University of Sussex

Computer Science Report

Location and Link Failure in a Distributed π-calculus

Adrian Francalanza
Matthew Hennessy

Report 2005:01 January 2005

Department of Informatics
University of Sussex
Brighton BN1 9QH

ISSN 1350–3170

Location and Link Failure in a Distributed π-calculus

ADRIAN FRANCALANZA and MATTHEW HENNESSY

ABSTRACT. We develop a behavioural theory of distributed systems in the presence of failures. The framework we use is that of Dπ, a language in which located processes, or agents, may migrate between dynamically created locations. These processes run on a distributed network, in which individual nodes may fail, or the links between them may be broken. The original language, Dπ, is extended by a new construct for detecting and reacting to these failures together with constructs that induce failure.

We define a bisimulation equivalence between these systems, based on labelled actions which record, in addition to the effect actions have on the processes, the actual state of the underlying network and the view of this state known to observers. We prove that the equivalence is *fully abstract*, in the sense that two systems will be differentiated if and only if, in some sense, there is a computational context, consisting of a network and an observer, which can see the difference.

Contents

1 Introduction

It is generally accepted that location transparency is not attainable over *wide-area networks*, [4], large computational infrastructures which may even span the globe. Because of this, various *location-aware* calculi and programming languages have arisen in the literature; not only do these emphasise the *distributed* nature of systems but they also assume that the various system components, processes or agents, are aware of their location in the network, and perhaps, also aware of some aspect of the underlying network topology. In these languages, computations take place at distinct locations, physical or virtual, and processes may migrate between the locations of which they are aware, to participate in such computations.

It is also argued in [4] that *failures*, and the ability to react to them, are also an inevitable facet of these infrastructures, which must be taken into account when designing languages for location-aware computation. The purpose of this paper is to:

- invent a simple framework, a distributed process calculus, for describing computations over a distributed network where individual nodes and links between the nodes are subject to failures.
- use this framework to develop a behavioural theory of distributed systems where these failures are taken into account.

Variants of this problem have already been addressed in a number of papers, such as the pioneering [2, 1] and subsequently [16]. In [2, 1] they seek to understand the behavioural effects of site failure by translating a location-aware calculus in which site failure is explicitly represented, into a *location-free* calculus while in [16], they seek to develop a behavioural theory directly in terms of the location-aware calculus itself. Our work can be seen as an extension of [16] where we treat more general forms of failure.

Our framework is an extension of the distributed calculus Dπ [11], in which system configurations now take the form

$$\Pi \triangleright N$$

where Π is a representation of the current state of the network, and N describes the current state of the (software) system executing in a distributed manner over the network. There is, of course, an enormous range of possibilities for Π, depending on which aspects of networks we wish to model. Here, we treat two representative examples: In this first, Π simply records the *status* of nodes in the network, that is, whether they are *alive* or *dead*; this corresponds more or less to the framework used in [2, 1] and [16]. In the second, we consider in addition the *connectivity* between nodes; again there are numerous possibilities to choose from, and as an example we consider *symmetric* links, rather than *uni-directional* links. In this case Π will record, in addition to the status of nodes, the status of the connections (links) between these nodes.

On the other hand, N will be more or less a standard system description from Dπ, consisting of a collection of communicating *located* processes, which also have the ability to create new locations (and their links in the network), and to migrate between them. We will also augment the language with a construct for reacting to network failures. We believe that this results in a succinct but expressive framework, in which many of the phenomena associated with network failures can be examined.

The behavioural theory is takes the form of *(weak) bisimulation equivalence*, [14] based on labelled actions of the form

$$\Pi \triangleright N \xrightarrow{\mu} \Pi' \triangleright N' \tag{1}$$

where the label μ represents the manner in which an observer, also running on the network Π, can interact with the system N. This interaction may change the state of the system, to N' in the usual manner, but it may also affect the nature of the underlying network. For example, an observer may extend the network by creating new locations; we also allow the observer to kill sites, or in the second framework, break links between sites, thereby capturing changes in the behaviour of N in response to dynamic failures.

In the framework with link failures, the definition of these actions turns out to be relatively sophisticated. Intuitively, the action (1) above is meant to simulate the interaction between an observer and the system. However, even though the system and the observer may initially share the same view of the underlying network, interactions quickly give rise to situations in which these views *diverge*. In general, observers may not be aware of the status of all the nodes and links in a network because they might be *unreachable*; the system, on the other hand may reach such nodes through the knowledge of scoped names. So in (1) above, the network representation Π needs to record the actual state of the underlying network, together with the observers *partial view* of it. This in turn will require developing variations on the actions (1) above, where the actual network representations Π, are replaced by more abstract representations.

In Section 2 we treat the case of *location failure*. We define the language DπLoc, an extension of Dπ [1] [11], with an additional operator $\mathsf{ping}\ l.P\lceil Q\rceil$, for checking the accessibility of a location l. We also allow in the language a construct for directly *killing* a location; although one would not expect to program using this construct, it's presence means that contextual equivalences will take into account the effect of location failure on system behaviour.

The reduction semantics, in Section 2.2 is given relative to a network representation in which locations may be alive or dead; thus effectively $\mathsf{ping}\ l.P\lceil Q\rceil$, running on any site, determines whether the site l is *alive* or *dead*. This is followed, in § 2.3, by a labelled transition system (lts) for DπLoc, which seeks to capture the interactions between a user and a system, running on a joint network. The choice of actions in this lts is justified by the first main result in the paper, Theorem 2.3.8, which states that the resulting *(weak) bisimulation equivalence* coincides with a naturally defined *contextual equivalence*, called *reduction barbed congruence*. This is a variation on the contextual equivalence originally defined for *CCS*, [14], in [17], proposed in [13], and adapted to Dπ in [8]. This result may be considered to be a generalisation of the work in [16], which treated a simple distributed version of CCS with no dynamic location creation, to Dπ.

In Section 3, the truly novel part of the paper, we add *permanent link failure*. The resulting language, DπF, is essentially the same as DπLoc, except that it also contains a construct for breaking links between nodes. However, the re-

[1]For convenience, we ignore most of the type system developed for Dπ, as it is orthogonal to the issues addressed in this paper

duction semantics is now given relative to network configuration, ranged over by Δ, which record both the status of nodes and their connectivity. Interestingly, ping $l.P\lceil Q\rceil$ executed on site k, now checks whether l is *accessible* from k; it may not be accessible either because l is dead or because the link between l and k is broken. Moreover, in DπF, *node creation* is more complicated, since one must also specify the connectivity of a new node; in DπF this is achieved using a simple type system.

In § 3.3 we present a variation of the actions (1) above for DπF. It turns out that much of the information in the network representations Δ is irrelevant; for example if a node k is dead, then it does not matter whether or not it contains a link to another node l. We argue that actions must be expressed as

$$\Sigma \triangleright N \xrightarrow{\mu} \Sigma' \triangleright N' \tag{2}$$

where Σ, Σ' are structures which provide sufficient information to determine both:

- the aspects of the actual networks $\Delta,\ \Delta'$ relevant to users.
- the user's (partial) knowledge of the actual network.

The resulting bisimulation equivalence can be used to demonstrate equivalencies between systems. At this point we show, through a series of examples, that in general, this bisimulation is too discriminating. We therefore revise the definition of these actions in § 3.4, essentially by abstracting from internal information present in the action labels, such that the resulting equivalence again coincides with *contextual equivalence*. The proof, which is quite technical, is then elaborated in Section 4.

The paper ends with a section on conclusions, suggestions for further research, and a description of related work.

2 Dπ with location failure

In this section we present an extension to Dπ called DπLoc, in which fail-stop location failure can be expressed.

2.1 DπLoc syntax

The syntax of DπLoc is given in Table 1 and assumes a set of *variables*, Vars, ranged over by $x, y, z, \ldots$, and a separate set of *names*, Names, ranged over by $n, m, \ldots$. This latter set is divided into locations, Locs, ranged over by $l, k, \ldots$ and channels, Chans, ranged over by $a, b, c, \ldots$. We use $u, v, \ldots$ to range over the set of *identifiers*, consisting of variables and names. When new names are created, they have associated with them a type, indicating whether they are to be used as a channel, $\mathtt{ch}$, or as a location, $\mathtt{loc[S]}$ with state $\mathtt{S}$, which can either be alive, $\mathtt{a}$, or dead, $\mathtt{d}$. A priori, there is not much sense in declaring a dead location,

Table 1. *Syntax of typed DπLoc*

Types

$$\mathtt{T},\mathtt{U} ::= \mathtt{loc}[\mathtt{S}] \mid \mathtt{ch} \qquad\qquad \mathtt{S},\mathtt{R} ::= \mathtt{a} \mid \mathtt{d}$$

Processes

$$\begin{array}{rllllll} P,Q ::= & u!\langle V\rangle.P & \mid u?(X).P & \mid *\,u?(X).P & \mid \mathsf{if}\; v{=}u.P\lceil Q\rceil & \mid P|Q \\ \mid & (\nu\, n:\mathtt{T})P & \mid \mathsf{go}\; u.P & \mid \mathsf{kill} & \mid \mathsf{ping}\; u.P\lceil Q\rceil & \mid \mathbf{0} \end{array}$$

Systems

$$M,N,O ::= l[\![P]\!] \qquad \mid N|M \qquad \mid (\nu\, n:\mathtt{T})N$$

but the presence of this construct will facilitate the definition of the reduction semantics.

There are three syntactic categories in DπLoc. The first, *local processes* ranged over by P, Q, includes the standard π-calculus constructs for communication, $a!\langle V\rangle.P$ and $a?(X).P$, replicated input, $*a?(X).P$, name restriction $(\nu\, n:\mathtt{T})P$, where $\mathtt{T}$ types n as a channel or a location name, comparison $\mathsf{if}\; v = u.P\lceil Q\rceil$, inaction, $\mathbf{0}$, and parallel composition, $P|Q$. The values transmitted as part of a communication, ranged over by V, consist of tuples of identifiers. When input on a channel, they are deconstructed using patterns, ranged over by X; patterns are simply tuples of variables, each having a unique occurrence.

The major innovation is a programming construct which allows processes to react to *perceived faults* in the underlying communication network. In addition to the Dπ migration construct $\mathsf{go}\; l.P$, [11], we add a testing construct, $\mathsf{ping}\; l.P\lceil Q\rceil$, inspired from [2, 1, 16]. This constructs acts as a conditional, based on the *perceived state* of the location l; thus if l is reachable, it launched process P, otherwise it launches Q.

As explained in the introduction, we also wish to consider the behaviour of systems under dynamic network faults. To simulate these instances, we also add to the language a construct for inducing faults, kill; even though this should not be considered part of the core language, its inclusion means that contextual equivalences will compare system behaviour in the presence of dynamic fail-stop [18] location failure.

The second syntactic category, ranged over by N, M, *systems*, is similar to that category in Dπ. They consist of *located processes*, terms of the form $l[\![P]\!]$, which can be composed together with the parallel operator $N \mid M$ and scoped to share private names as $(\nu\, n:\mathtt{T})N$. Note that, as with local processes, scoped names always have associated types; in the case of locations, these type carry the state of the scoped location (dead or alive).

In contrast to Dπ, DπLoc uses also an additional third level of *configurations*. At this level, we have a representation of the network on which the system is running. A typical configuration takes the form

$$\Pi \triangleright N$$

where Π represents the network state. This network representation is made up of two components $\langle \mathcal{N}, \mathcal{D} \rangle$:

- $\mathcal{N}$ is a set of all the free names used in the system N of the configuration. It contains both channel and location names and thus satisfies the condition $\mathcal{N} \subseteq \textsc{Names}$.

- $\mathcal{D}$ is referred to as a deadset, representing resources that cannot be used because a fault occurred to them. Since in DπLoc we only consider location failure, $\mathcal{D}$ is a set of dead location names, $\mathcal{D} \subseteq \mathbf{loc}(\mathcal{N})$, where $\mathbf{loc}(\mathcal{N})$ denotes the location names in $\mathcal{N}$.

For an arbitrary Π we use $\Pi_{\mathcal{N}}$ and $\Pi_{\mathcal{D}}$ to refer to its individual components.

Notation 2.1.1. The input constructs are binders for variables, while the scoping constructs $(\nu\, n : \mathtt{T})N$ and $(\nu\, n : \mathtt{T})P$ are binders for names. We assume the usual concepts of *free* and *bound* occurrences, and the associated notation, such as α-conversion and capture avoiding substitution of names for variables. Terms with no occurrences of free variables are called *closed*, and in the sequel, we will assume that all system level terms and configurations are closed.

Throughout the report a number of abbreviations are used to improve the readability of code. We often omit occurrences of $\mathbf{0}$ in synchronous constructs like input, output and conditional constructs. Thus, $a?(X)$, $a!\langle V\rangle$, $\mathsf{if}\ n = m.P$ and $\mathsf{ping}\ l.\lceil Q\rceil$ are shorthand for $a?(X).\mathbf{0}$, $a!\langle V\rangle.\mathbf{0}$, $\mathsf{if}\ n = m.P\lceil\mathbf{0}\rceil$ and $\mathsf{ping}\ l.\mathbf{0}\lceil Q\rceil$ respectively. Similarly, the abbreviation $\mathsf{if}\ n \neq m.P$ is stands for $\mathsf{if}\ n = m.\mathbf{0}\lceil P\rceil$. Also $a?().P$ denotes an input where the binding variable does not occur in P and $a!\langle\rangle.P$ denotes an output where no value is sent. We also write $\mathsf{go}\ l_1, \ldots, l_n.P$ as an abbreviation to the nested moves $\mathsf{go}\ l_1.(\ldots(\mathsf{go}\ l_n.P))$. Finally we will also omit occurrences of types from terms, unless they are relevant to the discussion at hand. ■

2.2 The reduction semantics of DπLoc

The reduction semantics of DπLoc is defined as a binary relation between *well-formed configurations*.

Definition 2.2.1 (Well-Formed Configurations). A configuration $\Pi \triangleright N$ is said to be *well formed* if every free name occurring in N is in $\Pi_{\mathcal{N}}$. ■

Table 2. *Local Reduction Rules for DπLoc*

Assuming $\Pi \vdash l : \mathbf{alive}$

(r-comm)
$$\frac{}{\Pi \triangleright l[\![a!\langle V\rangle.P]\!] \mid l[\![a?(X).Q]\!] \quad \longrightarrow \quad \Pi \triangleright l[\![P]\!] \mid l[\![Q\{^V/_X\}]\!]}$$

(r-rep)
$$\frac{}{\Pi \triangleright l[\![*a?(X).P]\!] \quad \longrightarrow \quad \Pi \triangleright l[\![a?(X).(P| * a?(X).P)]\!]}$$

(r-eq)
$$\frac{}{\Pi \triangleright l[\![\text{if } u=u.P\lceil Q\rceil]\!] \longrightarrow \Pi \triangleright l[\![P]\!]}$$

(r-neq)
$$\frac{}{\Pi \triangleright l[\![\text{if } u=v.P\lceil Q\rceil]\!] \longrightarrow \Pi \triangleright l[\![Q]\!]}\ u \neq v$$

(r-fork)
$$\frac{}{\Pi \triangleright l[\![P|Q]\!] \quad \longrightarrow \quad \Pi \triangleright l[\![P]\!] \mid l[\![Q]\!]}$$

The judgements of the reduction semantics are therefore of the form

$$\Pi \triangleright M \quad \longrightarrow \quad \Pi' \triangleright M'$$

where $\Pi \triangleright M$, $\Pi' \triangleright M'$ are well-formed configurations. The relation is defined to be the least one which satisfies the set of of rewriting rules in Table 2, Table 4 and Table 3. In the first batch of reduction rules (Table 2) we adapt the standard axioms for reduction in Dπ from [10]. The main change is all reductions, such as communication in the rule (r-comm) and testing for equality between identifiers, (r-eq) and (r-neq), require the location of the activity be alive in the network; there is the global requirement that $\Pi \vdash l : \mathbf{alive}$, which formally means that l must not be in $\Pi_{\mathcal{D}}$. Throughout the paper we use various notation for checking the status of a network, or updating it; this will be explained informally as it is introduced, with the formal definitions relegated to the appendix.

The rules for the novel constructs are in Table 3. Code migration is still asynchronous, but is now subject to the current state of the network: (r-go) says that if the destination location k is *accessible* from the source location l, denoted as $\Pi \vdash k \leftarrow l$, then the migration will be successful; otherwise, if k is inaccessible from l, $\Pi \nvdash k \leftarrow l$, then (r-ngo) states that the migration fails and the migrating code is lost. Similarly, the ping construct, continues as P at l if k is accessible from the current location but branches to Q at l if k is inaccessible. We note that in DπLoc, the only way for k to be inaccessible from l, is when the former is dead; later on in the paper, accessibility will also depend on the links between

Table 3. *Network Reduction Rules for DπLoc*

Assuming $\Pi \vdash l : \mathbf{alive}$

(r-go)
$$\frac{}{\Pi \triangleright l[\![\mathsf{go}\, k.P]\!] \longrightarrow \Pi \triangleright k[\![P]\!]}\ \Pi \vdash k \leftarrow l$$

(r-ngo)
$$\frac{}{\Pi \triangleright l[\![\mathsf{go}\, k.P]\!] \longrightarrow \Pi \triangleright k[\![\mathbf{0}]\!]}\ \Pi \nvdash k \leftarrow l$$

(r-ping)
$$\frac{}{\Pi \triangleright l[\![\mathsf{ping}\, k.P\lceil Q\rceil]\!] \longrightarrow \Pi \triangleright l[\![P]\!]}\ \Pi \vdash k \leftarrow l$$

(r-nping)
$$\frac{}{\Pi \triangleright l[\![\mathsf{ping}\, k.P\lceil Q\rceil]\!] \longrightarrow \Pi \triangleright l[\![Q]\!]}\ \Pi \nvdash k \leftarrow l$$

(r-new)
$$\frac{}{\Pi \triangleright l[\![(\nu\, n\!:\!\mathtt{T})P]\!] \longrightarrow \Pi \triangleright (\nu\, n\!:\!\mathtt{T})\, l[\![P]\!]}$$

(r-kill)
$$\frac{}{\Pi \triangleright l[\![\mathsf{kill}]\!] \longrightarrow (\Pi - l) \triangleright l[\![\mathbf{0}]\!]}$$

locations. Dynamic network faults are engendered in the obvious manner by (r-kill), and finally (r-new), allows us to export locally generated new names to the system level, as in Dπ.

The rules in Table 4 are adaptations of standard rules for the π-calculus. For instance, the first rule, (r-str), states that the reduction semantics is defined up to a *structural equivalence*, defined in the usual manner, as the least equivalence relation on systems which satisfies the set of rules and axioms in Table 5. The remaining reduction rules in Table 4 state that reduction is preserved by parallel composition and name scoping operations on configurations. But note that the rule for scoping, (r-ctxt-rest), uses an obvious notation $\Pi + n : \mathtt{T}$ for extending network representations with new names, which is formally defined in the Appendix. Note also that this rule needs to allow for the type of the scoped name to change; this is because types for locations actually carry *dynamic* state information, namely whether they are alive or dead, as explained in the following example.

Example 2.2.2. Consider the following system

$$\Pi \triangleright k[\![\mathsf{go}\, l.a?(x).P]\!] \;\; | \; l[\![(\nu\, k_0\!:\!\mathtt{loc[a]})(a!\langle k_0\rangle.Q \;\; | \; \mathsf{go}\, k_0.\mathsf{kill})]\!] \tag{3}$$

where Π is the network representation $\langle\{l, k, a\}, \emptyset\rangle$, consisting of the two *live* locations l, k; the addition of the construct $\mathsf{go}\ k_0.\,\mathsf{kill}$ indicates that we wish to consider the newly created location k_0, as defective, and thus it may become faulty some time in the future.

Table 4. *Contextual Reduction Rules for DπLoc*

(r-str)
$$\frac{\Pi \triangleright N' \equiv \Pi \triangleright N \quad \Pi \triangleright N \longrightarrow \Pi' \triangleright M \quad \Pi' \triangleright M \equiv \Pi' \triangleright M'}{\Pi \triangleright N' \longrightarrow \Pi' \triangleright M'}$$

(r-ctxt-rest)
$$\frac{\Pi + n : \mathtt{T} \triangleright N \quad \longrightarrow \quad \Pi' + n : \mathtt{U} \triangleright M}{\Pi \triangleright (\nu\, n : \mathtt{T})N \quad \longrightarrow \quad \Pi' \triangleright (\nu\, n : \mathtt{U})M}$$

(r-ctxt-par)
$$\frac{\Pi \triangleright N \quad \longrightarrow \quad \Pi' \triangleright N'}{\begin{array}{lcl}\Pi \triangleright N|M & \longrightarrow & \Pi' \triangleright N'|M \\ \Pi \triangleright M|N & \longrightarrow & \Pi' \triangleright M|N'\end{array}} \; \Pi \vdash M$$

Table 5. *Structural Rules for DπLoc*

(s-comm)	$N\|M \equiv M\|N$	
(s-assoc)	$(N\|M)\|M' \equiv N\|(M\|M')$	
(s-unit)	$N\|l[\![\mathbf{0}]\!] \equiv N$	
(s-extr)	$(\nu\, n{:}\mathtt{T})(N\|M) \equiv N\|(\nu\, n{:}\mathtt{T})M$	$n \notin \mathbf{fn}(N)$
(s-flip)	$(\nu\, n{:}\mathtt{T})(\nu\, m{:}\mathtt{U})N \equiv (\nu\, m{:}\mathtt{U})(\nu\, n{:}\mathtt{T})N$	
(s-inact)	$(\nu\, n{:}\mathtt{T})N \equiv N$	$n \notin \mathbf{fn}(N)$

As in Dπ, an application of (r-go), based on the fact that both k and l are alive, and (r-par-ctxt) on (3), yields

$$\Pi \triangleright l[\![a?(x).P]\!] \;\; | \; l[\![(\nu\, k_0 : \mathtt{loc[a]})a!\langle k_0\rangle.Q \;\; | \; \mathsf{go}\; k_0.\mathsf{kill}]\!]$$

which can be followed by an application of (r-fork), (r-new) (and (r-par-ctxt)) to launch a new location k_0 and get

$$\Pi \triangleright l[\![a?(x).P]\!] \;\; | \; (\nu\, k_0 : \mathtt{loc[a]})(\, l[\![a!\langle k_0\rangle.Q]\!] \;\; | \; l[\![\mathsf{go}\; k_0.\mathsf{kill}]\!]\,)$$

At this point, we can perform a communication on channel a using (r-par-comm), thereby enlarging the scope of $(\nu\, k_0 : \mathtt{loc[a]})$ through the structural rule (s-extr) and obtain

$$\Pi \triangleright (\nu\, k_0 : \mathtt{loc[a]})(\, l[\![P\{^{k_0}/_x\}]\!] \;\; | \; l[\![Q]\!] \;\; | \; l[\![\mathsf{go}\; k_0.\mathsf{kill}]\!]\,) \tag{4}$$

At this point we can analyse the novel reductions in DπLoc. In (4) the fault inducing process $\mathsf{go}\; k_0.\mathsf{kill}$ can move to k_0 since k_0 is described as alive by the type $\mathtt{loc[a]}$, thereby obtaining

$$\Pi \triangleright (\nu\, k_0 : \mathtt{loc[a]})(\, l[\![P\{^{k_0}/_x\}]\!] \;\; | \; l[\![Q]\!] \;\; | \; k_0[\![\mathsf{kill}]\!]\,) \tag{5}$$

Finally, (r-kill), followed by (r-ctxt-par), can be used to kill k_0 and derive

$$(\Pi + k_0 : \mathtt{loc[a]}) \triangleright l[\![P\{^{k_0}/_x\}]\!] \;\; | \; l[\![Q]\!] \;\; | \; k_0[\![\mathsf{kill}]\!] \quad \longrightarrow$$
$$(\Pi + k_0 : \mathtt{loc[d]}) \triangleright l[\![P\{^{k_0}/_x\}]\!] \;\; | \; l[\![Q]\!] \;\; | \; k_0[\![\mathbf{0}]\!]$$

where the type of k_0 changes from $\mathtt{loc[a]}$ to $\mathtt{loc[d]}$, and thus, an application of (r-ctxt-rest) reduces (5) to

$$\Pi \triangleright (\nu\, k_0 : \mathtt{loc[d]})(\; l[\![P\{^{k_0}/_x\}]\!] \;\; | \; l[\![Q]\!] \;\; | \; k_0[\![\mathbf{0}]\!] \;) \tag{6}$$

■

This example also serves to illustrate another important point that we shall refer to repeatedly in this report. In general, in a configuration

$$\Pi \triangleright N$$

Π denotes the network representation on which the system N is running. But there may be subsystems of N which are running on extended (internal) networks. For example in (3) above, the subsystem $l[\![a!\langle k_0\rangle.Q]\!]$ is running with respect to the network represented by $(\Pi + k_0 : \mathtt{loc[a]})$, while in (6) the subsystem $l[\![Q]\!]$ is running with respect to $(\Pi + k_0 : \mathtt{loc[d]})$.

Reduction barbed congruence

In view of the reduction semantics, we can now adapt the standard approach [9, 8] to obtain a contextual equivalence for DπLoc; we use the variation first proposed in [12]. We wish to compare the behaviour of systems running on the same network; consequently we use the following framework, borrowed from [8]:

Definition 2.2.3 (Typed Relation). A *typed relation* over systems is a family of binary relations between systems, $\mathcal{R}$, indexed by network representations. We write $\Pi \models M \;\; \mathcal{R} \;\; N$ to mean that systems M and N are related by $\mathcal{R}$ at index Π, that is $M \;\; \mathcal{R}_\Pi \;\; N$, and moreover $\Pi \triangleright M$ and $\Pi \triangleright N$ are valid configurations. ■

The definition of our equivalence hinges on what it means for a typed relation to be *contextual*, which must of course take into account the presence of the network. Our definition has two requirements:

- systems running on the network Π must be considered equivalent by all observers also running on Π
- systems must remain equivalent when the network is extended by new locations.

First let us define what kinds of observing systems are allowed to run on a given network.

Definition 2.2.4 (Observers). The intuition of *valid* observer system O in a distributed setting Π, denoted as $\Pi \vdash_{\mathsf{obs}} O$, is that O originates from some *live fresh* location k_0, migrates to any location in $\mathbf{loc}(\Pi_{\mathcal{N}})$ to interact with (observe) processes there and then returns back to the originating fresh location k_0 to compare its observations with other observers. For convenience, we often omit the mentioning of the fresh locations k_0 and place observing code immediately at locations in $\mathbf{loc}(\Pi_{\mathcal{N}})$. We note that, according to the definition of the reduction rule (r-ngo), observing code can never reach dead locations and we therefore have to encode this in our definition of $\Pi \vdash_{\mathsf{obs}} O$. For convenience, we also disallow observer to be located at scoped *dead* loactions, denoted as $\Pi \vdash_{\mathsf{obs}} \mathtt{T}$ and defined in the Appendix. $\Pi \vdash_{\mathsf{obs}} O$ is recursively defined as:-

- $\Pi \vdash_{\mathsf{obs}} l[\![P]\!]$ if $\mathbf{fn}(P) \subseteq \Pi_{\mathcal{N}}$ and $\Pi \vdash l : \mathbf{alive}$
- $\Pi \vdash_{\mathsf{obs}} (\nu\, n\!:\!\mathtt{T})N$ if $\Pi \vdash_{\mathsf{obs}} \mathtt{T}$ and $(\Pi + n\!:\!\mathtt{T}) \vdash_{\mathsf{obs}} N$
- $\Pi \vdash_{\mathsf{obs}} M \mid N$ if $\Pi \vdash_{\mathsf{obs}} M$ and $\Pi \vdash_{\mathsf{obs}} N$ ■

Definition 2.2.4, defining allowed observer systems, determines the definition of contextuality given below.

Definition 2.2.5 (Contextual typed relations). A typed relation $\mathcal{R}$ over configurations is *contextual* if:

(Parallel Systems)

- $\Pi \models M \,\mathcal{R}\, N$ and $\Pi \vdash_{\mathsf{obs}} O$ implies
  - $\Pi \models M|O \,\mathcal{R}\, N|O$
  - $\Pi \models O|M \,\mathcal{R}\, O|N$

(Network Extensions)

- $\Pi \models M \,\mathcal{R}\, N$ and $\Pi \vdash_{\mathsf{obs}} \mathtt{T}$, n fresh implies $\Pi + n\!:\!\mathtt{T} \models M \,\mathcal{R}\, N$

■

Definition 2.2.6 (Reduction barbed congruence). First we define the adaptation of the other standard relations required to define *reduction barbed congruence*.

- $\Pi \triangleright N \Downarrow_{a@l}$ denotes an *observable barb* exhibited by the configuration $\Pi \triangleright N$, on channel a at location l. Formally, it means that $\Pi \triangleright N \longrightarrow^{*} \Pi' \triangleright N'$ for some $\Pi' \triangleright N'$ such that $N' \equiv M|l[\![a!\langle V\rangle.Q]\!]$ and $\Pi \vdash l\!:\!\mathbf{alive}$. Then, we say a typed relation $\mathcal{R}$ over configurations is *barb preserving* whenever $\Pi \models N \;\mathcal{R}\; M$ and $\Pi \triangleright N \Downarrow_{a@l}$ implies $\Pi \triangleright M \Downarrow_{a@l}$.
- A typed relation $\mathcal{R}$ over configurations is *reduction closed* whenever $\Pi \models N \;\mathcal{R}\; M$ and $\Pi \triangleright N \longrightarrow \Pi' \triangleright N'$ implies $\Pi \triangleright M \longrightarrow^{*} \Pi' \triangleright M'$ for some $\mathcal{I}' \triangleright M'$ such that $\Pi' \models N' \;\mathcal{R}\; M'$.

Then $\cong$, called *reduction barbed congruence*, is the largest symmetric typed relation over configurations which is:

- barb preserving
- reduction closed
- contextual ■

We leave the reader to check that pointwise $\cong$ is an equivalence relation.

Example 2.2.7. Consider the systems onePkt and twoPkt defined as:

$$\mathsf{onePkt} \Leftarrow l[\![\mathsf{go}\ k.(a!\langle\rangle | b!\langle\rangle)]\!]$$
$$\mathsf{twoPkt} \Leftarrow l[\![\mathsf{go}\ k.a!\langle\rangle]\!] \mid l[\![\mathsf{go}\ k.b!\langle\rangle]\!]$$

They represent two different strategies for sending the messages $a!\langle\rangle|b!\langle\rangle$ from l to k. The first system, onePkt, transfers the two messages as one unit (one packet), whereas the second system, twoPkt, uses a distinct packet for every message. In a calculus with no network failure, it would be hard to distinguish between these two systems.

The two configurations are however not reduction barbed congruent in our calculus when run over the network $\Pi_{lk} = \langle\{l, k, a, b\}; \emptyset\rangle$, in which l, k are alive. This is formally stated as

$$\Pi_{lk} \models \mathsf{onePkt} \not\cong \mathsf{twoPkt}$$

and the reason why they are not is because they can exhibit different behaviour when l is subject to failure during the transfer of the packets. Formally, we can examine the behaviour of systems under this situation by considering their behaviour in the context

$$C[-] \quad = \quad [-] \,|\, l[\![\mathsf{kill}]\!]$$

By Definition 2.2.6, if we assume that $\Pi_{lk} \models \mathsf{onePkt} \cong \mathsf{twoPkt}$, then *contextuality* of $\cong$ would imply

$$\Pi_{lk} \triangleright \mathsf{onePkt} \,|\, l[\![\mathsf{kill}]\!] \;\cong \Pi_{lk} \triangleright \mathsf{twoPkt} \,|\, l[\![\mathsf{kill}]\!]$$

But we can show directly that the latter cannot be true, thereby contradicting our assumption. For example, using the reduction rules of Tables 2, 4 and 3 we can derive the following sequence of reductions for $\Pi_{lk} \triangleright \mathsf{twoPkt} \,|\, l[\![\mathsf{kill}]\!]$:

$$\begin{aligned}
\Pi_{lk} \triangleright l[\![\mathsf{go}\ k.a!\langle\rangle]\!] \mid l[\![\mathsf{go}\ k.b!\langle\rangle]\!] \mid l[\![\mathsf{kill}]\!] &\longrightarrow \Pi_{lk} \triangleright k[\![a!\langle\rangle]\!] \mid l[\![\mathsf{go}\ k.b!\langle\rangle]\!] \mid l[\![\mathsf{kill}]\!] \\
&\longrightarrow \Pi_{k} \triangleright k[\![a!\langle\rangle]\!] \mid l[\![\mathsf{go}\ k.b!\langle\rangle]\!] \\
&\longrightarrow \Pi_{k} \triangleright k[\![a!\langle\rangle]\!]
\end{aligned}$$

where Π_k is the network representation in which l is dead, that is $\langle\{l, k, a, b\}; \{l\}\rangle$

We also note that

$$\Pi_k \triangleright k[\![a!\langle\rangle]\!] \Downarrow_{a@k}$$
$$\Pi_k \triangleright k[\![a!\langle\rangle]\!] \not\Downarrow_{b@k}$$

However the left hand side, $\Pi_{lk} \triangleright \mathsf{onePkt} \,|\, l[\![\mathsf{kill}]\!]$ can never reduce to a configuration with such barbs. Formally, there is no configuration $\Pi \triangleright N$ such that

$$\Pi_{lk} \triangleright l[\![\mathsf{go}\; k.(a!\langle\rangle | b!\langle\rangle)]\!] \,|\, l[\![\mathsf{kill}]\!] \longrightarrow^* \Pi \triangleright N$$

where $\Pi \triangleright N \Downarrow_{a@k}$ and $\Pi \triangleright N \not\Downarrow_{b@k}$. ■

Example 2.2.8. Consider the two systems:

$$\mathsf{nonDet1} \Leftarrow (\nu\, k : \mathtt{loc}[\mathtt{a}])\; k[\![\mathsf{kill}]\!] \,|\, k[\![\mathsf{go}\; l.a!\langle\rangle]\!]$$
$$\mathsf{nonDet2} \Leftarrow (\nu\, b : \mathsf{ch})\; l[\![b!\langle\rangle]\!] \,|\, l[\![b?()]\!] \,|\, l[\![b?().a!\langle\rangle]\!]$$

Both systems exhibit a barb $a@l$ depending on different forms of *non-deterministic internal choices*; the internal choice used by nonDet1 is based on a scoped location k that may fail while the internal choice used by nonDet2 is based on two inputs competing for a single scoped output on channel b.

It turns out that these two systems are observationally equivalent when run over the simple network $\Pi_l = \langle\{l, a\}, \emptyset\rangle$, formally stated as

$$\Pi_l \models \mathsf{nonDet1} \cong \mathsf{nonDet2}$$

Nevertheless, Definition 2.2.6 exhibits a major limitation at this point, because it makes it quite hard to prove such an equivalence. Such a complication arises from the fact that the definitions of reduction barbed congruence requires us to reason about the behaviour of the two configurations under all possible contexts, which are infinite. ■

Example 2.2.9. Here we consider three implementations of a simple (abstract) server, executing on a network $\Pi = \langle\{l, k_1, k_2, serv, ret\}; \emptyset\rangle$ where three locations l, k_1 and k_2 are alive. The first is the most straightforward:

$$\mathsf{server} \Leftarrow (\nu\, data : \mathsf{ch})(l[\![req?(x,y).data!\langle x,y\rangle]\!] \,|\, l[\![data?(x,y).y!\langle f(x)\rangle]\!])$$

It simply takes in a request at the port *req* consisting of an argument, x, and a return channel on which to return the answer of the request, y. The server proceeds by forwarding the two parameters, x and y to an internal database, denoted by the scoped channel *data*; intuitively, the database looks up the mapping of the value x using some unspecified function $f()$ and returns the answer, $f(x)$, back onport y. The key aspect of this server is that all the processing is performed *locally* at location l. A typical client for such a server would have the following form, sending the name l as the value to be processed and *ret* as the return channel:

$$\mathsf{client} \Leftarrow l[\![req!\langle l, ret\rangle]\!]$$

By contrast, the next two server implementations introduce a degree of *distribution*, by processing the request across a number of locations:

$$\mathsf{srvDis} \Leftarrow (\nu\, data\!:\!\mathsf{ch})\left(\begin{array}{l} l[\![req?(x,y).\mathsf{go}\ k_1.data!\langle x,y\rangle]\!] \\ \mid k_1[\![data?(x,y).\mathsf{go}\ l.y!\langle f(x)\rangle]\!] \end{array}\right)$$

$$\mathsf{srv2Rt} \Leftarrow (\nu\, data\!:\!\mathsf{ch})\left(\begin{array}{l} l\left[\!\!\left[req?(x,y).(\nu\, sync\!:\!\mathsf{ch})\left(\begin{array}{l} \mathsf{go}\ k_1.data!\langle x, sync\rangle \\ \mid \mathsf{go}\ k_2, k_1.\, data!\langle x, sync\rangle \\ \mid synch?(x).y!\langle x\rangle \end{array}\right)\right]\!\!\right] \\ \mid k_1\left[\!\!\left[data?(x,y).\left(\begin{array}{l} \mathsf{go}\ l.\, y!\langle f(x)\rangle \\ \mathsf{go}\ k_2, l.\, y!\langle f(x)\rangle \end{array}\right)\right]\!\!\right] \end{array}\right)$$

Both servers, srvDis and $\mathsf{srv2Rt}$, distributed the internal database *remotely* at location k_1. Server srvDis thus receives a client request at l, migrates *directly* to k_1 and queries the database; the database then returns to l and reports back the processed value, $f(x)$, on the requested return channel y. The other server, $\mathsf{srv2Rt}$, accepts a client request at l, but attempts to access the unique remote database located at k_1 through *two different routes*, one *directly* from l to k_1 and the other *indirectly* from l through the intermediate node k_2 and then finally to k_1 where the database resides; similarly, the internal database of $\mathsf{srv2Rt}$ returns the answer $f(x)$ on y along these two routes. In a scenario where no fault occurs to k_1 and k_2, $\mathsf{srv2Rt}$ will receive two answers back at l. To solve this, the original requests are sent with a scoped return channel *sync*; a process waiting for answers on this channel at location l chooses non-deterministically between any two answers received and relays the answer on the original channel y.

We leave the reader to check that the local server, server and remote implementations, srvDis and $\mathsf{srv2Rt}$, are different, that is:

$$\Pi \models \mathsf{server} \not\cong \mathsf{srvDis} \quad \text{and} \quad \Pi \models \mathsf{server} \not\cong \mathsf{srv2Rt}$$

because of their behaviour in the context

$$C_2[-] = [-] \mid k_1[\![\mathsf{kill}]\!]$$

However, it turns out that the two remote server implementations are reduction barbed congruent in DπLoc:

$$\Pi \models \mathsf{srvDis} \cong \mathsf{srv2Rt}$$

Unfortunately, Definition 2.2.6 makes it hard to prove this statement because it uses quantification over all possible contexts. ■

Due to the problems associated with Definition 2.2.6, we need an inductive definition of behavioural equivalence that is easier to prove but still consistent with reduction barbed congruence. In the remainder of this section we define a *bisimulation equivalence* which allows us to relate DπLoc configurations in

Table 6. *Operational Rules(1) for DπLoc*

Assuming $\Pi \vdash l : \mathbf{alive}$

(l-out)

$$\overline{\Pi \triangleright l[\![a!\langle V\rangle.P]\!] \xrightarrow{l:a!\langle V\rangle} \Pi \triangleright l[\![P]\!]}$$

(l-in)

$$\overline{\Pi \triangleright l[\![a?(X).P]\!] \xrightarrow{l:a?(V)} \Pi \triangleright l[\![P\{V/X\}]\!]}\ V \subseteq \Pi_{\mathcal{N}}$$

(l-in-rep)

$$\overline{\Pi \triangleright l[\![*a?(X).P]\!] \xrightarrow{\tau} \Pi \triangleright l[\![a?(X).(P \mid *a?(Y).P\{Y/X\})]\!]}$$

(l-eq)

$$\overline{\Pi \triangleright l[\![\mathsf{if}\ u{=}u.P\lceil Q\rceil]\!] \xrightarrow{\tau} \Pi \triangleright l[\![P]\!]}$$

(l-neq)

$$\overline{\Pi \triangleright l[\![\mathsf{if}\ u{=}v.P\lceil Q\rceil]\!] \xrightarrow{\tau} \Pi \triangleright l[\![Q]\!]}\ u \neq v$$

(l-fork)

$$\overline{\Pi \triangleright l[\![P \mid Q]\!] \xrightarrow{\tau} \Pi \triangleright l[\![P]\!] \mid l[\![Q]\!]}$$

a tractable manner. It will turn out that this bisimulation equivalence coincides with reduction barbed congruence.

2.3 *A bisimulation equivalence for DπLoc*

We start by defining the labelled transition system on which we base our definitions of bisimulation equivalence.

Definition 2.3.1 (A labelled transition system for DπLoc). This consists of a collection of actions $\Pi \triangleright N \xrightarrow{\mu} \Pi' \triangleright N'$, where μ takes one of the forms:

- τ, representing internal action
- $(\tilde{n} : \tilde{\mathrm{T}})l : a?(V)$, representing the input of the value V along the channel a, located at l. Here $\tilde{n} : \tilde{\mathrm{T}}$ denotes the fresh names $\tilde{n}$ and their respective state information $\tilde{\mathrm{T}}$, introduced by an observer (context) as part of this action.
- $(\tilde{n} : \tilde{\mathrm{T}})l : a!\langle V\rangle$, the output of the value V along the channel a, located at l. Here $\tilde{n} : \tilde{\mathrm{T}}$ represented the names $\tilde{n}$ which are exported to an observer (context) as part of this action, together with their associated new state information $\tilde{\mathrm{T}}$.
- $\mathsf{kill} : l$, representing the killing of location l by an observer (context). ■

The transitions in the lts for DπLoc are defined as the least relations satisfying the axioms and rules in Tables 6, 8 and 7. Table 6 contains standard operational

Table 7. *Operational Rules(2) for DπLoc*

(l-open)

$$\frac{\Pi + n{:}\mathtt{T} \triangleright N \xrightarrow{(\tilde{n}\tilde{\mathtt{T}})l:a!\langle V\rangle} \Pi' \triangleright N'}{\Pi \triangleright (\nu\, n{:}\mathtt{T})N \xrightarrow{(n:\mathtt{T},\tilde{n}\tilde{\mathtt{T}})l:a!\langle V\rangle} \Pi' \triangleright N'}\; l, a \neq n \in V$$

(l-weak)

$$\frac{\Pi + n{:}\mathtt{T} \triangleright N \xrightarrow{(\tilde{n}\tilde{\mathtt{T}})l:a?(V)} \Pi' \triangleright N'}{\Pi \triangleright N \xrightarrow{(n:\mathtt{T},\tilde{n}\tilde{\mathtt{T}})l:a?(V)} \Pi' \triangleright N'}\; l, a \neq n \in V$$

(l-rest)

$$\frac{\Pi + n{:}\mathtt{T} \triangleright N \xrightarrow{\mu} \Pi' + n{:}\mathtt{U} \triangleright N'}{\Pi \triangleright (\nu\, n{:}\mathtt{T})N \xrightarrow{\mu} \Pi' \triangleright (\nu\, n{:}\mathtt{U})N'}\; n \notin \mathbf{fn}(\mu)$$

(l-par-ctxt)

$$\frac{\Pi \triangleright N \xrightarrow{\mu} \Pi' \triangleright N'}{\begin{array}{c}\Pi \triangleright N|M \xrightarrow{\mu} \Pi' \triangleright N'|M \\ \Pi \triangleright M|N \xrightarrow{\mu} \Pi' \triangleright M|N'\end{array}}\; \Pi \vdash M$$

(l-par-comm)

$$\frac{\Pi \triangleright N \xrightarrow{(\tilde{n}\tilde{\mathtt{T}})l:a!\langle V\rangle} \Pi' \triangleright N' \qquad \Pi \triangleright M \xrightarrow{(\tilde{n}\tilde{\mathtt{T}})l:a?(V)} \Pi'' \triangleright M'}{\begin{array}{c}\Pi \triangleright N|M \xrightarrow{\tau} \Pi \triangleright (\nu\, \tilde{n}{:}\tilde{\mathtt{T}})(N'|M') \\ \Pi \triangleright M|N \xrightarrow{\tau} \Pi \triangleright (\nu\, \tilde{n}{:}\tilde{\mathtt{T}})(M'|N')\end{array}}$$

rules inherited from distributed π-calculi such as Dπ; note, however, that actions can only occur at live locations. The rules in Table 7 are also adaptations of the standard rules for actions-in-context from [9] together with the rule (l-par-comm), for local communication. Here, we highlight the rule (r-weak), dealing with the learning of the existence of new location names and their state as a result of an input from the context; this rule was adopted from a variant used already in [9, 8]. Note also the general form of (l-rest), where the type of n may change from $\mathtt{T}$ to $\mathtt{U}$; this phenomena is inherited directly from (r-ctxt-rest) of Table 4 in the reduction semantics and explained in Example 2.2.9.

The rules dealing with the new constructs of DπLoc, are contained in Table 8, most of which are inherited from the reduction semantics. The only new one is (l-halt), where the action kill : l represents a failure induced by an observer. This is in contrast with the rule (l-kill), where l is killed by the system itself and the associated action is τ.

Table 8. *Operational Rules(3) for DπLoc*

Assuming $\Pi \vdash l : \textbf{alive}$

(l-kill)
$$\frac{}{\Pi \triangleright l[\![\mathsf{kill}]\!] \xrightarrow{\tau} (\Pi - l) \triangleright l[\![\mathbf{0}]\!]}$$

(l-halt)
$$\frac{}{\Pi \triangleright N \xrightarrow{\mathsf{kill}:l} (\Pi - l) \triangleright N}$$

(l-new)
$$\frac{}{\Pi \triangleright l[\![(\nu\, n : \mathrm{T})P]\!] \xrightarrow{\tau} \Pi \triangleright (\nu\, n : \mathrm{T})l[\![P]\!]}$$

(r-go)
$$\frac{}{\Pi \triangleright l[\![\mathsf{go}\; k.P]\!] \xrightarrow{\tau} \Pi \triangleright k[\![P]\!]}\; \Pi \vdash k \leftarrow l$$

(r-ngo)
$$\frac{}{\Pi \triangleright l[\![\mathsf{go}\; k.P]\!] \xrightarrow{\tau} \Pi \triangleright k[\![\mathbf{0}]\!]}\; \Pi \nvdash k \leftarrow l$$

(r-ping)
$$\frac{}{\Pi \triangleright l[\![\mathsf{ping}\; k.P\lceil Q\rceil]\!] \xrightarrow{\tau} \Pi \triangleright l[\![P]\!]}\; \Pi \vdash k \leftarrow l$$

(r-nping)
$$\frac{}{\Pi \triangleright l[\![\mathsf{ping}\; k.P\lceil Q\rceil]\!] \xrightarrow{\tau} \Pi \triangleright l[\![Q]\!]}\; \Pi \nvdash k \leftarrow l$$

The first sanity check we prove about our lts is the property that in an action

$$\Pi \triangleright N \quad \xrightarrow{\mu} \quad \Pi' \triangleright N'$$

were μ is an external action, the residual network Π' is completely determined by the network Π and the external action μ.

Definition 2.3.2 (Action residuals). The partial function after from tuples of network representations Π and external actions μ to network representation is defined as:

- Π after $(\tilde{n} : \tilde{\mathrm{T}})l : a!\langle V\rangle$ is defined as $\Pi + \tilde{n} : \tilde{\mathrm{T}}$
- Π after $(\tilde{n} : \tilde{\mathrm{T}})l : a?(V)$ is defined as $\Pi + \tilde{n} : \tilde{\mathrm{T}}$
- Π after kill : l is defined as $\Pi - l$ ■

Proposition 2.3.3. If $\Pi \triangleright N \xrightarrow{\mu} \Pi' \triangleright N'$ where μ is an external action, then Π' coincides with Π after μ

Proof. A straightforward induction on the inference of $\Pi \triangleright N \xrightarrow{\mu} \Pi' \triangleright N'$. □

The second sanity check is that the actions are indeed well-defined relations over configurations.

Proposition 2.3.4. The lts defined in Definition 2.3.1 forms a binary relation between well-defined configurations. That is, if $\Pi \triangleright N \xrightarrow{\mu} \Pi' \triangleright N'$ and $\Pi \vdash N$ then $\Pi' \vdash N'$.

Proof. By induction on the derivation of the action $\Pi \triangleright N \xrightarrow{\mu} \Pi' \triangleright N'$. Note that if μ is an external action then Proposition 2.3.3 gives the precise form of Π'. Moreover if it is the internal action τ then it is possible to prove that Π' either coincides with Π or takes the form $\Pi - l$ for some l live in Π. □

Using the lts of actions we can now define, in the standard manner, *weak bisimulation equivalence* over configurations. Our definition uses the standard notation for weak actions, namely $\stackrel{\mu}{\Longrightarrow}$ denotes $\Longrightarrow\xrightarrow{\mu}\Longrightarrow$, and $\stackrel{\widehat{\mu}}{\Longrightarrow}$ denotes

- $\xrightarrow{\tau}^{*}$ if $\mu = \tau$
- $\stackrel{\mu}{\Longrightarrow}$ otherwise.

Definition 2.3.5 (Weak bisimulation equivalence). This is denoted as $\approx$, and is defined to be the largest typed relation over configurations such that if $\Pi \models M \approx N$ then

- $\Pi \triangleright M \xrightarrow{\mu} \Pi' \triangleright M'$ implies $\Pi \triangleright N \stackrel{\hat{\mu}}{\Longrightarrow} \Pi' \triangleright N'$ such that $\Pi' \models M' \approx N'$
- $\Pi \triangleright N \xrightarrow{\mu} \Pi' \triangleright N'$ implies $\Pi \triangleright M \stackrel{\hat{\mu}}{\Longrightarrow} \Pi' \triangleright M'$ such that $\Pi' \models M' \approx N'$ ■

Equipped with our bisimulation definitions, we revisit some equivalence examples introduced in § 2.2 and show that they can be tractably proved to be equivalent. But before, we prove a useful result that allows us to give bisimulations *up-to* structural equivalence.

Proposition 2.3.6 (Structural equivalence and bisimulation). Let us define structural equivalence over configurations in the obvious way, overloading the symbol $\equiv$, that is:

$$\Pi \triangleright M \equiv \Pi \triangleright N \quad \text{iff} \quad M \equiv N \text{ and } \Pi \vdash M, N$$

Similar to any typed relation so far, we abbreviate $\Pi \triangleright M \equiv \Pi \triangleright N$ to $\Pi \models M \equiv N$. We now can state that structural equivalence over configurations is a bisimulation relation. Stated otherwise, $\equiv \subseteq \approx$

Proof. We proceed by defining the $\mathcal{R}$ as:

$$\mathcal{R} = \{\Pi \models M \,\mathcal{R}\, N \mid \Pi \models M \equiv N\}$$

It is clear that $\mathcal{R}$ is a typed relation; we only have to show that $\mathcal{R}$ it is a bisimulation. The proof proceeds by induction on the structure of $\Pi \triangleright M$ and $\Pi \triangleright N$. □

Example 2.3.7. We recall that in Example 2.2.8, we claimed that $\Pi_l \triangleright \mathsf{nonDet1}$ was equivalent to $\Pi_l \triangleright \mathsf{nonDet2}$. We here show that they are *bisimilar*, by giving the relation $\mathcal{R}$ defined as:

$$\mathcal{R} = \left\{ \begin{array}{ll} \langle \Pi_l \triangleright M & , \Pi_l \triangleright N \rangle \\ \langle \Pi_l - l \triangleright M & , \Pi_l - l \triangleright N \rangle \end{array} \;\middle|\; \langle M, N \rangle \in \mathcal{R}_{sys} \right\}$$

where:

$$\mathcal{R}_{sys} = \left\{ \begin{array}{ll} \langle \mathsf{nonDet1} & , \ \mathsf{nonDet2} \rangle \\ \langle (\nu\, k : \mathtt{loc[a]})\, k[\![\mathsf{kill}]\!] \,|\, l[\![a!\langle\rangle]\!] & , \ (\nu\, b : \mathsf{ch})\, l[\![b?()]\!] \,|\, l[\![a!\langle\rangle]\!] \rangle \\ \langle (\nu\, k : \mathtt{loc[d]})\, l[\![a!\langle\rangle]\!] & , \ (\nu\, b : \mathsf{ch})\, l[\![b?()]\!] \,|\, l[\![a!\langle\rangle]\!] \rangle \\ \langle (\nu\, k : \mathtt{loc[d]})\, k[\![\mathsf{go}\ l.a!\langle\rangle]\!] & , \ (\nu\, b : \mathsf{ch})\, l[\![b?().a!\langle\rangle]\!] \rangle \\ \langle (\nu\, k : \mathtt{loc[a]})\, k[\![\mathsf{kill}]\!] & , \ (\nu\, b : \mathsf{ch})\, l[\![b?()]\!] \rangle \\ \langle (\nu\, k : \mathtt{loc[d]})\, k[\![\mathbf{0}]\!] & , \ (\nu\, b : \mathsf{ch})\, l[\![b?()]\!] \rangle \end{array} \right\}$$

■

Of course we need to justify the use of bisimulations to relate systems. This is provided by the following result:

Theorem 2.3.8 (Soundness and Completeness for DπLoc). In DπLoc, $\Pi \models N \approx M$ if and only if $\Pi \models N \cong M$. ■

We omit the proof, as the result can be derived from the second characterisation result of the paper Theorem 4.1.10.

3 Location and Link Failure

In this section we extend the network representation to describe the state of physical links between sites. As explained in the Introduction, in such a setting we can then represent *connectivity failures*, resulting from faults in links between locations. Moreover, the liveness of such links affects in turn the semantics of ping, the construct used to detect faults.

The core language remains the same, although we need to add a new construct to induce link faults. With this extended notion of a network we redo the previous section, obtaining similar results; however the development is considerably more complicated.

3.1 DπF syntax

The syntax of DπF is a minor extension to that of DπLoc; Table 9 highlights the novelties. The main one is that new types are required for locations. Now when a new location is declared, in addition to its live/dead *status*, we have to also

Table 9. *Syntax of DπF*

Types

$\mathtt{T}, \mathtt{U}, \mathtt{W} ::= \mathtt{ch} \mid \mathtt{loc}[\mathtt{S}, \mathtt{C}]$

$\mathtt{S}, \mathtt{R} ::= \mathtt{a} \mid \mathtt{d}$
$\mathtt{C}, \mathtt{D} ::= \{u_1, \ldots, u_n\}$

Processes

$P, Q ::= \ldots \mid \mathsf{break}\ l$

Systems

$N, M ::= \ldots$

describe the live *connections* to other locations. Thus, in DπF, a location type is denoted as $\mathtt{loc}[\mathtt{S}, \mathtt{C}]$, where the first element $\mathtt{S}$ is inherited from Section 2, and the second element $\mathtt{C}$ is a set of locations $\{l_1, \ldots, l_n\}$. If a new k location is declared at this type, then it is intended to be linked in the underlying network with each of the locations l_i, although there will be complications; see Example 3.2.1. The only other modificiation to the syntax is the addition of the process construct $\mathsf{break}\ l$, which breaks a live connection between the location hosting the process and location l. Contextual equivalences then take into account the effect of link faults on system behaviour, in the same manner as the presence of kill takes node faults into account.

The final major extension in the DπF syntax is in the network representation; in a setting where not every node is interconnected, the network representation needs also to represent which nodes are connected apart from their current alive/dead status.

Definition 3.1.1 (Network representation). First let us introduce some notation to represent the *links* in a network. A binary relation $\mathcal{L}$ over locations is called a *link set* if it is:

- symmetric, that is, $\langle l, k\rangle \in \mathcal{L}$ implies $\langle k, l\rangle$ is also in $\mathcal{L}$
- reflexive, that is, $\langle l, k\rangle \in \mathcal{L}$ implies $\langle l, l\rangle$ and $\langle k, k\rangle$ are also in $\mathcal{L}$.

The latter property allows the smooth handling of the degenerate case of a process moving from a site l to l itself. Also for any linkset $\mathcal{L}$ we let $\mathbf{dom}(\mathcal{L})$ denote its domain; that is the collection of locations l, such that $\langle l, l\rangle \in \mathcal{L}$.

Then a *network representation* Δ is any triple $\langle \mathcal{N}, \mathcal{D}, \mathcal{L}\rangle$ where

- $\mathcal{N}$ is a set of names, as before; we now use $\mathbf{loc}(\mathcal{N})$ to represent the subset of $\mathcal{N}$ which are locations

- $\mathcal{D} \subseteq \mathbf{loc}(\mathcal{N})$ represents the set of dead locations, as before.
- $\mathcal{L} \subseteq \mathbf{loc}(\mathcal{N}) \times \mathbf{loc}(\mathcal{N})$ represents the set of connections between locations ■

As with DπLoc network representations, we use the notation $\Delta_{\mathcal{N}}$, $\Delta_{\mathcal{D}}$ and $\Delta_{\mathcal{L}}$ to refer to the individual components of Δ. We will also have various notation for checking properties of DπF network representations, and updating them; these will be explained informally, with the formal definitions relegated to the Appendix.

3.2 Reduction Semantics of DπF

The definition of well-formed configurations, Definition 2.2.1 generalises in a straightforward manner: we say $\Delta \triangleright M$ is a well-formed configuration if every free name occuring in M is also in $\Delta_{\mathcal{N}}$. Then the judgements of the reduction semantics take the form

$$\Delta \triangleright M \quad \longrightarrow \quad \Delta' \triangleright M'$$

where $\Delta \triangleright M$ and $\Delta' \triangleright M'$ are well-formed configurations. This is defined as the least relation which satisfies the rules in Table 2 and Table 4 (substituting Δ for Π) , all inherited from the reduction rules for DπLoc, together with the new reduction rules of Table 10, which we now explain. We note that, as usual, all of these rules require that the location where the activity is to occur is alive, $\Delta \vdash l : \mathbf{alive}$.

The most subtle but important changes to the network reductions rules are those concerning the constructs go and ping. Even though the general intuition remains the same to that of § 2.2, the former notion of k being accessible from l, used by rules such as (r-go) and (r-nping), and still denoted as $\Delta \vdash k \leftarrow l$, changes; for k to be *accessible* from l, two conditions must hold, namely that k is alive *and* that the link between l and k is alive as well. If any of these two conditions do not hold, then k is deemed to be *inaccessible* from the point of view of l, denoted as before as $\Delta \nvdash k \leftarrow l$. The more complex network representation has also an impact on the information that can be gathered by the construct ping; in DπLoc, if ping reduced using (r-nping), it meant that the location being tested for was dead; in DπF however, such a reduction merely means that the destination is inaccessible, which could be caused by a dead destination location, a broken link to the destination location or both.

At this point we note that, in DπF, since *not* every node is interconnected, it makes more sense to talk about *reachability* rather *accessibility* between nodes. A node k is *reachable* from l in Δ, denoted as $\Delta \vdash k \leftsquigarrow l$, if it is accessible using one or more migrations; the formal definition is relegated to the Appendix.

The other main change in Table 10 is the rule for creating new locations, (r-newl); here, the links to the new location k need to be calculated and the network Δ updated. This is achieved by the function inst$(\mathtt{T}, l, \Delta)$, the formal definition

Table 10. *New Network Reduction Rules for DπF*

Assuming $\Delta \vdash l : \mathbf{alive}$

(r-go)
$$\frac{}{\Delta \triangleright l[\![\mathsf{go}\, k.P]\!] \longrightarrow \Delta \triangleright k[\![P]\!]}\ \Delta \vdash k \leftarrow l$$

(r-ngo)
$$\frac{}{\Delta \triangleright l[\![\mathsf{go}\, k.P]\!] \longrightarrow \Delta \triangleright k[\![\mathbf{0}]\!]}\ \Delta \nvdash k \leftarrow l$$

(r-ping)
$$\frac{}{\Delta \triangleright l[\![\mathsf{ping}\, k.P\lceil Q\rceil]\!] \longrightarrow \Delta \triangleright l[\![P]\!]}\ \Delta \vdash k \leftarrow l$$

(r-nping)
$$\frac{}{\Delta \triangleright l[\![\mathsf{ping}\, k.P\lceil Q\rceil]\!] \longrightarrow \Delta \triangleright l[\![Q]\!]}\ \Delta \nvdash k \leftarrow l$$

(r-newc)
$$\frac{}{\Delta \triangleright l[\![(\nu\, c : \mathtt{ch})\, P]\!] \longrightarrow \Delta \triangleright (\nu\, c : \mathtt{ch})\, l[\![P]\!]}$$

(r-newl)
$$\frac{}{\Delta \triangleright l[\![(\nu\, k : \mathtt{loc}[\mathtt{S}, \mathtt{C}])\, P]\!] \longrightarrow \Delta \triangleright (\nu\, k : \mathtt{loc}[\mathtt{S}, \mathtt{D}])\, l[\![P]\!]}\ \mathtt{loc}[\mathtt{S}, \mathtt{D}] = \mathsf{inst}(\mathtt{loc}[\mathtt{S}, \mathtt{C}], l, \Delta)$$

(r-kill)
$$\frac{}{\Delta \triangleright l[\![\mathsf{kill}]\!] \longrightarrow (\Delta - l) \triangleright l[\![\mathbf{0}]\!]}$$

(r-brk)
$$\frac{}{\Delta \triangleright l[\![\mathsf{break}\, k]\!] \longrightarrow (\Delta - l \leftrightarrow k) \triangleright l[\![\mathbf{0}]\!]}\ \Delta \vdash l \leftrightarrow k$$

Table 11. *New Structural Rules for DπF*

...

(s-flip-1) $(\nu\, n : \mathtt{T})(\nu\, m : \mathtt{U})N \equiv (\nu\, m : \mathtt{U})(\nu\, n : \mathtt{T})N \qquad n \notin \mathbf{fn}(\mathtt{U})$

(s-flip-2) $(\nu\, n : \mathtt{T})(\nu\, m : \mathtt{U})N \equiv (\nu\, m : \mathtt{U}{-}n)(\nu\, n : \mathtt{T}{+}m)N \quad n \in \mathbf{fn}(\mathtt{U})$

...

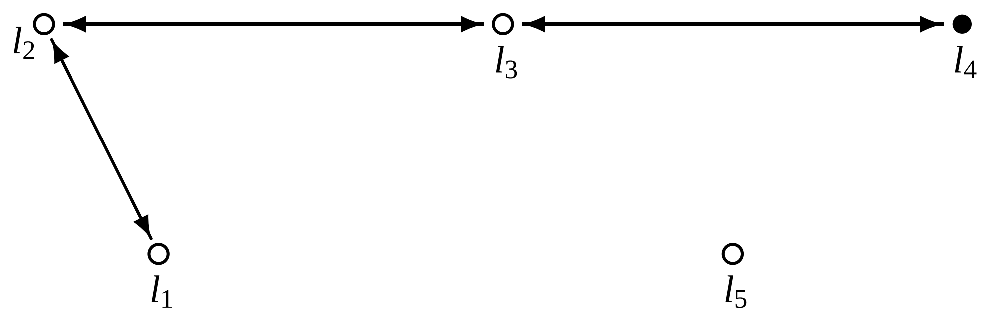
of which is relegated to the Appendix; intuitively $\mathsf{inst}(\mathtt{loc}[\mathtt{S}, \mathtt{C}], l, \Delta)$ returns the location type $\mathtt{loc}[\mathtt{S}, \mathtt{D}]$, where the set of locations $\mathtt{D}$, is the subset of locations in $\mathtt{C} \cup \{l\}$ which are *reachable* from l; this construction is further explained in Example 3.2.1 below. The final new rule in Table 10 is (r-brk), simulating the breaking of a link; the intuition behind the network updating function $\Delta - l \leftrightarrow k$ should be obvious.

To complete the reduction semantics of DπF we need to revise slighlty the rules in Table 5, defining the structural equivalence. The revision is detailed in Table 11; the rule (s-flip) is replaced by the two rules (s-flip-1) and (s-flip-2). This enables us to flip two successively scoped locations even if the first is used in the type of the second, that is there is a link between the two scoped locations.

Example 3.2.1. Consider the system:

$$\mathsf{launchNewLoc} \Leftarrow l_3[\![a!\langle l_1\rangle]\!] \mid l_3[\![a?(x).(\nu k : \mathtt{loc}[\mathsf{a}, \{x, l_2, l_4, l_5\}])P]\!]$$

running on a network Δ consisting of four locations $l_1..l_5$, all of which are alive except l_4, with l_2 connected to l_1 and l_3, and l_3 connected to l_4. Diagrammatically this is easily represented as:



where, open nodes ($\circ$) represent live locations and closed ones ($\bullet$) dead locations; we systematically omit reflexive links in these network diagrams. Formally describing Δ is more tedious:

- $\Delta_{\mathcal{N}}$ is $\{a,\ l_1,\ l_2,\ l_3,\ l_4,\ l_5\}$
- $\Delta_{\mathcal{D}}$ is $\{l_4\}$
- the link set $\Delta_{\mathcal{L}}$ is given by

$$\left\{\begin{array}{l}\langle l_1,l_1\rangle,\ \langle l_2,l_2\rangle,\ \langle l_3,l_3\rangle,\ \langle l_4,l_4\rangle,\ \langle l_5,l_5\rangle,\\ \langle l_1,l_2\rangle,\ \langle l_2,l_3\rangle,\ \langle l_3,l_4\rangle,\ \langle l_2,l_1\rangle,\ \langle l_3,l_2\rangle, \langle l_4,l_3\rangle\end{array}\right\}$$

Clearly there is considerable redundancy in this representation of link sets; $\Delta_{\mathcal{L}}$ can be more reasonably represented as:

$$\Delta_{\mathcal{L}} \;=\; \{l_1 \leftrightarrow l_2,\ l_2 \leftrightarrow l_3,\ l_3 \leftrightarrow l_4,\ l_5 \leftrightarrow l_5\}$$

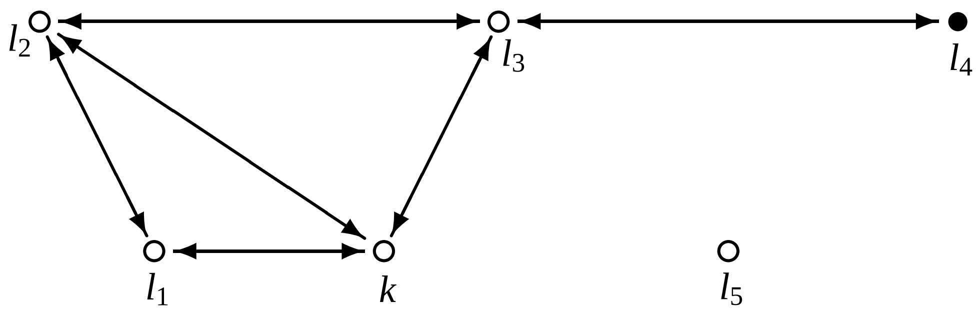
where $l \leftrightarrow k$ denotes the pair of pairs $\langle l,k\rangle$, $\langle k,l\rangle$ together with the reflexive pairs $\langle l,l\rangle$, $\langle k,k\rangle$; in such cases, a reflexive bi-direcitional link $l \leftrightarrow l$ would be used for completely disconnected nodes such as l_5. When we apply the reduction semantics to the configuration $\Delta \triangleright \mathsf{launchNewLoc}$, the rule (r-comm) is used first to allow the communication of the value l_1 along a, and then (r-newl) can be used to launch the declaration of k to the system level. However, the evaluation of $\mathsf{inst}(l_2, \mathtt{loc}[\mathsf{a}, \{l_1, l_2, l_4, l_5\}], \Delta)$ at launching turns out to be $\mathtt{loc}[\mathsf{a}, \{l_1, l_2, l_3\}]$ because:

- the location from where the k is launched, that is l_3, is automatically connected to k.
- l_1 and l_2 are reachable from the location where the new location k is launched, that is $\Delta \vdash l_1, l_2 \leftsquigarrow l_3$; l_2 is directly accessible from l_3 while l_1 is reachable indirectly through l_2
- l_4 and l_5 are not reachable from l_2; l_4 is dead and thus it is not accessible from any other node; l_5 on the other hand, is completely disconnected.

So the resulting configuration is:

$$\Delta \triangleright (\nu\, k : \mathtt{loc}[\mathtt{a}, \{l_1, l_2{,}_3\,\}])\;\; l_3[\![P\{^{l_1}\!/x\}]\!]$$

The network Δ of course does not change, but if we focus on the system $l_2[\![P\{^{l_1}\!/x\}]\!]$, we see that it is running on the internal network represented by:



■

This distinction between the internal networks used by different subsystems has already occurred in the semantics of DπLoc; see the discussion of Example 2.2.2. Nevertheless, we warn to the reader that there will be more serious consequences for DπF, due to the complex nature of reachability that comes into play.

Reduction barbed congruence

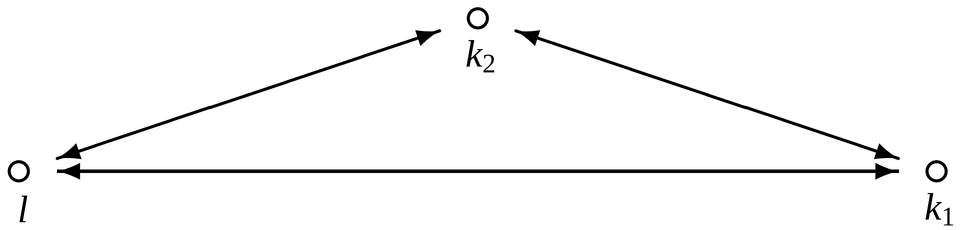
The definition of Reduction barbed congruence, Definition 2.2.6, originally developed for DπLoc configurations, can be adapted to apply also to DπF. The formal definition is delayed to Section 4, but let us use the the same notation,

$$\Delta \models M \cong N \tag{7}$$

to indicate that the systems M and N are equivalent relative to the network Δ; the discussion in the section only relies on an intuitive understanding of this concept.

Let us now reconsider the three implementations of a client server discussed in Example 2.2.9, but this time running on a network with explicit links. For convenience, in this and later examples, we systematically omit channel names from network representations. Moreover, we abbreviate the location type $\mathtt{loc}[\mathtt{a},\mathtt{C}]$ to $\mathtt{loc}[\mathtt{C}]$ when the status of location is understood to be alive.

Example 3.2.2. Let Δ represent the following network:



Formally Δ is determined by letting $\Delta_{\mathcal{N}}$ be $\{l, k_1, k_2\}$, $\Delta_{\mathcal{D}}$ be $\emptyset$ and $\Delta_{\mathcal{L}}$ be $\{l \leftrightarrow k_1,\ l \leftrightarrow k_2,\ k_1 \leftrightarrow k_2\}$.

The distributed server implementations, srvDis and $\mathsf{srv2Rt}$, presented earlier in Example 2.2.9, are no longer reduction barbed congruent relative to Δ, as in this extended setting, the behaviour of systems is also examined in the context of faulty links. It is sufficient to consider the possible barbs in the context of a client such as $l[\![req!\langle l, ret\rangle]\!]$ and a fault inducing context:

$$C_3 \;=\; [-] \,|\, l[\![\mathsf{break}\; k_1]\!]$$

which breaks the link $l \leftrightarrow k_1$. Stated otherwise, if the link $l \leftrightarrow k_1$ breaks, $\mathsf{srv2Rt}$ will still be able to operate normally and barb on $ret@l$; srvDis, on the other hand, may reach a state where it blocks since migrating back and forth from l to k_1 becomes prohibited and as a result, it would not be able to barb $ret@l$. However consider the alternative remote client srvMtr, defined as:

$$\mathsf{srvMtr} \;\Leftarrow\; (\nu\, data)\left(\begin{array}{l} l\left[\!\!\left[req?(x,y).(\nu sync)\left(\begin{array}{l}\mathsf{go}\; k_1.\, data!\langle x, sync\rangle \\ |\, \mathsf{monitor}\; k_1\lceil \mathsf{go}\; k_2, k_1.data!\langle x, sync\rangle\rceil \\ |\, sync?(x).y!\langle x\rangle\end{array}\right)\right]\!\!\right] \\ \\ |\, k_1\left[\!\!\left[data?(x,y).\left(\begin{array}{l}\mathsf{go}\; l.\, y!\langle f(x)\rangle \\ |\, \mathsf{monitor}\; l\lceil \mathsf{go}\; k_2, l.y!\langle f(x)\rangle\rceil\end{array}\right)\right]\!\!\right]\end{array}\right)$$

where the macro $\mathsf{monitor}\; k\lceil P\rceil$, is a process that repeatedly tests the accessibility of a location k from the hosting location, and launches P when k becomes inaccessible. It is formally defined as:

$$\mathsf{monitor}\; k\lceil P\rceil \Leftarrow (\nu\, test\!:\!\mathtt{ch})(\, test!\langle\rangle \,|\, *\, test?().\mathsf{ping}\; k.\, test!\langle\rangle\lceil P\rceil\,)$$

It turns out that $\Delta \models \mathsf{srv2Rt} \cong \mathsf{srvMntr}$ but once again, it is difficult to establish because of the formulation of reduction barbed congruence. ∎

In the next example we examine the interplay between dead nodes and dead links.

Example 3.2.3. Consider the following three networks,

$\Delta_1 = \Delta_l + k\!:\!\mathtt{loc}[\mathtt{d}, \{l\}] =$ 

$\Delta_2 = \Delta_l + k\!:\!\mathtt{loc}[\mathtt{d}, \emptyset] =$ 

$\Delta_3 = \Delta_l + k\!:\!\mathtt{loc}[\mathtt{a}, \emptyset] =$ 

These are the effective networks for the system $l[\![a!\langle k\rangle]\!]$ in the three configurations $\Delta_l \triangleright N_i$, where N_i are defined by

$$\begin{aligned} N_1 &\Leftarrow (\nu\, k : \mathtt{loc}[\mathtt{d}, \{l\}])\; l[\![a!\langle k\rangle]\!] \\ N_2 &\Leftarrow (\nu\, k : \mathtt{loc}[\mathtt{d}, \emptyset])\; l[\![a!\langle k\rangle]\!] \\ N_3 &\Leftarrow (\nu\, k : \mathtt{loc}[\mathtt{a}, \emptyset])\; l[\![a!\langle k\rangle]\!] \end{aligned}$$

and Δ_l is the simple network with one live location l:

$$\Delta_l = \langle \{l, a\}, \emptyset, \{l \leftrightarrow l\}\rangle$$

Intuitively, no observer can distinguish between these three configurations; even though some observer might obtain the scoped name k via the channel a at l, it cannot determine the difference in the state of the network. From rule (l-nmove) we conclude that any attempt to move from l, where the observer would be located, to k will fail. However, such a failure does not yield the observer enough information to determine the exact nature of the fault causing the failure: the observer holding k does not know whether the inaccessibility failure to k was caused by a node fault at k, a link fault between l and k or both. As we shall see later, we will be able to demonstrate $\Delta_l \models N_1 \cong N_2 \cong N_3$. ■

3.3 A labelled transition system for DπF

It would be tempting to mimic the development of Section 2.3 and define a bisimulation equivalence based on actions of the form

$$\Delta \triangleright M \xrightarrow{\mu} \Delta' \triangleright M'$$

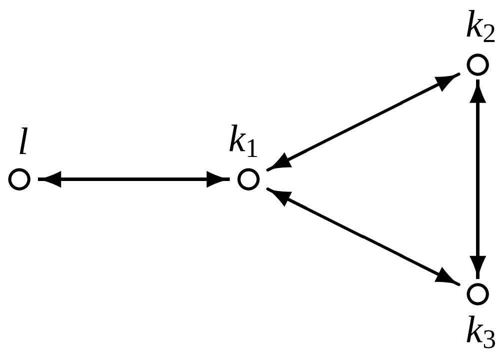
Here we argue that this would not be adequate, at least if the target is to characterise reduction barbed congruence.

Example 3.3.1. Let Δ_l be the network in which there is only one node l which is alive, defined earlier in Example 3.2.3, and consider the system:

$$M_1 \Leftarrow (\nu\, k_1 \!:\! \{l\})\, (\nu\, k_2 \!:\! \{k_1\})\, (\nu\, k_3 \!:\! \{k_1, k_2\})\; l[\![a!\langle k_2, k_3\rangle.P]\!]$$

Note that when M_1 is running on Δ_l, due to the new locations declared, the code

$l[\![a!\langle k_2, k_3\rangle.P]\!]$ is effectively running on the following *internal* network:



(8)

Let us now see to what knowledge of this internal network can be gained by an observer O at site l, such as $l[\![a?(x,y).O(x,y)]\!]$. Note, that prior to any interaction, O is running on the network Δ_l, and thus, is only aware of the unique location l. By inputting along a, it can gain knowledge of the two names k_1 and k_2, thereby evolving to $l[\![O(k_2,k_3)]\!]$. Yet, even though it is in possession of these two names, it cannot discover the link between them, due to the fact that it is not aware of the local name k_1; in other words, it cannot discover the full extent of the internal network (8) above.

This means that, there is now a difference between the actual network being used by the system, (8), and the observer's *view* of that network. Even worse, the formalism of our current network does not allow us to represent this (*external*) observer view of the network. ■

An lts semantics will have to record the differences between the network and the observers view of networks. This requires extra information being recorded in network representations.

Definition 3.3.2 (Effective network representations). An *effective network representation* Σ is a triple $\langle \mathcal{N}, \mathcal{O}, \mathcal{H}\rangle$, where:

- $\mathcal{N}$ is a set of names, as before, divided into $\mathbf{loc}(\mathcal{N})$ and $\mathbf{chan}(\mathcal{N})$,
- $\mathcal{O}$ is a *linkset*, denoting the live locations and links that are *observable* by the context.
- $\mathcal{H}$ is another *linkset*, denoting the live locations and links that are *hidden* (or unreachable) to the context.

The only consistency requirements are that:

1. $\mathbf{dom}(\mathcal{O}) \subseteq \mathbf{loc}(\mathcal{N})$ (the observable live state concerns locations in $\mathcal{N}$)
2. $\mathbf{dom}(\mathcal{H}) \subseteq \mathbf{loc}(\mathcal{N})$ (the hidden live state concerns locations in $\mathcal{N}$)
3. $\mathbf{dom}(\mathcal{O}) \cap \mathbf{dom}(\mathcal{H}) = \emptyset$ (live state cannot be both observable and hidden) ■

The intuition is that an observer running on a network representation Σ, knows about all the names in $\Sigma_{\mathcal{N}}$ and has access to all the locations in $\mathbf{dom}(\mathcal{O})$; as a result, it knows the state of every location in $\mathbf{dom}(\mathcal{O})$ and the live links between these locations. The observer, however, does not have access to the live locations in $\mathbf{dom}(\mathcal{H})$; as a result, it cannot determine the live links between them nor can it distinguish them from dead nodes. Dead nodes are encoded in Σ as $\mathbf{loc}(\mathcal{N})/\mathbf{dom}(\mathcal{O} \cup \mathcal{H})$, that is, all the location names in $\mathcal{N}$ that are not mentioned in either $\mathcal{O}$ or $\mathcal{H}$; these are conveniently denoted as the deadset $\Sigma_{\mathcal{D}}$. We also note that the effective network representation Σ does not represent live links where either end point is a dead node, since these can never be used nor observed. Summarising, Σ hold all the necessary information from the observer's point of view, that is, the names known, $\mathcal{N}$, the state known, $\mathcal{O}$, and the state that can potentially become known in future, as a result of scope extrusion, $\mathcal{H}$.

As usual we use notation such as $\Sigma_{\mathcal{N}}$, $\Sigma_{\mathcal{O}}$ and $\Sigma_{\mathcal{H}}$ to access the fields of Σ and note that any network representation Δ can be translated into an extended network representation $\Sigma(\Delta)$ in the obvious manner:

- the set of names remains unchanged, $\Sigma(\Delta)_{\mathcal{N}} = \Delta_{\mathcal{N}}$
- the accessible state and connections, $\Sigma(\Delta)_{\mathcal{O}}$, is simply $\Delta_{\mathcal{L}}$ less the dead nodes and links to these dead nodes, thus denotes as $\Delta_{\mathcal{L}}/\Delta_{\mathcal{D}}$.
- the hidden state, $\Sigma(\Delta)_{\mathcal{H}}$, is simply the empty set, since Δ does not encode any inaccessible live locations to the observer.

There is also an obvious operation for reducing an extended network representation Σ into a standard one, yielding: $\Delta(\Sigma)$:

- $\Delta(\Sigma)_{\mathcal{N}}$ is inherited directly from Σ.
- $\Delta(\Sigma)_{\mathcal{D}}$ is $\Sigma_{\mathcal{D}}$, which is obtained indirectly from $\mathbf{loc}(\Sigma_{\mathcal{N}})/\mathbf{dom}(\Sigma_{\mathcal{O}} \cup \mathcal{H})$ as stated earlier.
- $\Delta(\Sigma)_{\mathcal{L}}$ is simply $\Sigma_{\mathcal{O}} \cup \Sigma_{\mathcal{H}}$

We note two properties about the operation $\Delta(\Sigma)$; firstly, it does not represent any links to and between dead nodes in $\Delta(\Sigma)_{\mathcal{L}}$; secondly, it merges the accessible and inaccessible state into one single accessible state. Whenever we wish to forget about the distinction between the live accessible nodes and links in Σ and those unknown to the observer, we can transform Σ into the $\Sigma(\Delta(\Sigma))$; this we denote by $\uparrow(\Sigma)$.

For a discussion on how extended network representations allow us to accommodate the observers view in the example just discussed in Example 3.3.1, see Example 3.3.7 below.

Our lts for DπF will be defined in terms of judgements which take the form

$$\Sigma \triangleright M \xrightarrow{\mu} \Sigma' \triangleright M' \tag{9}$$

where the actions μ are the same as those used in the previous section, and both $\Sigma \triangleright M$ (and $\Sigma' \triangleright M'$) is an *effective* configuration, that is all the free names in M occur in $\Sigma_{\mathcal{N}}$. As stated earlier, in the configuration $\Sigma \triangleright M$, where Σ is the effective network $\langle \mathcal{N}, \mathcal{O}, \mathcal{H} \rangle$, only the information in $\mathcal{N}$ and $\mathcal{O}$ is available to an external observer, while the extra information in $\mathcal{H}$ is only available internally to the system M. This division makes more complicated the various operations for extracting information from, and extending networks. As usual, all the formal definitions are relegated to the Appendix, but it is necessary to go into some detail as to how effective networks are augmented with a new location. This will have to take into account the type of the new location, and in particular the existing locations to which it will be linked. For instance, the declaration of the new location $k : \mathtt{loc}[\mathtt{a}, \mathtt{C}]$, requires adding to the network a new live location k, linked to every live location in $\mathtt{C}$.

To simplify the task of defining effective network augmentations, we first define a special form of linkset called *components* together with its related notation, and then express network augmentations in terms of this definition.

Definition 3.3.3 (Component Linksets). We start by adapting the notion of reachability in a network, $\Delta \vdash k \leftarrow\!\!\!\leadsto l$, to linksets, now denoted as $\mathcal{L} \vdash k \leftarrow\!\!\!\leadsto l$. Thus for any linkset $\mathcal{L}$:

- $\mathcal{L} \vdash k \leftarrow l \stackrel{\text{def}}{=} \langle l, k \rangle \in \mathcal{L}$
- $\mathcal{L} \vdash k \leftarrow\!\!\!\leadsto l \stackrel{\text{def}}{=} \mathcal{L} \vdash k \leftarrow l$ or $\exists k'$ such that $\mathcal{L} \vdash k' \leftarrow l$ and $\mathcal{L} \vdash k \leftarrow\!\!\!\leadsto k'$

Based on this intuition, a *component linkset* (or component), denoted by $\mathcal{K}$, is a linkset that is *completely connected*, that is:

$$\forall\, l, k \in \mathbf{dom}(\mathcal{L}) \text{ we have } \mathcal{L} \vdash k \leftarrow\!\!\!\leadsto l$$

We note that any linkset $\mathcal{L}$ can be treated as the union of one or more components, that is:

$$\mathcal{L} = \bigcup_{i=1}^{n} \mathcal{K}_i$$

According to such treatment, a location $l \in \mathbf{dom}(\mathcal{L})$ can be used to identify a particular component $\mathcal{K}$ in the linkset $\mathcal{L}$. We use $\mathcal{L} \leftarrow\!\!\!\leadsto l$ to denote the component in $\mathcal{L}$ identified by a location l and this formally is defined as:

$$\mathcal{L} \leftarrow\!\!\!\leadsto l \stackrel{\text{def}}{=} \{\langle k, k' \rangle \mid \langle k, k' \rangle \in \mathcal{L} \text{ and } \mathcal{L} \vdash k \leftarrow\!\!\!\leadsto l\}$$

Similarly, a set of locations $\{l_1, \ldots, l_n\}$ can identify a number of components in

$\mathcal{L}$, denoted and defined as:

$$\mathcal{L} \leftrightsquigarrow \{l_1, \ldots, l_n\} \stackrel{\text{def}}{=} \bigcup_{i=1}^{n} \mathcal{L} \leftrightsquigarrow l_i$$

Finally, if $C = \{k_1, \ldots, k_n\}$ is a set of locations representing connections, and l is a location such that $l \notin C$, then $l \leftrightarrow C$ denotes the component defined as:

$$l \leftrightarrow C \stackrel{\text{def}}{=} \{\langle k, l\rangle,\ \langle l, k\rangle,\ \langle k, k\rangle \mid k \in C\} \cup \{\langle l, l\rangle\}$$

where locations in C are symmetrically related to l, while l is also related to itself. In the resultant component $l \leftrightarrow C$, all the locations in C in the component $l \leftrightarrow C$ are connected as a star formation to l and as a result, all locations are reachable from one another in *at most* two accesses by going through the central node l. Using previous shorthand notation, we could have alternatively defined $l \leftrightarrow C$ as:

$$l \leftrightarrow C \stackrel{\text{def}}{=} \{l \leftrightarrow k,\ \mid k \in C\}$$

■

Lemma 3.3.4 (Subtracting a Component from a Linkset). For any linkset $\mathcal{L}$ and component $\mathcal{K}$ such that $\mathcal{K} \subseteq \mathcal{L}$, the set $\mathcal{L}/\mathcal{K}$ is also a linkset.

Proof. Immediate from the fact that $\mathcal{L}$ can be expressed as $\bigcup_{i=1}^{n} \mathcal{K}_i$ where $\mathcal{K}$ must be equal to one $\mathcal{K}_i$. Thus, if $\mathcal{K} = \mathcal{K}_j$, the set $\bigcup_{j \neq i=1}^{n} \mathcal{K}_i$, which translates to $\mathcal{L}/\mathcal{K}$, would still be a linkset. □

We now revert our discussion back to effective network representation, and show how the definition of components facilitates the procedure for extending networks.

Definition 3.3.5 (Augmenting effective networks). Let n be fresh to the network Σ and $\mathtt{C}$ be a set of locations such that $\mathtt{C} \subseteq \mathbf{dom}(\Sigma_{\mathcal{O}})$. Then we define the operation $\Sigma + n : \mathtt{T}$ as:

- $\Sigma + n : \mathtt{ch} \stackrel{\text{def}}{=} \langle \Sigma_{\mathcal{N}} \cup \{n\},\ \Sigma_{\mathcal{O}},\ \Sigma_{\mathcal{H}}\rangle$
- $\Sigma + n : \mathtt{loc[d, C]} \stackrel{\text{def}}{=} \langle \Sigma_{\mathcal{N}} \cup \{n\},\ \Sigma_{\mathcal{O}},\ \Sigma_{\mathcal{H}}\rangle$
- $\Sigma + n : \mathtt{loc[a, C]} \stackrel{\text{def}}{=}$

$$\begin{array}{lll} \text{Case} & \mathtt{C} \cap \mathbf{dom}(\Sigma_{\mathcal{O}}) = \emptyset \text{ then} & \langle \Sigma_{\mathcal{N}} \cup \{n\},\ \Sigma_{\mathcal{O}},\ \mathcal{H}'\rangle \\ & & \text{where: } \mathcal{H}' = \Sigma_{\mathcal{H}} \cup (l \leftrightarrow \mathtt{C}) \\ & \mathtt{C} \cap \mathbf{dom}(\Sigma_{\mathcal{O}}) \neq \emptyset \text{ then} & \langle \Sigma_{\mathcal{N}} \cup \{n\},\ \mathcal{O}',\ \mathcal{H}'\rangle \\ & & \text{where: } \mathcal{O}' = \Sigma_{\mathcal{O}} \cup (l \leftrightarrow \mathtt{C}) \cup (\Sigma_{\mathcal{H}} \leftrightsquigarrow \mathtt{C}) \\ & & \quad \text{and } \mathcal{H}' = \Sigma_{\mathcal{H}}/(\Sigma_{\mathcal{H}} \leftrightsquigarrow \mathtt{C}) \end{array}$$

■

In the above definition, extending a network with a fresh channel is trivial; adding a fresh dead node is similarly simple, due to the fact that Σ does not represent dead nodes or links to dead nodes explicitly. The only subcase that deserves some explanation is that of adding fresh *live* nodes. A fresh live location is added to either Σ_O or $\Sigma_{\mathcal{H}}$ depending on its links. If it is not linked to any observable location, $\mathsf{C} \cap \mathbf{dom}(\Sigma_O) = \emptyset$, then the new fresh location is not reachable from the context and is therefore added to $\Sigma_{\mathcal{H}}$. If, on the other hand, it is linked to an observable location, $\mathsf{C} \cap \mathbf{dom}(\Sigma_O) \neq \emptyset$, then it becomes observable as well. There is also the case where the fresh location is linked to both observable and hidden locations, still represented above by the case where $\mathsf{C} \cap \mathbf{dom}(\Sigma_O) \neq \emptyset$; in such a case, the fresh location, together with any components in the hidden state linked to it, that is $\Sigma_{\mathcal{H}} \leftsquigarrow \mathsf{C}$, become observable and thus transfered from $\Sigma_{\mathcal{H}}$ to Σ_O. The following example elucidates this operation for extending effective networks.

Example 3.3.6. Consider the effective network Σ, representing six locations $l, k_1, \ldots, k_5$:

$$\Sigma \;=\; \langle \{l, k_1, k_2, k_3, k_4, k_5\},\ \{l \leftrightarrow l\},\ \{k_1 \leftrightarrow k_2, k_2 \leftrightarrow k_3, k_4 \leftrightarrow k_4\} \rangle$$

Accroding to Definition 3.3.2, l is the only observable location by the context; locations $k_{1..4}$ are alive but not reachable from any observable location while the remaining location, k_5, is dead since it is not in $\mathbf{dom}(\Sigma_O \cup \mathcal{H})$. Moreover, the linkset representing the hidden state, $\Sigma_{\mathcal{H}}$, can be partitioned into two components, $\mathcal{K}_1 = \{k_1 \leftrightarrow k_2, k_2 \leftrightarrow k_3\}$ and $\mathcal{K}_2 = \{k_4 \leftrightarrow k_4\}$ whereas the linkset representing the observable state, Σ_O, can only be partitioned into one component, itself.

The operation $\Sigma + k_0 : \mathtt{loc}[\mathsf{a}, \{l\}]$ would make the fresh location, k_0, observable in the resultant effective network since it is linked to, thus reachable from, the observable location l. On the other hand, the operation $\Sigma + k_0 : \mathtt{loc}[\mathsf{a}, \emptyset]$ would make k_0 hidden since it is a completely disconnected node, just like k_4. The operation $\Sigma + k_0 : \mathtt{loc}[\mathsf{a}, \{k_1\}]$ would still make k_0 hidden in the resultant effective network, since it is only link to the hidden node k_1. Finally, the operation $\Sigma + k_0 : \mathtt{loc}[\mathsf{a}, \{l, k_1\}]$ intersects with both Σ_O and $\Sigma_{\mathcal{H}}$. This means that k_0 itself becomes observable, but as a side effect, the components reachable through it, that is $\Sigma_{\mathcal{H}} \leftsquigarrow \{l, k_1\} = \mathcal{K}_1$, becomes observable as well. Thus, according to Definition 3.3.5, the updated network translates to:

$$\begin{array}{llll}
\multicolumn{4}{l}{\Sigma + k_0 : \mathtt{loc}[\mathsf{a}, \{l, k_1\}] \;=} \\
\langle\ \Sigma_{\mathcal{N}} \cup \{k_0\}, & \Sigma_O \cup (k_0 \leftrightarrow \{l, k_1\}) \cup (\Sigma_{\mathcal{H}} \leftsquigarrow \{l, k_1\}), & \Sigma_{\mathcal{H}} / (\Sigma_{\mathcal{H}} \leftsquigarrow \{l, k_1\}) & \rangle \\
\langle\ \Sigma_{\mathcal{N}} \cup \{k_0\}, & \Sigma_O \cup \{k_0 \leftrightarrow l, k_0 \leftrightarrow k_1\} \cup \mathcal{K}_1, & \Sigma_{\mathcal{H}} / \mathcal{K}_1 & \rangle \\
\langle\ \Sigma_{\mathcal{N}} \cup \{k_0\}, & \{l \leftrightarrow k_0, k_0 \leftrightarrow k_1, k_1 \leftrightarrow k_2, k_2 \leftrightarrow k_3\}, & \{k_4 \leftrightarrow k_4\} & \rangle
\end{array}$$

■

Table 12. *Network Operational Rules(2) for DπF*

Assuming $\Sigma \vdash l : \textbf{alive}$

(l-kill)
$$\frac{}{\Sigma \triangleright l[\![\text{kill}]\!] \xrightarrow{\tau} (\Sigma - l) \triangleright l[\![\mathbf{0}]\!]}$$

(l-brk)
$$\frac{}{\Sigma \triangleright l[\![\text{break } k]\!] \xrightarrow{\tau} \Sigma - (l \leftrightarrow k) \triangleright l[\![\mathbf{0}]\!]} \; \Sigma \vdash l \leftrightarrow k$$

(l-halt)
$$\frac{}{\Sigma \triangleright N \xrightarrow{\text{kill}:l} (\Sigma - l) \triangleright N} \; \Sigma \vdash_{\text{obs}} l : \textbf{alive}$$

(l-disc)
$$\frac{}{\Sigma \triangleright N \xrightarrow{l \not\leftrightarrow k} \Sigma - (l \leftrightarrow k) \triangleright N} \; \Sigma \vdash_{\text{obs}} l \leftrightarrow k$$

(l-go)
$$\frac{}{\Delta \triangleright l[\![\text{go } k.P]\!] \xrightarrow{\tau} \Delta \triangleright k[\![P]\!]} \; \Delta \vdash k \leftarrow l$$

(l-ngo)
$$\frac{}{\Delta \triangleright l[\![\text{go } k.P]\!] \xrightarrow{\tau} \Delta \triangleright k[\![\mathbf{0}]\!]} \; \Delta \nvdash k \leftarrow l$$

(l-ping)
$$\frac{}{\Delta \triangleright l[\![\text{ping } k.P\lceil Q \rceil]\!] \xrightarrow{\tau} \Delta \triangleright l[\![P]\!]} \; \Delta \vdash k \leftarrow l$$

(l-nping)
$$\frac{}{\Delta \triangleright l[\![\text{ping } k.P\lceil Q \rceil]\!] \xrightarrow{\tau} \Delta \triangleright l[\![Q]\!]} \; \Delta \nvdash k \leftarrow l$$

(l-newc)
$$\frac{}{\Delta \triangleright l[\![(\nu\, c : \mathtt{ch})\, P]\!] \xrightarrow{\tau} \Delta \triangleright (\nu\, c : \mathtt{ch})\, l[\![P]\!]}$$

(l-newl)
$$\frac{}{\Delta \triangleright l[\![(\nu\, k : \mathtt{loc}[\mathtt{S}, \mathtt{C}])\, P]\!] \xrightarrow{\tau} \Delta \triangleright (\nu\, k : \mathtt{loc}[\mathtt{S}, \mathtt{D}])\, l[\![P]\!]} \; \mathtt{loc}[\mathtt{S}, \mathtt{D}] = \text{inst}(\mathtt{loc}[\mathtt{S}, \mathtt{C}], l, \Delta)$$

Let us now return to the defintion of our lts for DπF. The transitions between effective configurations (9) are determined by the rules and axioms already given in Table 6 from Section 2.3, together with the new rules in Table 12 and Table 13. Most of the rules in Table 12 are inherited directly from their counterpart reduction rules in Table 10. The new rule is (l-disc) which introduces the new label $l \not\leftrightarrow k$ and models the breaking of a link from the observing context, in the same fashion as (l-fail) in Table 8 models external location killing. Both of these rules are subject to the condition that the location or link where the fault is injected is observable by the context.

The more challenging rules are found in Table 13. Most of these are slightly more subtle versions of the corresponding rules for DπLoc in Table 8; the sub-

Table 13. *Contextual Operational Rules(3) for DπF*

(l-open)

$$\frac{\Sigma + n:\mathtt{T} \triangleright N \xrightarrow{(\tilde{}n\tilde{\mathtt{T}})l:a!\langle V\rangle} \Sigma' \triangleright N'}{\Sigma \triangleright (\nu\, n:\mathtt{T})N \xrightarrow{(n:\mathtt{U},\tilde{}n\tilde{\mathtt{T}})l:a!\langle V\rangle} \Sigma' \triangleright N'}\ l, a \neq n \in V,\ \mathtt{U} = \mathtt{T}/\Sigma_{\mathcal{D}}$$

(l-weak)

$$\frac{\Sigma + n:\mathtt{T} \triangleright N \xrightarrow{(\tilde{}n\tilde{\mathtt{T}})l:a?(V)} \Sigma' \triangleright N'}{\Sigma \triangleright N \xrightarrow{(n:\mathtt{T},\tilde{}n\tilde{\mathtt{T}})l:a?(V)} \Sigma' \triangleright N'}\ l, a \neq n \in V,\ (\Sigma + \tilde{}n\tilde{\mathtt{T}}) \vdash_{\mathsf{obs}} \mathtt{T}$$

(l-rest)

$$\frac{\Sigma + n:\mathtt{T} \triangleright N \xrightarrow{\mu} \Sigma' + n:\mathtt{U} \triangleright N'}{\Sigma \triangleright (\nu\, n:\mathtt{T})N \xrightarrow{\mu} \Sigma' \triangleright (\nu\, n:\mathtt{U})N'}\ n \notin \mathbf{fn}(\mu)$$

(l-rest-typ)

$$\frac{\Sigma + k:\mathtt{T} \triangleright N \xrightarrow{(\tilde{}n\tilde{\mathtt{T}})l:a!\langle V\rangle} (\Sigma + \tilde{n}:\tilde{\mathtt{U}}) + k:\mathtt{U} \triangleright N'}{\Sigma \triangleright (\nu\, k:\mathtt{T})N \xrightarrow{(\tilde{}n\tilde{\mathtt{U}})l:a!\langle V\rangle} \Sigma + \tilde{n}:\tilde{\mathtt{U}} \triangleright (\nu\, k:\mathtt{U})N'}\ l, a \neq k \in \mathbf{fn}(\tilde{\mathtt{T}})$$

(l-par-ctxt)

$$\frac{\Sigma \triangleright N \xrightarrow{\mu} \Sigma' \triangleright N'}{\begin{array}{l}\Sigma \triangleright N|M \xrightarrow{\mu} \Sigma' \triangleright N'|M \\ \Sigma \triangleright M|N \xrightarrow{\mu} \Sigma' \triangleright M|N'\end{array}}\ \Sigma \vdash M$$

(l-par-comm)

$$\frac{\uparrow(\Sigma) \triangleright N \xrightarrow{(\tilde{}n\tilde{\mathtt{T}})l:a!\langle V\rangle} \Sigma' \triangleright N' \qquad \uparrow(\Sigma) \triangleright M \xrightarrow{(\tilde{}n\tilde{\mathtt{T}})l:a?(V)} \Sigma'' \triangleright M'}{\begin{array}{l}\Sigma \triangleright N|M \xrightarrow{\tau} \Sigma \triangleright (\nu\, \tilde{n}:\tilde{\mathtt{T}})(N'|M') \\ \Sigma \triangleright M|N \xrightarrow{\tau} \Sigma \triangleright (\nu\, \tilde{n}:\tilde{\mathtt{T}})(M'|N')\end{array}}$$

tleties are required to deal with the interaction between scoped location names and their occurrence in location types. For instance, the rule (l-open) filters the type of scope extruded locations by removing links to locations that are already dead and that will not affect the effective network Σ; this is done through the operation $\mathtt{T}/\Sigma_{\mathcal{D}}$ defined in the Appendix. A side condition is added to (l-weak), $(\Sigma + \tilde{n}:\tilde{\mathtt{T}}) \vdash_{\mathsf{obs}} \mathtt{T}$, limiting the types of imported fresh locations to only contain locations which are externally accessible, since intuitively, the context can only introduce fresh locations linked to locations it can access. The internal communication rule (l-par-comm) also changes slightly from the one given earlier for DπLoc; communication is defined in terms of the system view ($\uparrow(\Sigma)$) rather than the observer view dictated by Σ. The intuition for this alteration is that internal communication can still occur, even at locations that the observer cannot access, thus we denote the ability to output and input of systems with respect to the maximal observer view $\uparrow(\Sigma)$. Finally, a completely new rule is (l-rest-typ), which restricts the links exported in location types if one endpoint of the link is still scoped. The utility of this rule is illustrated further in the following example. The rules (l-rest) and (l-par-ctxt) remain unchanged for DπLoc.

Example 3.3.7. Let us revisit Example 3.3.1 to see how the effect of the observer O on M_1, running on the effective network Σ_l having only one location l which is alive, that is $\Sigma(\Delta_l)$. This effectively means calculating the result of M_1 performing an output on a at l.

It is easy to see that an application of (l-out), followed by two applications of (l-open) gives

$$\Sigma_l + k_1 : \{l\} \triangleright M_1' \xrightarrow{\alpha} \Sigma_l + k_1 : \{l\} + k_2 : \{k_1\} + k_3 : \{k_1, k_2\} \triangleright l[\![P]\!] \tag{10}$$

where M_1' is $(\nu\, k_2 : \{k_1\})(\nu\, k_3 : \{k_1, k_2\})l[\![a!\langle k_2, k_3\rangle.P]\!]$ and α is the action $(k_2 : \{k_1\}, k_3 : \{k_1, k_2\})l : a!\langle k_2, k_3\rangle$. Note that (l-rest) can not be applied to this judgement, since k_1 occurs free in the action α. However (10) can be re-arranged to read

$$\Sigma_l + k_1 : \{l\} \triangleright M_1' \xrightarrow{\alpha} \Sigma_l + k_2 : \emptyset + k_3 : \{k_2\} + k_1 : \{l, k_1, k_2\} \triangleright l[\![P]\!]$$

moving the addition of location k_1 in the reduct to the outmost position. At this point, (l-rest-typ) can be applied, to give

$$\Sigma_l \triangleright M_1 \xrightarrow{\beta} \Sigma_l + k_2 : \emptyset + k_3 : \{k_2\} \triangleright (\nu\, k_1 : \{l, k_1, k_2\})l[\![P]\!]$$

where β is the action $(k_2 : \emptyset,\ k_3 : \{k_2\})l : a!\langle k_2, k_3\rangle$,that is α filtered from any occurrence of k_1 in its bounded types. Note that the residual network representation, $\Sigma_l + k_2 : \emptyset + k_3 : \{k_2\}$ has a non-trivial internal network, not available to the observer. It evaluates to

$$\langle\{l, k_2, k_3\},\ \{l \leftrightarrow l\},\ \{k_2 \leftrightarrow k_3\}\rangle$$

and may be represented diagramatically by:

where the links of hidden components are denoted with dashed lines. ■

With these actions we can now define in the standard manner a bisimulation equivalence between configurations, which can be used as the basis for contextual reasoning. Let us write

$$\Sigma \models M \approx_{wrong} N$$

to mean that there is a (weak) bisimulation between the configurations $\Sigma \triangleright M$ and $\Sigma \triangleright N$ using the current actions. This new framework can be used to establish positive results. For example, for $\Sigma_{l,k} = \langle \{a, l, k\}, \{l \leftrightarrow k\}, \emptyset \rangle$, one can prove

$$\Sigma_{l,k} \models l[\![\mathsf{ping}\ k.\ a!\langle\rangle\lceil \mathbf{0} \rceil]\!] \approx_{wrong} k[\![\mathsf{go}\ l.a!\langle\rangle]\!]$$

by giving the relation $\mathcal{R}$ defined as:

$$\mathcal{R} = \left\{ \begin{array}{lll} \langle \Sigma_{l,k} \triangleright M & , \Sigma_{l,k} \triangleright N \rangle & \mid \langle M, N \rangle \in \mathcal{R}_{sys} \\ \langle \Sigma_{l,k} - l \triangleright M & , \Sigma_{l,k} - l \triangleright N \rangle & \mid \langle M, N \rangle \in \mathcal{R}_{sys} \\ \langle \Sigma_{l,k} - k \triangleright M & , \Sigma_{l,k} - k \triangleright N \rangle & \mid \langle M, N \rangle \in \mathcal{R}_{sys} \\ \langle \Sigma_{l,k} - l \leftrightarrow k \triangleright M & , \Sigma_{l,k} - l \leftrightarrow k \triangleright N \rangle & \mid \langle M, N \rangle \in \mathcal{R}_{sys} \\ \langle \Sigma_{l,k} - l, l \leftrightarrow k \triangleright M & , \Sigma_{l,k} - l, l \leftrightarrow k \triangleright N \rangle & \mid \langle M, N \rangle \in \mathcal{R}_{sys} \\ \langle \Sigma_{l,k} - k, l \leftrightarrow k \triangleright M & , \Sigma_{l,k} - k, l \leftrightarrow k \triangleright N \rangle & \mid \langle M, N \rangle \in \mathcal{R}_{sys} \\ \langle \Sigma_{l,k} - l, k \triangleright M & , \Sigma_{l,k} - l, k \triangleright N \rangle & \mid \langle M, N \rangle \in \mathcal{R}_{sys} \\ \langle \Sigma_{l,k} - l, k, l \leftrightarrow k \triangleright M & , \Sigma_{l,k} - l, k, l \leftrightarrow k \triangleright N \rangle & \mid \langle M, N \rangle \in \mathcal{R}_{sys} \end{array} \right\}$$

where

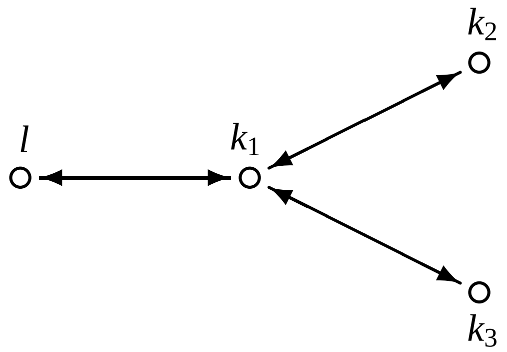
$$\mathcal{R}_{sys} = \left\{ \begin{array}{ll} \langle l[\![\mathsf{ping}\ k.\ a!\langle\rangle\lceil \mathbf{0} \rceil]\!] & , k[\![\mathsf{go}\ l.a!\langle\rangle]\!] \rangle \\ \langle l[\![a!\langle\rangle]\!] & , l[\![a!\langle\rangle]\!] \rangle \\ \langle l[\![\mathbf{0}]\!] & , l[\![\mathbf{0}]\!] \rangle \end{array} \right\}$$

However we can argue, at least informally, that this notion of equivalence is too discriminating and the labels too intentional, because it distinguishes between systems running on a network, where the differences in behaviour are difficult to observe. Problems arise when there is an interplay between *hidden* nodes, links and dead nodes.

Example 3.3.8. Let us consider a slight variation on the system M_1 used in Example 3.3.1 and Example 3.3.7:

$$M_2 \Leftarrow (\nu\, k_1 : \{l\})(\nu\, k_2 : \{k_1\})(\nu\, k_3 : \{k_1\})l[\![a!\langle k_2, k_3\rangle.P]\!]$$

again running on the simple (extended) network Σ_l. Note that here the code $l[\![a!\langle k_2, k_3\rangle.P]\!]$ is effectively running on the following internal network,



a slight variation on that for M_1. It turns out that

$$\Sigma_l \models M_1 \not\approx_{wrong} M_2$$

because the configurations give rise to *different* output actions, on a at l. The difference lies in the types at which the locations k_2 and k_3 are exported; in $\Sigma_l \triangleright M_1$ the output label is $\mu_1 = (k_2 : \emptyset,\ k_3 : \{k_2\})\, l : a!\langle k_2, k_3\rangle$ while with $\Sigma \triangleright M_2$ it is $\mu_2 = (k_2 : \emptyset,\ k_3 : \emptyset) l : a!\langle k_2, k_3\rangle$ - there is a difference in the type associated to the scope extruded location k_3.

However if k_1 does not occur in P, (for example if P is the trivial process $\mathbf{0}$) then k_1 can never be scope extruded to the observer and thus k_2 and k_3 will remain inaccessible in both systems. This means that the presence (or absence) of the link $k_2 \leftrightarrow k_3$ can never be checked and thus there should be no observable difference between M_1 and M_2 running on Σ. ■

Problems also arise when dealing with the presence of dead nodes.

Example 3.3.9. Let us reconsider the three configurations $\Sigma_l \triangleright N_i$ for $i = 1..3$ from Example 3.2.3. We have already argued that these three configurations should not be distinguished. However, our lts specifies that all three configurations perform the output with different scope extrusion labels, namely:

$$\Sigma_l \triangleright N_1 \xrightarrow{(k:\mathtt{loc}[\mathtt{d},\{l\}])l:a!\langle k\rangle} \langle\{l, k\}, \{l \leftrightarrow l\}, \emptyset\rangle \triangleright l[\![\mathbf{0}]\!]$$

$$\Sigma_l \triangleright N_2 \xrightarrow{(k:\mathtt{loc}[\mathtt{d},\emptyset])l:a!\langle k\rangle} \langle\{l, k\}, \{l \leftrightarrow l\}, \emptyset\rangle \triangleright l[\![\mathbf{0}]\!]$$

$$\Sigma_l \triangleright N_3 \xrightarrow{(k:\mathtt{loc}[\mathtt{a},\emptyset])l:a!\langle k\rangle} \langle\{l, k\}, \{l \leftrightarrow l\}, \{k \leftrightarrow k\}\rangle \triangleright l[\![\mathbf{0}]\!]$$

Therefore they will be distinguished by the bisimulation equivalence which uses these actions. ■

In order to obtain a bisimulation equivalence which coincides with reduction barbed congruence it is necessary to abstract away from some of the information contained in the types of newly exported location names.

3.4 A bisimulation equivalence for DπF

We first outline the revision to our labelled actions. Currently these are of the form $\mathtt{T} = \mathtt{ch}$ or $\mathtt{loc}[\mathtt{A}, \{k_1, \ldots k_n\}]$, where the latter indicates the liveness of a location and the nodes k_i to which it is linked. We change these to new types of the form $\mathtt{L}, \mathtt{K} = \{l_1 \leftrightarrow k_1, \ldots, l_i \leftrightarrow k_i\}$ where $\mathtt{L}, \mathtt{K}$ are components. Intuitively, these represent the new live nodes and links, which are made accessible to observers by the extrusion of the new location. Alternatively, this is the information which is added to the observable part of the network representation, Σ_O, as a result of the action. We have already developed the necessary technology to define these new types, in Definition 3.3.5.

Definition 3.4.1 (A (derived) labelled transition system for DπF). This consists of a collection of actions $\Sigma \triangleright N \overset{\mu}{\longmapsto} \Sigma' \triangleright N'$, where μ takes one of the forms:

- (internal action) - τ
- (bounded input) - $(\tilde{n} : \tilde{\mathtt{L}})l : a?(V)$
- (bounded output) - $(\tilde{n} : \tilde{\mathtt{L}})l : a!\langle V\rangle$
- (external location kill) - $\mathsf{kill} : l$
- (external link break) - $l \not\leftrightarrow k$ ■

The transitions in the derived lts for DπF are defined as the least relations satisfying the axioms and rules in Table 6 of Section 2.3, Tables 12 and 13 given earlier in this section and Table 14. The rules (l-deriv-2) and (l-deriv-3) transform the types of bound names using the function $\mathsf{lnk}(\tilde{n} : \tilde{\mathtt{T}}, \Sigma)$ defined below in Definition 3.4.2.

Definition 3.4.2 (Link Types). Let us first define the function $\mathsf{lnk}()$ for single typed names $n : \mathtt{T}$ and then extend it to sequences of typed names $\tilde{n} : \tilde{\mathtt{T}}$. Recalling Definition 3.3.5 for augmenting networks, the only case where $\Sigma + n : \mathtt{T}$ adds anything to the observable state of the effective network, Σ_O, is when $\mathtt{T} = \mathtt{loc}[\mathtt{a}, \mathtt{C}]$ where $\mathtt{C} \cap \mathbf{dom}(\Sigma_O) \neq \emptyset$. In such a case, we add the new location n with all its live connections denoted as $l \leftrightarrow \mathtt{C}$, and any components that were previously unreachable but have become reachable from Σ_O as a result of n, denoted as $\Sigma_{\mathcal{H}} \leftrightsquigarrow \mathtt{C}$.

When $\mathtt{C} \cap \mathbf{dom}(\Sigma_O) = \emptyset$, then the node added is unreachable to the observing contexts and we add n and its live links to $\Sigma_{\mathcal{H}}$ but nothing to Σ_O; if $\mathtt{T} = \mathtt{loc}[\mathtt{d}, \mathtt{C}]$

Table 14. *The derived lts for DπF*

(l-deriv-1)

$$\frac{\Sigma \triangleright N \xrightarrow{\mu} \Sigma' \triangleright N'}{\Sigma \triangleright N \overset{\mu}{\longmapsto} \Sigma' \triangleright N'}\ \mu \in \{\tau, \mathsf{kill} : l, l \not\leftrightarrow k\}$$

(l-deriv-2)

$$\frac{\Sigma \triangleright N \xrightarrow{(\tilde{n}:\tilde{\mathtt{T}})l:a!\langle V\rangle} \Sigma' \triangleright N'}{\Sigma \triangleright N \overset{(\tilde{n}:\tilde{\mathtt{L}})l:a!\langle V\rangle}{\longmapsto} \Sigma' \triangleright N'}\ \tilde{\mathtt{L}} = \mathsf{lnk}(\tilde{n}:\tilde{\mathtt{T}}, \Sigma)$$

(l-deriv-3)

$$\frac{\Sigma \triangleright N \xrightarrow{(\tilde{n}:\tilde{\mathtt{T}})l:a?(V)} \Sigma' \triangleright N'}{\Sigma \triangleright N \overset{(\tilde{n}:\tilde{\mathtt{L}})l:a?(V)}{\longmapsto} \Sigma' \triangleright N'}\ \tilde{\mathtt{L}} = \mathsf{lnk}(\tilde{n}:\tilde{\mathtt{T}}, \Sigma)$$

then we do not add anything to either $\Sigma_{\mathcal{O}}$ or $\Sigma_{\mathcal{H}}$ as is the case for $\mathtt{T} = \mathtt{ch}$. Based on this definition of $\Sigma{+}n{:}\mathtt{T}$, we give the following definition for $\mathsf{lnk}(n{:}\mathtt{T}, \Sigma)$:

$$\mathsf{lnk}(n{:}\mathtt{T}, \Sigma) \stackrel{\text{def}}{=} \begin{cases} (n \leftrightarrow \mathtt{C}) \cup (\Sigma_{\mathcal{H}} \leftrightsquigarrow \mathtt{C}) & \text{if } \mathtt{T} = \mathtt{loc}[\mathtt{a}, \mathtt{C}] \text{ and } \mathtt{C} \cap \mathbf{loc}(\Sigma_{\mathcal{O}}) \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases}$$

This function is extended to sequences of typed names in the obvious manner:

$$\mathsf{lnk}(n, \tilde{n}{:}\mathtt{T}, \tilde{\mathtt{T}}, \Sigma) \quad = \quad \mathsf{lnk}(n{:}\mathtt{T}, \Sigma), \mathsf{lnk}(\tilde{n}{:}\tilde{\mathtt{T}}, \Sigma')$$

where Σ' denotes $\Sigma + n : \mathtt{T}$. ■

These revised actions give rise to a new (weak) bisimulation equivalence over configurations, $\approx$, defined in the usual way, but based on *derived actions*. We use

$$\Sigma \models M \approx N$$

to mean that the configurations $\Sigma \triangleright M$ and $\Sigma \triangleright N$ are bisimilar.

Example 3.4.3. Here we re-examine the systems in Example 3.3.8 and Example 3.3.9. We recall that in Example 3.3.8 we had the following actions with respect to the original lts: -

$$\Sigma_l \triangleright M_1 \xrightarrow{\mu_1} \Sigma + k_2{:}\emptyset + k_3{:}\{k_2\} \triangleright (\nu\, k_1{:}\{l, k_2, k_3\})\, l[\![P]\!]$$

$$\Sigma_l \triangleright M_2 \xrightarrow{\mu_2} \Sigma + k_2{:}\emptyset + k_3{:}\emptyset \triangleright (\nu\, k_1{:}\{l, k_2, k_3\})\, l[\![P]\!]$$

But Σ_l contains only one accessible node l; extending it with the new node k_2, linked to nothing does not increase the set of accessible nodes. Further increasing it with a new node k_3, linked to the inaccessible k_2 (in the case of $\Sigma \triangleright M_1$) also leads to no increase in the accessible nodes. Correspondingly, the calculations of $\mathsf{lnk}(k_2{:}\emptyset, \Sigma)$ and $\mathsf{lnk}(k_3{:}\{k_2\}, (\Sigma + k_2{:}\emptyset))$ both lead to the empty link set.

Formally, we get the derived action

$$\Sigma \triangleright M_1 \overset{\alpha}{\longmapsto} \Sigma + k_2 : \emptyset + k_3 : \{k_2\} \triangleright (\nu\, k_1 : \{l, k_2, k_3\})\, l[\![P]\!]$$

where α is $(k_2 : \emptyset,\ k_3 : \emptyset)l : a!\langle k_2, k_3\rangle$. Similar calculations gives exactly the same derived action from M_2:

$$\Sigma \triangleright M_2 \overset{\alpha}{\longmapsto} \Sigma + k_2 : \emptyset + k_3 : \emptyset \triangleright (\nu\, k_1 : \{l, k_2, k_3\})\, l[\![P]\!]$$

Furthermore, if P contains no occurrence of k_1, we can go on to show

$$\Sigma + k_2 : \emptyset + k_3 : \{k_2\} \triangleright (\nu\, k_1 : \{l, k_2, k_3\}) l[\![P]\!] \ \approx\ \Sigma + k_2 : \emptyset + k_3 : \emptyset \triangleright (\nu\, k_1 : \{l, k_2, k_3\}) l[\![P]\!]$$

On the other hand, if P is $a!\langle k_1\rangle$, the subsequent transitions are different:

$$((\Sigma + k_2 : \emptyset) + k_3 : \{k_2\}) \triangleright (\nu\, k_1 : \{l, k_2, k_3\}) l[\![P]\!] \overset{\beta_1}{\longmapsto} \ldots$$
$$((\Sigma + k_2 : \emptyset) + k_3 : \emptyset) \triangleright (\nu\, k_1 : \{l, k_2, k_3\}) l[\![P]\!] \overset{\beta_2}{\longmapsto} \ldots$$

where

$$\beta_1 \text{ is } (k_1 : \{k_1 \leftrightarrow k_2, k_1 \leftrightarrow k_3, k_2 \leftrightarrow k_3\}) l : a!\langle k_1\rangle$$
$$\beta_2 \text{ is } (k_1 : \{k_1 \leftrightarrow k_2, k_1 \leftrightarrow k_3\}) l : a!\langle k_1\rangle$$

We note that the link type associated with β_1 includes the additional component $\{k_2 \leftrightarrow k_3\}$, that was previously hidden, but is now made accessible as a result of scope extruding k_1; β_2 on the other hand, does not have this information in its link type. Based on this discrepancy between β_1 and β_2 we have

$$\Sigma_l \models M_1 \not\approx M_2$$

Revisiting Example 3.3.9, the three different actions of $\Sigma_l \triangleright N_1$, $\Sigma_l \triangleright N_2$ and $\Sigma_l \triangleright N_3$ now abstract to the same action $\Sigma_l \triangleright N_i \overset{\alpha}{\longmapsto} \ldots \triangleright l[\![\mathbf{0}]\!]$ for $i = 1..3$ where α is the label $(k : \emptyset)l : a!\langle k\rangle$. Thus we have

$$\Sigma_l \models N_i \approx N_j \quad \text{where } i, j = 1..3$$

as required. ■

4 Full-Abstraction

The purpose of this section is to show that our revised bisimulation equivalence is the correct one, in the sense that it coincides with some contextual equivalence appropriate to DπF; we need a generalisation of Theorem 2.3.8. This means developing a notion of reduction barbed congruence for DπF, similar to that in Definition 2.2.6 for DπLoc.

4.1 Reduction barbed congruence for DπF

The key to the definition is the isolation of the externally observable information in an extended environment. We use $\mathcal{I}$ to range over *knowledge representations*, pairs $\langle\mathcal{N},\mathcal{O}\rangle$ where

- $\mathcal{N}$ is a set of names, as usual divided into $\mathbf{loc}(\mathcal{N})$ and $\mathbf{chan}(\mathcal{N})$,
- $\mathcal{O}$ is a linkset over $\mathcal{N}$.

These can be obtained from effective networks in the obvious manner:

$$\mathcal{I}(\Sigma) \stackrel{\text{def}}{=} \langle\Sigma_{\mathcal{N}},\Sigma_{\mathcal{O}}\rangle$$

The key property of this subset of the information in a network representation is that it is preserved by derived actions:

Definition 4.1.1 (Action residuals). We overload the partial function after from Definition 2.3.2 so that now it ranges over knowledge representations $\mathcal{I}$ and external derived actions μ. The newly defined partial function returns knowledge representation, defined as:

- $\mathcal{I}$ after $(\tilde{n} : \tilde{\mathrm{L}})l : a!\langle V\rangle$ is defined as $\mathcal{I} + \tilde{n} : \tilde{\mathrm{L}}$
- $\mathcal{I}$ after $(\tilde{n} : \tilde{\mathrm{L}})l : a?(V)$ is defined as $\mathcal{I} + \tilde{n} : \tilde{\mathrm{L}}$
- $\mathcal{I}$ after $\mathsf{kill} : l$ is defined as $\mathcal{I} - l$
- $\mathcal{I}$ after $l \leftrightarrow\!\!\!\!/\; k$ is defined as $\mathcal{I} - l \leftrightarrow k$ ■

Proposition 4.1.2. If $\Sigma \triangleright N \stackrel{\mu}{\longmapsto} \Sigma' \triangleright N'$ where μ is a derived external action, then $\mathcal{I}(\Sigma')$ coincides with $\mathcal{I}(\Sigma)$ after μ

Proof. A straightforward induction on the inference of $\Sigma \triangleright N \stackrel{\mu}{\longmapsto} \Sigma' \triangleright N'$. □

We find appropriate to use $\mathcal{I}$ as a means to define *the right circumstances*, necessary to allow an action μ to happen. Thus, we define the following predicate.

Definition 4.1.3 (Action Conditions). The predicate allows is defined over knowledge representations and action labels as:

- $\mathcal{I}$ allows τ is always true
- $\mathcal{I}$ allows $(\tilde{n} : \tilde{\mathrm{L}})l : a!\langle V\rangle$ is true whenever $\mathcal{I} \vdash l : \mathbf{alive}$
- $\mathcal{I}$ allows $(\tilde{n} : \tilde{\mathrm{L}})l : a?(V)$ is true whenever $\mathcal{I} \vdash l : \mathbf{alive}$, $V \subseteq \mathcal{I}_{\mathcal{N}}$ and $\mathbf{dom}(\tilde{\mathrm{L}}) \subseteq (\mathbf{dom}(\mathcal{I}_{\mathcal{O}}) \cup \tilde{n})$
- $\mathcal{I}$ allows $\mathsf{kill} : l$ is true whenever $\mathcal{I} \vdash l : \mathbf{alive}$
- $\mathcal{I}$ allows $l \leftrightarrow\!\!\!\!/\; k$ is true whenever $\mathcal{I} \vdash l : \mathbf{alive}, k : \mathbf{alive}$ ■

The following proposition states that the definition of allows is adequate with respect to our derived actions.

Proposition 4.1.4 (Adequacy of Allows). If $\Sigma \triangleright N \stackrel{\mu}{\longmapsto} \Sigma' \triangleright N'$ then $\mathcal{I}(\Sigma)$ allows μ.

Proof. By induction on the derivation of $\Sigma \triangleright N \stackrel{\mu}{\longmapsto} \Sigma' \triangleright N'$. □

We next establish that external actions of a configuration $\Pi \triangleright N$ are a function of the system N and the knowledge representation $\mathcal{I}(\Sigma)$. Before we prove this, we prove a lemma relating actions with the structure of the systems.

Lemma 4.1.5 (Derived Actions and Systems).

- if $\Sigma \triangleright N \xmapsto{(\tilde{n}\tilde{\mathtt{L}})l:a!\langle V\rangle} \Sigma + \tilde{n}:\tilde{\mathtt{T}} \triangleright N'$ where $\tilde{\mathtt{L}} = \mathsf{lnk}(\tilde{n}:\tilde{\mathtt{T}}, \Sigma)$ then
  - $N \equiv (\nu\, \tilde{n}:\tilde{\mathtt{T}})(\nu\, \tilde{m}:\tilde{\mathtt{U}})M|l[\![a!\langle V\rangle.P]\!]$
  - $N' \equiv (\nu\, \tilde{m}:\tilde{\mathtt{U}})M|l[\![P]\!]$
- if $\Sigma \triangleright N \xmapsto{(\tilde{n}\tilde{\mathtt{L}})l:a?(V)} \Sigma + \tilde{n}:\tilde{\mathtt{T}} \triangleright N'$ where $\tilde{\mathtt{L}} = \mathsf{lnk}(\tilde{n}:\tilde{\mathtt{T}}, \Sigma)$ then
  - $N \equiv (\nu\, \tilde{m}:\tilde{\mathtt{U}})M|l[\![a?(X).P]\!]$
  - $N' \equiv (\nu\, \tilde{m}:\tilde{\mathtt{U}})M|l[\![P\{^V/_X\}]\!]$
- if $\Sigma \triangleright N \stackrel{\tau}{\longmapsto} \Sigma' \triangleright N'$ where $\Sigma \vdash l : \textbf{alive}$ and $\Sigma' \nvdash l : \textbf{alive}$ then
  - $N \equiv N'|l[\![\mathsf{kill}]\!]$
- if $\Sigma \triangleright N \stackrel{\tau}{\longmapsto} \Sigma' \triangleright N'$ where $\Sigma \vdash l \leftrightarrow k$ and $\Sigma' \nvdash l \leftrightarrow k$ then
  - $N \equiv N'|l[\![\mathsf{break}\ k]\!]$ or $N \equiv N'|k[\![\mathsf{break}\ l]\!]$

Proof. A straightforward induction on the inference of $\Sigma \triangleright N \xrightarrow{(\tilde{n}\tilde{\mathtt{L}})l:a!\langle V\rangle} \Sigma' \triangleright N'$, $\Sigma \triangleright N \xrightarrow{(\tilde{n}\tilde{\mathtt{L}})l:a?(V)} \Sigma' \triangleright N'$ and $\Sigma \triangleright N \stackrel{\tau}{\longrightarrow} \Sigma' \triangleright N'$. □

Proposition 4.1.6. If $\Sigma \triangleright N \stackrel{\mu}{\longmapsto} \Sigma' \triangleright N'$ where μ is an external action, and $\mathcal{I}(\Sigma'')$ allows μ for some Σ'', then $\Sigma'' \triangleright N \stackrel{\mu}{\longmapsto} \Sigma''' \triangleright N'$ for some Σ'''

Proof. By induction on the inference of $\Sigma \triangleright N \stackrel{\mu}{\longmapsto} \Sigma' \triangleright N'$, using Lemma 4.1.5 to infer the structure of N from μ. □

These lemmas indicate that these knowledge representations are the appropriate level of abstractions at which to generalise Definitions 2.2.3, 2.2.5 and 2.2.6 to DπF.

Definition 4.1.7 (Typed Relations for DπF). A *typed relation* over extended configurations is a binary relation between such configurations with the property that

$$\Sigma \triangleright M \;\mathcal{R}\; \Sigma' \triangleright N \;\text{ implies }\; \mathcal{I}(\Sigma) = \mathcal{I}(\Sigma')$$

We can mimic the notation in Definition 2.2.3 by writing

$$\mathcal{I} \models \Sigma \triangleright M \;\mathcal{R}\; \Sigma' \triangleright N$$

to mean that systems $\Sigma \triangleright M$ and $\Sigma' \triangleright N$ are related by $\mathcal{R}$ and that both $\mathcal{I}(\Sigma)$ and $\mathcal{I}(\Sigma')$ coincide with $\mathcal{I}$. ■

The definition of contextuality depends on what a given $\mathcal{I}$ allows to be observable; for this we adapt Definition 2.2.4.

Definition 4.1.8 (Observables). For any $\mathcal{I}$ let:

- $\mathcal{I} \vdash l : \mathbf{alive}$, if l is in $\mathbf{dom}(\mathcal{I}_O)$; this implies that l is not only alive, but in the accessible part of any Σ such that $\mathcal{I}(\Sigma)$ coincides with $\mathcal{I}$.
- $\mathcal{I} \vdash l \leftrightarrow k$, if $l \leftrightarrow k \in \mathcal{I}_O$; this implies that the link $l \leftrightarrow k$ is not only alive, but in the accessible part of any Σ such that $\mathcal{I}(\Sigma)$ coincides with $\mathcal{I}$.
- $\mathcal{I} \vdash \mathtt{T}$ if $\mathtt{T}$ is either $\mathtt{ch}$ or $\mathtt{loc}[\mathtt{a}, \mathtt{C}]$ such that $\mathtt{C} \subseteq \mathbf{dom}(\mathcal{I}_O)$.

We can now define the relation $\mathcal{I} \vdash O$ as:

- $\mathcal{I} \vdash l[\![P]\!]$ if $\mathbf{fn}(P) \subseteq \mathcal{I}_N$ and $\mathcal{I} \vdash l : \mathbf{alive}$
- $\mathcal{I} \vdash (\nu n : \mathtt{T})N$ if $\mathcal{I} \vdash \mathtt{T}$ and $(\mathcal{I} + n : \mathtt{T}) \vdash_{\mathsf{obs}} N$
- $\mathcal{I} \vdash M \mid N$ if $\mathcal{I} \vdash M$ and $\mathcal{I} \vdash N$

We can now adapt the notation of Definition 2.2.4 as:

$$\Delta \vdash_{\mathsf{obs}} l : \mathbf{alive},\; l \leftrightarrow k,\; \mathtt{T},\; O \;\stackrel{\text{def}}{=}\; \mathcal{I}(\Sigma(\Delta)) \vdash l : \mathbf{alive},\; l \leftrightarrow k,\; \mathtt{T},\; O$$
$$\Sigma \vdash_{\mathsf{obs}} l : \mathbf{alive},\; l \leftrightarrow k,\; \mathtt{T},\; O \;\stackrel{\text{def}}{=}\; \mathcal{I}(\Sigma) \vdash l : \mathbf{alive},\; l \leftrightarrow k,\; \mathtt{T},\; O$$

The intuition of $\Delta \vdash_{\mathsf{obs}} O$ and $\Sigma \vdash_{\mathsf{obs}} O$ are still the same as that of Definition 2.2.4: an observer O is restricted to the observable network. However, the updated definition reflects the fact that the observable network is now not only defined in terms of live nodes but live, *reachable* nodes. ■

As a result of this adaptation, we can carry forward to the section the definition of contextual typed relations, defined earlier in 2.2.5. However, before we go on an define reduction barbed congruence for DπF terms, we need also to update the notion of a barb; a barb is observable by the context in DπF, if the location at which the barb occurs is alive *and observable*.

Definition 4.1.9. $\Sigma \triangleright N \Downarrow_{a@l}$ denotes an *observable barb* exhibited by the configuration $\Sigma \triangleright N$, on channel a at location l. Formally, it means that $\Delta(\Sigma) \triangleright N \longrightarrow^* \Delta(\Sigma') \triangleright N'$ for some $\Sigma' \triangleright N'$ such that $N' \equiv M | l[\![a!\langle V \rangle.Q]\!]$ and $\mathcal{I}\Sigma \vdash_{\mathsf{obs}} l : \textbf{alive}$. ■

With these modifications, Definition 2.2.6 can be applied to obtain a definition of *reduction barbed congruence* for DπF, which we denote by

$$\mathcal{I} \models \Sigma_1 \triangleright M_1 \;\cong\; \Sigma_2 \triangleright M_2 \quad \text{whenever} \quad \mathcal{I}(\Sigma_1) = \mathcal{I}(\Sigma_2)$$

Note that this enables us to compare aribtrary configurations, $\Sigma_1 \triangleright M_1$ and $\Sigma_2 \triangleright M_2$, but it can be specialised to simply comparing systems running on the same network. Let us write

$$\Sigma \models M \cong N$$

to mean that $\mathcal{I}(\Sigma) \models \Sigma \triangleright M \cong \Sigma \triangleright N$. Then, for example, the informal notation (7) used in Section 3.2 can be taken to mean

$$\Sigma(\Delta) \vdash M \cong N$$

The second main result of the paper can now be stated:

Theorem 4.1.10. Suppose $\mathcal{I}(\Sigma_1) = \mathcal{I}(\Sigma_2) = \mathcal{I}$, for any effective configurations $\Sigma_1 \triangleright M_1$, $\Sigma_2 \triangleright M_2$ in DπF. Then:

$$\mathcal{I} \models \Sigma_1 \triangleright M_1 \;\cong\; \Sigma_1 \triangleright M_1 \quad \text{if and only if} \quad \Sigma_1 \triangleright M_1 \approx \Sigma_2 \triangleright M_2$$

This general result can also be specialised to the notation for comparing systems relative to a given network:

Corollary 4.1.11. In DπF, $\Sigma \models N \cong M$ if and only if $\Sigma \models N \;\approx\; M$. ■

The proof of the general theorem , which is quite complex, is detailed in the following two sections. The first section outlines the proof for *soundness*, that is, the adequacy of the derived action bisimulation as a means show that two configurations are reduction barbed congruent:

$$\Sigma_1 \triangleright M_1 \approx \Sigma_2 \triangleright M_2 \quad \text{implies} \quad \mathcal{I} \models \Sigma_1 \triangleright M_1 \;\cong\; \Sigma_1 \triangleright M_1$$

The second section outlines the proof for *completness*, that is, for any two configurations that are reduction barbed congruent, we can give a derived action bisimulation to show this:

$$\mathcal{I} \models \Sigma_1 \triangleright M_1 \;\cong\; \Sigma_1 \triangleright M_1 \quad \text{implies} \quad \Sigma_1 \triangleright M_1 \approx \Sigma_2 \triangleright M_2$$

4.2 Soundness

The main task in proving that derived action bisimulation is sound is showing that $\approx$ is contextual. In addition to this, we also need to prove some preliminary results relating our lts with the reduction semantics of DπF.

We start by proving that the derived lts is *closed* over well formed effective configurations. We prove this with the aid of the following lemma, stating that there is also a special relationship between silent actions and residual networks.

Lemma 4.2.1. Internal transitions do not change the state of the network, unless a kill or a $\mathsf{break}\ l$ process in the configuration itself is consumed. Stated otherwise, if $\Sigma \triangleright N \overset{\tau}{\longmapsto} \Sigma' \triangleright N'$ then Σ' is either:-

1. Σ
2. $\Sigma - l$
3. $\Sigma - l \leftrightarrow k$

Proof. A straightforward induction on the inference of $\Sigma \triangleright N \overset{\tau}{\longmapsto} \Sigma' \triangleright N'$. □

Proposition 4.2.2 (Closure). The derived lts given in Definition 3.4.1 forms a binary relation between well-defined effective configurations. Stated otherwise, if $\Sigma \vdash N$ and $\Sigma \triangleright N \overset{\mu}{\longmapsto} \Sigma' \triangleright N'$ then $\Sigma' \vdash N'$.

Proof. By case analysis on the form of μ. We use Proposition 4.1.2 when μ is an external action and Lemma B.0.1 in sub-cases where we need to show that $\Sigma + n : \mathsf{T}$ is still a valid effective network. When μ is an internal action, $\mu = \tau$, we use Lemma 4.2.1. □

The next important sanity check for our lts is that our formulation of *internal activity*, namely $\overset{\tau}{\longmapsto}$, is in agreement, in some sense, with the reduction semantics.

Proposition 4.2.3 (Reductions correspond to τ-actions).

- $\Sigma \triangleright N \longrightarrow \Sigma' \triangleright N'$ implies $\Sigma \triangleright N \overset{\tau}{\longmapsto} \Sigma' \triangleright N''$ for some $N'' \equiv N'$
- $\Sigma \triangleright N \overset{\tau}{\longmapsto} \Sigma' \triangleright N'$ implies $\Sigma \triangleright N \longrightarrow \Sigma' \triangleright N'$

Proof. The proof for the first clause is by induction on why $\Sigma \triangleright N \longrightarrow \Sigma' \triangleright N'$. The proof for the second clause is also by induction. Since the internal transition rule (l-par-comm) is defined in terms of input and output actions, we make use of Lemma 4.1.5 in our induction. □

We now embark on the main task of this section, that of showing that our bisimulation, $\approx$, is contextual. This proof relies heavily on the Composition and Decomposition Lemmas stated below, explaining how actions can be composed of, or decomposed into, other actions. Both Composition and Decomposition Lemmas make use of the following (specific) lemma, which is a slight variation on Proposition 4.1.6; we note that we could not have used Proposition 4.1.6 in this case because the type of the bound input action changes as shown below.

Lemma 4.2.4 (Input actions and the maximal observer view).

- If $\Sigma \triangleright N \xmapsto{(\tilde{n}\tilde{\mathtt{K}})l:a?(V)} \Sigma' \triangleright N'$ then $\uparrow(\Sigma) \triangleright N \xmapsto{(\tilde{n}\tilde{\mathtt{L}})l:a?(V)} \Sigma'' \triangleright N'$ where $\tilde{\mathtt{K}} = \tilde{\mathtt{L}}/\mathbf{dom}(\Sigma_{\mathcal{H}})$.
- If $\uparrow(\Sigma) \triangleright N \xmapsto{(\tilde{n}\tilde{\mathtt{L}})l:a?(V)} \Sigma' \triangleright N'$ and $\mathcal{I}(\Sigma) \vdash l : \mathbf{alive}$ then $\Sigma \triangleright N \xmapsto{(\tilde{n}\tilde{\mathtt{K}})l:a?(V)} \Sigma'' \triangleright N'$ where $\tilde{\mathtt{K}} = \tilde{\mathtt{L}}/\mathbf{dom}(\Sigma_{\mathcal{H}})$.

Proof. The proof uses Lemma 4.1.5 to infer the structure of N and the progresses by induction on the structure of N, similar to the proof for Proposition 4.1.6. □

Lemma 4.2.5 (Composition).

- Suppose $\Sigma \triangleright M \xmapsto{\mu} \Sigma' \triangleright M'$. If $\Sigma \vdash N$ for arbitrary system N, then $\Sigma \triangleright M|N \xmapsto{\mu} \Sigma' \triangleright M'|N$ and $\Sigma \triangleright N|M \xmapsto{\mu} \Sigma \triangleright N|M$.
- Suppose $\Sigma \triangleright M \xmapsto{(\tilde{n}\tilde{\mathtt{L}})l:a!\langle V\rangle} \Sigma' \triangleright M'$ and $\Sigma \triangleright N \xmapsto{(\tilde{n}\tilde{\mathtt{K}})l:a?(V)} \Sigma'' \triangleright N'$ where $\tilde{\mathtt{K}} = \tilde{\mathtt{L}}/\mathbf{dom}(\Sigma_{\mathcal{H}})$. Then
  - $\Sigma \triangleright M|N \xmapsto{\tau} \Sigma \triangleright (\nu\, \tilde{n} : \tilde{\mathtt{T}})M'|N'$ where $\tilde{\mathtt{L}} = \mathsf{lnk}(\tilde{n} : \tilde{\mathtt{T}}, \Sigma)$
  - $\Sigma \triangleright N|M \xmapsto{\tau} \Sigma \triangleright (\nu\, \tilde{n} : \tilde{\mathtt{T}})N'|M'$ where $\tilde{\mathtt{L}} = \mathsf{lnk}(\tilde{n} : \tilde{\mathtt{T}}, \Sigma)$

Proof. The proof for the first clause is trivial, by using (l-par-ctxt). We here outline the proof for the second clause. From the hypothesis

$$\Sigma \triangleright M \xmapsto{(\tilde{n}\tilde{\mathtt{L}})l:a!\langle V\rangle} \Sigma + \tilde{n} : \tilde{\mathtt{T}} \triangleright M' \tag{11}$$

and Proposition 4.1.4 we know $\mathcal{I}(\Sigma)$ allows $(\tilde{n} : \tilde{\mathtt{L}})l : a!\langle V\rangle$ and thus $\mathcal{I}(\Sigma) \vdash l :$ **alive**. It is obvious that $\mathcal{I}(\uparrow(\Sigma)) \vdash l : \mathbf{alive}$ as well and hence

$$\mathcal{I}(\uparrow(\Sigma)) \text{ allows } (\tilde{n} : \tilde{\mathtt{L}})l : a!\langle V\rangle \tag{12}$$

From (11), (12) and Proposition 4.1.6 we derive

$$\uparrow(\Sigma) \triangleright M \xmapsto{(\tilde{n}\tilde{\mathtt{L}})l:a!\langle V\rangle} \uparrow(\Sigma) + \tilde{n} : \tilde{\mathtt{T}} \triangleright M'$$

and from (l-deriv-2) we conclude

$$\uparrow(\Sigma) \triangleright M \xrightarrow{(\tilde{n}\tilde{\mathtt{T}})l:a!\langle V\rangle} \uparrow(\Sigma) + \tilde{n} : \tilde{\mathtt{T}} \triangleright M' \tag{13}$$

From the hypotheses $\Sigma \triangleright N \xmapsto{(\tilde{n}\tilde{\mathtt{K}})l:a?(V)} \Sigma'' \triangleright N'$ and $\tilde{\mathtt{K}} = \tilde{\mathtt{L}}/\mathbf{dom}(\Sigma_{\mathcal{H}})$ and Lemma 4.2.4 we immediately derive

$$\uparrow(\Sigma) \triangleright N \xmapsto{(\tilde{n}\tilde{\mathtt{L}})l:a?(V)} \uparrow(\Sigma) + \tilde{n} : \tilde{\mathtt{T}} \triangleright N'$$

and by (l-deriv-3) we derive

$$\uparrow(\Sigma) \triangleright N \xrightarrow{(\tilde{n}\tilde{\mathtt{T}})l:a?(V)} \uparrow(\Sigma) + \tilde{n} : \tilde{\mathtt{T}} \triangleright N' \tag{14}$$

Hence, by (13), (14), (l-par-comm) and (l-deriv-1) we conclude

$$\Sigma \triangleright M|N \overset{\tau}{\longmapsto} \Sigma \triangleright (\nu\, \tilde{n}:\tilde{\mathrm{T}})M'|N'$$
$$\Sigma \triangleright N|M \overset{\tau}{\longmapsto} \Sigma \triangleright (\nu\, \tilde{n}:\tilde{\mathrm{T}})N'|M'$$

as required. □

Lemma 4.2.6 (Decomposition). Suppose $\Sigma \triangleright M|N \overset{\mu}{\longmapsto} \Sigma' \triangleright M'$ where $\Sigma \vdash_{\mathsf{obs}} M$ or $\Sigma \vdash_{\mathsf{obs}} N$. Then, one of the following conditions hold:

1. M' is $M''|N$, where $\Sigma \triangleright M \overset{\mu}{\longmapsto} \Sigma' \triangleright M''$.
2. M' is $M|N'$ and $\Sigma \triangleright N \overset{\mu}{\longmapsto} \Sigma' \triangleright N'$.
3. M' is $(\nu\, \tilde{n}:\tilde{\mathrm{T}})M''|N'$, μ is τ, $\Sigma' = \Sigma$ and either
   - $\Sigma \triangleright M \xmapsto{(\tilde{n}\tilde{\mathrm{L}})l:a!\langle V\rangle} \Sigma'' \triangleright M''$ and $\Sigma \triangleright N \xmapsto{(\tilde{n}\tilde{\mathrm{K}})l:a?(V)} \Sigma''' \triangleright N'$
   - $\Sigma \triangleright M \xmapsto{(\tilde{n}\tilde{\mathrm{K}})l:a?(V)} \Sigma'' \triangleright M''$ and $\Sigma \triangleright N \xmapsto{(\tilde{n}\tilde{\mathrm{L}})l:a!\langle V\rangle} \Sigma''' \triangleright N'$

   where $\tilde{\mathrm{K}} = \tilde{\mathrm{L}}/\mathbf{dom}(\Sigma_{\mathcal{H}})$

Proof. The proof progressed by induction on the derivation of $\Sigma \triangleright M|N \overset{\mu}{\longmapsto} \Sigma' \triangleright M'$. We focus on the case where $\mu = \tau$, and the last two rules used in our derivation were (l-deriv-1) and (l-par-comm). From the inductive hypothesis of (l-par-comm) we derive

$$\Sigma' = \Sigma \tag{15}$$
$$M' \text{ is } (\nu\, \tilde{n}:\tilde{\mathrm{T}})M'|N' \tag{16}$$
$$\uparrow(\Sigma) \triangleright M \xrightarrow{(\tilde{n}\tilde{\mathrm{T}})l:a!\langle V\rangle} \uparrow(\Sigma) + \tilde{n}:\tilde{\mathrm{T}} \triangleright M' \tag{17}$$
$$\uparrow(\Sigma) \triangleright N \xrightarrow{(\tilde{n}\tilde{\mathrm{T}})l:a?(V)} \uparrow(\Sigma) + \tilde{n}:\tilde{\mathrm{T}} \triangleright N' \tag{18}$$

or viceversa. From (17), (18), (l-deriv-2) and (l-deriv-3) we get

$$\uparrow(\Sigma) \triangleright M \xmapsto{(\tilde{n}\tilde{\mathrm{L}})l:a!\langle V\rangle} \uparrow(\Sigma) + \tilde{n}:\tilde{\mathrm{T}} \triangleright M' \tag{19}$$
$$\uparrow(\Sigma) \triangleright N \xmapsto{(\tilde{n}\tilde{\mathrm{L}})l:a?(V)} \uparrow(\Sigma) + \tilde{n}:\tilde{\mathrm{T}} \triangleright N' \tag{20}$$

From the assumption that $\Sigma \vdash_{\mathsf{obs}} M$ or $\Sigma \vdash_{\mathsf{obs}} N$ we derive $\Sigma \vdash_{\mathsf{obs}} l:\mathbf{alive}$ meaning

$$\mathcal{I}(\Sigma) \vdash l:\mathbf{alive} \tag{21}$$

From (21) we derive

$$\mathcal{I}(\Sigma) \text{ allows } (\tilde{n} : \tilde{\mathrm{T}})l : a!\langle V\rangle \tag{22}$$

and by (19), (22) and Proposition 4.1.6 we deduce

$$\Sigma \triangleright M \xmapsto{(\tilde{n}\tilde{\mathrm{L}})l:a!\langle V\rangle} \Sigma'' \triangleright M'$$

Moreover, by (20), (21) and Lemma 4.2.4 we deduce

$$\uparrow(\Sigma) \triangleright N \xrightarrow{(\tilde{n}\tilde{\mathtt{K}})l{:}a?(V)} \uparrow(\Sigma) + \tilde{n} : \tilde{\mathtt{T}} \triangleright N'$$

where $\tilde{\mathtt{K}} = \tilde{\mathtt{L}}/\mathbf{dom}(\Sigma_{\mathcal{H}})$ as required. □

We now turn our attention to the actual proof for the main proposition of this section, namely that bisimulation, $\approx$, is contextual. We prove this by inductively defining the largest contextual relation whose base element are bisimilar configurations and then show its closure with respect to our derived actions. Based on such a proof, we still require three (specific) lemmas to help us stitch up this proof and guarantee closure. The first lemma is prompted by the first two conditions of the Decomposition Lemma 4.2.6, namely that observing code may alter the state of the network by inducing failure. We thus need the following lemma to guarantee closure.

Lemma 4.2.7. Suppose $\Sigma_1 \triangleright M_1 \approx \Sigma_2 \triangleright M_2$. Then there exists some M_2', M_2'' such that:

- $\Sigma_2 \triangleright M_2 \overset{\hat{\tau}}{\Longmapsto} \Sigma_2 \triangleright M_2'$ and $(\Sigma_2 - l) \triangleright M_2' \overset{\tau}{\Longmapsto} (\Sigma_2 - l) \triangleright M_2''$ such that $(\Sigma_1 - l) \triangleright M_1 \approx (\Sigma_2 - l) \triangleright M_2''$
- $\Sigma_2 \triangleright M_2 \overset{\hat{\tau}}{\Longmapsto} \Sigma_2 \triangleright M_2'$ and $(\Sigma_2 - l \leftrightarrow k) \triangleright M_2' \overset{\tau}{\Longmapsto} (\Sigma_2 - l \leftrightarrow k) \triangleright M_2''$ such that $(\Sigma_1 - l \leftrightarrow k) \triangleright M_1 \approx (\Sigma_2 - l \leftrightarrow k) \triangleright M_2''$

Proof. We here prove the first clause and leave the second similar clause for the interested reader. If $\Sigma_1 \nvdash l : \mathbf{alive}$ then $\Sigma_1 - l$ is simply Σ_1 and the result is trivial. Otherwise $\Sigma_1 \triangleright M_1 \overset{\mathsf{kill}:l}{\longmapsto} \Sigma_1 - l \triangleright M_1$ and hence $\Sigma_2 \triangleright M_2 \overset{\mathsf{kill}:l}{\Longmapsto} \Sigma_2 - l \triangleright M''$ for some $\Sigma_2 - l \triangleright M''$ such that $\Sigma_1 - l \triangleright M_1 \approx \Sigma_2 - l \triangleright M''$. By expanding our the derivation $\Sigma_2 \triangleright M \overset{\mathsf{kill}:l}{\Longmapsto} (\Sigma_2 - l) \triangleright M''$ we get the required missing M' to complete the proof. □

The next two proofs concern observing code. In the definition of contextual relations, we validate observer code, O, with respect to the external view of a network Σ, that is $\Sigma \vdash_{\mathsf{obs}} O$. We here prove that such a relationship is preserved by actions and network extensions that may involve revealing more hidden components to the observer.

Lemma 4.2.8 (Observers and Actions). If $\Sigma \vdash_{\mathsf{obs}} O$ and $\Sigma \triangleright O \overset{\mu}{\longmapsto} \Sigma\ \mathsf{after}\ \mu \triangleright O'$ then $(\Sigma\ \mathsf{after}\ \mu) \vdash_{\mathsf{obs}} O'$.

Proof. The proof is similar to that of Proposition4.2.2. We use Lemma 4.1.5 to infer the structure of O, O' from μ and Lemma 4.1.2 to infer the structure of $\Sigma\ \mathsf{after}\ \mu$ and then show that $(\Sigma\ \mathsf{after}\ \mu) \vdash_{\mathsf{obs}} O'$. □

Lemma 4.2.9 (Observers and Network extensions). If $\Sigma{+}n:\mathtt{U} \vdash_{\mathsf{obs}} O$ where $\Sigma \vdash_{\mathsf{obs}} \mathtt{U}$, that is n is only linked to locations in the observable part of Σ and thus no hidden state is revealed as a result of the extension, then $\Sigma{+}n:\mathtt{T} \vdash_{\mathsf{obs}} O$ for any $\mathtt{T}$ where $\mathtt{U} = \mathtt{T}/\mathbf{dom}(\Sigma_{\mathcal{H}})$.

Proof. The proof progresses by a simple induction on the structure of O. □

We are finally in a position to prove that our bisimulation, $\approx$, is a contextual relation, according to Definition 2.2.5.

Proposition 4.2.10 (Contextuality of Behavioural Equivalence). If two configurations are bisimilar, they are also bisimilar under any context. Stated otherwise, $\mathcal{I} \models \Sigma_1 \triangleright M_1 \approx \Sigma_2 \triangleright M_2$ implies that for $\mathcal{I} \vdash O, \mathtt{T}$ and n fresh in $\mathcal{I}$ we have:

- $\mathcal{I} \models \Sigma_1 \triangleright M_1|O \approx \Sigma_2 \triangleright M_2|O$ and $\mathcal{I} \models \Sigma_1 \triangleright O|M_1 \approx \Sigma_2 \triangleright O|M_2$
- $\mathcal{I}{+}n:\mathtt{T} \models \Sigma_1{+}n:\mathtt{T} \triangleright M_1 \approx \Sigma_2{+}n:\mathtt{T} \triangleright M_2$

Proof. The proof progresses by the inductive definition a relation $\mathcal{R}$ as the largest typed relation over configurations satisfying:

$$\mathcal{R} = \left\{ \begin{array}{ll} \langle \Sigma_1 \triangleright M_1,\ \Sigma_2 \triangleright M_2 \rangle & \mid \Sigma_1 \triangleright M_1 \approx \Sigma_2 \triangleright M_2 \\[2ex] \begin{array}{l} \langle \Sigma_1 \triangleright M_1|O,\ \Sigma_2 \triangleright M_2|O \rangle \\ \langle \Sigma_1 \triangleright O|M_1,\ \Sigma_2 \triangleright O|M_2 \rangle \end{array} & \left| \ \Sigma_1 \triangleright M_1 \mathcal{R} \Sigma_2 \triangleright M_2 \right. \\[4ex] \langle \Sigma_1{+}n:\mathtt{T} \triangleright M_1|O,\ \Sigma_2{+}n:\mathtt{T} \triangleright M_2|O \rangle & \left| \begin{array}{l} \mathcal{I} \models \Sigma_1 \triangleright M_1\ \mathcal{R}\ \Sigma_2 \triangleright M_2, \\ \mathcal{I} \vdash \mathtt{T} \text{ and } n \text{ is fresh} \end{array} \right. \\[4ex] \langle \Sigma_1 \triangleright (\nu\, n:\mathtt{T})M_1,\ \Sigma_2 \triangleright (\nu\, n:\mathtt{U})M_2 \rangle & \mid \Sigma_1{+}n:\mathtt{T} \triangleright M_1 \mathcal{R} \Sigma_2{+}n:\mathtt{U} \triangleright M_2 \end{array} \right\}$$

and showing that $\mathcal{R} \subseteq \approx$; since $\approx$ is the biggest possible relation, this would mean that it is contextual. We note that our definition of contextual relations, Definition 2.2.5, would amount to a special case of the contexts defined for $\mathcal{R}$ because it is only defined in terms of the second and third cases of the relation $\mathcal{R}$, namely contexts involving more systems in parallel and contexts involving a bigger network. The fourth and last context case, that of name scoping, is required to ensure the closure of $\mathcal{R}$. All this is fairly standard with the exception that the type at which names are scoped in the fourth case may not be the same because of the potentially different hidden states in Σ_1 and Σ_2.

Before we delve into the actual proof we also note that Lemma 4.2.7 can be easily extended from $\approx$ to $\mathcal{R}$ as:

Lemma 4.2.11. If $\Sigma_1 \triangleright M_1 \;\mathcal{R}\; \Sigma_2 \triangleright M_2$, then there exist some M_2', M_2'' such that:

- $\Sigma_2 \triangleright M_2 \stackrel{\widehat{\tau}}{\Longrightarrow} \Sigma_2 \triangleright M_2'$ and $\Sigma_2 - l \triangleright M_2' \stackrel{\tau}{\Longrightarrow} \Sigma_2 - l \triangleright M_2''$, where $\Sigma_1 - l \triangleright M_1 \mathcal{R} \Sigma_2 - l \triangleright M_2''$
- $\Sigma_2 \triangleright M_2 \stackrel{\widehat{\tau}}{\Longrightarrow} \Sigma_2 \triangleright M_2'$ and $\Sigma_2 - l \leftrightarrow k \triangleright M_2' \stackrel{\tau}{\Longrightarrow} \Sigma_2 - l \leftrightarrow k \triangleright M_2''$, where $\Sigma_1 - l \leftrightarrow k \triangleright M_1 \;\mathcal{R}\; \Sigma_2 - l \leftrightarrow k \triangleright M_2''$

The proof for the above is by induction on why $\Sigma_1 \triangleright M_1 \mathcal{R} \Sigma_2 \triangleright M_2$; the base case follows from Lemma 4.2.7 and the three inductive cases are straightforward.

To prove that $\mathcal{R}$ is a bisimulation, we take an arbitrary $\mathcal{I} \models \Sigma_1 \triangleright M_1 \;\mathcal{R}\; \Sigma_2 \triangleright M_2$ and any action $\Sigma_1 \triangleright M_1 \stackrel{\mu}{\longmapsto} \Sigma_1' \triangleright M_1'$; we then have to show that $\Sigma_2 \triangleright M_2$ can match this move by performing a weak action $\Sigma_2 \triangleright M_2 \stackrel{\hat{\mu}}{\Longmapsto} \Sigma_2' \triangleright M_2'$ such that $\mathcal{I}' \models \Sigma_1' \triangleright M_1' \;\; \mathcal{R} \;\; \Sigma_2' \triangleright M_2'$. The proof progress by induction on why $\mathcal{I} \models \Sigma_1 \triangleright M_1 \;\mathcal{R}\; \Sigma_2 \triangleright M_2$; The first case, that is if $\mathcal{I} \models \Sigma_1 \triangleright M_1 \approx \Sigma_2 \triangleright M_2$ is immediate; the remaining three cases require a bit more work. We here focus on the second case, where

$$\Sigma_1 \triangleright M_1|O \,\mathcal{R}\, \Sigma_2 \triangleright M_2|O \text{ because } \mathcal{I} \models \Sigma_1 \triangleright M_1 \;\mathcal{R}\; \Sigma_2 \triangleright M_2 \text{ and } \mathcal{I} \vdash O \tag{23}$$

which is also the most involving and leave the remaining two cases for the interested reader.

We thus assume $\Sigma_1 \triangleright M_1|O \stackrel{\mu}{\longmapsto} \Sigma_1' \triangleright M_1'$. We decompose this action using the Decomposition Lemma 4.2.6 and focus on the most difficult case, where

$$M_1' \text{ is } (\nu\, \tilde{n}:\tilde{\mathtt{T}})M_1'|O',\; \mu \text{ is } \tau \text{ and } \Sigma_1' = \Sigma_1 \tag{24}$$

$$\Sigma_1 \triangleright M_1 \xmapsto{(\tilde{n}:\tilde{\mathtt{L}})l:a!\langle V\rangle} \Sigma_1 + \tilde{n}:\tilde{\mathtt{T}} \triangleright M_1' \tag{25}$$

$$\Sigma_1 \triangleright O \xmapsto{(\tilde{n}:\tilde{\mathtt{K}})l:a?(V)} \Sigma_1 + \tilde{n}:\tilde{\mathtt{U}} \triangleright O' \text{ where } \tilde{\mathtt{U}} = \tilde{\mathtt{T}}/\mathbf{dom}(\Sigma_{1\mathcal{H}}) \tag{26}$$

From (23) and (25) we derive the matching weak action

$$\Sigma_2 \triangleright M_2 \stackrel{(\tilde{n}:\tilde{\mathtt{L}})l:a!\langle V\rangle}{\Longmapsto} \Sigma_2' + \tilde{n}:\tilde{\mathtt{W}} \triangleright M_2' \;\mathcal{R}\; \Sigma_1 + \tilde{n}:\tilde{\mathtt{T}} \triangleright M_1' \tag{27}$$

where we note the different types $\tilde{\mathtt{T}}$ and $\tilde{\mathtt{W}}$ at which the two networks Σ_1 and Σ_2 are updated; there are updates to the hidden part of the networks which we abstract away in the linktype $\tilde{\mathtt{L}}$. From (27) and the hypothesis of (l-deriv-2) we obtain

$$\Sigma_2 \triangleright M_2 \stackrel{(\tilde{n}:\tilde{\mathtt{W}})l:a!\langle V\rangle}{\Longrightarrow} \Sigma_2' + \tilde{n}:\tilde{\mathtt{W}} \triangleright M_2'$$

which can be decomposed as

$$\Sigma_2 \triangleright M_2 \quad \Longrightarrow \quad \Sigma_2'' \triangleright M_2'' \tag{28}$$

$$\Sigma_2'' \triangleright M_2'' \xrightarrow{(\tilde{n}\tilde{\mathtt{W}})l:a!\langle V\rangle} \Sigma_2''+\tilde{n}:\tilde{\mathtt{W}} \triangleright M_2''' \tag{29}$$

$$\Sigma_2''+\tilde{n}:\tilde{\mathtt{W}} \triangleright M_2''' \quad \Longrightarrow \quad \Sigma_2'+\tilde{n}:\tilde{\mathtt{W}} \triangleright M_2' \tag{30}$$

From (28), $\mathcal{I} \vdash O$ and (l-par-ctxt) we get

$$\Sigma_2 \triangleright M_2|O \Longrightarrow \Sigma_2'' \triangleright M_2''|O \tag{31}$$

From the fact that $\mathcal{I}(\Sigma_1) = \mathcal{I}(\Sigma_2)$ and $\mathcal{I}(\Sigma_1+\tilde{n}:\tilde{\mathtt{T}}) = \mathcal{I}(\Sigma_2'+\tilde{n}:\tilde{\mathtt{W}})$ from (27) we know that the visible part of Σ_2'' and Σ_2' did not change as a result of the silent transitions in (28) and (30) and thus

$$\mathcal{I}(\Sigma_2'') \;=\; \mathcal{I}(\Sigma_2') \;=\; \mathcal{I}(\Sigma_2) \;=\; \mathcal{I}(\Sigma_1) \tag{32}$$

and by (32), (26) and Lemma 4.1.6 we get

$$\Sigma_2'' \triangleright O \xrightarrow{(\tilde{n}\tilde{\mathtt{U}})l:a?(V)} \Sigma_2''+\tilde{n}:\tilde{\mathtt{U}} \triangleright O' \text{ where } \tilde{\mathtt{U}} = \tilde{\mathtt{W}}/\mathbf{dom}(\Sigma_{2\,\mathcal{H}}'') \tag{33}$$

At this point we note that from (32) and (23) we derive

$$\Sigma_2'' \vdash_{\mathsf{obs}} O \tag{34}$$

and from (33), (34), Lemma 4.2.8 and Lemma 4.2.9 we obtain

$$\mathcal{I}(\Sigma_2''+\tilde{n}:\tilde{\mathtt{U}}) \vdash O' \text{ and } \mathcal{I}(\Sigma_2''+\tilde{n}:\tilde{\mathtt{W}}) \vdash O' \tag{35}$$

Combining the derived action of (29) using (l-deriv-2), the derived action of (33) using (l-deriv-3), (34), and the Composition Lemma 4.2.5, we obtain

$$\Sigma_2'' \triangleright M_2''|O \xrightarrow{\tau} \Sigma_2'' \triangleright (\nu\,\tilde{n}:\tilde{\mathtt{W}})M_2'''|O' \tag{36}$$

From (30), (35) and (l-par-ctxt) we obtain

$$\Sigma_2''+\tilde{n}:\tilde{\mathtt{W}} \triangleright M_2'''|O' \Longrightarrow \Sigma_2'+\tilde{n}:\tilde{\mathtt{W}} \triangleright M_2'|O'$$

and by applying (l-rest) we get

$$\Sigma_2'' \triangleright (\nu\,\tilde{n}:\tilde{\mathtt{W}})M_2'''|O' \Longrightarrow \Sigma_2' \triangleright (\nu\,\tilde{n}:\tilde{\mathtt{W}})M_2'|O' \tag{37}$$

and thus by combining (31), (36) and (37) and then applying (l-deriv-1) we obtain the matching move

$$\Sigma_2 \triangleright M_2|O \stackrel{\tau}{\Longmapsto} \Sigma_2' \triangleright (\nu\,\tilde{n}:\tilde{\mathtt{W}})M_2'|O' \tag{38}$$

The only thing remaining is to show that the two residuals are in $\mathcal{R}$, that is

$$\Sigma_1 \triangleright (\nu\,\tilde{n}:\tilde{\mathtt{T}})M_1'|O' \;\;\mathcal{R}\;\; \Sigma_2' \triangleright (\nu\,\tilde{n}:\tilde{\mathtt{W}})M_2'|O'$$

From (27) we know

$$\mathcal{I}' \models \Sigma_1+\tilde{n}:\tilde{\mathtt{T}} \triangleright M_1' \quad \mathcal{R} \quad \Sigma_2'+\tilde{n}:\tilde{\mathtt{W}} \triangleright M_2' \tag{39}$$

and from (35) and (39) we deduce $\mathcal{I}' \vdash O'$ and thus from the definition of $\mathcal{R}$ we obtain

$$\mathcal{I}' \models \Sigma_1 + \tilde{n}:\tilde{\mathtt{T}} \triangleright M_1'|O' \quad \mathcal{R} \quad \Sigma_2' + \tilde{n}:\tilde{\mathtt{W}} \triangleright M_2'|O'$$

and again from the last case of the definition of $\mathcal{R}$

$$\mathcal{I} \;\models\; \Sigma_1 \triangleright (\nu\,\tilde{n}:\tilde{\mathtt{T}})M_1'|O' \quad \mathcal{R} \quad \Sigma_2' \triangleright (\nu\,\tilde{n}:\tilde{\mathtt{W}})M_2'|O'$$

as required. □

We now conclude this section by showing that bisimulation is sound with respect to reduction barbed congruence.

Proposition 4.2.12 (Soundness).

$$\mathcal{I} \models \Sigma_1 \triangleright M_1 \approx \Sigma_2 \triangleright M_2 \text{ implies } \mathcal{I} \models \Sigma_1 \triangleright M_1 \cong \Sigma_2 \triangleright M_2$$

Proof. To prove the above statement, it is sufficient to check that $\approx$ satisfies the defining properties of $\cong$. It is obviously reduction closed, from the relationship between τ-actions and the reduction semantics given in Proposition 4.2.3. Barb preserving is also straightforward, from Proposition 4.2.3 and the direct relationship between barbs and output actions. Finally, Proposition 4.2.10 proves that $\approx$ is also contextual. □

4.3 Completness

In this section we prove that our bisimulation is also *complete* with respect to reduction barbed congruence. This entails showing that reduction barbed congruence is preserved by actions, based on the proof developed earlier in [9, 8]. At the heart of this proof, we show that the effect of each external action can be mimicked precisely by a specific context, a concept we refer to as *definability*.

We start this section by proving an obvious, though not explicit, property stating that reduction babrbed congruence is preserved by scoping.

Proposition 4.3.1 (Scoping and reduction barbed congruence). If two configurations are reduction barbed congruent, scoping a channel or location name on both sides would still yield two reduction barbed congruent configurations. Stated otherwise,

$$(\Sigma_M + n:\mathtt{T}) \triangleright M \cong (\Sigma_N + n:\mathtt{U}) \triangleright N \text{ implies } \Sigma_M \triangleright (\nu\,n:\mathtt{T})M \cong \Sigma_N \triangleright (\nu\,n:\mathtt{U})N$$

Proof. We define the relation $\mathcal{R}$ as:

$$\mathcal{R} = \Big\{ \langle \Sigma_M \triangleright (\nu\,n:\mathtt{T})M,\, \Sigma_N \triangleright (\nu\,n:\mathtt{U})N \rangle \quad \Big| (\Sigma_M + n:\mathtt{T}) \triangleright M \cong (\Sigma_N + n:\mathtt{U}) \triangleright N \Big\}$$

and prove that $\mathcal{R}$ has the defining properties of $\cong$. It is clearly reduction closed using (r-ctxt-res); it is also easy to show it is barb preserving since $\Sigma_M \triangleright (\nu\,n:$

$\mathtt{T})M \Downarrow_{a@l}$ implies $(\Sigma_M + n:\mathtt{T}) \triangleright M \Downarrow_{a@l}$. Finally, contextuality is also trivial. As an example, assume $\mathcal{I}(\Sigma_M) \vdash O$ and we have to show that

$$\Sigma_M \triangleright O \,|\, (\nu\, n:\mathtt{T})(M)\, \mathcal{R}\, \Sigma_N \triangleright O \,|\, (\nu\, n:\mathtt{U})N.$$

It is clear that $\Sigma_M + n:\mathtt{T} \vdash_{\mathsf{obs}} O$ and $\Sigma_N + n:\mathtt{U} \vdash_{\mathsf{obs}} O$ and thus by contextuality of $\cong$, we have $(\Sigma_M + n:\mathtt{T}) \triangleright O \,|\, M \cong (\Sigma_N + n:\mathtt{U}) \triangleright O \,|\, N$ from which the result follows. □

Our external actions can affect both the system part of our configuration as well as the network representation and the main differences between the definability proofs presented here and those in [9, 8] lie in the effects an action has on the network representation. In the following proofs, we model an action's effect on a network using two different kinds of new construct introduced in DπF; the first kind of constructs *induce* faults as changes in the network representation and these include kill and $\mathsf{break}\ l$; the second kind *observe* the current state of the network and the only example is the $\mathsf{ping}\ l.P\lceil Q\rceil$ construct. The first lemma we consider, establishes a relationship between the labels $\mathsf{kill}: l$ and $l \nleftrightarrow k$ and the constructs inducing faults in the observable network representation; this proof is complicated by the asynchronous nature of the constructs kill and $\mathsf{break}\ l$.

Lemma 4.3.2 (Inducing faults).

- Suppose $\Sigma \vdash_{\mathsf{obs}} l : \mathbf{alive}$. Then:
  - $\Sigma \triangleright N \xmapsto{\mathsf{kill}:l} \Sigma' \triangleright N'$ implies $\Sigma \triangleright N|l[\![\mathsf{kill}]\!] \longrightarrow \Sigma' \triangleright N'$
  - $\Sigma \triangleright N|l[\![\mathsf{kill}]\!] \longrightarrow \Sigma' \triangleright N'$, where $\Sigma' \nvdash_{\mathsf{obs}} l : \mathbf{alive}$ implies $\Sigma \triangleright N \xmapsto{\mathsf{kill}:l} \Sigma' \triangleright N''$ such that $N' \equiv N''$
- Suppose $\Sigma \vdash_{\mathsf{obs}} l \leftrightarrow k$. Then:
  - $\Sigma \triangleright N \xmapsto{l \nleftrightarrow k} \Sigma' \triangleright N'$ implies $\Sigma \triangleright N|l[\![\mathsf{break}\ k]\!] \longrightarrow \Sigma' \triangleright N'$
  - $\Sigma \triangleright N|l[\![\mathsf{break}\ k]\!] \longrightarrow \Sigma' \triangleright N'$, where $\Sigma' \nvdash_{\mathsf{obs}} l \leftrightarrow k$ implies $\Sigma \triangleright N \xmapsto{l \nleftrightarrow k} \Sigma' \triangleright N''$ such that $N' \equiv N''$

Proof. The first clause for the action $\mathsf{kill}: l$ is proved by induction on the derivation $\Sigma \triangleright N \xrightarrow{\mathsf{kill}:l} \Sigma' \triangleright N'$. The second clause uses induction on the structure of $\Sigma \triangleright N$, with a subsidiary induction on the derivation of $\Sigma \triangleright N|l[\![\mathsf{kill}]\!] \longrightarrow \Sigma'' \triangleright N''$. The proof for the two clauses of the action $l \nleftrightarrow k$ is similar. □

In the next lemma we show that for any network Σ, the context can determine the exact state of the observable network Σ_O. We define the process $verStat_k^{\mathcal{I}}(x)$ that runs at a location k, which should be connected to all observable locations in a network Σ. It returns an output on the parameterised channel

x if and only if $\mathcal{I}(\Sigma) = \mathcal{I}$. Its implementation is based on the state observing construct $\mathsf{ping}\ l.P\lceil Q\rceil$; the sub-process, $verObs_k^{\mathcal{I}}(x)$, first checks that all *inaccessible* locations in $\mathcal{I}$, expressed as $\mathcal{I}' + l : \emptyset$ below, are indeed inaccessible and then checks that the *accessible* locations, expressed as $\mathcal{I} + l : \mathtt{L}$ where $\mathtt{L} \neq \emptyset$, satisfy the state declared in $\mathcal{I}$, using the sub-process $verLoc_k(x, y_1, y_2, z)$. This last subprocess, goes to the parameterised location x and checks that all its live connections and dead connections correspond to y_1 and y_2 respectively, returning a output on channel z if it is the case. The following lemma formalises the intuition that when run at an appropriate location, $verStat_k^{\mathcal{I}}(x)$ does satisfy the intended behaviour.

$$verStat_k^{\mathcal{I}}(x) \Leftarrow (\nu\, sync)\left(\begin{array}{l} verObs_k^{\mathcal{I}}(sync) \\ \mid verNObs(\mathbf{loc}(\mathcal{I}_{\mathcal{N}})/\mathbf{dom}(\mathcal{I}_{O}), sync) \\ \mid \underbrace{sync?().\ldots.sync?()}_{|\mathbf{loc}(\mathcal{N})|}.x!\langle\rangle \end{array}\right)$$

$$\begin{array}{rcl} verObs_k^{\langle\emptyset,\emptyset\rangle}(x) & \Leftarrow & \mathbf{0} \\ verObs_k^{\mathcal{I}+n:\emptyset}(x) & \Leftarrow & verObs_k^{\mathcal{I}}(x) \mid \mathsf{ping}\ l.\lceil y!\langle\rangle\rceil \\ verObs_k^{\mathcal{I}+l:\mathtt{L}}(x),\ \mathtt{L}\neq\emptyset & \Leftarrow & verObs_k^{\mathcal{I}}(x) \mid verLoc_k(l, \mathbf{dom}(\mathtt{L}), \mathbf{loc}(\mathcal{I}_{\mathcal{N}})/\mathbf{dom}(\mathtt{L}), x) \end{array}$$

$$verLoc_k(x, y_1, y_2, z) \Leftarrow (\nu\, sync)\mathsf{go}\ x.\left(\begin{array}{l} \prod_{l\in y_1} \mathsf{go}\ l.\mathsf{go}\ x.sync!\langle\rangle \\ \mid \prod_{l\in y_2} \mathsf{ping}\ l.\lceil sync!\langle\rangle\rceil \\ \mid \underbrace{sync?().\ldots.sync?()}_{|\mathbf{loc}(\mathcal{I}_{\mathcal{N}})|}.\mathsf{go}\ k.z!\langle\rangle \end{array}\right)$$

Lemma 4.3.3 (Observable Network). If for arbitrary network representation Σ:

$$\Sigma_+ = \Sigma + k_0 : \mathtt{loc}[\mathtt{a}, \mathbf{dom}(\Sigma_O)] + \textsc{succ} : \mathtt{ch}$$

Then,

$$\Sigma_+ \triangleright k_0[\![verStat_{k_0}^{\mathcal{I}}(\textsc{succ})]\!] \longrightarrow^* \Sigma_+ \triangleright k_0[\![\textsc{succ}!\langle\rangle]\!] \text{ iff } \mathcal{I} = \mathcal{I}(\Sigma)$$

Proof. We prove this lemma by contradiction. We analyse all the possible cases why $\mathcal{I} \neq \mathcal{I}(\Sigma)$ and then show that for each of these cases,

$$\Sigma_+ \triangleright k_0[\![verStat_{k_0}^{\mathcal{I}}(\textsc{succ})]\!] \not\longrightarrow^* \Sigma_+ \triangleright k_0[\![\textsc{succ}!\langle\rangle]\!] \qquad \square$$

We are now in a position to prove definability for every external action in DπF. We use $\mathbf{bn}(\mu)$ to denote the bound names (and their corresponding types) in the action μ; note this is empty for all actions apart from bound input and bound output. In order to complete the proof, we also require the following lemma.

Lemma 4.3.4. $\Sigma + n : \mathtt{T} \triangleright N \longrightarrow \Sigma' + n : \mathtt{T} \triangleright N'$ where $n \notin \mathbf{fn}(N)$ iff $\Sigma \vdash N$ and $\Sigma \triangleright N \longrightarrow \Sigma' \triangleright N'$

Proof. The proofs are by induction on the structure of N for $\Sigma \vdash N$ and by induction on the derivations of $\Sigma \triangleright N \longrightarrow \Sigma' \triangleright N'$ and $\Sigma + n : \mathtt{T} \triangleright N \longrightarrow \Sigma' + n : \mathtt{T} \triangleright N'$. □

Proposition 4.3.5 (Definability). Assume that for an arbitrary network representation Σ, the network Σ_+ denotes:

$$\Sigma_+ = \Sigma + k_0 : \mathtt{loc}[\mathtt{a}, \mathbf{dom}(\Sigma_O)], \textsc{succ} : \mathtt{ch}, \textsc{fail} : \mathtt{ch}$$

where k_0, $\textsc{succ}$ and $\textsc{fail}$ are fresh to Σ_N. Thus, for every external action μ and network representation Σ, every non-empty finite set of names Nm where $\Sigma_N \subseteq Nm$, every fresh pair of channel names $\textsc{succ}$, $\textsc{fail} \notin Nm$, and every fresh location name $k_0 \notin Nm$ connected to all observable locations in Σ_O, there exists a system $T^\mu(Nm, \textsc{succ}, \textsc{fail}, k_0)$ with the property that $\Sigma_+ \vdash_{\mathsf{obs}} T^\mu(Nm, \textsc{succ}, \textsc{fail}, k_0)$, such that:

1. $\Sigma \triangleright N \xrightarrow{\mu} \Sigma' + \mathbf{bn}(\mu) \triangleright N'$ implies
   $\Sigma_+ \triangleright N \,|\, T^\mu(Nm, \textsc{succ}, \textsc{fail}, k_0) \Longrightarrow \Sigma'_+ \triangleright (\nu\, \mathbf{bn}(\mu))\, N' \mid k_0[\![\textsc{succ}!\langle \mathbf{bn}(\mu)\rangle]\!]$

2. $\Sigma_+ \triangleright N \,|\, T^\mu(Nm, \textsc{succ}, \textsc{fail}, k_0) \Longrightarrow \Sigma'_+ \triangleright N'$,
   where $\Sigma'_+ \triangleright N' \Downarrow_{\textsc{succ}@k_0}$, $\Sigma'_+ \triangleright N' \not\Downarrow_{\textsc{fail}@k_0}$ implies that
   $N' \equiv (\nu\, \mathbf{bn}(\mu)) N'' | k_0[\![\textsc{succ}!\langle \mathbf{bn}(\mu)\rangle]\!]$ for some N''
   such that $\Sigma \triangleright N \stackrel{\mu}{\Longrightarrow} \Sigma' + \mathbf{bn}(\mu) \triangleright N''$.

Proof. We have to prove that the above two clauses are true for all of the four external actions. If μ is the bound input action $(\tilde{n} : \tilde{\mathtt{L}})l : a?(V)$, where $\tilde{\mathtt{L}} = \mathsf{lnk}(\tilde{n} : \tilde{\mathtt{T}}, \Sigma)$ for some $\tilde{\mathtt{T}}$, the required system is

$$(\nu\, \tilde{n} : \tilde{\mathtt{T}})(l[\![a!\langle V\rangle.\mathsf{go}\ k_0.\textsc{fail}?().\textsc{succ}!\langle\rangle]\!] \,|\, k_0[\![\textsc{fail}!\langle\rangle]\!])$$

For the output case where μ is $(\tilde{n} : \tilde{\mathtt{L}})l : a!\langle V\rangle$, the required $T^\mu(Nm, \textsc{succ}, \textsc{fail}, k_0)$

is

$$k_0[\![\textsc{fail}!\langle\rangle]\!] \mid$$

$$l\left[\!\!\left[a?(X).(\nu\, sync)\begin{pmatrix}\prod_{i=1}^{m} \mathsf{if}\ x_i \notin Nm.sync!\langle\rangle \mid \prod_{j=m+1}^{|X|} \mathsf{if}\ x_j = v_j.sync!\langle\rangle \\ \mid \underbrace{sync?()..sync?()}_{|X|}.\mathsf{go}\ k_0.(\nu c)\begin{pmatrix} verNwStat^{\mathcal{I}}_{k_0}(x_1..x_m, c) \\ \mid c?(x).\begin{pmatrix}\textsc{fail}?().\textsc{succ}!\langle x_1..x_m\rangle \\ \mid \mathsf{go}\ x..\mathsf{kill}\end{pmatrix}\end{pmatrix}\end{pmatrix}\right]\!\!\right]$$

such that

$$verNwStat^{\mathcal{I}}_{k_0}(x_1 \dots x_m, y) \Leftarrow (\nu\, k' : \mathtt{T}_{k'})\mathsf{go}\ k'.(\nu d)\begin{pmatrix} verStat^{\mathcal{I}+(x_1..x_m:\tilde{\mathtt{K}})}_{k'}(d) \\ \mid d?().\mathsf{go}\ k_0.y!\langle k'\rangle \end{pmatrix}$$

and $\mathtt{T}_{k'} = \mathtt{loc}[\mathtt{a}, Nm \cup \{x_1..x_m\}]$, $\tilde{\mathtt{K}} = \tilde{\mathtt{L}}\{^{x_1..x_m}/_{\tilde{n}}\}$

For the sake of presentation ,we assume that the first $v_1 \dots v_m$ in $V = v_1 \dots v_{|V|}$ in μ are bound, and the remaining $v_{m+1} \dots v_{|V|}$ are free; a more general test can be construct for arbitrary ordering of bound names in V using the same principles used for this test. We also use the conditional $\mathsf{if}\ x \notin Nm.P$ as an abbreviation for the obvious nested negative comparisons between x and each name in Nm.

The test works in two stages. Similar to the tests in [9, 8], the first stage performs the appropriate test for every input variable x_i, releasing $sync!\langle\rangle$ if the test is successful; if x_i is expected to be a bound name in μ, then we make sure it is fresh to Nm; otherwise x_i is matched with the corresponding free name. Another process waits for input on $|V|$ successful tests, that is $|V|$ inputs on the scoped channel *sync* and then releases the code for the second stage.

The second stage deals with the verification of any new live connections and locations that become reachable as a result of the fresh names inputted. To avoid complicated routing to reach these new locations, $verNwStat^{\mathcal{I}}_{k_0}(x_1 \dots x_m, y)$ creates a new location k' from the location k_0, with a location type that attempts to connect to any name in Nm together with the fresh bound names just inputted $x_1 \dots x_m$; recalling Example 3.2.1, we note that the purpose of this procedure is to short-circuit our way to the newly reachable locations. We afterwards run $verStat^{\mathcal{I}+(x_1..x_m:\tilde{\mathtt{K}})}_{k'}(c)$ from this new location k', to verify that the new observable network state is indeed $\mathcal{I} + \tilde{n} : \tilde{\mathtt{L}}$. If this is the case, we signal on the continuation channel d the fresh location k', which triggers a process that goes back to location k_0 and signals once again on another continuation channel, denoted by the variable y, but eventually parameterised by the scoped channel c in the testing context above. This triggers two parallel processes; the first one consumes the barb $\textsc{fail}$ and releases an output on $\textsc{succ}$ with the bound names $x_1 \dots x_m$, whereas the second one goes back to k' and kills it for cleaning up purposes.

In addition to bound input and bound output, we have two non-standard actions $\mathsf{kill} : l$ and $l \nleftrightarrow k$ and the test required for these actions are :

$$l[\![\mathsf{kill}]\!] \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\mathsf{ping}\ l.\mathsf{ping}\ l.\lceil \textsc{fail}?().\textsc{succ}!\langle\rangle\rceil]\!]$$

and

$$l[\![\mathsf{break}\ k]\!] \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid (\nu\, sync)\begin{pmatrix} l[\![\mathsf{ping}\ k.\mathsf{ping}\ k.\lceil \mathsf{go}\ k_0.sync!\langle\rangle\rceil]\!] \\ \mid k[\![\mathsf{ping}\ l.\mathsf{ping}\ l.\lceil \mathsf{go}\ k_0.sync!\langle\rangle\rceil]\!] \\ \mid k_0[\![sync?().sync?().\textsc{fail}?().\textsc{succ}!\langle\rangle]\!] \end{pmatrix}$$

respectively.

Since inducing faults is an asynchronous operation, the actual killing of a location or breaking of a link is independent of its observation. The observation of a kill at l is carried out from k_0 by two successive pings, first observing that l is alive and subsequently observing that l has become dead. The observation of a link break between l and k is slightly more complicated, because it needs to be tested from one of the connected locations l and k. The test is carried out, as in the previous case, through two successive pings; the first ping determines that k is accessible from l (or viceversa) while the second determine that it is not anymore. However, k (or viceversa l) can become inaccessible because it died and not because the link broke; to ensure that k (or l) became inaccessible because of a link failure, we perform the test from both endpoints, l and k, and synchronise at k_0.

We next give an outline of the proof for one of the non-standard actions, $\mathsf{kill} : l$; the proof of definability for $l \nleftrightarrow k$ is similar, whereas the proof for the remaining two actions can be extracted from [9, 8]. For the first clause, from $\Sigma \triangleright N \xmapsto{\mathsf{kill}:l} \Sigma' \triangleright N'$ we know that $\Sigma \vdash_{\mathsf{obs}} l : \mathbf{alive}$, thus $\Sigma_+ \vdash_{\mathsf{obs}} l : \mathbf{alive}$, which means we can perform the reduction involving a positive ping:

$$\begin{aligned} \Sigma_+ \triangleright N \mid l[\![\mathsf{kill}]\!] \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\mathsf{ping}\ l.\mathsf{ping}\ l.\lceil \textsc{fail}?().\textsc{succ}!\langle\rangle\rceil]\!] &\longrightarrow \\ \Sigma_+ \triangleright l[\![\mathsf{kill}]\!] \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\mathsf{ping}\ l.\lceil \textsc{fail}?().\textsc{succ}!\langle\rangle\rceil]\!] & \end{aligned} \tag{40}$$

From $\Sigma \triangleright N \xrightarrow{\mathsf{kill}:l} \Sigma' \triangleright N'$, $\mathcal{I}(\Sigma_+)$ allows $\mathsf{kill} : l$ and Proposition 4.1.6 we derive

$$\Sigma_+ \triangleright N \xrightarrow{\mathsf{kill}:l} \Sigma'_+ \triangleright N' \quad \text{where } \Sigma_+ \vdash_{\mathsf{obs}} l : \mathbf{alive} \text{ and } \Sigma'_+ \nvdash_{\mathsf{obs}} l : \mathbf{alive} \tag{41}$$

and from (41) and Lemma 4.3.2 we get

$$\Sigma_+ \triangleright N \mid l[\![\mathsf{kill}]\!] \longrightarrow \Sigma'_+ \triangleright N'$$

and (r-par-ctxt) we derive

$$\begin{aligned} \Sigma_+ \triangleright N \mid l[\![\mathsf{kill}]\!] \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\mathsf{ping}\ l.\lceil \textsc{fail}?().\textsc{succ}!\langle\rangle\rceil]\!] &\longrightarrow \\ \Sigma'_+ \triangleright N' \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\mathsf{ping}\ l.\lceil \textsc{fail}?().\textsc{succ}!\langle\rangle\rceil]\!] & \end{aligned} \tag{42}$$

Subsequently we derive the sequence of reductions

$$\begin{aligned}\Sigma'_+ \triangleright N' \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\mathsf{ping}\ l.\lceil\textsc{fail}?().\textsc{succ}!\langle\rangle\rceil]\!] &\longrightarrow \\ \Sigma'_+ \triangleright N' \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\textsc{fail}?().\textsc{succ}!\langle\rangle]\!] &\longrightarrow \\ \Sigma'_+ \triangleright N' \mid k_0[\![\textsc{succ}!\langle\rangle]\!]\end{aligned} \tag{43}$$

Combining the reductions in (40), (42) and (43) we prove the first clause.

For the second clause, the set of barbs $\Sigma'_+ \triangleright N' \Downarrow_{\textsc{succ}@k_0}$, $\Sigma'_+ \triangleright N' \not\Downarrow_{\textsc{fail}@k_0}$ can *only* be obtained through the sequence of reductions

$$\Sigma_+ \triangleright N \mid l[\![\mathsf{kill}]\!] \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\mathsf{ping}\ l.\mathsf{ping}\ l.\lceil\textsc{fail}?().\textsc{succ}!\langle\rangle\rceil]\!] \Longrightarrow \tag{44}$$

$$\Sigma^1_+ \triangleright N^1 \mid l[\![\mathsf{kill}]\!] \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\mathsf{ping}\ l.\mathsf{ping}\ l.\lceil\textsc{fail}?().\textsc{succ}!\langle\rangle\rceil]\!] \longrightarrow$$

$$\Sigma^1_+ \triangleright N^1 \mid l[\![\mathsf{kill}]\!] \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\mathsf{ping}\ l.\lceil\textsc{fail}?().\textsc{succ}!\langle\rangle\rceil]\!] \Longrightarrow \tag{45}$$

$$\Sigma^2_+ \triangleright N^2 \mid l[\![\mathsf{kill}]\!] \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\mathsf{ping}\ l.\lceil\textsc{fail}?().\textsc{succ}!\langle\rangle\rceil]\!] \longrightarrow \tag{46}$$

$$\Sigma^2_+ - l \triangleright N^2 \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\mathsf{ping}\ l.\lceil\textsc{fail}?().\textsc{succ}!\langle\rangle\rceil]\!] \Longrightarrow \tag{47}$$

$$\Sigma^3_+ - l \triangleright N^3 \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\mathsf{ping}\ l.\lceil\textsc{fail}?().\textsc{succ}!\langle\rangle\rceil]\!] \longrightarrow$$

$$\Sigma^3_+ - l \triangleright N^3 \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\textsc{fail}?().\textsc{succ}!\langle\rangle]\!] \Longrightarrow \tag{48}$$

$$\Sigma^4_+ - l \triangleright N^4 \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\textsc{fail}?().\textsc{succ}!\langle\rangle]\!] \longrightarrow$$

$$\Sigma^4_+ - l \triangleright N^4 \mid k_0[\![\textsc{succ}!\langle\rangle]\!] \Longrightarrow \tag{49}$$

$$\Sigma'_+ \triangleright N' \mid k_0[\![\textsc{succ}!\langle\rangle]\!]$$

From (46) and Lemma 4.3.2 we deduce

$$\begin{aligned}\Sigma^2_+ \triangleright N^2 \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\mathsf{ping}\ l.\lceil\textsc{fail}?().\textsc{succ}!\langle\rangle\rceil]\!] &\xmapsto{\mathsf{kill}:l} \\ \Sigma^2_+ - l \triangleright N^2 \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\mathsf{ping}\ l.\lceil\textsc{fail}?().\textsc{succ}!\langle\rangle\rceil]\!]\end{aligned}$$

and by the inductive hypothesis of (l-par-ctxt), the fact that $\mathcal{I}(\Sigma^2)$ allows $\mathsf{kill} : l$ and Proposition 4.1.6, we derive

$$\Sigma^2 \triangleright N^2 \xrightarrow{\mathsf{kill}:l} \Sigma^2 - l \triangleright N^2 \tag{50}$$

From (44), (45), (47), (48) and (49) and (r-par-ctxt) obtain

$$\begin{aligned}&\Sigma_+ \triangleright N \Longrightarrow \Sigma^1_+ \triangleright N^1 \Longrightarrow \Sigma^2_+ \triangleright N^2 \\ &\Sigma^2 - l_+ \triangleright N^2 \Longrightarrow \Sigma^3_+ \triangleright N^3 \Longrightarrow \Sigma^4_+ \triangleright N^4 \Longrightarrow \Sigma_+ \triangleright N'\end{aligned} \tag{51}$$

and from (51) and Lemma 4.3.4 we obtain

$$\begin{aligned}&\Sigma \triangleright N \Longrightarrow \Sigma^1 \triangleright N^1 \Longrightarrow \Sigma^2 \triangleright N^2 \\ &\Sigma^2 - l \triangleright N^2 \Longrightarrow \Sigma^3 \triangleright N^3 \Longrightarrow \Sigma^4 \triangleright N^4 \Longrightarrow \Sigma' \triangleright N'\end{aligned} \tag{52}$$

Finally, using Proposition 4.2.3 to convert the reductions in (52) into weak silent actions and merging these with (50) we obtain as required

$$\Sigma \triangleright N \overset{\mathsf{kill}:l}{\models\!\!\Longrightarrow} \equiv \Sigma' \triangleright N' \qquad \square$$

The result of Proposition 4.3.5 means that intuitively we can provoke the action $\Sigma \triangleright N \overset{\mu}{\Longrightarrow} \Sigma' \triangleright N'$ by extending Σ with a fresh location k_0 and fresh channels $\textsc{succ}$ and $\textsc{fail}$ and placing N in parallel with $T^{\mu}(Nm, \textsc{succ}, \textsc{fail}, k_0)$ for a suitably chosen Nm. But in the case of actions where $\mathbf{bn}(\mu) \neq \emptyset$ we do not get precisely the residual $\Pi' \triangleright N'$ but instead $\Sigma''_{+} \triangleright (\nu\, \mathbf{bn}(\mu))\, N \,|\, k_0[\![\textsc{succ}!\langle \mathbf{bn}(\mu)\rangle]\!]$where $\Sigma'' + \mathbf{bn}(\mu) = \Sigma'$. We therefore state and prove a variant the extrusion lemma in [9, 8], which enables us to recover the residual $\Sigma' \triangleright N'$ from $\Sigma''_{+} \triangleright (\nu\, \mathbf{bn}(\mu))N \,|\, k_0[\![\textsc{succ}!\langle \mathbf{bn}(\mu)\rangle]\!]$; this lemma uses the preliminary lemma below, which we chose to extract as an important step of the proof.

Lemma 4.3.6. Suppose δ, k_0 are fresh to the systems M, $k[\![P(X)]\!]$. Suppose also that $k \in \mathsf{C}$. Then:

$$\begin{aligned}\Sigma \models (\nu\, \tilde{n} : \tilde{\mathrm{T}})(M \,|\, k[\![P(\tilde{n})]\!]) \;\cong&\\ &(\nu\, \tilde{n} : \tilde{\mathrm{T}})(\nu\, \delta : \mathsf{ch})(\nu\, k_0 : \mathtt{loc}[\mathsf{a}, \mathsf{C}])(M \,|\, k_0[\![\delta!\langle \tilde{n}\rangle]\!] \,|\, k_0[\![\delta?(X).\mathsf{go}\; k.P(X)]\!])\end{aligned}$$

Proof. We note that the left hand system can be obtained from the right hand system in two reductions, communication on δ and migrating from k_0 to k, that cannot be interfered with by any context. It is easy to come up with a bisimulation proving that the two systems are reduction barbed congruent. □

Lemma 4.3.7 (Extrusion). Suppose $\textsc{succ}, \textsc{fail}, k_0$ are fresh to the network representations Σ^M, Σ^N, M and N. Then

$$\begin{aligned}\mathcal{I} \models \Sigma^M_{+} \triangleright (\nu\, \tilde{n} : \tilde{\mathrm{T}})M|k_0[\![\textsc{succ}!\langle \tilde{n}\rangle]\!] \cong \Sigma^N_{+} \triangleright (\nu\, \tilde{n} : \tilde{\mathrm{U}})N|k_0[\![\textsc{succ}!\langle \tilde{n}\rangle]\!]&\\ \text{implies } \Sigma^M + \tilde{n} : \tilde{\mathrm{T}} \triangleright M \cong \Sigma^N + \tilde{n} : \tilde{\mathrm{U}} \triangleright N&\end{aligned}$$

Proof. We define the relation $\mathcal{R}$ as:

$$\mathcal{R} = \left\{ \langle \Sigma^M + \tilde{n} : \tilde{\mathrm{T}} \triangleright M, \Sigma^N + \tilde{n} : \tilde{\mathrm{U}} \triangleright N \rangle \;\middle|\; \begin{aligned} \Sigma^M_{+} \triangleright (\nu\, \tilde{n} : \tilde{\mathrm{T}})M|k_0[\![\textsc{succ}!\langle \tilde{n}\rangle]\!] \cong&\\ \Sigma^N_{+} \triangleright (\nu\, \tilde{n} : \tilde{\mathrm{U}})N|k_0[\![\textsc{succ}!\langle \tilde{n}\rangle]\!]& \end{aligned} \right\}$$

and show that $\mathcal{R}$ satisfies the defining properties of $\cong$. It is obviously reduction closed. We here outline the proof for the barb preserving and contextuality properties.

Suppose $\Sigma^M + \tilde{n} : \tilde{\mathrm{T}} \triangleright M \;\mathcal{R}\; \Sigma^N + \tilde{n} : \tilde{\mathrm{U}} \triangleright N$ and $\Sigma^M + \tilde{n} : \tilde{\mathrm{T}} \triangleright M \Downarrow_{a@l}$; we have to show $\Sigma^N + \tilde{n} : \tilde{\mathrm{U}} \triangleright N \Downarrow_{a@l}$. If $l, a \notin \tilde{n}$ this is straightforward since in this case $\Sigma^M_{+} \triangleright (\nu\, \tilde{n} : \tilde{\mathrm{T}})M|k_0[\![\textsc{succ}!\langle \tilde{n}\rangle]\!] \Downarrow_{a@l}$, by barb preserving, $\Sigma^N_{+} \triangleright (\nu\, \tilde{n} : \tilde{\mathrm{U}})N|k_0[\![\textsc{succ}!\langle \tilde{n}\rangle]\!] \Downarrow_{a@l}$ which can only be because $\Sigma^N + \tilde{n} : \tilde{\mathrm{U}} \triangleright N \Downarrow_{a@l}$. So suppose, as an example, that $a \in \tilde{n}$. Even though we no longer have that $\Sigma^M_{+} \triangleright (\nu\, \tilde{n} : \tilde{\mathrm{T}})M|k_0[\![\textsc{succ}!\langle \tilde{n}\rangle]\!] \Downarrow_{a@l}$, the restricted name a can be extruded via $\textsc{succ}$ through the system:

$$T_a \Leftarrow k_0[\![\textsc{succ}?(X).\mathsf{mv}\; l(X_a?().\mathsf{go}\; k_0.\delta!\langle\rangle)]\!]$$

where δ is a fresh channel and X_a is the variable x_i where a is bound on input.

Since $\Sigma^M \triangleright M \Downarrow_{a@l}$ it follows that

$$\Sigma^M_+ + \delta:\mathsf{ch} \triangleright (\nu\,\tilde{n}:\tilde{\mathrm{T}})M|k_0[\![\textsc{succ}!\langle\tilde{n}\rangle]\!] \,|\, T_a \Downarrow_{\delta@k_0}$$

From the definition of $\cong$, we know

$$\Sigma^M_+ + \delta:\mathsf{ch} \triangleright (\nu\,\tilde{n}:\tilde{\mathrm{T}})M|k_0[\![\textsc{succ}!\langle\tilde{n}\rangle]\!] \,|\, T_a \cong \Sigma^N_+ + \delta:\mathsf{ch} \triangleright (\nu\,\tilde{n}:\tilde{\mathrm{U}})N|k_0[\![\textsc{succ}!\langle\tilde{n}\rangle]\!] \,|\, T_a$$

and by barb preservation we conclude

$$\Sigma^N_+ + \delta:\mathsf{ch} \triangleright (\nu\,\tilde{n}:\tilde{\mathrm{U}})N|k_0[\![\textsc{succ}!\langle\tilde{n}\rangle]\!] \,|\, T_a \Downarrow_{\delta@k_0}$$

which only be because $\Sigma^N \triangleright N \Downarrow_{a@l}$ as required.

The case for when $n = l$ is similar, only that instead of T_a we use the system:

$$T_l \Leftarrow k_0[\![\textsc{succ}?(X).(\nu\,k:(\mathbf{dom}(\mathcal{I}'_O)\cup X_l))\mathsf{go}\,k, X_l.a?().\mathsf{go}\,k, k_0.\delta!\langle\rangle]\!]$$

This system is similar to T_a with the exception that a specific location k is created so that we short-circuit our route to l, similar to the procedure we used earlier in the definability proof of bound outputs (see Proposition 4.3.5).

We still have to show that $\mathcal{R}$ is contextual. As an example we show that it is preserved by parallel system contexts and leave the simpler case, that for network extensions, to the interested reader. Suppose $\mathcal{I} \models \Sigma^M \triangleright M \;\mathcal{R}\; \Sigma^N \triangleright N$; we have to show that for arbitrary $k[\![P]\!]$ such that $\mathcal{I} \vdash k[\![P]\!]$ then we have $\mathcal{I} \models \Sigma^M \triangleright M \,|\, k[\![P]\!] \;\mathcal{R}\; \Sigma^N \triangleright N \,|\, k[\![P]\!]$.

By definition of $\mathcal{R}$, we have $\mathcal{I} \models \Sigma^M \triangleright M \;\mathcal{R}\; \Sigma^N \triangleright N$ because

$$\mathcal{I}' \models \Sigma^M_+ \triangleright (\nu\,\tilde{n}:\tilde{\mathrm{T}})M|k_0[\![\textsc{succ}!\langle\tilde{n}\rangle]\!] \;\cong\; \Sigma^N_+ \triangleright (\nu\,\tilde{n}:\tilde{\mathrm{U}})N|k_0[\![\textsc{succ}!\langle\tilde{n}\rangle]\!] \tag{53}$$

We define the system

$$T_{k[P]} \;\Leftarrow\; k_0[\![\textsc{succ}?(X).\mathsf{go}\,k'_0.\delta!\langle X\rangle \,|\, (X)\mathsf{go}\,k.P]\!]$$

where δ, k'_0 are fresh names and $(X)\mathsf{go}\,k.P$ substitutes all occurrences of $\tilde{n}$ in $\mathsf{go}\,k.P$ by the appropriate variables $x_i \in X$. From $\mathcal{I} \vdash k[\![P]\!]$ we deduce that $\mathcal{I}'' \vdash T_{k[P]}$ for $\mathcal{I}'' = \mathcal{I}' + \delta : \mathsf{ch} + k'_0 : \mathtt{loc}[\mathsf{a}, \mathbf{dom}(\mathcal{I}'_O)]$ and subsequently, by contextuality of $\cong$ and (53), we obtain

$$\mathcal{I}'' \models \Sigma^M_{++} \triangleright M' \,|\, T_{k[P]} \;\cong\; \Sigma^N_{++} \triangleright N' \,|\, T_{k[P]} \tag{54}$$

where

$$\begin{aligned} M' &= (\nu\,\tilde{n}:\tilde{\mathrm{T}})M \,|\, k_0[\![\textsc{succ}!\langle\tilde{n}\rangle]\!] \\ N' &= (\nu\,\tilde{n}:\tilde{\mathrm{U}})N \,|\, k_0[\![\textsc{succ}!\langle\tilde{n}\rangle]\!] \\ \Sigma^M_{++} &= \Sigma^M_+ + \delta:\mathsf{ch} + k'_0:\mathtt{loc}[\mathsf{a}, \mathbf{dom}(\mathcal{I}'_O)] \\ \Sigma^N_{++} &= \Sigma^N_+ + \delta:\mathsf{ch} + k'_0:\mathtt{loc}[\mathsf{a}, \mathbf{dom}(\mathcal{I}'_O)] \end{aligned}$$

From (54) and Proposition 4.3.1 we deduce that we can scope $\textsc{succ}$ and k_0 to obtain

$$\mathcal{I}' \models \Sigma^M_+ \triangleright (\nu\,\textsc{succ}, k_0)M' \,|\, T_{k[P]} \;\cong\; \Sigma^N_+ \triangleright (\nu\,\textsc{succ}, k_0)N' \,|\, T_{k[P]} \tag{55}$$

and by Lemma 4.3.6 and we get

$$\mathcal{I}' \models \Sigma^M_+ \triangleright (\nu\,\tilde{n}:\tilde{\mathrm{T}})M \,|\, k[\![P]\!] \,|\, k_0'[\![\delta!\langle\tilde{n}\rangle]\!] \;\cong\; \Sigma^N_+ \triangleright (\nu\,\tilde{n}:\tilde{\mathrm{T}})N \,|\, k[\![P]\!] \,|\, k_0'[\![\delta!\langle\tilde{n}\rangle]\!] \quad (56)$$

from which, by definition of $\mathcal{R}$, we derive $\mathcal{I} \models \Sigma^M \triangleright M|k[\![P]\!] \;\mathcal{R}\; \Sigma^N \triangleright N|k[\![P]\!]$ as required. □

Proposition 4.3.8 (Completness).

$$\mathcal{I} \models \Sigma^1 \triangleright M_1 \cong \Sigma^2 \triangleright M_2 \;\text{ implies }\; \mathcal{I} \models \Sigma^1 \triangleright M_1 \approx \Sigma^2 \triangleright M_2$$

Proof. Suppose $\Sigma^1 \triangleright M_1 \overset{\mu}{\longmapsto} \Sigma^1_1 \triangleright M_1'$; we must find a move $\Sigma^2 \triangleright M_2 \overset{\widehat{\mu}}{\Longmapsto} \Sigma^2_1 \triangleright M_2'$ such that $\Sigma^1_1 \triangleright M_1' \cong \Sigma^2_1 \triangleright M_2'$. If μ is an internal move then the matching move is obtained from the fact that $\cong$ is reduction closed, together with Proposition 4.2.3. If μ is an external action, then by choosing *Nm* so that it contains all the free names in $\mathcal{I}_N$ and choosing fresh $\textsc{succ}$, $\textsc{fail}$, k_0, from the first part of Proposition 4.3.5 and the assumption $\Sigma^1 \triangleright M_1 \overset{\mu}{\longmapsto} \Sigma^1_1 + \mathbf{bn}(\mu) \triangleright M_1'$ we obtain

$$\Sigma^1_+ \triangleright M_1|T^\mu(Nm, \textsc{succ}, \textsc{fail}, k_0) \Longrightarrow \Sigma^1_{1+} \triangleright (\nu\,\mathbf{bn}(\mu))M_1' \,|\, k_0[\![\textsc{succ}!\langle\mathbf{bn}(\mu)\rangle]\!]$$

By contextuality and reduction closure of $\cong$, we know that there is a matching move

$$\Sigma^2_+ \triangleright M_2|T^\mu(Nm, \textsc{succ}, \textsc{fail}, k_0) \Longrightarrow \Sigma \triangleright N$$

for some $\Sigma \triangleright N$ such that $\Sigma^1_{1+} \triangleright (\nu\,\mathbf{bn}(\mu))M_1' \,|\, k_0[\![\textsc{succ}!\langle\mathbf{bn}(\mu)\rangle]\!] \cong \Sigma \triangleright N$. This in turn means that $\Sigma \triangleright N \Downarrow_{\textsc{succ}@k_0}$ and $\Sigma \triangleright N \not\Downarrow_{\textsc{fail}@k_0}$ and so the second part of Proposition 4.3.5 now gives that $\Sigma \triangleright N \equiv \Sigma^2_{1+} \triangleright (\nu\,\mathbf{bn}(\mu))M_2' \,|\, k_0[\![\textsc{succ}!\langle\mathbf{bn}(\mu)\rangle]\!]$ for some Σ^2_{1+}, M_2' such that $\Sigma^2 \triangleright M_2 \overset{\mu}{\Longmapsto} \Sigma^2_1 + \mathbf{bn}(\mu) \triangleright M_2'$. This is the required matching move, since the Extrusion Lemma 4.3.7, gives us the required

$$\Sigma^1_1 + \mathbf{bn}(\mu) \triangleright M_1' \cong \Sigma^2_1 + \mathbf{bn}(\mu) \triangleright M_2'$$ □

5 Conclusions

We have presented a simple extension of Dπ, in which there is an explicit representation of the underlying network on which processes execute, exhibiting both node and link failures. Our main result is a *fully-abstract* bisimulation equivalence with which we can reason about the behaviour of systems in the presence of dynamic network failures. To the best of our knowledge, this is the first time system behaviour in the presence of *link* failure has ever been investigated.

Our starting point was the work by Hennessy and Riely [16] on bisimulation techniques for a distributed variant of CCS with location failure. We adapted this work to Dπ and defined a reduction semantics to describe the behaviour of systems in the presence of node and link failures; a core aspect of this adaptation is the encoding of node status and connections as type information. We then applied techniques for actions dependent on the observer's knowledge, developed for the π-calculus in [9] and Dπ in [8], to characterise a natural notion of barbed congruence.

It is often argued that a theory of node failure only would be sufficient to study link failure; intuitively link failure could be simulated by introducing a new intermediate node for each link, and movement via a link could be modelled in two steps, to and from the intermediate node. Such an approach, however, goes against our wish, expressed earlier in the Introduction. We would prefer to develop a theory in terms of the calculus itself, rather than indirectly via a translation to a lower-level calculus, since the resulting bisimulations, encoded in terms of these intermediate nodes, would be much more complex to express and cumbersome to work with. Moreover, we think that it is unlikely that the resulting theory would be fully-abstract due to the fact that, in the encoding proposed above, we would be decomposing atomic actions such as location migration into two or more sub-actions; it is well-known that, even for a simple calculus such as CCS, bisimulation equivalence is not preserved by translating actions *a* into sub-actions *begin-a* followed by *end-a*; see [7].

Rather than being a body of work that could be directly applied to real case scenarios, we believe our work is best viewed as a succinct well-founded framework from which numerous variations could be considered. For example links between sites could be uni-directional, rather than symmetric, or $\mathsf{ping}\ l.P\lceil Q\rceil$ could test for a *path* from the current site to l, rather than a direct connection. One could also limit the use of the fault inducing actions $\mathsf{kill} : l$ and $l \nleftrightarrow k$; for instance, disallowing them in the definition of the contextual equivalences would give a behavioural theory between systems running on *static* but possibly defective networks; allowing only n occurrences would lead the way to a theory of *fault-tolerance*. More generally, one could allow the *recovery* of failures, in which dead nodes, or broken links may randomly be restored. Adapting our lts and the resulting bisimulation equivalence to such scenarios are in some cases straightforward, and in others, serious undertakings; a typical example of the former is the introduction of uni-directional links, while failure recovery would probably fall into the latter.

Having said this, we feel that our framework, as it currently stands, lends itself well for analysing routing problems and studying the interplay between faults at the network level and their observation at the system level. In our more immediate research, we should be able to demonstrate the applicability of our bisimulations to establish the correctness of systems in the presence of failures; we intend to use our approach to develop a theory of *fault-tolerance* and to apply it to example systems from the literature.

Related work: There have been a number of studies on process behaviour in the presence of *permanent node failure* only, in addition to our starting point [16]. That closest to our work is the already cited [2, 1]; as already mentioned, their approach to developing reasoning tools is also quite different from ours. Rather than develop, justify and use bisimulations in the source language of interest, in their case π_l and π_{1l}, they propose a translation into a version of the π-calculus without locations, and use reasoning tools on the translations. But most importantly, they do show that for certain π_{1l} terms, it is sufficient to reason on their translations. Elsewhere, permanent location failure with hierarchical dependencies have been studied by Fournet, Gonthier, Levy and Remy in [6]. Berger [3] was the first to study a π-calculus extension that models *transient* location failure with persistent code and communication failures, while Nestmann, Merro and Fuzzatti [15] employ a tailor made process calculus to express standard results in distributed systems, such as [5].

A Notation

Here we give the formal definitions for the various notation we have introduced for extracting information from network representations, and for updating them.

A.1 DπLoc Notation

Recall that for DπLoc a network repsresentation Π consists of the tuple $\langle \mathcal{N}, \mathcal{D} \rangle$, where $\mathcal{N}$ is a set of names known and $\mathcal{D}$ is the set of dead locations. We thus define the following judgements:

$$
\begin{array}{llll}
\Pi \vdash a\!:\!\mathtt{ch} & \stackrel{\text{def}}{=} & a \in \Pi_{\mathcal{N}} & \textit{(valid channels)} \\
\Pi \vdash l\!:\!\mathtt{loc[d]} & \stackrel{\text{def}}{=} & l \in \Pi_{\mathcal{N}} \ \wedge \ l \in \Pi_{\mathcal{D}} & \textit{(valid dead location)} \\
\Pi \vdash l\!:\!\mathtt{loc[a]} & \stackrel{\text{def}}{=} & l \in \Pi_{\mathcal{N}} \ \wedge \ l \notin \Pi_{\mathcal{D}} & \textit{(valid live location)} \\
\Pi \vdash l : \mathbf{alive} & \stackrel{\text{def}}{=} & \Pi \vdash l\!:\!\mathtt{loc[a]} & \textit{(live locations)} \\
\Pi \vdash k \leftarrow l & \stackrel{\text{def}}{=} & \Pi \vdash k\!:\!\mathbf{alive},\ l\!:\!\mathbf{alive} & \textit{(k accessible from l)} \\
\Pi \vdash M & \stackrel{\text{def}}{=} & \mathbf{fn}(M) \in \Pi_{\mathcal{N}} & \textit{(valid systems)}
\end{array}
$$

We also define the following operations:

$$
\begin{array}{llll}
\Pi + a\!:\!\mathtt{ch} & \stackrel{\text{def}}{=} & \langle \Pi_{\mathcal{N}} \cup \{a\}, \Pi_{\mathcal{D}} \rangle & \textit{(adding fresh channel)} \\
\Pi + l\!:\!\mathtt{loc[a]} & \stackrel{\text{def}}{=} & \langle \Pi_{\mathcal{N}} \cup \{l\}, \Pi_{\mathcal{D}} \rangle & \textit{(adding fresh live location)} \\
\Pi + l\!:\!\mathtt{loc[d]} & \stackrel{\text{def}}{=} & \langle \Pi_{\mathcal{N}} \cup \{l\}, \Pi_{\mathcal{D}} \cup \{l\} \rangle & \textit{(adding fresh dead location)}
\end{array}
$$

$$
\Pi - l \stackrel{\text{def}}{=} \begin{cases} \langle \Pi_{\mathcal{N}}, \Pi_{\mathcal{D}} \cup \{l\} \rangle & \text{if } l \in \Pi_{\mathcal{N}} \\ \Pi & \text{otherwise} \end{cases} \qquad \textit{(killing a location)}
$$

A.2 DπF Notation

Network representations in DπF are based on the notion of linksets $\mathcal{L}$. We define the following operations and judgements, using a set of locations C:

$$
\begin{array}{rcll}
\mathcal{L}/C & \stackrel{\text{def}}{=} & \{\langle k_1, k_2 \rangle \mid \langle k_1, k_2 \rangle \in \mathcal{L} \text{ and neither } k_1, k_2 \notin C\} & (\textit{filtering}) \\
\mathcal{L} \vdash k \leftarrow l & \stackrel{\text{def}}{=} & \langle l, k \rangle \in \mathcal{L} & (\textit{accessibility}) \\
\mathcal{L} \vdash k \leftsquigarrow l & \stackrel{\text{def}}{=} & \mathcal{L} \vdash k \leftarrow l \text{ or } \exists k'.\, \mathcal{L} \vdash k' \leftarrow l \text{ and } \mathcal{L} \vdash k \leftsquigarrow k' & (\textit{reachability}) \\
l \leftrightarrow C & \stackrel{\text{def}}{=} & \{l \leftrightarrow k \mid k \in C\} & (\textit{component creation}) \\
\mathcal{L} \leftsquigarrow l & \stackrel{\text{def}}{=} & \{k \leftrightarrow k' \mid k \leftrightarrow k' \in \mathcal{L} \text{ and } \mathcal{L} \vdash k \leftsquigarrow l\} & (\textit{component reference})
\end{array}
$$

For DπF we have two kinds of network representations, ranged over by Δ and Σ. We define the following operations on them:

$$\begin{array}{rcll}
\Delta - l & \stackrel{\text{def}}{=} & \langle \Delta_{\mathcal{N}},\ \Delta_{\mathcal{D}} \cup \{l\},\ \Delta_{\mathcal{L}} \rangle & (\textit{location killing}) \\
\Sigma - l & \stackrel{\text{def}}{=} & \langle \Sigma_{\mathcal{N}},\ \Sigma_{\mathcal{O}}/\{l\},\ \Sigma_{\mathcal{L}}/\{l\} \rangle & (\textit{location killing}) \\
\\
\Delta - l \leftrightarrow k & \stackrel{\text{def}}{=} & \langle \Delta_{\mathcal{N}},\ \Delta_{\mathcal{D}},\ \Delta_{\mathcal{L}}/\{\langle l,k\rangle, \langle k,l\rangle\} \rangle & (\textit{link breaking}) \\
\Sigma - l \leftrightarrow k & \stackrel{\text{def}}{=} & \langle \Sigma_{\mathcal{N}},\ \Sigma_{\mathcal{O}}/\{\langle l,k\rangle, \langle k,l\rangle\},\ \Sigma_{\mathcal{L}}/\{\langle l,k\rangle, \langle k,l\rangle\} \rangle & (\textit{link breaking}) \\
\\
\Delta + a{:}\mathtt{ch} & \stackrel{\text{def}}{=} & \langle \Delta_{\mathcal{N}} \cup \{a\},\ \Delta_{\mathcal{D}},\ \Sigma_{\mathcal{L}} \rangle & (\textit{adding a channel}) \\
\Sigma + a{:}\mathtt{ch} & \stackrel{\text{def}}{=} & \langle \Sigma_{\mathcal{N}} \cup \{a\},\ \Sigma_{\mathcal{O}},\ \Sigma_{\mathcal{H}} \rangle & (\textit{adding a channel}) \\
\\
\Delta + l{:}\mathtt{loc}[\mathsf{d},\mathsf{C}] & \stackrel{\text{def}}{=} & \langle \Delta_{\mathcal{N}} \cup \{l\},\ \Delta_{\mathcal{D}} \cup \{l\},\ \Sigma_{\mathcal{L}} \cup l \leftrightarrow \mathsf{C} \rangle & (\textit{adding a location}) \\
\Delta + l{:}\mathtt{loc}[\mathsf{a},\mathsf{C}] & \stackrel{\text{def}}{=} & \langle \Delta_{\mathcal{N}} \cup \{l\},\ \Delta_{\mathcal{D}},\ \Sigma_{\mathcal{L}} \cup l \leftrightarrow \mathsf{C} \rangle & \\
\Sigma + l{:}\mathtt{loc}[\mathsf{d},\mathsf{C}] & \stackrel{\text{def}}{=} & \langle \Sigma_{\mathcal{N}} \cup \{l\},\ \Sigma_{\mathcal{O}},\ \Sigma_{\mathcal{H}} \rangle & (\textit{adding a location}) \\
\Sigma + l{:}\mathtt{loc}[\mathsf{a},\mathsf{C}] & \stackrel{\text{def}}{=} & &
\end{array}$$

$$\begin{array}{lll}
\text{Case} & \mathsf{C} \cap \mathbf{dom}(\Sigma_{\mathcal{O}}) = \emptyset & \text{then } \langle \Sigma_{\mathcal{N}} \cup \{n\},\ \Sigma_{\mathcal{O}},\ \mathcal{H}' \rangle \\
& & \quad \text{where: } \mathcal{H}' = \Sigma_{\mathcal{H}} \cup (l \leftrightarrow \mathsf{C}) \\
& \mathsf{C} \cap \mathbf{dom}(\Sigma_{\mathcal{O}}) \neq \emptyset & \text{then } \langle \Sigma_{\mathcal{N}} \cup \{n\},\ \mathcal{O}',\ \mathcal{H}' \rangle \\
& & \quad \text{where: } \mathcal{O}' = \Sigma_{\mathcal{O}} \cup (l \leftrightarrow \mathsf{C}) \cup (\Sigma_{\mathcal{H}} \leftrightsquigarrow \mathsf{C}) \\
& & \qquad \text{and } \mathcal{H}' = \Sigma_{\mathcal{H}}/(\Sigma_{\mathcal{H}} \leftrightsquigarrow \mathsf{C})
\end{array}$$

We next define translations from one network representation to the other, together with the definition of the observer network knowledge for every representation.

$$\begin{array}{rcll}
\Sigma(\Delta) & \stackrel{\text{def}}{=} & \langle \Delta_{\mathcal{N}},\ \Delta_{\mathcal{L}}/\Delta_{\mathcal{D}},\ \emptyset \rangle & (\textit{from } \Delta \textit{ to } \Sigma) \\
\Delta(\Sigma) & \stackrel{\text{def}}{=} & \langle \Sigma_{\mathcal{N}},\ (\mathbf{loc}(\Sigma_{\mathcal{N}})/\mathbf{dom}(\Sigma_{\mathcal{O}} \cup \Sigma_{\mathcal{H}})),\ \Sigma_{\mathcal{O}} \cup \Sigma_{\mathcal{H}} \rangle & (\textit{from } \Sigma \textit{ to } \Delta) \\
\mathcal{I}(\Sigma) & \stackrel{\text{def}}{=} & \langle \Sigma_{\mathcal{N}},\ \Sigma_{\mathcal{O}} \rangle & (\textit{observer knowledge}) \\
\mathcal{I}(\Delta) & \stackrel{\text{def}}{=} & \mathcal{I}(\Sigma(\Delta)) &
\end{array}$$

Finally, we define judgements made using the various network representations. Ideally we would like that distinct network representations that have the same semantic interpretations yield the same judgements as shown below.

$$
\begin{array}{rcll}
\Sigma \vdash l : \mathbf{alive} & \stackrel{\text{def}}{=} & l \in \mathbf{dom}(\Sigma_{\mathcal{O}} \cup \Sigma_{\mathcal{H}}) & (\textit{live locations}) \\
\Sigma \vdash l \leftrightarrow k & \stackrel{\text{def}}{=} & l \leftrightarrow k \in \Sigma_{\mathcal{O}} \cup \Sigma_{\mathcal{H}} & (\textit{live link}) \\
\Sigma \vdash \mathtt{T} & \stackrel{\text{def}}{=} & \mathbf{fn}(\mathtt{T}) \subseteq \Sigma_{\mathcal{N}} & (\textit{valid types}) \\
\Sigma \vdash n : \mathtt{T}, \tilde{n} : \tilde{\mathtt{T}} & \stackrel{\text{def}}{=} & \Sigma \vdash \mathtt{T} \text{ and } \Sigma + n : \mathtt{T} \vdash \tilde{n} : \tilde{\mathtt{T}} & \\
\Sigma \vdash N & \stackrel{\text{def}}{=} & \mathbf{fn}(N) \subseteq \Sigma_{\mathcal{N}} & (\textit{valid systems}) \\
\Sigma \vdash k \leftarrow l & \stackrel{\text{def}}{=} & \Sigma_{\mathcal{O}} \vdash k \leftarrow l \text{ or } \Sigma_{\mathcal{O}} \vdash k \leftarrow l & (\textit{accessibility}) \\
\Sigma \vdash k \leftsquigarrow l & \stackrel{\text{def}}{=} & \Sigma_{\mathcal{O}} \vdash k \leftsquigarrow l \text{ or } \Sigma_{\mathcal{O}} \vdash k \leftsquigarrow l & (\textit{reachability})
\end{array}
$$

$$
\Delta \vdash l : \mathbf{alive},\ l \leftrightarrow k,\ \mathtt{T},\ N \ \stackrel{\text{def}}{=}\ \Sigma(\Delta) \vdash l : \mathbf{alive},\ l \leftrightarrow k,\ \mathtt{T},\ N
$$

$$
\begin{array}{rcll}
\mathcal{I} + n : \mathtt{L} & \stackrel{\text{def}}{=} & \langle \mathcal{I}_{\mathcal{N}} \cup \{n\},\ \mathcal{I}_{\mathcal{O}} \cup \mathtt{L} \rangle & (\textit{updates}) \\
\mathcal{I} \vdash l : \mathbf{alive} & \stackrel{\text{def}}{=} & l \in \mathbf{dom}(\mathcal{I}_{\mathcal{O}}) & (\textit{live locations}) \\
\mathcal{I} \vdash l \leftrightarrow k & \stackrel{\text{def}}{=} & l \leftrightarrow k \in \mathcal{I}_{\mathcal{O}} & (\textit{live link}) \\
\mathcal{I} \vdash \mathtt{T} & \stackrel{\text{def}}{=} & \mathbf{fn}(\mathtt{T}) \subseteq \mathbf{dom}(\mathcal{I}_{\mathcal{O}}) & (\textit{valid types}) \\
\mathcal{I} \vdash l[\![P]\!] & \stackrel{\text{def}}{=} & \mathbf{fn}(P) \subseteq \mathcal{I}_{\mathcal{N}} \text{ and } l \in \mathbf{dom}(\mathcal{I}_{\mathcal{O}}) & (\textit{valid systems}) \\
\mathcal{I} \vdash (\nu\, n : \mathtt{T})N & \stackrel{\text{def}}{=} & \mathcal{I} \vdash \mathtt{T} \text{ and } \mathcal{I} + n : \mathtt{T} \vdash N & \\
\mathcal{I} \vdash N|M & \stackrel{\text{def}}{=} & \mathcal{I} \vdash N \text{ and } \mathcal{I} \vdash M &
\end{array}
$$

$$
\begin{array}{rcll}
\Delta \vdash_{\mathsf{obs}} l : \mathbf{alive},\ l \leftrightarrow k,\ \mathtt{T},\ N & \stackrel{\text{def}}{=} & \mathcal{I}(\Delta) \vdash l : \mathbf{alive},\ l \leftrightarrow k,\ \mathtt{T},\ N & (\textit{external judgments}) \\
\Sigma \vdash_{\mathsf{obs}} l : \mathbf{alive},\ l \leftrightarrow k,\ \mathtt{T},\ N & \stackrel{\text{def}}{=} & \mathcal{I}(\Sigma) \vdash l : \mathbf{alive},\ l \leftrightarrow k,\ \mathtt{T},\ N &
\end{array}
$$

Finally we outline a number of operations on types used in reduction rules and transition rules.

$$
\begin{array}{rcll}
\mathsf{ch}/\{l_1, .., l_n\} & \stackrel{\text{def}}{=} & \mathsf{ch} & (\textit{type filtering}) \\
\mathtt{loc}[\mathtt{C}]/\{l_1, .., l_n\} & \stackrel{\text{def}}{=} & \mathtt{loc}[\mathtt{C}/\{l_1, .., l_n\}] &
\end{array}
$$

$$
\begin{array}{rcll}
\mathsf{inst}(\mathtt{loc}[\mathtt{C}], l, \Delta) & \stackrel{\text{def}}{=} & \mathtt{loc}[\{k \mid k \in \mathtt{C} \text{ and } \Delta \vdash k \leftsquigarrow l\}] & (\textit{instantiate}) \\
\mathsf{inst}(\mathtt{loc}[\mathtt{C}], l, \Sigma) & \stackrel{\text{def}}{=} & \mathtt{loc}[\{k \mid k \in \mathtt{C} \text{ and } \Sigma \vdash k \leftsquigarrow l\}] &
\end{array}
$$

$$
\mathsf{lnk}(n : \mathtt{T}, \Sigma) \ \stackrel{\text{def}}{=}\
\begin{array}{l}
(n \leftrightarrow \mathtt{C}) \cup (\Sigma_{\mathcal{H}} \leftsquigarrow \mathtt{C}) \\
\quad \text{if } \mathtt{T} = \mathtt{loc}[\mathtt{a}, \mathtt{C}] \text{ and } \mathtt{C} \cap \mathbf{loc}(\Sigma_{\mathcal{O}}) \neq \emptyset \\
\emptyset \quad \text{otherwise}
\end{array}
\quad (\textit{link types})
$$

B Auxilliary Proofs

We here prove a lemma that is used to show that our lts of § 3.3 is closed over valid effective configurations.

Lemma B.0.1 (Valid Effective Network Updates). If Σ is a valid effective network, n is fresh in Σ and the type $\mathtt{T}$ is a valid type with respect to Σ, denoted as $\Sigma \vdash \mathtt{T}$ (see Appendix for definition) then $\Sigma + n : \mathtt{T}$ is a valid effective network.

Proof. The cases where $\mathtt{T} = \mathtt{ch}$ and $\mathtt{T} = \mathtt{loc}[\mathtt{d}, \mathtt{C}]$ are trivial so we focus our attention to the case where $\mathtt{T} = \mathtt{loc}[\mathtt{d}, \mathtt{C}]$; at this point, according to Definition 3.3.5, we have two possible subcases:

- If $\mathtt{C} \cap \mathbf{dom}(\Sigma_{\mathcal{O}}) = \emptyset$ then $\Sigma + n : \mathtt{loc}[\mathtt{a}, \mathtt{C}]$ has the form $\langle \Sigma_{\mathcal{N}} \cup \{n\},\ \Sigma_{\mathcal{O}},\ \mathcal{H}' \rangle$ where $\mathcal{H}' = \Sigma_{\mathcal{H}} \cup (l \leftrightarrow \mathtt{C})$. To prove that this resultant network is a valid effective network, we have to show that it adheres to the three consistency requirements, defined earlier in Definition 3.3.2:

  1. $\mathbf{dom}(\Sigma_{\mathcal{O}}) \subseteq \mathbf{loc}(\Sigma_{\mathcal{N}} \cup \{n\})$. This is immediate from the fact that Σ is valid and thus $\mathbf{dom}(\Sigma_{\mathcal{O}}) \subseteq \mathbf{loc}(\Sigma_{\mathcal{N}})$.

  2. $\mathbf{dom}(\mathcal{H}') \subseteq \mathbf{loc}(\Sigma_{\mathcal{N}} \cup \{n\})$ and that $\mathcal{H}'$ is a linkset. The inclusion is obtained from the fact that $\mathbf{dom}(\Sigma_{\mathcal{H}}) \subseteq \mathbf{loc}(\Sigma_{\mathcal{N}})$ and the assumption that $\mathbf{loc}(\mathtt{loc}[\mathtt{a}, \mathtt{C}]) \subseteq \mathbf{loc}(\Sigma_{\mathcal{N}})$. The fact that $\mathcal{H}' = \Sigma_{\mathcal{H}} \cup l \leftrightarrow \mathtt{C}$ is a linkset is immediate from the fact that $l \leftrightarrow \mathtt{C}$ is a component.

  3. $\mathbf{dom}(\Sigma_{\mathcal{O}}) \cap \mathbf{dom}(\mathcal{H}') = \emptyset$. This is immediately obtained from the assumptions that $\mathbf{dom}(\Sigma_{\mathcal{O}}) \cap \mathbf{dom}(\Sigma_{\mathcal{H}}) = \emptyset$, $n \notin \Sigma_{\mathcal{N}}$ and the condition for this subcase, that is $\mathtt{C} \cap \mathbf{dom}(\Sigma_{\mathcal{O}}) = \emptyset$.

- If $(\mathtt{C} \cap \mathbf{dom}(\Sigma_{\mathcal{O}}) \neq \emptyset)$ then $\Sigma + n : \mathtt{loc}[\mathtt{a}, \mathtt{C}]$ has the form $\langle \Sigma_{\mathcal{N}} \cup \{n\},\ \mathcal{O}',\ \mathcal{H}' \rangle$ where $\mathcal{O}' = \Sigma_{\mathcal{O}} \cup (l \leftrightarrow \mathtt{C} \cup (\Sigma_{\mathcal{H}} \leftrightsquigarrow \mathtt{C})$ and $\mathcal{H}' = \Sigma_{\mathcal{H}} / (\Sigma_{\mathcal{H}} \leftrightsquigarrow \mathtt{C})$. One again, we have to prove that $\Sigma + n : \mathtt{loc}[\mathtt{a}, \mathtt{C}]$ satisfies the three consistency conditions:

  1. $\mathbf{dom}(\mathcal{O}') \subseteq \mathbf{loc}(\Sigma_{\mathcal{N}} \cup \{n\})$ and that $\mathcal{O}'$ is a linkset. The proof here progresses similar to the second requirement of the previous subcase.

  2. $\mathbf{dom}(\mathcal{H}') \subseteq \mathbf{loc}(\Sigma_{\mathcal{N}} \cup \{n\})$ and that $\mathcal{H}'$ is a linkset. The proof for the inclusion is a simpler version of the above subcases, while the requirement that $\mathcal{H}' = \Sigma_{\mathcal{H}} / \Sigma_{\mathcal{H}} \leftrightsquigarrow \mathtt{C}$ is a linkset is obtained from the fact that $\Sigma_{\mathcal{H}} \leftrightsquigarrow \mathtt{C}$ is a component and Lemma 3.3.4.

  3. $\mathbf{dom}(\mathcal{O}') \cap \mathbf{dom}(\mathcal{H}') = \emptyset$. This is obtained from the assumptions that $\mathbf{dom}(\Sigma_{\mathcal{O}}) \cap \mathbf{dom}(\Sigma_{\mathcal{H}}) = \emptyset$, $n \notin \Sigma_{\mathcal{N}}$ and the structure of $\mathcal{O}'$ and $\mathcal{H}'$. $\square$

References

[1] Roberto M. Amadio. An asynchronous model of locality, failure, and process mobility. In D. Garlan and D. Le Métayer, editors, *Proceedings of the 2nd International Conference on Coordination Languages and Models (COORDINATION'97)*, volume 1282, pages 374–391, Berlin, Germany, 1997. Springer-Verlag.

[2] Roberto M. Amadio and Sanjiva Prasad. Localities and failures. *FSTTCS: Foundations of Software Technology and Theoretical Computer Science*, 14, 1994.

[3] Martin Berger. Basic theory of reduction congruence for two timed asynchronous π-calculi. In *Proc. CONCUR'04*, 2004.

[4] Luca Cardelli. Wide area computation. In *Proceedings of* 26^{th} *ICALP*, Lecture Notes in Computer Science, pages 10–24. Springer-Verlag, 1999.

[5] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, March 1996.

[6] Cedric Fournet, Georges Gonthier, Jean Jaques Levy, and Remy Didier. A calculus of mobile agents. *CONCUR 96*, LNCS 1119:406–421, August 1996.

[7] R.J. van Glabbeek and U. Goltz. Equivalence notions for concurrent systems and refinement of actions (extended abstract). In A. Kreczmar and G. Mirkowska, editors, Proceedings 14^{th} Symposium on *Mathematical Foundations of Computer Science*, MFCS '89, Porąbka-Kozubnik, Poland, August/September 1989, volume 379 of *lncs*, pages 237–248. Springer-Verlag, 1989.

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

[9] Matthew Hennessy and Julian Rathke. Typed behavioural equivalences for processes in the presence of subtyping. *Mathematical Structures in Computer Science*, 14:651–684, 2004.

[10] Matthew Hennessy and James Riely. Resource access control in systems of mobile agents. In Uwe Nestmann and Benjamin C. Pierce, editors, *HLCL98: High-Level Concurrent Languages (Nice, France, September 12, 1998)*, volume 16(3), pages 3–17. Elsevier Science Publishers, 1998.

[11] Matthew Hennessy and James Riely. Resource access control in systems of mobile agents. *Information and Computation*, 173:82–120, 2002.

[12] K. Honda and N. Yoshida. On reduction-based process semantics. *Theoretical Computer Science*, 152(2):437–486, 1995.

[13] Kohei Honda and Nobuko Yoshida. A uniform type structure for secure information fbw. In *29th Annual Symposium on Principles of Programming Languages*. ACM, January 2002.

[14] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[15] Nestmann, Fuzzati, and Merro. Modeling consensus in a process calculus. In *CONCUR: 14th International Conference on Concurrency Theory*. LNCS, Springer-Verlag, 2003.

[16] James Riely and Matthew Hennessy. Distributed processes and location failures. *Theoretical Computer Science*, 226:693–735, 2001.

[17] Davide Sangiorgi and David Walker. *The π-calculus*. Cambridge University Press, 2001.

[18] Richard D. Schlichting and Fred B. Schneider. Fail-stop processors: An approach to designing fault-tolerant computing systems. *Computer Systems*, 1(3):222–238, 1983.