsewhere can be adapted to our setting. Our processes use patterns to parse data, and queries to query trees. This conceptual separation does not exclude the possibility for the query language to be

[7] The creation of new peers is not an operation which can be performed from within a system, and therefore we do not provide an operation to create new locations. Nevertheless, we will be able to carry on compositional reasoning, hence analyze networks with respect to arbitrary additions of peers.

[8] The communication constructs, which use polyadic synchronization, were inspired by the $^e\pi$-calculus [7].

Figure 5: Syntax: Core X$d\pi$ processes

$\boldsymbol{P}, \boldsymbol{Q}, \boldsymbol{R}$ ::=		process terms
	$\mathbf{0}$	nil process
	$\boldsymbol{P} \mid \boldsymbol{P}$	composition of processes
	$(\nu\, c)\boldsymbol{P}$	private channel c with scope $\boldsymbol{P}$
	$\overline{\boldsymbol{l}\cdot\boldsymbol{c}}\langle\widetilde{\boldsymbol{v}}\rangle$	at $\boldsymbol{l}$, output values $\widetilde{\boldsymbol{v}}$ on $\boldsymbol{c}$
	$\boldsymbol{l}\cdot\boldsymbol{c}(\widetilde{\pi}).\boldsymbol{P}$	at $\boldsymbol{l}$, input on $\boldsymbol{c}$ of $\widetilde{\pi}$, continue with $\boldsymbol{P}$
		$(\mathit{distinct}(\widetilde{\pi}),\ \mathit{fv}(\widetilde{\pi}) \cap \mathit{fv}(\boldsymbol{l}) = \emptyset)$
	$!\boldsymbol{l}\cdot\boldsymbol{c}(\widetilde{\pi}).\boldsymbol{P}$	lazy replication of an input process
		$(\mathit{distinct}(\widetilde{\pi}),\ \mathit{fv}(\widetilde{\pi}) \cap \mathit{fv}(\boldsymbol{l}) = \emptyset)$
	$\boldsymbol{l}\cdot\mathsf{go}\,\boldsymbol{m}.\boldsymbol{P}$	at $\boldsymbol{l}$, go to $\boldsymbol{m}$, continue with $\boldsymbol{P}$
	$\boldsymbol{A} \circ \langle \boldsymbol{l}, \widetilde{\boldsymbol{v}} \rangle$	at $\boldsymbol{l}$, run script $\boldsymbol{A}$ with parameters $\widetilde{\boldsymbol{v}}$
	$\boldsymbol{l}\cdot\mathsf{req}_{\boldsymbol{p}}\langle\boldsymbol{c}\rangle$	at $\boldsymbol{l}$, request query $\boldsymbol{p}$ with return channel $\boldsymbol{c}$

$\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$ ::= $c \mid \mathsf{c} \mid x$	private/service channel, or variable
$\boldsymbol{v}$ ::= $\boldsymbol{c} \mid \boldsymbol{l} \mid \boldsymbol{p} \mid \boldsymbol{A} \mid \boldsymbol{E} \mid \boldsymbol{T}$	value terms

$a, b, c \in \mathcal{C}_{\mathrm{p}}$ (countably infinite) (Private Channels)

$\mathsf{a}, \mathsf{b}, \mathsf{c} \in \mathcal{C}_{\mathrm{s}}$ (Service Channels)

$v, u \in \mathcal{U} \stackrel{\text{def}}{=} \{\boldsymbol{v} \ :\ \mathit{fv}(\boldsymbol{v}) = \emptyset\}$ (Values)

$\pi \in \mathcal{K} \stackrel{\text{def}}{=} \mathcal{V} \cup \{\boldsymbol{V} \ :\ \mathit{cval}(\boldsymbol{V}) = \emptyset \ \mathit{and}\ \mathit{distinct}(\boldsymbol{V})\}$ (Patterns)

$P, Q, R \in \mathcal{P} \stackrel{\text{def}}{=} \{\boldsymbol{P} \ :\ \mathit{fv}(\boldsymbol{P}) = \emptyset \ \mathit{and}\ \mathit{wf}(\boldsymbol{P})\}$ (Processes)

$A \in \mathcal{A} \stackrel{\text{def}}{=} \left\{ (x, \widetilde{\pi})\boldsymbol{P} \ :\ \begin{array}{c} \mathit{fn}(\boldsymbol{P}) = \emptyset, \mathit{fv}(\boldsymbol{P}) \subseteq \mathit{fv}(x, \widetilde{\pi}), \\ \mathit{distinct}(x, \widetilde{\pi}),\ \mathit{dom}(\boldsymbol{P}) = \{x\} \end{array} \right\}$ (Scripts)

We define wf, $\mathit{distinct}$, dom, cval in Figure 22 and fv, fn in Figure 23.
Well formedness ensures that the continuation of a process is properly located (for example $\mathit{wf}(\boldsymbol{l}\cdot\boldsymbol{c}(\widetilde{\pi}).\boldsymbol{P})$ and $\mathit{wf}(\boldsymbol{m}\cdot\mathsf{go}\,\boldsymbol{l}.\boldsymbol{P})$ require that $\boldsymbol{P}$ is located at $\boldsymbol{l}$).
Notation: $\boldsymbol{l}\cdot\boldsymbol{P} \stackrel{\text{def}}{=} \boldsymbol{P}$ if $\mathit{dom}(\boldsymbol{P}) = \{\boldsymbol{l}\}$; $\boldsymbol{l}\cdot\overline{\boldsymbol{m}\cdot\boldsymbol{c}}\langle\widetilde{\boldsymbol{v}}\rangle \stackrel{\text{def}}{=} \boldsymbol{l}\cdot\mathsf{go}\,\boldsymbol{m}.\overline{\boldsymbol{m}\cdot\boldsymbol{c}}\langle\widetilde{\boldsymbol{v}}\rangle$.

based on pattern matching itself.

Scripts. A script $(x, \widetilde{\pi})P$ represents the code of P parameterized on x, a placeholder for the location where the script is going to be run, and $\widetilde{\pi}$, other optional parameters of P. By the side condition on the free variables of scripts, we impose that scripts remain statically defined until they are deployed dynamically by instantiation of their parameters. The application construct $A \circ \langle l, \widetilde{v} \rangle$ passes the parameters $\widetilde{v}$ to the script A and runs it in the working space of l. Note that application is defined only when the first parameter passed to the

script is a location. Communication in Core X$d\pi$ is higher-order, in the sense that processes may send scripts over channels, possibly as leaves inside trees.

Migration. Process migration, which we represent explicitly, models communication across locations: the process $m{\cdot}\mathsf{go}\, l.P$ represents a (higher-order) message from m addressed to l containing a request to run the (closed) code P. When P is an output process, we use the abbreviation $l{\cdot}\overline{m{\cdot}c}\langle\widetilde{v}\rangle$ for $l{\cdot}\mathsf{go}\, m.\overline{m{\cdot}c}\langle\widetilde{v}\rangle$. The well-formedness condition $\mathit{wf}(P)$ requires that the continuation process must be correctly located at the destination location. Due to the peer-to-peer nature of our domain, each location is ready to receive and run any incoming code, so we do not need to provide an explicit operation to run a received process. In some cases, it may also be desirable to give control to each location regarding which code to accept and which to refuse. We leave that task to an eventual superimposed security infrastructure. Using an asynchronous form of communication offers a simple way to model failures within the system. The success of a migration step just depends on the existence of location l. In contrast, the migration rules for other mobile calculi (for example $d\pi$ [14]) assume that migration is always possible. Our choice has an important effect on the behavioural equivalences studied in Section 3.1.

Interaction with the store. Core X$d\pi$ processes access the local tree by using a request operation $l{\cdot}\mathsf{req}_p\langle c\rangle$ parametric in a query-update expression p and a channel c. The effect of evaluating expression p is to modify the local tree and to return a list of query results on the specified channel. Research on query and update languages for XML is still very active [25], and an in-depth study goes beyond the scope of this paper. Therefore, rather than committing to any particular choice, we parameterize our definitions with respect to an arbitrary query language. Our request operation is defined for any query which given a tree returns an updated tree and a list of results.

2.4 Reduction semantics

Network contexts are pairs of process and store contexts (see Figure 6). For example, if $C_{\mathcal{N}} = (- \uplus B, (\nu\, c)-)$ then $C_{\mathcal{N}}[(D, P)] = (D \uplus B, (\nu\, c)P)$. We omit the subscripts from contexts when no ambiguity can arise. The reduction relation $\longrightarrow$ for Core X$d\pi$ describes process interaction, the interaction between processes and data, and the movement of processes across locations. The formal definition is given in Table 7. It relies on a standard notion of structural congruence for processes and networks defined in the Appendix, in Figure 21.

Rules (CRed Context) and (CRed Struct) are standard contextual rules which allow reduction under parallel composition, restriction and structural congruence. There are two rules for process movement between locations: rule (CRed Stay) describes the case where the process is already at the target location, and rule (CRed Go) allows a process $l{\cdot}\mathsf{go}\, m.P$ to move from l to m. Rule (CRed Com) states that if an output $\overline{l{\cdot}a}\langle\widetilde{v}\rangle$ and an input $l{\cdot}a(\widetilde{\pi}).\boldsymbol{P}$ on the same channel a are in the same location l (part of the store), and the values $\widetilde{v}$ match the input patterns $\widetilde{\pi}$ (there is a substitution a σ such that $\widetilde{v} = \widetilde{\pi}\sigma$), then communication

Figure 6: Syntax: contexts

$$\mathcal{K}_{\mathcal{P}} \stackrel{\text{def}}{=} C_{\mathcal{P}}[-] ::= \; - \mid P \mid C_{\mathcal{P}}[-] \mid C_{\mathcal{P}}[-] \mid P \mid (\nu\, c) C_{\mathcal{P}}[-] \qquad \text{(Process Contexts)}$$

$$\mathcal{K}_{\mathcal{S}} \stackrel{\text{def}}{=} C_{\mathcal{S}}[-] ::= \; - \mid C_{\mathcal{S}}[-] \uplus D \qquad \text{(Store Contexts)}$$

$$\mathcal{K}_{\mathcal{N}} \stackrel{\text{def}}{=} C_{\mathcal{N}}[-,-] ::= (C_{\mathcal{S}}[-], C_{\mathcal{P}}[-]),\ \mathit{dom}(C_{\mathcal{P}}) \subseteq \mathit{dom}(C_{\mathcal{S}}) \qquad \text{(Networks Contexts)}$$

$$(C_{\mathcal{S}}[-], C_{\mathcal{P}}[-])[(D, P)] \stackrel{\text{def}}{=} (C_{\mathcal{S}}[D], C_{\mathcal{P}}[P]) \qquad \text{(Context Application)}$$

Notation: $\{l \mapsto T\}(l) \stackrel{\text{def}}{=} T$; $\quad (D \uplus B)(l) \stackrel{\text{def}}{=} \begin{cases} D(l) & \text{if } l \in \mathit{dom}(D) \\ B(l) & \text{if } l \in \mathit{dom}(B) \end{cases}$.

Convention: $D \uplus B$ is defined if and only if $\mathit{dom}(D) \cap \mathit{dom}(B) = \emptyset$.
Function dom is defined in Figure 22.

Figure 7: Semantics: reduction relation for Core X$d\pi$

$$(\{l \mapsto T\}, l{\cdot}\mathsf{go}\, l.P \mid Q) \;\longrightarrow\; (\{l \mapsto T\}, P \mid Q) \qquad \text{(CRed Stay)}$$

$$(\{l \mapsto T\} \uplus \{m \mapsto S\}, l{\cdot}\mathsf{go}\, m.P \mid Q) \;\longrightarrow\; (\{l \mapsto T\} \uplus \{m \mapsto S\}, P \mid Q) \qquad \text{(CRed Go)}$$

$$(\{l \mapsto T\}, \overline{l{\cdot}c}\langle\widetilde{\pi}\sigma\rangle \mid l{\cdot}c(\widetilde{\pi}).\boldsymbol{P} \mid Q) \;\longrightarrow\; (\{l \mapsto T\}, \boldsymbol{P}\sigma \mid Q) \qquad \text{(CRed Com)}$$

$$(\{l \mapsto T\}, \overline{l{\cdot}c}\langle\widetilde{\pi}\sigma\rangle \mid !l{\cdot}c(\widetilde{\pi}).\boldsymbol{P} \mid Q) \;\longrightarrow\; (\{l \mapsto T\}, !l{\cdot}c(\widetilde{\pi}).\boldsymbol{P} \mid \boldsymbol{P}\sigma \mid Q) \qquad \text{(CRed Com!)}$$

$$(\{l \mapsto T\}, (x, \widetilde{\pi})\boldsymbol{P} \circ \langle l, \widetilde{\pi}\sigma\rangle \mid Q) \;\longrightarrow\; (\{l \mapsto T\}, \boldsymbol{P}\{{}^{l}/_{x}\}\sigma \mid Q) \qquad \text{(CRed Run)}$$

$$\frac{\mathfrak{E}(p, T) = (T', U_1 \mid \ldots \mid U_n \mid \varnothing)}{(\{l \mapsto T\}, l{\cdot}\mathsf{req}_p\langle c\rangle \mid Q) \longrightarrow (\{l \mapsto T'\}, \overline{l{\cdot}c}\langle \mathtt{r}[\,U_1\,] \mid \ldots \mid \mathtt{r}[\,U_n\,] \mid \varnothing\rangle \mid Q)} \qquad \text{(CRed Request)}$$

$$\text{(CRed Context)}\ \frac{N \longrightarrow N'}{C_{\mathcal{N}}[N] \longrightarrow C_{\mathcal{N}}[N']} \qquad \text{(CRed Struct)}\ \frac{N \equiv M \longrightarrow M' \equiv N'}{N \longrightarrow N'}$$

Convention: in this table c ranges over $\mathcal{C}_{\mathrm{p}} \cup \mathcal{C}_{\mathrm{s}}$.

takes place and execution proceeds with $\boldsymbol{P}\sigma$. Rule (CRed Com!) is similar, but leaves the replicated input process $!l{\cdot}a(\widetilde{\pi}).\boldsymbol{P}$ in place for further use. We show an example of the communication of a private channel over a service channel below. The reduction step involves the use of structural congruence to extend the scope of the restricted name before communication (*scope extrusion*):

$$\begin{aligned} &(\{l \mapsto T\}, (\nu\, c)(\overline{l{\cdot}\mathtt{a}}\langle c, \mathtt{b}[\,]\rangle) \mid l{\cdot}\mathtt{a}(x, \mathtt{b}[\,y\,]).(\overline{l{\cdot}x}\langle y\rangle \mid \boldsymbol{P})) \\ \longrightarrow &(\{l \mapsto T\}, (\nu\, c)(\overline{l{\cdot}c}\langle\varnothing\rangle \mid \boldsymbol{P}\{{}^{c}/_{x}, {}^{\varnothing}/_{y}\})) \end{aligned}$$

Rule (CRed Run) runs a script, passing as the first parameter the name of the location where it is going to run. Rule (CRed Request) applies the query denoted

Figure 8: Syntax: Sam queries

$\widehat{p}, \widehat{q}$ **::=**		path expressions
	ε	empty path
	$\mathtt{A}/\widehat{p}$	follow an edge with label in set $\mathtt{A}$, then $\widehat{p}$
	$\Box\widehat{p}$	follow occurrences of $\widehat{p}$ anywhere
p, q **::=**	$\widehat{p}(\pi)\boldsymbol{V}$	queries: follow $\widehat{p}$, match π and insert $\boldsymbol{V}$

$\mathtt{A}, \mathtt{B} \in \wp(\mathcal{E})$ (Label Sets)
$p, q \in \mathcal{Q}$ (Queries)

Notation: $\mathsf{copy}_{\widehat{p}}(\pi) \stackrel{\text{def}}{=} \widehat{p}(\pi)\pi$; $\mathsf{cut}_{\widehat{p}}(\pi) \stackrel{\text{def}}{=} \widehat{p}(\pi)\varnothing$; $\mathsf{paste}_{\widehat{p}}\langle E\rangle \stackrel{\text{def}}{=} \widehat{p}(x)E\mathtt{|}x$; $*/\widehat{p} \stackrel{\text{def}}{=} \mathcal{E}/\widehat{p}$; $\mathtt{a}/\widehat{p} \stackrel{\text{def}}{=} \{\mathtt{a}\}/\widehat{p}$.
Convention: we omit a trailing ε from a path, for example we write $\mathtt{A}/$ for $\mathtt{A}/\varepsilon$.

by $\boldsymbol{p}$ on the local tree T, obtaining an updated tree T' which replaces T, and a list of results $U_1\mathtt{|}...\mathtt{|}U_n\mathtt{|}\varnothing$ which is turned into a tree of results labelled with $\mathtt{r}$ (a reserved label used to denote results) sent on channel c at l. We show a simple example of update, supposing that p is a query which extracts from a tree the data found by following the path $\mathtt{a}/\mathtt{b}$:

$$(\{l \mapsto \mathtt{a}[\,\mathtt{b}[\,V_1\,]\mathtt{|}\mathtt{b}[\,V_2\,]\mathtt{|}\mathtt{a}[\,U\,]\,]\}, l{\cdot}\mathsf{req}_p\langle c\rangle \mid P)$$
$$\longrightarrow\longrightarrow (\{l \mapsto \mathtt{a}[\,\mathtt{b}[\,]\mathtt{|}\mathtt{b}[\,]\mathtt{|}\mathtt{a}[\,U\,]\,]\}, \overline{l{\cdot}c}\langle \mathtt{r}[\,V_1\,]\mathtt{|}\mathtt{r}[\,V_2\,]\rangle \mid P)$$

Note that the subtrees V_i removed from the store are returned as results by the output on c.

2.5 A sample query and update language

In this section, we define a particular query and update language inspired by XPath [37] which will be used in the examples later on. The result of evaluating a query against a piece of data (when defined) is a pair consisting of a new piece of data, intended to replace the original one, and a list of results, intended to be used by the continuation of the process that executed the query.

The syntax for queries is given in Figure 8. A query $\widehat{p}(\pi)\boldsymbol{V}$ is formed by a path expression $\widehat{p}$ followed by an update expression $(\pi)\boldsymbol{V}$. A path expression $\mathtt{A}/p$ applied to a tree $\mathtt{a}[\,V\,]\mathtt{|}T$ evaluates p on V if $\mathtt{a}$ is in the set $\mathtt{A}$, and evaluates itself on the rest of the tree T. A recursive expression $\Box p$ applied to a tree $\mathtt{a}[\,V\,]\mathtt{|}T$ evaluates p on any node in the tree in a bottom up fashion.[9] First it evaluates $\Box p$ on V and T, obtaining the updated items V' and T', then it

[9] Suppose we chose a top-down strategy instead. A simple query like "add a subtree $\mathtt{a}[\,]$ inside any branch labelled $\mathtt{a}$" on the tree $\mathtt{a}[\,]$ should be ruled out, because its evaluation diverges: each time a new subtree is added there is a new branch to update. Inconsistencies

Figure 9: Semantics: query evaluation for Sam

$$\mathfrak{E}((\pi)\boldsymbol{V}, \pi\sigma) = (\boldsymbol{V}\sigma, \pi\sigma \shortmid \varnothing) \qquad \text{(Eval Match)}$$

$$\mathfrak{E}((\pi)\boldsymbol{V}, U) = (U, \varnothing) \quad \textit{if } U \neq \pi\sigma \qquad \text{(Eval Mismatch)}$$

$$\frac{\mathtt{a} \in \mathtt{A} \quad \mathfrak{E}(p, V) = (V', L) \quad \mathfrak{E}(\mathtt{A}/p, T) = (T', L')}{\mathfrak{E}(\mathtt{A}/p, \mathtt{a}[\,V\,] \shortmid T) = (\mathtt{a}[\,V'\,] \shortmid T', L \shortparallel L')} \qquad \text{(Eval Edge Follow)}$$

$$\frac{\mathtt{a} \notin \mathtt{A} \quad \mathfrak{E}(\mathtt{A}/p, T) = (T', L')}{\mathfrak{E}(\mathtt{A}/p, \mathtt{a}[\,V\,] \shortmid T) = (\mathtt{a}[\,V\,] \shortmid T', L')} \qquad \text{(Eval Edge Discard)}$$

$$\mathfrak{E}(\mathtt{A}/p, U) = (U, \varnothing) \quad \textit{if } U \neq \mathtt{a}[\,V\,] \shortmid T \qquad \text{(Eval Not Edge)}$$

$$\frac{\begin{array}{c}\mathfrak{E}(\Box p, V) = (V', L) \\ \mathfrak{E}(\Box p, T) = (T', L') \\ \mathfrak{E}(p, \mathtt{a}[\,V'\,] \shortmid T') = (T'', L'')\end{array}}{\mathfrak{E}(\Box p, \mathtt{a}[\,V\,] \shortmid T) = (T'', L \shortparallel L' \shortparallel L'')} \qquad \text{(Eval Anywhere Tree)}$$

$$\mathfrak{E}(\Box p, U) = \mathfrak{E}(p, U) \quad \textit{where } U \neq \mathtt{a}[\,V\,] \shortmid T \qquad \text{(Eval Anywhere Else)}$$

The infix operator on lists $\shortparallel$ appends its second argument to its first argument. Query evaluation $\mathfrak{E}$ is a partial function from $\mathcal{Q} \times \mathcal{D}$ to $\mathcal{D} \times \mathit{lists}(\mathcal{D})$.

evaluates p on $\mathtt{a}[\,V'\,] \shortmid T'$, combining the results together. The update expression $(\pi)\boldsymbol{V}$ is a binding pattern π followed by a data term $\boldsymbol{V}$. When $(\pi)\boldsymbol{V}$ is applied to each data item U selected by $\widehat{p}$, such that $U = \pi\sigma$ for a closing substitution σ, the expression returns the new data item $\boldsymbol{V}\sigma$ and the result U. If there is no such substitution, the expression returns the original data U and the empty result $\varnothing$. The formal definition of query evaluation is given in Figure 9.

Definition 2.2 (Sample Query Language) *The sample query language* Sam *is the triple* $(\mathcal{Q}, \mathit{fv}, \mathfrak{E})$ *where* $\mathcal{Q}$ *is defined in Figure 8,* $\mathfrak{E}$ *is defined in Figure 9 and* fv *is defined as* $\mathit{fv}(\widehat{p}(\pi)\boldsymbol{V}) = \mathit{fv}(\boldsymbol{V}) \setminus \mathit{fv}(\pi)$.

We show now how Sam is capable of expressing some intuitive tree manipulations, using the macros defined in Figure 8. The query $q = \mathsf{copy}_{\mathtt{b}/}(y@x)$ reads the query and the location of any pointer contained in branches labelled `b` at the top level. For example,

$$\mathfrak{E}(q, \mathtt{b}[\,T\,] \shortmid \mathtt{b}[\,p@l\,] \shortmid \mathtt{b}[\,p'@l'\,]) = (\mathtt{b}[\,T\,] \shortmid \mathtt{b}[\,p@l\,] \shortmid \mathtt{b}[\,p'@l'\,], p@l \shortmid p'@l')$$

The query $q = \mathsf{cut}_{\mathtt{a}/\Box\mathtt{b}/}(x)$ removes the contents of any branch labelled `b` found after an initial branch `a`, and returns the removed data as the results. For

of this kind are well-known in languages for updating trees, and there is no general agreement on which strategy should be preferred. Our results do not depend on the strategy chosen for our query language.

example,

$$\mathfrak{E}(q, \mathtt{b}[\,V\,]\mid\mathtt{a}[\,\mathtt{c}[\,\mathtt{b}[\,U\,]\,]\,]) = (\mathtt{b}[\,V\,]\mid\mathtt{a}[\,\mathtt{c}[\,\mathtt{b}[\,]\,]\,], U)$$

The query $q = \mathsf{paste}_{\mathtt{a}/*/}\langle \mathtt{e}[\,]\rangle$ adds a branch $\mathtt{e}[\,]$ to any child of $\mathtt{a}$, by reading the the contents of each child and pasting them back with prefixed the new branch $\mathtt{e}[\,]$. For example,

$$\mathfrak{E}(q, \mathtt{a}[\,\mathtt{b}[\,]\mid\mathtt{c}[\,\mathtt{d}[\,]\,]\,]) = (\mathtt{a}[\,\mathtt{b}[\,\mathtt{e}[\,]\,]\mid\mathtt{c}[\,\mathtt{e}[\,]\mid\mathtt{d}[\,]\,]\,], \varnothing\mid(\mathtt{d}[\,]))$$

where the results are the list $\varnothing\mid(\mathtt{d}[\,])$ where the first element is the empty tree (the contents of $\mathtt{b}$) and the second element is tree $\mathtt{d}[\,]$ (the contents of $\mathtt{c}$). Note that the query for pasting data is defined only if each selected node (each child of $\mathtt{a}$) contains a tree[10], since otherwise the resulting tree would be ill-formed. The query $\Box(\pi)\boldsymbol{V}$ where $\pi = \mathtt{b}[\,\langle x\rangle\,]\mid y$ and $\boldsymbol{V} = \mathtt{c}[\,\langle x\rangle\,]\mid y$ relabels each branch $\mathtt{b}$ containing a script to $\mathtt{c}$:

$$\mathfrak{E}(\Box(\pi)\boldsymbol{V}, \mathtt{a}[\,\mathtt{b}[\,\langle A\rangle\,]\mid\mathtt{b}[\,T\,]\mid\mathtt{b}[\,\langle A'\rangle\,]\,]) = (\mathtt{a}[\,\mathtt{c}[\,\langle A\rangle\,]\mid\mathtt{b}[\,T\,]\mid\mathtt{c}[\,\langle A'\rangle\,]\,], L)$$

where the results $L = (\mathtt{b}[\,\langle A\rangle\,]\mid\mathtt{b}[\,T\,]\mid\mathtt{c}[\,\langle A'\rangle\,])\mid(\mathtt{b}[\,\langle A'\rangle\,])$ correspond to the two trees to which the pattern was applied successfully. Note that in the first result, which is the last computed, the last branch has already been relabelled.

On the other hand, our sample query language is not sophisticated enough to express (atomically) a query like "delete each branch labelled $\mathtt{a}$ which contains a branch labelled $\mathtt{b}$", because if we do not know in advance where a branch labelled $\mathtt{b}$ occurs in the contents of $\mathtt{a}$, we cannot write a pattern to select only the nodes containing $\mathtt{b}$. This limitation can be easily overcome by adopting a richer pattern language or by enriching the path expressions with conditions on the contents of nodes.

2.6 Example: Web services

We now describe a simple implementation of macros for defining and calling services in Core X$d\pi$.

Service definition and service call. A service definition is characterized by the invoking location $\boldsymbol{l}$, the name of the service $\boldsymbol{a}$, its input pattern $\widetilde{\pi}$, its body $(x)\boldsymbol{P}$ and its output pattern $\widetilde{\omega}$. The service at location $\boldsymbol{l}$ receives on channel $\boldsymbol{a}$ the input parameters $\widetilde{\pi}$, a location name y and a channel name z (the latter parameters are used to return the result). The body $(x)\boldsymbol{P}$ takes a fresh channel name c (bound to x) in input, performs some arbitrary computation, and outputs the result on channel c. Note that the variables in $\widetilde{\pi}$ may bind in $\boldsymbol{P}$. A forwarding process inputs the result from channel c according to the output pattern $\widetilde{\omega}$, and forwards it to location y on channel z.

(Service Definition) $\quad \boldsymbol{l}\cdot\mathsf{Define}\ \boldsymbol{a}(\widetilde{\pi})\ \mathsf{as}\ (x)\boldsymbol{P}\ \mathsf{output}\ \langle\widetilde{\omega}\rangle$
$\quad\quad \stackrel{\text{def}}{=}\ !\boldsymbol{l}\cdot\boldsymbol{a}(\widetilde{\pi}, y, z).(\nu\, c)((x)\boldsymbol{P} \circ \langle c\rangle \mid \boldsymbol{l}\cdot c(\widetilde{\omega}).\boldsymbol{l}\cdot\overline{y\cdot z}\langle\widetilde{\omega}\rangle)$

[10] A type system regulating the contents of trees could prevent processes from getting stuck because of undefined queries.

The service call is dual. It specifies the location $\boldsymbol{l}$ from which the service is invoked (and to which it is returned), the location $\boldsymbol{m}$ and the name $\boldsymbol{a}$ of the service, its parameters $\widetilde{\boldsymbol{v}}$, and a continuation process $\boldsymbol{Q}$ with patterns $\widetilde{\pi}$ for parsing the results.

(Service Call)
$$\begin{aligned} &\boldsymbol{l}\cdot\mathsf{Call}\ \boldsymbol{m}\cdot\boldsymbol{a}\langle\widetilde{\boldsymbol{v}}\rangle\ \mathsf{return}\ (\widetilde{\pi})\boldsymbol{Q} \\ &\stackrel{\text{def}}{=}\ (\nu\, c)(\boldsymbol{l}\cdot\overline{\boldsymbol{m}\cdot\boldsymbol{a}}\langle\widetilde{\boldsymbol{v}},\boldsymbol{l},c\rangle \mid \boldsymbol{l}\cdot c(\widetilde{\pi}).\boldsymbol{Q}) \end{aligned}$$

The parameters $\boldsymbol{l}$ and c sent on $\boldsymbol{a}$ are used by the forwarding process in the service definition to return the result to the continuation process $\boldsymbol{Q}$. For example, a service providing querying capabilities on its local store, and the corresponding service call, could be defined respectively as

$$m\cdot\mathsf{Define}\ \mathtt{query}(x_1)\ \mathsf{as}\ (x)\mathsf{req}_{x_1}\langle x\rangle\ \mathsf{output}\ \langle x_2\rangle$$

$$l\cdot\mathsf{Call}\ m\cdot\mathtt{query}\langle p\rangle\ \mathsf{return}\ (x)\boldsymbol{Q}$$

The service takes as input a query x_1 and executes the corresponding request on the local store. The forwarding part of the service definition will intercept the request result and send it on c at l, where it is passed on to $\boldsymbol{Q}$ on variable x.

Subscriptions. We can easily generalize service definitions to cover the case of *push services*, which send a stream of results to a client in reply to a single service call. The only difference between the code below and (Service Definition) is the presence of a replicated input in the forwarding process

(Push Service)
$$\begin{aligned} &\boldsymbol{l}\cdot\mathsf{Push}\ \boldsymbol{a}(\widetilde{\pi})\ \mathsf{as}\ (x)\boldsymbol{P}\ \mathsf{output}\ \langle\widetilde{\omega}\rangle \\ &\stackrel{\text{def}}{=}\ !\boldsymbol{l}\cdot\boldsymbol{a}(\widetilde{\pi},y,z).(\nu\, c)((x)\boldsymbol{P}\circ\langle c\rangle \mid !\boldsymbol{l}\cdot c(\widetilde{\omega}).\boldsymbol{l}\cdot\overline{y\cdot z}\langle\widetilde{\omega}\rangle) \end{aligned}$$

The corresponding service subscription waits for multiple results on channel c:

(Subscription)
$$\begin{aligned} &\boldsymbol{l}\cdot\mathsf{Subscribe}\ \boldsymbol{m}\cdot\boldsymbol{a}\langle\widetilde{\boldsymbol{v}}\rangle\ \mathsf{return}\ (\widetilde{\pi})\boldsymbol{Q} \\ &\stackrel{\text{def}}{=}\ (\nu\, c)(\boldsymbol{l}\cdot\overline{\boldsymbol{m}\cdot\boldsymbol{a}}\langle\widetilde{\boldsymbol{v}},\boldsymbol{l},c\rangle \mid !\boldsymbol{l}\cdot c(\widetilde{\pi}).\boldsymbol{Q}) \end{aligned}$$

If desired, the streamed results received by the client can be combined together using a loop.

Result forwarding. In order to have complete control on the return parameters, in certain cases we will bypass the service call code, and use only a migration step followed by a service invocation. For example, let

$$\mathit{Service} = m\cdot\mathsf{Define}\ \mathtt{query}(x_1)\ \mathsf{as}\ (x)\mathsf{req}_{x_1}\langle x\rangle\ \mathsf{output}\ \langle x_2\rangle$$

and consider the network

$$N = (D, (\nu\, c)\begin{pmatrix} l\cdot\overline{m\cdot\mathtt{query}}\langle(w)w,n,c\rangle \\ \mid\ \mathit{Service} \\ \mid\ n\cdot c(x).\mathsf{req}_{(w)x}\langle c\rangle \end{pmatrix})$$

where $D = \{l \mapsto T_0, m \mapsto S, n \mapsto \varnothing\}$. The service invocation migrates to m and triggers the service, passing as return parameters n and c. The local request at m copies the whole tree S and forwards it to c on n:

$$\begin{aligned} N &\longrightarrow^* (D, (\nu\, c)(\mathit{Service} \mid m{\cdot}\overline{n{\cdot}c}\langle S\rangle \mid n{\cdot}c(x).\mathsf{req}_{(w)x}\langle c\rangle) \\ &\longrightarrow^* (\{l \mapsto T_0, m \mapsto S, n \mapsto S\}, \mathit{Service} \mid (\nu\, c)(\overline{n{\cdot}c}\langle\varnothing\rangle) \end{aligned}$$

At n the code listening on c receives the result and replaces the local tree. We will follow this strategy in several examples, redirecting the results of a service to a location different from the one that issued the service call.

3 Behavioural Equivalences

We investigate behavioural equivalences for Core X$d\pi$. First we define network equivalence, which dictates when two networks can be considered indistinguishable with respect to an externally defined comparison. Our network equivalence is parametric with respect to the language used for querying and updating documents.

We also define a process equivalence, which establishes when two processes can replace each other in a network without affecting network equivalence. We would like to reason about the equivalence of groups of processes, possibly interacting across several locations, and obtain results which are robust with respect to changes in the data stored in the local repositories and the behaviour of other parallel processes. Even if we decided to restrict our optimizations to the processes running in a single local peer, we must be ready to reason about partial network specifications in the case where parts of the local process migrate to other locations in order to interact with remote data or services, reinforcing the case for global reasoning. The structure of Core X$d\pi$ networks, where processes are located explicitly and are separated from the data store, facilitates this process. For example, we can express a partial specification of a network by means of located processes running in parallel, possibly sharing private names. Located processes are equivalent if the networks obtained by composing them with arbitrary stores are equivalent. However, process equivalence is hard to use directly because it requires a costly property of closure under contexts. Instead, we use a labelled transition system to define a coinductive equivalence relation (called *domain bisimilarity*), which does not quantify over contexts and which entails process equivalence.

The definition of domain bisimilarity is non-standard, due to the fact that scripts (which can appear in data) are part of the values, and process equivalences are sensitive to the set of locations constituting the network. We address these two problems by adapting existing techniques for translating messages containing scripts into ones where each script is replaced by a first-order value [30, 17], and by generalizing the notion of bisimulation to families of relations indexed by sets of locations.

3.1 Reduction and Domain Congruence

Reduction Congruence. Network equivalence for Core X$d\pi$ is a standard reduction-closed, contextual equivalence which preserves some observation predicates. In [18] we consider different choices of observation predicates (in particular the shape of the data tree of a location and the presence of output actions in a process), and we study the formal relationship between the corresponding reduction congruences. Here, we focus on request observations because it seems natural to observe the effect processes have on data. In addition, the resulting reduction congruence coincides with the one resulting from output observation and implies the one resulting from tree-shape observations.

Definition 3.1 (Request observation predicate) *We define the* request observation predicate $\downarrow_{l\cdot p}$ *as* $(D, P) \downarrow_{l\cdot p} \iff \exists C, c, Q.\ P \equiv C[l\cdot\mathsf{req}_p\langle c\rangle \mid Q]$ *and the* weak *observation predicate* $\Downarrow_{l\cdot p}$ *as* $N \Downarrow_{l\cdot p} \iff \exists N'.N \longrightarrow^* N'$ *and* $N' \downarrow_{l\cdot p}$.

Definition 3.2 (Reduction Congruence) Reduction congruence $\simeq$ *on Core Xdπ networks is the largest symmetric relation* $\dot{\simeq}$ *which is*

- *observation preserving:* $N \dot{\simeq} M \Longrightarrow \forall l, p.N \Downarrow_{l\cdot p} \Longrightarrow M \Downarrow_{l\cdot p}$
- *reduction closed:* $N \dot{\simeq} M \Longrightarrow \forall N'.N \longrightarrow^* N' \Longrightarrow \exists M'.M \longrightarrow^* M'$ *and* $N' \dot{\simeq} M'$
- *contextual:* $N \dot{\simeq} M \Longrightarrow \forall C.C[N] \dot{\simeq} C[M]$.

For example, we have

$$(\{l \mapsto T\}, \overline{l\cdot a}\langle c\rangle) \not\simeq (\{l \mapsto T\}, \overline{l\cdot a}\langle b\rangle)$$

because the context $K = (-, - \mid l\cdot a(x).\overline{l\cdot x} \mid l\cdot c.l\cdot\mathsf{req}_p\langle a\rangle)$ can tell a difference between the two processes.

In order to use equational reasoning, it is important to remark that reduction congruence is an equivalence relation (the proof is completely standard).

Observation 3.3 (Equivalence) *Reduction congruence* $\simeq$ *is an equivalence relation.*

Since reduction congruence is based on contexts which do not inhibit reduction, a simple test for equivalence consists in checking if two terms can reduce to each other.

Lemma 3.4 (Mutual Reduction) *If* $N \longrightarrow^* M$ *and* $M \longrightarrow^* N$ *then* $N \simeq M$.

Proof. We show that the relation

$$\dot{\simeq} = \left\{(C[N], C[M]) \ : \ N \longrightarrow^* M,\ M \longrightarrow^* N\right\}$$

is contained in $\simeq$. The proof is symmetric. Consider an arbitrary pair $(N, M) \in \dot{\simeq}$. Suppose $N \Downarrow_{l\cdot p}$. It must be the case that $N \longrightarrow^* N' \downarrow_{l\cdot p}$. By hypothesis

$M \longrightarrow^* N$, hence $M \Downarrow_{l \cdot p}$. Suppose $N \longrightarrow^* N'$. By hypothesis $M \longrightarrow^* N$, hence $M \longrightarrow^* N'$. By definition of $\longrightarrow^*$, $N' \dot{\simeq} N'$. We need to show that for an arbitrary $C[-]$, $C[N] \dot{\simeq} C[M]$. Since $C[-]$ is a reduction context, $N \longrightarrow^* M \Longrightarrow C[N] \longrightarrow^* C[M]$ and $M \longrightarrow^* N \Longrightarrow C[M] \longrightarrow^* C[N]$. By definition, $C[N] \dot{\simeq} C[M]$. □

Domain Congruence. We now define process equivalence for Core X$d\pi$. This equivalence depends on the locations present in the network, but not on the actual contents of the stores. Consider replacing the definition of a service at location l, which uses only local data, with one located at m (where there is a cached copy of the same data) and providing an equivalent functionality. If location m is connected, then the behaviour of the services is the same. On the other hand, if location m is not connected, the behaviour of the services is different. With network equivalence, the connected locations are those in the domain of the store. With process equivalence, we must state explicitly the locations which we assume to be part of the network. As a consequence, process equivalence is indexed by a *domain* (a set of locations) Λ.

A Core X$d\pi$ process can be seen as a partial specification of a network, describing only some of the processes running in some of the locations. This point of view is useful for reasoning about replacing components which are part of some distributed data-exchange protocol. Accordingly, we say that two processes P and Q are equivalent with respect to a domain Λ if all the networks containing *at least* the locations in Λ and either P or Q, are equivalent.

Besides comparing partial network specifications, process equivalences can be useful for example to replace optimized pieces of code inside a specific process. For that purpose, we need a more general class of process contexts which include prefixes.

Definition 3.5 (Extended Contexts) Extended process contexts $\mathcal{K}_{\mathrm{f}}$ *are the terms generated by*

$$C ::= \ - \ \mid C \mid \boldsymbol{P} \mid \boldsymbol{P} \mid C \mid (\nu\, c)C \mid \boldsymbol{l}\cdot\boldsymbol{a}(\widetilde{\pi}).C \mid !\boldsymbol{l}\cdot\boldsymbol{a}(\widetilde{\pi}).C \mid \boldsymbol{l}\cdot\mathsf{go}\,\boldsymbol{m}.C$$

Unless we specify otherwise, from now on we use $C[-]$ to denote extended contexts.

Definition 3.6 (Domain Congruence) *Given a set of location names Λ, we define the induced* domain congruence $\sim^{\Lambda}$ *on processes by*

$$\sim^{\Lambda} = \{(\boldsymbol{P}, \boldsymbol{Q}) \ : \ \forall D, C[-].\ \Lambda \subseteq \mathit{dom}(D) \Longrightarrow (D, C[\boldsymbol{P}]) \simeq (D, C[\boldsymbol{Q}])\}$$

where each $C[-]$ is closing for both $\boldsymbol{P}$ and $\boldsymbol{Q}$.

Domain congruence is monotonic: the larger the set of locations which we assume to be part of the network, the larger the number of processes which we can equate.

Observation 3.7 (Monotonicity) *If $\Lambda \subseteq \Lambda'$ then $\sim^{\Lambda} \subseteq \sim^{\Lambda'}$.*

Figure 10: Notation for asynchronous processes

(Forwarder) $\boldsymbol{l}\cdot\mathsf{FW}(\boldsymbol{a},\boldsymbol{b},\widetilde{\pi}) \stackrel{\text{def}}{=} \boldsymbol{l}\cdot\boldsymbol{a}(\widetilde{\pi}).\overline{\boldsymbol{l}\cdot\boldsymbol{b}}\langle\widetilde{\pi}\rangle$

(Equator) $\boldsymbol{l}\cdot\mathsf{EQ}(\boldsymbol{a},\boldsymbol{b},\widetilde{\pi}) \stackrel{\text{def}}{=} !\boldsymbol{l}\cdot\mathsf{FW}(\boldsymbol{a},\boldsymbol{b},\widetilde{\pi}) \mid !\boldsymbol{l}\cdot\mathsf{FW}(\boldsymbol{b},\boldsymbol{a},\widetilde{\pi})$

(Distributed Forwarder) $\boldsymbol{l}\cdot\mathsf{dFW}(\boldsymbol{a},\boldsymbol{m},\boldsymbol{b},\widetilde{\pi}) \stackrel{\text{def}}{=} \boldsymbol{l}\cdot\boldsymbol{a}(\widetilde{\pi}).\boldsymbol{l}\cdot\overline{\boldsymbol{m}\cdot\boldsymbol{b}}\langle\widetilde{\pi}\rangle$

(Distributed Equator) $\mathsf{dEQ}(\boldsymbol{l},\boldsymbol{a},\boldsymbol{m},\boldsymbol{b},\widetilde{\pi}) \stackrel{\text{def}}{=} !\boldsymbol{l}\cdot\mathsf{dFW}(\boldsymbol{a},\boldsymbol{m},\boldsymbol{b},\widetilde{\pi}) \mid !\boldsymbol{m}\cdot\mathsf{dFW}(\boldsymbol{b},\boldsymbol{l},\boldsymbol{a},\widetilde{\pi})$

Proof. Follows easily by Definition 3.6. □

Due to the several explicit (and implicit) universal quantifications involved in Definition 3.6, it is very difficult to show directly that two processes are domain congruent. For this reason, in Section 3.2 we will introduce a proof method which does not require closure under contexts and which entails domain congruence.

Asynchronous Laws. Core X$d\pi$ is an extension of the asynchronous π-calculus, so we consider some equational laws inspired by the latter. Consider the process definitions given in Figure 10. The *asynchrony law*, stating that the presence of a communication buffer cannot be observed, holds also in Core X$d\pi$(see Section 3.2.2 for a proof):

$$!\boldsymbol{l}\cdot\mathsf{FW}(\boldsymbol{a},\boldsymbol{a},\widetilde{\pi}) \sim_r^{\Lambda} \mathbf{0}$$

The law stating that two channels $\boldsymbol{a}$ and $\boldsymbol{b}$ cannot be distinguished if they are part of the same equator does not hold. For example,

$$\boldsymbol{l}\cdot\mathsf{EQ}(\boldsymbol{a},\boldsymbol{b},\widetilde{\pi}) \mid \overline{\boldsymbol{l}\cdot\boldsymbol{c}}\langle\boldsymbol{a}\rangle \not\sim_r^{\Lambda} \boldsymbol{l}\cdot\mathsf{EQ}(\boldsymbol{a},\boldsymbol{b},\widetilde{\pi}) \mid \overline{\boldsymbol{l}\cdot\boldsymbol{c}}\langle\boldsymbol{b}\rangle$$

because a context could intercept the channel name $\boldsymbol{a}$ and use it in some fresh location m where $\boldsymbol{a}$ and $\boldsymbol{b}$ are not equated. We have instead a new law about equating located channels across different locations:

$$\mathsf{dEQ}(l,\boldsymbol{a},m,\boldsymbol{b},\widetilde{\pi}) \mid \overline{l\cdot\boldsymbol{a}}\langle\widetilde{\pi}\sigma\rangle \sim_r^{\{l,m\}} \mathsf{dEQ}(l,\boldsymbol{a},m,\boldsymbol{b},\widetilde{\pi}) \mid \overline{m\cdot\boldsymbol{b}}\langle\widetilde{\pi}\sigma\rangle$$

This law could be useful to show that we can replicate Web services (improving efficiency) without the clients needing to be aware of the change.

3.2 Bisimilarity

In this Section, we define a coinductive equivalence relation (*bisimilarity*), which does not quantify over contexts and which entails domain congruence.

3.2.1 Labelled transition system

A typical proof that processes are bisimilar involves a universal quantification over labelled transitions. Since Core X$d\pi$ values include scripts, and labels

Figure 11: Syntax: configurations

$\boldsymbol{K}$ ::=		configuration terms
	$\boldsymbol{P}$	process terms (built as for Core X$d\pi$)
	$\boldsymbol{K} \mid \boldsymbol{K}$	parallel composition
	$(\nu\, c)\boldsymbol{K}$	restriction
	$\langle k \Leftarrow A \rangle$	definition for trigger name k with script A
$\boldsymbol{A}$ ::=	$A \mid x \mid k$	script or variable or trigger name

$h, i, j, k \in \mathcal{Y}$ $\quad (\mathcal{Y} \cap \mathcal{C}_{\mathrm{p}} = \emptyset,\ \mathcal{Y}$ countably infinite) (Trigger Names)

$K, L \in \mathcal{W} \stackrel{\text{def}}{=} \{\boldsymbol{K} \ : \ \mathit{fv}(\boldsymbol{K}) = \emptyset,\ \mathit{unique}(\boldsymbol{K})\}$ (Configurations)

$A \in \mathcal{A}_{\mathcal{W}} \stackrel{\text{def}}{=} \{A \ : \ A \in \mathcal{A},\ \mathit{triggers}(A) = \emptyset\}$ (Scripts)

$C_{\mathcal{W}}[-]$ ::= $- \mid C_{\mathcal{W}}[-] \mid K \mid K \mid C_{\mathcal{W}}[-] \mid (\nu\, c) C_{\mathcal{W}}[-]$ (Configuration Contexts)

The function *triggers* and the predicate *unique* are defined in Figure 12.
Apart from the redefinition of $\boldsymbol{A}$, A and p (see below), the grammars for trees, processes and values are the same as for Core X$d\pi$.

For trigger definitions, we adopt the following notation:

$\Theta^{\widetilde{k}} \stackrel{\text{def}}{=} \langle k_1 \Leftarrow A_1 \rangle \mid \ldots \mid \langle k_n \Leftarrow A_n \rangle$, where all k_i are distinct, $n \geq 0$;
$\Theta^{\widetilde{k} \curvearrowleft \widetilde{j}} \stackrel{\text{def}}{=} \Theta^{\widetilde{k}}\{{}^{\widetilde{j}}/_{\widetilde{k}}\}$, when $\{{}^{\widetilde{j}}/_{\widetilde{k}}\}$ is defined;
$\Theta \stackrel{\text{def}}{=} \Theta^{\widetilde{k}}$, when $\widetilde{k}$ is not important.
$t^{\langle k \Leftarrow A \rangle} \stackrel{\text{def}}{=} t\{{}^{A}/_{k}\}$, for any term t, and similarly for $t^{\Theta^{\widetilde{k}}}$.

typically include values, we risk falling back to quantifying over processes. Following the approach of [30, 17], we avoid this problem by translating messages containing scripts into ones where each script is replaced by a *trigger name* (a first-order value), and by placing in parallel to the process being analyzed some *definitions* associating to each trigger name the code of the corresponding script. By including these definitions in the code, we are able to analyze also the interaction between scripts and their contexts.

Configurations. We introduce *configurations*, which are processes extended with the trigger names and definitions mentioned above. The formal syntax is given in Figure 11. Note that $\boldsymbol{A}$ denotes now a script, a variable or a trigger name, hence processes can syntactically contain triggers. Nonetheless, scripts and queries are not allowed to contain triggers. In fact, trigger names and definitions are merely intermediate terms arising during the analysis of the transition of a process, and are not meant to be part of the user syntax. For a configuration K to be well-formed, no two definitions in K can have the same trigger name (predicate $\mathit{unique}(K)$, defined in Figure 12). As a convention, we

Figure 12: Function *triggers* and predicate *unique*.

$$\mathit{triggers}(t) = \mathit{fn}(t) \cap \mathcal{Y}$$

$$\frac{\mathit{unique}(\boldsymbol{K}) \quad \mathit{unique}(\boldsymbol{K}') \quad \mathit{dfs}(\boldsymbol{K}) \cap \mathit{dfs}(\boldsymbol{K}') = \emptyset}{\mathit{unique}(\boldsymbol{K} \mid \boldsymbol{K}')} \qquad \frac{\mathit{unique}(\boldsymbol{K})}{\mathit{unique}((\nu\, c)\boldsymbol{K})} \qquad \mathit{unique}(\langle k \Leftarrow A \rangle)$$

The function *dfs*, returning the triggers defined by a configuration, is given by

$$\mathit{dfs}(\boldsymbol{K} \mid \boldsymbol{K}') = \mathit{dfs}(\boldsymbol{K}) \cup \mathit{dfs}(\boldsymbol{K}') \quad \mathit{dfs}((\nu\, c)\boldsymbol{K}) = \mathit{dfs}(\boldsymbol{K}) \quad \mathit{dfs}(\langle k \Leftarrow A \rangle) = \{k\}$$

let Θ, Ω and Φ range on groups of definitions. Note also that two groups of definitions $\Theta^{\widetilde{k}}$ and $\Theta^{\widetilde{j}}$ identified by the same name but by different vectors of triggers can in principle be arbitrarily different: it is an important syntactic convention which helps to simplify the notation, and is often used in the rest of the section. In the Appendix, in Figure 21 and Figure 23, we extend $\equiv$, *fv* and *fn* to configurations. In Figure 24 we extend the function *dom* of Figure 22 to configurations, and we define a function *scripts* returning the scripts present in a piece of data.

Queries. Queries used for updating can mention constant data, which may contain scripts. We assume two functions, *scripts* and *triggers*, which given a query return respectively the set of scripts and triggers it contains. The only condition that we need to impose on query evaluation consists of it not being dependent on the particular structure of scripts. In other words, if we replace a script in a query with a trigger name, then the result of the query should be equivalent up to substitution of the script for the trigger. Moreover, any script returned by the query must occur in the input tree or in the query itself. The condition is formalized below.

Definition 3.8 (Script Independence) *Let $\mathfrak{L} = (\mathcal{Q}, \mathit{fv}, \mathfrak{E})$ be an arbitrary query language, let p, T be such that $\mathfrak{E}(p, T) = (S, L)$, and let p_0, T_0 be their first-order versions, such that $\mathit{scripts}(p_0) = \mathit{scripts}(T_0) = \emptyset$ and $p = p_0^{\Theta^{\widetilde{j}}}, T = T_0^{\Omega^{\widetilde{k}}}$ for some $\Theta^{\widetilde{j}}, \Omega^{\widetilde{k}}$.*

The query language $\mathfrak{L}$ is script independent *if for all $\Theta^{\widetilde{j}}, \Omega^{\widetilde{k}}$ there exist $\Theta^{\widetilde{h}}$ and $\Omega^{\widetilde{i}}$ such that*

- *query evaluation does not depend on the structure of scripts: there are S_0, L_0 with $\mathit{scripts}(S_0) = \mathit{scripts}(L_0) = \emptyset$ such that $\mathfrak{E}(p_0^{\Theta^{\widetilde{j}}}, T_0^{\Omega^{\widetilde{k}}}) = (S_0^{\Theta^{\widetilde{h}}}, L_0^{\Omega^{\widetilde{i}}})$*
- *no new scripts are introduced: for any definition $\langle k \Leftarrow A \rangle$ occurring in $\Theta^{\widetilde{h}}$ or $\Omega^{\widetilde{i}}$ there must be a definition $\langle k' \Leftarrow A \rangle$ occurring in $\Theta^{\widetilde{j}}$ or $\Omega^{\widetilde{k}}$.*

Figure 13: Extraction relation

$$\mathfrak{X}(l) = (l; \mathbf{0}) \qquad \frac{\mathfrak{X}(\boldsymbol{p}) = (\boldsymbol{p}'; \Theta)}{\mathfrak{X}(\boldsymbol{p}@\boldsymbol{l}) = (\boldsymbol{p}'@\boldsymbol{l}; \Theta)}$$

$$\mathfrak{X}(p) = \mathfrak{X}_{\mathcal{Q}}(p) \qquad \mathfrak{X}(\varnothing) = (\varnothing; \mathbf{0})$$

$$\mathfrak{X}(A) = (k; \langle k \Leftarrow A \rangle) \qquad \mathfrak{X}(x) = (x; \mathbf{0})$$

$$\mathfrak{X}(c) = (c; \mathbf{0}) \qquad \mathfrak{X}(k) = (k; \mathbf{0})$$

$$\frac{\mathfrak{X}(\boldsymbol{E}) = (\boldsymbol{E}'; \Theta^{\widetilde{k}}) \qquad \mathfrak{X}(\boldsymbol{T}) = (\boldsymbol{T}'; \Omega^{\widetilde{h}}) \qquad (\{\widetilde{k}, \widetilde{h}\} \cap \mathit{fn}(\boldsymbol{E} \shortmid \boldsymbol{T})) \cup (\{\widetilde{k}\} \cap \{\widetilde{h}\}) = \emptyset}{\mathfrak{X}(\boldsymbol{E} \shortmid \boldsymbol{T}) = (\boldsymbol{E}' \shortmid \boldsymbol{T}'; \Theta^{\widetilde{k}} \,|\, \Omega^{\widetilde{h}})} \qquad \frac{\mathfrak{X}(\boldsymbol{A}) = (\boldsymbol{v}; \Theta)}{\mathfrak{X}(\langle \boldsymbol{A} \rangle) = (\langle \boldsymbol{v} \rangle; \Theta)}$$

$$\frac{\mathfrak{X}(\boldsymbol{v}) = (\boldsymbol{v}'; \Theta^{\widetilde{k}}) \qquad \mathfrak{X}(\widetilde{\boldsymbol{v}}) = (\widetilde{\boldsymbol{v}'}; \Omega^{\widetilde{h}}) \qquad (\{\widetilde{k}, \widetilde{h}\} \cap \mathit{fn}(\boldsymbol{v}, \widetilde{\boldsymbol{v}})) \cup (\{\widetilde{k}\} \cap \{\widetilde{h}\}) = \emptyset}{\mathfrak{X}(\boldsymbol{v}, \widetilde{\boldsymbol{v}}) = (\boldsymbol{v}', \widetilde{\boldsymbol{v}'}; \Theta^{\widetilde{k}} \,|\, \Omega^{\widetilde{h}})} \qquad \frac{\mathfrak{X}(\boldsymbol{V}) = (\boldsymbol{V}'; \Theta)}{\mathfrak{X}(\mathtt{a}[\,\boldsymbol{V}\,]) = (\mathtt{a}[\,\boldsymbol{V}'\,]; \Theta)}$$

$$\frac{\mathfrak{X}(T) = (T'; \Theta^{\widetilde{k}}) \qquad \mathfrak{X}(D) = (D'; \Omega^{\widetilde{h}}) \qquad (\{\widetilde{k}, \widetilde{h}\} \cap \mathit{fn}(T, D)) \cup (\{\widetilde{k}\} \cap \{\widetilde{h}\}) = \emptyset}{\mathfrak{X}(\{l \mapsto T\} \uplus D) = (\{l \mapsto T'\} \uplus D'; \Theta^{\widetilde{k}} \,|\, \Omega^{\widetilde{h}})} \qquad \mathfrak{X}(\emptyset) = (\emptyset; \mathbf{0})$$

Notation: in this table c ranges over $\mathcal{C}_{\mathrm{p}} \cup \mathcal{C}_{\mathrm{s}}$.
The relation $\mathfrak{X}_{\mathcal{Q}}$ is specified in Definition 3.9.

Extracting scripts from values. Our strategy consists of translating values containing scripts into values containing trigger names only, extracting at the same time the corresponding definitions. For that purpose, we define in Figure 13 an extraction relation $\mathfrak{X}$ which applies to Core X$d\pi$ data and stores, and returns the corresponding first-order terms and the definitions extracted.

The definition of $\mathfrak{X}$ is straightforward. The only points worth noting are that the premises of the rules for tuples, tree and store composition make sure that the trigger names remain disjoint, and that the rule for scripts replaces a script with a trigger and records the corresponding definition. The rule for queries invokes a specialized extraction relation $\mathfrak{X}_{\mathcal{Q}}$ which depends on the query language. $\mathfrak{X}_{\mathcal{Q}}$ can behave similarly to $\mathfrak{X}$, relating each query with its first-order version and the corresponding definitions, or can behave differently (for example being the identity function on queries and the constant 0 on configurations), as long as it satisfies the basic properties requested by the definition given below.

Definition 3.9 (Query Extraction) *The relation $\mathfrak{X}_{\mathcal{Q}}$ can be any subset of $\mathcal{Q}_{\mathrm{C}} \times \mathcal{Q}_{\mathrm{C}} \times \mathcal{W}$ satisfying the condition that if $\mathfrak{X}_{\mathcal{Q}}(p) = (p'; K)$ then*

1. *K are well-formed definitions: $K = \Theta^{\widetilde{k}}$;*

Figure 14: Labels for the transition system

α_l **::=**		transition labels
	$(\widetilde{a},\widetilde{k})\overline{l{\cdot}c}\langle\widetilde{v}\rangle$	output of $\widetilde{v}$ on c at l, extruding $\widetilde{a},\widetilde{k}$
	$l{\cdot}c(\widetilde{v})$	input of $\widetilde{v}$ on c at l
	$l{\cdot}\tau$	internal reduction at l
	$(\widetilde{k})l{\cdot}\mathsf{req}\langle p\rangle(T)$	request p at l extruding $\widetilde{k}$, obtaining result T
	$l{\cdot}k(\widetilde{v})$	run the script defined by k with parameters $l,\widetilde{v}$
	$(\widetilde{a},\widetilde{k})\overline{l{\cdot}j}\langle\widetilde{v}\rangle$	assume running the script defined by trigger j with parameters $l,\widetilde{v}$, extruding $\widetilde{a},\widetilde{k}$

Convention: in the syntax above, c ranges over $\mathcal{C}_\mathrm{p}\cup\mathcal{C}_\mathrm{s}$, and t over $\mathcal{C}_\mathrm{p}\cup\mathcal{C}_\mathrm{s}\cup\mathcal{Y}$.

labels α_l	names n	bound names bn
$(\widetilde{a},\widetilde{k})\overline{l{\cdot}t}\langle\widetilde{v}\rangle$	$\{\widetilde{a}\}\cup\{\widetilde{k}\}\cup\{t\}\cup\mathit{fn}(\widetilde{v})$	$\{\widetilde{a}\}\cup\{\widetilde{k}\}$
$l{\cdot}t(\widetilde{v})$	$\{t\}\cup\mathit{fn}(\widetilde{v})$	$\emptyset$
$l{\cdot}\tau$	$\emptyset$	$\emptyset$
$(\widetilde{k})l{\cdot}\mathsf{req}\langle p\rangle(T)$	$\{\widetilde{k}\}\cup\mathit{triggers}(p)\cup\mathit{fn}(T)$	$\{\widetilde{k}\}$

$$\mathit{fn}(\alpha_l)=\mathit{n}(\alpha_l)\setminus\mathit{bn}(\alpha_l)$$

2. *trigger names can be extended as long as there are no clashes:* $\mathit{triggers}(p')=\mathit{triggers}(p)\cup\{\widetilde{k}\}$ *and* $\mathit{triggers}(p)\cap\{\widetilde{k}\}=\emptyset$*;*
3. *the new trigger names are defined up-to renaming: for all* $\widetilde{j}$ *distinct from* $\mathit{triggers}(p)$, $\mathfrak{X}_\mathcal{Q}(p)=(p'\{\widetilde{j}/\widetilde{k}\};\Theta^{\widetilde{k}\curvearrowleft\widetilde{j}})$*;*
4. *substitution is the inverse of extraction: if* $\mathfrak{X}_\mathcal{Q}(p)=(p';\Theta)$ *then* $p=p'^{\Theta}$*.*

Under the assumption (that we adopt henceforth) that $\mathfrak{X}_\mathcal{Q}$ respects Definition 3.9, the effects of relation $\mathfrak{X}$ can be reversed by replacing, in the extracted first-order term, the new trigger names by the corresponding definitions.

Observation 3.10 (Extraction) *For any given term* t*, if* $\mathfrak{X}(t)=(t';\Theta)$ *then* $t=t'^{\Theta}$*.*

Proof. By induction on the derivation of $\mathfrak{X}(t)=(t';\Theta)$. □

Labelled transition system. Our labelled transition system incorporates ideas on asynchronous transitions from [15], and on translating higher-order actions into first order actions from [30, 17].

Figure 15: Labelled transition system

(Lts Com)
$$\overline{\overline{l{\cdot}c}\langle\widetilde{\pi}\sigma\rangle \mid l{\cdot}c(\widetilde{\pi}).\boldsymbol{P} \xrightarrow{l\cdot\tau} \boldsymbol{P}\sigma}$$

(Lts !Com)
$$\overline{\overline{l{\cdot}c}\langle\widetilde{\pi}\sigma\rangle \mid !l{\cdot}c(\widetilde{\pi}).\boldsymbol{P} \xrightarrow{l\cdot\tau} \boldsymbol{P}\sigma \mid !l{\cdot}c(\widetilde{\pi}).\boldsymbol{P}}$$

(Lts Run)
$$\frac{}{(x,\widetilde{\pi})\boldsymbol{P} \circ \langle l, \widetilde{\pi}\sigma\rangle \xrightarrow{l\cdot\tau} \boldsymbol{P}\{{}^{l}/_{x}\}\sigma}$$

(Lts In)
$$\frac{\mathit{scripts}(\widetilde{v}) = \emptyset}{0 \xrightarrow{l\cdot c(\widetilde{v})} \overline{l{\cdot}c}\langle\widetilde{v}\rangle}$$

(Lts Out)
$$\frac{\mathfrak{X}(\widetilde{v}) = (\widetilde{v'}; \Theta^{\widetilde{k}})}{\overline{l{\cdot}c}\langle\widetilde{v}\rangle \xrightarrow{(\widetilde{k})\overline{l\cdot c}\langle\widetilde{v'}\rangle} \Theta^{\widetilde{k}}}$$

(Lts Trigger)
$$\frac{\mathfrak{X}(\widetilde{v}) = (\widetilde{v'}; \Theta^{\widetilde{k}}) \quad j \notin \{\widetilde{k}\}}{j \circ \langle l, \widetilde{v}\rangle \xrightarrow{(\widetilde{k})\overline{l\cdot j}\langle\widetilde{v'}\rangle} \Theta^{\widetilde{k}}}$$

(Lts Open)
$$\frac{a \in \mathit{fn}(\widetilde{v}) \setminus \{\widetilde{a}, c\} \qquad K \xrightarrow{(\widetilde{a},\widetilde{k})\overline{l\cdot c}\langle\widetilde{v}\rangle} K'}{(\nu\, a)K \xrightarrow{(a,\widetilde{a},\widetilde{k})\overline{l\cdot c}\langle\widetilde{v}\rangle} K'}$$

(Lts Req)
$$\frac{\mathfrak{X}_{\mathcal{Q}}(p) = (p', \Theta^{\widetilde{k}}) \qquad \mathit{scripts}(T) = \emptyset \quad T = \mathtt{r}[\,U_1\,]\mid \ldots \mathtt{r}[\,U_n\,]\mid\varnothing}{l{\cdot}\mathsf{req}_p\langle c\rangle \xrightarrow{(\widetilde{k})l\cdot\mathsf{req}\langle p'\rangle(T)} \overline{l{\cdot}c}\langle T\rangle \mid \Theta^{\widetilde{k}}}$$

(Lts Def)
$$\frac{\mathit{scripts}(\sigma) = \emptyset}{\langle k \Leftarrow (x,\widetilde{\pi})\boldsymbol{P}\rangle \xrightarrow{l\cdot k(\widetilde{\pi}\sigma)} \langle k \Leftarrow (x,\widetilde{\pi})\boldsymbol{P}\rangle \mid \boldsymbol{P}\{{}^{l}/_{x}\}\sigma}$$

(Lts Go)
$$\frac{}{l{\cdot}\mathsf{go}\, m.P \xrightarrow{m\cdot\tau} P}$$

(Lts Res)
$$\frac{K \xrightarrow{\alpha_l} K' \quad a \notin n(\alpha_l)}{(\nu\, a)K \xrightarrow{\alpha_l} (\nu\, a)K'}$$

(Lts Par)
$$\frac{K \xrightarrow{\alpha_l} K' \quad \mathit{rel}(\alpha_l, L)}{K \mid L \xrightarrow{\alpha_l} K' \mid L}$$

(Lts Struct)
$$\frac{K \equiv L \xrightarrow{\alpha_l} L' \equiv K'}{K \xrightarrow{\alpha_l} K'}$$

Notation: $\mathit{rel}(\alpha_l, K) \stackrel{\text{def}}{=} \mathit{bn}(\alpha_l) \cap \mathit{fn}(K) = \emptyset$.
Convention: in this table c ranges over $\mathcal{C}_{\mathrm{p}} \cup \mathcal{C}_{\mathrm{s}}$, and t over $\mathcal{C}_{\mathrm{p}} \cup \mathcal{C}_{\mathrm{s}} \cup \mathcal{Y}$.

The labels of the transition system record what kind of interaction with the external environment is necessary for a configuration to evolve into another. Labels, along with the notions of their names, free names and bound names, are defined in Figure 14. Each label, including the one for internal reduction, shows explicitly the location where interaction takes place. By using appropriate conditions on the function *scripts* in the rules of the labelled transition system (*lts* for short), we will guarantee that labels are first-order, as planned. The formal definition of the lts is given in Figure 15. We discuss the more interesting transition rules. Rules (Lts Com), (Lts !Com) and (Lts Run) closely mimic the corresponding reduction rules. These transitions do not require interaction with the external environment, so the label $l{\cdot}\tau$ requires only the existence of loca-

tion l. Rule (Lts In) provides a first-order output message from the environment which can be used to analyze the continuation of an input process by deriving a further transition using the communication rules. Rule (Lts Out) states that a potentially higher-order output $\overline{l}{\cdot}c\langle\widetilde{v}\rangle$ evolves to the definitions that are extracted from $\widetilde{v}$ to obtain $\widetilde{v'}$, and carries in the label the first-order version of the process. The intuition is that a bisimilar process will be required to perform the same first-order transition, and a potential incompatibility between the original higher-order messages will be detected by analyzing the resulting configurations ($\Theta^{\widetilde{k}}$ for the first process). Rule (Lts Trigger) states that the application of a trigger name to the potentially higher-order parameters $\widetilde{v}$ evolves to the definitions that are extracted from $\widetilde{v}$ to obtain $\widetilde{v'}$, and carries in the label the first-order version of the process, similarly to the case for output. Rule (Lts Open) is standard. Not that it applies to transitions originated using (Lts Out) or (Lts Trigger). Rule (Lts Req) can be interpreted as the combination of an output of p and an input of T on a special name req. Rule (Lts Def) analyzes the script of a definition for all its possible (first-order) input parameters.[11] It is akin to performing an asynchronous input transition to receive the parameters for the script from the context, and a communication step to instantiate the script.

The sample query and update language. We conclude this section by extending Sam (Definition 2.2) to deal with trigger names, and showing that it respects our assumptions (Observation 3.12).

Definition 3.11 ($\mathsf{Sam}^{\#}$) *The query language $\mathsf{Sam}^{\#}$ is defined as Sam, with the exception that trees can contain also trigger names in the same position as scripts (i.e. within $\langle - \rangle$). The extraction relation $\mathfrak{X}_{\mathcal{Q}}$, and the functions scripts and triggers are defined on $\mathsf{Sam}^{\#}$ by*

$$\frac{\mathfrak{X}(\boldsymbol{V}) = (\boldsymbol{U};\Theta)}{\mathfrak{X}_{\mathcal{Q}}(\widehat{p}(\pi)\boldsymbol{V}) = (\widehat{p}(\pi)\boldsymbol{U};\Theta)}$$

$$\mathit{scripts}(\widehat{p}(\pi)\boldsymbol{V}) = \mathit{scripts}(\boldsymbol{V}) \qquad \mathit{triggers}(\widehat{p}(\pi)\boldsymbol{V}) = \mathit{triggers}(\boldsymbol{V})$$

Observation 3.12 (Properties of $\mathsf{Sam}^{\#}$) *(i) $\mathsf{Sam}^{\#}$ is script independent, and (ii) $\mathfrak{X}_{\mathcal{Q}}$ for $\mathsf{Sam}^{\#}$ respects Definition 3.9.*

Proof. Point (i) follows by induction on the derivation of $\mathfrak{E}$. The idea is that query evaluation does not depend on the structure of scripts, and by rule (Eval Match) only scripts coming from the query and the input tree can occur in the result and the output tree. Point (ii) follows by induction on the derivation of $\mathfrak{X}_{\mathcal{Q}}$. □

[11] Both in (Lts Def) and (Lts In) we do not need to consider higher-order values. This is due to the fact that the bisimilarity relation that we will consider turns out to be closed with respect to parallel composition with definitions (Theorem B.15), and hence already takes into account the effects of scripts received from the environment.

3.2.2 Domain bisimilarity

We introduce our bisimulation equivalence. The intuition is that, when two bisimilar processes are running in a location domain Λ, if a process makes an action α_l with $l \in \Lambda$ then the other one must be able to mimic it, possibly relying on the existence of other locations in Λ. Since the location domain can be extended to $\Lambda \cup \Lambda'$ by composing networks, we need to make sure that also the actions mentioning locations in Λ' are matched, this time within a larger relation parameterized by $\Lambda \cup \Lambda'$.

The definition of bisimilarity relies on the following derived transition relations.

Definition 3.13 (Derived Transition Relations) *Consider the lts defined in Figure 15. Given $l \in \Lambda$, we use the notation*

$$\xrightarrow{\tau}_\Lambda \stackrel{\text{def}}{=} \xrightarrow{l\cdot\tau}; \quad \xrightarrow{l\cdot\tau}\!\!\!\!\rightarrow_\Lambda \stackrel{\text{def}}{=} \xrightarrow{\tau}\!{}^*_\Lambda; \quad \xrightarrow{\alpha_l}\!\!\!\!\rightarrow_\Lambda \stackrel{\text{def}}{=} \xrightarrow{\tau}\!{}^*_\Lambda \circ \xrightarrow{\alpha_l} \circ \xrightarrow{\tau}\!{}^*_\Lambda \quad \textit{when } \alpha_l \neq l\cdot\tau.$$

Definition 3.14 (Domain Bisimilarity) *A family of symmetric relations on configurations (indexed with sets of locations) $\dot{\approx} = \{\dot{\approx}_\Lambda \ : \ \Lambda \subseteq \mathcal{L}\}$ is a* domain bisimulation *if $K \dot{\approx}_\Lambda L$ and $K \xrightarrow{\alpha_l} K'$ implies:*

1. *if $l \in \Lambda$ with $\mathit{rel}(\alpha_l, L)$ then $L \stackrel{\alpha_l}{\twoheadrightarrow}_\Lambda L'$ and $K' \dot{\approx}_\Lambda L'$;*
2. *if $l \notin \Lambda$ then $K \dot{\approx}_{\Lambda\cup\{l\}} L$.*

Domain bisimilarity $\approx = \{\approx_\Lambda \ : \ \Lambda \subseteq \mathcal{L}\}$ is the (point-wise) largest domain bisimulation: if $\dot{\approx}$ is a domain bisimulation, then $\dot{\approx}_\Lambda \subseteq \approx_\Lambda$ for all Λ. Two open processes $\boldsymbol{P}, \boldsymbol{Q}$ are Λ-bisimilar if and only if for all closing substitutions σ, $\boldsymbol{P}\sigma \approx_\Lambda \boldsymbol{Q}\sigma$.

Remark 3.15 (Initial Elements) *To show $K \approx_\Lambda L$ for a specific Λ, we can exhibit a domain bisimulation $\dot{\approx} = \{\dot{\approx}_\Delta \ : \ \Delta \subseteq \mathcal{L}\}$ such that $K \dot{\approx}_\Lambda L$ and $\dot{\approx}_\Delta$ is the empty set for all Δ smaller than Λ.*

A proof that the largest domain bisimulation indeed exists in our non-standard setting can be found in [18]. Domain bisimilarity is a coinductive relation, preserved by structural congruence and monotonic in the domain Λ. Under the mild assumption that the query language does not depend on scripts (Definition 3.8), domain bisimilarity enjoys two important properties which make it a useful proof method for domain congruence:

- domain bisimilarity is a *congruence*, that is embedding open processes in extended contexts preserves bisimilarity;
- domain bisimilarity is a *sound approximation* of the domain congruence induced by request observables, that is if two processes are bisimilar then they are request-congruent.

These important properties are summarized by the theorem below.

Theorem 3.16 (Properties of Domain Bisimilarity) *Assuming a script-independent query language:*

1. *If $K \approx_\Lambda L$, $K \equiv K'$ and $L \equiv L'$, then $K' \approx_\Lambda L'$.*
2. *For all sets of locations Λ, Λ', if $\Lambda \subsetneq \Lambda'$ then $\approx_\Lambda \subsetneq \approx_{\Lambda'}$.*
3. *For all full process contexts $C \in \mathcal{K}_\mathrm{f}$ (Definition 3.5), if $\boldsymbol{P} \approx_\Lambda \boldsymbol{Q}$ then $C[\boldsymbol{P}] \approx_\Lambda C[\boldsymbol{Q}]$.*
4. *For all $\Lambda, \boldsymbol{P}, \boldsymbol{Q}$ (where $\boldsymbol{P}$ and $\boldsymbol{Q}$ have no free trigger names), if $\boldsymbol{P} \approx_\Lambda \boldsymbol{Q}$ then $\boldsymbol{P} \sim^\Lambda \boldsymbol{Q}$.*

Proof. The proofs for (1) and (2) are in Appendix B.1. The proofs for (3) and (4) are respectively in Appendix B.2 and Appendix B.3. □

We now give a first example of the proof method. Larger examples are given in Chapter 4.

Example 3.17 (Proof Method) *Recall the asynchrony law of Section 3.1. It states that a communication buffer cannot be distinguished from the empty process. By definition, $!\boldsymbol{l}\cdot\mathsf{FW}(\boldsymbol{a}, \boldsymbol{a}, \widetilde{\pi}) \approx_\Lambda \mathbf{0}$ if for any closing substitution σ, $(!\boldsymbol{l}\cdot\mathsf{FW}(\boldsymbol{a}, \boldsymbol{a}, \widetilde{\pi}))\sigma \approx_\Lambda \mathbf{0}\sigma$. Given an arbitrary σ we have that $(!\boldsymbol{l}\cdot\mathsf{FW}(\boldsymbol{a}, \boldsymbol{a}, \widetilde{\pi}))\sigma = !l\cdot\mathsf{FW}(a, a, \widetilde{\pi})$ for some a, l. To show that $!l\cdot\mathsf{FW}(a, a, \widetilde{\pi}) \approx_\Lambda \mathbf{0}$, we need to give a domain bisimulation $\dot{\approx} = \{\dot{\approx}_\Delta\}$ such that $\dot{\approx}_\Lambda$ contains the two processes. Since structural congruence preserves bisimilarity (Point 1 of Theorem 3.16), we reason up-to $\equiv$.*

For each Δ, we begin with a relation $\mathcal{R}^0_\Delta = \{(!l\cdot\mathsf{FW}(a, a, \widetilde{\pi}), \mathbf{0})\}$ containing the pair that we want to prove bisimilar. By definition of bisimilarity, we must close the relation under transitions. Due to (Lts In) we must close the relation under parallel compositions with arbitrary output processes: $\mathcal{R}^1_\Delta = \{(M \mid !l\cdot\mathsf{FW}(a, a, \widetilde{\pi}), M)\}$ where $M = \prod_{0 \le i \le n} \overline{l_i \cdot c_i}\langle \widetilde{v_i} \rangle$, $\mathit{dom}(M) \subseteq \Delta$ and $\mathit{scripts}(\widetilde{v_i}) = \emptyset$ (note that $\mathcal{R}^1_\Delta = \mathcal{R}^0_\Delta$ if $n = 0$). The possible tau transitions arising from the interaction of $!l\cdot\mathsf{FW}(a, a, \widetilde{\pi})$ and an output $\overline{l\cdot a}\langle \widetilde{v} \rangle$ where $\widetilde{v} = \widetilde{\pi}\sigma$ are already covered because by (Lts !Com) and (Lts Struct), $\overline{l\cdot a}\langle \widetilde{v} \rangle \mid !l\cdot\mathsf{FW}(a, a, \widetilde{\pi}) \xrightarrow{l\cdot\tau} \overline{l\cdot a}\langle \widetilde{v} \rangle \mid !l\cdot\mathsf{FW}(a, a, \widetilde{\pi})$. Again by definition of bisimilarity, we must make the relation symmetric, hence we conclude with $\dot{\approx}_\Delta = \mathcal{R}^1_\Delta \cup (\mathcal{R}^1_\Delta)^{-1}$.

Incompleteness. In general, domain bisimilarity is a more restrictive equivalence than domain congruence. The property is intrinsic to our choice of giving a proof method parametric in the chosen query and update language. In fact, without specializing the labelled transition system to a particular language, we are forced to distinguish request transitions as soon as queries are syntactically different. On the other hand, equivalences dependent on specific knowledge of

the semantics of queries would lead to optimizations which are no longer correct when the query language changes.[12]

Example 3.18 (Incompleteness) *Consider the query language* $\mathsf{Sam}^{\#}$ *and the process definition*

$$X(\mathtt{a},\mathtt{b}) \stackrel{\text{def}}{=} (\nu\, c)(\overline{l{\cdot}c} \mid !l{\cdot}c.(\nu\, e)\begin{pmatrix} l{\cdot}\mathsf{req}_{(x)\mathtt{a}[\,]}\langle e\rangle \mid \\ l{\cdot}e(x).(\nu\, e')(l{\cdot}\mathsf{req}_{(x)\mathtt{b}[\,]}\langle e'\rangle \mid l{\cdot}e'(x).\overline{l{\cdot}c}) \end{pmatrix})$$

The process loops, replacing at each iteration whatever tree is at l *first with* $\mathtt{a}[\,]$ *and then with* $\mathtt{b}[\,]$*. We have* $X(\mathtt{a},\mathtt{b}) \sim^{\{l\}} X(\mathtt{b},\mathtt{a})$*, because once the two processes are inserted in the same store, they can always reduce to each other. On the other hand, we have that* $X(\mathtt{a},\mathtt{b}) \not\approx_{\{l\}} X(\mathtt{b},\mathtt{a})$ *because the request transitions cannot be matched.*

4 Distributed Query Patterns

In this Section, we use bisimilarity to study some communication patterns used by servers in distributed query systems to answer queries from clients. In Core X$d\pi$, distributed queries take the form of processes which retrieve and combine data from different locations by using remote communication and local requests. We show that some existing patterns [29] can be combined together obtaining a flexible infrastructure which is provably equivalent to an intuitive specification of the intended behaviour. By exploiting process migration, we also propose a new communication pattern, and we show that it is behaviourally equivalent to a naive, less efficient one.

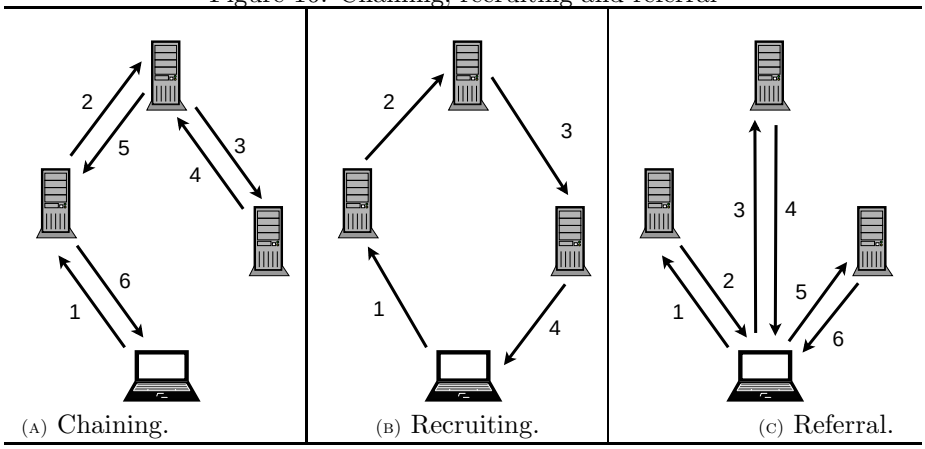
4.1 Chaining, recruiting and referral

We now consider *chaining*, *recruiting* and *referral*, three distributed query patterns studied by Sahuguet *et al.* in [29, 27] and described below. These patterns are interesting because, as will soon be apparent, they are simple yet can express ways of answering requests which are non-trivial, and display different levels of cooperation between the parties involved.

The usage of these patterns presupposes an architecture of servers sharing a common communication protocol for answering cooperatively the queries issued by clients. The protocol consists of alternative actions which depend on the contents of a query and on the local data, and is implemented by dedicated services running on each peer. The distributed querying infrastructure obtained

[12] Even if, for the sake of argument, we fixed a concrete query-update language and knew everything about its semantic equivalences, it would still be unclear how to deal with the case of Example 3.18. There, the initial updates that two processes can perform are by no means equivalent, yet by an "idempotence" argument the overall behaviours turn out to be equivalent. A complete bisimilarity would need to be able to consider is some way the cumulative effect of request transitions, also when interleaved with communication steps, in order to equate sequences of updates with the same global effect on the data-store.

Figure 16: Chaining, recruiting and referral



(A) Chaining. (B) Recruiting. (C) Referral.

by combining the three query patterns is very flexible and can provide location independence to the clients. In fact, a client simply needs to invoke a service on a peer acting as the "entry point" to the network in order to get access to data which may reside on some other server unknown to the client itself.

We now describe the three patterns. In each case, a server receiving a query will try to execute it locally, and if that is not possible, will take alternative action.

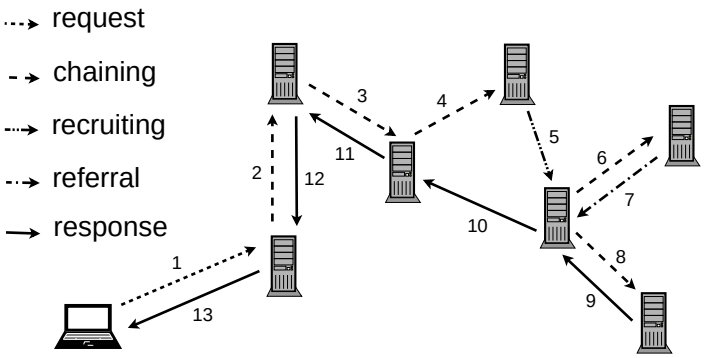
Chaining (Figure 16.(A)): if a server cannot deal directly with the call, it re-issues it to an alternative server, waits for the answer, and then returns the answer to the client.

Recruiting (Figure 16.(B)): if a server cannot deal directly with the call, it forwards it to another server (without noticing the client), so that the result will eventually return to the client without further intervention of the first server. To implement this pattern the address for returning the result must be a parameter of the call, and the client must be willing to accept asynchronous connections.

Referral (Figure 16.(C)): if a server cannot deal directly with the call, it suggests to the client an alternative server which might be able to. This strategy requires active collaboration from the client, which must be ready to contact the alternative server.

When each server involved in answering a request is able to use any of the patterns above, the flow of the data from the initial service call to the final answer can become complex and involve arbitrary combinations of the patterns, as in the example shown in Figure 17.

Figure 17: Combining the query patterns



4.1.1 Implementing the patterns

We now describe, step by step, some Core X$d\pi$ code which implements a system where a client request can be answered by servers using an arbitrary combination of chaining, recruiting and referral. The code is based on services which retrieve and combine data from different locations by exploiting remote communication and local requests. In Section 2.6, we have seen how to represent service calls in Core X$d\pi$:

$$(\nu\, c)(l{\cdot}\overline{m{\cdot}a}\langle\widetilde{v}, l, c\rangle \mid l{\cdot}c(\widetilde{\pi}).\boldsymbol{P})$$

where a is the name of the service to be invoked at location m with parameters $\widetilde{v}$ yielding a result to be passed on the channel c local to l, and $\boldsymbol{P}$ is the code for handling the results, which are expected to match pattern $\widetilde{\pi}$. In this section, a service call will carry four parameters: a tree T used to represent a condition, checked using pattern matching, that a server must satisfy in order to provide the right service (for example specifying the kind of result expected), a query p which is meant to be run on the store of the service matching tag T, and the return parameters m and c stating the location and the channel where the result should be returned. This approach can be easily applied also to service call having more parameters.

A client must be able to deal with the referral query pattern, therefore its code consists essentially of a loop. The loop consists of calling a first server (which could in principle provide the final result, terminating the loop), and then repeating the same call at the alternative addresses received in unsuccessful replies, until a reply containing the final result is received. The context defined below implements the loop at location m:

$$m{\cdot}\mathsf{Ref}_{(n,l,s,T,p,z)}[-] \stackrel{\text{def}}{=} (\nu\, c)\left(\begin{array}{l} m{\cdot}c(\mathtt{OK}[\,], z).- \quad \mid \\ \overline{m{\cdot}c}\langle\mathtt{REF}[\,], l\rangle \mid\, !m{\cdot}c(\mathtt{REF}[\,], x).m{\cdot}\overline{x{\cdot}s}\langle T, p, n, c\rangle \end{array}\right)$$

It is parametric in the location n where the result must be returned, the location l of the first server to be interrogated, and the parameters of the call: the service

Figure 18: Syntax: abbreviations

(Internal Choice) $\displaystyle\bigoplus_{i\in 1..n} {}^{\boldsymbol{l}}\boldsymbol{P}_i \stackrel{\text{def}}{=} (\nu\, c)(\overline{\boldsymbol{l}\cdot c}\,|\prod_{i\in 1..n} \boldsymbol{l}\cdot c.\boldsymbol{P}_i) \qquad c \notin \mathit{fn}(\boldsymbol{P}_i)$

Notation: $\displaystyle\prod_{i\in 1..n} \boldsymbol{P}_i \stackrel{\text{def}}{=} \boldsymbol{P}_1 \,|\, \ldots \,|\, \boldsymbol{P}_n; \qquad \boldsymbol{P}_1 \oplus^l \ldots \oplus^l \boldsymbol{P}_n \stackrel{\text{def}}{=} \bigoplus_{i\in 1..n} {}^{\boldsymbol{l}}\boldsymbol{P}_i.$

name s and condition T, the actual query p and the variable z for binding the result in the continuation. The context uses a private channel c to implement the referral loop and uses the tags $\texttt{OK}[\,]$ and $\texttt{REF}[\,]$ as guards to exit or continue the loop. Any process built using this context always starts the referral loop by invoking s at l and then waiting for two possible answers: either a referral message with the tag $\texttt{REF}[\,]$ and the name of an alternative location (bound to x), which starts another iteration of the loop against the corresponding server, or a result message with the tag $\texttt{OK}[\,]$ and the result of the service call (bound to z), which terminates the loop and passes the result on to the process which replaces the context hole "$-$".

The server filters calls based on the parameter T in order to decide whether they can be served locally or not. Its code, which uses the conventions of Figure 18, consists of the following two processes, run in parallel:

$$l\cdot\mathsf{Local}_{(s,T)} \stackrel{\text{def}}{=} !l\cdot s(T,x,y,z).(\nu\, c)(l\cdot\mathsf{req}_x\langle c\rangle \,|\, l\cdot c(w).l\cdot\overline{y\cdot z}\langle\texttt{OK}[\,],w\rangle)$$

$$l\cdot\mathsf{Remote}_{(s,\Delta)} \stackrel{\text{def}}{=} \prod_{(m,S_m)\in\Delta} !l\cdot s(S_m,x,y,z).\left(\begin{array}{l} l\cdot\overline{m\cdot s}\langle S_m,x,y,z\rangle \\ \oplus^l\, l\cdot\overline{y\cdot z}\langle\texttt{REF}[\,],m\rangle \\ \oplus^l\, l\cdot\mathsf{Ref}_{(l,m,s,S_m,x,w)}[l\cdot\overline{y\cdot z}\langle\texttt{OK}[\,],w\rangle]. \end{array}\right)$$

If the first parameter matches T, the server runs the query (bound to x) on the local data and sends the result back to the client on channel z at y. If the first parameter does not match T, the server selects another server more appropriate for that request out of the set Δ relating servers to tags specifying their services (the outermost parallel composition of the remote process). It then invokes s on the chosen server using either chaining (third branch of the choice), or recruiting (first branch), or referral (second branch). In the case of chaining, the server runs the same code as the client with different parameters. Notice that the code handling the result forwards the result to the client instead of using it locally.

Installation. In order to use these patterns, the code implementing the services must be installed somehow on each participating server. We can assume that it is pre-installed on each peer, or we can install it "on demand" using either process migration or a specialized service which runs scripts. For example, consider the code of a service $\boldsymbol{P}$ parametric in the location x where the service is run and some other initialization pattern π. If we assume that an arbitrary location l exists then, given an arbitrary initialization parameter $v = \pi\sigma$, it is easy to see that running the code at m is equivalent to installing the service

code from l

$$\boldsymbol{P}\{^{m}/_{x}\}\sigma \sim^{\{l\}} l{\cdot}\mathsf{go}\, m.(x,\pi)\boldsymbol{P} \circ \langle m, v\rangle.$$

Alternatively, one could use a dedicated installation service *Inst* at location m which receives an abstraction and some parameters, and runs the abstraction locally

$$\boldsymbol{P}\{^{m}/_{x}\}\sigma \sim^{\{l\}} (\nu\, \mathit{Inst})(l{\cdot}\overline{m{\cdot}\mathit{Inst}}\langle(x,\pi)\boldsymbol{P}, v\rangle \mid !m{\cdot}\mathit{Inst}(y,z).y \circ \langle m, z\rangle).$$

4.1.2 Relating the patterns to a specification

We use a simple system with a client and two servers as an example of how to reason using our equivalences. The reasoning is analogous in the case of multiple servers. The client is on peer m, and runs the code

$$m{\cdot}\mathsf{Client}_{(l,s,\mathtt{a}[\,])} \stackrel{\text{def}}{=} m{\cdot}\mathsf{Ref}_{(m,l,s,\mathtt{a}[\,],p,z)}[m{\cdot}\boldsymbol{P}]$$

where the service is requested to match the tag $\mathtt{a}[\,]$, and the continuation process $\boldsymbol{P}$ is an arbitrary process located at m which does not contain free occurrences of channel c mentioned in the definition of $\mathsf{Ref}_{(-)}$. A server is composed by the parallel composition of the branches dealing with local and remote processing, as described in Section 4.1.1:

$$l_1{\cdot}\mathsf{Server}_{(s,T,l_2,S)} \stackrel{\text{def}}{=} l_1{\cdot}\mathsf{Remote}_{(s,\{(l_2,S)\})} \mid l_1{\cdot}\mathsf{Local}_{(s,T)}.$$

We consider two processes P_1 and P_2, where a client requests from the server at l_1 the data specified by $\mathtt{a}[\,]$ (served locally) or $\mathtt{b}[\,]$ (served remotely at l_2).

$$\mathsf{Servers} \stackrel{\text{def}}{=} l_1{\cdot}\mathsf{Server}_{(s,\mathtt{a}[\,],l_2,\mathtt{b}[\,])} \mid l_2{\cdot}\mathsf{Server}_{(s,\mathtt{b}[\,],l_1,\mathtt{a}[\,])}$$

$$P_1 \stackrel{\text{def}}{=} m{\cdot}\mathsf{Client}_{(l_1,s,\mathtt{a}[\,])} \mid \mathsf{Servers}$$

$$P_2 \stackrel{\text{def}}{=} m{\cdot}\mathsf{Client}_{(l_1,s,\mathtt{b}[\,])} \mid \mathsf{Servers}$$

We compare P_1 and P_2 with Q_1 and Q_2 defined below, which provide a specification of the expected behaviour respectively of P_1 and P_2. Each process goes directly from m to the relevant location, fetches the data returned by query p, and goes back to paste it as the new data tree of m:

$$m{\cdot}\mathsf{Spec}_{(l)} \stackrel{\text{def}}{=} m{\cdot}\mathsf{go}\, l.(\nu\, c)(l{\cdot}\mathsf{req}_p\langle c\rangle \mid l{\cdot}c(z).l{\cdot}\mathsf{go}\, m.m{\cdot}\boldsymbol{P})$$

$$Q_1 \stackrel{\text{def}}{=} m{\cdot}\mathsf{Spec}_{(l_1)} \mid \mathsf{Servers}$$

$$Q_2 \stackrel{\text{def}}{=} m{\cdot}\mathsf{Spec}_{(l_2)} \mid \mathsf{Servers}.$$

An important difference between the client and the specification is that the client sends an output message to a service which can in principle be intercepted by some process external to the protocol which performs an input on the same service channel. To rule out this undesired interference, we restrict the name

of the service s both in each P_i and Q_i, with the side effect of preventing also the unharmful case in which several clients use the services at the same time.[13]

We can show, in a domain containing both l_1 and l_2, the following equivalences:

$$(\nu\, s)P_1 \sim^{\{l_1,l_2\}} (\nu\, s)Q_1 \qquad (\nu\, s)P_2 \sim^{\{l_1,l_2\}} (\nu\, s)Q_2$$

Hence, by definition, we can replace $(\nu\, s)P_i$ by $(\nu\, s)Q_i$ in any network, and preserve network equivalence.

Proof of equivalence. By virtue of Theorem B.21, a formal proof of each equivalence above would involve showing the existence of an appropriate domain bisimulation containing the relevant pair, along the lines of the examples of Section 3.2.2.

We show here a sketch for the case for P_2 and Q_2. In order to make the proof more manageable, we adopt the simplifying assumption that the query p does not contain scripts. The proof of the general case follows a similar structure. Moreover, we use implicitly the closure of bisimilarity under structural congruence.

We start by analyzing the non-input transitions of the two processes, and then we indicate how to build a domain bisimulation by pairing compatible states and dealing with input transitions. Consider $S_0 = (\nu\, s)Q_2$. By structural congruence, $S_0 \equiv m{\cdot}\mathsf{Spec}_{(l_2)} \mid (\nu\, s)\mathsf{Servers}$, which makes it easy to see that $\mathsf{Servers}$ does not have transitions, because of the restriction on s. Hence, we concentrate on $m{\cdot}\mathsf{Spec}_{(l_2)}$. All it can do is a tau transition at l_2 corresponding to a migration step to reach a state S_1, followed by a request transition at l_2 to become

$$S_2 = (\nu\, s)((\nu\, c)(\overline{l_2{\cdot}c}\langle V\rangle \mid l_2{\cdot}c(z).l_2{\cdot}\mathsf{go}\, m.m{\cdot}\boldsymbol{P}) \mid \mathsf{Servers})$$

where V stands for a generic result obtained by the request. In turn, this process can only do a local communication followed by a migration (both tau transitions, respectively at l_2 and m) to become

$$S_3^V \;=\; (\nu\, s)((\nu\, c)(m{\cdot}\boldsymbol{P}\{^V/_z\}) \mid \mathsf{Servers}).$$

Using the hypothesis $c \notin \mathit{fn}(\boldsymbol{P})$, we obtain

$$(\nu\, s)((\nu\, c)(m{\cdot}\boldsymbol{P}\{^V/_z\}) \mid \mathsf{Servers}) \equiv (\nu\, s)(m{\cdot}\boldsymbol{P}\{^V/_z\} \mid \mathsf{Servers}).$$

Hence, we have shown that the transition for S_0 that we need to match is

$$S_0 \xrightarrow{l_2{\cdot}\mathsf{req}\langle p\rangle(V)} \!\!\rightarrow S_3^V = S_0 \xrightarrow{\tau} S_1 \xrightarrow{l_2{\cdot}\mathsf{req}\langle p\rangle(V)} S_2^V \xrightarrow{\tau} \!\!\rightarrow S_3^V\,.$$

All we need to show now is that also starting from $(\nu\, s)P_2$, for each possible execution path, we can only do an analogous weak request transitions, and reach a state equivalent either to S_3 above. We now analyze the transitions of $S_0' = (\nu\, s)P_2$. First the client $m{\cdot}\mathsf{Client}_{(l_1,s,\mathsf{b}[])}$ performs a tau transition at m

[13] In future, we plan to consider less restrictive ways to rule out this kind of interference using the type based techniques for linearity of Yoshida *et al.* [42].

corresponding to the initialization of the loop and then one at l_1 corresponding to the migration of the service call. The whole process becomes

$$S'_1 = (\nu\, s, c)(C_0 \mid \overline{l_1 \cdot s}\langle \mathtt{b}[\,], p, m, c\rangle \mid \mathsf{Servers})$$

where

$$C_0 = m \cdot s(\mathtt{OK}[\,], z).m \cdot \boldsymbol{P} \mid !m \cdot s(\mathtt{REF}[\,], x).m \cdot \overline{x \cdot s}\langle \mathtt{b}[\,], p, n, c\rangle.$$

Because of $\mathtt{b}[\,]$, $l_1 \cdot \mathsf{Server}_{(s,\mathtt{a}[\,],l_2,\mathtt{b}[\,])}$ receives the call in the remote branch (the one for l_2 with tag $\mathtt{b}[\,]$). Nondeterministically, the process at l_1 evolves to either

(recruiting) $S'_2 = l_1 \cdot \mathsf{Server}_{(s,\mathtt{a}[\,],l_2,\mathtt{b}[\,])} \mid l_1 \cdot \overline{l_2 \cdot s}\langle \mathtt{b}[\,], p, m, c\rangle$

(referral) $S'_3 = l_1 \cdot \mathsf{Server}_{(s,\mathtt{a}[\,],l_2,\mathtt{b}[\,])} \mid l_1 \cdot \overline{m \cdot c}\langle \mathtt{REF}[\,], l_2\rangle$

(chaining) $S'_4 = l_1 \cdot \mathsf{Server}_{(s,\mathtt{a}[\,],l_2,\mathtt{b}[\,])} \mid l_1 \cdot \mathsf{Ref}_{(l_1,l_2,s,\mathtt{b}[\,],p,w)}[l_1 \cdot \overline{m \cdot c}\langle \mathtt{OK}[\,], w\rangle].$

Both the communication for receiving the call and the choice of the branch to execute are two tau transitions at l_1, so $S'_1 \xrightarrow{\tau}\!\!\!\rightarrow S'_i$ for $i \in 2..4$. We now consider the transitions of each choice branch.

Recruiting. If recruiting is chosen, the server at l_1 performs a tau transition at l_2 corresponding to the forwarding of the client request, becoming

$$S'_{2,1} = (\nu\, s, c)(C_0 \mid R_1 \mid l_1 \cdot \mathsf{Server}_{(s,\mathtt{a}[\,],l_2,\mathtt{b}[\,])} \mid \overline{l_2 \cdot s}\langle \mathtt{b}[\,], p, m, c\rangle \mid l_2 \cdot \mathsf{Server}_{(s,\mathtt{b}[\,],l_1,\mathtt{a}[\,])})$$

where R_1 is a deadlocked process containing the code for the two discarded choice branches. Due to the parameter $\mathtt{b}[\,]$, server l_2 receives the call in the local branch, performing another tau transition at l_2:

$$S'_{2,2}(\nu\, s, c)(\ldots \mid l_2 \cdot \mathsf{Server}_{(s,\mathtt{b}[\,],l_1,\mathtt{a}[\,])} \mid (\nu\, c')(l_2 \cdot \mathsf{req}_p\langle c'\rangle \mid l \cdot c'(w).l_2 \cdot \overline{m \cdot c}\langle \mathtt{OK}[\,], w\rangle)).$$

This process can only do a request transition (say obtaining data V) to $S^V_{2,3}$, followed by a tau transition (corresponding to local communication) at l_2, becoming

$$S'_{2,4} = (\nu\, s, c)(C_0 \mid R_1 \mid \mathsf{Servers} \mid l_2 \cdot \overline{m \cdot c}\langle \mathtt{OK}[\,], V\rangle),$$

where V stands for a generic result obtained by the request. After two tau transitions at m, corresponding to migration and local communication between C_0 and $\overline{m \cdot c}\langle \mathtt{OK}[\,], V\rangle$, we obtain

$$S^V_{2,5} \;=\; (\nu\, s)(m \cdot \boldsymbol{P}\{^V/_x\} \mid (\nu\, c)(!m \cdot s(\mathtt{REF}[\,], x).m \cdot \overline{x \cdot s}\langle \mathtt{b}[\,], p, n, c\rangle) \mid R_1 \mid \mathsf{Servers}),$$

where we stop, with

$$S'_2 \xrightarrow{\tau}\!\!\!\rightarrow S'_{2,2} \xrightarrow{l_2 \cdot \mathsf{req}\langle p\rangle(V)} S^V_{2,3} \xrightarrow{\tau}\!\!\!\rightarrow S_{2,5}V.$$

Referral. In the case of referral, the server performs two tau transitions at m which correspond to the forwarding of the message referring location l_2, and to a second iteration of the referral loop of the client. The client then sends a new call to l_2 (a tau transition at l_2) and becomes

$$S'_{3,1} = (\nu\, s, c)(C_0 \mid R_2 \mid l_1{\cdot}\mathsf{Server}_{(s,\mathsf{a}[\,],l_2,\mathsf{b}[\,])} \mid \overline{l_2{\cdot}s}\langle \mathsf{b}[\,], p, m, c\rangle \mid l_2{\cdot}\mathsf{Server}_{(s,\mathsf{b}[\,],l_1,\mathsf{a}[\,])})$$

where R_2 is a deadlocked process containing the code for the two discarded choice branches. From now on, the transitions are the same as in the case for recruiting until we obtain the process

$$S^V_{3,4} \;=\; (\nu\, s)(m{\cdot}\boldsymbol{P}\{^V/_x\} \mid (\nu\, c)(!m{\cdot}s(\mathtt{REF}[\,], x).m{\cdot}\overline{x{\cdot}s}\langle \mathsf{b}[\,], p, n, c\rangle) \mid R_2 \mid \mathsf{Servers}),$$

where we stop, with

$$S'_3 \xrightarrow{\tau}\!\!\!\rightarrow S'_{3,2} \xrightarrow{l_2{\cdot}\mathsf{req}\langle p\rangle(V)} S^V_{3,3} \xrightarrow{\tau}\!\!\!\rightarrow S^V_{3,4}.$$

Chaining. In the case of chaining, the reasoning is similar. The first transitions correspond to a local communication at l_1 starting the referral loop of the server and a migration followed by communication at l_2 to start the local branch of that service. At this point, the process performs a request transition analogous to the one in the previous cases, and tau transitions corresponding to a local communication to get the result at l_2, a migration to l_1, and a local communication to terminate the referral loop of the server. The continuation process performs the transitions corresponding to the migration of the final result to m and to the communication to terminate the referral loop of the client, reaching

$$S^V_{4,3} \;=\; (\nu\, s)(m{\cdot}\boldsymbol{P}\{^V/_x\} \mid (\nu\, c)(!m{\cdot}s(\mathtt{REF}[\,], x).m{\cdot}\overline{x{\cdot}s}\langle \mathsf{b}[\,], p, n, c\rangle) \mid R_3 \mid \mathsf{Servers}),$$

where R_3 is a deadlocked process containing the code for the two discarded choice branches and the residual of the referral loop at l_1, and we stop, with

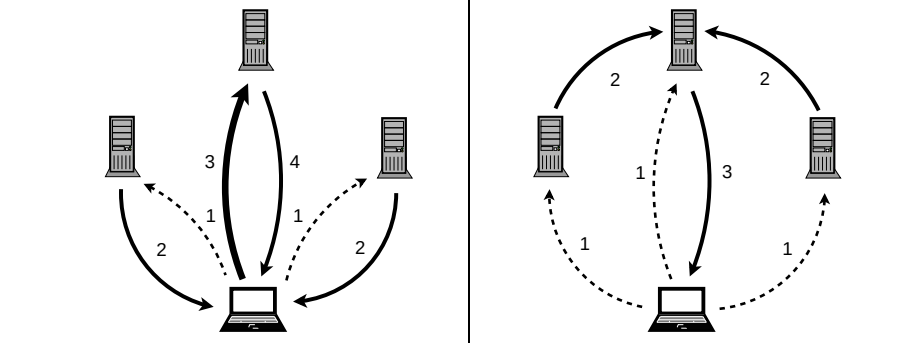
$$S'_4 \xrightarrow{\tau}\!\!\!\rightarrow S'_{4,1} \xrightarrow{l_2{\cdot}\mathsf{req}\langle p\rangle(V)} S^V_{4,2} \xrightarrow{\tau}\!\!\!\rightarrow S^V_{4,3}.$$

Intuitively, S_0, S'_2, S'_3 and S'_4 are all equivalent states, because they are structurally equivalent to a process of the form

$$(\nu\, s)(m{\cdot}\boldsymbol{P}\{^V/_z\} \mid \mathsf{Servers} \mid \delta).$$

The bisimulation relation we are looking for is obtained in three steps. First, we pair each of the states S_0, S_1 preceding the request transition in the lts of $(\nu\, s)Q_2$ with each $S'_0, \ldots, S'_{2,1}, S'_{2,2}, \ldots, S'_{4,1}$ preceding a request transition in the lts of $(\nu\, s)P_2$ (and vice versa). Second, we pair each of the states S^V_2, S^V_3 following the request transition giving a particular result V in the lts of $(\nu\, s)Q_2$ with each $S^V_{i,j}$ following a request transition giving the same result in the lts of $(\nu\, s)P_2$ (and vice versa). Third, we close the relation obtained so far under parallel composition with the output messages derived by input transitions (as shown explicitly in Example 3.17). The relation defined above can be shown to be a domain bisimulation by formally checking the definition.

Figure 19: Rendez-vous



(A) Query-shipping. (B) Rendez-vous.

4.2 Rendez-vous and shipping

In the previous example, the infrastructure of servers implementing the distributed query patterns was fixed in advance, while the actual interactions between them were determined at run-time. The messages exchanged between different locations were always service calls or their results. Now, we consider a more flexible scenario which exploits code mobility.

Data-shipping and *query-shipping* are two traditional database techniques for distributed query evaluation: the first consists of evaluating locally a query on remote data by asking for the relevant data to be sent from the remote sources; the second consists of delegating the evaluation of a query to one of the remote sources in order to reduce the bandwidth used by data transfers. In the next section, we propose a distributed query pattern, called *rendez-vous*, which combines data and query shipping by using code mobility and private channels. The idea is to give a client the ability to ship result-handling code to another location, and to redirect the results of arbitrary service calls towards the location containing the result-handler. Within an infrastructure of services such as the one used above for chaining, recruiting and referral, this pattern can help to save bandwidth by eliminating unnecessary data transfers.

4.2.1 The rendez-vous query pattern

We now compare the query-shipping and rendez-vous patterns by giving a concrete example where a client calls a remote service using as parameters two large sets of data obtained by other remote service calls.

Suppose that on location l there is a specialized service $l{\cdot}\texttt{Join}(x_1, x_2, y, z)$ which returns on channel z at location y the result of joining the data bound to x_1 with the data bound to x_2. Suppose moreover that a client running on location m wants to join some data obtained by query p at location l_1 with other data obtained by query q at location l_2. We assume that l_1 and l_2 run

the services described in Section 4.1.2, that l_1 (respectively l_2) serves locally the requests tagged by $\mathtt{a}[\,]$ (respectively $\mathtt{b}[\,]$).

Query shipping. The client can use query shipping: it first invokes the query services at locations l_1 and l_2, then passes on the results as inputs to the join service on location l (see Figure 19(A)). Below we give the code of a client implementing this approach:

$$m{\cdot}\mathsf{ClientQ} \stackrel{\text{def}}{=} (\nu\, c, c_1, c_2) \left(\begin{array}{l} \quad m{\cdot}\overline{l_1{\cdot}\mathtt{s}}\langle \mathtt{a}[\,], p, m, c_1\rangle \\ \mid\ m{\cdot}\overline{l_2{\cdot}\mathtt{s}}\langle \mathtt{b}[\,], q, m, c_2\rangle \\ \mid\ m{\cdot}c_1(\mathtt{OK}[\,], x_1).m{\cdot}c_2(\mathtt{OK}[\,], x_2).m{\cdot}\overline{l{\cdot}\mathtt{Join}}\langle x_1, x_2, m, c\rangle \\ \mid\ m{\cdot}c(z).m{\cdot}\boldsymbol{P} \end{array} \right)$$

It starts sending off the two service calls to l_1 and l_2 and then waits for the results respectively on c_1 and c_2 to bind them to x_1 and x_2. The remaining code is a standard service call for the join service at l with parameters x_1 and x_2, binding the final result to z in the continuation $m{\cdot}\boldsymbol{P}$, which can be an arbitrary process.

Rendez-vous. In order to save bandwidth, a better strategy is to request the query services at l_1 and l_2 to forward their results to location l, and to install at l a process which collects the two results and invokes the join service locally, asking for the final result to be returned at location m (see Figure 19(B)). Below we give a context implementing the general pattern, with two holes for inserting the code to handle the intermediate results at l and the final result at m. The code is parametric in the tags T_i and the queries p_i used to determine the partial results, the variables x_i for binding them in the intermediate code at "$-_1$", and the variable z for binding the final result in the continuation code at "$-_2$":

$$m{\cdot}\mathsf{RzV}_{(T_1,p_1,x_1,T_2,p_2,x_2,z)}[-]_1[-]_2 \stackrel{\text{def}}{=}$$

$$(\nu\, c, c_1, c_2) \left(\begin{array}{l} \quad m{\cdot}\overline{l_1{\cdot}\mathtt{s}}\langle T_1, p_1, l, c_1\rangle \\ \mid\ m{\cdot}\overline{l_2{\cdot}\mathtt{s}}\langle T_2, p_2, l, c_2\rangle \\ \mid\ m{\cdot}\mathsf{go}\, l.l{\cdot}c_1(\mathtt{OK}[\,], x_1).l{\cdot}c_2(\mathtt{OK}[\,], x_2).-_1 \\ \mid\ m{\cdot}c(z).-_2 \end{array} \right)$$

The code given above can be easily parameterized also on the number, the names and the locations of the services involved, and can be adapted to return the final results at an arbitrary location on an arbitrary channel.

We give below the code for a client, equivalent to ClientQ, which uses the rendez-vous strategy:

$$m{\cdot}\mathsf{ClientR} \stackrel{\text{def}}{=} m{\cdot}\mathsf{RzV}_{(\mathtt{a}[\,],p,x_1,\mathtt{b}[\,],q,x_2,z)}[\overline{l{\cdot}\mathtt{Join}}\langle x_1, x_2, m, c\rangle][m{\cdot}\boldsymbol{P}]$$

The code for handling the intermediate results consists in a local call to the join service, whereas the continuation is the same generic process used for ClientQ.

4.2.2 Equivalence of the patterns

Consider the process $\mathsf{Servers}$ defined in Section 4.1 consisting in the parallel compositions of the servers for implementing chaining, recruiting and referral at locations l_1 and l_2. The clients given above, each in parallel with $\mathsf{Servers}$, are equivalent in any network regardless of what locations are present.

$$(\nu\, s)(m{\cdot}\mathsf{ClientQ} \mid \mathsf{Servers}) \sim^{\emptyset} (\nu\, s)(m{\cdot}\mathsf{ClientR} \mid \mathsf{Servers})$$

In order to make the proof more manageable, we adopt once again the simplifying assumption that p and q do not contain scripts, and we use implicitly the closure of bisimilarity under structural congruence.

First of all, we simplify the problem further by studying an equation relating only the parts of the client processes above which are different from each other and which play a significant role in the proof. The full result follows by exploiting the closure of bisimilarity under parallel composition, restriction and structural congruence (Theorem B.15 and Proposition B.1) to recover the processes of the original statement.

Consider the definitions

$$m{\cdot}\mathsf{ClientQ}' \stackrel{\text{def}}{=} (\nu\, c_1, c_2)\left(\begin{array}{l} m{\cdot}\overline{l_1{\cdot}\mathtt{s}}\langle \mathtt{a}[\,], p, m, c_1\rangle \\ \mid\ m{\cdot}\overline{l_2{\cdot}\mathtt{s}}\langle \mathtt{b}[\,], q, m, c_2\rangle \\ \mid\ m{\cdot}c_1(\mathtt{OK}[\,], x_1).m{\cdot}c_2(\mathtt{OK}[\,], x_2).m{\cdot}\overline{l{\cdot}\mathtt{Join}}\langle x_1, x_2, m, c\rangle \end{array}\right)$$

$$m{\cdot}\mathsf{ClientR}' \stackrel{\text{def}}{=} (\nu\, c_1, c_2)\left(\begin{array}{l} m{\cdot}\overline{l_1{\cdot}\mathtt{s}}\langle \mathtt{a}[\,], p, l, c_1\rangle \\ \mid\ m{\cdot}\overline{l_2{\cdot}\mathtt{s}}\langle \mathtt{b}[\,], q, l, c_2\rangle \\ \mid\ m{\cdot}\mathtt{go}\, l.l{\cdot}c_1(\mathtt{OK}[\,], x_1).l{\cdot}c_2(\mathtt{OK}[\,], x_2).\overline{l{\cdot}\mathtt{Join}}\langle x_1, x_2, m, c\rangle \end{array}\right)$$

Our goal is to build a domain bisimulation containing the pair

$$\big((\nu\, s)(m{\cdot}\mathsf{ClientQ}' \mid \mathsf{Servers})\,,\ (\nu\, s)(m{\cdot}\mathsf{ClientR}' \mid \mathsf{Servers})\big).$$

The construction of the proof is summarized by the diagrams in Figure 20. We represent the transitions of the two processes above in the form of lattices of states related by the lts (to be read in the direction of the arrows). Like in Section 4.1.2, we do not consider input transitions at this stage. The dotted arcs indicate the states from the two diagrams which will be paired in the bisimulation.

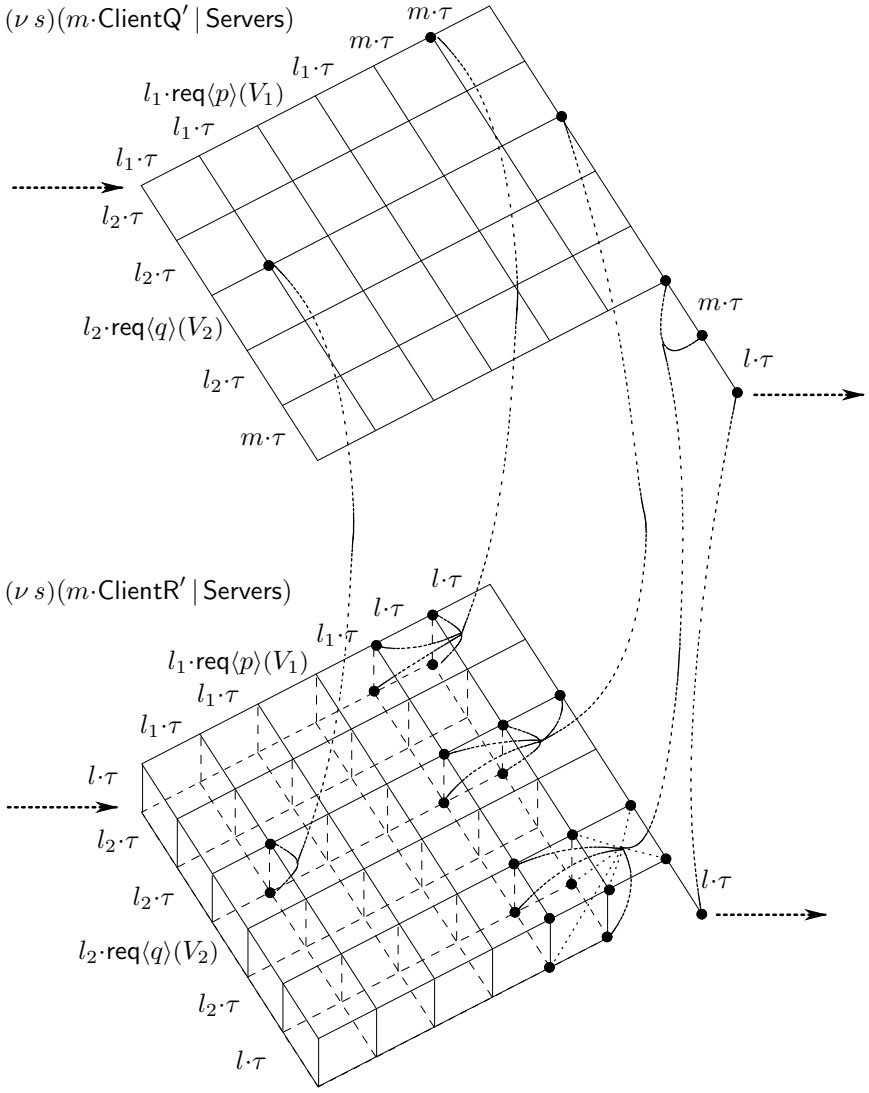
Building the transition diagrams. We describe the steps leading to the transitions. Later, we will explain how to build the bisimulation relation.

We begin with $(\nu\, s)(m{\cdot}\mathsf{ClientQ}' \mid \mathsf{Servers})$, corresponding to the top diagram of Figure 20. The starting state is the one pointed to by an arrow on the left of the diagram. We follow the top-left border of the diagram. Consider the sub-processes $m{\cdot}\overline{l_1{\cdot}\mathtt{s}}\langle \mathtt{a}[\,], p, m, c_1\rangle$ and

$$l_1{\cdot}\mathsf{Local}_{(s,\mathtt{a}[\,])} \stackrel{\text{def}}{=} !l_1{\cdot}s(\mathtt{a}[\,], x, y, z).(\nu\, c')(l_1{\cdot}\mathsf{req}_x\langle c'\rangle \mid l_1{\cdot}c'(w).l_1{\cdot}\overline{y{\cdot}z}\langle \mathtt{OK}[\,], w\rangle).$$

Together they can perform, in order:

Figure 20: Bisimulation Diagrams



1. a migration step from m to l_1;
2. an internal communication on s at l_1;
3. a request transition generating an output $\overline{l_1{\cdot}c'}\langle V_1\rangle$, where V_1 is the data obtained by query p;
4. an internal communication on c';

5. a migration to m.

We are left with the processes $l_1{\cdot}\mathsf{Local}_{(s,\mathtt{a}[])}$ and $\overline{m{\cdot}c_1}\langle \mathtt{Ok}[], V_1\rangle$. The second process can communicate with

$$m{\cdot}c_1(\mathtt{OK}[], x_1).m{\cdot}c_2(\mathtt{OK}[], x_2).m{\cdot}\overline{l{\cdot}\mathtt{Join}}\langle x_1, x_2, m, c\rangle,$$

and we are left with

$$m{\cdot}c_2(\mathtt{OK}[], x_2).m{\cdot}\overline{l{\cdot}\mathtt{Join}}\langle V_1, x_2, m, c\rangle.$$

Independently, $m{\cdot}\overline{l_2{\cdot}\mathtt{s}}\langle \mathtt{b}[], q, m, c_2\rangle$ and $l_2{\cdot}\mathsf{Local}_{(s,\mathtt{b}[])}$ can mimic the 5 steps above (represented in the diagram by the bottom-left border), becoming $l_2{\cdot}\mathsf{Local}_{(s,\mathtt{b}[])}$ and $\overline{m{\cdot}c_2}\langle \mathtt{Ok}[], V_2\rangle$. These two independent groups of respectively 6 and 5 ordered transitions give a lattice of 42 processes related by the lts, where the bottom element is the initial process, and the top element (at the intersection between the top and bottom-right borders) is the process

$$(\nu\, s)\left(\begin{array}{l}\overline{m{\cdot}c_2}\langle \mathtt{Ok}[], V_2\rangle\\ \mid m{\cdot}c_2(\mathtt{OK}[], x_2).m{\cdot}\overline{l{\cdot}\mathtt{Join}}\langle V_1, x_2, m, c\rangle\\ \mid \mathsf{Servers}\end{array}\right)$$

This process can only perform a communication on channel c_2 and a migration from m to l, becoming the point on the right with the outgoing arrow

$$(\nu\, s)(\overline{l{\cdot}\mathtt{Join}}\langle V_1, V_2, m, c\rangle \mid \mathsf{Servers})$$

By a similar reasoning, we can derive for the process $(\nu\, s)(m{\cdot}\mathsf{ClientQ}' \mid \mathsf{Servers})$ the lattice of $36+42+1$ processes reported in the bottom diagram of Figure 20. The vertical transition possible from each of the 36 states in the lower layer of the diagram is the initial migration step from m to l of the result handling code, where

$$m{\cdot}\mathsf{go}\, l.l{\cdot}c_1(\mathtt{OK}[], x_1).l{\cdot}c_2(\mathtt{OK}[], x_2).\overline{l{\cdot}\mathtt{Join}}\langle x_1, x_2, m, c\rangle$$

becomes process

$$l{\cdot}c_1(\mathtt{OK}[], x_1).l{\cdot}c_2(\mathtt{OK}[], x_2).\overline{l{\cdot}\mathtt{Join}}\langle x_1, x_2, m, c\rangle.$$

Only when this transition has occurred, can communication on c_1 at l happen (hence the additional 7 states appearing only in the upper layer). The final state, after communication on c_2 at l has happened, is once again

$$(\nu\, s)(\overline{l{\cdot}\mathtt{Join}}\langle V_1, V_2, m, c\rangle \mid \mathsf{Servers}).$$

Building the bisimulation relation. We now describe how to pair-up the processes (states) of Figure 20 to build a suitable bisimulation. First, we relate each of the 25 processes in the top diagram reachable from the initial one after at most 4 transitions along each axis, with the two corresponding processes in the bottom diagram (as shown by the left-most dotted arc in Figure 20). Then,

we relate each of the 10 processes obtained in the top diagram after 5 transitions along one axis and at most 4 on the other axis with the corresponding 4 processes in the bottom diagram (as shown by the second dotted arc in Figure 20). Next, we relate each of the 5 processes obtained in the top diagram after 6 transitions along the first axis and at most 4 along the second axis, with the corresponding 5 processes in the bottom diagram (as shown by the third dotted arc in Figure 20). Then, we relate the two processes reachable after 11 and 12 transitions in the top diagram with the 10 processes of the bottom diagram to which they are joined by the fourth arc in Figure 20. We also relate the process obtained in the top diagram after 5 transitions along each axis with the 8 processes at the vertices of the corresponding cube in the bottom diagram. Finally, we associate the final process in the top diagram with the final process in the bottom diagram (the fifth dotted arc in Figure 20).

We take the symmetric closure of this relation, and we close it under parallel composition with output messages like in Example 3.17. Note that in the diagram we have shown the transitions for one possible choice of the data items V_1 and V_2 obtained as results of the request transition. To be completely formal, the reasoning above must be quantified on all possible result values, by pairing the corresponding states as in the example of Section 4.1.2.

Following a simulation step. We consider now an example to explain the rationale behind the pairing of states in the relation. We focus on the simulation of $(\nu\, s)(m{\cdot}\mathsf{ClientQ}' \mid \mathsf{Servers})$ by $(\nu\, s)(m{\cdot}\mathsf{ClientR}' \mid \mathsf{Servers})$ which is subtle. The other direction is straightforward.

In the top diagram, the process connected to the leftmost dotted arc can perform a weak transition $\xrightarrow{l_1 \cdot \mathsf{req}\langle p\rangle(V_1)}\!\!\!\!\rightarrow_{\{l_1,m\}}$ to become the process connected to the third dotted arc. In the bottom diagram, the top process connected to the arc can simulate the step by performing a weak transition $\xrightarrow{l_1 \cdot \mathsf{req}\langle p\rangle(V_1)}\!\!\!\!\rightarrow_{\{l_1\}}$ to become the top-left process of those connected to the bottom end of the third arc. The bottom process connected to the first arc cannot simulate the transition by reaching the same process, because that would involve using location l. Instead, it performs a transition $\xrightarrow{l_1 \cdot \mathsf{req}\langle p\rangle(V_1)}\!\!\!\!\rightarrow_{\{l_1\}}$ to become the bottom-left process of those connected to the bottom end of the third arc, which is also in the relation. Also the states reached by the process on the top diagram by performing one or two transitions the less are related to the two states of the bottom diagram mentioned above. The idea is that the last two tau transitions at m cannot be matched by tau transitions at l, in fact they do not need to be matched at all, so the processes in the bottom diagram stay the same.

The association between the states of the two diagrams above is not completely straightforward because we are showing the most general result in which domain congruence holds for the empty domain ($\sim^{\emptyset}$). In this case, we had to make sure that for each transition between processes in the top diagram there was a corresponding transition in the second diagram (possibly null) which related two processes bisimilar to the original ones *involving only locations used also by the original transition*. The problems are due to the additional transi-

tions at l which are possible in the second diagram. If we tried to prove $\sim^{\{l\}}$ instead, the association between processes would have been straightforward.

5 Conclusions

We previously introduced X$d\pi$ [11], a calculus for describing the interaction between processes and active data across distributed locations. In this paper, we have concentrated on Core X$d\pi$, the explicitly-located version of X$d\pi$, in order to study process equivalences. Both calculi, and their formal relationship, are studied in detail in [18]. In Section 3.1, we defined two contextual equivalences for Core X$d\pi$ networks and processes. Network equivalence dictates when two networks can be considered indistinguishable by an observer looking at the interface between processes and local stores. Process equivalence is such that, when we place equivalent processes in equivalent network contexts, we obtain equivalent networks. Both equivalences are parametric with respect to the language used for querying and updating documents, and can be instantiated to specific cases. Contextual equivalences are difficult to use directly. In Section 3.2 we defined *domain bisimilarity*, a coinductive equivalence relation which entails process equivalence. Its non-standard definition, due to the fact that scripts (which can appear in data) are part of the values, and process equivalences are sensitive to the set of locations constituting the network. Our investigations required new reasoning techniques, as well as well-established ones. Domain bisimilarity is intrinsically incomplete, due to its being parametric on a query and update language.

An important design choice, enabling us to study how properties of data can be affected by process interaction, was to model data and processes at the same level of abstraction, rather than encoding data into processes, as customary in the π-calculus [20, 21, 22, 32]. Whilst such an encoding makes sense when using the π-calculus as a low level concurrency modelling language, it becomes a burden when reasoning about the coordination of higher-level processes. Our choice also gave us the opportunity to keep our language modular with respect to the choice of a query language, which can be easily adapted from the existing literature on XML [5].

We delegate migration control to external security checks that can be superimposed on the language. It would be interesting to include an explicit construct to constrain process migration at the location level, perhaps based on types, along the lines of [13]. We have indeed already considered such an extension, which is not included in the present work because it is of limited interest in the untyped setting. Migration has been included in Core X$d\pi$ only to maintain a closer correspondence with X$d\pi$, but is not necessary. In fact, each located action already contains information about where it is to be executed. For example, in the process below we can imagine that each input on a_i at l_i is followed by an implicit migration step $l_i{\cdot}\mathtt{go}\ l_{i+1}$ to the next location:

$$l_1{\cdot}a_1(x).l_2{\cdot}a_2(y).\overline{l_3{\cdot}a_3}\langle x, y\rangle,$$

intuitively corresponds to

$$l_1{\cdot}a_1(x).l_1{\cdot}\mathsf{go}\ \ l_2.l_2{\cdot}a_2(y).l_2{\cdot}\mathsf{go}\ \ l_3.\overline{l_3{\cdot}a_3}\langle x,y\rangle.$$

Overall, if explicit migration were to be discarded, the presentation of our model would be slightly simpler. Nevertheless, we have decided to remain with our original presentation because the encoding of X$d\pi$ in Core X$d\pi$ and its full abstraction (reported in [18]) are interesting results in their own right, confirming the informal thesis of [7] that locations can be encoded, without divergence, in the π-calculus with polyadic synchronization.

Using domain bisimilarity, in Section 4 we studied some communication patterns employed by servers in distributed query systems to answer queries from clients. Queries took the form of processes which retrieve and combine data from different locations by using remote communication and local requests. In particular, we considered chaining, recruiting and referral, three distributed query patterns studied in [29, 27] which are interesting because, despite their simplicity, they can express ways of answering requests which are non-trivial and display different levels of cooperation between the parties involved. By exploiting process migration, we have proposed the rendez-vous query pattern which can help to save bandwidth in certain applications. A challenging application for future work, is to extend the distributed query pattern examples to model a robust system where the servers return streams of results which can dry out or be restarted, and show its equivalence to a simpler, non-streaming specification. We believe that our techniques are suitable for this task, but we need automated tools and symbolic techniques (such as the *open bisimulation* of Sangiorgi [31], or up-to techniques [23] such as "up to confluent reduction"), to help producing manageable bisimilarity proofs. For example, already in the example given in Section 4.2.2, the bisimulation relation was difficult to represent succinctly because each state of one process could be related to several states of the other process.

Studying types for Core X$d\pi$ is ongoing work. A type system would be useful to guarantee the absence of run-time errors, refine the behavioural equivalences, guarantee the conformance of data trees to schemas, and study security properties. Given the use of mobile code in our systems, in the absence of trust, we face the problem of protecting a host from a potentially malicious agent. This problem could be tackled by type-checking each agent dynamically entering a location [13] (possibly relying on the ability of a location to infer the type of the agent), or by using the Proof Carrying Code [24] approach (to send a migrating process along with its type), or by a combination of both techniques. A type system usually restricts the number of terms that are admissible in a calculus. Hence, the behavioural equivalences can become easier to verify (since there can be less-counter-examples), and some laws that do not hold in the untyped calculus become valid for the typed fragment. An obvious theoretical question arising from the definition of a type system for Core X$d\pi$, is to understand how the behavioural equivalences are affected by typing, for example along the lines of [26, 12].

References

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

[2] Lucia Acciai and Michele Boreale. XPi: A typed process calculus for XML messaging. In *Proc. of FMOODS'05*, 2005.

[3] Omar Benjelloun. Active XML: A data-centric perspective on Web services. Ph.D. Thesis, Universtity of Paris XI, 2002.

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

[5] Angela Bonifati and Stefano Ceri. Comparative analysis of five XML query languages. *SIGMOD Record*, 29(1):68–91, 2000.

[6] Allen Brown, Cosimo Laneve, and Greg Meredith. PiDuce: a process calculus with native XML datatypes. In *Proc. of WSFM'05*. LNCS, 2005.

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

[8] Giuseppe Castagna, Rocco De Nicola, and Daniele Varacca. Semantic subtyping for the pi-calculus. In *Proc. of LICS'05*, pages 92–101. IEEE Computer Society Press, 2005.

[9] Murdoch J. Gabbay. The π-calculus in FM. In *Thirty-five years of Automath*, volume 28, pages 71–123. Kluwer Academic Press, 2003.

[10] Murdoch J. Gabbay and Andrew M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing*, 13:341–363, 2002. Special issue in honour of Rod Burstall.

[11] Philippa Gardner and Sergio Maffeis. Modelling dynamic Web data. *Theoretical Computer Science*, 342:104–131, 2005.

[12] Matthew Hennessy, Massimo Merro, and Julian Rathke. Towards a behavioural theory of access and mobility control in distributed systems. *Theoretical Computer Science*, 322(3):615–669, 2004.

[13] Matthew Hennessy, Julian Rathke, and Nobuko Yoshida. safeDpi: A language for controlling mobile code. In *Proc. of FoSSaCS 2004*, volume 2987 of *LNCS*, pages 241–256. Springer Verlag, 2004.

[14] Matthew Hennessy and James Riely. Resource access control in systems of mobile agents. In *HLCL '98*, volume 16.3 of *ENTCS*, pages 3–17. Elsevier Science Publishers, 1998.

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

[16] Haruo Hosoya and Benjamin C. Pierce. Xduce: A statically typed xml processing language. *ACM Transactions on Internet Technology*, 3(2):117–148, 2003.

[17] Alan Jeffrey and Julian Rathke. Contextual equivalence for higher-order pi-calculus revisited. *Logical Methods in Computer Science*, 1(1:4), 2005.

[18] Sergio Maffeis. *Dynamic Web Data: A Process Algebraic Approach*. PhD thesis, Imperial College London, September 2005.

[19] Sergio Maffeis and Philippa Gardner. Behavioural equivalences for dynamic Web data. In *Proc. of Ifip WCC-TCS'04*, Kluwer Academic Press, pages 535–548, August 2004.

[20] Robin Milner. *Communication and concurrency*. Prentice-Hall, Inc., 1989.

[21] Robin Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, May 1999.

[22] 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.

[23] Robin Milner and Davide Sangiorgi. Techniques of weak bisimulation up-to. In *Proc. of CONCUR '92*, volume 2987 of *LNCS*, pages 41–77. Springer-Verlag, 1992.

[24] George Necula and Peter Lee. Safe, untrusted agents using proof-carrying code. In *Mobile Agents and Security*, pages 61–91. Springer Verlag, 1998.

[25] David Neven, Thomas Schwentick, and Dan Suciu. Foundations of semistructured data. Dagstuhl Seminar Proceedings 05061. Available online from `http://drops.dagstuhl.de/portals/05061/`, 2005.

[26] Benjamin C. Pierce and Davide Sangiorgi. Behavioral equivalence in the polymorphic pi-calculus. In *Proc. of POPL '97*, pages 242–255. ACM Press, 1997.

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

[28] Arnaud Sahuguet, Benjamin Pierce, and Val Tannen. Distributed Query Optimization: Can Mobile Agents Help? Unpublished draft, 2000.

[29] Arnaud Sahuguet and Val Tannen. ubql, a language for programming distributed query systems. In *Proc. of webDB'01*, pages 37–42, 2001.

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

[31] Davide Sangiorgi. A theory of bisimulation for the pi-calculus. In *Proc. of CONCUR '93*, pages 127–142, Springer Verlag, 1993.

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

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

[34] Tim Sheard and Simon Peyton Jones. Template meta-programming for haskell. In *Proc. of Haskell '02*, pages 1–16. ACM Press, 2002.

[35] Walid Taha. Metaocaml: A compiled, type-safe multi-stage programming language. Available online from http://www.cs.rice.edu/taha/MetaOCaml/, 2001.

[36] UDDI. Universal Description, Discovery, and Integration of Business for the Web (UDDI) 3.0. http://www.uddi.org, 2005.

[37] W3C. XML Path Language (XPath) Version 1.0. http://w3.org/TR/xpath, 1999.

[38] W3C. Extensible Markup Language (XML) 1.0 (2nd edition). http://www.w3.org/TR/REC-xml.html, 2000.

[39] W3C. Web Services Description Language (WSDL) 1.1. http://w3.org/TR/wsdl, 2001.

[40] W3C. Web Services Activity. http://www.w3.org/2002/ws, 2002.

[41] W3C. Simple Object Acess Protocol (SOAP) Version 1.2. http://w3.org/TR/SOAP, 2003.

[42] Nobuko Yoshida, Martin Berger, and Kohei Honda. Strong normalisation in the π-calculus. *Information and Computation*, 191(2):145–202, 2004.

A Figures

Figure 21: Full structural congruence

$(\nu\, c)\mathbf{0} \equiv \mathbf{0}$ (CStruct Res PNil)

$c \notin \mathit{fn}(\boldsymbol{P}) \Longrightarrow \boldsymbol{P} \mid (\nu\, c)\boldsymbol{Q} \equiv (\nu\, c)(\boldsymbol{P} \mid \boldsymbol{Q})$ (CStruct Res PPar)

$(\nu\, c)(\nu\, d)\boldsymbol{P} \equiv (\nu\, d)(\nu\, c)\boldsymbol{P}$ (CStruct Res PRes)

$\boldsymbol{P} \mid (\boldsymbol{Q} \mid \boldsymbol{Q}') \equiv (\boldsymbol{P} \mid \boldsymbol{Q}) \mid \boldsymbol{Q}'$ (CStruct Par Assoc)

$\boldsymbol{P} \equiv \boldsymbol{Q} \Longrightarrow \boldsymbol{Q} \equiv \boldsymbol{P}$ (CStruct Par Comm)

$\boldsymbol{P} \mid \mathbf{0} \equiv \boldsymbol{P}$ (CStruct Par Zero)

$\boldsymbol{P} \equiv \boldsymbol{Q} \Longrightarrow (\nu\, c)\boldsymbol{P} \equiv (\nu\, c)\boldsymbol{Q}$ (CStruct Cong Res)

$\boldsymbol{P} \equiv \boldsymbol{P}' \Longrightarrow \boldsymbol{P} \mid \boldsymbol{Q} \equiv \boldsymbol{P}' \mid \boldsymbol{Q}$ (CStruct Cong Par)

$\boldsymbol{P} \equiv \boldsymbol{Q} \Longrightarrow \boldsymbol{l}\cdot\boldsymbol{a}(\widetilde{\pi}).\boldsymbol{P} \equiv \boldsymbol{l}\cdot\boldsymbol{a}(\widetilde{\pi}).\boldsymbol{Q}$ (CStruct Cong In)

$\boldsymbol{P} \equiv \boldsymbol{Q} \Longrightarrow !\boldsymbol{l}\cdot\boldsymbol{a}(\widetilde{\pi}).\boldsymbol{P} \equiv !\boldsymbol{l}\cdot\boldsymbol{a}(\widetilde{\pi}).\boldsymbol{Q}$ (CStruct Cong !In)

$\boldsymbol{P} \equiv \boldsymbol{Q} \Longrightarrow \boldsymbol{l}\cdot\mathsf{go}\,\boldsymbol{m}.\boldsymbol{P} \equiv \boldsymbol{l}\cdot\mathsf{go}\,\boldsymbol{m}.\boldsymbol{Q}$ (CStruct Cong Go)

$P \equiv Q \Longrightarrow (D, P) \equiv (D, Q)$ (CStruct Proc)

$(\nu\, c)\mathbf{0} \equiv \mathbf{0}$ (Struct Res CNil)

$c \notin \mathit{fn}(K) \Longrightarrow K \mid (\nu\, c)K' \equiv (\nu\, c)(K \mid K')$ (Struct Res CPar)

$(\nu\, c)(\nu\, d)K \equiv (\nu\, d)(\nu\, c)K$ (Struct Res CRes)

$K \mid (K' \mid L) \equiv (K \mid K') \mid L$ (Struct Par CAssoc)

$K \equiv K' \Longrightarrow K' \equiv K$ (Struct Par CComm)

$K \mid \mathbf{0} \equiv K$ (Struct Par CZero)

$K \equiv K' \Longrightarrow (\nu\, c)K \equiv (\nu\, c)K'$ (Struct Cong CRes)

$K \equiv L \Longrightarrow K \mid K' \equiv L \mid K'$ (Struct Cong CPar)

$t \equiv t$ (CStruct Refl)

$t \equiv t' \Longrightarrow t' \equiv t$ (CStruct Symm)

$t \equiv t''$ *and* $t'' \equiv t' \Longrightarrow t \equiv t'$ (CStruct Trans)

Notation: t ranges over $\boldsymbol{P}$ or K.

Figure 22: Predicates wf, $\mathit{distinct}$ and functions dom, cval

$$\mathit{wf}(\mathbf{0}) \qquad \mathit{wf}(\overline{\boldsymbol{l}\cdot\boldsymbol{c}}\langle\widetilde{\boldsymbol{v}}\rangle) \qquad \frac{\mathit{wf}(\boldsymbol{P}) \quad \mathit{wf}(\boldsymbol{Q})}{\mathit{wf}(\boldsymbol{P} \mid \boldsymbol{Q})}$$

$$\frac{\mathit{wf}(\boldsymbol{P})}{\mathit{wf}((\nu\, c)\boldsymbol{P})} \qquad \frac{\mathit{dom}(\boldsymbol{P}) = \{\boldsymbol{l}\} \quad \mathit{wf}(\boldsymbol{P})}{\mathit{wf}(\boldsymbol{l}\cdot\boldsymbol{c}(\widetilde{\pi}).\boldsymbol{P})} \qquad \frac{\mathit{dom}(\boldsymbol{P}) = \{\boldsymbol{l}\} \quad \mathit{wf}(\boldsymbol{P})}{\mathit{wf}(!\boldsymbol{l}\cdot\boldsymbol{c}(\widetilde{\pi}).\boldsymbol{P})}$$

$$\frac{\mathit{dom}(\boldsymbol{P}) = \{\boldsymbol{m}\} \quad \mathit{wf}(\boldsymbol{P})}{\mathit{wf}(\boldsymbol{l}\cdot\mathsf{go}\ \boldsymbol{m}.\boldsymbol{m}\cdot\boldsymbol{P})} \qquad \mathit{wf}(\boldsymbol{A} \circ \langle \boldsymbol{l}, \widetilde{\boldsymbol{v}}\rangle) \qquad \mathit{wf}(\boldsymbol{l}\cdot\mathsf{req}_{\boldsymbol{p}}\langle\boldsymbol{c}\rangle)$$

$$\frac{\mathit{distinct}(\pi) \quad \mathit{distinct}(\widetilde{\pi}) \quad \mathit{fv}(\pi) \cap \mathit{fv}(\widetilde{\pi}) = \emptyset}{\mathit{distinct}(\pi, \widetilde{\pi})} \qquad \frac{\mathit{distinct}(\boldsymbol{A})}{\mathit{distinct}(\langle \boldsymbol{A}\rangle)}$$

$$\frac{\mathit{distinct}(\boldsymbol{E}) \quad \mathit{distinct}(\boldsymbol{T}) \quad \mathit{fv}(\boldsymbol{E}) \cap \mathit{fv}(\boldsymbol{T}) = \emptyset}{\mathit{distinct}(\boldsymbol{E}\triangleright\boldsymbol{T})} \qquad \frac{\mathit{distinct}(\boldsymbol{V})}{\mathit{distinct}(\mathsf{a}[\,\boldsymbol{V}\,])}$$

$$\mathit{distinct}(\varnothing) \qquad \mathit{distinct}(x) \qquad \frac{\mathit{fv}(\boldsymbol{l}) \cap \mathit{fv}(\boldsymbol{p}) = \emptyset}{\mathit{distinct}(\boldsymbol{p}@\boldsymbol{l})}$$

$$\mathit{dom}(D) = \text{domain of } D \qquad \mathit{dom}(\mathbf{0}) = \emptyset$$

$$\mathit{dom}(\boldsymbol{P} \mid \boldsymbol{Q}) = \mathit{dom}(\boldsymbol{P}) \cup \mathit{dom}(\boldsymbol{Q}) \qquad \mathit{dom}((\nu\, c)\boldsymbol{P}) = \mathit{dom}(\boldsymbol{P})$$

$$\mathit{dom}(\overline{\boldsymbol{l}\cdot\boldsymbol{c}}\langle\widetilde{\boldsymbol{v}}\rangle) = \{\boldsymbol{l}\} \qquad \mathit{dom}(\boldsymbol{l}\cdot\boldsymbol{c}(\widetilde{\pi}).\boldsymbol{P}) = \{\boldsymbol{l}\}$$

$$\mathit{dom}(!\boldsymbol{l}\cdot\boldsymbol{c}(\widetilde{\pi}).\boldsymbol{P}) = \{\boldsymbol{l}\} \qquad \mathit{dom}(\boldsymbol{l}\cdot\mathsf{go}\ \boldsymbol{m}.\boldsymbol{m}\cdot\boldsymbol{P}) = \{\boldsymbol{l}\}$$

$$\mathit{dom}(\boldsymbol{A} \circ \langle \boldsymbol{l}, \widetilde{\boldsymbol{v}}\rangle) = \{\boldsymbol{l}\} \qquad \mathit{dom}(\boldsymbol{l}\cdot\mathsf{req}_{\boldsymbol{p}}\langle\boldsymbol{c}\rangle) = \{\boldsymbol{l}\}$$

$$\mathit{dom}(-) = \emptyset \qquad \mathit{dom}(C_{\mathcal{S}}[-] \uplus D) = \mathit{dom}(D) \cup \mathit{dom}(C_{\mathcal{S}}[-])$$

$$\left.\begin{matrix}\mathit{dom}(P \mid C_{\mathcal{P}}[-]) \\ \mathit{dom}(C_{\mathcal{P}}[-] \mid P)\end{matrix}\right\} = \mathit{dom}(P) \cup \mathit{dom}(C_{\mathcal{P}}[-]) \qquad \mathit{dom}((\nu\, c)C_{\mathcal{P}}[-]) = \mathit{dom}(C_{\mathcal{P}}[-])$$

$$\frac{t \in \mathcal{A} \cup \mathcal{Q}}{\mathit{cval}(t) = \{t\}} \qquad \mathit{cval}(\boldsymbol{E}\triangleright\boldsymbol{T}) = \mathit{cval}(\boldsymbol{E}) \cup \mathit{cval}(\boldsymbol{T})$$

$$\mathit{cval}(\mathsf{a}[\,\boldsymbol{V}\,]) = \mathit{cval}(\boldsymbol{V}) \qquad \mathit{cval}(\langle\boldsymbol{A}\rangle) = \mathit{cval}(\boldsymbol{A})$$

$$\mathit{cval}(\boldsymbol{p}@\boldsymbol{l}) = \mathit{cval}(\boldsymbol{p}) \qquad \mathit{cval}(\varnothing) = \mathit{cval}(l) = \mathit{cval}(x) = \emptyset$$

Figure 23: Free variables and free names for Core X$d\pi$

$$
\begin{array}{rr}
\mathit{fv}((x,\widetilde{\pi})\boldsymbol{P}) = \mathit{fv}(\boldsymbol{P}) \setminus \mathit{fv}(x,\widetilde{\pi}) & \mathit{fv}(\boldsymbol{0}) = \emptyset \\
\mathit{fv}(\boldsymbol{P} \mid \boldsymbol{Q}) = \mathit{fv}(\boldsymbol{P}) \cup \mathit{fv}(\boldsymbol{Q}) & \mathit{fv}((\nu\, c)\boldsymbol{P}) = \mathit{fv}(\boldsymbol{P}) \\
\mathit{fv}(\boldsymbol{l}{\cdot}\boldsymbol{c}(\widetilde{\pi}).\boldsymbol{l}{\cdot}\boldsymbol{P}) = \mathit{fv}(\boldsymbol{l}) \cup \mathit{fv}(\boldsymbol{c}) \cup (\mathit{fv}(\boldsymbol{P}) \setminus \mathit{fv}(\widetilde{\pi})) & \mathit{fv}(\overline{\boldsymbol{l}{\cdot}\boldsymbol{c}}\langle\widetilde{\boldsymbol{v}}\rangle) = \mathit{fv}(\boldsymbol{l}) \cup \mathit{fv}(\boldsymbol{c}) \cup \mathit{fv}(\widetilde{\boldsymbol{v}}) \\
\mathit{fv}(!\boldsymbol{l}{\cdot}\boldsymbol{c}(\widetilde{\pi}).\boldsymbol{l}{\cdot}\boldsymbol{P}) = \mathit{fv}(\boldsymbol{l}) \cup \mathit{fv}(\boldsymbol{c}) \cup (\mathit{fv}(\boldsymbol{P}) \setminus \mathit{fv}(\widetilde{\pi})) & \mathit{fv}(\boldsymbol{l}{\cdot}\mathsf{go}\ \boldsymbol{m}.\boldsymbol{m}{\cdot}\boldsymbol{P}) = \mathit{fv}(\boldsymbol{l}) \cup \mathit{fv}(\boldsymbol{m}) \cup \mathit{fv}(\boldsymbol{P}) \\
\mathit{fv}(\boldsymbol{A} \circ \langle \boldsymbol{l}, \widetilde{\boldsymbol{v}}\rangle) = \mathit{fv}(\boldsymbol{A}) \cup \mathit{fv}(\boldsymbol{l}) \cup \mathit{fv}(\widetilde{\boldsymbol{v}}) & \mathit{fv}(\boldsymbol{l}{\cdot}\mathsf{req}_{\boldsymbol{p}}\langle\boldsymbol{c}\rangle) = \mathit{fv}(\boldsymbol{l}) \cup \mathit{fv}(\boldsymbol{p}) \cup \mathit{fv}(\boldsymbol{c})
\end{array}
$$

$$
\begin{array}{rr}
\mathit{fn}(\boldsymbol{0}) = \emptyset & \mathit{fn}(\boldsymbol{P} \mid \boldsymbol{Q}) = \mathit{fn}(\boldsymbol{P}) \cup \mathit{fn}(\boldsymbol{Q}) \\
\mathit{fn}((\nu\, c)\boldsymbol{P}) = \mathit{fn}(\boldsymbol{P}) \setminus \{c\} & \mathit{fn}(\overline{\boldsymbol{l}{\cdot}\boldsymbol{c}}\langle\widetilde{\boldsymbol{v}}\rangle) = \mathit{fn}(\boldsymbol{c}) \cup \mathit{fn}(\widetilde{\boldsymbol{v}}) \\
\mathit{fn}(\boldsymbol{l}{\cdot}\boldsymbol{c}(\widetilde{\pi}).\boldsymbol{P}) = \mathit{fn}(\boldsymbol{c}) \cup \mathit{fn}(\boldsymbol{P}) & \mathit{fn}(!\boldsymbol{l}{\cdot}\boldsymbol{c}(\widetilde{\pi}).\boldsymbol{P}) = \mathit{fn}(\boldsymbol{c}) \cup \mathit{fn}(\boldsymbol{P}) \\
\mathit{fn}(\boldsymbol{l}{\cdot}\mathsf{go}\ \boldsymbol{m}.\boldsymbol{m}{\cdot}\boldsymbol{P}) = \mathit{fn}(\boldsymbol{P}) & \mathit{fn}(\boldsymbol{A} \circ \langle \boldsymbol{l}, \widetilde{\boldsymbol{v}}\rangle) = \mathit{fn}(\widetilde{\boldsymbol{v}}) \\
\mathit{fn}(\boldsymbol{l}{\cdot}\mathsf{req}_{\boldsymbol{p}}\langle\boldsymbol{c}\rangle) = \mathit{fn}(\boldsymbol{c}) & \mathit{fn}((D, P)) = \mathit{fn}(P)
\end{array}
$$

$$
\begin{array}{rr}
\mathit{fv}(\boldsymbol{K} \mid \boldsymbol{K}') = \mathit{fv}(\boldsymbol{K}) \cup \mathit{fv}(\boldsymbol{K}') & \mathit{fv}((\nu\, c)\boldsymbol{K}) = \mathit{fv}(\boldsymbol{K}) \\
\mathit{fv}(\langle k \Leftarrow A\rangle) = \emptyset & \mathit{fv}(k) = \emptyset
\end{array}
$$

$$
\begin{array}{rr}
\mathit{fn}(\boldsymbol{K} \mid \boldsymbol{K}') = \mathit{fn}(\boldsymbol{K}) \cup \mathit{fn}(\boldsymbol{K}') & \mathit{fn}((\nu\, c)\boldsymbol{K}) = \mathit{fn}(\boldsymbol{K}) \\
\mathit{fn}(\langle k \Leftarrow A\rangle) = \{k\} & \mathit{fn}(k) = \{k\}
\end{array}
$$

Figure 24: Functions *dom* and *scripts* for configurations

$$
\mathit{dom}(\boldsymbol{K} \mid \boldsymbol{K}') = \mathit{dom}(\boldsymbol{K}) \cup \mathit{dom}(\boldsymbol{K}') \quad \mathit{dom}((\nu\, c)\boldsymbol{K}) = \mathit{dom}(\boldsymbol{K}) \quad \mathit{dom}(\langle k \Leftarrow A\rangle) = \emptyset
$$

$$
\begin{array}{rr}
\mathit{scripts}(v, \widetilde{v}) = \mathit{scripts}(v) \cup \mathit{scripts}(\widetilde{v}) & \mathit{scripts}(\{^{v}/_{x}\}) = \mathit{scripts}(v) \\
\mathit{scripts}(E \triangleright T) = \mathit{scripts}(E) \cup \mathit{scripts}(T) & \mathit{scripts}(\varnothing) = \emptyset \\
\mathit{scripts}(\mathsf{a}[\,V\,]) = \mathit{scripts}(V) & \mathit{scripts}(\langle A\rangle) = \{A\} \\
\mathit{scripts}(A) = \{A\} & \mathit{scripts}(c) = \mathit{scripts}(\mathsf{c}) = \emptyset \\
\mathit{scripts}(p@l) = \mathit{scripts}(p) & \mathit{scripts}(l) = \emptyset
\end{array}
$$

Assumption: *scripts* on queries is given as part of the query language definition.

B Results and proofs

This section gives the formal proofs of the properties of domain bisimilarity.

To follow more easily certain common steps in the proofs, it may be helpful to keep in mind that: private and service channel names are distinct; a script is well-formed only if it has no free private channel or trigger names; configurations are well-formed if for any trigger name there is at most one definition.

B.1 Basic properties

We study some basic properties of domain bisimilarity which will be useful to prove the main results of congruence and soundness. A first property is that structural congruence preserves bisimilarity. We will use this implicitly in the rest of the section.

Proposition B.1 (Bisimilarity Up-To Structural Congruence) *If $K \approx_\Lambda L$, $K \equiv K'$ and $L \equiv L'$, then $K' \approx_\Lambda L'$.*

Proof. The family of relations $\dot{\approx}$ with generic element

$$\dot{\approx}_\Delta = \{(K', L') \;:\; K' \equiv K \approx_\Delta L \equiv L'\}$$

is a domain bisimulation. Follows by using rule (Lts Struct). □

By definition of bisimilarity, the smaller the domain Λ, the less likely that two processes are bisimilar. In fact, we need to check for matching actions first in Λ, then in any Λ' containing Λ. The underlying intuition is that if we can rely on a larger set of locations to be connected to the network, then we can perform more optimizations.

Proposition B.2 (Monotonicity) *Domain bisimilarity is monotonic: for all sets of locations Λ, Λ', if $\Lambda \subsetneq \Lambda'$ then $\approx_\Lambda \subsetneq \approx_{\Lambda'}$.*

Proof.

(⊆) Follows by Definition 3.14, noticing that using rule (Lts In) it is always possible to make an input action at location l, for any l not in Λ.

(⊊) If $\Lambda \subsetneq \Lambda'$ then there exists an m such that $m \in \Lambda' \setminus \Lambda$. Consider the two processes $P = \overline{l{\cdot}a}$ and $Q = l{\cdot}\mathsf{go}\, m.m{\cdot}\overline{l{\cdot}a}$, where $l \neq m$. Clearly, $P \not\approx_\Lambda Q$ because $P \xrightarrow{\overline{l{\cdot}a}} \mathbf{0}$ but there is no Q' such that $Q \overset{\overline{l{\cdot}a}}{\twoheadrightarrow}_\Lambda Q'$. To show that $P \approx_{\Lambda'} Q$, let $\mathcal{R}_\Delta$ be the set containing the pairs

$$(M \mid P, M \mid Q), \quad (M \mid P, M \mid m{\cdot}\overline{l{\cdot}a}), \quad (M, M)$$

for any M of the form

$$\prod_{0 \le i \le n} \overline{l_i{\cdot}c_i}\langle \widetilde{v}_i \rangle,\ \mathit{dom}(M) \subseteq \Delta$$

where $\mathit{scripts}(\widetilde{v}_i) = \emptyset$ for all i. The family $\dot{\approx}$, where $\dot{\approx}_\Delta = \mathcal{R}_\Delta \cup (\mathcal{R}_\Delta)^{-1}$ for each Δ containing Λ' and $\dot{\approx}_\Delta = \emptyset$ otherwise, is a domain bisimulation containing (P, Q), hence $P \approx_{\Lambda'} Q$.

□

The lemma given below is a standard technical lemma relating the transitions in the lts with the syntactic structure of configurations, up-to structural congruence. It is used in many proofs, sometimes implicitly.

Lemma B.3 (Transition Correspondence) *The transitions of the lts are in close correspondence with the structure of configurations.*

1. $K \xrightarrow{(\widetilde{a},\widetilde{k})\overline{l\cdot c}\langle\widetilde{v}\rangle} K'$ *if and only if* $K \equiv (\nu\,\widetilde{a})(L \mid \overline{l\cdot c}\langle\widetilde{v'}\rangle)$ *where* $c \notin \{\widetilde{a}\}$, $\{\widetilde{a}\} \subseteq \mathit{fn}(\widetilde{v'})$ *and* $K' \equiv L \mid \Theta^{\widetilde{k}}$ *where* $\mathfrak{X}(\widetilde{v'}) = (\widetilde{v}; \Theta^{\widetilde{k}})$.

2. $K \xrightarrow{l\cdot c(\widetilde{v})} K'$ *if and only if* $K' \equiv K \mid \overline{l\cdot c}\langle\widetilde{v}\rangle$, $\mathit{scripts}(\widetilde{v}) = \emptyset$ *and* $\mathit{rel}(l\cdot c(\widetilde{v}), K)$.

3. $K \xrightarrow{l\cdot\tau} K'$ *if and only if*
   - $K \equiv (\nu\,\widetilde{a})(L \mid l\cdot c(\widetilde{\pi}).\boldsymbol{P} \mid \overline{l\cdot c}\langle\widetilde{\pi}\sigma\rangle)$ *and* $K' \equiv (\nu\,\widetilde{a})(L \mid \boldsymbol{P}\sigma)$, *or*
   - $K \equiv (\nu\,\widetilde{a})(L \mid !l\cdot c(\widetilde{\pi}).\boldsymbol{P} \mid \overline{l\cdot c}\langle\widetilde{\pi}\sigma\rangle)$ *and* $K' \equiv (\nu\,\widetilde{a})(L \mid !l\cdot c(\widetilde{\pi}).\boldsymbol{P} \mid \boldsymbol{P}\sigma)$, *or*
   - $K \equiv (\nu\,\widetilde{a})(L \mid (x, \widetilde{\pi})\boldsymbol{P} \circ \langle l, \widetilde{\pi}\sigma\rangle)$ *and* $K' \equiv (\nu\,\widetilde{a})(L \mid \boldsymbol{P}\{^{l}/_{x}\}\sigma)$, *or*
   - $K \equiv (\nu\,\widetilde{a})(L \mid m\cdot\mathsf{go}\ l.P)$ *and* $K' \equiv (\nu\,\widetilde{a})(L \mid P)$.

4. $K \xrightarrow{(\widetilde{k})l\cdot\mathsf{req}\langle p\rangle(T)} K'$ *if and only if* $K \equiv (\nu\,a)(L \mid l\cdot\mathsf{req}_{p'}\langle c\rangle)$ *and* $K' \equiv (\nu\,a)(L \mid \overline{l\cdot c}\langle T\rangle \mid \Theta^{\widetilde{k}})$ *for some* p' *such that* $\mathfrak{X}_{\mathcal{Q}}(p') = (p, \Theta^{\widetilde{k}})$ *and some* T *such that* $\mathit{scripts}(T) = \emptyset$ *and* T *has the form* $\mathtt{r}[\,U_1\,]_{\mathtt{l}} \ldots \mathtt{r}[\,U_n\,]_{\mathtt{l}}\varnothing$.

5. $K \xrightarrow{l\cdot k(\widetilde{\pi}\sigma)} K'$ *if and only if* $K \equiv L \mid \langle k \Leftarrow (x, \widetilde{\pi})\boldsymbol{P}\rangle$ *and* $K' \equiv K \mid \langle k \Leftarrow (x, \widetilde{\pi})\boldsymbol{P}\rangle \mid \boldsymbol{P}\{^{l}/_{x}\}\sigma$ *and* $\mathit{scripts}(\sigma) = \emptyset$.

6. $K \xrightarrow{(\widetilde{a},\widetilde{k})\overline{l\cdot j}\langle\widetilde{v}\rangle} K'$ *if and only if* $K \equiv (\nu\,\widetilde{a})(L \mid j \circ \langle l, \widetilde{v'}\rangle)$ *where* $j \notin \{\widetilde{k}\}$, $\{\widetilde{a}\} \subseteq \mathit{fn}(\widetilde{v'})$ *and* $K' \equiv L \mid \Theta^{\widetilde{k}}$ *where* $\mathfrak{X}(\widetilde{v'}) = (\widetilde{v}; \Theta^{\widetilde{k}})$.

Proof.

($\Longleftarrow$) Follows easily by definition of lts.

($\Longrightarrow$) By induction on the depth n of the derivation tree in the premise for the labelled transition. We give the case for bound output as an example (point 1).

($n = 0$) Suppose $K \xrightarrow{(\widetilde{a},\widetilde{k})\overline{l{\cdot}c}\langle\widetilde{v}\rangle} K'$ is derived by directly applying (Lts Out). It must be the case that $K = \overline{l{\cdot}c}\langle\widetilde{v'}\rangle$, where $\mathfrak{X}(\widetilde{v'}) = (\widetilde{v}; \Theta^{\widetilde{k}})$ and $K' = \Theta^{\widetilde{k}}$.

($n = m + 1$) A derivation of depth $m+1$ for $K \xrightarrow{(\widetilde{a},\widetilde{k})\overline{l{\cdot}c}\langle\widetilde{v}\rangle} K'$ must be obtained by applying one of the rules (Lts Res), (Lts Par), (Lts Struct) or (Lts Open) to a derivation of depth m. The case for (Struct) is trivial. If rule (Res) is applied then we must have that $(\nu\, d)K_1 \xrightarrow{(\widetilde{a},\widetilde{k})\overline{l{\cdot}c}\langle\widetilde{v}\rangle} (\nu\, d)K_1'$, where $d \notin n((\widetilde{a},\widetilde{k})\overline{l{\cdot}c}\langle\widetilde{v}\rangle)$, follows from the premise $K_1 \xrightarrow{(\widetilde{a},\widetilde{k})\overline{l{\cdot}c}\langle\widetilde{v}\rangle} K_1'$. By inductive hypothesis, $K_1 \equiv (\nu\,\widetilde{a})(L \mid \overline{l{\cdot}c}\langle\widetilde{v'}\rangle)$ where $c \notin \{\widetilde{a}\}$, $\{\widetilde{a}\} \subseteq fn(\widetilde{v'})$ and $K_1' \equiv L \mid \Theta^{\widetilde{k}}$ where $\mathfrak{X}(\widetilde{v'}) = (\widetilde{v}; \Theta^{\widetilde{k}})$. Since $\mathfrak{X}$ does not affect session channels, $d \notin fn(\widetilde{v'}) \setminus \{\widetilde{a}\}$. By structural congruence, $(\nu\, d)K_1 \equiv (\nu\,\widetilde{a})((\nu\, d)(L) \mid \overline{l{\cdot}c}\langle\widetilde{v'}\rangle)$, and $(\nu\, d)K_1' \equiv (\nu\, d)(L) \mid \Theta^{\widetilde{k}}$. The cases for (Par) and (Open) are similar.

□

The next step towards the main proofs consists of generalizing the variant lemma of [15] to bijective substitutions (here called *switchings*) on channel and trigger names.[14] By using switchings rather than generic substitutions we obtain a purely coinductive proof. Below we let a, b, c range over channel or trigger names, and we consider only well-sorted substitutions (replacing channels for channels and triggers for triggers).

Definition B.4 (Switching) *Given a term t with a function fn returning its free names, a* switching $^{a\frown b}$ *is a bijective substitution $\{^c/_a, {}^a/_b\}\{^b/_c\}$ such that $c \notin fn(t) \cup \{a, b\}$. We denote by $^{\widetilde{a}\frown\widetilde{b}}$ the switching $^{a_1\frown b_1} \ldots {}^{a_n\frown b_n}$ where both $\widetilde{a}$ and $\widetilde{b}$ are vectors of distinct names.*

Observation B.5 (Switching Properties) *Switching is self-dual $K = (K^{a\frown b})^{a\frown b}$ and symmetric $K^{a\frown b} = K^{b\frown a}$.*

Proof. Follows from the definition of switching and substitution. □

We note below that both the extraction and the transition relations do not depend on specific names, hence they are fully compatible with switching, and α-conversion.

Lemma B.6 (Switching Extraction) *Extraction preserves switching: if $\mathfrak{X}(\widetilde{v}) = (\widetilde{v'}, \Theta^{\widetilde{k}})$ then $\mathfrak{X}(\widetilde{v}^{a\frown b}) = (\widetilde{v'}^{a\frown b}; (\Theta^{\widetilde{k}})^{a\frown b})$.*

[14] Our approach is reminiscent of the permutation-based approach to abstract syntax developed by Gabbay and Pitts [10]. In particular, it may be interesting in future work to compare our use of switchings with the work of Gabbay [9] on the π-calculus.

Proof. By a simple induction on the derivation of $\mathfrak{X}(\widetilde{v}) = (\widetilde{v'}, \Theta^{\widetilde{k}})$, where the base case for queries uses Definition 3.9. □

Lemma B.7 (Switching Transitions) *If* $K \xrightarrow{\alpha_l} K'$ *then* $K^{a\frown b} \xrightarrow{\alpha_l^{a\frown b}} K'^{a\frown b}$, *provided that* $\mathit{bn}(\alpha_l) \cap \mathit{fn}(K^{a\frown b}) \cap \{a, b\} = \emptyset$.

Proof. By case analysis on α_l, using Lemma B.3. We show the case for the bound output when the name of the channel used for output occurs in the switching; the other cases are simpler. Let $\rho = {}^{a\frown b}$. Without loss of generality, suppose that $K \xrightarrow{\alpha_l} K'$ where $\alpha_l = (b, \widetilde{b})\overline{l{\cdot}a}\langle\widetilde{v'}\rangle$ and b does not occur in $\widetilde{b}$. By Lemma B.3, $K \equiv (\nu\, b, \widetilde{b})(L \mid \overline{l{\cdot}a}\langle\widetilde{v}\rangle)$ where $a \notin \left\{b, \widetilde{b}\right\}$, $\left\{b, \widetilde{b}\right\} \subseteq \mathit{fn}(\widetilde{v})$ and $K' \equiv L \mid \Theta^{\widetilde{k}}$ where $\mathfrak{X}(\widetilde{v}) = (\widetilde{v'}; \Theta^{\widetilde{k}})$. Let c be a fresh name. By α-conversion and Observation B.5, we have $K \equiv (\nu\, c, \widetilde{b})(L\{{}^{c}/_{b}\} \mid \overline{l{\cdot}a}\langle\widetilde{v}\{{}^{c}/_{b}\}\rangle)\rho\rho$. Applying the inner switching, we obtain $K \equiv (\nu\, c, \widetilde{b})(L\{{}^{c}/_{b}\}\rho \mid \overline{l{\cdot}b}\langle\widetilde{v}\{{}^{c}/_{b}\}\rho\rangle)\rho$. Since $\{{}^{c}/_{b}\}$ has replaced b with c,

$$K \equiv (\nu\, c, \widetilde{b})(L\{{}^{c}/_{b}\}\{{}^{b}/_{a}\} \mid \overline{l{\cdot}b}\langle\widetilde{v}\{{}^{c}/_{b}\}\{{}^{b}/_{a}\}\rangle)\rho$$

Since a does not appear free anymore, we can alpha-convert c with a in the term above, obtaining

$$K \equiv (\nu\, a, \widetilde{b})(L\{{}^{c}/_{b}\}\{{}^{b}/_{a}\}\{{}^{a}/_{c}\} \mid \overline{l{\cdot}b}\langle\widetilde{v}\{{}^{c}/_{b}\}\{{}^{b}/_{a}\}\{{}^{a}/_{c}\}\rangle)\rho$$

By definition of switching, $K \equiv (\nu\, a, \widetilde{b})(L\rho \mid \overline{l{\cdot}b}\langle\widetilde{v}\rho\rangle)\rho$. By Observation B.5, $K\rho \equiv (\nu\, a, \widetilde{b})(L\rho \mid \overline{l{\cdot}b}\langle\widetilde{v}\rho\rangle\rho)$. By Lemma B.6, $\mathfrak{X}(\widetilde{v}\rho) = (\widetilde{v'}\rho; \Theta^{\widetilde{k}}\rho)$. By (Lts Out), $K\rho \xrightarrow{\alpha_l\rho} K'\rho$. □

The lemma below shows that bisimilarity is closed with respect to switchings, a property needed to show that it is transitive.

Lemma B.8 (Variant) *(i) If* $K \approx_\Lambda L$ *then* $K^{a\frown b} \approx_\Lambda L^{a\frown b}$. *(ii) If* $b \notin \mathit{fn}(K, L)$ *then* $K \approx_\Lambda L \Longrightarrow K\{{}^{b}/_{a}\} \approx_\Lambda L\{{}^{b}/_{a}\}$.

Proof. (i) Let $\rho = {}^{a\frown b}$. We show that the family $\dot{\approx}$ with generic element $\dot{\approx}_\Delta = \{(K\rho, L\rho) \;:\; K \approx_\Delta L\}$ is a domain bisimulation. Assume $K \approx_\Delta L$ for some Δ. Suppose $K\rho \xrightarrow{\alpha_l} K'$ and $l \notin \Delta$. By $K \approx_\Delta L$, $K \approx_{\Delta\cup\{l\}} L$, hence $(K\rho, L\rho) \in \dot{\approx}_{\Delta\cup\{l\}}$. Suppose instead $l \in \Delta$ and $\mathit{rel}(\alpha_l, L\rho)$. By Lemma B.7, $K \xrightarrow{\alpha_l\rho} K'\rho$ with $\mathit{rel}(\alpha_l\rho, L)$. By bisimilarity, $L \xrightarrow{\alpha_l\rho}\!\!\!\!\rightarrow_\Delta L'$ with $K'\rho \approx_\Delta L'$. By Lemma B.7, $L\rho \xrightarrow{\alpha_l}\!\!\!\!\rightarrow_\Delta L'\rho$. By definition, $(K'\rho\rho, L'\rho) \in \dot{\approx}_\Delta$. We conclude because, by Observation B.5, $K'\rho\rho = K'$. (ii) Follows from (i) by definition of switching. □

Domain bisimilarity is an equivalence relation. This property is very important, because in the rest of the proofs in this section we will often rely on symmetry and transitivity.

Proposition B.9 (Equivalence) *Domain bisimilarity is an equivalence relation.*

Proof. Reflexivity and symmetry are immediate. Transitivity states that if $K \approx_\Lambda M$ and $M \approx_\Lambda L$ then $K \approx_\Lambda L$. We show that the family $\dot{\approx}$ with generic element

$$\dot{\approx}_\Delta = \{(K, L) \ : \ K \approx_\Delta M,\, M \approx_\Delta L\}$$

is a domain bisimulation. Let Δ be arbitrary and suppose $K \xrightarrow{\alpha_l} K'$ with $l \notin \Delta$. By Definition 3.14, $K \approx_{\Delta\cup\{l\}} M$ and $M \approx_{\Delta\cup\{l\}} L$, hence $(K, L) \in \dot{\approx}_{\Delta\cup\{l\}}$. If $l \in \Delta$ and $\mathit{rel}(\alpha_l, L)$ then there are two cases, determined by the relevance of α_l to M. If $\mathit{rel}(\alpha_l, M)$, the proof is straightforward. If α_l is not relevant to M, the action α_l must necessarily have some bound names $\widetilde{c}$ such that $\{\widetilde{c}\} \subseteq \mathit{fn}(M)$. By the second premise of (Lts Par) used to derive the bound transition, $\{\widetilde{c}\} \cap \mathit{fn}(K) = \emptyset$. Let $\widetilde{a}$ have the same length as $\widetilde{c}$, and be such that $\{\widetilde{a}\} \cap \mathit{fn}(K, L, M) = \emptyset$. By Lemma B.8, $K = K\{\widetilde{a}/\widetilde{c}\} \approx_\Delta M\{\widetilde{a}/\widetilde{c}\} = M'$. By the same argument, $M' \approx_\Delta L$. Since now $\mathit{rel}(\alpha_l, M')$, the proof is straightforward. □

B.2 Congruence

Our next objective is to show that domain bisimilarity is a congruence. We already know that it is an equivalence relation (Proposition B.9), and that it preserves switchings (Lemma B.8). We also know how to relate labelled transitions with the syntactic structure of configurations (Lemma B.3). Using these tools, it is pretty easy to show that bisimilarity is closed under the restriction operator and, for processes, under prefixes. Most of the work in this section is dedicated to show *directly* closure under parallel composition. In contrast, Jeffrey and Rathke [17] for example show the soundness of their bisimilarity with respect to barbed congruence by using an auxiliary reduction-closed relation, which is closed under parallel composition. Showing the corresponding completeness result, they derive that also bisimilarity is closed under parallel composition. We cannot use their approach due to the inherent incompleteness of domain bisimilarity (see Section 3.2.2).

Lemma B.10 (Restriction) *Bisimilarity is closed under restriction: $K \approx_\Lambda L \implies (\nu\,\widetilde{c})K \approx_\Lambda (\nu\,\widetilde{c})L$.*

Proof. The family $\dot{\approx}$ with generic element

$$\dot{\approx}_\Delta = \{(K_1, L_1) \ : \ K_1 \equiv (\nu\,\widetilde{c})K,\, L_1 \equiv (\nu\,\widetilde{c})L,\, K \approx_\Delta L\}$$

is a domain bisimulation. Suppose $K_1 \xrightarrow{\alpha_l} K_1'$. The proof is by cases on α_l using Lemma B.3. We show the case for $\alpha_l = l{\cdot}c(\widetilde{v})$ which is the most interesting. If $l \notin \Delta$, the proof is easy. Suppose $l \in \Delta$. By Lemma B.3, $K_1' \equiv K_1 \,|\, \overline{l{\cdot}c}\langle\widetilde{v}\rangle$. By hypothesis, $K_1 \equiv (\nu\,\widetilde{c})K$, $K \approx_\Delta L$ and $L_1 \equiv (\nu\,\widetilde{c})L$. By α-conversion, $K_1 \equiv$

Figure 25: Merge operator for configurations

$$\langle\!\langle \boldsymbol{P} \rangle\!\rangle = \boldsymbol{P} \qquad \text{(Merge Proc)}$$
$$\langle\!\langle (\nu\, c)K \rangle\!\rangle = (\nu c)\langle\!\langle K \rangle\!\rangle \qquad \text{(Merge Res)}$$
$$\langle\!\langle K \mid \langle k \Leftarrow A \rangle \rangle\!\rangle = \langle\!\langle K^{\langle k \Leftarrow A \rangle} \rangle\!\rangle \qquad \text{(Merge Def)}$$

$(\nu\, \widetilde{c'})K\{\widetilde{c'}/\widetilde{c}\}$ for a fresh tuple of names $\widetilde{c'}$. By (Lts Struct), (Lts Par) and (Lts In), $(\nu\, \widetilde{c'})K\{\widetilde{c'}/\widetilde{c}\} \xrightarrow{l\cdot c(\widetilde{v})} (\nu\, \widetilde{c'})(K\{\widetilde{c'}/\widetilde{c}\}) \mid \overline{l\cdot c}\langle\widetilde{v}\rangle$ and $K\{\widetilde{c'}/\widetilde{c}\} \xrightarrow{l\cdot c(\widetilde{v})} K\{\widetilde{c'}/\widetilde{c}\} \mid \overline{l\cdot c}\langle\widetilde{v}\rangle$. By Lemma B.8, $K\{\widetilde{c'}/\widetilde{c}\} \approx_\Delta L\{\widetilde{c'}/\widetilde{c}\}$, hence $L\{\widetilde{c'}/\widetilde{c}\} \xrightarrow{l\cdot c(\widetilde{v})}\!\!\!\!\rightarrow_\Delta L' \approx_\Delta K\{\widetilde{c'}/\widetilde{c}\} \mid \overline{l\cdot c}\langle\widetilde{v}\rangle$. By (Lts Res) and freshness of $\widetilde{c'}$, $(\nu\, \widetilde{c'})(L\{\widetilde{c'}/\widetilde{c}\}) \xrightarrow{l\cdot c(\widetilde{v})}\!\!\!\!\rightarrow_\Delta (\nu\, \widetilde{c'})L'$. By (Lts Struct), $(\nu\, \widetilde{c})L \xrightarrow{l\cdot c(\widetilde{v})}\!\!\!\!\rightarrow_\Delta L'_1 \equiv (\nu\, \widetilde{c'})L'$. By α-conversion and freshness of $\widetilde{c'}$, $K'_1 \equiv (\nu\, c')(K\{c'/c\} \mid \overline{l\cdot c}\langle\widetilde{v}\rangle)$, hence $K'_1 \dot{\approx}_\Delta L'_1$. □

Following Jeffrey and Rathke [17], we define in Figure 25 a *merge* operator $\langle\!\langle - \rangle\!\rangle$ to reconstruct processes from configurations. The merge operator of [17] though is partial, due to a potential circularity of references between trigger names and definitions. Since scripted processes in definitions cannot contain triggers, our merge operator is total. This operator plays a substantial rôle in showing that $\approx_\Lambda$ is closed under parallel composition.

Before showing the properties of the merge operator, we illustrate three simple properties of the extraction function: it does not remove trigger names; it associates a definition to each trigger name it introduces; and we can recover the initial term by substituting the new definitions in the result term.

Lemma B.11 (Extraction Properties) *Suppose* $\mathfrak{X}(\widetilde{v}) = (\widetilde{v'}; \Theta^{\widetilde{k}})$, *The following properties hold:*

1. *if* $k \in \mathit{fn}(\widetilde{v})$ *then* $k \in \mathit{fn}(\widetilde{v'})$;
2. *if* $k \in \mathit{fn}(\widetilde{v'}) \setminus \{\widetilde{k}\}$ *then* $k \in \mathit{fn}(\widetilde{v})$;
3. $\widetilde{v} = \widetilde{v'}^{\Theta^{\widetilde{k}}}$.

Proof. By induction on the structure of $\widetilde{v}$, using Definition 3.9. □

Since definitions can appear only at the top level and scripts cannot contain free private channel names, we can always use structural equivalence to factor any configuration into the parallel composition of a process and a group of definitions. We will make substantial use of this property to show closure under parallel composition of $\approx_\Lambda$. Merging a configuration corresponds to substituting the script in each definition for the corresponding trigger names in the process

term of the configuration. Hence, the merge operator preserves the transitions that do not involve triggers names for which there is a corresponding definition. Moreover, if two configurations are bisimilar, then they must define the same trigger names.

Lemma B.12 (Merge Properties) *The merge operator satisfies the following properties:*

1. *factorization: for any well-formed K, there exist a process P and a configuration $\Theta^{\widetilde{k}}$ such that $K \equiv P \,|\, \Theta^{\widetilde{k}}$ and $\langle\!\langle K \rangle\!\rangle \equiv P^{\Theta^{\widetilde{k}}}$;*
2. *transition preservation: for any $\Theta^{\widetilde{k}}, \Theta^{\widetilde{j}}$, if $P \xrightarrow{\alpha_l} P'$ and $\left\{\widetilde{k}, \widetilde{j}\right\} \cap n(\alpha_l) = \emptyset$ then $\langle\!\langle P \,|\, \Theta^{\widetilde{k}} \rangle\!\rangle \,|\, \Theta^{\widetilde{j}} \xrightarrow{\alpha_l} \langle\!\langle P' \,|\, \Theta^{\widetilde{k}} \rangle\!\rangle \,|\, \Theta^{\widetilde{j}}$;*
3. *If $K \approx_\Lambda L$ then $K \equiv P \,|\, \Theta^{\widetilde{k}}$ and $L \equiv Q \,|\, \Omega^{\widetilde{k}}$, and each pair of corresponding definitions contains scripts with the same patterns: that is, $\langle k \Leftarrow (\widetilde{\pi}) P_k \rangle$ in $\Theta^{\widetilde{k}}$ implies $\langle k \Leftarrow (\widetilde{\pi}) Q_k \rangle$ in $\Omega^{\widetilde{k}}$, and viceversa.*

Proof.

1. By induction on the structure of K.
2. Follows from syntactic reasoning using Lemma B.3. We show the case for request transitions, the other ones are similar. If $P \xrightarrow{(\widetilde{h}) l \cdot \mathsf{req} \langle p \rangle (T)} P'$, by point(4) of Lemma B.3, $P \equiv (\nu\, a)(Q \,|\, l \cdot \mathsf{req}_{p'} \langle c \rangle)$ and $P' \equiv (\nu\, a)(Q \,|\, \overline{l \cdot c} \langle T \rangle \,|\, \Theta^{\widetilde{h}})$ for some p' such that $\mathfrak{X}_{\mathcal{Q}}(p') = (p, \Theta^{\widetilde{h}})$ and some appropriate T. By hypothesis, $\left\{\widetilde{k}, \widetilde{j}\right\} \cap n(\alpha_l) = \emptyset$, hence $\left\{\widetilde{k}, \widetilde{j}\right\} \cap fn(p) = \emptyset$. By point (1) of Lemma B.11, $\left\{\widetilde{k}, \widetilde{j}\right\} \cap fn(p') = \emptyset$. Since definitions do not contain free trigger names, $\left\{\widetilde{k}, \widetilde{j}\right\} \cap fn(\Theta^{\widetilde{h}}) = \emptyset$. By definition of merge, $\langle\!\langle P \,|\, \Theta^{\widetilde{k}} \rangle\!\rangle \equiv (\nu\, a)(\langle\!\langle Q \,|\, \Theta^{\widetilde{k}} \rangle\!\rangle \,|\, l \cdot \mathsf{req}_{p'} \langle c \rangle)$ and $\langle\!\langle P' \,|\, \Theta^{\widetilde{k}} \rangle\!\rangle \equiv (\nu\, a)(\langle\!\langle Q \,|\, \Theta^{\widetilde{k}} \rangle\!\rangle \,|\, \overline{l \cdot c} \langle T \rangle \,|\, \Theta^{\widetilde{h}})$. By definition of lts, $\langle\!\langle P \,|\, \Theta^{\widetilde{k}} \rangle\!\rangle \xrightarrow{(\widetilde{h}) l \cdot \mathsf{req} \langle p \rangle (T)} \langle\!\langle P' \,|\, \Theta^{\widetilde{k}} \rangle\!\rangle$ and $\langle\!\langle P \,|\, \Theta^{\widetilde{k}} \rangle\!\rangle \,|\, \Theta^{\widetilde{j}} \xrightarrow{(\widetilde{h}) l \cdot \mathsf{req} \langle p \rangle (T)} \langle\!\langle P' \,|\, \Theta^{\widetilde{k}} \rangle\!\rangle \,|\, \Theta^{\widetilde{j}}$.
3. By point (1) above, $K \equiv P \,|\, \Theta^{\widetilde{k}}$ and $L \equiv Q \,|\, \Theta^{\widetilde{j}}$. The argument is by contradiction. Suppose that $\Theta^{\widetilde{k}} \equiv \Theta^{\widetilde{h}} \,|\, \langle k \Leftarrow (\widetilde{\pi}) P_k \rangle$ and $k \notin \widetilde{j}$. By definition of lts, $K \xrightarrow{l \cdot k (\widetilde{\pi} \sigma)} K'$ but, since $k \notin \widetilde{j}$, it is not possible to derive a corresponding (weak) transition for L, which contradicts $K \approx_\Lambda L$. The case for a different pattern π is analogous.

□

The lemma below analyzes the relationship between bisimilarity and definitions. We start noting that if we remove from the two configurations the

definitions for the same set of names, bisimilarity is preserved. Then, we note that the configurations obtained by duplicating existing definitions, using arbitrary fresh trigger names, remain bisimilar. These properties will be useful for showing that bisimilarity is closed under parallel composition.

Lemma B.13 (Bisimilarity and Definitions) *Let K and L be well-formed configurations.*

1. *If $K \mid \Theta^{\widetilde{k}} \approx_\Lambda L \mid \Omega^{\widetilde{k}}$ then $K \approx_\Lambda L$.*
2. *If $K \mid \Theta^{\widetilde{k}} \approx_\Lambda L \mid \Omega^{\widetilde{k}}$ then $K \mid \Theta^{\widetilde{k}} \mid \Theta^{\widetilde{k}\frown\widetilde{j}} \approx_\Lambda L \mid \Omega^{\widetilde{k}} \mid \Omega^{\widetilde{k}\frown\widetilde{j}}$.*

Proof.

1. The family $\dot{\approx}$ with generic element

$$\dot{\approx}_\Delta = \left\{ (K, L) \;\; : \;\; K \mid \Theta^{\widetilde{k}} \approx_\Delta L \mid \Omega^{\widetilde{k}} \right\}$$

is a domain bisimulation. Follows by analyzing the transitions of K. The intuition is every transition by $K \mid \Theta^{\widetilde{k}}$ originating from K must be matched by $L \mid \Omega^{\widetilde{k}}$ using a (weak) transition originating from L alone, since $\Theta^{\widetilde{k}}$ and $\Omega^{\widetilde{k}}$ can perform only (Lts Def) transitions. We show the most interesting case, for (Lts Req). Suppose $K \xrightarrow{(\widetilde{h})\iota\cdot\mathsf{req}\langle p\rangle(T)} K'$. By hypothesis, $K \mid \Theta^{\widetilde{k}} \approx_\Lambda L \mid \Omega^{\widetilde{k}}$ for some $\widetilde{k}$ such that $\{\widetilde{h}\} \cap \{\widetilde{k}\} = \emptyset$. By definition of lts, $K \mid \Theta^{\widetilde{k}} \xrightarrow{(\widetilde{h})\iota\cdot\mathsf{req}\langle p\rangle(T)} K' \mid \Theta^{\widetilde{k}}$. By bisimilarity, $L \mid \Omega^{\widetilde{k}} \xrightarrow{\tau}\!\!\!\!\rightarrow L' \xrightarrow{(\widetilde{h})\iota\cdot\mathsf{req}\langle p\rangle(T)} L'' \xrightarrow{\tau}\!\!\!\!\rightarrow L'''$ and $K' \mid \Theta^{\widetilde{k}} \approx_\Lambda L'''$. By definition of lts and by induction on the number of tau transitions, we have that $L \xrightarrow{\tau}\!\!\!\!\rightarrow L_1$ where $L' = L_1 \mid \Omega^{\widetilde{k}}$. The transition $L_1 \mid \Omega^{\widetilde{k}} \xrightarrow{(\widetilde{h})\iota\cdot\mathsf{req}\langle p\rangle(T)} L''$ must follow by repeatedly applying (Lts Par), starting from the premise $L_1 \xrightarrow{(\widetilde{h})\iota\cdot\mathsf{req}\langle p\rangle(T)} L_2$, hence $L'' = L_2 \mid \Omega^{\widetilde{k}}$. Again, by definition of lts and by induction on the number of tau transitions, we have that $L_2 \xrightarrow{\tau}\!\!\!\!\rightarrow L_3$ where $L''' = L_3 \mid \Omega^{\widetilde{k}}$. By composing the transitions, we obtain $L \xrightarrow{(\widetilde{h})\iota\cdot\mathsf{req}\langle p\rangle(T)}\!\!\!\!\rightarrow L_3$, and we conclude because $K' \mid \Theta^{\widetilde{k}} \approx_\Lambda L_3 \mid \Omega^{\widetilde{k}}$.

2. The family $\dot{\approx}$ with generic element

$$\dot{\approx}_\Delta = \left\{ (K \mid \Theta^{\widetilde{k}} \mid \Theta^{\widetilde{k}\frown\widetilde{j}}, L \mid \Omega^{\widetilde{k}} \mid \Omega^{\widetilde{k}\frown\widetilde{j}}) \;\; : \;\; K \mid \Theta^{\widetilde{k}} \approx_\Delta L \mid \Omega^{\widetilde{k}} \right\}$$

is a domain bisimulation. Follows by analysis of the transitions of $K \mid \Theta^{\widetilde{k}} \mid \Theta^{\widetilde{k}\frown\widetilde{j}}$, by syntactic reasoning using Lemma B.7. The transitions by K and by L are treated like in point (1) above. The intuition for the (Lts Def) transitions originated from $\Theta^{\widetilde{k}\frown\widetilde{j}}$ is that since $\Theta^{\widetilde{k}}$ and $\Theta^{\widetilde{k}\frown\widetilde{j}}$ have

the same transitions up-to renaming of triggers, every process generated by trigger transitions from $\Theta^{\widetilde{k}\curvearrowright\widetilde{j}}$ could also be generated by $\Theta^{\widetilde{k}}$ alone. All that is needed is to match the transition of $\Theta^{\widetilde{k}\curvearrowright\widetilde{j}}$ with a corresponding one by $\Omega^{\widetilde{k}\curvearrowright\widetilde{j}}$, which exists because $K \mid \Theta^{\widetilde{k}} \approx_\Lambda L \mid \Omega^{\widetilde{k}}$ and $\Omega^{\widetilde{k}}$ can match $\Theta^{\widetilde{k}}$.

□

Lemma B.14 (Parallel Composition) *Bisimilarity is closed under parallel composition:* $K \approx_\Lambda L \Longrightarrow K \mid M \approx_\Lambda L \mid M$.

Proof. In order to show closure under parallel composition, we will identify a domain bisimulation $\dot{\approx}$ containing all the pairs of the form $(K \mid M, L \mid M)$ such that K is bisimilar to L, plus any other pair of terms generated by the labelled transition system. In particular, we must handle with care the terms generated by a communication steps between K (or L) and M involving scripts. The idea is that we represent explicitly, using the merge operators, the definitions corresponding to the communicated scripts. More in detail, by point 3 of Lemma B.12 we know that, since $K \approx_\Lambda L$, then $K \equiv \Theta^{\widetilde{k}} \mid P'$ and $L \equiv \Omega^{\widetilde{k}} \mid Q'$ for some appropriate $P', \Theta^{\widetilde{k}}, Q', \Omega^{\widetilde{k}}$. Moreover, using point 1 of Lemma B.12 we can rewrite $P' \equiv \langle\!\langle P \mid \Theta^{\widetilde{m}} \rangle\!\rangle$ and $Q' \equiv \langle\!\langle Q \mid \Omega^{\widetilde{m}} \rangle\!\rangle$ for some appropriate $P, \Theta^{\widetilde{m}}, Q, \Omega^{\widetilde{m}}$. That is, $K \equiv \Theta^{\widetilde{k}} \mid \langle\!\langle P \mid \Theta^{\widetilde{m}} \rangle\!\rangle$ and $L \equiv \Omega^{\widetilde{k}} \mid \langle\!\langle Q \mid \Omega^{\widetilde{m}} \rangle\!\rangle$. Again by point 1 of Lemma B.12, we also know that $M \equiv \langle\!\langle \Phi^{\widetilde{n}} \mid R \rangle\!\rangle \mid \Phi^{\widetilde{i}}$ for some $R, \Phi^{\widetilde{i}}, \Phi^{\widetilde{n}}$. The terms $\Theta^{\widetilde{m}}, \Omega^{\widetilde{m}}$ and $\Phi^{\widetilde{n}}$ represent definitions corresponding to the scripts that respectively K, L or M may communicate in a future transition.

Our candidate bisimulation is the family $\dot{\approx}$ with generic element $\dot{\approx}_\Delta$ defined (up to $\equiv$) by the pairs

$$\big((\nu\,\widetilde{c})(\Theta^{\widetilde{k}} \mid \langle\!\langle P \mid \Theta^{\widetilde{m}} \mid \Phi^{\widetilde{h}} \rangle\!\rangle \mid \langle\!\langle \Theta^{\widetilde{j}} \mid \Phi^{\widetilde{n}} \mid R \rangle\!\rangle \mid \Phi^{\widetilde{i}}), (\nu\,\widetilde{c})(\Omega^{\widetilde{k}} \mid \langle\!\langle Q \mid \Omega^{\widetilde{m}} \mid \Phi^{\widetilde{h}} \rangle\!\rangle \mid \langle\!\langle \Omega^{\widetilde{j}} \mid \Phi^{\widetilde{n}} \mid R \rangle\!\rangle \mid \Phi^{\widetilde{i}})\big)$$

(where all the $\widetilde{h}, \widetilde{i}, \widetilde{j}, \widetilde{k}, \widetilde{m}, \widetilde{m}$ are distinct) such that

$$\Theta^{\widetilde{k}} \mid \Theta^{\widetilde{m}} \mid \Theta^{\widetilde{j}} \mid P \approx_\Delta \Omega^{\widetilde{k}} \mid \Omega^{\widetilde{m}} \mid \Omega^{\widetilde{j}} \mid Q$$

The extra terms $\Phi^{\widetilde{h}}, \Theta^{\widetilde{j}}, \Omega^{\widetilde{j}}$ represent the definitions corresponding to the scripts that respectively M, K or L may have communicated using a labelled transition to either K or L, or to M. Note that $\{(K \mid M, L \mid M) \;:\; K \approx_\Delta L\}$ is contained in $\dot{\approx}_\Delta$ (up to $\equiv$), by choosing $\Phi^{\widetilde{h}} = \Theta^{\widetilde{j}} = \Phi^{\widetilde{j}} = \mathbf{0}$ and $\widetilde{c}$ empty.

We now proceed to show that $\dot{\approx}$ is a domain bisimulation. For readability, we omit the subscript Δ on weak transitions $\stackrel{a}{\longrightarrow}_\Delta$, and we use the abbreviations

$$\begin{aligned}
K_* &= (\nu\,\widetilde{c})(\Theta^{\widetilde{k}} \mid \langle\!\langle P \mid \Theta^{\widetilde{m}} \mid \Phi^{\widetilde{h}} \rangle\!\rangle \mid \langle\!\langle \Theta^{\widetilde{j}} \mid \Phi^{\widetilde{n}} \mid R \rangle\!\rangle \mid \Phi^{\widetilde{i}}) \\
L_* &= (\nu\,\widetilde{c})(\Omega^{\widetilde{k}} \mid \langle\!\langle Q \mid \Omega^{\widetilde{m}} \mid \Phi^{\widetilde{h}} \rangle\!\rangle \mid \langle\!\langle \Omega^{\widetilde{j}} \mid \Phi^{\widetilde{n}} \mid R \rangle\!\rangle \mid \Phi^{\widetilde{i}}) \\
K_1 &= \Theta^{\widetilde{k}} \mid \Theta^{\widetilde{m}} \mid \Theta^{\widetilde{j}} \mid P \\
L_1 &= \Omega^{\widetilde{k}} \mid \Omega^{\widetilde{m}} \mid \Omega^{\widetilde{j}} \mid Q
\end{aligned}$$

Suppose $K_* \xrightarrow{\alpha_l} K'$. The proof is by cases on α_l, where we only consider the subcases with $l \in \Delta$ as the others follow from the definition of domain bisimulation.

For each case we use Lemma B.3, and pattern matching between the syntax of the terms above and of the terms in the lemma. We start with the case for input transitions, which is the easiest.

- $(K_* \xrightarrow{l\cdot k(\widetilde{\pi}\sigma)} K')$: We aim to bring the script instantiated by the transition inside the leftmost merge operator in K_*. In order to do so, we need to avoid both the capture of private channel names in σ by $\widetilde{c}$, and clashes between trigger names in sigma and the vectors $\widetilde{h}, \widetilde{m}$. We split the proof in two cases, depending on whether $k \in \widetilde{k}$ or $k \in \widetilde{i}$.

  - $\Theta^{\widetilde{k}} \xrightarrow{l\cdot k(\widetilde{\pi}\sigma)} \Theta^{\widetilde{k}} \,|\, \boldsymbol{P}_k\sigma$, where $\langle k \Leftarrow (\widetilde{\pi})\boldsymbol{P}_k\rangle$ is in $\Theta^{\widetilde{k}}$. To avoid clashes between trigger names, we choose for some fresh $\widetilde{h'}, \widetilde{m'}$ and, using standard properties of substitution, rewrite

    $$\langle\!\langle P \,|\, \Theta^{\widetilde{m}} \,|\, \Phi^{\widetilde{h}} \rangle\!\rangle = \langle\!\langle P\{\widetilde{m'}/\widetilde{m}\}\{\widetilde{h'}/\widetilde{h}\} \,|\, \Theta^{\widetilde{m}\frown\widetilde{m'}} \,|\, \Phi^{\widetilde{h}\frown\widetilde{h'}} \rangle\!\rangle$$

    Let $\rho = \{\widetilde{m'}/\widetilde{m}\}\{\widetilde{h'}/\widetilde{h}\}\{\widetilde{c'}/\widetilde{c}\}$ for a fresh $\widetilde{c'}$, and recall that the private channel names $\widetilde{c}$ cannot appear free in definitions (by well-formedness of scripts). By α-conversion,

    $$K' \equiv (\nu\, \widetilde{c'})(\Theta^{\widetilde{k}} \,|\, \langle\!\langle \boldsymbol{P}_k\sigma \,|\, P\rho \,|\, \Theta^{\widetilde{m}\frown\widetilde{m'}} \,|\, \Phi^{\widetilde{h}\frown\widetilde{h'}} \rangle\!\rangle \,|\, \langle\!\langle \Theta^{\widetilde{j}} \,|\, \Phi^{\widetilde{n}} \,|\, R\{\widetilde{c'}/\widetilde{c}\} \rangle\!\rangle \,|\, \Phi^{\widetilde{i}})$$

    By $K_1 \approx_\Delta L_1$ and Lemma B.8, $K_2 = K_1\rho \approx_\Delta L_1\rho = L_2$. Since $\Theta^{\widetilde{k}}$ occurs in K_2, $K_2 \xrightarrow{l\cdot k(\widetilde{\pi}\sigma)} K_2 \,|\, \boldsymbol{P}_k\sigma = K_2'$. By $K_2 \approx_\Delta L_2$, $L_2 \xrightarrow{\tau}\!\!\!\!\rightarrow \xrightarrow{l\cdot k(\widetilde{\pi}\sigma)} \xrightarrow{\tau}\!\!\!\!\rightarrow \Omega^{\widetilde{k}} \,|\, \Omega^{\widetilde{m}\frown\widetilde{m'}} \,|\, \Omega^{\widetilde{j}} \,|\, Q' = L_2'$ with $L_2' \approx_\Delta K_2'$. We will now use the transition between L_2 and L_2' to derive an appropriate one between L_* and L'. Since the action $\xrightarrow{l\cdot k(\widetilde{\pi}\sigma)}$ necessarily originated by $\Omega^{\widetilde{k}}$, which contains the definition $\langle k \Leftarrow (\widetilde{\pi})\boldsymbol{Q}_k\rangle$, we can reorder the reduction obtaining

    $$L_2 \xrightarrow{l\cdot k(\widetilde{\pi}\sigma)} \Omega^{\widetilde{k}} \,|\, \Omega^{\widetilde{m}\frown\widetilde{m'}} \,|\, \Omega^{\widetilde{j}} \,|\, Q\rho \,|\, \boldsymbol{Q}_k\sigma \xrightarrow{\tau}\!\!\!\!\rightarrow \Omega^{\widetilde{k}} \,|\, \Omega^{\widetilde{m}\frown\widetilde{m'}} \,|\, \Omega^{\widetilde{j}} \,|\, Q'$$

    By α-conversion, properties of substitutions and definition of lts,

    $$L_* \xrightarrow{l\cdot k(\widetilde{\pi}\sigma)} L''$$

    $$L'' = (\nu\, \widetilde{c'})(\Omega^{\widetilde{k}} \,|\, \langle\!\langle \boldsymbol{Q}_k\sigma \,|\, Q\rho \,|\, \Omega^{\widetilde{m}\frown\widetilde{m'}} \,|\, \Phi^{\widetilde{h}\frown\widetilde{h'}} \rangle\!\rangle \,|\, \langle\!\langle \Omega^{\widetilde{j}} \,|\, \Phi^{\widetilde{n}} \,|\, R\{\widetilde{c'}/\widetilde{c}\} \rangle\!\rangle \,|\, \Phi^{\widetilde{i}})$$

    where we have brought $\boldsymbol{Q}_k$ inside the leftmost merge operator, in order to preserve the general structure that we have imposed on

terms in $\dot{\approx}_\Delta$. By syntactical reasoning, it must be the case that $\boldsymbol{Q}_k\sigma \mid Q\rho \xrightarrow{\tau} Q'$. By point 2 of Lemma B.12,

$$L'' \xrightarrow{\tau} (\nu\, \widetilde{c'})(\Omega^{\widetilde{k}} \mid \langle\!\langle Q' \mid \Omega^{\widetilde{m} \frown \widetilde{m'}} \mid \Phi^{\widetilde{h} \frown \widetilde{h'}} \rangle\!\rangle \mid \langle\!\langle \Omega^{\widetilde{j}} \mid \Phi^{\widetilde{n}} \mid R\{\widetilde{c'}/\widetilde{c}\} \rangle\!\rangle \mid \Phi^{\widetilde{i}}) = L'$$

and we conclude because, since $K'_2 \approx_\Lambda L'_2$, we have $(K', L') \in \dot{\approx}_\Delta$.

– $\Phi^{\widetilde{i}} \xrightarrow{l \cdot k(\widetilde{\pi}\sigma)} \Phi^{\widetilde{i}} \mid \boldsymbol{R}_k\sigma$: similar to the previous case but simpler, since, instead of using the hypothesis $K_1 \approx_\Delta L_1$, it is enough to use syntactical reasoning.

- $(K_* \xrightarrow{l \cdot a(\widetilde{v})} K')$: Similar to the previous case.
- $(K_* \xrightarrow{(\widetilde{b_*}, \widetilde{k_*})\overline{l \cdot a}\langle\widetilde{v}\rangle} K')$: We distinguish two cases, depending on whether the output transition is originated by R or P.

  – Suppose $\langle\!\langle \Theta^{\widetilde{j}} \mid \Phi^{\widetilde{n}} \mid R \rangle\!\rangle \xrightarrow{(\widetilde{b'}, \widetilde{k'}, \widetilde{i'})\overline{l \cdot a}\langle\widetilde{v}\rangle} K_0$, where $\widetilde{c} = \widetilde{c'}, \widetilde{b}$ and $\widetilde{b_*} = \widetilde{b}, \widetilde{b'}$. We assume that the trigger names $\widetilde{k'}$ come from $\Theta^{\widetilde{j}}$, whereas the $\widetilde{i'}$ come from $\Phi^{\widetilde{n}}$ or R. Unfolding the definition of merge, by Lemma B.3 we have that $\langle\!\langle \Theta^{\widetilde{j}} \mid \Phi^{\widetilde{n}} \mid R \rangle\!\rangle = R^{\Theta^{\widetilde{j}}\Phi^{\widetilde{n}}} \equiv (\nu\, \widetilde{b'})(\overline{l \cdot a}\langle \widetilde{v_1}^{\Theta^{\widetilde{j}}} \rangle \mid R_1^{\Theta^{\widetilde{j}}\Phi^{\widetilde{n}}})$ with $\{\widetilde{b'}\} \subseteq \mathit{fn}(\widetilde{v_1})$, where $\widetilde{v_1} = \widetilde{v_*}^{\Phi^{\widetilde{n}}}$ for some appropriate $\widetilde{v_*}$. Moreover, we have $K_0 \equiv R_1^{\Theta^{\widetilde{j}}\Phi^{\widetilde{n}}} \mid \Theta^{\widetilde{k'}} \mid \Phi^{\widetilde{i'}}$, where $\mathfrak{X}(\widetilde{v_1}^{\Theta^{\widetilde{j}}}) = (\widetilde{v}; \Theta^{\widetilde{k'}} \mid \Phi^{\widetilde{i'}})$. To find out how to split the definitions produced by the extraction into $\Theta^{\widetilde{k'}}$ and $\Phi^{\widetilde{i'}}$, we assume to have first applied the extraction $\mathfrak{X}(\widetilde{v_1}) = (\widetilde{v'}; \Phi^{\widetilde{i'}})$, which ensures that the definitions in $\Phi^{\widetilde{i'}}$ come from R or $\Phi^{\widetilde{n}}$. Then, applying $\mathfrak{X}(\widetilde{v_1}^{\Theta^{\widetilde{j}}}) = (\widetilde{v}; \Theta^{\widetilde{k'}} \mid \Phi^{\widetilde{i'}})$ we can infer that for each $\langle k' \Leftarrow P_0 \rangle$ in $\Theta^{\widetilde{k'}}$ there is $\langle j \Leftarrow P_0 \rangle$ in $\Theta^{\widetilde{j}}$, since, by points 1 and 2 of Lemma B.11, we have that $\widetilde{v}\{\widetilde{k'}/\widetilde{j'}\} = \widetilde{v'}$, where $\widetilde{j'}$ are the triggers in $\widetilde{j}$ occurring also in v_1. With this information, by definition of lts, we can rearrange K' to respect our general pattern

$$K' \equiv (\nu\, \widetilde{c'})(\Theta^{\widetilde{k}} \mid \Theta^{\widetilde{k'}} \mid \langle\!\langle P \mid \Theta^{\widetilde{m}} \mid \Phi^{\widetilde{h}} \rangle\!\rangle \mid \langle\!\langle \Theta^{\widetilde{j}} \mid \Phi^{\widetilde{n}} \mid R_1 \rangle\!\rangle \mid \Phi^{\widetilde{i}} \mid \Phi^{\widetilde{i'}})$$

By applying the same argument to L_*, $\langle\!\langle \Omega^{\widetilde{j}} \mid \Phi^{\widetilde{n}} \mid R \rangle\!\rangle = R^{\Omega^{\widetilde{j}}\Phi^{\widetilde{n}}}$ and

$$R^{\Omega^{\widetilde{j}}\Phi^{\widetilde{n}}} \equiv (\nu\, \widetilde{b'})(\overline{l \cdot a}\langle \widetilde{v_1}^{\Omega^{\widetilde{j}}} \rangle \mid R_1^{\Omega^{\widetilde{j}}\Phi^{\widetilde{n}}}) \xrightarrow{(\widetilde{b'}, \widetilde{k'}, \widetilde{i'})\overline{l \cdot a}\langle\widetilde{v}\rangle} R_1^{\Omega^{\widetilde{j}}\Phi^{\widetilde{n}}} \mid \Omega^{\widetilde{k'}} \mid \Phi^{\widetilde{i'}}$$

for $\mathfrak{X}(\widetilde{v_1}^{\Omega^{\widetilde{j}}}) = (\widetilde{v}; \Omega^{\widetilde{k'}} \mid \Phi^{\widetilde{i'}})$, where for each $\langle k' \Leftarrow Q_0 \rangle$ in $\Omega^{\widetilde{k'}}$ there is a $\langle j \Leftarrow Q_0 \rangle$ in $\Omega^{\widetilde{j}}$. By definition of lts,

$$L_* \xrightarrow{(\widetilde{b}, \widetilde{b'}, \widetilde{k'}, \widetilde{i'})\overline{l \cdot a}\langle\widetilde{v}\rangle} L' \equiv (\nu\, \widetilde{c'})(\Omega^{\widetilde{k}} \mid \Omega^{\widetilde{k'}} \mid \langle\!\langle Q \mid \Omega^{\widetilde{m}} \mid \Phi^{\widetilde{h}} \rangle\!\rangle \mid \langle\!\langle \Omega^{\widetilde{j}} \mid \Phi^{\widetilde{n}} \mid R_1 \rangle\!\rangle \mid \Phi^{\widetilde{i}} \mid \Phi^{\widetilde{i'}})$$

By $K_1 \approx_\Delta L_1$ and point 2 of Lemma B.13,

$$\Theta^{\widetilde{k}} \,|\, \Theta^{\widetilde{k'}} \,|\, \Theta^{\widetilde{m}} \,|\, \Theta^{\widetilde{j}} \,|\, P \approx_\Delta \Omega^{\widetilde{k}} \,|\, \Omega^{\widetilde{k'}} \,|\, \Omega^{\widetilde{m}} \,|\, \Omega^{\widetilde{j}} \,|\, Q$$

and we conclude because $(K', L') \in \dot{\approx}_\Delta$.

– Suppose $\langle\!\langle P \,|\, \Theta^{\widetilde{m}} \,|\, \Phi^{\widetilde{h}} \rangle\!\rangle \xrightarrow{(\widetilde{b'}, \widetilde{k'}, \widetilde{m'}, \widetilde{i'})\overline{l \cdot a}\langle \widetilde{v} \rangle} K_0$, where $\widetilde{c} = \widetilde{c'}, \widetilde{b}$ and $\widetilde{b_*} = \widetilde{b}, \widetilde{b'}$. We assume $\widetilde{i'}$ are the new trigger names from $\Phi^{\widetilde{h}}$, $\widetilde{m'}$ the ones from $\Theta^{\widetilde{m}}$ and $\widetilde{k'}$ the ones from P. Differently from the previous case, we need to keep track explicitly of the triggers coming from $\Theta^{\widetilde{m}}$, which will correspond in L_* to triggers coming from $\Omega^{\widetilde{m}}$. We have that

$$\langle\!\langle P \,|\, \Theta^{\widetilde{m}} \,|\, \Phi^{\widetilde{h}} \rangle\!\rangle = P^{\Theta^{\widetilde{m}} \Phi^{\widetilde{h}}} \equiv (\nu\, \widetilde{b'})(\overline{l \cdot a}\langle \widetilde{v_1}^{\,\Theta^{\widetilde{m}} \Phi^{\widetilde{h}}} \rangle \,|\, P_1^{\Theta^{\widetilde{m}} \Phi^{\widetilde{h}}})$$

with $\{\widetilde{b'}\} \subseteq \mathit{fn}(\widetilde{v})$, and $K_0 \equiv P_1^{\Theta^{\widetilde{m}} \Phi^{\widetilde{h}}} \,|\, \Theta^{\widetilde{k'}} \,|\, \Theta^{\widetilde{m'}} \,|\, \Phi^{\widetilde{i'}}$, where we assume that $\widetilde{k'}$, $\widetilde{m'}$ and $\widetilde{i'}$ are disjoint from $\widetilde{k}$, $\widetilde{m}$ and $\widetilde{i}$. Moreover, we have

$$\begin{aligned} \mathfrak{X}(\widetilde{v_1}^{\,\Theta^{\widetilde{m}} \Phi^{\widetilde{h}}}) &= (\widetilde{v}; \Theta^{\widetilde{k'}} \,|\, \Theta^{\widetilde{m'}} \,|\, \Phi^{\widetilde{i'}}) \\ \mathfrak{X}(\widetilde{v_1}^{\,\Theta^{\widetilde{m}}}) &= (\widetilde{v''}; \Theta^{\widetilde{k'}} \,|\, \Theta^{\widetilde{m'}}) \\ \mathfrak{X}(\widetilde{v_1}) &= (\widetilde{v'}; \Theta^{\widetilde{k'}}) \end{aligned}$$

Hence, by Lemma B.11, for each $\langle i' \Leftarrow R_0 \rangle$ in $\Phi^{\widetilde{i'}}$ there is $\langle h \Leftarrow R_0 \rangle$ in $\Phi^{\widetilde{h}}$, and for each $\langle m' \Leftarrow P_0 \rangle$ in $\Theta^{\widetilde{m'}}$ there is $\langle m \Leftarrow P_0 \rangle$ in $\Theta^{\widetilde{m}}$. By definition of lts,

$$K' \equiv (\nu\, \widetilde{c'})(\Theta^{\widetilde{k}} \,|\, \Theta^{\widetilde{k'}} \,|\, \Theta^{\widetilde{m'}} \,|\, \langle\!\langle P_1 \,|\, \Theta^{\widetilde{m}} \,|\, \Phi^{\widetilde{h}} \rangle\!\rangle \,|\, \langle\!\langle \Theta^{\widetilde{j}} \,|\, \Phi^{\widetilde{n}} \,|\, R \rangle\!\rangle \,|\, \Phi^{\widetilde{i}} \,|\, \Phi^{\widetilde{i'}})$$

where we have used the information gathered above on $\Theta^{\widetilde{k'}}$, $\Theta^{\widetilde{m'}}$ and $\Phi^{\widetilde{i'}}$ to decide how to rearrange K', in order to fit our general pattern. By definition of lts,

$$K_1 \xrightarrow{(\widetilde{b'}, \widetilde{k'})\overline{l \cdot a}\langle \widetilde{v'} \rangle} P_1 \,|\, \Theta^{\widetilde{k}} \,|\, \Theta^{\widetilde{k'}} \,|\, \Theta^{\widetilde{m}} \,|\, \Theta^{\widetilde{j}} = K_1'$$

where $\{\widetilde{b'}\} \subseteq \mathit{fn}(\widetilde{v'})$ and $\mathfrak{X}(\widetilde{v_1}) = (\widetilde{v'}; \Theta^{\widetilde{k'}})$. By $K_1 \approx_\Delta L_1$ and point 3 of Lemma B.12,

$$L_1 \xrightarrow{(\widetilde{b'}, \widetilde{k'})\overline{l \cdot a}\langle \widetilde{v'} \rangle} \!\!\!\rightarrow Q_1 \,|\, \Omega^{\widetilde{k}} \,|\, \Omega^{\widetilde{k'}} \,|\, \Omega^{\widetilde{m}} \,|\, \Omega^{\widetilde{j}} = L_1'$$

and $K_1' \approx_\Delta L_1'$. We now derive a corresponding transition for L_*. Since none of the transitions above can be generated by a definition, we can deduce

$$Q \xrightarrow{\tau} \!\!\!\rightarrow (\nu\, \widetilde{b'})(\overline{l \cdot a}\langle \widetilde{v_2} \rangle \,|\, Q_2) = Q_3, \quad Q_3 \xrightarrow{(\widetilde{b'}, \widetilde{k'})\overline{l \cdot a}\langle \widetilde{v'} \rangle} Q_2 \,|\, \Omega^{\widetilde{k'}} \xrightarrow{\tau} \!\!\!\rightarrow Q_1 \,|\, \Omega^{\widetilde{k'}}$$

where $\mathfrak{X}(\widetilde{v_2}) = (\widetilde{v'}; \Omega^{\widetilde{k'}})$. By Lemma B.11, $\widetilde{v_2}$ has exactly the same occurrences of trigger names in $\widetilde{h}$ as $\widetilde{v'}$, which are the same of $\widetilde{v_1}$. By point 2 of Lemma B.12, $\langle\!\langle Q \,|\, \Omega^{\widetilde{m}} \,|\, \Phi^{\widetilde{h}} \rangle\!\rangle \xrightarrow{\tau} \!\!\!\rightarrow \langle\!\langle Q_3 \,|\, \Omega^{\widetilde{m}} \,|\, \Phi^{\widetilde{h}} \rangle\!\rangle$. By syntactical reasoning

$$\langle\!\langle Q_3 \,|\, \Omega^{\widetilde{m}} \,|\, \Phi^{\widetilde{h}} \rangle\!\rangle \xrightarrow{(\widetilde{b'},\widetilde{k'},\widetilde{m'},\widetilde{i'})\overline{l\cdot a}\langle\widetilde{v}\rangle} \langle\!\langle Q_2 \,|\, \Omega^{\widetilde{m}} \,|\, \Phi^{\widetilde{h}} \rangle\!\rangle \,|\, \Omega^{\widetilde{k'}} \,|\, \Omega^{\widetilde{m'}} \,|\, \Phi^{\widetilde{i'}}$$

where $\mathfrak{X}(\widetilde{v_2}^{\Omega^{\widetilde{m}}\Phi^{\widetilde{h}}}) = (\widetilde{v}; \Omega^{\widetilde{k'}} \,|\, \Omega^{\widetilde{m'}} \,|\, \Phi^{\widetilde{i'}})$. By point 2 of Lemma B.12 and by definition of lts,

$$\langle\!\langle Q_2 \,|\, \Omega^{\widetilde{m}} \,|\, \Phi^{\widetilde{h}} \rangle\!\rangle \,|\, \Omega^{\widetilde{k'}} \,|\, \Omega^{\widetilde{m'}} \,|\, \Phi^{\widetilde{i'}} \xrightarrow{\tau} \!\!\!\rightarrow \langle\!\langle Q_1 \,|\, \Omega^{\widetilde{m}} \,|\, \Phi^{\widetilde{h}} \rangle\!\rangle \,|\, \Omega^{\widetilde{k'}} \,|\, \Omega^{\widetilde{m'}} \,|\, \Phi^{\widetilde{i'}}$$

By definition of lts, $L_* \xrightarrow{(\widetilde{b},\widetilde{b'},\widetilde{k'},\widetilde{m'},\widetilde{i'})\overline{l\cdot a}\langle\widetilde{v}\rangle} \!\!\!\rightarrow L'$ and

$$L' \equiv (\nu\, \widetilde{c'})(\Omega^{\widetilde{k}} \,|\, \Omega^{\widetilde{k'}} \,|\, \Omega^{\widetilde{m'}} \,|\, \langle\!\langle Q_1 \,|\, \Omega^{\widetilde{m}} \,|\, \Phi^{\widetilde{h}} \rangle\!\rangle \,|\, \langle\!\langle \Omega^{\widetilde{j}} \,|\, \Phi^{\widetilde{n}} \,|\, R \rangle\!\rangle \,|\, \Phi^{\widetilde{i}} \,|\, \Phi^{\widetilde{i'}})$$

By $K_1' \approx_\Delta L_1'$ and point 2 of Lemma B.13,

$$\Theta^{\widetilde{k}} \,|\, \Theta^{\widetilde{k'}} \,|\, \Theta^{\widetilde{m'}} \,|\, \Theta^{\widetilde{m}} \,|\, \Theta^{\widetilde{j}} \,|\, P_1 \approx_\Delta \Omega^{\widetilde{k}} \,|\, \Omega^{\widetilde{k'}} \,|\, \Omega^{\widetilde{m'}} \,|\, \Omega^{\widetilde{m}} \,|\, \Omega^{\widetilde{j}} \,|\, Q_1$$

and we conclude because $(K', L') \in \dot{\approx}_\Delta$.

- $(K_* \xrightarrow{(\widetilde{b_*},\widetilde{k_*})\overline{l\cdot j}\langle\widetilde{v}\rangle} K')$: Analogous to the case for output.
- $(K_* \xrightarrow{(\widetilde{k})l\cdot \mathsf{req}\langle p\rangle(T)} K')$: By combining the argument for input and output.
- $(K_* \xrightarrow{l\cdot\tau} K')$: First we analyze the case where the transition is determined by the interaction of R and P, then the case where the transition is derived by R or P in isolation.

  Interaction. We analyze the transitions resulting from an interaction between R and P. We must distinguish four cases depending on whether R or P receives the value, and whether the value is received by a replicated input. Suppose $\langle\!\langle P \,|\, \Theta^{\widetilde{m}} \,|\, \Phi^{\widetilde{h}} \rangle\!\rangle \,|\, \langle\!\langle \Theta^{\widetilde{j}} \,|\, \Phi^{\widetilde{n}} \,|\, R \rangle\!\rangle \xrightarrow{\tau} K_0$.

  Replicated input by R: By Lemma B.3, we have that

  $$R^{\Theta^{\widetilde{j}}\Phi^{\widetilde{n}}} \equiv (\nu\, \widetilde{b})(R_1^{\Theta^{\widetilde{j}}\Phi^{\widetilde{n}}} \,|\, !l\cdot a(\widetilde{\pi}).R_2^{\Theta^{\widetilde{j}}\Phi^{\widetilde{n}}})$$

  where $\widetilde{b}$ is fresh with respect to P and $\widetilde{c}$, and $a \notin \{\widetilde{b}\}$. We also have that $P^{\Theta^{\widetilde{m}}\Phi^{\widetilde{h}}} \equiv (\nu\, \widetilde{c'})(\overline{l\cdot a}\langle\widetilde{v}^{\Theta^{\widetilde{m}}\Phi^{\widetilde{h}}}\rangle \,|\, P_1^{\Theta^{\widetilde{m}}\Phi^{\widetilde{h}}})$, where $\widetilde{c'}$ is fresh with respect to R and $\widetilde{c}$, $\{\widetilde{c'}\} \subseteq fn(v^{\Theta^{\widetilde{m}}\Phi^{\widetilde{h}}})$, and $\widetilde{v}^{\Theta^{\widetilde{m}}\Phi^{\widetilde{h}}} = \widetilde{\pi}\sigma'$. Moreover,

  $$K_0 \equiv (\nu\, \widetilde{c'})(P_1^{\Theta^{\widetilde{m}}\Phi^{\widetilde{h}}} \,|\, (\nu\, \widetilde{b})(R_1^{\Theta^{\widetilde{j}}\Phi^{\widetilde{n}}} \,|\, !l\cdot a(\widetilde{\pi}).R_2^{\Theta^{\widetilde{j}}\Phi^{\widetilde{n}}} \,|\, R_2^{\Theta^{\widetilde{j}}\Phi^{\widetilde{n}}}\sigma'))$$

Since scripts cannot contain free private names, $\{\widetilde{c'}\} \subseteq \mathit{fn}(v^{\Theta^{\widetilde{m}}\Phi^{\widetilde{h}}})$ implies $\{\widetilde{c'}\} \subseteq \mathit{fn}(v)$. Since patterns cannot contain scripts (or trigger names), there exists σ such that $\widetilde{v} = \widetilde{\pi}\sigma$ and $\sigma^{\Theta^{\widetilde{m}}\Phi^{\widetilde{h}}} = \sigma'$. We want to derive a configuration K' of the right form. By definition of merge, $P_1^{\Theta^{\widetilde{m}}\Phi^{\widetilde{h}}} = \langle\!\langle P_1 \mid \Theta^{\widetilde{m}} \mid \Phi^{\widetilde{h}} \rangle\!\rangle$. Before rewriting

$$R_* = (\nu\, \widetilde{b})(R_1^{\Theta^{\widetilde{j}}\Phi^{\widetilde{n}}} \mid !l{\cdot}a(\widetilde{\pi}).R_2^{\Theta^{\widetilde{j}}\Phi^{\widetilde{n}}} \mid R_2^{\Theta^{\widetilde{j}}\Phi^{\widetilde{n}}}\sigma^{\Theta^{\widetilde{m}}\Phi^{\widetilde{h}}})$$

in terms of the merge operator, we want to to be explicit about the scripts occurring in $\widetilde{v}$. Suppose $\mathfrak{X}(\widetilde{v}) = (\widetilde{v'}; \Theta^{\widetilde{k'}})$ for fresh $\widetilde{k'}$. Since $\widetilde{v} = \widetilde{\pi}\sigma$ and $\widetilde{v'}$ differs from $\widetilde{v}$ only for having triggers replacing scripts, there exists ρ such that $\widetilde{v'} = \widetilde{\pi}\rho$. By Lemma B.11, $\widetilde{v} = \widetilde{v'}^{\Theta^{\widetilde{k'}}}$. Since $\widetilde{v} = \widetilde{\pi}\sigma$, $\widetilde{v'} = \widetilde{\pi}\rho$ and $\widetilde{\pi}$ cannot contain trigger names, we have that $\sigma = \rho^{\Theta^{\widetilde{k'}}}$, and both $\widetilde{v}$ and $\widetilde{v'}$ have the same occurrences of triggers in $\widetilde{h}$ and $\widetilde{m}$. Without loss of generality, we assume that $\left\{\widetilde{j}, \widetilde{n}\right\} \cap \mathit{fn}(\rho) = \emptyset$. By standard properties of substitution, $R_2^{\Theta^{\widetilde{j}}\Phi^{\widetilde{n}}}\sigma^{\Theta^{\widetilde{m}}\Phi^{\widetilde{h}}} = (R_2\rho\{\widetilde{m'}/\widetilde{m}\}^{\Theta^{\widetilde{m}}\frown\widetilde{m'}\Phi^{\widetilde{h}}\Theta^{\widetilde{k'}}})^{\Theta^{\widetilde{j}}\Phi^{\widetilde{n}}}$, where the vector $\widetilde{m'}$ is fresh. By definition of merge,

$$R_* = \langle\!\langle \Theta^{\widetilde{j}} \mid \Theta^{\widetilde{k'}} \mid \Theta^{\widetilde{m}\frown\widetilde{m'}} \mid \Phi^{\widetilde{n}} \mid (\nu\, \widetilde{b})(R_1 \mid !l{\cdot}a(\widetilde{\pi}).R_2 \mid R_2\rho\{\widetilde{m'}/\widetilde{m}\}^{\Phi^{\widetilde{h}}}) \rangle\!\rangle$$

By definition of lts,

$$K' \equiv (\nu\, \widetilde{c}, \widetilde{c'})(\Theta^{\widetilde{k}} \mid \langle\!\langle P_1 \mid \Theta^{\widetilde{m}} \mid \Phi^{\widetilde{h}} \rangle\!\rangle \mid R_* \mid \Phi^{\widetilde{i}})$$

By definition of lts,

$$K_1 \xrightarrow{(\widetilde{c'},\widetilde{k'})\overline{l{\cdot}a}\langle\widetilde{v'}\rangle} P_1 \mid \Theta^{\widetilde{j}} \mid \Theta^{\widetilde{m}} \mid \Theta^{\widetilde{k}} \mid \Theta^{\widetilde{k'}} = K_1'$$

where, as noted above, $\mathfrak{X}(\widetilde{v}) = (\widetilde{v'}; \Theta^{\widetilde{k'}})$. By $K_1 \approx_\Lambda L_1$,

$$L_1 \xrightarrow{\tau}\!\!\!\!\rightarrow (\nu\, \widetilde{c'})(Q_1 \mid \overline{l{\cdot}a}\langle\widetilde{v_2}\rangle) \mid \Omega^{\widetilde{j}} \mid \Omega^{\widetilde{m}} \mid \Omega^{\widetilde{k}} = L_2$$

$$L_2 \xrightarrow{(\widetilde{c'},\widetilde{k'})\overline{l{\cdot}a}\langle\widetilde{v'}\rangle} Q_1 \mid \Omega^{\widetilde{j}} \mid \Omega^{\widetilde{m}} \mid \Omega^{\widetilde{k}} \mid \Omega^{\widetilde{k'}} = L_3$$

where $\mathfrak{X}(\widetilde{v_2}) = (\widetilde{v'}; \Omega^{\widetilde{k'}})$, and

$$L_3 \xrightarrow{\tau}\!\!\!\!\rightarrow Q' \mid \Omega^{\widetilde{j}} \mid \Omega^{\widetilde{m}} \mid \Omega^{\widetilde{k}} \mid \Omega^{\widetilde{k'}} = L_1'$$

with $K_1' \approx_\Lambda L_1'$. By point 2 of Lemma B.12, since $Q \xrightarrow{\tau}\!\!\!\!\rightarrow (\nu\, \widetilde{c'})(Q_1 \mid \overline{l{\cdot}a}\langle\widetilde{v_2}\rangle)$,

$$\langle\!\langle Q \mid \Omega^{\widetilde{m}} \mid \Phi^{\widetilde{h}} \rangle\!\rangle \xrightarrow{\tau}\!\!\!\!\rightarrow \langle\!\langle (\nu\, \widetilde{c'})(Q_1 \mid \overline{l{\cdot}a}\langle\widetilde{v_2}\rangle) \mid \Omega^{\widetilde{m}} \mid \Phi^{\widetilde{h}} \rangle\!\rangle = (\nu\, \widetilde{c'})(Q_1^{\Phi^{\widetilde{h}}} \mid \overline{l{\cdot}a}\langle\widetilde{v_2}^{\Omega^{\widetilde{m}}\Phi^{\widetilde{h}}}\rangle)$$

By syntactical reasoning, $L_* \xrightarrow{\tau} \overset{\tau}{\longrightarrow} L''$, where

$$L'' = (\nu\, \widetilde{c}, \widetilde{c'})(\Omega^{\widetilde{k}} \mid \langle\!\langle Q_1 \mid \Omega^{\widetilde{m}} \mid \Phi^{\widetilde{h}} \rangle\!\rangle \mid R'_* \mid \Phi^{\widetilde{i}})$$

where, using an argument similar to that used for R_*,

$$R'_* = \langle\!\langle \Omega^{\widetilde{j}} \mid \Omega^{\widetilde{k'}} \mid \Omega^{\widetilde{m} \frown \widetilde{m'}} \mid \Phi^{\widetilde{n}} \mid (\nu\, \widetilde{b})(R_1 \mid !l{\cdot}a(\widetilde{\pi}).R_2 \mid R_2 \rho\{\widetilde{m'}/\widetilde{m}\}^{\Phi^{\widetilde{h}}}) \rangle\!\rangle$$

Note that R'_* is essentially R_* where each Θ is replaced by an Ω. By point 2 of Lemma B.12, since $Q_1 \xrightarrow{\tau} Q'$ we have that $L'' \xrightarrow{\tau} L'$, where

$$L' \equiv (\nu\, \widetilde{c}, \widetilde{c'})(\Omega^{\widetilde{k}} \mid \langle\!\langle Q' \mid \Omega^{\widetilde{m}} \mid \Phi^{\widetilde{h}} \rangle\!\rangle \mid R'_* \mid \Phi^{\widetilde{i}})$$

By $K'_1 \approx_\Delta L'_1$ and point 2 of Lemma B.13,

$$\Theta^{\widetilde{k}} \mid \Theta^{\widetilde{k'}} \mid \Theta^{\widetilde{m} \frown \widetilde{m'}} \mid \Theta^{\widetilde{m}} \mid \Theta^{\widetilde{j}} \mid P_1 \approx_\Delta \Omega^{\widetilde{k}} \mid \Omega^{\widetilde{k'}} \mid \Omega^{\widetilde{m} \frown \widetilde{m'}} \mid \Omega^{\widetilde{m}} \mid \Omega^{\widetilde{j}} \mid Q_1$$

and we conclude because $(K', L') \in \dot{\approx}_\Delta$.

Input by R: analogous to the previous case.

Input by P: By Lemma B.3, we have $R^{\Phi^{\widetilde{n}} \Theta^{\widetilde{j}}} \equiv (\nu\, \widetilde{c'})(R_1^{\Phi^{\widetilde{n}} \Theta^{\widetilde{j}}} \mid \overline{l{\cdot}a}\langle \widetilde{v}^{\Theta^{\widetilde{j}}} \rangle)$, where $\widetilde{c'}$ is fresh and $\{\widetilde{c'}\} \subseteq \mathit{fn}(\widetilde{v})$. Moreover, $P^{\Theta^{\widetilde{m}} \Phi^{\widetilde{h}}} \equiv (\nu\, \widetilde{b})(P_1^{\Theta^{\widetilde{m}} \Phi^{\widetilde{h}}} \mid l{\cdot}a(\widetilde{\pi}).P_2^{\Theta^{\widetilde{m}} \Phi^{\widetilde{h}}})$, where $\widetilde{b}$ is fresh and $\widetilde{v} = \widetilde{\pi}\sigma$ (since scripts and trigger names cannot appear in patterns). Additionally, $K_0 \equiv (\nu\, \widetilde{c'})((\nu\, \widetilde{b})(P_1^{\Theta^{\widetilde{m}} \Phi^{\widetilde{h}}} \mid P_2^{\Theta^{\widetilde{m}} \Phi^{\widetilde{h}}} \sigma^{\Theta^{\widetilde{j}}}) \mid R_1^{\Phi^{\widetilde{n}} \Theta^{\widetilde{j}}})$. In order to derive a K' of a suitable form, we follow a strategy similar to the one used in the case of replicated input by R. Let $\widetilde{v'} = \widetilde{v}\{\widetilde{j'}/\widetilde{j}\}$ for some vector of fresh triggers $\widetilde{j'}$. Since $\widetilde{v} = \widetilde{\pi}\sigma$ and $\widetilde{\pi}$ cannot contain triggers, $\widetilde{v'} = \widetilde{\pi}\sigma\{\widetilde{j'}/\widetilde{j}\}$. By standard properties of substitution, $\widetilde{v}^{\Theta^{\widetilde{j}}} = \widetilde{v'}^{\Theta^{\widetilde{j} \frown \widetilde{j'}}}$. Suppose $\mathfrak{X}(\widetilde{v'}) = (\widetilde{v_1}; \Phi^{\widetilde{h'}})$ for some fresh $\widetilde{h'}$ such that $\widetilde{v_1} = \widetilde{\pi}\rho$. By Lemma B.11 and by freshness of $\widetilde{j'}, \widetilde{h'}$, we have $\widetilde{v'} = \widetilde{v_1}^{\Phi^{\widetilde{h'}}}$, $\rho = \sigma\{\widetilde{j'}/\widetilde{j}\}^{\Phi^{\widetilde{h'}}}$ and $P_2^{\Theta^{\widetilde{m}} \Phi^{\widetilde{h}}} \sigma^{\Theta^{\widetilde{j}}} = P_2 \rho^{\Theta^{\widetilde{j} \frown \widetilde{j'}} \Phi^{\widetilde{h'}} \Theta^{\widetilde{m}} \Phi^{\widetilde{h}}}$. By definition of lts,

$$K' \equiv (\nu\, \widetilde{c}, \widetilde{c'})(\Theta^{\widetilde{k}} \mid \langle\!\langle (\nu\, \widetilde{b})(P_1 \mid P_2\rho) \mid \Theta^{\widetilde{m}} \mid \Theta^{\widetilde{j} \frown \widetilde{j'}} \mid \Phi^{\widetilde{h}} \mid \Phi^{\widetilde{h'}} \rangle\!\rangle \mid \langle\!\langle \Theta^{\widetilde{j}} \mid \Phi^{\widetilde{n}} \mid R_1 \rangle\!\rangle \mid \Phi^{\widetilde{i}})$$

By definition of lts,

$$K_1 \xrightarrow{l{\cdot}a(\widetilde{v_1})} \xrightarrow{\tau} (\nu\, \widetilde{b})(P_1 \mid P_2\rho) \mid \Omega^{\widetilde{j}} \mid \Theta^{\widetilde{m}} \mid \Theta^{\widetilde{k}} = K'_1$$

By $K_1 \approx_\Lambda L_1$ and composing the weak actions of L_1,

$$L_1 \xrightarrow{l{\cdot}a(\widetilde{v_1})} \Rightarrow Q' \mid \Omega^{\widetilde{j}} \mid \Omega^{\widetilde{m}} \mid \Omega^{\widetilde{k}} = L'_1$$

and $K'_1 \approx_\Lambda L'_1$. By syntactical reasoning,

$$Q \xrightarrow{\tau} \Rightarrow Q_1 \xrightarrow{l{\cdot}a(\widetilde{v_1})} Q_1 \mid \overline{l{\cdot}a}\langle \widetilde{v_1} \rangle \xrightarrow{\tau} \Rightarrow Q'$$

By syntactical reasoning and by point 2 of Lemma B.12,

$$L_* \xrightarrow{\tau} \twoheadrightarrow (\nu\, \widetilde{c})(\Omega^{\widetilde{k}} \mid \langle\!\langle Q_1 \mid \Omega^{\widetilde{m}} \mid \Phi^{\widetilde{h}} \rangle\!\rangle \mid \langle\!\langle \Omega^{\widetilde{j}} \mid \Phi^{\widetilde{n}} \mid R \rangle\!\rangle \mid \Phi^{\widetilde{i}}) = L''$$

By structural congruence, by $\widetilde{v'} = \widetilde{v_1}^{\,\Phi^{\widetilde{h'}}}$ and $\widetilde{v}^{\Omega^{\widetilde{j}}} = \widetilde{v'}^{\,\Omega^{\widetilde{j}} \frown \widetilde{j'}}$, and since $R^{\Phi^{\widetilde{n}}\Omega^{\widetilde{j}}} \equiv (\nu\, \widetilde{c'})(R_1^{\Phi^{\widetilde{n}}\Omega^{\widetilde{j}}} \mid \overline{l{\cdot}a}\langle \widetilde{v}^{\Omega^{\widetilde{j}}} \rangle)$,

$$L'' \equiv (\nu\, \widetilde{c}, \widetilde{c'})(\Omega^{\widetilde{k}} \mid \langle\!\langle Q_1 \mid \overline{l{\cdot}a}\langle \widetilde{v_1} \rangle \mid \Omega^{\widetilde{m}} \mid \Omega^{\widetilde{j} \frown \widetilde{j'}} \mid \Phi^{\widetilde{h}} \mid \Phi^{\widetilde{h'}} \rangle\!\rangle \mid \langle\!\langle \Omega^{\widetilde{j}} \mid \Phi^{\widetilde{n}} \mid R_1 \rangle\!\rangle \mid \Phi^{\widetilde{i}})$$

By point 2 of Lemma B.12, $L'' \xrightarrow{\tau} \twoheadrightarrow L'$ where

$$L' = (\nu\, \widetilde{c}, \widetilde{c'})(\Omega^{\widetilde{k}} \mid \langle\!\langle Q' \mid \Omega^{\widetilde{m}} \mid \Omega^{\widetilde{j} \frown \widetilde{j'}} \mid \Phi^{\widetilde{h}} \mid \Phi^{\widetilde{h'}} \rangle\!\rangle \mid \langle\!\langle \Omega^{\widetilde{j}} \mid \Phi^{\widetilde{n}} \mid R_1 \rangle\!\rangle \mid \Phi^{\widetilde{i}})$$

By point 2 of Lemma B.13, $K_1' \mid \Theta^{\widetilde{j} \frown \widetilde{j'}} \approx_\Lambda L_1' \mid \Omega^{\widetilde{j} \frown \widetilde{j'}}$. and we conclude because $(K', L') \in \dot{\approx}_\Delta$.

Replicated input by P: similar to the previous case.

Isolation. We show the case for R, as the case for P is similar. There are four ways to derive the sub-transition $\langle\!\langle \Theta^{\widetilde{j}} \mid \Phi^{\widetilde{n}} \mid R \rangle\!\rangle \xrightarrow{l{\cdot}\tau} M_1$.

- $R^{\Phi^{\widetilde{n}}\Theta^{\widetilde{j}}} \equiv (\nu\, \widetilde{a})(R_1^{\Phi^{\widetilde{n}}\Theta^{\widetilde{j}}} \mid l{\cdot}c(\widetilde{\pi}).R_2^{\Phi^{\widetilde{n}}\Theta^{\widetilde{j}}} \mid \overline{l{\cdot}c}\langle \widetilde{\pi}\sigma^{\Phi^{\widetilde{n}}\Theta^{\widetilde{j}}} \rangle)$ and

  $$M_1 \equiv (\nu\, \widetilde{a})(R_1^{\Phi^{\widetilde{n}}\Theta^{\widetilde{j}}} \mid (R_2^{\Phi^{\widetilde{n}}\Theta^{\widetilde{j}}})\sigma^{\Phi^{\widetilde{n}}\Theta^{\widetilde{j}}}) \equiv \langle\!\langle \Theta^{\widetilde{j}} \mid \Phi^{\widetilde{n}} \mid (\nu\, \widetilde{a})(R_1 \mid R_2\sigma) \rangle\!\rangle$$

  By syntactical reasoning, we can also derive

  $$\langle\!\langle \Omega^{\widetilde{j}} \mid \Phi^{\widetilde{n}} \mid R \rangle\!\rangle \equiv (\nu\, \widetilde{a})(R_1^{\Phi^{\widetilde{n}}\Omega^{\widetilde{j}}} \mid l{\cdot}c(\widetilde{\pi}).R_2^{\Phi^{\widetilde{n}}\Omega^{\widetilde{j}}} \mid \overline{l{\cdot}c}\langle \widetilde{\pi}\sigma^{\Phi^{\widetilde{n}}\Omega^{\widetilde{j}}} \rangle)$$

  $$\xrightarrow{l{\cdot}\tau} (\nu\, \widetilde{a})(R_1^{\Phi^{\widetilde{n}}\Omega^{\widetilde{j}}} \mid (R_2^{\Phi^{\widetilde{n}}\Omega^{\widetilde{j}}})\sigma^{\Phi^{\widetilde{n}}\Omega^{\widetilde{j}}}) = \langle\!\langle \Omega^{\widetilde{j}} \mid \Phi^{\widetilde{n}} \mid (\nu\, \widetilde{a})(R_1 \mid R_2\sigma) \rangle\!\rangle$$

  and we conclude because, by definition of lts, we can use this transition to derive a transition for L_* matching the one of K_*, with the resulting states K' and L' still in $\dot{\approx}_\Delta$.
- $R^{\Phi^{\widetilde{n}}\Theta^{\widetilde{j}}} \equiv (\nu\, \widetilde{a})(R_1^{\Phi^{\widetilde{n}}\Theta^{\widetilde{j}}} \mid !l{\cdot}c(\widetilde{\pi}).R_2^{\Phi^{\widetilde{n}}\Theta^{\widetilde{j}}} \mid \overline{l{\cdot}c}\langle \widetilde{\pi}\sigma^{\Phi^{\widetilde{n}}\Theta^{\widetilde{j}}} \rangle)$: analogous to the previous case.
- $R^{\Phi^{\widetilde{n}}\Theta^{\widetilde{j}}} \equiv (\nu\, \widetilde{a})(R_1^{\Phi^{\widetilde{n}}\Theta^{\widetilde{j}}} \mid m{\cdot}\mathtt{go}\ l.R_2^{\Phi^{\widetilde{n}}\Theta^{\widetilde{j}}})$: similar to the previous cases, although the reasoning on $R_2^{\Phi^{\widetilde{n}}\Theta^{\widetilde{j}}}$ is carried on at location l.
- $R^{\Phi^{\widetilde{n}}\Theta^{\widetilde{j}}} \equiv (\nu\, \widetilde{a})(R_1^{\Phi^{\widetilde{n}}\Theta^{\widetilde{j}}} \mid (x, \widetilde{\pi})R_2 \circ \langle l, \widetilde{\pi}\sigma^{\Phi^{\widetilde{n}}\Theta^{\widetilde{j}}} \rangle)$ and

  $$M_1 \equiv (\nu\, \widetilde{a})(R_1^{\Phi^{\widetilde{n}}\Theta^{\widetilde{j}}} \mid R_2\{{}^{l}/_{x}\}\sigma^{\Phi^{\widetilde{n}}\Theta^{\widetilde{j}}})$$

  where we have used the equation $R_2^{\Phi^{\widetilde{n}}\Theta^{\widetilde{j}}} = R_2$, which holds because scripts cannot contain trigger names. In order for $R^{\Phi^{\widetilde{n}}\Theta^{\widetilde{j}}}$ to have the form given above, R itself must have one of the three forms given below.

1. If $R \equiv (\nu\,\widetilde{a})(R_1 \mid (x,\widetilde{\pi})R_2 \circ \langle l, \widetilde{\pi}\sigma\rangle)$ then, by definition of script, $((x,\widetilde{\pi})R_2 \circ \langle l, \widetilde{\pi}\sigma\rangle)^{\Phi^{\widetilde{n}}\Theta^{\widetilde{j}}} = (x,\widetilde{\pi})R_2 \circ \langle l, \widetilde{\pi}\sigma^{\Phi^{\widetilde{n}}\Theta^{\widetilde{j}}}\rangle$ and the reasoning is similar to the cases above.
2. If $R \equiv (\nu\,\widetilde{a})(R_1 \mid k \circ \langle l, \widetilde{\pi}\sigma\rangle)$ where $\Phi^{\widetilde{n}} = \Phi^{\widetilde{n_1}} \mid \langle k \Leftarrow (x,\widetilde{\pi})R_2\rangle \mid \Phi^{\widetilde{n_2}}$ with $\widetilde{n_1}, k, \widetilde{n_2} = \widetilde{n}$, then we have

$$R^{\Phi^{\widetilde{n}}\Theta^{\widetilde{j}}} \equiv (\nu\,\widetilde{a})(R_1^{\Phi^{\widetilde{n}}\Theta^{\widetilde{j}}} \mid (x,\widetilde{\pi})R_2 \circ \langle l, \widetilde{\pi}\sigma^{\Phi^{\widetilde{n}}\Theta^{\widetilde{j}}}\rangle)$$
$$R^{\Phi^{\widetilde{n}}\Omega^{\widetilde{j}}} \equiv (\nu\,\widetilde{a})(R_1^{\Phi^{\widetilde{n}}\Omega^{\widetilde{j}}} \mid (x,\widetilde{\pi})R_2 \circ \langle l, \widetilde{\pi}\sigma^{\Phi^{\widetilde{n}}\Omega^{\widetilde{j}}}\rangle)$$

   and the reasoning is once again analogous to case 1.
3. The last and most interesting case arises if $R \equiv (\nu\,\widetilde{a})(R_1 \mid k \circ \langle l, \widetilde{\pi}\sigma\rangle)$ and $\Theta^{\widetilde{j}} = \Theta^{\widetilde{j_1}} \mid \langle k \Leftarrow (x,\widetilde{\pi})R_2\rangle \mid \Theta^{\widetilde{j_2}}$, where $\widetilde{j_1}, k, \widetilde{j_2} = \widetilde{j}$. In this case, a tau transition by K_* corresponds to a (Lts Def) transition by K_1. Without loss of generality, we assume that the names $\widetilde{a}$ are fresh. In order to derive an appropriate K', we need to be explicit about the scripts in $\widetilde{\pi}\sigma$. Hence, suppose that $\mathfrak{X}(\widetilde{\pi}\sigma) = (\widetilde{\pi}\rho; \Phi^{\widetilde{n'}})$ where $\widetilde{n'}$ is fresh.
   By definition of lts, $K_* \xrightarrow{\tau} K'$ where

$$K' = (\nu\,\widetilde{c})(\Theta^{\widetilde{k}} \mid \langle\!\langle P \mid \Theta^{\widetilde{m}} \mid \Phi^{\widetilde{h}}\rangle\!\rangle \mid \langle\!\langle (\nu\,\widetilde{a})(R_1 \mid R_2\{^l/_x\}\rho) \mid \Phi^{\widetilde{n'}} \mid \Phi^{\widetilde{n}} \mid \Theta^{\widetilde{j}}\rangle\!\rangle \mid \Phi^{\widetilde{i}})$$

   Since R_2 comes from the definition $\langle k \Leftarrow (x,\widetilde{\pi})R_2\rangle$ which is part of $\Theta^{\widetilde{j}}$ (hence was originated by some previous transition of P), we need to move $R_2R_2\{^l/_x\}\rho$ inside the leftmost merge operator.

$$K' \equiv (\nu\,\widetilde{c},\widetilde{a})(\Theta^{\widetilde{k}} \mid \langle\!\langle P \mid R_2\{^l/_x\}\rho \mid \Theta^{\widetilde{m}} \mid \Phi^{\widetilde{h}} \mid \Phi^{\widetilde{n'}}\rangle\!\rangle \mid \langle\!\langle R_1 \mid \Phi^{\widetilde{n}} \mid \Theta^{\widetilde{j}}\rangle\!\rangle \mid \Phi^{\widetilde{i}})$$

   With K' of this form, we will derive a transition from K_1 to a suitable K_1', and use the bisimilarity hypothesis to derive a matching transition for L_*. By definition of lts,

$$K_1 \xrightarrow{l\cdot k(\widetilde{\pi}\rho)} \Theta^{\widetilde{k}} \mid \Theta^{\widetilde{m}} \mid \Theta^{\widetilde{j}} \mid P \mid R_2\{^l/_x\}\rho = K_1'$$

   By $K_1 \approx_\Lambda L_1$ and by point 3 of Lemma B.12 we know that $\Omega^{\widetilde{j}} = \Omega^{\widetilde{j_1}} \mid \langle k \Leftarrow (x,\widetilde{\pi}')R_3\rangle \mid \Omega^{\widetilde{j_2}}$. Using our standard reasoning on the hypothesis $K_1 \approx_\Lambda L_1$, we can derive the transition

$$L_1 \xrightarrow{l\cdot k(\widetilde{\pi}\rho)}\!\!\!\!\rightarrow \Omega^{\widetilde{k}} \mid \Omega^{\widetilde{m}} \mid \Omega^{\widetilde{j}} \mid Q' = L_1' \approx_\Lambda K_1'$$

   Breaking down the transition into $\xrightarrow{\tau}\!\!\!\!\rightarrow \xrightarrow{l\cdot k(\widetilde{\pi}\rho)}\!\!\!\!\rightarrow \xrightarrow{\tau}\!\!\!\!\rightarrow$ and using point 2 of Lemma B.12, we obtain

$$L_* \xrightarrow{\tau}\!\!\!\!\rightarrow L' = (\nu\,\widetilde{c},\widetilde{a})(\Omega^{\widetilde{k}} \mid \langle\!\langle Q' \mid \Omega^{\widetilde{m}} \mid \Phi^{\widetilde{h}} \mid \Phi^{\widetilde{n'}}\rangle\!\rangle \mid \langle\!\langle R_1 \mid \Phi^{\widetilde{n}} \mid \Omega^{\widetilde{j}}\rangle\!\rangle \mid \Phi^{\widetilde{i}})$$

   and hence $(K', L') \in \dot{\approx}_\Delta$.

□

We can finally prove that domain bisimilarity is a congruence on both configurations and, more importantly, open processes. The extension to open processes does not involve significant difficulties because, by definition, bisimilarity for open processes is closed under arbitrary substitutions.

Theorem B.15 (Congruence) *Domain bisimilarity is a congruence:*

1. *for all configuration contexts $C \in \mathcal{K}_{\mathcal{W}}$ (Figure 11), if $K \approx_\Lambda L$ then $C[K] \approx_\Lambda C[L]$;*
2. *for all extended process contexts $C \in \mathcal{K}_{\mathrm{f}}$ (Definition 3.5), if $\boldsymbol{P} \approx_\Lambda \boldsymbol{Q}$ then $C[\boldsymbol{P}] \approx_\Lambda C[\boldsymbol{Q}]$.*

Proof.

1. By Lemma B.10, $K \approx_\Lambda L \implies (\nu\, \widetilde{c})K \approx_\Lambda (\nu\, \widetilde{c})L$. By Lemma B.14, $K \approx_\Lambda L \implies K \mid M \approx_\Lambda L \mid M$. By Proposition B.1, $K \mid M \approx_\Lambda L \mid M \implies M \mid K \approx_\Lambda M \mid L$.

2. We need to show that

   (a) $\boldsymbol{P} \approx_\Lambda \boldsymbol{Q} \implies (\nu\, \widetilde{c})\boldsymbol{P} \approx_\Lambda (\nu\, \widetilde{c})\boldsymbol{Q}$;
   (b) $\boldsymbol{P} \approx_\Lambda \boldsymbol{Q} \implies \boldsymbol{P} \mid \boldsymbol{R} \approx_\Lambda \boldsymbol{Q} \mid \boldsymbol{R}$;
   (c) $\boldsymbol{P} \approx_\Lambda \boldsymbol{Q} \implies \boldsymbol{l}{\cdot}\boldsymbol{c}(\widetilde{\pi}).\boldsymbol{P} \approx_\Lambda \boldsymbol{l}{\cdot}\boldsymbol{c}(\widetilde{\pi}).\boldsymbol{Q}$;
   (d) $\boldsymbol{P} \approx_\Lambda \boldsymbol{Q} \implies !\boldsymbol{l}{\cdot}\boldsymbol{c}(\widetilde{\pi}).\boldsymbol{P} \approx_\Lambda !\boldsymbol{l}{\cdot}\boldsymbol{c}(\widetilde{\pi}).\boldsymbol{Q}$;
   (e) $\boldsymbol{P} \approx_\Lambda \boldsymbol{Q} \implies \boldsymbol{l}{\cdot}\mathsf{go}\,\boldsymbol{m}.\boldsymbol{P} \approx_\Lambda \boldsymbol{l}{\cdot}\mathsf{go}\,\boldsymbol{m}.\boldsymbol{Q}$.

   By definition $\boldsymbol{P} \approx_\Lambda \boldsymbol{Q}$ if and only if $\boldsymbol{P}\sigma \approx_\Lambda \boldsymbol{Q}\sigma$ for all closing substitutions σ.

   (a) Consider an arbitrary closing substitution σ for $\boldsymbol{P}, \boldsymbol{Q}$. Since we assume substitutions to be capture avoiding, $((\nu\, \widetilde{c})\boldsymbol{P})\sigma \equiv (\nu\, \widetilde{c'})(\boldsymbol{P}\{^{c'}/_{c}\}\sigma)$ and $((\nu\, \widetilde{c})\boldsymbol{Q})\sigma \equiv (\nu\, \widetilde{c'})(\boldsymbol{Q}\{^{c'}/_{c}\}\sigma)$. By hypothesis, $\boldsymbol{P}\{^{c'}/_{c}\}\sigma \approx_\Lambda \boldsymbol{Q}\{^{c'}/_{c}\}\sigma$. By point 1 above, $(\nu\, \widetilde{c'})(\boldsymbol{P}\{^{c'}/_{c}\}\sigma) \approx_\Lambda (\nu\, \widetilde{c'})(\boldsymbol{Q}\{^{c'}/_{c}\}\sigma)$.
   (b) Similar to point 2a, using point 1.
   (c) Let the family $\dot{\approx}$ have the generic element

   $$\dot{\approx}_\Delta = \approx_\Delta \cup \{(\boldsymbol{l}{\cdot}\boldsymbol{c}(\widetilde{\pi}).\boldsymbol{P}\sigma \mid M, \boldsymbol{l}{\cdot}\boldsymbol{c}(\widetilde{\pi}).\boldsymbol{Q}\sigma \mid M) \;\; : \;\; \boldsymbol{P}\sigma \approx_\Delta \boldsymbol{Q}\sigma\}$$

   where σ is a closing substitution, $M = \prod_{n \geq 0} \overline{l_i{\cdot}c_i}\langle \widetilde{v}_i \rangle$ and $\mathit{scripts}(\widetilde{v}_i) = \emptyset$.

   The thesis follows by showing that $\dot{\approx}$ is a domain bisimulation, using point 1.
   (d) Similar to point 2c, using also transitivity of $\approx_\Delta$ (Lemma B.9).

(e) Let the family $\dot{\approx}$ have the generic element $\dot{\approx}_\Delta$ given by

$$\text{if } m \notin \Delta\text{: } \dot{\approx}_\Delta = \approx_\Delta \cup \{(\boldsymbol{l}\cdot\mathsf{go}\,\boldsymbol{m}.\boldsymbol{P}\sigma \mid M, \boldsymbol{l}\cdot\mathsf{go}\,\boldsymbol{m}.\boldsymbol{Q}\sigma \mid M) \;:\; \boldsymbol{P}\sigma \approx_\Delta \boldsymbol{Q}\sigma\}$$

$$\text{if } m \in \Delta\text{: } \dot{\approx}_\Delta = \approx_\Delta \cup \left\{ \begin{array}{r} (\boldsymbol{l}\cdot\mathsf{go}\,\boldsymbol{m}.\boldsymbol{P}\sigma \mid M, \boldsymbol{l}\cdot\mathsf{go}\,\boldsymbol{m}.\boldsymbol{Q}\sigma \mid M), \\ (\boldsymbol{P}\sigma \mid M, \boldsymbol{l}\cdot\mathsf{go}\,\boldsymbol{m}.\boldsymbol{Q}\sigma \mid M), \\ (\boldsymbol{l}\cdot\mathsf{go}\,\boldsymbol{m}.\boldsymbol{P}\sigma \mid M, \boldsymbol{Q}\sigma \mid M) \end{array} \;:\; \boldsymbol{P}\sigma \approx_\Delta \boldsymbol{Q}\sigma \right\}$$

where $\mathit{dom}(M) \subseteq \Delta$, σ is a closing substitution, $M = \prod_{n\geq 0} \overline{l_i\cdot c_i}\langle \widetilde{v}_i \rangle$ and $\mathit{scripts}(\widetilde{v}_i) = \emptyset$. The family $\dot{\approx}$ is a domain bisimulation.

□

B.3 Soundness

In this section, we show *soundness*: if two processes are domain bisimilar with respect to Λ, then they are request congruent with respect to Λ. Our strategy for proving the soundness of $\approx_\Lambda$ consists of three main steps. First, we define an auxiliary relation $\asymp$ on Core X$d\pi$ networks such that two networks are in the relation if the corresponding processes, in parallel with the definitions extracted from the scripts in the corresponding stores, are Λ-bisimilar. Second we show that $\asymp$ is included in $\sim^\Lambda$, and third we use $\asymp$ as a stepping stone to relate $\approx_\Lambda$ with $\sim^\Lambda$.

We begin comparing reductions and transitions. If a configuration K can perform a tau transition to become K', then the process $\langle\!\langle K \rangle\!\rangle$ obtained by merging the configuration can reduce to $\langle\!\langle K' \rangle\!\rangle$, for any store compatible with K. On the other hand, if a process does a reduction step then, according to the lts, it can either perform a request transition or a tau transition, depending on whether (CRed Request) was used in the derivation.

Lemma B.16 (Reductions) *Tau transitions between configurations imply reductions between the corresponding networks. For all D, K such that $\mathit{dom}(K) \subseteq \mathit{dom}(D)$*

1. *if $K \xrightarrow{l\cdot\tau} K'$ then $(D, \langle\!\langle K \rangle\!\rangle) \to (D, \langle\!\langle K' \rangle\!\rangle)$;*
2. *if $K \xrightarrow{\tau} \!\!\!\!\to_\Lambda K'$ then $(D, \langle\!\langle K \rangle\!\rangle) \to^* (D, \langle\!\langle K' \rangle\!\rangle)$.*

Proof. Point 1 follows from point 2 of Lemma B.12. Point 2 follows from point 1 noticing that tau transitions do not increase the domain of a configuration. □

Lemma B.17 (Transitions) *Reductions between networks imply tau or request transitions between the corresponding configurations. If $(D, P) \longrightarrow (D_1, P_1)$ then one of the following holds:*

1. *$P \xrightarrow{l\cdot\tau} P_1$ and $D = \{l \mapsto T\} \uplus E = D_1$;*

2. $P \xrightarrow{(\widetilde{k})l\cdot \mathsf{req}\langle p'\rangle(T')} P_2 \,|\, \Theta^{\widetilde{k}}$ *and* $\begin{cases} P_1 \equiv \langle\!\langle P_2 \,|\, \Theta^{\widetilde{i}} \rangle\!\rangle, \\ D = \{l \mapsto T\} \uplus E, \\ D_1 = \{l \mapsto T_1\} \uplus E, \end{cases}$ *where there exists a path* p *such that* $\begin{cases} \mathfrak{E}(p, T) = (T_1, U_1 \shortmid ... \shortmid U_n \shortmid \varnothing), \\ \mathfrak{X}(p) = (p'; \Theta^{\widetilde{k}}), \\ \mathfrak{X}(\mathtt{r}[\,U_1\,] \shortmid \ldots \shortmid \mathtt{r}[\,U_n\,] \shortmid \varnothing) = (T'; \Theta^{\widetilde{i}}). \end{cases}$

Proof. Both points follow by induction on the depth of the derivation tree of $(D, P) \to (D_1, P_1)$, using points 3 and 4 of Lemma B.3 to derive the labelled transition from the structure of the processes as revealed by the reduction step. □

We know by Definition 3.8 that a script-independent query language, starting from queries and input trees which are equivalent up-to substitutions of scripts for trigger names, gives equivalent output trees and results. The lemma below relates this notion to extraction.

Lemma B.18 (Extraction) *Consider an arbitrary script-independent query language. Suppose* $\mathfrak{X}(T) = (T_0; \Theta^{\widetilde{k}})$, $\mathfrak{X}(p) = (p'; \Theta^{\widetilde{j}})$, $\mathfrak{E}(p, T) = (T_1, U_1 \shortmid ... \shortmid U_n \shortmid \varnothing)$, $\mathfrak{X}(\mathtt{r}[\,U_1\,] \shortmid ... \shortmid \mathtt{r}[\,U_n\,] \shortmid \varnothing) = (T'; \Theta^{\widetilde{i}})$ *and* $\mathfrak{X}(T_1) = (T_1'; \Theta^{\widetilde{h}})$.

1. *for any definition* $\langle k \Leftarrow A \rangle$ *occurring in* $\Theta^{\widetilde{i}}$ *or* $\Theta^{\widetilde{h}}$ *there must be a definition* $\langle k' \Leftarrow A \rangle$ *occurring in* $\Theta^{\widetilde{k}}$ *or* $\Theta^{\widetilde{j}}$.
2. *if* $\mathfrak{X}(S) = (T_0; \Omega^{\widetilde{k}})$ *and* $\mathfrak{X}(q) = (p'; \Omega^{\widetilde{j}})$, *then* $\mathfrak{E}(q, S) = (S_1; V_1 \shortmid ... \shortmid V_n \shortmid \varnothing)$, $\mathfrak{X}(\mathtt{r}[\,V_1\,] \shortmid \ldots \shortmid \mathtt{r}[\,V_n\,] \shortmid \varnothing) = (T'; \Omega^{\widetilde{i}})$ *and* $\mathfrak{X}(S_1) = (T_1'; \Omega^{\widetilde{h}})$.

Proof. Both points follow easily by Definition 3.8 and Observation 3.10. □

We need to compare domain bisimilarity, which is defined on configurations without taking the store into account, with domain congruence, which is defined using reduction congruence (a relation on networks). We can do this because bisimilarity requires a correspondence between matching actions of two configurations, which implies that the stores in the networks corresponding to the configurations can diverge, after each reduction step, only up-to equivalent scripts. To formalize this intuition, we introduce the relation $\asymp$ on networks.

Definition B.19 (Candidate Relation) *We define the* candidate relation $\asymp$ *by*

$$\asymp \stackrel{\text{def}}{=} \left\{ ((D, P), (B, Q)) \;\; : \;\; \mathfrak{X}(D) = (D'; \Theta^{\widetilde{k}}),\ \mathfrak{X}(B) = (D'; \Omega^{\widetilde{k}}),\ P \,|\, \Theta^{\widetilde{k}} \approx_\Lambda Q \,|\, \Omega^{\widetilde{k}} \right\}$$

where $\mathit{dom}(B) = \mathit{dom}(D) = \Lambda$ *and* $(\mathit{fn}(P) \cup \mathit{fn}(Q)) \cap \mathcal{Y} = \emptyset$.

We can show now that the candidate relation $\asymp$ is sound with respect to $\simeq$, the relation on networks inducing request congruence.

Lemma B.20 *The candidate relation is contained in the reduction congruence induced by request observables:* $\asymp \subseteq \simeq$.

Proof. By definition of $\simeq$, we need to show that $\asymp$ is (1) observation preserving, (2) contextual, and (3) reduction-closed.

1. Follows from the definition of request observables, the hypothesis $P \mid \Theta^{\widetilde{k}} \approx_\Lambda Q \mid \Omega^{\widetilde{k}}$ and Lemma B.16, noticing that $\Theta^{\widetilde{k}}$ and $\Omega^{\widetilde{k}}$ cannot perform tau or request transitions.

2. Consider a generic reduction context $(E \uplus -, (\nu\, \widetilde{c})(R \mid -))$. By definition of $\mathfrak{X}$, $\mathfrak{X}(E \uplus D) = (E' \uplus D'; \Phi^{\widetilde{j}} \mid \Theta^{\widetilde{k}})$ and $\mathfrak{X}(E \uplus B) = (E' \uplus D'; \Phi^{\widetilde{j}} \mid \Omega^{\widetilde{k}})$. By hypothesis, $P \mid \Theta^{\widetilde{k}} \approx_\Lambda Q \mid \Omega^{\widetilde{k}}$. By Theorem B.15, $(\nu\, \widetilde{c})(R \mid P \mid \Theta^{\widetilde{k}} \mid \Phi^{\widetilde{j}}) \approx_\Lambda (\nu\, \widetilde{c})(R \mid Q \mid \Omega^{\widetilde{k}} \mid \Phi^{\widetilde{j}})$. Since scripts have no private channel names, by structural congruence we conclude that $(\nu\, \widetilde{c})(R \mid P) \mid \Theta^{\widetilde{k}} \mid \Phi^{\widetilde{j}} \approx_\Lambda (\nu\, \widetilde{c})(R \mid Q) \mid \Omega^{\widetilde{k}} \mid \Phi^{\widetilde{j}}$.

3. Suppose $(D, P) \asymp (B, Q)$ and $(D, P) \longrightarrow (D_1, P_1)$. We need to show that $(B, Q) \stackrel{*}{\longrightarrow} (B_1, Q_1) \asymp (D_1, P_1)$. For convenience, we report below what $(D, P) \asymp (B, Q)$ means:

$$\mathfrak{X}(D) = (D'; \Theta^{\widetilde{k}}) \tag{1}$$

$$\mathfrak{X}(B) = (D'; \Omega^{\widetilde{k}}) \tag{2}$$

$$P \mid \Theta^{\widetilde{k}} \approx_\Lambda Q \mid \Omega^{\widetilde{k}} \tag{3}$$

$$\mathit{dom}(B) = \mathit{dom}(D) = \Lambda \tag{4}$$

   By Lemma B.17, there are two cases:

   1. $P \xrightarrow{l \cdot \tau} P_1$ and $D = \{l \mapsto T\} \uplus E = D_1$.
      By definition of lts and by equation (3) above,

$$\begin{array}{ccc} P \mid \Theta^{\widetilde{k}} & \xrightarrow{\quad l \cdot \tau \quad} & P_1 \mid \Theta^{\widetilde{k}} \\ \approx_\Lambda \big\downarrow & & \big\uparrow \approx_\Lambda \\ Q \mid \Omega^{\widetilde{k}} \xrightarrow{\tau}\!\!\!\!\rightarrow_\Lambda & \xrightarrow{\quad l \cdot \tau \quad} & \xrightarrow{\tau}\!\!\!\!\rightarrow_\Lambda Q_1 \mid \Omega^{\widetilde{k}} \end{array}$$

      By syntactical reasoning, $Q \xrightarrow{l \cdot \tau}\!\!\!\!\rightarrow_\Lambda Q_1$.
      By Lemma B.16, $(B, Q) \rightarrow^* (B, Q_1)$. By (1), (2) and (4) we conclude with

$$\begin{array}{ccc} (D, P) & \longrightarrow & (D, P_1) \\ \asymp \big\downarrow & & \big\uparrow \asymp \\ (B, Q) & \stackrel{*}{\longrightarrow} & (B, Q_1) \end{array}$$

2. $P \xrightarrow{(\widetilde{j})l\cdot\mathsf{req}\langle p'\rangle(T')} P_2 \,|\, \Theta^{\widetilde{j}}$ where

$$P_1 \equiv \langle\!\langle P_2 \,|\, \Theta^{\widetilde{i}} \rangle\!\rangle,$$
$$D = \{l \mapsto T\} \uplus E,$$
$$D_1 = \{l \mapsto T_1\} \uplus E$$

for some p such that

$$\mathfrak{E}(p, T) = (T_1, U_1 \,!\, \ldots \,!\, U_n \,!\, \varnothing),$$
$$\mathfrak{X}(p) = (p'; \Theta^{\widetilde{j}}),$$
$$\mathfrak{X}(\mathtt{r}[\,U_1\,]\,! \ldots !\,\mathtt{r}[\,U_n\,]\,!\,\varnothing) = (T'; \Theta^{\widetilde{i}}).$$

By definition of lts and by (3),

$$\begin{array}{ccc} P \,|\, \Theta^{\widetilde{k}} & \xrightarrow{(\widetilde{j})l\cdot\mathsf{req}\langle p'\rangle(T')} & P_2 \,|\, \Theta^{\widetilde{j}} \,|\, \Theta^{\widetilde{k}} \\ \approx_\Lambda \big\downarrow & & \big\uparrow \approx_\Lambda \\ Q \,|\, \Omega^{\widetilde{k}} \xrightarrow{\tau}\!\!\!\!\rightarrow_\Lambda & \xrightarrow{(\widetilde{j})l\cdot\mathsf{req}\langle p'\rangle(T')} & \xrightarrow{\tau}\!\!\!\!\rightarrow_\Lambda Q_2 \,|\, \Omega^{\widetilde{j}} \,|\, \Omega^{\widetilde{k}} \end{array} \tag{5}$$

By syntactical reasoning and point 4 of Lemma B.3,

$$Q \xrightarrow{\tau}\!\!\!\!\rightarrow_\Lambda (\nu\, a)(Q_3 \,|\, l\cdot\mathsf{req}_q\langle c\rangle) = Q_M \tag{6}$$

$$Q_M \xrightarrow{(\widetilde{j})l\cdot\mathsf{req}\langle p'\rangle(T')} (\nu\, a)(Q_3 \,|\, \overline{l\cdot c}\langle T'\rangle) \,|\, \Omega^{\widetilde{j}} \xrightarrow{\tau}\!\!\!\!\rightarrow_\Lambda Q_2 \,|\, \Omega^{\widetilde{j}} \tag{7}$$

for some q such that $\mathfrak{X}(q) = (p'; \Omega^{\widetilde{j}})$. By (6) and Lemma B.16,

$$(B, Q) \longrightarrow^* (B, Q_M).$$

By (1) and definition of $\mathfrak{X}$,

$$D' = \{l \mapsto T_0\} \uplus E_0 \;\; \textit{and} \;\; \Theta^{\widetilde{k}} = \Theta^{\widetilde{k'}} \,|\, \Theta^{\widetilde{k''}}$$

where $\mathfrak{X}(\{l \mapsto T\}) = (\{l \mapsto T_0\}; \Theta^{\widetilde{k'}})$ and $\mathfrak{X}(E) = (E_0; \Theta^{\widetilde{k''}})$.
By by (2) and a similar argument,

$$B = \{l \mapsto S\} \uplus E' \;\; \textit{and} \;\; \Omega^{\widetilde{k}} = \Omega^{\widetilde{k'}} \,|\, \Omega^{\widetilde{k''}}$$

where $\mathfrak{X}(\{l \mapsto S\}) = (\{l \mapsto T_0\}; \Omega^{\widetilde{k'}})$ and $\mathfrak{X}(E') = (E_0; \Omega^{\widetilde{k''}})$.
Suppose $\mathfrak{X}(T_1) = (T_1'; \Theta^{\widetilde{h}})$. By point 2 of Lemma B.18,

$$\mathfrak{E}(q, S) = (S_1; V_1 \,!\, \ldots \,!\, V_n \,!\, \varnothing),$$
$$\mathfrak{X}(\mathtt{r}[\,V_1\,]\,! \ldots !\,\mathtt{r}[\,V_n\,]\,!\,\varnothing) = (T'; \Omega^{\widetilde{i}}),$$
$$\mathfrak{X}(S_1) = (T_1'; \Omega^{\widetilde{h}}).$$

By definition of reduction,

$$(B, Q_M) \longrightarrow (B_1, (\nu\, a)(Q_3 \mid \overline{l{\cdot}c}\langle \mathtt{r}[\, V_1\,]_{\mathtt{l}} \ldots {}_{\mathtt{l}}\mathtt{r}[\, V_n\,]_{\mathtt{l}}\varnothing\rangle))$$

By point 2 of Lemma B.17,

$$(\nu\, a)(Q_3 \mid \overline{l{\cdot}c}\langle \mathtt{r}[\, V_1\,]_{\mathtt{l}} \ldots {}_{\mathtt{l}}\mathtt{r}[\, V_n\,]_{\mathtt{l}}\varnothing\rangle) \equiv \langle\!\langle (\nu\, a)(Q_3 \mid \overline{l{\cdot}c}\langle T'\rangle) \mid \Omega^{\widetilde{i}}\rangle\!\rangle$$

By syntactical reasoning on (7),

$$(\nu\, a)(Q_3 \mid \overline{l{\cdot}c}\langle T'\rangle) \xrightarrow{\tau} \!\!\!\!\rightarrow_{\Lambda} Q_2.$$

By point 2 of Lemma B.12,

$$\langle\!\langle (\nu\, a)(Q_3 \mid \overline{l{\cdot}c}\langle T'\rangle) \mid \Omega^{\widetilde{i}}\rangle\!\rangle \xrightarrow{\tau} \!\!\!\!\rightarrow_{\Lambda} \langle\!\langle Q_2 \mid \Omega^{\widetilde{i}}\rangle\!\rangle.$$

By Lemma B.16,

$$(B_1, \langle\!\langle (\nu\, a)(Q_3 \mid \overline{l{\cdot}c}\langle T'\rangle) \mid \Omega^{\widetilde{i}}\rangle\!\rangle) \longrightarrow^{*} (B_1, \langle\!\langle Q_2 \mid \Omega^{\widetilde{i}}\rangle\!\rangle)$$

By two applications of point 1 of Lemma B.18, for any definition $\langle i \Leftarrow A\rangle$ occurring in $\Theta^{\widetilde{i}}$ or $\Theta^{\widetilde{h}}$ there must be a definition $\langle k' \Leftarrow A\rangle$ occurring in $\Theta^{\widetilde{k}}$ or $\Theta^{\widetilde{j}}$.
By (5) and by Lemma B.13,

$$P_2 \mid \Theta^{\widetilde{h}} \mid \Theta^{\widetilde{k''}} \mid \Theta^{\widetilde{i}} \approx_{\Lambda} Q_2 \mid \Omega^{\widetilde{h}} \mid \Omega^{\widetilde{k''}} \mid \Omega^{\widetilde{i}}.$$

By using an appropriate instance of the candidate bisimulation in the proof of Lemma B.14,

$$\langle\!\langle P_2 \mid \Theta^{\widetilde{i}}\rangle\!\rangle \mid \Theta^{\widetilde{h}} \mid \Theta^{\widetilde{k''}} \approx_{\Lambda} \langle\!\langle Q_2 \mid \Omega^{\widetilde{i}}\rangle\!\rangle \mid \Omega^{\widetilde{h}} \mid \Omega^{\widetilde{k''}},$$

and we conclude with

$$\begin{array}{ccc} (D, P) & \longrightarrow & (D_1, \langle\!\langle P_2 \mid \Theta^{\widetilde{i}}\rangle\!\rangle) \\ \asymp \big\downarrow & & \big\uparrow \asymp \\ (B, Q) & \longrightarrow^{*} & (B_1, \langle\!\langle Q_2 \mid \Omega^{\widetilde{i}}\rangle\!\rangle) \end{array}$$

□

We have now all the tools necessary to show the soundness of domain bisimilarity with respect to request congruence.

Theorem B.21 (Soundness) *Domain bisimilarity is a sound approximation of the domain congruence induced by request observables: for all $\Lambda, \boldsymbol{P}, \boldsymbol{Q}$ where $(\mathit{fn}(\boldsymbol{P}) \cup \mathit{fn}(\boldsymbol{Q})) \cap \mathcal{Y} = \emptyset$, if $\boldsymbol{P} \approx_{\Lambda} \boldsymbol{Q}$ then $\boldsymbol{P} \sim^{\Lambda} \boldsymbol{Q}$.*

Proof. By definition, $\boldsymbol{P} \sim^{\Lambda} \boldsymbol{Q}$ if and only if $(D, C[\boldsymbol{P}]) \simeq (D, C[\boldsymbol{Q}])$ for all $D, C[-]$ such that $\Lambda \subseteq \mathit{dom}(D)$ and $C[-]$ does not contain trigger names and is closing for both $\boldsymbol{P}$ and $\boldsymbol{Q}$. Suppose $\boldsymbol{P} \approx_{\Lambda} \boldsymbol{Q}$ and consider some arbitrary $D, C[-]$ respecting the conditions above. Suppose $\mathfrak{X}(D) = (D'; \Theta^{\widetilde{k}})$. By point 2 of Theorem B.15, $C[\boldsymbol{P}] \,|\, \Theta^{\widetilde{k}} \approx_{\Lambda} C[\boldsymbol{Q}] \,|\, \Theta^{\widetilde{k}}$. By Definition B.19, $(D, C[\boldsymbol{P}]) \asymp (D, C[\boldsymbol{Q}])$. By Lemma B.20, $(D, C[\boldsymbol{P}]) \simeq (D, C[\boldsymbol{Q}])$. □