A Theory of System Behaviour in the Presence of Node and Link Failure

Adrian Francalanza [a], Matthew Hennessy [b]

[a]*Imperial College, London SW7 2BZ, England*

[b]*University of Sussex, Brighton BN1 9RH, England.*

Abstract

We develop a behavioural theory of distributed programs in the presence of failures such as nodes crashing and links breaking. The framework we use is that of Dπ, a language in which located processes, or agents, may migrate between dynamically created locations. In our extended framework, these processes run on a distributed network, in which individual nodes may crash in fail-stop fashion or the links between these nodes may become permanently broken. The original language, Dπ, is also extended by a ping construct for detecting and reacting to these failures.

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 effect on 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 surrounding network and an observer, which can see the difference.

Key words:
distributed calculi, node and link failure, reduction barbed congruence, labelled transition systems, bisimulation

1 Introduction

It is generally accepted that *partial failures* are a major factor for precluding location transparency in distributed settings such as *wide-area networks*, [4], large computational infrastructures which may even span the globe. Because of this, various *location-aware* calculi have arisen in the literature to model the behaviour of distributed programs in the presence of failures [2,1,26], and to study the correctness of algorithms is such a setting [21,25,24,14]. The purpose of this paper is to:

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

Our point of departure is Dπ [20], a simple distributed version of the standard π-calculus [27], where the locations that host processes model closely physical network nodes. Ignoring the type system developed for Dπ, which is orthogonal to the issues addressed here, we consider the following three abstract server implementations as motivation:

$$\mathsf{server} \Leftarrow (\nu\, data)\begin{pmatrix} l[\![req?(x,y).data!\langle x,y\rangle]\!] \\ \mid l[\![data?(x,y).y!\langle f(x)\rangle]\!] \end{pmatrix}$$

$$\mathsf{servD} \Leftarrow (\nu\, data)\begin{pmatrix} l[\![req?(x,y).\mathsf{go}\ k_1.data!\langle x,y\rangle]\!] \\ \mid k_1[\![data?(x,y).\mathsf{go}\ l.y!\langle f(x)\rangle]\!] \end{pmatrix}$$

$$\mathsf{servD2Rt} \Leftarrow (\nu\, data)\left(\begin{array}{l} l\left[\!\!\left[req?(x,y).(\nu sync)\begin{pmatrix} \mathsf{go}\ k_1.data!\langle x, sync\rangle \\ \mid \mathsf{go}\ k_2.\,\mathsf{go}\ k_1.\,data!\langle x, sync\rangle \\ \mid sync?(x).y!\langle x\rangle \end{pmatrix}\right]\!\!\right] \\ \mid k_1\left[\!\!\left[data?(x,y).\begin{pmatrix} \mathsf{go}\ l.\,y!\langle f(x)\rangle \\ \mid \mathsf{go}\ k_2.\,\mathsf{go}\ l.\,y!\langle f(x)\rangle \end{pmatrix}\right]\!\!\right] \end{array}\right)$$

The three systems server, servD and $\mathsf{servD2Rt}$ implement a server that accepts a single request for processing on channel *req* at location l with two arguments, x the value to be processed and y the return channel on which to return the result of the processing. A typical client for these servers would have the form $l[\![req!\langle n, ret\rangle]\!]$, sending the name n as the value to be looked up and *ret* as the return channel.

Every server forwards the request to an internal database hidden from the client, denoted by the scoped channel *data*, which processes the value using an unspecified function $f(x)$. The three implementations differ by where the internal database is located and how it is handled. More specifically, server holds the database *locally* at l and carries out all the processing there; in contrast, servD and $\mathsf{servD2Rt}$ distribute the database *remotely* at location k_1. The latter two server implementations also differ by how the remote database is accessed: servD accesses the database using the direct route from l to k_1; $\mathsf{servD2Rt}$ forwards the service requests along two concurrent routes, that is the direct one from l to k_1 and an indirect route using an intermediary node k_2 and non-deterministically selects one of two results if both routes are active.

Intuitively, these three server implementations are not equivalent because they exhibit distinct behaviour in a setting with node and link failure. For instance, if node k_1 fails, servD and servD2Rt may not be able to service a client request whereas server would continue to work seamlessly. Moreover, servD and servD2Rt are also distinct because if the link between l and k_1 breaks, servD may block and not serve a request while servD2Rt would still operate as intended. Despite the fact that these three implementations are qualitatively different, it is hard to distinguish between them in Dπ theories such as [18].

In this paper, we develop a behavioural theory that tells these three systems apart. We use extended Dπ configurations of the form $\Sigma \triangleright N$ where Σ is a representation of the current state of the network, and N consists of the systems such as those we have just seen, that is software executing in a distributed manner over Σ. Here Σ records the set of nodes in the network, their *status* (whether they are *alive* or *dead*), and their *connectivity* (the set of symmetric links between these nodes). This results in a succinct but expressive framework, in which many of the phenomena associated with practical distributed settings, such as routing algorithms and ad-hoc network discoveries, can be examined.

The corresponding behavioural theory takes the form of *(weak) bisimulation equivalence*, based on labelled actions

$$\Sigma \triangleright N \xrightarrow{\mu} \Sigma' \triangleright N' \qquad (1)$$

where the label μ represents the manner in which an observer, also running on the network Σ, can interact with the system N. This interaction may not only change the state of the system, to N', in the usual manner, but also affect the nature of the underlying network. For instance, an observer may extend the network by creating new locations or otherwise induce faults in the network by killing sites or breaking links between sites, thereby capturing at least some of the reaction of N to dynamic failures.

It turns out that the definition of the actions in (1) needs to be relatively sophisticated: although the system and the observer may initially share the same view of the underlying network, Σ, interactions quickly give rise to situations in which these views *diverge*. More specifically, observers may learn of new nodes in the system as a result of interaction (scope extrusion), but at the same time, cannot determine the state of such nodes and the code executing at them either because the newly discovered nodes are *completely disconnected* or because the observer does not have enough information to *determine a route* which leads to these nodes. As a result, in (1) above, the network representation Σ needs to somehow record the actual full state of the underlying network, together with the *observer's partial view* of it.

Table 1. *Syntax of typed DπF*

Types

$$\mathtt{T,U,W} ::= \mathtt{ch} \mid \mathtt{loc[S,C]} \qquad \mathtt{S} ::= \mathtt{a} \mid \mathtt{d} \qquad \mathtt{C,D} ::= \{u_1,\ldots,u_n\}$$

Processes

$$P,Q ::= u!\langle V\rangle.P \mid u?(X).P \mid *\,u?(X).P \mid \textsf{if } v{=}u \textsf{ then } P \textsf{ else } Q \mid \mathbf{0} \mid P|Q \mid (\nu\, n{:}\mathtt{T})P$$
$$\mid \textsf{go } u.P \mid \textsf{kill} \mid \textsf{break } u \mid \textsf{ping } u.P \textsf{ else } Q$$

Systems

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

The paper is organised as follows: Section 2 introduces our language, DπF, its reduction semantics and a corresponding contextual equivalence, based on the notion of reduction barbed congruence of Honda *et al* [22]. In Section 3 we present an initial definition of actions for DπF, based on the general approach of [19]. The resulting bisimulation equivalence can be used to demonstrate equivalences between systems, but we show, by a series of examples, that it is too discriminating. In Section 4 , we revise the definition of these actions, by abstracting from internal information present in the action labels, and demonstrate, through a series of examples, that the resulting bisimulation equivalence corresponds, in some sense, to the contextual equivalence defined earlier in Section 2. Finally, in Section 5, we state and prove the main result of the paper, that is that the refined bisimulation is indeed *fully abstract* with respect to aforementioned contextual equivalence; this means that two systems will be differentiated by the bisimulation equivalence if and only if, in some sense, there is a computational context, consisting of a network and an observer, which can see the difference. Section 6 concludes with an overview of related work and future directions.

2 The language

We assume a set of *variables* VARS, ranged over by $x,y,z,\ldots$ and a separate set of *names*, NAMES, ranged over by $n,m,\ldots$, which is divided into locations, LOCS, ranged over by $l,k,\ldots$ and channels, CHANS, ranged over by $a,b,c,\ldots$. Finally we use $u,v,\ldots$ to range over the set of *identifiers*, consisting of either variables and names.

The syntax of DπF is given in Table 1, where the main syntactic category is that of *systems*, ranged over by M,N; these are essentially a collection of *located processes*, or *agents* $l[\![P]\!]$, but there may also be occurrences of typed *scoped names*, $(\nu\, n : \mathtt{T})N$. Our syntax is based on the language Dπ, [20], which comes endowed with a rich type system for regulating *access control*. Since this is orthogonal to our concerns we ignore this type system. Instead we use a very simple notion of type, which simply records the proposed use of an identifier. Thus, if n is used as a channel in N, then $\mathtt{T}$ is simply $\mathtt{ch}$; however if it is a location then $\mathtt{T} = \mathtt{loc}[\mathtt{S}, \mathtt{C}]$ records it's *status* $\mathtt{S}$, whether it is alive $\mathtt{a}$ or dead $\mathtt{d}$, and the set of locations $\mathtt{C}$ to which it is linked, $\{l_1, \ldots, l_n\}$. Note that these $\mathtt{T}$ should not be considered as *static types*; in particular, as a computation proceeds, the status and connectivity of locations may change. Another change from the original Dπ is that we do not speficy the location where a channel can be used; here a channel can be used at any location.

The syntax for agents, P, Q, is an extension of that in Dπ. There are input and output on channels; here V is a tuple of identifiers, and X a tuple of distinct variables, to be interpreted as a pattern. We also have the standard forms of parallel, replicated input, local declarations, a test for equality between identifiers and an asynchronous migration construct. As we shall see once we introduce the reduction semantics of this language, migration under failure assumes a different semantics from that in the original Dπ, but its characteristic asynchrony is still preserved in our language. Processes are also extended with a a conditional ping construct, $l[\![\mathsf{ping}\ k.P\ \mathsf{else}\ Q]\!]$, in the style of [2,1,26], branching to $l[\![P]\!]$ or $l[\![Q]\!]$ depending on the *accessibility* of k from l. It acts as a form of perfect failure detector [5], the implementation of which typically necessitates tighter synchronisation between locations. Despite this apparent limitation [1], the ping construct still describes the asynchrony between network failure and failure discovery/detection as two distinct and independent events. Together, the ping and asynchronous migration operations give a programming level of abstraction close an idealised form of the IP/ICMP layers in the Internet protocol suite[23]. The semantics of new location process is also different from that of Dπ, since it is subject to restrictions imposed by the present state of the network as well. In particular, new locations can only short-circuit paths of connections between locations but cannot provide a new path for two unreachable nodes. Finally, we have two new constructs to simulate failures in the style of [26]; $l[\![\mathsf{kill}]\!]$ kills the location l, while $k[\![\mathsf{break}\ l]\!]$ breaks the link between l and k, if it exists. We are not really interested in programming with these last two operators. Nevertheless, when we come to consider *contextual behaviour*, their presence will mean that the behaviour will take into account the effects of *dynamic* failures.

We relegate the standard notions of *free* and *bound* occurrences of both names and variables to the appendix (see Section A) and assume the associated concepts of α-conversion and *substitution*; see [27,17] for similar definitions. It is worth emphasising that location names, and indeed identifiers, may occur in types, and this must be taken into account when these concepts are defined. Furthermore, we

[1] The user can always program a weaker form of failure detector from the present ping which may non-deterministically give false unreachability branchings like an unreliable failure detector [5]. The programming of such a construction will become clearer once we introduce the reduction rules.

will assume that channel communication is well-sorted (for any output $a!\langle V\rangle.P$ and input $a?(X).Q$ on any channel a, we have $|V| = |X|$) that all system terms are *closed*, that is they have no free occurrences of variables.

Network representations: Reductions of systems are defined with respect to a network representation, Δ, describing the current state of the network. Intuitively Δ records the set of locations in existence, whether they are alive or dead, and any live links between them.

Definition 1 (Link sets) *Any binary relation $\mathcal{L}$ over the set of locations* Locs *is called a* linkset. *We use $\mathbf{dom}(\mathcal{L})$ to denote its domain, that is the collection of locations l such that $\langle l, k\rangle \in \mathcal{L}$, for some k.*

A linkset is meant to represent both location liveness and a collection of symmetric links *between locations. Specifically we write $\mathcal{L} \vdash l : \mathbf{alive}$ whenever $\langle l, l\rangle \in \mathcal{L}$ and $\mathcal{L} \vdash l \leftrightarrow k$ whenever*

- $\langle l, k\rangle \in \mathcal{L}$
- *or* $\langle k, l\rangle \in \mathcal{L}$.

The *reflexive* interpretation of link-sets expresses liveness and at the same time permits the smooth handling of the degenerate case of a process moving from a site l to l itself.

Definition 2 (Components) *A subset $\mathcal{K}$ of a linkset $\mathcal{L}$ is called a* component, *if all locations in $\mathcal{K}$ are mutually accessible; that is, using the obvious notation, $\mathcal{L} \vdash k_1 \leftrightarrow^* k_2$ for every $k_1, k_2 \in \mathcal{K}$. Every location $l \in \mathbf{dom}(\mathcal{L})$, generates a component:*

$$[\, l\,]_{\mathcal{L}} = \{\langle k_1, k_2\rangle \in \mathcal{L} \mid \mathcal{L} \vdash l \leftrightarrow^* k_1 \text{ or } \mathcal{L} \vdash l \leftrightarrow^* k_2\}$$

In the special case where every location $l \in \mathbf{dom}(\mathcal{L})$ is alive, that is $\mathcal{L} \vdash l : \mathbf{alive}$, then it is easy to check that every linkset can be partitioned into components. *For one can verify that*

- $\mathcal{L} = \bigcup_{l \in \mathbf{dom}(\mathcal{L})} [\, l\,]_{\mathcal{L}}$
- $[\, l_1\,]_{\mathcal{L}} \cap [\, l_2\,]_{\mathcal{L}} \neq \emptyset$ *implies* $[\, l_1\,]_{\mathcal{L}} = [\, l_2\,]_{\mathcal{L}}$.

Components, will play an essential role in Section 4

Definition 3 (Network representation) *A* network representation, Δ, *is any tuple $\langle \mathcal{N}, \mathcal{L}\rangle$ where*

- *$\mathcal{N}$ is a set of names, divided into* $\mathbf{loc}(\mathcal{N})$, *location names, and* $\mathbf{chan}(\mathcal{N})$, *channel names*
- *$\mathcal{L} \subseteq \mathbf{loc}(\mathcal{N}) \times \mathbf{loc}(\mathcal{N})$ is a* linkset *representing* both *the set of live locations, sometimes refered to as the* liveset *of Δ and the set of connections between locations.*

Notation: For convenience we use $\Delta_{\mathcal{N}}$, and $\Delta_{\mathcal{L}}$ to denote the individual components of a network representation Δ, and we use the following notation for extracting information from Δ:

- $\Delta \vdash l : \textbf{alive}$ whenever $\Delta_{\mathcal{L}} \vdash l : \textbf{alive}$.
- $\Delta \vdash l \leftrightarrow k$ if $\Delta_{\mathcal{L}} \vdash l \leftrightarrow k$.
- $\Delta \vdash k \leftrightsquigarrow l$ if $\Delta \vdash l \leftrightarrow k$, $\Delta \vdash l : \textbf{alive}$ and $\Delta \vdash k : \textbf{alive}$.

Thus $\Delta \vdash k \leftrightsquigarrow l$ not only means that is there a link between k and l but both ends of the link are alive; we will refer to this as a *live* link.

To update network representations we use the following:

- *Extending a network:* $\Delta + n : \mathtt{T}$ is only defined when n is *fresh* to Δ; if $\mathtt{T}$ is $\mathtt{ch}$, this simply adds n to the channel component of $\Delta_{\mathcal{N}}$. But if it is the location type then in addition to adding n to the location component of $\Delta_{\mathcal{N}}$, it needs to add in the new links determined by the location type $\mathtt{T}$, and possibly update liveness information in $\Delta_{\mathcal{L}}$ for n. Formally we have

$$\begin{aligned}
\Delta + a{:}\mathtt{ch} &= \langle \Delta_{\mathcal{N}} \cup \{a\},\ \Delta_{\mathcal{L}} \rangle \\
\Delta + k{:}\mathtt{loc}[\mathtt{a}, \{l_1, \ldots l_n\}] &= \langle \Delta_{\mathcal{N}} \cup \{k\},\ \Delta_{\mathcal{L}} \cup \{\langle l_i, k\rangle\} \cup \{\langle k, k\rangle\} \rangle \\
\Delta + k{:}\mathtt{loc}[\mathtt{d}, \{l_1, \ldots l_n\}] &= \langle \Delta_{\mathcal{N}} \cup \{k\},\ \Delta_{\mathcal{L}} \cup \{\langle l_i, k\rangle\} \rangle
\end{aligned}$$

- *location killing:* $\Delta - l$ is always defined; it simply removes l from the liveset of Δ, if it is there:

$$\Delta - l = \langle \Delta_{\mathcal{N}},\ \Delta_{\mathcal{L}} \setminus \{\langle l, l\rangle\} \rangle$$

- *link breaking:* this operation, $\Delta - l \leftrightarrow k$ is also always defined; it removes from $\Delta_{\mathcal{L}}$ any representation of the link between l and k:

$$\Delta - l \leftrightarrow k = \langle \Delta_{\mathcal{N}},\ \Delta_{\mathcal{L}} \setminus \{\langle l, k\rangle, \langle k, l\rangle\} \rangle$$

Reduction semantics: A *configurations* consists of a pair $\Delta \triangleright N$, where every free name in N occurs in the name component of Δ. We define reductions to take place between such configurations; thus they take the form of a binary relation

$$\Delta \triangleright N \longrightarrow \Delta' \triangleright N' \qquad (2)$$

where Δ and Δ' in (2) are network representations. The novelty of these judgements arises from the fact that certain nodes may not be interconnected, and indeed some may not be alive.

The rules governing the reductions (2) are given in Tables 2, 3 and 4; note that every rule depends on the requirement that l, the location of the activity, is currently alive; this is the intent of the predicate $\Delta \vdash l : \textbf{alive}$. Table 2 gives the standard rules for local communication, and the management of replication, matching and parallelism, derived from the corresponding rules for Dπ in [20]. The communication rule (r-comm) depends on a standard notion of *substitution* $Q\{^V\!/\!_X\}$, the details of which we omit. Intuitively the value V is matched against the pattern X, and the resulting substitution is applied to Q. Of course if V does not match X a runtime error occurs, but these could be eliminated by the use of a simple, and standard, type system.

The rules in Table 3 are more interesting. Rules (r-go) and (r-ngo) state that a migration is successful depending on the accessibility of the destination; migration is asynchronous in the sense that code at the source location still migrates, irrespective of the destination's accessibility. Similarly, (r-ping) and (r-nping) are subject to the same condition for the respective branchings; they however have a more synchronous flavour to them since the branching outcome is visible at the testing location. Ping may also be seen as a form of perfect failure detector [5]; note however that $l[\![\mathsf{ping}\ k.P\ \mathsf{else}\ Q]\!]$ yields *partial information* about the state of the underlying network. More precisely, it can only determine that k is inaccessible, but does not give information on whether this is caused by the failure of node k, the absence of the link $l \leftrightarrow k$, or both; see Example 4. The rules (r-kill), (r-brk) make the obvious changes to the current network. Finally (r-newc) and (r-newl) regulates the generation of new names. We consciously choose not to express name generation as a (reversible) structural rule since, in practice, this would require some form of resource acquisition and initialisation which may not be reversible; for similar reasons (r-fork) is not structural since we interpret it as thread spawning. But perhaps a stronger justification for our design choice is given by (r-newl), describing the launching of new location. In this case, location creation cannot be reversed and recreated because creation depends on the current state of the network which may change during computation. More specifically, in $l[\![(\nu\, k : \mathtt{loc}[\mathsf{a}, \mathsf{C}])\, P]\!]$ the location l *requests* to generate a new (live) location k, with connections to the locations mentioned in C. But the ability to establish these connections depends on the existing connectivity of the the parent location l. There are a number of reasonable possibilities as to what to do in this situation: we felt it was most realistic to establish a connection from the new location k to the parent l, and in addition, to only establish connections to those locations in C which are *accessible* from the parent l via a sequence of live links; see Example 5. Note also that we do not have a reduction rule for launching new dead locations; one can easily be added, but there is no reason why any such new locations should ever be generated.

Finally, in Table 4 we have an adaptation of the standard *contextual* rules, which allow the basic reductions to occur in *evaluation contexts*. The rule (r-str) allows reductions up to a structural equivalence, in the standard manner, using the identities in Table 5. The only non-trivial identities in Table 5 are (s-flip-1) and (s-flip-2), where the types of the successively scoped locations need to be changed if they denote a link between them, thus avoiding unwanted name capture. The operations

Table 2. *Local Reduction Rules for DπF*

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

(r-comm)

$$\frac{}{\Delta \rhd l\llbracket a!\langle V\rangle.P\rrbracket \mid l\llbracket a?(X).Q\rrbracket \quad\longrightarrow\quad \Delta \rhd l\llbracket P\rrbracket \mid l\llbracket Q\{^{V}\!/X\}\rrbracket}$$

(r-rep)

$$\frac{}{\Delta \rhd l\llbracket * a?(X).P\rrbracket \quad\longrightarrow\quad \Delta \rhd l\llbracket a?(X).(P| * a?(X).P)\rrbracket}$$

(r-fork)

$$\frac{}{\Delta \rhd l\llbracket P|Q\rrbracket \quad\longrightarrow\quad \Delta \rhd l\llbracket P\rrbracket \mid l\llbracket Q\rrbracket}$$

(r-eq)

$$\frac{}{\Delta \rhd l\llbracket \textsf{if } u = u \textsf{ then } P \textsf{ else } Q\rrbracket \longrightarrow \Delta \rhd l\llbracket P\rrbracket}$$

(r-neq)

$$\frac{}{\Delta \rhd l\llbracket \textsf{if } u = v \textsf{ then } P \textsf{ else } Q\rrbracket \longrightarrow \Delta \rhd l\llbracket Q\rrbracket}\ u \neq v$$

Table 3. *Network Reduction Rules for DπF*

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

(r-go)

$$\frac{}{\Delta \rhd l\llbracket \textsf{go } k.P\rrbracket \longrightarrow \Delta \rhd k\llbracket P\rrbracket}\ \Delta \vdash k \leftrightsquigarrow l$$

(r-ngo)

$$\frac{}{\Delta \rhd l\llbracket \textsf{go } k.P\rrbracket \longrightarrow \Delta \rhd k\llbracket \mathbf{0}\rrbracket}\ \Delta \nvdash k \leftrightsquigarrow l$$

(r-ping)

$$\frac{}{\Delta \rhd l\llbracket \textsf{ping } k.P \textsf{ else } Q\rrbracket \longrightarrow \Delta \rhd l\llbracket P\rrbracket}\ \Delta \vdash k \leftrightsquigarrow l$$

(r-nping)

$$\frac{}{\Delta \rhd l\llbracket \textsf{ping } k.P \textsf{ else } Q\rrbracket \longrightarrow \Delta \rhd l\llbracket Q\rrbracket}\ \Delta \nvdash k \leftrightsquigarrow l$$

(r-kill)

$$\frac{}{\Delta \rhd l\llbracket \textsf{kill}\rrbracket \longrightarrow (\Delta - l) \rhd l\llbracket \mathbf{0}\rrbracket}$$

(r-brk)

$$\frac{}{\Delta \rhd l\llbracket \textsf{break } k\rrbracket \longrightarrow (\Delta - l \leftrightarrow k) \rhd l\llbracket \mathbf{0}\rrbracket}$$

(r-newc)

$$\frac{}{\Delta \rhd l\llbracket (\nu\, c : \mathrm{ch})\, P\rrbracket \longrightarrow \Delta \rhd (\nu\, c : \mathrm{ch})\, l\llbracket P\rrbracket}$$

(r-newl)

$$\frac{}{\Delta \rhd l\llbracket (\nu\, k : \mathtt{loc}[\mathtt{a}, \mathtt{C}])\, P\rrbracket \longrightarrow \Delta \rhd (\nu\, k : \mathtt{loc}[\mathtt{a}, \mathtt{D}])\, l\llbracket P\rrbracket}\ \mathtt{D} = \{m \in \mathtt{C} \cup \{l\} \mid \Delta \vdash l \leftrightsquigarrow^{*} m\}$$

Table 4. *Contextual Reduction Rules for DπF*

(r-str)

$$\frac{N' \equiv N \quad \Delta \triangleright N \longrightarrow \Delta' \triangleright M \quad M \equiv M'}{\Delta \triangleright N' \longrightarrow \Delta' \triangleright M'}$$

(r-ctxt-rest)

$$\frac{\Delta + n : \mathtt{T} \triangleright N \;\longrightarrow\; \Delta' + n : \mathtt{U} \triangleright M}{\Delta \triangleright (\nu\, n : \mathtt{T})N \;\longrightarrow\; \Delta' \triangleright (\nu\, n : \mathtt{U})M}$$

(r-ctxt-par)

$$\frac{\Delta \triangleright N \;\longrightarrow\; \Delta' \triangleright N'}{\Delta \triangleright N|M \;\longrightarrow\; \Delta' \triangleright N'|M}$$

Table 5. *Structural Rules for DπF*

(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-1)	$(\nu\, n{:}\mathtt{T})(\nu\, m{:}\mathtt{U})N \equiv (\nu\, m{:}\mathtt{U})(\nu\, n{:}\mathtt{T})N$	$n \notin \mathbf{fn}(\mathtt{U}),\ m \notin \mathbf{fn}(\mathtt{T})$
(s-flip-2)	$(\nu\, l{:}\mathtt{T})(\nu\, k{:}\mathtt{U})N \equiv (\nu\, k : \mathtt{U}{-}l)(\nu\, l : \mathtt{T}{+}k)N$	$l \in \mathbf{fn}(\mathtt{U})$
(s-inact)	$(\nu\, n{:}\mathtt{T})N \equiv N$	$n \notin \mathbf{fn}(N)$

$\mathtt{T} - l$ and $\mathtt{T} + l$ have the obvious definitions:

$$\mathtt{T} - l = \begin{cases} \mathtt{T} & \text{if } \mathtt{T} = \mathtt{ch} \\ \mathtt{loc}[\mathtt{S}, \mathtt{C} \setminus \{l\}] & \text{if } \mathtt{T} = \mathtt{loc}[\mathtt{S}, \mathtt{C}] \end{cases} \quad \mathtt{T} + l = \begin{cases} \mathtt{T} & \text{if } \mathtt{T} = \mathtt{ch} \\ \mathtt{loc}[\mathtt{S}, \mathtt{C} \cup \{l\}] & \text{if } \mathtt{T} = \mathtt{loc}[\mathtt{S}, \mathtt{C}] \end{cases}$$

The rules (r-ctxt-par) and (r-ctxt-rest) allow reductions to occur under contexts; note that the latter is somewhat non-standard, but as reductions may induce faults in the network, it may be that the status and connectivity of the scoped (location) name n is affected by the reduction, thereby changing $\mathtt{T}$ to $\mathtt{U}$. Also in the former it is worth remarking, that since the reduction semantics is only defined on configurations, the free names of M are guaranteed to occur in Δ.

This completes our exposition of the reduction semantics. At this point, we should point out that in a configuration such as $\Delta \triangleright N$, contrary to what we have implied up to now, Δ does not give a completely true representation of the network on which the code in N is running; the type information associated with scoped locations encodes parts of the network Δ that is hidden from the observer.

Example 4 (Syntax and Reductions) *Let Δ represent the network $\langle\{l, a\}; \{\langle l, l\rangle\}\rangle$ consisting of a channel a and a live node l and M_1 the system*

$$(\nu k_2 : \mathtt{loc}[\mathsf{a}, \emptyset])\,(\nu k_1 : \mathtt{loc}[\mathsf{d}, \{l, k_2\}])\; (l[\![a!\langle k_2\rangle.P]\!] \,|\, k_2[\![Q]\!])$$

Here M_1 has two new locations k_1, k_2, where k_1 is dead and linked to the existing node l and k_2 is alive linked to k_1. Although Δ only contains one node l, the located process $l[\![a!\langle k_2\rangle.P]\!]$ (as well as $k_2[\![Q]\!]$) is running on a network of three nodes, *two of which, k_1, k_2 are scoped, that is not available to other systems. We can informally represent this implicit network by*

$$\begin{tikzcd}
\overset{l}{\circ} \arrow[r, leftrightarrow] & \overset{k_1}{\bullet} \arrow[r, leftrightarrow] & \overset{k_2}{\circ}
\end{tikzcd}$$

where the nodes $\circ$ and $\bullet$ denote live and dead nodes respectively. Note that the same network could be denoted by the system N_1

$$(\nu k_1 : \mathtt{loc}[\mathsf{d}, \{l\}])\,(\nu k_2 : \mathtt{loc}[\mathsf{a}, \{k_1\}])\; (l[\![a!\langle k_2\rangle.P]\!] \,|\, k_2[\![Q]\!])$$

Note also that the two systems are structurally equivalent, $M_1 \equiv N_1$, through (s-flip-2). *As a notational abbreviation, in future examples we will omit the status annotation* a *in live location declarations and simply denote location types of the form* $\mathtt{loc}[\mathsf{a}, \mathsf{C}]$ *as* C*; so for example system N_1 would be given as*

$$(\nu k_1 : \mathtt{loc}[\mathsf{d}, \{l\}])\,(\nu k_2 : \{k_1\})\; (l[\![a!\langle k_2\rangle.P]\!] \,|\, k_2[\![Q]\!])$$

If O is an observer defined as

$$l[\![a?(x).\mathsf{ping}\, x.ok!\langle\rangle \;\mathsf{else}\, Nok!\langle\rangle]\!]$$

then the configuration $\Delta \triangleright N_1|O$ can reduce in two steps to

$$\Delta \triangleright N_1|O \longrightarrow \cdot \longrightarrow (\nu k_1 : \mathtt{loc}[\mathsf{d}, \{l\}])\,(\nu k_2 : \{k_1\})\; (l[\![P]\!] \,|\, k_2[\![Q]\!] \,|\, l[\![Nok!\langle\rangle]\!])$$

The rules (r-comm), (r-str) *and* (s-extr) *are used for the first reduction involving communication and subsequent scope extrusion of k_2 on channel a at l, and* (r-nping) *for the second reduction involving the ping test by the observer on the newly discovered location k_2; when describing derivation of reductions we generally do not mention the use of contextual rules of Table 4.*

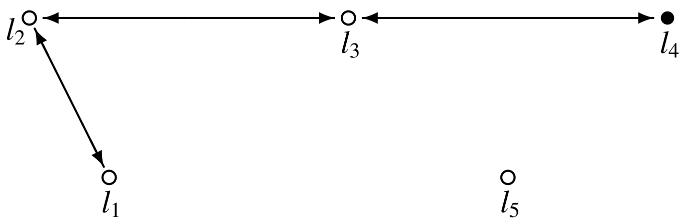
This example highlights the fact that even though the observer has taken the negative branch when pinging k_2, it can only deduce that k_2 is unreachable from the host location, l; it cannot infer anything else about location k_2 in terms of its status, its connections to other locations, and the code residing there, $k_2[\![Q]\!]$.

The inability of observers to discover the full extent of the structure of the implicit network of a system and to interact with code located at inaccessible locations, as seen in Example 4, constitutes a major difficulty in developing a theory for this language. This will be discussed in more detail in Section 3 and Section 4.

Example 5 *Consider the system* launchNewLoc *defined by*

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

running on a network Δ *consisting of five locations* $l_1, \ldots, l_5$*, 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:*



Formally describing Δ *is slightly more tedious:*

- $\Delta_{\mathcal{N}}$ *is* $\{a,\ l_1,\ l_2,\ l_3,\ l_4,\ l_5\}$
- $\Delta_{\mathcal{L}}$ *is given by* $\{\underbrace{\langle l_1, l_1\rangle,\ \langle l_2, l_2\rangle,\ \langle l_3, l_3\rangle,\ \langle l_5, l_5\rangle}_{\textit{live locations}},\ \underbrace{\langle l_1, l_2\rangle,\ \langle l_2, l_3\rangle,\ \langle l_3, l_4\rangle}_{\textit{live links}}\}$.

When we apply the reduction semantics to the configuration $\Delta \triangleright$ launchNewLoc*, the rule (*r-comm*) is used first to allow the communication of the value* l_1 *along* a*, that is*

$$\Delta \triangleright \textit{launchNewLoc} \longrightarrow \Delta \triangleright l_3[\![(\nu k : \{l_1, l_2, l_4, l_5\})P\{^{l_1}/_{x}\}]\!]$$

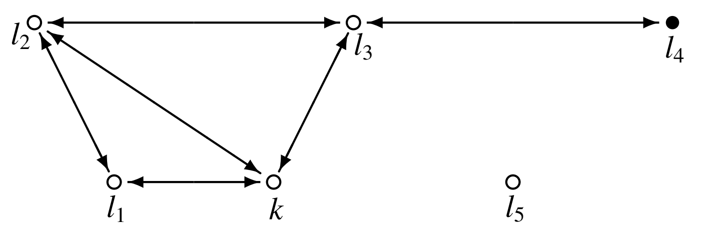
We highlight that the communication instantiates the variable x *in the type of* k *to* l_1*. At this point (*r-newl*) can be used to launch the declaration of* k *to the system level. However, when launched,* k *turns out to be connected only to* $\{l_1, l_2, l_3\}$ *because:*

- *the location from where the new location* k *is launched, that is* l_3*, is automatically connected to* k
- l_1 *and* l_2 *are reachable from the location where* k *is launched, namely* l_3*; we have* $\Delta \vdash l_3 \leftrightsquigarrow^* l_1$ *and* $\Delta \vdash l_3 \leftrightsquigarrow^* l_2$*.*
- l_4 *and* l_5 *are not accessible from* l_3*;* 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:

$$\Delta \triangleright (\nu k : \{l_1, l_2, l_3\})\ \ l_3[\![P\{^{l_1}/_{x}\}]\!]$$

The network Δ *of course does not change, but if we focus on the system* $l_3[\![P\{^{l_1}/_{x}\}]\!]$*, we see that it is running on the implicit network represented by:*



Reduction Barbed Congruence: We borrow the framework of [18] to define a variant of a contextual equivalence, originally proposed in [22], to be able to compare the behaviour of arbitrary systems M and N running on the same network Δ, denoted as:

$$\Delta \models M \cong N \tag{3}$$

We first require some preliminary definitions.

Definition 6 (Typed Relation) *A* typed relation *over systems is a family of binary relations between systems, $\mathcal{R}$, indexed by network representations. We write $\Delta \models M \;\mathcal{R}\; N$ to mean that systems M and N are related by $\mathcal{R}$ at index Δ, that is $M \;\mathcal{R}_\Delta\; N$, and moreover $\Delta \triangleright M$ and $\Delta \triangleright N$ 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.

First let us define what kinds of observing systems are allowed to run on a given network. The intuition of a *valid* observer system O in a distributed setting Δ, denoted as $\Delta \vdash_O O$, is that O originates from some live location HOME, *fresh* to the observed system, migrates to any location in $\mathbf{loc}(\Delta_N)$ to interact with (observe) processes there and then returns back to the originating fresh location HOME to compare its observations with other observers. Our formal definition will not actually mention this fresh home location HOME, as we leave it to the observer itself to both generate and manage it. But a major consequence of this view of observers is that, in view of the reduction rule (r-ngo), observing code can never reach dead locations; this constraint is reflected in the following formal definition of $\Delta \vdash_O O$.

Definition 7 (Observers) *$\Delta \vdash_O O$ is the least relation which satisfies:*

- $\Delta \vdash_O l[\![P]\!]$ *if* $\mathbf{fn}(P) \subseteq \Delta_N$ *and* $\Delta \vdash l : \mathbf{alive}$
- $\Delta \vdash_O (\nu n{:}\mathrm{T})N$ *if* $(\Delta + n{:}\mathrm{T}) \vdash_O N$
- $\Delta \vdash_O M \,|\, N$ *if* $\Delta \vdash_O M$ *and* $\Delta \vdash_O N$

These observers are the main ingredient of the following definition of contextuality.

Definition 8 (Contextual typed relations) *A typed relation $\mathcal{R}$ over configurations is* contextual *if:*

(Parallel Systems)

- $\Delta \models M \,\mathcal{R}\, N$ *and* $\Delta \vdash_{\mathrm{O}} O$ *implies* $\Delta \models M|O \,\mathcal{R}\, N|O$ *and* $\Delta \models O|M \,\mathcal{R}\, O|N$

(Network Extensions)

- $\Delta \models M \,\mathcal{R}\, N$ *and n fresh to* Δ *implies* $\Delta{+}n{:}\mathtt{T} \models M \,\mathcal{R}\, N$

Definition 9 (Reduction barbed congruence) *First we define the adaptation of the other standard relations required to define* reduction barbed congruence.

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

Reduction Closed: *A typed relation $\mathcal{R}$ over configurations is* reduction closed *whenever* $\Delta \models N \;\mathcal{R}\; M$ *and* $\Delta \triangleright N \longrightarrow \Delta' \triangleright N'$ *implies* $\Delta \triangleright M \longrightarrow^* \Delta' \triangleright M'$ *for some* $\Delta' \triangleright M'$ *such that* $\Delta' \models N' \;\mathcal{R}\; M'$*.*

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 for each index Δ the relation $\cong_\Delta$ is an equivalence relation.

As expected, in a setting with both node and link failures one can discriminate more than in a setting with node failures only. In particular, we were unable to encode a synchronous move in our calculus, even in the presence of the ping construct which offers perfect failure detection; we discuss this in the following example.

Example 10 (Synchronous Moves) *Consider the construct* $\mathsf{move}\, k.P \;\mathsf{else}\; Q$ *which attempts to migrate P to k from the current destination and if it fails, launches Q locally; such an intuitive construct is commonly found in distributed computing libraries, such as [23] for TCP, and languages for distributed computing, such as*

[16]. Assuming $\Delta \vdash l : \mathbf{alive}$*, the behaviour of this construct could be defined as*

$$\textsf{(r-move)} \quad \frac{}{\Delta \triangleright l[\![\textsf{move}\ k.P\ \textsf{else}\ Q]\!] \longrightarrow \Delta \triangleright k[\![P]\!]}\ \Delta \vdash k \leftrightsquigarrow l$$

$$\textsf{(r-nmove)} \quad \frac{}{\Delta \triangleright l[\![\textsf{move}\ k.P\ \textsf{else}\ Q]\!] \longrightarrow \Delta \triangleright l[\![Q]\!]}\ \Delta \nvdash k \leftrightsquigarrow l$$

It would be tempting to implement the move *construct in our language, considering* $l[\![\textsf{mv}\ k.P\ \textsf{else}\ Q]\!]$ *as a macro for*

$$l\left[\!\!\left[(\nu\, a, b)\begin{pmatrix}\textsf{go}\ k.(b?().P \mid \textsf{go}\ l.a?().\textsf{go}\ k.b!\langle\rangle)\\ \mid a!\langle\rangle \mid \textsf{monitor}_a\ k.Q\end{pmatrix}\right]\!\!\right]$$

where $a, b \notin \mathbf{fn}(P, Q)$*. In essence, our implementation sends P to k as an input guarded process on the scoped channel b, goes back to the source location, l, to signal the successful arrival of P at k, by synchronising on the scoped channel a, and finally goes back to k to release P by outputting on b. This implementation uses mutual exclusion on the scoped channel a, together with the macro* $\textsf{monitor}_a\ k.Q$*. This macro repeatedly tests the accessibility of a location k from the hosting location l and launches Q locally at l when k becomes inaccessible. Before it performs every accessibility test,* $\textsf{monitor}_a\ k.Q$ *synchronizes on the channel a; this ensures that either* $k[\![P]\!]$ *or* $l[\![Q]\!]$ *(but not both) is eventually released, as required. The monitor process, derived from [10,14], which we denote by* $\textsf{monitor}_a\ k.Q$ *is encoded in our language as:*

$$(\nu\, test : \mathrm{ch})(\, test!\langle\rangle \mid *\, test?().a?().\textsf{ping}\ k.\,(test!\langle\rangle \mid a!\langle\rangle)\ \textsf{else}\ Q\,)$$

Assuming $\Delta_{l,k} = \langle\{l, k\}, \{\langle l, l\rangle, \langle k, k\rangle, \langle l, k\rangle\}\rangle$*, the construct* $l[\![\textsf{move}\ k.P\ \textsf{else}\ Q]\!]$ *and its corresponding implementation* $l[\![\textsf{mv}\ k.P\ \textsf{else}\ Q]\!]$ *turn out to be observational equivalent in a framework where* only location failure *is allowed.*

This is however not true *in our setting, where link failures may also occur, because from the contextuality property of* $\cong$ *we can conclude*

$$\Delta_{l,k} \models l[\![\textsf{move}\ k.P\ \textsf{else}\ Q]\!] \mid l[\![\textsf{break}\ k]\!] \ncong l[\![\textsf{mv}\ k.P\ \textsf{else}\ Q]\!] \mid l[\![\textsf{break}\ k]\!] \quad (4)$$

More specifically, on the right hand side of (4) we can reduce to a state where $b?().P$ *reaches k successfully while* $\textsf{go}\ l.a?().\textsf{go}\ k.b!\langle\rangle$ *also manages to go back to l as well, and subsequently synchronises on channel a successfully, thereby blocking* $\textsf{monitor}_a\ k.Q$*. Now if at this point, the link* $l \leftrightarrow k$ *breaks because* $l[\![\textsf{break}\ k]\!]$ *reduces, the residue that has to go back to k to trigger* $b?().P$*, that is* $\textsf{go}\ k.b!\langle\rangle$*, cannot reach k and we end up with a situation where both locations l and k are alive, but both* *branches P and Q are blocked. This state can never be reached by the configuration* $\Delta_{l,k} \triangleright l[\![\mathsf{move}\, k.P\ \mathsf{else}\, Q]\!] \mid l[\![\mathsf{break}\, k]\!]$ *on the left hand side, where branching is an atomic operation*[2] *.*

Example 11 (Distributed Servers) *Let* Δ *represent the following network:*

$$\begin{tikzcd}
 & k_2 \arrow[dl, leftrightarrow] \arrow[dr, leftrightarrow] & \\
l \arrow[rr, leftrightarrow] & & k_1
\end{tikzcd}$$

Formally Δ *is determined by letting* $\Delta_{\mathcal{N}}$ *be* $\{l, k_1, k_2, \mathsf{req}, \mathsf{ret}\}$ *and* $\Delta_{\mathcal{L}}$ *be* $\{\langle l,l\rangle,\ \langle k_1,k_1\rangle,\ \langle k_2,k_2\rangle,\ \langle l,k_1\rangle,\ \langle l,k_2\rangle,\ \langle k_1,k_2\rangle\}$.

The distributed server implementations, servD *and* $\mathsf{srvD2Rt}$*, presented earlier in the Introduction, Section 1, 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 which breaks the link* $l \leftrightarrow k_1$.

$$C_3 \;=\; [-] \mid l[\![req!\langle l, ret\rangle]\!] \mid l[\![\mathsf{break}\, k_1]\!]$$

Stated otherwise, if the link $l \leftrightarrow k_1$ *breaks,* $\mathsf{srv2Rt}$ *will still be able to operate normally and perform a barb on ret@l;* servD*, 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 emit a barb ret@l. However consider the alternative remote server* srvMtr*, defined as:*

$$(\nu\, data)\left(\begin{array}{l}
l\left[\!\!\left[req?(x,y).(\nu sync)\left(\begin{array}{l}\mathsf{go}\, k_1.\, data!\langle x, sync\rangle\\ \mid \mathsf{monitor}\, k_1.\mathsf{go}\, k_2.\mathsf{go}\, k_1.data!\langle x, sync\rangle\\ \mid sync?(x).y!\langle x\rangle\end{array}\right)\right]\!\!\right]\\[2em]
\mid k_1\left[\!\!\left[data?(x,y).\left(\begin{array}{l}\mathsf{go}\, l.\, y!\langle f(x)\rangle\\ \mid \mathsf{monitor}\, l.\mathsf{go}\, k_2.\mathsf{go}\, l.y!\langle f(x)\rangle\end{array}\right)\right]\!\!\right]
\end{array}\right)$$

using a simplified version of the monitor macro of Example 10 which does not synchronise on the channel a for every test. It is defined as

[2] The same situation, that is $\mathsf{go}\ k.b!\langle\rangle$ not being able to reach k, can also happen in a setting where only location failure can occur. However, this can only be caused as a result of k failing, which prohibits any observer from determining whether $b?().P$ was triggered or not.

$$\textsf{monitor}\, k.Q \Leftarrow (\nu\, test\colon \mathrm{ch})(\, test!\langle\rangle \mid *\, test?().\textsf{ping}\, k.\, test!\langle\rangle\ \textsf{else}\, Q\,)$$

In order to establish $\Delta \models \textsf{srv2Rt} \cong \textsf{srvMntr}$ *directly from the definition of reduction barbed congruence, we would need to compare the behaviour of the two systems relative to all valid contexts. But in the next section we will develop more realistic methods for establishing such identities.*

In the next example we examine the interplay between dead nodes and dead links and their respective observation. This example exposes some of the complications we shall encounter when we develop a bisimulation theory for our language.

Example 12 (Network Observations) *Let* Δ_l *be the simple network with one live location l, denoted and depicted as:*

$$\Delta_l = \langle\{l,a\},\{\langle l,l\rangle\}\rangle \qquad = \qquad \circ$$

and consider the following three networks,

$$\Delta_1 \;=\; \Delta_l + k\colon\mathtt{loc[d,}\{l\}\mathtt{]} \;=\; \begin{tikzcd} \overset{l}{\circ} \arrow[rr, leftrightarrow] & & \overset{k}{\bullet} \end{tikzcd}$$

$$\Delta_2 \;=\; \Delta_l + k\colon\mathtt{loc[d,}\emptyset\mathtt{]} \;=\; \begin{tikzcd} \overset{l}{\circ} & & \overset{k}{\bullet} \end{tikzcd}$$

$$\Delta_3 \;=\; \Delta_l + k\colon\mathtt{loc[a,}\emptyset\mathtt{]} \;=\; \begin{tikzcd} \overset{l}{\circ} & & \overset{k}{\circ} \end{tikzcd}$$

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

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

respectively. As in Example 4, 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 rules (r-ngo) *and* (r-nping)*, we conclude that any attempt to move to or test k from l, where the observer would be located, 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 A partial view labelled transition system for DπF

It would be tempting to define a bisimulation equivalence based on actions of the form

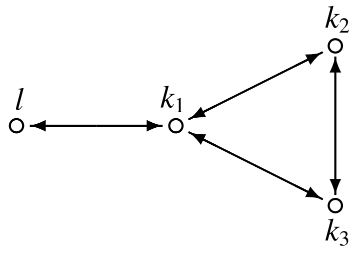
$$\Delta \triangleright M \xrightarrow{\mu} \Delta' \triangleright M'$$

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

Example 13 (Partial Views) *Let Δ_l be the network in which there is only one node l which is alive, defined earlier in Example 12, and consider the system M_1 defined by*

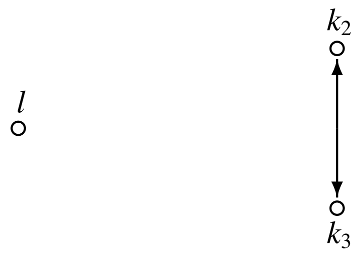
$$(\nu k_1 : \{l\})\, (\nu k_2 : \{k_1\})\, (\nu k_3 : \{k_1, k_2\})\; l[\![a!\langle k_2, k_3\rangle.P]\!] \mid k_2[\![Q]\!]$$

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 implicit *network:*



(5)

Let us now see to what knowledge of this implicit network can be gained by an observer O at site l, such as $l[\![a?(x,y).O(x,y)]\!]$, where $O(x,y)$ is a process with free variables x and 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_2 and k_3, thereby evolving to $l[\![O(k_2, k_3)]\!]$. Yet, even though it is in possession of these two names, it cannot discover their status in terms of liveness of the nodes and links between them and other free locations, such as $k_2 \leftrightarrow k_3$, nor can it interact with code residing at any of these locations, such as $k_2[\![Q]\!]$. This is due to the fact that the observer is not aware of the local name k_1 and thus cannot determine a path to k_2 and k_3 and discover the full extent of the sub-network with nodes l, k_2 and k_3, which can be denoted diagrammatically as



Rather, the observer's view of the sub-network with nodes l, k_2 and k_3 looks more like

$$\overset{l}{\circ} \qquad\qquad\qquad\qquad\qquad\qquad \overset{k_2}{?} \qquad\qquad \overset{k_3}{?}$$

This means that there is now a difference between the actual network being used by the system, (5), and the observer's view *of that network; the observer has a* partial view *of the network. However, the formalism of our current network does not allow us to represent the differing observer partial view of the network.*

A closer inspection of the above example reveals that our requirements for multiple network views are even more complicated. For instance, the network status information relating to the newly discovered nodes k_2 and k_3 are not necessarily hidden forever from the observer and may become accessible later through subsequent interactions. For example, if P and $O(k_2, k_3)$ stand for

$$b!\langle k_1\rangle.P' \quad \textit{and} \qquad\qquad b?(x).\mathsf{go}\ x.\mathsf{go}\ k_2.\mathsf{ping}\ k_3.R_1\ \ \mathsf{else}\, R_2$$

respectively, the observer can interact on channel b with P, gain the knowledge of the scoped location k_1, determine the live path $l \leftrightsquigarrow k_1 \leftrightsquigarrow k_2$ and subsequently discover information such as the liveness of k_2 and k_3 and the live link $k_2 \leftrightsquigarrow k_3$.

An lts semantics has to record the differences between the network and the observers view of networks. A new network representation for our specific case therefore needs first and foremost, distinguish between observable nodes and unobservable ones such as k_2 and k_3, but also to keep network status (that is liveness and connections) relating to the unobservable nodes k_2 and k_3 that may later become observable. This requires extra information being recorded in network representations.

Definition 14 (Effective LinkSet) *A linkset $\mathcal{L}$ is effective if it satisfies the condition:*

$$\mathcal{L} \nvdash l : \textbf{alive} \ \ \textit{implies} \ \ \nexists k.\mathcal{L} \vdash l \leftrightarrow k$$

Effective linkset do not represent links where one of the endpoints is dead since these turn out to be unobservable by the constructs of DπF. The operation

$$|\mathcal{L}| \ = \ \mathcal{L} \setminus \{\langle l, k\rangle, \langle k, l\rangle \mid \mathcal{L} \nvdash l : \textbf{alive}\}$$

converts any linkset into an effective linkset by removing links with dead endpoints. For any set of names $N \subseteq$ Names we also define the filtering effective linkset operation

$$\mathcal{L} \setminus N = \{l \leftrightarrow k \mid \ l \leftrightarrow k \in \mathcal{L} \ \wedge \ \{l, k\} \cap N = \emptyset\}$$

Definition 15 (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 an* effective linkset*, denoting the live locations and links between them that are* observable *by the context*
- $\mathcal{H}$ *is another* effective linkset*, denoting the live locations and links between them 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.*

Effective network representations embody the notion of *observer partial view* discusses above. 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 not represented directly in Σ, but they can be easily seen to be $\mathbf{loc}(\mathcal{N}) \setminus \mathbf{dom}(\mathcal{O} \cup \mathcal{H})$; that is, all the location names in $\mathcal{N}$ that are not mentioned in either $\mathcal{O}$ or $\mathcal{H}$. We will refer to them 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 holds all the necessary information from the observer's point of view, that is, the names known, $\mathcal{N}$, the known state, $\mathcal{O}$, and the state that can potentially become known in future, as a result of scope extrusion, $\mathcal{H}$.

As before, 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 effective 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 $|\Delta_{\mathcal{L}}|$
- the hidden state, $\Sigma(\Delta)_{\mathcal{H}}$, is simply the empty set $\emptyset$, since Δ does not encode any live locations inaccessible to the observer.

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

- $\Delta(\Sigma)_{\mathcal{N}}$ is inherited directly from Σ.
- $\Delta(\Sigma)_{\mathcal{L}}$ 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 not longer distinguishes between accessible and inaccessible states. Whenever we wish to forget about such distinctions in Σ, we can transform Σ into the $\Sigma(\Delta(\Sigma))$; this we abbreviate to $\uparrow(\Sigma)$. For a discussion on how effective network representations allow us to accommodate the observers view, as discussed in Example 13, see Example 21 below.

We need to generalise the notation developed on page 7 for standard network representations Δ to these effective representations Σ; these were judgements relating to the system view of the network. In addition, since Σ also describes the observer partial-view, we define similar notation for this restricted view, and denote such judgements by $\vdash_{\mathsf{O}}$. Extracting information from effective networks is straightforward:

Definition 16 (System/Partial-View Information Extraction from Effective Networks)

- $\Sigma \vdash l : \mathbf{alive}$ *whenever* $\Delta(\Sigma) \vdash l : \mathbf{alive}$.
- $\Sigma \vdash l \leftrightarrow k$ *whenever* $\Delta(\Sigma) \vdash l \leftrightarrow k$
- $\Sigma \vdash l \leftrightsquigarrow k$ *if* $\Sigma \vdash l \leftrightarrow k$ *and* $\Sigma \vdash l, k : \mathbf{alive}$.
- $\Sigma \vdash_{\mathsf{O}} l : \mathbf{alive}$ *whenever* $\Sigma_{O} \vdash l : \mathbf{alive}$
- $\Sigma \vdash_{\mathsf{O}} l \leftrightarrow k$ *whenever* $\Sigma_{O} \vdash l \leftrightarrow k$
- $\Sigma \vdash_{\mathsf{O}} O$ *whenever* $\begin{cases} O = l[\![P]\!] & \textit{and}\ \Sigma \vdash_{\mathsf{O}} l : \mathbf{alive},\ \mathbf{fn}(P) \subseteq \Sigma_{\mathcal{N}} \\ O = (\nu n : \mathtt{T})M & \textit{and}\ \mathbf{fn}(\mathtt{T}) \subseteq \mathbf{dom}(\Sigma_{O}),\ \Sigma + n : \mathtt{T} \vdash_{\mathsf{O}} M \\ O = M|N & \textit{and}\ \Sigma \vdash_{\mathsf{O}} M,\ \Sigma \vdash_{\mathsf{O}} N \end{cases}$

It is worth pointing out that $\Sigma \vdash l \leftrightarrow k$ if and only if $\Sigma \vdash k \leftrightsquigarrow l$, because Σ is an effective linkset and thus only records links between live locations, as stated earlier in Definition 14. Definition 16 also extends Definition 7 (valid observers) to effective networks, denoted as $\Sigma \vdash_{\mathsf{O}} O$: even though valid observers may still use any known name in $\Sigma_{\mathcal{N}}$ as before, they can now only be located at accessible locations, that is $\Sigma \vdash_{\mathsf{O}} l : \mathbf{alive}$, and define new locations that are only connected to accessible locations, that is $\mathbf{fn}(\mathtt{T}) \subseteq \mathbf{dom}(\Sigma_{O})$, so as to respect the partial view.

We can not rely on $\Delta(\Sigma)$ in order to define how the effective network representation Σ is updated. This has to be done directly; modifying the liveness conditions is straightforward:

- $\Sigma - l = \langle \Sigma_{\mathcal{N}},\ \Sigma_{O} \setminus \{\langle l,k\rangle, \langle k,l\rangle \,|\, k \in \mathbf{dom}(\Sigma_{O})\},\ \Sigma_{\mathcal{L}} \setminus \{\langle l,k\rangle, \langle k,l\rangle \,|\, k \in \mathbf{dom}(\Sigma_{\mathcal{H}})\}\rangle$
- $\Sigma - l \leftrightarrow k = \langle \Sigma_{\mathcal{N}},\ \Sigma_{O} \setminus \{\langle l,k\rangle,\ \langle k,l\rangle\},\ \Sigma_{\mathcal{H}} \setminus \{\langle l,k\rangle,\ \langle k,l\rangle\}\rangle$

However augmenting effective representations is non-trivial, at least in the case of adding a new live location. In Σ the observable links Σ_{O} and the hidden links $\Sigma_{\mathcal{H}}$ are separate sets. But suppose we wish to augment Σ with a new location l at type $\mathtt{loc[a}, \{k_1, \ldots k_n\}]$, which we will denote by $\Sigma + l : \mathtt{loc[a}, \{k_1, \ldots k_n\}]$. The difficulty arises if there is some k_i which is observable, that is in $\mathbf{dom}(\Sigma_O)$, and some other k_j in $\mathbf{dom}(\Sigma_{\mathcal{H}})$. In this case any unobservable links involving k_j, or accessible from k_j, now become observable, via the newly observable link between k_i and l, and thence via the link from l to k_j.

It will be convenient to first define a function which returns the set of new links which have to be added to the observable component of Σ when a fresh name is added.

Definition 17 (Link types) *If Σ is an effective network representation, and n is fresh to Σ, let $\mathsf{lnk}_O(n : \mathtt{T}, \Sigma)$ be defined by the following two clauses:*

- $\mathsf{lnk}_O(l : \mathtt{loc[a}, C\mathtt{]}, \Sigma) = \begin{cases} \{\langle l, k\rangle \mid k \in C\} \cup \{\langle l, l\rangle\} \cup \bigcup_{k \in \mathtt{C}} [\, k\,]_{\Sigma_{\mathcal{H}}} & \textit{if } \mathtt{C} \cap \mathbf{dom}(\Sigma_O) \neq \emptyset \\ \emptyset & \textit{if } \mathtt{C} \cap \mathbf{dom}(\Sigma_O) = \emptyset \end{cases}$
- $\mathsf{lnk}_O(n : \mathtt{T}, \Sigma) = \emptyset$, *otherwise*

In most cases this is actually empty, but in all cases it returns a linkset which is a component*, as defined on page 6; that is a linkset in which all nodes are mutually accessible.*

With this function we can now define how to augment an effective network representation. In the following definitions we assume a and l are *fresh* to Σ:

- $\Sigma + a : \mathtt{ch} \;=\; \langle \Sigma_{\mathcal{N}} \cup \{a\},\ \Sigma_O,\ \Sigma_{\mathcal{H}}\rangle$
- $\Sigma + l : \mathtt{loc[d, C]} \;=\; \langle \Sigma_{\mathcal{N}} \cup \{l\},\ \Sigma_O,\ \Sigma_{\mathcal{H}}\rangle$
- $\Sigma + l : \mathtt{loc[a, C]} \;=\; \begin{cases} \langle \Sigma_{\mathcal{N}} \cup \{l\},\ \Sigma_O,\ \mathcal{H}_1\rangle & \text{if } \mathtt{C} \cap \mathbf{dom}(\Sigma_O) = \emptyset \\ \langle \Sigma_{\mathcal{N}} \cup \{l\},\ O_2,\ \mathcal{H}_2\rangle & \text{if } \mathtt{C} \cap \mathbf{dom}(\Sigma_O) \neq \emptyset \end{cases}$

$$\begin{aligned} \text{where } \mathcal{H}_1 &= \Sigma_{\mathcal{H}} \cup \{\langle l, k\rangle \mid k \in \mathtt{C}\} \cup \{\langle l, l\rangle\} \\ O_2 &= \Sigma_O \cup \mathsf{lnk}_O(l : \mathtt{loc[a, C]}, \Sigma) \\ \mathcal{H}_2 &= \Sigma_{\mathcal{H}} \setminus \mathsf{lnk}_O(l : \mathtt{loc[a, C]}, \Sigma) \end{aligned}$$

Here the fresh live location l is added to either Σ_O or $\Sigma_{\mathcal{H}}$ depending on its links. If it is not linked to any observable location, $\mathtt{C} \cap \mathbf{dom}(\Sigma_O) = \emptyset$, 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, $\mathtt{C} \cap \mathbf{dom}(\Sigma_O) \neq \emptyset$, then it becomes observable as well. Moreover, in this case we have to make observable all previously hidden links now made accessible by the fact that l becomes observable, as we have explained above; these links, $\mathsf{lnk}_O(l : \mathtt{loc[a, C]}, \Sigma)$, have to be transferred from $\Sigma_{\mathcal{H}}$ to Σ_O. The following example elucidates this operation for extending effective networks.

Example 18 (Effective Networks and Partial Views) *Consider the effective net-*

work Σ, with six locations $l, k_1, \ldots, k_5$:

$$\Sigma \;=\; \left\langle \begin{array}{l} \overbrace{\{l, k_1, k_2, k_3, k_4, k_5\}}^{\mathcal{N}},\ \overbrace{\{\langle l, l\rangle\}}^{\mathcal{O}}, \\ \underbrace{\{\langle k_1, k_1\rangle, \langle k_2, k_2\rangle, \langle k_3, k_3\rangle, \langle k_1, k_2\rangle, \langle k_2, k_3\rangle, \langle k_4, k_4\rangle\}}_{\mathcal{H}} \end{array} \right\rangle$$

According to Definition 15, l is the only observable location by the context; locations $k_1, \ldots, k_4$ are alive but not reachable from any observable location while the remaining location, k_5, is dead since it is not in $\mathbf{dom}(\Sigma_{\mathcal{O}} \cup \Sigma_{\mathcal{H}})$*. Moreover, the linkset representing the hidden state, $\Sigma_{\mathcal{H}}$, can be partitioned into two components, $\mathcal{K}_1 = \{\langle k_1, k_1\rangle, \langle k_2, k_2\rangle, \langle k_3, k_3\rangle, \langle k_1, k_2\rangle, \langle k_2, k_3\rangle\}$ and $\mathcal{K}_2 = \{\langle k_4, k_4\rangle\}$ whereas the linkset representing the observable state, $\Sigma_{\mathcal{O}}$, consists of one component, $\{\langle l, l\rangle\}$.*

The operation $\Sigma + k_0 : \mathtt{loc}[\mathsf{a}, \{l\}]$ makes the fresh location, k_0, observable in the resulting effective network, since it is linked to (thus reachable from) the observable location l. The operations $\Sigma + k_0 : \mathtt{loc}[\mathsf{a}, \emptyset]$ and $\Sigma + k_0 : \mathtt{loc}[\mathsf{a}, \{k_1\}]$ both make k_0 hidden in the resulting network because in both cases k_0 is not linked to any observable nodes: in $\Sigma + k_0 : \mathtt{loc}[\mathsf{a}, \emptyset]$ k_0 is completely disconnected whereas in $\Sigma + k_0 : \mathtt{loc}[\mathsf{a}, \{k_1\}]$ k_0 is only linked to the hidden node k_1.

Finally, the operation $\Sigma + k_0 : \mathtt{loc}[\mathsf{a}, \{l, k_1\}]$ affects both $\Sigma_{\mathcal{O}}$ and $\Sigma_{\mathcal{H}}$. This means that k_0 itself becomes observable, since it is linked to l; but as a side effect, the hidden components reachable through it, that is $[\, k_1 \,]_{\mathcal{L}} = \mathcal{K}_1$, becomes observable as well; here $\mathsf{lnk}_{\mathcal{O}}(k_0 : \mathtt{loc}[\mathsf{a}, \{l, k_1\}], \Sigma)$ consists of $\{\langle k_0, l\rangle,\ \langle k_0, k_1\rangle\} \cup \{k_0, k_0\} \cup \mathcal{K}_1$.

Thus, according to the above definition, this updated network translates to:

$$\Sigma + k_0 : \mathtt{loc}[\mathsf{a}, \{l, k_1\}] \;=\; \langle \Sigma_{\mathcal{N}} \cup \{k_0\},\ \Sigma_{\mathcal{O}} \cup (\{\langle k_0, l\rangle,\ \langle k_0, k_1\rangle\}) \cup \mathcal{K}_1,\ \Sigma_{\mathcal{H}} \setminus \mathcal{K}_1 \rangle$$

$$= \left\langle \begin{array}{l} \overbrace{\{l, k_1, k_2, k_3, k_4, k_5, k_0\}}^{\mathcal{N}}, \\ \underbrace{\{\langle l, l\rangle, \langle k_1, k_1\rangle, \langle k_2, k_2\rangle, \langle k_3, k_3\rangle, \langle k_0, k_0\rangle, \langle l, k_0\rangle, \langle k_0, k_1\rangle, \langle k_1, k_2\rangle, \langle k_2, k_3\rangle\}}_{\mathcal{O}}, \underbrace{\{\langle k_4, k_4\rangle\}}_{\mathcal{H}} \end{array} \right\rangle$$

Definition 19 (Effective Configuration) *A system M subject to an effective network Σ is said to be an effective configurations iff* $\mathbf{fn}(M) \subseteq \Sigma_{\mathcal{N}}$.

As stated earlier, in the configuration $\Sigma \triangleright M$ only the information in $\Sigma_{\mathcal{N}}$ and $\Sigma_{\mathcal{O}}$ is available to an external observer, while the extra information in $\Sigma_{\mathcal{H}}$ is only available internally to the system M.

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

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

(l-out)

$$\frac{}{\Sigma \triangleright l[\![a!\langle V\rangle.P]\!] \xrightarrow{l:a!\langle V\rangle} \Sigma \triangleright l[\![P]\!]}\ \Sigma \vdash_{\mathcal{O}} l : \textbf{alive}$$

(l-fork)

$$\frac{}{\Sigma \triangleright l[\![P \mid Q]\!] \xrightarrow{\tau} \Sigma \triangleright l[\![P]\!] \mid l[\![Q]\!]}$$

(l-in)

$$\frac{}{\Sigma \triangleright l[\![a?(X).P]\!] \xrightarrow{l:a?(V)} \Sigma \triangleright l[\![P\{^{V}\!/\!_{X}\}]\!]}\ \Sigma \vdash_{\mathcal{O}} l : \textbf{alive},\ V \subseteq \Sigma_{\mathcal{N}}$$

(l-in-rep)

$$\frac{}{\Sigma \triangleright l[\![*a?(X).P]\!] \xrightarrow{\tau} \Sigma \triangleright l[\![a?(X).(P \mid *a?(X).P)]\!]}$$

(l-eq)

$$\frac{}{\Sigma \triangleright l[\![\mathsf{if}\ u = u\ \mathsf{then}\ P\ \mathsf{else}\ Q]\!] \xrightarrow{\tau} \Sigma \triangleright l[\![P]\!]}$$

(l-neq)

$$\frac{}{\Sigma \triangleright l[\![\mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q]\!] \xrightarrow{\tau} \Sigma \triangleright l[\![Q]\!]}\ u \neq v$$

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

$$\Sigma \triangleright M \xrightarrow{\mu} \Sigma' \triangleright M' \qquad (6)$$

where μ can be an internal action, τ, an input action, $(\tilde{n} : \tilde{\mathrm{T}})l : a?(V)$ or an output, $(\tilde{n} : \tilde{\mathrm{T}})l : a!\langle V\rangle$, adopted from [19,18]; we also have the novel labels, $\mathsf{kill} : l$ and $l \not\leftrightarrow k$, denoting external location killing and link breaking respectively.

The transitions between effective configurations (6) are determined by the rules and axioms given in Table 6, Table 7 and Table 8. Most of the τ-transition rules in Table 6 and Table 7 are inherited directly from their counterpart reduction rules in Table 3; notice that instances such as (l-go) and (l-ping) etc. make use of the entire state of Σ, precisely because they are internal transitions. However transitions which seek to capture the interaction with an observer can only make use of the (partial-view) *observable* information in Σ, that is $\tilde{n} \subseteq \Sigma_{\mathcal{N}}$, $\Sigma \vdash_{\mathcal{O}} l : \textbf{alive}$ and $\Sigma \vdash_{\mathcal{O}} l \leftrightarrow k$. For instance, the new rule (l-halt), for the killing of a location by an observer, is subject to the side-condition that the location liveness is observable by the context. There is a similar constraint on (l-disc), the action corresponding to an observer injecting a link fault in the system. Similarly, in (l-out) and (l-in), which describe data exchange with the observer, there are constraints on the values exchanged and the location at which the exchange takes place; these reflect the valid observer constraints specified earlier in Definition 16.

Table 7. *Network Operational Rules(2) for DπF*

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

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

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

(l-halt)
$$\frac{}{\Sigma \triangleright N \xrightarrow{\mathsf{kill}:l} (\Sigma - l) \triangleright N}\ \Sigma \vdash_{\mathcal{O}} l : \textbf{alive}$$

(l-disc)
$$\frac{}{\Sigma \triangleright N \xrightarrow{l \leftrightarrow k} \Sigma - (l \leftrightarrow k) \triangleright N}\ \Sigma \vdash_{\mathcal{O}} l \leftrightarrow k, l \neq k$$

(l-go)
$$\frac{}{\Sigma \triangleright l[\![\mathsf{go}\ k.P]\!] \xrightarrow{\tau} \Sigma \triangleright k[\![P]\!]}\ \Sigma \vdash k \leftrightsquigarrow l$$

(l-ping)
$$\frac{}{\Sigma \triangleright l[\![\mathsf{ping}\ k.P\ \mathsf{else}\ Q]\!] \xrightarrow{\tau} \Sigma \triangleright l[\![P]\!]}\ \Sigma \vdash k \leftrightsquigarrow l$$

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

(l-nping)
$$\frac{}{\Sigma \triangleright l[\![\mathsf{ping}\ k.P\ \mathsf{else}\ Q]\!] \xrightarrow{\tau} \Sigma \triangleright l[\![Q]\!]}\ \Sigma \nvdash k \leftrightsquigarrow l$$

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

(l-newl)
$$\frac{}{\Sigma \triangleright l[\![(\nu\, k : \mathtt{loc}[\mathsf{a}, \mathsf{C}])\, P]\!] \xrightarrow{\tau} \Sigma \triangleright (\nu\, k : \mathtt{loc}[\mathsf{a}, \mathsf{D}])\, l[\![P]\!]}\ \mathsf{D} = \{m \in \mathsf{C} \cup \{l\} \mid \Sigma \vdash l \leftrightsquigarrow^{*} m\}$$

The more challenging rules are found in Table 8: they are adaptations of the standard rules for actions-in-context from [19], extended 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 $\mathtt{T} \setminus \Sigma_{\mathcal{D}}$ defined as expected:

$$\mathsf{ch} \setminus \{l_1, \dots, l_n\} = \mathsf{ch} \qquad \mathtt{loc}[\mathsf{S}, \mathsf{C}] \setminus \{l_1, \dots, l_n\} = \mathtt{loc}[\mathsf{S}, \mathsf{C} \setminus \{l_1, \dots, l_n\}]$$

Recall that $\Sigma_{\mathcal{D}}$ is the set of dead locations in Σ. A side condition is added to (l-weak), $\mathbf{fn}(U) \subseteq (\mathbf{dom}(\Sigma_{\mathcal{O}}) \cup \{\tilde{n}\})$, 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; once again this reflects the restriction on valid observers specified earlier in Definition 16.

Table 8. *Contextual Operational Rules(3) for DπF*

(l-open)

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

(l-weak)

$$\frac{\Sigma + n : \mathtt{U} \triangleright N \xrightarrow{(\tilde{n}:\tilde{\mathtt{T}})l:a?(V)} \Sigma' \triangleright N'}{\Sigma \triangleright N \xrightarrow{(n:\mathtt{U},\tilde{n}:\tilde{\mathtt{T}})l:a?(V)} \Sigma' \triangleright N'} \; n \in V \setminus \{l, a\},\ \mathbf{fn}(U) \subseteq (\mathbf{dom}(\Sigma_{O}) \cup \{\tilde{n}\})$$

(l-rest-typ)

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

(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-par-ctxt)

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

(l-par-comm)

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

The internal communication rule (l-par-comm) also contains subtleties: communication is defined in terms of the system view ($\uparrow(\Sigma)$) rather than the observer view dictated by Σ; the intuition is that internal communication can still occur, even at locations that the observer cannot access. Thus the premises are defined in terms of the ability to output and input of systems with respect to the maximal observer view, $\uparrow(\Sigma)$. The rules (l-rest) and (l-par-ctxt) should be relatively straightforward. Finally, (l-rest-typ) is a completely novel rule which filter any links exported in location types when the other endpoint of the link is still scoped. For brevity, the premise of this rule exploits the fact that the network $\Sigma + k_1 : \mathtt{loc}[\mathsf{S}_1, \mathsf{C}_1] + k_2 : \mathtt{loc}[\mathsf{S}_2, \mathsf{C}_2]$ can be also expressed as $\Sigma + k_2 : \mathtt{loc}[\mathsf{S}_2, \mathsf{C}_2 \setminus \{k_1\}] + k_1 : \mathtt{loc}[\mathsf{S}_1, \mathsf{C}_1 \cup \{k_2\}]$ whenever $k_1 \in \mathsf{C}_2$. The utility of (l-rest-typ) is illustrated in Example 21.

Example 20 (Scope Extruding Network Information) *Consider the extended network Σ where l and k_1 are accessible by the context, that is $\Sigma \vdash_O l : \textbf{alive},\ k_1 : \textbf{alive}$. For the effective configuration $\Sigma \triangleright (\nu\, k_2 : \{k_1\})l[\![a!\langle k_2\rangle]\!]$ we can derive the transition*

$$\Sigma \triangleright (\nu\, k_2 : \{k_1\})l[\![a!\langle k_2\rangle]\!] \xrightarrow{(k_2:\{k_1\})l:a!\langle k_2\rangle} \Sigma + k_2 : \{k_1\} \triangleright l[\![\mathbf{0}]\!]$$

using (l-out) *and* (l-open)*. The transition label denotes the scope extrusion of the fresh location k_2 with the information that it is connected to k_1. We can also have the sequence of transitions*

$$\begin{aligned} &\Sigma \triangleright (\nu\, k_2 : \{k_1\})l[\![a!\langle k_2\rangle]\!] \xrightarrow{\mathsf{kill}:k_1} \\ &\qquad \Sigma - k_1 \triangleright (\nu\, k_2 : \{k_1\})l[\![a!\langle k_2\rangle]\!] \xrightarrow{(k_2:\emptyset)l:a!\langle k_2\rangle} \\ &\qquad\qquad (\Sigma - k_1) + k_2 : \emptyset \triangleright (\nu\, k_2 : \{k_1\})l[\![a!\langle k_2\rangle]\!] \end{aligned}$$

whereby the context first kills k_1 before performing the input on a at l. The second transition is still derived using (l-out) *and* (l-open) *but, this time ,the side-condition $\mathtt{U} = \mathtt{T} \setminus \Sigma_{\mathcal{D}}$ of* (l-open) *ensures that we scope extrude k_2 with different information, stating that it is completely disconnected, thus inaccessible (hidden). This transition captures the fact that even though the link between k_2 and k_1 still exists, this information cannot be discovered and used by the context to access k_2. In fact, k_2 is added to Σ_O in the resulting configuration of the first case of scope extrusion but added to $(\Sigma - k_1)_{\mathcal{H}}$ in the second case.*

Example 21 *Let us revisit Example 13 to see the effect of the observer O on M_1; this observer runs 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.*

If we consider a sub-derivation where k_1 is not restricted, then an application of (l-out) *and* (l-par-ctxt)*, followed by two applications of* (l-open) *gives*

$$\Sigma_l + k_1 : \{l\} \triangleright M_1' \xrightarrow{\alpha} \Sigma_l + k_1 : \{l\} + k_2 : \{k_1\} + k_3 : \{k_1, k_2\} \triangleright l[\![P]\!] \,|\, k_2[\![Q]\!] \tag{7}$$

where M_1' is $(\nu\, k_2 : \{k_1\})(\nu\, k_3 : \{k_1, k_2\})l[\![a!\langle k_2, k_3\rangle.P]\!] \,|\, k_2[\![Q]\!]$ 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 (7) can be re-arranged to read*

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

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*

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

where β is the action $(k_2 : \emptyset, k_3 : \{k_2\})l : a!\langle k_2, k_3\rangle$; that is β is filtered of any occurrence of k_1 in its bound types.

Note that the residual network representation, $\Sigma_l{+}k_2 : \emptyset{+}k_3 : \{k_2\}$ describes partial-view network, with a hidden part that is not available to the observer. Eliding any channel names, the network evaluates to

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

where the liveness of k_2 and k_2 and the connection between them is hidden. This effective network may be represented diagrammatically as:

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 lts actions. This new framework can be used to establish positive results. For example, for $\Sigma_{l,k} = \langle\{a, l, k\}, \{\langle l, l\rangle, \langle k, k\rangle, \langle l, k\rangle\}, \emptyset\rangle$, one can prove

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

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

$$\mathcal{R} = \left\{\begin{array}{lll}
\langle \Sigma_{l,k} \triangleright M & , \Sigma_{l,k} \triangleright N\rangle & \mid \langle M, N\rangle \in \mathcal{R}_{sys} \\
\langle \Sigma_{l,k} - l \triangleright M & , \Sigma_{l,k} - l \triangleright N\rangle & \mid \langle M, N\rangle \in \mathcal{R}^1_{sys} \\
\langle \Sigma_{l,k} - k \triangleright M & , \Sigma_{l,k} - k \triangleright N\rangle & \mid \langle M, N\rangle \in \mathcal{R}^2_{sys} \\
\langle \Sigma_{l,k} - l \leftrightarrow k \triangleright M & , \Sigma_{l,k} - l \leftrightarrow k \triangleright N\rangle & \mid \langle M, N\rangle \in \mathcal{R}^3_{sys} \\
\langle \Sigma_{l,k} - l, l \leftrightarrow k \triangleright M & , \Sigma_{l,k} - l, l \leftrightarrow k \triangleright N\rangle & \mid \langle M, N\rangle \in \mathcal{R}^3_{sys} \\
\langle \Sigma_{l,k} - k, l \leftrightarrow k \triangleright M & , \Sigma_{l,k} - k, l \leftrightarrow k \triangleright N\rangle & \mid \langle M, N\rangle \in \mathcal{R}^3_{sys} \\
\langle \Sigma_{l,k} - l, k \triangleright M & , \Sigma_{l,k} - l, k \triangleright N\rangle & \mid \langle M, N\rangle \in \mathcal{R}^3_{sys} \\
\langle \Sigma_{l,k} - l, k, l \leftrightarrow k \triangleright M & , \Sigma_{l,k} - l, k, l \leftrightarrow k \triangleright N\rangle & \mid \langle M, N\rangle \in \mathcal{R}^3_{sys}
\end{array}\right\}$$

where

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

$$\mathcal{R}^1_{sys} = \mathcal{R}_{sys} \cup \{\langle l[\![\mathsf{ping}\; k.\, a!\langle\rangle \;\mathsf{else}\; \mathbf{0}]\!],\; l[\![\mathbf{0}]\!]\rangle\}$$

$$\mathcal{R}^2_{sys} = \mathcal{R}_{sys} \cup \{\langle l[\![\mathbf{0}]\!],\; k[\![\mathsf{go}\; l.a!\langle\rangle]\!]\rangle\}$$

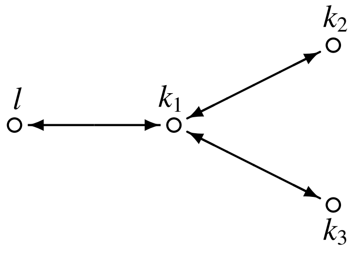
$$\mathcal{R}^3_{sys} = \mathcal{R}_{sys} \cup \left\{ \langle l[\![\mathbf{0}]\!],\; k[\![\mathsf{go}\; l.a!\langle\rangle]\!]\rangle,\; \langle l[\![\mathsf{ping}\; k.\, a!\langle\rangle \;\mathsf{else}\; \mathbf{0}]\!],\; l[\![\mathbf{0}]\!]\rangle \right\}$$

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 impossible to observe. Problems arise because the current labels contain information relating to the *hidden part* of an effective network, which is not observable by valid contexts.

Example 22 *Let us consider a slight variation on the system M_1 used in Example 13 and Example 21:*

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

again running on the (effective) network $\Sigma_l = \langle\{l, a\}, \{\langle l, l\rangle\}, \emptyset\rangle$. Note that the code $l[\![a!\langle k_2, k_3\rangle.P]\!] \,|\, k_2[\![R]\!]$ is now effectively running on the following implicit network,



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

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

This is not because $k_2[\![Q]\!]$ in M_1 and $k_2[\![R]\!]$ in M_2 may be different - the condition $\Sigma \vdash_O l$: **alive** *in* (l-out) *and* (l-in) *of Table 6 prohibits input or output labels at hidden locations - but rather 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; as we have seen, in $\Sigma_l \triangleright M_1$ the output label is $\beta = (k_2 : \emptyset,\, k_3 : \{k_2\})\, l : a!\langle k_2, k_3\rangle$ while with $\Sigma_l \triangleright M_2$ it is $\beta' = (k_2 : \emptyset,\, k_3 : \emptyset) l : a!\langle k_2, k_3\rangle$. More specifically,* *there is a difference in the type associated to the scope extruded location k_3; in the first label, the scope extruded k_3 is linked to the newly scope extruded k_2 whereas in the second label it is not.*

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

$$\Sigma_l \models M_1 \;\cong\; M_2$$

The extra label information relating to the hidden part of the network is also indistinguishable from information relating to dead nodes, as is shown in the following example.

Example 23 *Let us reconsider the three configurations $\Sigma_l \triangleright N_i$ for $i = 1, 2, 3$ from Example 12 where $\Sigma_l = \Sigma(\Delta_l) = \langle\{l, a\}, \{\langle l, l\rangle\}, \emptyset\rangle$. 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:*

$$\begin{aligned}
\Sigma_l \triangleright N_1 &\xrightarrow{(k{:}\mathtt{loc}[\mathtt{d},\{l\}])l{:}a!\langle k\rangle} \langle\{l, k\}, \{\langle l, l\rangle\}, \emptyset\rangle \triangleright l[\![\mathbf{0}]\!]\\
\Sigma_l \triangleright N_2 &\xrightarrow{(k{:}\mathtt{loc}[\mathtt{d},\emptyset])l{:}a!\langle k\rangle} \langle\{l, k\}, \{\langle l, l\rangle\}, \emptyset\rangle \triangleright l[\![\mathbf{0}]\!]\\
\Sigma_l \triangleright N_3 &\xrightarrow{(k{:}\mathtt{loc}[\mathtt{a},\emptyset])l{:}a!\langle k\rangle} \langle\{l, k\}, \{\langle l, l\rangle\}, \{\langle k, k\rangle\}\rangle \triangleright l[\![\mathbf{0}]\!]
\end{aligned}$$

More specifically,

- *$\Sigma_l \triangleright N_1$'s transition label states that the scope extruded location k is* dead *and linked to l*
- *$\Sigma_l \triangleright N_2$'s label states that the scope extruded location k is* dead *and completely disconnected*
- *whereas $\Sigma_l \triangleright N_3$'s label states that k is* alive *but completely disconnected, and thus added to the hidden part of Σ_l.*

With these labels, the three configurations will be distinguished by the bisimulation equivalence $\approx_{wrong}$ whereas, according to $\cong$ we have

$$\Sigma_l \;\models\; N_1 \;\cong\; N_2 \;\cong\; N_3$$

since no observer would be able to distinguish the difference in the state of k.

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.

4 A bisimulation equivalence for DπF

We first outline the revision to our labelled transitions. Currently, the actions of these transitions use types of the form $\mathtt{T} = \mathtt{ch}$ or $\mathtt{loc}[\mathtt{S}, \{k_1, \ldots k_n\}]$, where the latter indicates the liveness of a location and the nodes k_i to which it is directly linked. We change these to new types of the form

$$\mathtt{L}, \mathtt{K} = \{\langle l_1, k_1\rangle, \ldots, \langle l_i, k_i\rangle\}$$

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 a 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. This means that our labels now describe information at the level of *accessibility paths* between locations as opposed to direct connections only. We have already developed the necessary technology to define these new types, in Definition 17.

Definition 24 (A (derived) labelled transition system for DπF) *This consists of a collection of transitions $\Sigma \triangleright N \overset{\mu}{\longmapsto} \Sigma' \triangleright N'$, 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) - $l \nleftrightarrow k$*

The transitions in the derived lts for DπF are defined in Table 9. The rules (l-deriv-2) and (l-deriv-3) transform the types of bound names using the function $\mathsf{lnk}_O(\tilde{n} : \tilde{\mathtt{T}}, \Sigma)$; this is simply a version of the function defined in Definition 17 to deal with sequences of type declarations:

$$\mathsf{lnk}_O((n, \tilde{n}):(\mathtt{T}, \tilde{\mathtt{T}}), \Sigma) \quad = \quad \mathsf{lnk}_O(n:\mathtt{T}, \Sigma), \mathsf{lnk}_O(\tilde{n}:\tilde{\mathtt{T}}, (\Sigma + n : \mathtt{T}))$$

These revised transitions give rise to a new *(weak) bisimulation equivalence* over configurations, $\approx$, defined in the usual way, but based on *derived actions*. Our definition uses the standard notation for weak actions, namely $\overset{\mu}{\Longmapsto}$ denotes $\left(\overset{\tau}{\longmapsto}^*\right) \overset{\mu}{\longrightarrow} \left(\overset{\tau}{\longmapsto}^*\right)$, and $\overset{\widehat{\mu}}{\Longmapsto}$ denotes

- $\overset{\tau}{\longmapsto}^*$ if $\mu = \tau$
- $\overset{\mu}{\Longmapsto}$ otherwise.

Definition 25 (Weak bisimulation equivalence) *This is denoted as $\approx$ and defined as the largest relation over configurations such that if $\Sigma_M \triangleright M \approx \Sigma_N \triangleright N$ then*

Table 9. *The derived lts for DπF*

(l-deriv-1)

$$\frac{\Sigma \triangleright N \xrightarrow{\mu} \Sigma' \triangleright N'}{\Sigma \triangleright N \overset{\mu}{\longmapsto} \Sigma' \triangleright N'}\ \mu \in \{\tau, \mathsf{kill} : l, l \leftrightarrow\!\!\!\!/\ k\}$$

(l-deriv-2)

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

(l-deriv-3)

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

- $\Sigma_M \triangleright M \overset{\mu}{\longmapsto} \Sigma'_M \triangleright M'$ *implies* $\Sigma_N \triangleright N \overset{\hat{\mu}}{\Longmapsto} \Sigma'_N \triangleright N'$ *such that* $\Sigma'_M \triangleright M' \approx \Sigma'_N \triangleright N'$
- $\Sigma_N \triangleright N \overset{\mu}{\longmapsto} \Sigma'_N \triangleright N'$ *implies* $\Sigma_M \triangleright M \overset{\hat{\mu}}{\Longmapsto} \Sigma'_M \triangleright M'$ *such that* $\Sigma'_M \triangleright M' \approx \Sigma'_N \triangleright N'$

Example 26 *Here we re-examine the systems in Example 22 and Example 23. We recall that in Example 22 we had the following labelled transitions with respect to the original lts:*

$$\Sigma_l \triangleright M_1 \xrightarrow{\mu_1} \Sigma + k_2 : \emptyset + k_3 : \{k_2\} \triangleright (\nu\, k_1 : \{l, k_2, k_3\})\, l[\![P]\!] \,|\, k_2[\![Q]\!]$$
$$\Sigma_l \triangleright M_2 \xrightarrow{\mu_2} \Sigma + k_2 : \emptyset + k_3 : \emptyset \triangleright (\nu\, k_1 : \{l, k_2, k_3\})\, l[\![P]\!] \,|\, k_2[\![R]\!]$$

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* $\mathsf{lnk}_O(k_2 : \emptyset, \Sigma)$ *and* $\mathsf{lnk}_O(k_3 : \{k_2\}, (\Sigma + k_2 : \emptyset))$ *both lead to the empty link set.*

Formally, we get the derived action

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

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*:*

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

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

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

Since k_1 cannot ever be scope extruded to the observer, we are guaranteed that the state of k_2 and k_3 together with the code located at these locations, that is $k_2[\![Q]\!]$ and $k_2[\![R]\!]$, are forever inaccessible to the observer. This means that we can match any τ-move by $k_2[\![Q]\!]$ on the left hand side by the empty move (and viceversa for $k_2[\![R]\!]$) and any move by $l[\![P]\!]$ on either side by that same identical move.

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

$$((\Sigma + k_2 : \emptyset) + k_3 : \{k_2\}) \triangleright (\nu\, k_1 : \{l, k_2, k_3\}) l[\![P]\!] \,|\, k_2[\![Q]\!] \;\xmapsto{\alpha_1}\; \ldots$$
$$((\Sigma + k_2 : \emptyset) + k_3 : \emptyset) \triangleright (\nu\, k_1 : \{l, k_2, k_3\}) l[\![P]\!] \,|\, k_2[\![R]\!] \;\xmapsto{\alpha_2}\; \ldots$$

where

$$\alpha_1 \text{ is } (k_1 : \{k_1 \leftrightarrow k_2, k_1 \leftrightarrow k_3, k_2 \leftrightarrow k_3\}) l : a!\langle k_1\rangle$$
$$\alpha_2 \text{ is } (k_1 : \{k_1 \leftrightarrow k_2, k_1 \leftrightarrow k_3\}) l : a!\langle k_1\rangle$$

We note that the link type associated with β_1 includes the additional component $\{\langle k_2, k_3\rangle\}$, 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 α_1 and α_2 we have

$$\Sigma_l \triangleright M_1 \not\approx \Sigma_l \triangleright M_2$$

In addition, if M_1 and M_2 were running on the same network, say (5), and $k_2[\![Q]\!]$ and $k_2[\![R]\!]$ were different systems, this could be verified after the scope extrusion of k_1: scope extruding k_1 would make k_2 observable, enabling (l-out) *and* (l-in) *to be applied to the code Q and R running at k_2.*

Example 27 *Revisiting Example 23, 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 $\Sigma_l \triangleright N_i \xmapsto{\alpha} \ldots \triangleright l[\![\mathbf{0}]\!]$ for $i = 1, 2, 3$ where α is the label $(k : \emptyset) l : a!\langle k\rangle$. Thus we have*

$$\Sigma_l \triangleright N_i \approx \Sigma_l \triangleright N_j \quad \textit{where } i, j = 1, 2, 3$$

as required.

5 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 reduction barbed congruence (Definition 9). We have already given a translation from a standard configuration into an effective configuration, by simply translating the network representation as

$\Sigma(\Delta)$. We next redefine reduction barbed congruence for effective configurations. The reduction relation over effective configurations

$$\Sigma \triangleright M \longrightarrow \Sigma' \triangleright M'$$

is simply obtained by reusing the rules defined earlier in Tables 2, 3, 4 and 5, substituting Σ for Δ[3]. Note that all the side-conditions used in these rules can also be applied to Σ using the notation developed on page 21.

We refine the notion of a barb for effective configurations, intuitively restricting them to observable locations, that is those in Σ_O.

Definition 28 $\Sigma \triangleright N \Downarrow_{a@l}$ *denotes an* observable barb *exhibited by the configuration* $\Sigma \triangleright N$*, on channel a at location l. Formally, it means that* $\Sigma \triangleright N \longrightarrow^* \Sigma' \triangleright N'$ *for some* $\Sigma' \triangleright N'$ *such that* $N' \equiv (\nu\, \tilde{n} : \tilde{\mathrm{T}})M | l[\![a!\langle V\rangle.Q]\!]$ *where* $l, a \notin \tilde{n}$ *and* $l \in \mathbf{dom}(\Sigma'_O)$*.*

We also extend the definition of contextual relations to effective configurations. The novel aspect of this extended definition is that we now relate configurations with arbitrary effective networks, as long as their observable part, that is Σ_N and Σ_O, coincide. Because of this, valid observers, that is $\Sigma \vdash_{\mathrm{O}} O$, valid network extensions, that is $\mathbf{fn}(\mathrm{T}) \in \mathbf{dom}(\Sigma_O)$, and name freshness, that is $n \notin \Sigma_N$, in the definition below need only be defined with respect to one effective network.

Definition 29 (Contextual relations over effective configurations) *A relation* $\mathcal{R}$ *over effective configurations is* contextual *if:*

(Observable Network equality)

- $\Sigma \triangleright M \,\mathcal{R}\, \Sigma' \triangleright N$ *implies* $\Sigma_N = \Sigma'_N$ *and* $\Sigma_O = \Sigma'_O$

(Parallel Systems)

- $\Sigma \triangleright M \,\mathcal{R}\, \Sigma' \triangleright N$ *and* $\Sigma \vdash_{\mathrm{O}} O$ *implies*
  - $\Sigma \triangleright M|O \,\mathcal{R}\, \Sigma' \triangleright N|O$
  - $\Sigma \triangleright O|M \,\mathcal{R}\, \Sigma' \triangleright O|N$

(Network Extensions)

- $\Sigma \triangleright M \,\mathcal{R}\, \Sigma' \triangleright N$ *and* $\mathbf{fn}(\mathrm{T}) \subseteq \mathbf{dom}(\Sigma_O)$, $n \notin \Sigma_N$ *implies* $\Sigma + n : \mathrm{T} \triangleright M \,\mathcal{R}\, \Sigma' + n : \mathrm{T} \triangleright N$

With these modifications, we extend Definition 9 for *reduction barbed congruence*

[3] In this translation, we only lose the ability to break links with dead endpoints. These reductions were in a sense redundant in the original reduction semantics because these links could not be used for code migration and pinging.

for effective configurations, denoted as

$$\Sigma_M \triangleright M \cong \Sigma_N \triangleright N$$

and defined as the largest relation over effective configurations that is *barb preserving*, *reduction closed* and *contextual*.

Note that this enables us to compare arbitrary configurations, $\Sigma_M \triangleright M$ and $\Sigma_N \triangleright N$, 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 $\Sigma \triangleright M \cong \Sigma \triangleright N$. Then, for example, the notation (3) used in Section 2 can be taken to mean

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

where the effective network $\Sigma(\Delta)$ has no hidden state.

At this point, we are in a position to state the main result of the paper:

Theorem 30 *Suppose $\Sigma \triangleright M$, $\Sigma' \triangleright N$ are effective configurations in DπF such that $\Sigma_N = \Sigma'_N$ and $\Sigma_O = \Sigma'_O$. Then*

$$\Sigma \triangleright M \cong \Sigma' \triangleright N \ \textit{ if and only if } \ \Sigma \triangleright M \approx \Sigma' \triangleright N$$

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

Corollary 31 *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 to show that two configurations are reduction barbed congruent:

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

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

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

For the purposes of these proofs, we restrict the definition of bisimulation, Definition 25, to configurations $\Sigma' \triangleright M$, $\Sigma \triangleright N$ whose network names Σ_N and Σ'_N, denoting the free names known by M and N, are identical. This reasonable assumption restricts us to compare configurations which "know about" the same free names, but this turns out to be in accordance with our definition of reduction barbed congruence. This assumption also suffices to guarantee that when two configurations are bisimilar, their observable network is equivalent (see Proposition 33).

5.1 Soundness

The main task in proving that derived action bisimulation is sound is showing that $\approx$ is contextual.

One aspect worth highlighting about any two bisimilar configurations $\Sigma \triangleright M$, $\Sigma' \triangleright N$ in $\approx$ is that the observable parts of the respective effective networks Σ and Σ', that is $\Sigma_{\mathcal{N}}$,$\Sigma'_{\mathcal{N}}$ and Σ_{O}, Σ'_{O}, coincide (up to our symmetric interpretation of links, that is $l \leftrightarrow k$ is the same as $k \leftrightarrow l$); we show this in Proposition 33. This allows us to smoothly apply the Definition 29, Contextuality, which also requires configurations to have the same observable network. Intuitively, if they did not, one configuration could transition with a label whose form depends on the observable part of the network, such as $\mathsf{kill} : k$ or $l \nleftrightarrow k$ for Σ_O and $l : a?(V)$ for $\Sigma_{\mathcal{N}}$, that the other could not (weakly) match. For this purpose, we find it convenient to

- denote the observable pair $\langle \mathcal{N}, \mathcal{O} \rangle$ in an effective configuration as $\mathcal{I}$.
- refer to the observable part of an effective network Σ as $\mathcal{I}(\Sigma)$.

We start by proving our earlier claim that the observable networks of bisimilar configurations coincide. This proposition uses a lemma stating that there is a special relationship between derived *silent actions* and residual networks: 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.

Lemma 32 *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 33 (Bisimulation and Observable Networks)

$$\textit{If } \Sigma^M \triangleright M \approx \Sigma^N \triangleright N \textit{ then } \mathcal{I}(\Sigma^M) = \mathcal{I}(\Sigma^N).$$

PROOF. We already assume that $\Sigma^M_{\mathcal{N}} = \Sigma^N_{\mathcal{N}}$; we just need to show that $\Sigma^M_{O} = \Sigma^N_{O}$. Assume $\langle l, k \rangle \in \Sigma^M_O$. Since Σ^M_O is an effective linkset, we know also that $\langle l, l \rangle \in \Sigma^M_O$

(and also $\langle k,k\rangle \in \Sigma^M_O$). Thus from Definition 16 we obtain $\Sigma^M \vdash_O l : \textbf{alive}$ and $\Sigma^M \vdash_O l \leftrightarrow k$. From $\Sigma^M \vdash_O l : \textbf{alive}$, (l-halt) and (l-deriv-1) we get

$$\Sigma^M \triangleright M \xmapsto{\mathsf{kill}:l} (\Sigma^M - l) \triangleright M \tag{8}$$

Similarly, from $\Sigma^M \vdash_O l \leftrightarrow k$, (l-disc) and (l-deriv-1) we obtain

$$\Sigma^M \triangleright M \xmapsto{l \nleftrightarrow k} (\Sigma^M - l \leftrightarrow k) \triangleright M \tag{9}$$

From the hypothesis that $\Sigma^M \triangleright M \approx \Sigma^N \triangleright N$, (8) and (9) we get

$$\Sigma^N \triangleright N \overset{\mathsf{kill}:l}{\models\!\!\!\Longrightarrow} \Sigma' \triangleright N' \approx (\Sigma^M - l) \triangleright M \tag{10}$$

$$\Sigma^N \triangleright N \overset{l \nleftrightarrow k}{\models\!\!\!\Longrightarrow} \Sigma' \triangleright N' \approx (\Sigma^M - l \leftrightarrow k) \triangleright M \tag{11}$$

We can expand the weak transition in (10) into

$$\Sigma^N \triangleright N \overset{\widehat{\tau}}{\models\!\!\!\Longrightarrow} \Sigma'' \triangleright N'' \tag{12}$$

$$\Sigma'' \triangleright N'' \xmapsto{\mathsf{kill}:l} \Sigma'' - l \triangleright N'' \tag{13}$$

$$\Sigma'' - l \triangleright N'' \overset{\widehat{\tau}}{\models\!\!\!\Longrightarrow} \Sigma' \triangleright N' \tag{14}$$

and the weak transition in (11) into

$$\Sigma^N \triangleright N \overset{\widehat{\tau}}{\models\!\!\!\Longrightarrow} \Sigma'''' \triangleright N'''' \tag{15}$$

$$\Sigma'''' \triangleright N'''' \xmapsto{l \nleftrightarrow k} \Sigma'''' - l \leftrightarrow k \triangleright N'' \tag{16}$$

$$\Sigma'''' - l \leftrightarrow k \triangleright N'''' \overset{\widehat{\tau}}{\models\!\!\!\Longrightarrow} \Sigma''' \triangleright N''' \tag{17}$$

From (13) and (16) we deduce

$$\Sigma'' \vdash_O l : \textbf{alive} \tag{18}$$

$$\Sigma'''' \vdash_O l \leftrightarrow k \tag{19}$$

Lemma 32 leads us to conclude that (12) could not have resuscitated l in Σ^N and similarly that (15) could not have created a link between l and k in Σ^N. Thus, by (18), (19), (12), (15) and Lemma 32 we conclude that

$$\Sigma^N \vdash_O l \leftrightarrow k \text{ which also implies } \Sigma^N \vdash_O l : \textbf{alive}$$

and thus

$$\langle l,l\rangle \in \Sigma^N_O \qquad \langle l,k\rangle \text{ or } \langle k,l\rangle \in \Sigma^N_O$$

The same argument is used to prove inclusion in the reverse direction.

This contextuality proof for $\approx$ 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 lemmas. The first Lemma states that in our original transition system of Section 3, $\Sigma \triangleright M \xrightarrow{\mu} \Sigma' \triangleright M'$, the reduct effective network Σ' is a function of the redex effective network Σ and the external action μ.

Definition 34 (Action residuals) *The partial function* after *ranges over effective networks* Σ *and external actions* μ*. It returns effective networks, defined as:*

- Σ $\mathsf{after}\,(\tilde{n} : \tilde{\mathrm{T}})l : a!\langle V\rangle$ *is defined as* $\Sigma + \tilde{n} : \tilde{\mathrm{T}}$
- Σ $\mathsf{after}\,(\tilde{n} : \tilde{\mathrm{T}})l : a?(V)$ *is defined as* $\Sigma + \tilde{n} : \tilde{\mathrm{T}}$
- Σ $\mathsf{after\ kill} : l$ *is defined as* $\Sigma - l$
- Σ $\mathsf{after}\ l \not\leftrightarrow k$ *is defined as* $\Sigma - l \leftrightarrow k$

Proposition 35 *If* μ *is an external action and* $\Sigma \triangleright N \xrightarrow{\mu} \Sigma' \triangleright N'$ *then* $\Sigma' = \Sigma$ $\mathsf{after}\,\mu$.

PROOF. A straightforward induction on the inference of $\Sigma \triangleright N \xrightarrow{\mu} \Sigma' \triangleright N'$.

The second lemma below relates actions and pre/post conditions on Σ with the structure of the systems.

Lemma 36 (Actions and Systems)

- *if* $\Sigma \triangleright N \xrightarrow{(\tilde{n}:\tilde{\mathrm{T}})l:a!\langle V\rangle} \Sigma' \triangleright N'$ *then*
  - $\cdot$ $N \equiv (\nu\,\tilde{n} : \tilde{\mathrm{T}}')(\nu\,\tilde{m} : \tilde{\mathrm{U}})M|l[\![a!\langle V\rangle.P]\!]$ *where* $\tilde{\mathrm{T}} = \tilde{\mathrm{T}}' \setminus \Sigma_{\mathcal{D}}$.
  - $\cdot$ $N' \equiv (\nu\,\tilde{m} : \tilde{\mathrm{U}})M|l[\![P]\!]$
- *if* $\Sigma \triangleright N \xrightarrow{(\tilde{n}:\tilde{\mathrm{T}})l:a?(V)} \Sigma' \triangleright N'$ *then*
  - $\cdot$ $N \equiv (\nu\,\tilde{m} : \tilde{\mathrm{U}})M|l[\![a?(X).P]\!]$
  - $\cdot$ $N' \equiv (\nu\,\tilde{m} : \tilde{\mathrm{U}})M|l[\![P\{^{V}\!/\!_{X}\}]\!]$
- *if* $\Sigma \triangleright N \overset{\tau}{\longmapsto} \Sigma' \triangleright N'$ *where* $\Sigma \vdash l : \textbf{alive}$ *and* $\Sigma' \nvdash l : \textbf{alive}$ *then*
  - $\cdot$ $N \equiv N'|l[\![\mathsf{kill}]\!]$
- *if* $\Sigma \triangleright N \overset{\tau}{\longmapsto} \Sigma' \triangleright N'$ *where* $\Sigma \vdash l \leftrightarrow k$ *and* $\Sigma' \nvdash l \leftrightarrow k$ *then*
  - $\cdot$ $N \equiv N'|l[\![\mathsf{break}\ k]\!]$ *or* $N \equiv N'|k[\![\mathsf{break}\ l]\!]$

PROOF. A straightforward induction on the inference of $\Sigma \triangleright N \xrightarrow{(\tilde{n}:\tilde{\mathrm{T}})l:a!\langle V\rangle} \Sigma' \triangleright N'$, $\Sigma \triangleright N \xrightarrow{(\tilde{n}:\tilde{\mathrm{T}})l:a?(V)} \Sigma' \triangleright N'$ and $\Sigma \triangleright N \overset{\tau}{\longmapsto} \Sigma' \triangleright N'$.

The third lemma required by the Composition and Decomposition stems from the use of the function $\uparrow(\Sigma)$ in the inductive hypothesis of the rule (l-par-comm) of Table 8.

Lemma 37 (Input/Output Actions and the Maximal Observer View)

- *If* $\Sigma \triangleright N \xrightarrow{(\tilde{n}:\tilde{\mathsf{T}})l:a!\langle V\rangle} \Sigma' \triangleright N'$ *then* $\uparrow(\Sigma) \triangleright N \xrightarrow{(\tilde{n}:\tilde{\mathsf{T}})l:a!\langle V\rangle} \uparrow(\Sigma') \triangleright N'$.
- *If* $\Sigma \triangleright N \xrightarrow{(\tilde{n}:\tilde{\mathsf{U}})l:a?(V)} \Sigma' \triangleright N'$ *and* $\tilde{\mathsf{U}} = \tilde{\mathsf{T}} \setminus \mathbf{dom}(\Sigma_{\mathcal{H}})$ *then* $\uparrow(\Sigma) \triangleright N \xrightarrow{(\tilde{n}:\tilde{\mathsf{T}})l:a?(V)} \uparrow(\Sigma') \triangleright N'$
.
- *If* $\uparrow(\Sigma) \triangleright N \xrightarrow{(\tilde{n}:\tilde{\mathsf{T}})l:a!\langle V\rangle} \Sigma' \triangleright N'$ *and* $\Sigma \vdash_{O} l\!:\!\mathbf{alive}$ *then* $\Sigma \triangleright N \xrightarrow{(\tilde{n}:\tilde{\mathsf{T}})l:a!\langle V\rangle} \Sigma'' \triangleright N'$.
- *If* $\uparrow(\Sigma) \triangleright N \xrightarrow{(\tilde{n}:\tilde{\mathsf{T}})l:a?(V)} \Sigma' \triangleright N'$ *and* $\Sigma \vdash_{O} l\!:\!\mathbf{alive}$ *then* $\Sigma \triangleright N \xrightarrow{(\tilde{n}:\tilde{\mathsf{U}})l:a?(V)} \Sigma'' \triangleright N'$ *where* $\tilde{\mathsf{U}} = \tilde{\mathsf{T}} \setminus \mathbf{dom}(\Sigma_{\mathcal{H}})$.

PROOF. The proof uses Lemma 36 to infer the structure of N and then proceeds by induction on the structure of N.

The Composition and Decomposition Lemmas cover all the cases of how an action can be composed and decomposed. In our lts we have only one instance where an action can be decomposed into different actions in the premises, namely τ, which can be constructed through interacting (bound) input and (bound) output actions. All other actions cannot be decomposed and are preserved by parallel contexts.

Lemma 38 (Composition)

- *Suppose* $\Sigma \triangleright M \overset{\mu}{\longmapsto} \Sigma' \triangleright M'$. *If* $\mathbf{fn}(N) \subseteq \Sigma_{\mathcal{N}}$, *that is for arbitrary system* N *that consists only of names known in* $\Sigma_{\mathcal{N}}$, *then* $\Sigma \triangleright M|N \overset{\mu}{\longmapsto} \Sigma' \triangleright M'|N$ *and* $\Sigma \triangleright N|M \overset{\mu}{\longmapsto} \Sigma' \triangleright N|M'$.
- *Suppose* $\Sigma \triangleright M \xmapsto{(\tilde{n}:\tilde{\mathsf{L}})l:a!\langle V\rangle} \Sigma' \triangleright M'$ *and* $\Sigma \triangleright N \xmapsto{(\tilde{n}:\tilde{\mathsf{K}})l:a?(V)} \Sigma'' \triangleright N'$ *where* $\tilde{\mathsf{K}} = \tilde{\mathsf{L}} \setminus \mathbf{dom}(\Sigma_{\mathcal{H}})$. *Then*
  - $\Sigma \triangleright M|N \overset{\tau}{\longmapsto} \Sigma \triangleright (\nu\,\tilde{n}\!:\!\tilde{\mathsf{T}})M'|N'$ *where* $\tilde{\mathsf{L}} = \mathsf{lnk}_{O}(\tilde{n}\!:\!\tilde{\mathsf{T}}, \Sigma)$
  - $\Sigma \triangleright N|M \overset{\tau}{\longmapsto} \Sigma \triangleright (\nu\,\tilde{n}\!:\!\tilde{\mathsf{T}})N'|M'$ *where* $\tilde{\mathsf{L}} = \mathsf{lnk}_{O}(\tilde{n}\!:\!\tilde{\mathsf{T}}, \Sigma)$

PROOF. The proof for the first clause is trivial, by using (l-deriv-1), (l-deriv-2) or (l-deriv-3), depending on the structure of μ to extract the original transition, (l-par-ctxt) to compose N and then again (l-deriv-1), (l-deriv-2) or (l-deriv-3) to obtain the derived transition. We here outline the proof for the more complicated second clause. From the structure of the derived action, we know that the hypothesis

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

is derived using (l-deriv-2) and from the inductive hypothesis of this rule we know

$$\Sigma \triangleright M \xrightarrow{(\tilde{n}:\tilde{\mathtt{T}})l:a!\langle V\rangle} \Sigma' \triangleright M' \text{ where } \tilde{\mathtt{L}} = \mathsf{lnk}_O(\tilde{n}:\tilde{\mathtt{T}}, \Sigma) \tag{20}$$

From (20), and Lemma 37 we immediately get

$$\uparrow(\Sigma) \triangleright M \xrightarrow{(\tilde{n}:\tilde{\mathtt{T}})l:a!\langle V\rangle} \uparrow(\Sigma') \triangleright M' \tag{21}$$

Similarly, from the hypothesis $\Sigma \triangleright N \xmapsto{(\tilde{n}:\tilde{\mathtt{K}})l:a?(V)} \Sigma'' \triangleright N'$ and the inductive hypothesis of (l-deriv-3) we get

$$\Sigma \triangleright N \xrightarrow{(\tilde{n}:\tilde{\mathtt{U}})l:a?(V)} \Sigma'' \triangleright N' \text{ where } \tilde{\mathtt{K}} = \mathsf{lnk}_O(\tilde{n}:\tilde{\mathtt{U}}, \Sigma) \tag{22}$$

From the hypothesis $\tilde{\mathtt{K}} = \tilde{\mathtt{L}} \setminus \mathbf{dom}(\Sigma_{\mathcal{H}})$, the condition $\tilde{\mathtt{L}} = \mathsf{lnk}_O(\tilde{n}:\tilde{\mathtt{T}}, \Sigma)$ of (20) and the condition $\tilde{\mathtt{K}} = \mathsf{lnk}_O(\tilde{n}:\tilde{\mathtt{U}}, \Sigma)$ of (22) we obtain

$$\tilde{\mathtt{U}} = \tilde{\mathtt{T}} \setminus \mathbf{dom}(\Sigma_{\mathcal{H}}) \tag{23}$$

We recall that $\uparrow(-)$ collapses the observable and hidden parts of a network into one observable part. Thus, by (22), (23) and Lemma 37 we immediately get

$$\uparrow(\Sigma) \triangleright N \xrightarrow{(\tilde{n}:\tilde{\mathtt{T}})l:a?(V)} \uparrow(\Sigma'') \triangleright N' \tag{24}$$

Hence, by (21), (24), (l-par-comm) and (l-deriv-1) we conclude

$$\Sigma \triangleright M|N \xmapsto{\tau} \Sigma \triangleright (\nu\, \tilde{n}:\tilde{\mathtt{T}})M'|N'$$
$$\Sigma \triangleright N|M \xmapsto{\tau} \Sigma \triangleright (\nu\, \tilde{n}:\tilde{\mathtt{T}})N'|M'$$

as required.

Lemma 39 (Decomposition) *Suppose $\Sigma \triangleright M|N \xmapsto{\mu} \Sigma' \triangleright M'$ where $\Sigma \vdash_O M$ or $\Sigma \vdash_O N$. Then, one of the following conditions hold:*

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

where $\tilde{\mathtt{K}} = \tilde{\mathtt{L}} \setminus \mathbf{dom}(\Sigma_{\mathcal{H}})$, $\tilde{\mathtt{K}} = \mathsf{lnk}_O(\tilde{n}:\tilde{\mathtt{U}}, \Sigma)$ *and* $\tilde{\mathtt{L}} = \mathsf{lnk}_O(\tilde{n}:\tilde{\mathtt{T}}, \Sigma)$

PROOF. The proof proceeds by induction on the derivation of $\Sigma \triangleright M|N \xmapsto{\mu} \Sigma' \triangleright M'$. We focus on case (3) where $\mu = \tau$, and the last two rules used in our derivation were (l-deriv-1) (Table 9) and (l-par-comm) (Table 8). From the premises of (l-par-comm) we derive

$$\Sigma' = \Sigma \tag{25}$$
$$M' \text{ is } (\nu\,\tilde{n}:\tilde{\mathrm{T}})M''|N' \tag{26}$$
$$\uparrow(\Sigma) \triangleright M \xrightarrow{(\tilde{n}:\tilde{\mathrm{T}})l:a!\langle V\rangle} \Sigma' \triangleright M' \tag{27}$$
$$\uparrow(\Sigma) \triangleright N \xrightarrow{(\tilde{n}:\tilde{\mathrm{T}})l:a?(V)} \Sigma'' \triangleright N' \tag{28}$$

or viceversa. From the assumption that $\Sigma \vdash_{\mathrm{O}} M$ or $\Sigma \vdash_{\mathrm{O}} N$ we derive

$$\Sigma \vdash_{\mathrm{O}} l:\mathbf{alive} \tag{29}$$

And from (27), (28), (29) and Lemma 37 we get

$$\Sigma \triangleright M \xrightarrow{(\tilde{n}:\tilde{\mathrm{T}})l:a!\langle V\rangle} \Sigma''' \triangleright M'$$
$$\Sigma \triangleright N \xrightarrow{(\tilde{n}:\tilde{\mathrm{U}})l:a?(V)} \Sigma'''' \triangleright N' \text{ where } \tilde{\mathrm{U}} = \tilde{\mathrm{T}} \setminus \mathbf{dom}(\Sigma_{\mathcal{H}})$$

and using Proposition 35 we can rewrite the residual networks as

$$\Sigma \triangleright M \xrightarrow{(\tilde{n}:\tilde{\mathrm{T}})l:a!\langle V\rangle} \Sigma + \tilde{n} : \tilde{\mathrm{T}} \triangleright M' \tag{30}$$
$$\Sigma \triangleright N \xrightarrow{(\tilde{n}:\tilde{\mathrm{U}})l:a?(V)} \Sigma + \tilde{n} : \tilde{\mathrm{U}} \triangleright N' \text{ where } \tilde{\mathrm{U}} = \tilde{\mathrm{T}} \setminus \mathbf{dom}(\Sigma_{\mathcal{H}}) \tag{31}$$

From (30) and (31) and (l-deriv-2) and (l-deriv-3) we obtain

$$\Sigma \triangleright M \xmapsto{(\tilde{n}:\tilde{\mathrm{L}})l:a!\langle V\rangle} \Sigma + \tilde{n} : \tilde{\mathrm{T}} \triangleright M' \text{ where } \tilde{\mathrm{L}} = \mathsf{lnk}_{O}(\tilde{n}:\tilde{\mathrm{T}}, \Sigma) \tag{32}$$
$$\Sigma \triangleright N \xmapsto{(\tilde{n}:\tilde{\mathrm{K}})l:a?(V)} \Sigma + \tilde{n} : \tilde{\mathrm{U}} \triangleright N' \text{ where } \tilde{\mathrm{K}} = \mathsf{lnk}_{O}(\tilde{n}:\tilde{\mathrm{U}}, \Sigma) \tag{33}$$

Finally, from $\tilde{\mathrm{U}} = \tilde{\mathrm{T}} \setminus \mathbf{dom}(\Sigma_{\mathcal{H}})$ from (31), $\tilde{\mathrm{L}} = \mathsf{lnk}_{O}(\tilde{n} : \tilde{\mathrm{T}}, \Sigma)$ from (32) and $\tilde{\mathrm{K}} = \mathsf{lnk}_{O}(\tilde{n}:\tilde{\mathrm{U}}, \Sigma)$ from (33) we obtain

$$\tilde{\mathrm{K}} = \tilde{\mathrm{L}} \setminus \mathbf{dom}(\Sigma_{\mathcal{H}}) \tag{34}$$

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 still require a number of propositions and lemmas that help us stitch up this proof. The first proposition establishes relationships between derived actions, observers, the system N and the observable part of the network $\mathcal{I}(\Sigma)$ of a configuration $\Sigma \triangleright N$.

Proposition 40 (Derived Actions, Observers and Observable Networks)

(1) If $\Sigma \triangleright N \overset{\mu}{\longmapsto} \Sigma' \triangleright N'$ *where* μ *is an external derived action, and* $\mathcal{I}(\Sigma) = \mathcal{I}(\Sigma'')$ *for some* Σ''*, then* $\Sigma'' \triangleright N \overset{\mu}{\longmapsto} \Sigma''' \triangleright N'$.
(2) If $\Sigma \vdash_{\mathsf{O}} O$ *and* $\mathcal{I}(\Sigma) = \mathcal{I}(\Sigma')$ *then* $\Sigma' \vdash_{\mathsf{O}} O$
(3) If $\Sigma \vdash_{\mathsf{O}} O$*,* $\Sigma \triangleright O \overset{\tau}{\longmapsto} \Sigma' \triangleright O'$ *and for some* Σ'' *we have* $\mathcal{I}(\Sigma) = \mathcal{I}(\Sigma'')$*, then* $\Sigma'' \triangleright O \overset{\tau}{\longmapsto} \Sigma''' \triangleright O'$

PROOF. The proof for the first clause is by induction on the inference of $\Sigma \triangleright N \overset{\mu}{\longmapsto} \Sigma' \triangleright N'$, using Lemma 36 to infer the structure of N from μ in the case of derived input/output actions. If the external action is either $\mathsf{kill} : l$ or $l \nleftrightarrow k$ we use (l-halt) and (l-disc) respectively instead. The proof for the second clause is by induction on the structure of O and the definition of $\Sigma \vdash_{\mathsf{O}} O$. The proof of the third clause proceeds by induction on $\Sigma \triangleright O \overset{\tau}{\longmapsto} \Sigma' \triangleright O'$. We note that the third clause differs from the first clause of Proposition 40 because it deals with valid observers and internal moves (as opposed to systems and external moves).

We also prove a specific lemma that generalises scoping rule (l-rest) over derived actions.

Lemma 41 *If* $\Sigma + n : \mathtt{T} \triangleright M \overset{\mu}{\longmapsto} \Sigma' + n : \mathtt{T} \triangleright M'$ *and* $n \notin \mathbf{fn}(\mu)$*, then* $\Sigma \triangleright (\nu n : \mathtt{T})M \overset{\mu}{\longmapsto} \Sigma' \triangleright (\nu n : \mathtt{T})M'$

PROOF. By case analysis of μ, rules (l-deriv-1), (l-deriv-2), (l-deriv-3) and (l-rest).

The next two specific lemmas concern valid observers, their relationship with their transitions and network extensions. The first one states that when a valid observer transitions over Σ, it still remains a valid observer with respect to Σ.

Lemma 42 (Observers and Actions) *If* $\Sigma \vdash_{\mathsf{O}} O$ *and* $\Sigma \triangleright O \overset{\mu}{\longmapsto} \Sigma' \triangleright O'$ *then* $\Sigma' \vdash_{\mathsf{O}} O'$.

PROOF. We use Lemma 36 to infer the structure of O, O' from μ and Lemma 35 to infer the structure of Σ after μ and then show that $(\Sigma \ \mathsf{after}\ \mu) \vdash_{\mathsf{O}} O'$.

The second specific lemma states that if we change the links of an observable node, and this change does not alter the set of visible nodes, that is we do not connect the node to hidden nodes, then the set of valid observers remains unaltered.

Lemma 43 (Observers and Network extensions) *If* $\Sigma + n : \mathtt{U} \vdash_{\mathsf{O}} O$ *where* $\mathbf{fn}(\mathtt{U}) \in \mathbf{dom}(\Sigma_{O})$ *then* $\Sigma + n : \mathtt{T} \vdash_{\mathsf{O}} O$ *for any* $\mathtt{T}$ *where* $\mathtt{U} = \mathtt{T} \setminus \mathbf{dom}(\Sigma_{\mathcal{H}})$.

PROOF. The proof proceeds by a simple induction on the structure of O. We note that $\Sigma \vdash_O \mathtt{U}$ ensures that 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 $\Sigma{+}n{:}\mathtt{U}$.

The last lemma required before we can prove contextuality of $\approx$ is prompted by the first two conditions of the Decomposition Lemma 39, namely that observing code may alter the state of the network by inducing failure. We thus need the following lemma to guarantee closure.

Lemma 44 *Suppose $\Sigma_1 \triangleright M_1 \approx \Sigma_2 \triangleright M_2$. Then there exists some M', M'' such that:*

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

PROOF. We here prove the first clause and leave the second similar clause for the interested reader. If $\Sigma_1 \nvdash l : \textbf{alive}$ 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 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.

Contextuality is proved by inductively defining the least contextual relation over effective configurations (Definition 29), whose base elements are bisimilar configurations, and then show that this relation is closed with respect to our derived actions.

Proposition 45 (Contextuality of Bisimulation Equivalence) *If two configurations are bisimilar, they are also bisimilar under any context. Stated otherwise, $\Sigma_1 \triangleright M_1 \approx \Sigma_2 \triangleright M_2$ implies that:*

- *For any valid observer, $\Sigma_i \vdash_O O$ for $i = 1, 2$ we have $\Sigma_1 \triangleright M_1|O \approx \Sigma_2 \triangleright M_2|O$ and $\Sigma_1 \triangleright O|M_1 \approx \Sigma_2 \triangleright O|M_2$*
- *For any n fresh in Σ_1, Σ_2 and any valid observer type $\Sigma_i \vdash_O \mathtt{T}$ for $i = 1, 2$ we have $\Sigma_1{+}n{:}\mathtt{T} \triangleright M_1 \approx \Sigma_2{+}n{:}\mathtt{T} \triangleright M_2$*

PROOF. The proof proceeds by inductively defining a relation $\mathcal{R}$ as the least relation over effective configurations satisfying:

$$\mathcal{R} = \left\{ \begin{array}{l|l} \langle \Sigma_1 \triangleright M_1,\ \Sigma_2 \triangleright M_2 \rangle & \Sigma_1 \triangleright M_1 {\approx} \Sigma_2 \triangleright M_2 \\ & \\ \langle \Sigma_1 \triangleright M_1|O,\ \Sigma_2 \triangleright M_2|O \rangle & \Sigma_1 \triangleright M_1 \mathcal{R} \Sigma_2 \triangleright M_2 \text{ and} \\ \langle \Sigma_1 \triangleright O|M_1,\ \Sigma_2 \triangleright O|M_2 \rangle & \Sigma_i \vdash_{\mathrm{O}} O \text{ for } i = 1,2 \\ & \\ \langle \Sigma_1 {+} n{:}\mathrm{T} \triangleright M_1,\ \Sigma_2 {+} n{:}\mathrm{T} \triangleright M_2 \rangle & \begin{array}{l} \Sigma_1 \triangleright M_1\ \mathcal{R}\ \Sigma_2 \triangleright M_2, \\ \mathbf{fn}(\mathrm{T}) \in \mathbf{dom}(\Sigma_{iO}) \text{ for } i = 1,2 \text{ and } n \text{ is fresh} \end{array} \\ & \\ \langle \Sigma_1 \triangleright (\nu\, n{:}\mathrm{T}) M_1,\ \Sigma_2 \triangleright (\nu\, n{:}\mathrm{U}) M_2 \rangle & \Sigma_1 {+} n{:}\mathrm{T} \triangleright M_1\ \mathcal{R}\ \Sigma_2 {+} n{:}\mathrm{U} \triangleright M_2 \end{array} \right\}$$

and showing that $\mathcal{R} \subseteq \approx$; by co-induction, since $\approx$ is the largest possible relation, this would mean that it is contextual. We note that our definition of contextual relations, Definition 8, 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 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 44 can be easily extended from $\approx$ to $\mathcal{R}$ as:

Lemma 46 *If* $\Sigma_1 \triangleright M_1\ \mathcal{R}\ \Sigma_2 \triangleright M_2$*, then there exist some* M', M'' *such that:*

- $\Sigma_2 \triangleright M_2 \overset{\hat{\tau}}{\Longrightarrow} \Sigma_2 \triangleright M'$ *and* $\Sigma_2 - l \triangleright M' \overset{\tau}{\Longrightarrow} \Sigma_2 - l \triangleright M''$*, where* $\Sigma_1 - l \triangleright M_1 \mathcal{R} \Sigma_2 - l \triangleright M''$
- $\Sigma_2 \triangleright M_2 \overset{\hat{\tau}}{\Longrightarrow} \Sigma_2 \triangleright M'$ *and* $\Sigma_2 - l \leftrightarrow k \triangleright M' \overset{\tau}{\Longrightarrow} \Sigma_2 - l \leftrightarrow k \triangleright M''$*, where* $\Sigma_1 - l \leftrightarrow k \triangleright M_1\ \mathcal{R}\ \Sigma_2 - l \leftrightarrow k \triangleright M''$

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 44 and the three inductive cases are straightforward.

Similarly, also Proposition 33 can be extended to $\mathcal{R}$ as

Proposition 47 *If* $\Sigma_M \triangleright M\ \mathcal{R}\ \Sigma_N \triangleright N$ *then* $\mathcal{I}(\Sigma_M) = \mathcal{I}(\Sigma_N)$

Throughout the proof, when validating observer code, we will use interchangeably the notation $\Sigma_1 \vdash_{\mathrm{O}} O$ and $\Sigma_2 \vdash_{\mathrm{O}} O$ for cases where it is assumed that $\mathcal{I}(\Sigma_1) = \mathcal{I}(\Sigma_2)$.

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

$$\Sigma_1 \triangleright M_1|O \,\mathcal{R}\, \Sigma_2 \triangleright M_2|O \text{ because } \Sigma_1 \triangleright M_1 \,\mathcal{R}\, \Sigma_2 \triangleright M_2 \text{ and } \Sigma_1 \vdash O \tag{35}$$

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

We thus assume $\Sigma_1 \triangleright M_1|O \xmapsto{\mu} \Sigma_1' \triangleright M_1'$. We decompose this action using the Decomposition Lemma 39 and focus on the most difficult case, where

$$M_1' \text{ is } (\nu \tilde{n}:\tilde{\mathtt{T}})M_1'|O',\ \mu \text{ is } \tau \text{ and } \Sigma_1' = \Sigma_1 \tag{36}$$

$$\Sigma_1 \triangleright M_1 \xmapsto{(\tilde{n}:\tilde{\mathtt{L}})l:a!\langle V\rangle} \Sigma_1 + \tilde{n}:\tilde{\mathtt{T}} \triangleright M_1' \tag{37}$$

$$\Sigma_1 \triangleright O \xmapsto{(\tilde{n}:\tilde{\mathtt{K}})l:a?(V)} \Sigma_1 + \tilde{n}:\tilde{\mathtt{U}} \triangleright O' \tag{38}$$

$$\text{where } \tilde{\mathtt{K}} = \tilde{\mathtt{L}} \setminus \mathbf{dom}(\Sigma_{1\mathcal{H}}),\ \tilde{\mathtt{K}} = \mathsf{lnk}_O(\tilde{n}:\tilde{\mathtt{U}}, \Sigma) \text{ and } \tilde{\mathtt{L}} = \mathsf{lnk}_O(\tilde{n}:\tilde{\mathtt{T}}, \Sigma) \tag{39}$$

From (35), (37) and the inductive hypothesis we derive the matching weak action

$$\Sigma_2 \triangleright M_2 \overset{(\tilde{n}:\tilde{\mathtt{L}})l:a!\langle V\rangle}{\Longmapsto} \Sigma_2' + \tilde{n}:\tilde{\mathtt{W}} \triangleright M_2' \tag{40}$$

$$\text{where } \Sigma_2' + \tilde{n}:\tilde{\mathtt{W}} \triangleright M_2' \;\mathcal{R}\; \Sigma_1 + \tilde{n}:\tilde{\mathtt{T}} \triangleright M_1' \tag{41}$$

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 may be updates to the hidden part of the networks which we abstract away in the linktype $\tilde{\mathtt{L}}$.

Now (40) can be decomposed as

$$\Sigma_2 \triangleright M_2 \quad \overset{\hat{\tau}}{\Longmapsto} \quad \Sigma_2'' \triangleright M_2'' \tag{42}$$

$$\Sigma_2'' \triangleright M_2'' \xmapsto{(\tilde{n}:\tilde{\mathtt{L}})l:a!\langle V\rangle} \Sigma_2'' + \tilde{n}:\tilde{\mathtt{W}} \triangleright M_2''' \tag{43}$$

$$\Sigma_2'' + \tilde{n}:\tilde{\mathtt{W}} \triangleright M_2''' \quad \overset{\hat{\tau}}{\Longmapsto} \quad \Sigma_2' + \tilde{n}:\tilde{\mathtt{W}} \triangleright M_2' \tag{44}$$

From (42), $\Sigma_2 \vdash O$ and the Composition Lemma 38 we get

$$\Sigma_2 \triangleright M_2|O \overset{\hat{\tau}}{\Longmapsto} \Sigma_2'' \triangleright M_2''|O \tag{45}$$

From the hypothesis (35) we know $\mathcal{I}(\Sigma_1) = \mathcal{I}(\Sigma_2)$, from (41) and Proposition 47 we know $\mathcal{I}(\Sigma_1 + \tilde{n} : \tilde{\mathtt{T}}) = \mathcal{I}(\Sigma_2' + \tilde{n} : \tilde{\mathtt{W}})$ and Lemma 32 we know that the visible part of Σ_2'' and Σ_2' did not change as a result of the silent transitions in (42) and (44) and thus

$$\mathcal{I}(\Sigma_2'') = \mathcal{I}(\Sigma_2') = \mathcal{I}(\Sigma_2) = \mathcal{I}(\Sigma_1) \tag{46}$$

Hence by (46), (38) and Lemma 40 we get

$$\Sigma_2'' \triangleright O \xrightarrow{(\tilde{n}:\tilde{\mathtt{K}})l:a?(V)} \Sigma_2'' + \tilde{n} : \tilde{\mathtt{U}} \triangleright O' \text{ where } \tilde{\mathtt{U}} = \tilde{\mathtt{W}} \setminus \mathbf{dom}(\Sigma''_{2\mathcal{H}}) \tag{47}$$

At this point we note that from our inductive hypothesis (35), and (46) we derive

$$\Sigma_2'' \vdash_{\mathsf{O}} O \tag{48}$$

from (47), (48) and Lemma 42 we deduce

$$\Sigma_2'' + \tilde{n} : \tilde{\mathtt{U}} \vdash_{\mathsf{O}} O' \tag{49}$$

and from (49) and Lemma 43 we obtain

$$\Sigma_2'' + \tilde{n} : \tilde{\mathtt{W}} \vdash_{\mathsf{O}} O' \tag{50}$$

Combining (43), (47), (48) and the Composition Lemma 38, we obtain

$$\Sigma_2'' \triangleright M_2''|O \overset{\tau}{\longmapsto} \Sigma_2'' \triangleright (\nu\, \tilde{n} : \tilde{\mathtt{W}}) M_2'''|O' \tag{51}$$

Similarly, from (44), (50) and the Composition Lemma 38 we obtain

$$\Sigma_2'' + \tilde{n} : \tilde{\mathtt{W}} \triangleright M_2'''|O' \overset{\hat{\tau}}{\Longmapsto} \Sigma_2' + \tilde{n} : \tilde{\mathtt{W}} \triangleright M_2'|O' \tag{52}$$

and by applying Lemma 41 on (52) we get

$$\Sigma_2'' \triangleright (\nu\, \tilde{n} : \tilde{\mathtt{W}}) M_2'''|O' \overset{\hat{\tau}}{\Longmapsto} \Sigma_2' \triangleright (\nu\, \tilde{n} : \tilde{\mathtt{W}}) M_2'|O' \tag{53}$$

Thus, by combining (45), (51) and (53) we obtain the matching move

$$\Sigma_2 \triangleright M_2|O \overset{\tau}{\Longmapsto} \Sigma_2' \triangleright (\nu\, \tilde{n} : \tilde{\mathtt{W}}) M_2'|O' \tag{54}$$

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

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

Recalling (41) we know

$$\Sigma_1 + \tilde{n} : \tilde{\mathtt{T}} \triangleright M_1' \;\mathcal{R}\; \Sigma_2' + \tilde{n} : \tilde{\mathtt{W}} \triangleright M_2' \tag{55}$$

which, by Proposition 47, means that

$$\mathcal{I}(\Sigma_1 + \tilde{n} : \tilde{\mathtt{T}}) = \mathcal{I}(\Sigma_2' + \tilde{n} : \tilde{\mathtt{W}}) \tag{56}$$

Now from (50), (46), Lemma 43 and (56) we deduce

$$\Sigma_1 + \tilde{n} : \tilde{\mathtt{T}} \vdash_{\mathrm{O}} O' \text{ and } \Sigma_2 + \tilde{n} : \tilde{\mathtt{W}} \vdash_{\mathrm{O}} O' \tag{57}$$

and thus from the definition of $\mathcal{R}$ we obtain

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

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

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

as required.

We now conclude this part by showing that bisimulation is *sound* with respect to reduction barbed congruence. Before however, we still require one important lemma stating that our formulation of *internal activity*, namely $\overset{\tau}{\longmapsto}$, is in agreement, in some sense, with the reduction semantics.

Proposition 48 (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 36 in our induction to infer the structure of the communicating subterms.

We are finally in a position to prove that our bisimulation, $\approx$, is a contextual relation, according to Definition 8.

Proposition 49 (Soundness)

$$\Sigma_1 \triangleright M_1 \approx \Sigma_2 \triangleright M_2 \textit{ implies } \Sigma_1 \triangleright M_1 \cong \Sigma_2 \triangleright M_2$$

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 48. Barb preserving is also straightforward, from Proposition 48 and the direct relationship between barbs and output actions. Finally, Proposition 45 proves that $\approx$ is also contextual.

5.2 Completeness

It remains to be shown 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 [19,18]. 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 barbed congruence is preserved by scoping. Stated otherwise, if two configurations are reduction barbed congruent, scoping a channel or location name on both sides would still yield two reduction barbed congruent configurations.

Proposition 50 (Scoping and reduction barbed congruence)

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

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

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

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

$$\Sigma_M \vdash_{\mathsf{O}} O \tag{58}$$

and we have to show that

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

From (58) we know that $n \notin \mathbf{fn}(O)$ and thus

$$\Sigma_M + n{:}\mathtt{T} \vdash_{\mathsf{O}} O \text{ and } \Sigma_N + n{:}\mathtt{U} \vdash_{\mathsf{O}} O \tag{59}$$

Hence by contextuality of $\cong$ and (59), 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 [19,18] 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 new constructs 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 $\mathsf{ping}\ l.P\ \mathsf{else}\ Q$ 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 51 (Inducing faults)

- *Suppose* $\Sigma \vdash_O l : \textbf{alive}$. *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* $\Sigma' \nvdash_O l : \textbf{alive}$ *implies* $\Sigma \triangleright N \xmapsto{\mathsf{kill}:l} \Sigma' \triangleright N''$ *such that* $N' \equiv N''$
- *Suppose* $\Sigma \vdash_O l \leftrightarrow k$. *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* $\Sigma' \nvdash_O l \leftrightarrow k$ *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 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.

We next show that for any network Σ, the context can determine the exact state of the observable network $\mathcal{I}(\Sigma)$. We note that Σ_N denotes all the names known by the observer so far; the main part of this proof thus consists in showing that the observer can also determine Σ_O and just Σ_O. To show this we define an observer $verStat_k(\mathcal{L}, x)$ running at a location k, which is assumed to be *connected to all observable locations* in Σ_O - we are guaranteed to always have such a completely connected location where to run $verStat_k(\mathcal{L}, x)$ since contexts can extend configurations by a fresh location with such a property; see Definition 29. Apart from the channel x and the location k, the observer $verStat_k(\mathcal{L}, x)$ is instantiated with $\mathcal{L}$, which denotes the network knowledge the observer intends to verify. We show that this specific observer can produce the barb $x@k$ if and only if the linkset it is checking for, $\mathcal{L}$, is equal to (modulo symmetric links) Σ_O.

We find it convenient to define some notation. We start by formalising linkset equality, denotes as $\mathcal{L} \approx \mathcal{L}'$, as the symmetric relation over linksets such that:

- $\langle l, k\rangle \in \mathcal{L}$ implies $\langle l, k\rangle$ or $\langle k, l\rangle \in \mathcal{L}'$
- $\langle l, k\rangle \in \mathcal{L}'$ implies $\langle l, k\rangle$ or $\langle k, l\rangle \in \mathcal{L}$

In addition, for every effective linkset $\mathcal{L}$ we define an operation $l \leftrightarrow \mathcal{L}$ which returns a linkset that represents the state relating to location l in $\mathcal{L}$; the linkset returned denotes the liveness of l and any links it has with other live locations. Formally we define this operation as

$$l \leftrightarrow \mathcal{L} = \{\langle k, k' \rangle \mid \langle k, k' \rangle \in \mathcal{L} \wedge l = k \vee l = k'\}$$

We note that since $\mathcal{L}$ is assumed to be an effective linkset, if $l \in \mathbf{dom}(\mathcal{L})$ then $\langle l, l \rangle \in l \leftrightarrow \mathcal{L}$. In fact $l \leftrightarrow \mathcal{L}$ is either $\emptyset$ or of the form

$$\{\langle l, l \rangle, \langle l, k_1 \rangle, \ldots, \langle l, k_i \rangle, \langle k_{i+1}, l \rangle, \ldots, \langle k_n, l \rangle\} \tag{60}$$

where $k_1 \ldots k_n$ denote the connections l has with other live locations in $\mathcal{L}$ (we also know $\forall 1 \leq i \leq n.\langle k_i, k_i \rangle \in \mathcal{L}$). When a linkset $\mathcal{L}$ observes the general form of (60), we find it convenient to denote it as $\mathcal{L}^l$ to show that it is a linkset solely concerned with the state of l.

The operation $l \leftrightarrow \mathcal{L}$ provides a systematic way to divide an effective linkset such that every subdivision focusses on the information relating to a single location, as stated through the following lemma.

Lemma 52 (Effective Linkset Subdivision) *If $\mathcal{L}$ is an effective linkset then*

$$\mathcal{L} \approx \bigcup_{\mathcal{L} \vdash l:\mathbf{alive}} l \leftrightarrow \mathcal{L}$$

PROOF. Immediate from Definition 14 since $\langle l, k \rangle \in \mathcal{L}$ implies $\langle l, l \rangle \in \mathcal{L}$ and $\langle k, k \rangle \in \mathcal{L}$ and thus all links in $\mathcal{L}$ will be included in one of the sub-divisions.

We incrementally build the observer which can uniquely identify the observable network Σ_O in an effective network Σ. We define the process:

$$\mathit{verLocState}_{k_0}(\mathcal{L}^l) \Leftarrow \mathsf{go}\ l.(\nu\, s) \left(\begin{array}{l} \prod_{k \in \mathbf{dom}(\mathcal{L}^l)} \mathsf{ping}\ k.s!\langle\rangle\ \mathsf{else}\ \mathbf{0} \\ \mid \prod_{k \in (\mathbf{loc}(\Sigma_N) \setminus \mathbf{dom}(\mathcal{L}^l))} \mathsf{ping}\ k.\mathbf{0}\ \mathsf{else}\ s!\langle\rangle \\ \mid \underbrace{s?() \ldots s?()}_{|\mathbf{loc}(\Sigma_N)|}.\mathsf{go}\ k_0.\mathit{sync}!\langle\rangle \end{array} \right)$$

The following lemma states that for any network Σ where $k_0 \notin \Sigma_N$ the process $\mathit{verLocState}_{k_0}(\mathcal{L}^l)$ located at k_0, which is in turn connected to l, reduces to the system $k_0[\![\mathit{sync}!\langle\rangle]\!]$ if and only if the state relating to l in Σ is equal to $\mathcal{L}^l$, that is $l \leftrightarrow \Sigma_O \approx \mathcal{L}^l$.

Lemma 53 (Observable Location State) *For any Σ, if Σ_+ is the extended network* $\Sigma + k_0 : \mathtt{loc}[\mathtt{a}, \mathbf{dom}(\Sigma_O)] + sync : \mathtt{ch}$ *then*

(1) If $\Sigma_+ \triangleright k_0[\![verLocState_{k_0}(\mathcal{L}^l)]\!] \longrightarrow^* \Sigma_+ \triangleright k_0[\![sync!\langle\rangle]\!]$ then $\mathcal{L}^l \approx l{\leftrightarrow}\Sigma_O$
(2) If $\mathcal{L}^l \not\approx l{\leftrightarrow}\Sigma_O$ then $\Sigma_+ \triangleright k_0[\![verLocState_{k_0}(\mathcal{L}^l)]\!] \not\longrightarrow^* \Sigma_+ \triangleright k_0[\![sync!\langle\rangle]\!]$

PROOF. We have two clauses:

(1) We trace back on the sequence of reductions of

$$\Sigma_+ \triangleright k_0[\![verLocState_{k_0}(\mathcal{L}^l)]\!] \longrightarrow^* \Sigma_+ \triangleright k_0[\![sync!\langle\rangle]\!] \tag{61}$$

From the structure of $verLocState_{k_0}(\mathcal{L}^l)$ we know that the last reduction must have been a successful migration from l to k_0 from which we deduce that $\langle l, l\rangle \in \mathcal{L}^l$ and $\langle l, l\rangle \in \Sigma_O$. Tracing further back, we know that some reduction of (61) must have produced $|\mathbf{loc}(\Sigma_N)|$ outputs on the scoped channel s. This means that

(a) Every subprocess in $\prod_{k\in\mathbf{dom}(\mathcal{L}^l)} \mathsf{ping}\ k.s!\langle\rangle\ \mathsf{else}\ \mathbf{0}$ must have pinged successfully and from the side condition of (r-ping) we deduce that $\langle l, k\rangle \in \Sigma_O$ or $\langle k, l\rangle \in \Sigma_O$. This conclusion can be reformulated as

$$\langle l, k\rangle \in \mathcal{L}^l \text{ implies } \langle k, k'\rangle \in l{\leftrightarrow}\Sigma_O \text{ and } k = l \vee k' = l \tag{62}$$

(b) Every subprocess in $\prod_{k\in(\mathbf{loc}(\Sigma_N)\backslash\mathbf{dom}(\mathcal{L}^l))} \mathsf{ping}\ k.\mathbf{0}\ \mathsf{else}\ s!\langle\rangle$ must have pinged unsuccessfully and from the side condition of (r-nping) and the fact that Σ_O is an effective linkset we deduce that $\langle l, k\rangle \notin \Sigma_O$ and $\langle k, l\rangle \notin \Sigma_O$. This conclusion can be reformulated as

$$\langle l, k\rangle \notin \mathcal{L}^l \text{ implies } \langle k, k'\rangle \notin l{\leftrightarrow}\Sigma_O \text{ where } k = l \vee k' = l$$

which can be expressed without negative set inclusions as

$$\langle l, k\rangle \in l{\leftrightarrow}\Sigma_O \text{ implies } \langle k, k'\rangle \in \mathcal{L}^l \text{ and } k = l \vee k' = l \tag{63}$$

From (62) and (63) we deduce $\mathcal{L}^l \approx l{\leftrightarrow}\Sigma_O$.

(2) For the second clause we apply similar reasoning to show that $\Sigma_+ \triangleright k_0[\![verLocState_{k_0}(\mathcal{L}^l)]\!]$ must block before reducing to $\Sigma_+ \triangleright k_0[\![sync!\langle\rangle]\!]$

We define the aforementioned observer $verStat_{k_0}(\mathcal{L}, x)$ as:

$$verStat_{k_0}(\mathcal{L}, x) \Leftarrow (\nu\, sync)\; k_0 \left[\!\!\left[\begin{array}{l} \prod_{l \in \mathbf{dom}(\mathcal{L})} verLocState_{k_0}(l \leftrightarrow \mathcal{L}) \\ \mid \prod_{l \in (\mathbf{loc}(\Sigma_N) \setminus \mathbf{dom}(\mathcal{L}))} \mathsf{ping}\ l.\mathbf{0}\ \mathsf{else}\ sync!\langle\rangle \\ \mid \underbrace{sync?().\ldots.sync?()}_{|\mathbf{loc}(\Sigma_N)|}.x!\langle\rangle \end{array} \right]\!\!\right]$$

The first group of subprocesses $verLocState_{k_0}(l \leftrightarrow \mathcal{L})$ ensure that all the locations mentioned in $\mathcal{L}$ do have the links mentioned in the linkset and just those. The second group of subprocesses $\mathsf{ping}\ l.\mathbf{0}\ \mathsf{else}\ sync!\langle\rangle$ ensure that there are no more accessible locations from k_0 apart from the ones mentioned in $\mathcal{L}$.

Lemma 54 (Observable network) *For any Σ, and for any effective linkset $\mathcal{L} \not\approx \Sigma_O$, if $\Sigma_+ = \Sigma + k_0 : \mathtt{loc}[\mathtt{a}, \mathbf{dom}(\Sigma_O)] + \mathrm{succ} : \mathtt{ch}$ implies*

(1) $\Sigma_+ \triangleright verStat_{k_0}(\Sigma_O, \mathrm{succ}) \longrightarrow^* \Sigma_+ \triangleright k_0[\![\mathrm{succ}!\langle\rangle]\!]$
(2) $\Sigma_+ \triangleright verStat_{k_0}(\mathcal{L}, \mathrm{succ}) \not\longrightarrow^* \Sigma_+ \triangleright k_0[\![\mathrm{succ}!\langle\rangle]\!]$

PROOF. We have two clauses to prove:

(1) By Lemma 52 we know that the subprocesses of $verStat_{k_0}(\Sigma_O, \mathrm{succ})$ cover all of Σ_O and by Lemma 53 we know that every component will produce $k_0[\![sync!\langle\rangle]\!]$. By definition of Σ_+ we also know that these are the only locations accessible from k_0 so all pings will produce $k_0[\![sync!\langle\rangle]\!]$. As a result the system can reduce to $k_0[\![\mathrm{succ}]\!]$.

(2) If $\mathcal{L}$ does not include any of the locations in Σ_O then one of the pings will trivially not produce $k_0[\![sync!\langle\rangle]\!]$. If $\mathcal{L}$ contains an l that is not in Σ_O then the first migration of the subprocess $verLocState_{k_0}(l \leftrightarrow \mathcal{L})$ will immediately fail and thus never produce $k_0[\![sync!\langle\rangle]\!]$. Finally if $\mathbf{dom}(\mathcal{L}) = \mathbf{dom}(\Sigma_O)$ but still some links do not correspond, then there will be an l such that $l \leftrightarrow \mathcal{L} \not\approx l \leftrightarrow \Sigma_O$. By Lemma 53 we know this will not produce $k_0[\![sync!\langle\rangle]\!]$. As a result the system can never reduce to $k_0[\![\mathrm{succ}]\!]$.

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 in the action μ; note that 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 55 $\Sigma + n : \mathrm{T} \triangleright N \longrightarrow \Sigma' + n : \mathrm{T} \triangleright N'$ *where* $n \notin \mathbf{fn}(N)$ *iff* $\mathbf{fn}(N) \subseteq \Sigma_N$ *and* $\Sigma \triangleright N \longrightarrow \Sigma' \triangleright N'$

PROOF. The proofs are 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 56 (Definability) *Assume that for an arbitrary network representation Σ, the network Σ_+ denotes:*

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

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 $\Sigma_+ \vdash_O T^\mu(Nm, \textsc{succ}, \textsc{fail}, k_0)$, such that:

(1) $\Sigma \triangleright N \overset{\mu}{\longmapsto} \Sigma' + \mathbf{bn}(\mu) : \tilde{\mathtt{T}} \triangleright N'$ *for some* $\tilde{\mathtt{T}}$ *implies*
$\Sigma_+ \triangleright N \,|\, T^\mu(Nm, \textsc{succ}, \textsc{fail}, k_0) \Longrightarrow \Sigma'_+ \triangleright (\nu\, \mathbf{bn}(\mu) : \tilde{\mathtt{T}})\ (N' \mid k_0[\![\textsc{succ}!\langle \mathbf{bn}(\mu) \rangle]\!])$

(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*
$N' \equiv (\nu\, \mathbf{bn}(\mu) : \tilde{\mathtt{T}})\, (N'' | k_0[\![\textsc{succ}!\langle \mathbf{bn}(\mu) \rangle]\!])$ *for some* N''
such that $\Sigma \triangleright N \overset{\mu}{\Longmapsto} \Sigma' + \mathbf{bn}(\mu) : \tilde{\mathtt{T}} \triangleright N''$.

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 $\tilde{\mathtt{L}} = \mathsf{lnk}_O(\tilde{n} : \tilde{\mathtt{T}}, \Sigma)$ 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

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

such that

$$verNwStat_{k_0}(x_1 \ldots x_m, \mathcal{L}, y) \Leftarrow (\nu\, k' : \mathtt{T}_{k'})\mathsf{go}\ k'.(\nu d) \begin{pmatrix} verStat_{k'}(\mathcal{L} \cup \tilde{\mathtt{L}}, d) \\ \mid d?().\mathsf{go}\ k_0.y!\langle k' \rangle \end{pmatrix}$$

and $\mathtt{T}_{k'} = \mathtt{loc}[\mathsf{a}, Nm \cup \{x_1..x_m\}]$

In the above context we exploit the fact that we can have variables in location types that are not yet instantiated; they can then be replaced by actual location names through input, as we saw earlier in Example 5. For the sake of presentation, in the above context we assume that the first $v_1 \ldots v_m$ in $V = v_1 \ldots v_{|V|}$ in μ are bound, and the remaining $v_{m+1} \ldots v_{|V|}$ are free; a more general test can be constructed for arbitrary ordering of bound names in V using the same principles used for this test. We also use the conditional if $x \notin Nm.P$ as an abbreviation for the obvious nested negative comparisons between x and each name $n_i \in Nm$, that is

$$\text{if } x \notin Nm.P \Leftarrow \text{if } x = n_1 \text{ then } \mathbf{0} \text{ else } \ldots \text{if } x = n_{|Nm|} \text{ then } \mathbf{0} \text{ else } P$$

The test works in two stages. Similar to the tests in [19,18], 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, we use a slightly augmented version of the process $verStat_{k_0}(\mathcal{L}, y)$ from Lemma 54 called $verNwStat_{k_0}(x_1 \ldots x_m, \mathcal{L}, y)$. It 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 $x_1 \ldots x_m$ - the purpose of this procedure is to *short-circuit* our way to the newly reachable locations (see Example 5). We afterwards run $verStat_{k'}(\mathcal{L} \cup \tilde{\mathrm{L}}, d)$ from this new location k' and some fresh scoped location d, to verify that the new observable network state is indeed $\Sigma_O \cup \tilde{\mathrm{L}}$. 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 FAIL and releases an output on SUCC with the bound names $x_1 \ldots x_m$, whereas the second process goes back to k' to kill it for housekeeping purposes.

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

$$l[\![\text{kill}]\!] \mid k_0[\![\text{FAIL}!\langle\rangle]\!] \mid k_0[\![\text{ping } l.(\text{ping } l.\mathbf{0} \text{ else FAIL}?().\text{SUCC}!\langle\rangle) \text{ else } \mathbf{0}]\!]$$

and

$$l[\![\mathsf{break}\ k]\!] \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid (\nu\, sync)\left(\begin{array}{l} l[\![\mathsf{ping}\ k.(\mathsf{ping}\ k.\mathbf{0}\ \mathsf{else}\ \mathsf{go}\ k_0.sync!\langle\rangle)\ \mathsf{else}\ \mathbf{0}]\!] \\ \mid k[\![\mathsf{ping}\ l.(\mathsf{ping}\ l.\mathbf{0}\ \mathsf{else}\ \mathsf{go}\ k_0.sync!\langle\rangle)\ \mathsf{else}\ \mathbf{0}]\!] \\ \mid k_0[\![sync?().sync?().\textsc{fail}?().\textsc{succ}!\langle\rangle]\!] \end{array}\right)$$

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 less straightforward, because it cannot be tested for from the observer location k_0 directly, but from the connected locations l or k. It is even more complicated because it needs to be tested from *both sides*, l and k: 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 (two successive pings, the first to determine that k is accessible from l, or viceversa, and the second to determine that it is not anymore) from both endpoints, l and k, and synchronise at k_0.

The proof for the bound input and bound output actions can be extracted from [19,18]; we have an additional check for the output case were apart from checking that scope extruded names are fresh (as in [19,18]), we also verify that the links and nodes attached to scope extruded names (as linksets) are indeed the *only* nodes and links newly accessible to the observer. This follows from the use of *verStat* in the output testing context and Lemma 54.

We here 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. For the first clause, from $\Sigma \triangleright N \overset{\mathsf{kill}:l}{\longmapsto} \Sigma' \triangleright N'$, (l-deriv-1) and (l-halt) we know that $\Sigma \vdash_{\mathsf{O}} l : \mathbf{alive}$, thus

$$\Sigma_+ \vdash_{\mathsf{O}} l : \mathbf{alive} \tag{64}$$

which means we can perform the ping reduction, releasing the positive branch:

$$\begin{array}{r} \Sigma_+ \triangleright N \mid l[\![\mathsf{kill}]\!] \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\mathsf{ping}\ l.\mathsf{ping}\ l.\mathbf{0}\ \mathsf{else}\ \textsc{fail}?().\textsc{succ}!\langle\rangle]\!] \longrightarrow \\ \Sigma_+ \triangleright l[\![\mathsf{kill}]\!] \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\mathsf{ping}\ l.\mathbf{0}\ \mathsf{else}\ \textsc{fail}?().\textsc{succ}!\langle\rangle]\!] \end{array} \tag{65}$$

From (64), (l-halt) and (l-deriv-1) we derive

$$\Sigma_+ \triangleright N \xrightarrow{\mathsf{kill}:l} \Sigma'_+ \triangleright N' \quad \text{where } \Sigma_+ \vdash_{\mathsf{O}} l : \mathbf{alive} \text{ and } \Sigma'_+ \nvdash_{\mathsf{O}} l : \mathbf{alive} \tag{66}$$

and from (66) and Lemma 51 we get

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

and (r-par-ctxt) we derive

$$\begin{aligned}\Sigma_+ \triangleright N \,|\, l[\![\mathsf{kill}]\!] \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\mathsf{ping}\ l.\mathbf{0}\ \mathsf{else}\ \textsc{fail}?().\textsc{succ}!\langle\rangle]\!] \longrightarrow \\ \Sigma'_+ \triangleright N' \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\mathsf{ping}\ l.\mathbf{0}\ \mathsf{else}\ \textsc{fail}?().\textsc{succ}!\langle\rangle]\!]\end{aligned} \tag{67}$$

Subsequently we derive the sequence of reductions

$$\begin{aligned}\Sigma'_+ \triangleright N' \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\mathsf{ping}\ l.\mathbf{0}\ \mathsf{else}\ \textsc{fail}?().\textsc{succ}!\langle\rangle]\!] \longrightarrow \\ \Sigma'_+ \triangleright N' \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\textsc{fail}?().\textsc{succ}!\langle\rangle]\!] \longrightarrow \\ \Sigma'_+ \triangleright N' \mid k_0[\![\textsc{succ}!\langle\rangle]\!]\end{aligned} \tag{68}$$

Combining the reductions in (65), (67) and (68) 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

$$\begin{aligned}
&\Sigma_+ \triangleright N \mid l[\![\mathsf{kill}]\!] \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\mathsf{ping}\ l.\mathsf{ping}\ l.\mathbf{0}\ \mathsf{else}\ \textsc{fail}?().\textsc{succ}!\langle\rangle]\!] \Longrightarrow && (69)\\
&\Sigma^1_+ \triangleright N^1 \mid l[\![\mathsf{kill}]\!] \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\mathsf{ping}\ l.\mathsf{ping}\ l.\mathbf{0}\ \mathsf{else}\ \textsc{fail}?().\textsc{succ}!\langle\rangle]\!] \longrightarrow &&\\
&\Sigma^1_+ \triangleright N^1 \mid l[\![\mathsf{kill}]\!] \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\mathsf{ping}\ l.\mathbf{0}\ \mathsf{else}\ \textsc{fail}?().\textsc{succ}!\langle\rangle]\!] \Longrightarrow && (70)\\
&\Sigma^2_+ \triangleright N^2 \mid l[\![\mathsf{kill}]\!] \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\mathsf{ping}\ l.\mathbf{0}\ \mathsf{else}\ \textsc{fail}?().\textsc{succ}!\langle\rangle]\!] \longrightarrow && (71)\\
&\Sigma^2_+ - l \triangleright N^2 \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\mathsf{ping}\ l.\mathbf{0}\ \mathsf{else}\ \textsc{fail}?().\textsc{succ}!\langle\rangle]\!] \Longrightarrow && (72)\\
&\Sigma^3_+ - l \triangleright N^3 \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\mathsf{ping}\ l.\mathbf{0}\ \mathsf{else}\ \textsc{fail}?().\textsc{succ}!\langle\rangle]\!] \longrightarrow &&\\
&\Sigma^3_+ - l \triangleright N^3 \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\textsc{fail}?().\textsc{succ}!\langle\rangle]\!] \Longrightarrow && (73)\\
&\Sigma^4_+ - l \triangleright N^4 \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\textsc{fail}?().\textsc{succ}!\langle\rangle]\!] \longrightarrow &&\\
&\Sigma^4_+ - l \triangleright N^4 \mid k_0[\![\textsc{succ}!\langle\rangle]\!] \Longrightarrow && (74)\\
&\Sigma'_+ \triangleright N' \mid k_0[\![\textsc{succ}!\langle\rangle]\!] &&
\end{aligned}$$

From (71) and Lemma 51 we deduce

$$\begin{aligned}\Sigma^2_+ \triangleright N^2 \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\mathsf{ping}\ l.\mathbf{0}\ \mathsf{else}\ \textsc{fail}?().\textsc{succ}!\langle\rangle]\!] \xmapsto{\mathsf{kill}:l} \\ \Sigma^2_+ - l \triangleright N^2 \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\mathsf{ping}\ l.\mathbf{0}\ \mathsf{else}\ \textsc{fail}?().\textsc{succ}!\langle\rangle]\!]\end{aligned}$$

and by (l-deriv-1) and the inductive hypothesis of (l-halt) we know

$$\Sigma^2_+ \vdash_{\mathsf{O}} l : \mathbf{alive}\ \text{thus}\ \Sigma^2 \vdash_{\mathsf{O}} l : \mathbf{alive} \tag{75}$$

and by (75), (l-halt) and (l-deriv-1) we derive

$$\Sigma^2 \triangleright N^2 \overset{\mathsf{kill}:l}{\longmapsto} \Sigma^2 - l \triangleright N^2 \tag{76}$$

From (69), (70), (72), (73) and (74) and (r-par-ctxt) obtain

$$\begin{aligned} &\Sigma_+ \triangleright N \Longrightarrow \Sigma^1_+ \triangleright N^1 \Longrightarrow \Sigma^2_+ \triangleright N^2 \\ &\Sigma^2 - l_+ \triangleright N^2 \Longrightarrow \Sigma^3_+ \triangleright N^3 \Longrightarrow \Sigma^4_+ \triangleright N^4 \Longrightarrow \Sigma_+ \triangleright N' \end{aligned} \tag{77}$$

and from (77) and Lemma 55 we obtain

$$\begin{aligned} &\Sigma \triangleright N \Longrightarrow \Sigma^1 \triangleright N^1 \Longrightarrow \Sigma^2 \triangleright N^2 \\ &\Sigma^2 - l \triangleright N^2 \Longrightarrow \Sigma^3 \triangleright N^3 \Longrightarrow \Sigma^4 \triangleright N^4 \Longrightarrow \Sigma' \triangleright N' \end{aligned} \tag{78}$$

Finally, using Proposition 48 to convert the reductions in (78) into weak silent actions and merging these with (76) we obtain as required

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

The result of Proposition 56 (Definability) 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 $\Sigma' \triangleright N'$ but instead $\Sigma''_+ \triangleright (\nu\, \mathbf{bn}(\mu) : \tilde{\mathrm{T}})\, N \,|\, k_0[\![\textsc{succ}!\langle \mathbf{bn}(\mu)\rangle]\!]$ where $\Sigma'' + \mathbf{bn}(\mu) : \tilde{\mathrm{T}} = \Sigma'$.

We therefore state and prove a variant of the extrusion lemma in [19,18], which enables us to recover the residual $\Sigma' \triangleright N'$ from $\Sigma''_+ \triangleright (\nu\, \mathbf{bn}(\mu) : \tilde{\mathrm{T}}) N \,|\, k_0[\![\textsc{succ}!\langle \mathbf{bn}(\mu)\rangle]\!]$; this lemma uses the preliminary lemma below, which we chose to extract as an important step of the proof.

Lemma 57 *Suppose δ, k_0 are fresh to the systems M, $k[\![P(X)]\!]$. Suppose also that $k \in \mathsf{C}$. Then:*

$$\begin{aligned} &\Sigma \models (\nu\, \tilde{n} : \tilde{\mathrm{T}})(M \,|\, k_1[\![P(\tilde{n})]\!] \,|\, k_2[\![Q(\tilde{n})]\!]) \;\cong \\ &\qquad (\nu\, \tilde{n} : \tilde{\mathrm{T}})(\nu\, \delta : \mathtt{ch})(\nu\, k_0 : \mathtt{loc}[\mathsf{a}, \mathsf{C}]) \left(\begin{array}{l} M \,|\, k_0[\![\delta!\langle \tilde{n}\rangle]\!] \\ \,|\, k_0[\![\delta?(X).\mathit{go}\, k_1.P(X) \,|\, \mathit{go}\, k_2.Q(X)]\!] \end{array} \right) \end{aligned}$$

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 58 (Extrusion) *Suppose* SUCC, FAIL, k_0 *are fresh to* Σ^M, Σ^N, *M and N. Then*

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

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

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

and show that $\mathcal{R}$ satisfies the defining properties of $\cong$. From reduction closure of $\cong$, it follows that $\mathcal{R}$ is also reduction closed. We here outline the proof for the barb preserving and contextuality properties.

To show barb preservation, we assume $\Sigma^M + \tilde{n} : \tilde{\mathrm{T}} \triangleright M \;\; \mathcal{R} \;\; \Sigma^N + \tilde{n} : \tilde{\mathrm{U}} \triangleright N$ and $\Sigma^M + \tilde{n}\!:\!\tilde{\mathrm{T}} \triangleright M \Downarrow_{a@l}$ and then 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 $\Sigma^M_+ \triangleright (\nu\, \tilde{n}\!:\!\tilde{\mathrm{T}})M|k_0[\![\mathrm{succ}!\langle\tilde{n}\rangle]\!] \Downarrow_{a@l}$, by barb preservation, $\Sigma^N_+ \triangleright (\nu\, \tilde{n} : \tilde{\mathrm{U}})N|k_0[\![\mathrm{succ}!\langle\tilde{n}\rangle]\!] \Downarrow_{a@l}$ 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 $\Sigma^M_+ \triangleright (\nu\, \tilde{n} : \tilde{\mathrm{T}})M|k_0[\![\mathrm{succ}!\langle\tilde{n}\rangle]\!] \Downarrow_{a@l}$, the restricted name a can be extruded via SUCC through the system:

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

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

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

From the definition of $\cong$, we know

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

and by barb preservation we conclude

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

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

The case for when $l \in \tilde{n}$ is similar, only that instead of T_a we use the system:

$$T_l \Leftarrow k_0[\![\mathrm{succ}?(X).(\nu\, k\!:\!(\mathbf{dom}({\Sigma^M_+}_O) \cup X_l))\mathsf{go}\; k.\mathsf{go}\; X_l.a?().\mathsf{go}\; k.\mathsf{go}\; k_0.\delta!\langle\rangle]\!]$$

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 56).

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 $(\Sigma^M + \tilde{n} : \tilde{\mathrm{T}}) \triangleright M \;\mathcal{R}\; (\Sigma^N + \tilde{n} : \tilde{\mathrm{U}}) \triangleright N$; we have to show that for arbitrary $k[\![P]\!]$ such that $(\Sigma^M + \tilde{n} : \tilde{\mathrm{T}}) \vdash_{\mathrm{O}} k[\![P]\!]$ then we have $(\Sigma^M + \tilde{n} : \tilde{\mathrm{T}}) \triangleright M \,|\, k[\![P]\!] \;\mathcal{R}\; (\Sigma^N + \tilde{n} : \tilde{\mathrm{U}}) \triangleright N \,|\, k[\![P]\!]$.

By definition of $\mathcal{R}$, we have $(\Sigma^M + \tilde{n} : \tilde{\mathrm{T}}) \triangleright M \;\mathcal{R}\; (\Sigma^N + \tilde{n} : \tilde{\mathrm{U}}) \triangleright N$ because

$$\Sigma^M_+ \triangleright (\nu\, \tilde{n} : \tilde{\mathrm{T}}) M | k_0[\![\textsc{succ}!\langle \tilde{n} \rangle]\!] \cong \Sigma^N_+ \triangleright (\nu\, \tilde{n} : \tilde{\mathrm{U}}) N | k_0[\![\textsc{succ}!\langle \tilde{n} \rangle]\!] \tag{79}$$

We define the system

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

where δ, k_0' are fresh names and $\mathsf{go}\,k.P\{^X\!/\!_{\tilde{n}}\}$ substitutes all occurrences of $\tilde{n}$ in $\mathsf{go}\,k.P$ by the appropriate variables $x_i \in X$. From $\Sigma^M + \tilde{n} : \tilde{\mathrm{T}} \vdash_{\mathrm{O}} k[\![P]\!]$ we deduce that $\Sigma^M_+ + \delta : \mathtt{ch} + k_0' : \mathtt{loc}[\mathsf{a}, \mathbf{dom}(\Sigma^M_{+\,O})] \vdash_{\mathrm{O}} T_{k[\![P]\!]}$ and subsequently, by contextuality of $\cong$ and (79), we obtain

$$\Sigma^M_{++} \triangleright M' \,|\, T_{k[\![P]\!]} \cong \Sigma^N_{++} \triangleright N' \,|\, T_{k[\![P]\!]} \tag{80}$$

where

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

From (80) and Proposition 50 we deduce that we can scope $\textsc{succ}$ and k_0 to obtain

$$\Sigma'_+ \triangleright (\nu\, \textsc{succ}, k_0) M' \,|\, T_{k[\![P]\!]} \cong \Sigma''_+ \triangleright (\nu\, \textsc{succ}, k_0) N' \,|\, T_{k[\![P]\!]} \tag{81}$$

where

$$\Sigma'_+ = \Sigma^M + \delta : \mathtt{ch} + k_0' : \mathtt{loc}[\mathsf{a}, \mathbf{dom}(\Sigma^M{}_O)] \tag{82}$$

$$\Sigma''_+ = \Sigma^N + \delta : \mathtt{ch} + k_0' : \mathtt{loc}[\mathsf{a}, \mathbf{dom}(\Sigma^N{}_O)] \tag{83}$$

By applying Lemma 57 on both sides of the equivalence and then through transitivity of $\cong$ e get

$$\Sigma'_+ \triangleright (\nu\, \tilde{n} : \tilde{\mathrm{T}}) M \,|\, k[\![P]\!] \,|\, k_0'[\![\delta!\langle \tilde{n} \rangle]\!] \cong \Sigma''_+ \triangleright (\nu\, \tilde{n} : \tilde{\mathrm{T}}) N \,|\, k[\![P]\!] \,|\, k_0'[\![\delta!\langle \tilde{n} \rangle]\!] \tag{84}$$

from which, by definition of $\mathcal{R}$, and by (82) and (83) we derive $\Sigma^M + \tilde{n} : \tilde{\mathrm{T}} \triangleright M|k[\![P]\!]\ \mathcal{R}\ \Sigma^N + \tilde{n} : \tilde{\mathrm{U}} \triangleright N|k[\![P]\!]$ as required.

Proposition 59 (Completeness)

$$\Sigma^1 \triangleright M_1 \cong \Sigma^2 \triangleright M_2 \ \ \textit{implies}\ \ \Sigma^1 \triangleright M_1 \approx \Sigma^2 \triangleright M_2$$

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 48. If μ is an external action, then by choosing Nm so that it contains all the free names in Σ^1_N (which is equal to Σ^2_N) and choosing fresh $\textsc{succ}$, $\textsc{fail}$, k_0, from the first part of Proposition 56 and the assumption $\Sigma^1 \triangleright M_1 \overset{\mu}{\longmapsto} \Sigma^1_2 + \mathbf{bn}(\mu) : \tilde{\mathrm{T}} \triangleright M_1'$ (we here rewrite Σ^1_1 as $\Sigma^1_2 + \mathbf{bn}(\mu) : \tilde{\mathrm{T}}$)we obtain

$$\Sigma^1_+ \triangleright M_1|T^\mu(Nm, \textsc{succ}, \textsc{fail}, k_0) \Longrightarrow \Sigma^1_{2+} \triangleright (\nu\, \mathbf{bn}(\mu) : \tilde{\mathrm{T}})M_1' \,|\, k_0[\![\textsc{succ}!\langle \mathbf{bn}(\mu)\rangle]\!]$$

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 $\Sigma^1_{1+} \triangleright (\nu\, \mathbf{bn}(\mu) : \tilde{\mathrm{T}})M_1' \,|\, k_0[\![\textsc{succ}!\langle \mathbf{bn}(\mu)\rangle]\!] \cong \Sigma \triangleright N$. 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 56 now gives that $\Sigma \triangleright N \equiv \Sigma^2_{1+} \triangleright (\nu\, \mathbf{bn}(\mu) : \tilde{\mathrm{T}})M_2' \,|\, k_0[\![\textsc{succ}!\langle \mathbf{bn}(\mu)\rangle]\!]$ for some Σ^2_{1+}, M_2' such that $\Sigma^2 \triangleright M_2 \overset{\mu}{\Longmapsto} \Sigma^2_1 + \mathbf{bn}(\mu) : \tilde{\mathrm{T}} \triangleright M_2'$. This is the required matching move, since the Extrusion Lemma 58, gives us the required

$$\Sigma^1_2 + \mathbf{bn}(\mu) : \tilde{\mathrm{T}} \triangleright M_1' \cong \Sigma^2_1 + \mathbf{bn}(\mu) : \tilde{\mathrm{T}} \triangleright M_2'$$

6 Conclusions

In this paper, we have extended an adaptation of Dπ [17] with an explicit representation of the underlying network, exhibiting both *node and link failures*. We have introduced a ping construct and adapted the migration construct ofDπ to provide a level of abstraction close an idealised form of the IP/ICMP layers in the Internet protocol suite [23]. We also encoded node status and connections as *type information* and then defined a reduction semantics to describe the behaviour of systems in the presence of node and link failures. Subsequently, we applied techniques for actions dependent on the observer's knowledge, developed for the π-calculus in [19] and Dπ in [18], to characterise a natural notion of barbed congruence. 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 and partial accessibility of nodes. In order to obtain this goal, the work also provided the following original contributions:

- A novel process calculus approach, encoding location status information (liveness and linkage) as types.
- A definition of contextual equivalence based on *partial-views* which evolve over the course of computation; these partial views are not set, as in [18], but may decrease through failure and increase through node scope extrusion.
- A corresponding bisimulation theory characterising this equivalence using a novel derived lts based on the notion of paths. The actions of the derived lts take into account not only the direct links that can be observed as part of a location scope extrusion, but also whole components of a network that are made accessible as a result.

We consciously chose to develop the theory in terms of a representation of nodes and links, despite the possible view that representation of nodes *only* is sufficient - this would typically entail encoding a link between location l and k as an intermediary node lk, encoding migration from l to k as a two-step migration from l to lk and lk to k, and finally encoding link failure as the intermediary node lk failing. There are various reasons for describing both node and link failure.

- For a start, network representation with partial connection between nodes is very natural in itself since WANs are often *not a clique*; programming for tolerating link failure is moreover subtly different from that for tolerating node failure, as shown in Example 10.
- Also, the resulting calculus also gives rise to an interesting theory of partial views, as shown in Examples 12, 13 and 22. We feel that these factors are a sufficient justification why link failure deserves to be investigated in its own right.
- We go further, and develop a setting that allows us to study *directly* the interplay between node and link failure and their respective observation from the software's point of view.
- Finally, we forgo the option of encoding link failure because it is unlikely that a theory resulting from an encoding into a *nodes only* calculus would be fully abstract, due to the fact that any encoding would typically decompose atomic reductions such as migration into sub-reductions, which in turn affects the resulting bisimulation equivalence; see [15].

The extended abstract for this work was presented at [12] and all the detailed proofs appeared already as part of the first author's thesis[11]. To the best of our knowledge, this is the first body of work that studies system behaviour in the presence of both *permanent node and link* failure in a unified setting, investigates the resulting natural notion of *partial views* arising in this setting and characterises the partial-view contextual equivalence through an lts based on accessibility paths.

Related Work: There have been a number of studies on process behaviour in the presence of *permanent node failure* only, amongst them [26], our point of departure. In this work, they developed bisimulation techniques for a distributed variant of CCS with location failure. Our work is also very close to the pioneering work [2,1]; their approach to developing reasoning tools is however 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 these translations.

Partial connections between locations have been studied in [8,6,9,7] where distributed Linda-like programs are equipped with connect, co-connect and disconnect software primitives that dynamically change the accessibility of locations. This body of work addresses numerous issue such as the choice of barbs in a setting of partial connections. In their latest work[9], they describe an observational equivalence yielding a notion of partial view which is very similar to ours and give a bisimulation equivalence which characterise the observational equivalence with partial views. Despite these commonalities, their work differs from ours in many respects. Their interpretation of connections is different from ours, since their aim is to program with these constructs as one would do at a TCP layer of abstraction[23], establishing connections between two locations for remote communications and disconnecting afterwards; we do not attempt to program with our break construct and rather apply breaks (and kills) non-deterministically to model permanent failure. Their model of computation is based on tuple-spaces rather than channel communication and the network information, such as existing links, is described at the system level instead of being encoded as type information, as in our case. Most importantly though, their solution for the bisimulation characterisation of their observational equivalence is different from ours. In particular, they employ separate, simpler labels for location scope extrusion and individual link discovery. Consequently, their bisimulations *disentangle* scope extrusion from the discovery of newly accessible nodes that result from the scope extrusion. This separation turns out to give a much simpler and more elegant completeness proof than the one in Section 5.2. We however believe that it is natural to keep together information relating name extrusion and the new network accessible as a result of the extrusion. Moreover, the bisimulations resulting from our labels batch multiple related transitions and intermediate states (scope extrusion and multiple link discoveries) under one single transition. Our rationale has thus been to employ labels carrying more information (as types) and incur more complication in proving the correctness of our lts but then have an lts that permits smaller bisimulations than the ones with otherwise simpler labels.

Another work dealing with partial connections is [24], whereby they describe a process calculus with broadcasts which are subject to partial connections between sites. The emphasis of this work is to develop static analysis for proving the correctness of routing protocols for ad-hoc networks. Even though they give an lts for this calculus with partial connections, they do not study any equivalence properties for the bisimulation arising from this lts. More crucially, their calculus does not express any scoping of location names, the scope extrusion of which posed the main difficulty in establishing sound and complete equivalence theories for distributed calculi with partial connections in our case.

On a technical level, our work is also considerably different from [18], even though our theory may be seen as an extension of theirs; in this work, migration permissions give rise to a sort of unidirectional links. For a start, in [18] they assume that there is always an observable location giving migration rights to every other location, which effectively links all locations in one direction to such a location (whether observable or non-observable). This is fundamentally different from our setting where, at most, new context locations can be linked to presently accessible locations only. This also impacts their framework in more than one way. Whereas their notion of contextual equivalence is based on a fixed set of accessible locations, our equivalence assumes a dynamic set of accessible locations, which changes through failure and scope extrusion. More importantly though, while they could obtain a characterising lts whose labels are based directly on the fixed accessible set, we required a more complex derived lts at the level of *accessibility paths*. More concretely, in [18] a location's accessibility depended solely on the type at which it is scope extruded; in our case, a location's accessibility also depends, in an indirect manner, on the scope extrusion of subsequent locations that may yield an accessibility path to it. All of this is further complicated in our setting by dynamic failure, which changes the state of the underlying network during execution; in [18] migration permissions are never revoked.

Elsewhere, permanent location failure with hierarchical dependencies have been studied by Fournet *et al* [10]. Berger [3] was the first to study a π-calculus extension that models *transient* location failure with persistent code and communication failures, while Nestmann *et al* [25] employ a tailor-made process calculus to study standard results in distributed systems, such as [5].

Future Work: Our study is far from conclusive; rather than being a body of work that could be directly applied to real case scenarios, we believe that this 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 $\mathsf{ping}\ l.P\ \mathsf{else}\ Q$ 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. More generally, one could allow the *recovery* of faults, in which dead nodes or broken links may randomly be restored; transient faults are also directly related to issues such as *persistence* and *volatility* of code. As we stated often, we never intended to program with location and link failure but where more interested in applying their effects on computation in a non-deterministic way. A whole area of research that is still relatively unexplored in process calculi is that of attaching probabilities to failures; our framework can be seen as an ideal starting point for such work. One further avenue worth exploring is how partial links between locations and partial-view equivalence can be adapted to asses the fault-tolerance of a system[13]. 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 fault recovery and persistence would probably fall into the latter; higher-order theories of Dπ may need to be considered in the latter case.

The expressivity of the present calculus warrant further investigation. The graph structure imposed on DπF locations should also be flexible enough to express other location structures as instances of the calculus. For instance, a hierarchical location structure such as that used in the distributed join-calculus can be elegantly encoded in DπF by imposing restrictions on the starting graph structure and the types of the new locations to be created. Moreover, by restricting the observer's view to the root nodes of this encoding, we can also encode the failure of a subtree as the breaking of the link connecting the root of the subtree to the remainder of the tree.

Finally we hope that some extended form of our framework can be used to study distributed algorithms in the style of [14], where distributed computation needs to be aware of the *dynamic* computing context in which it is executing; various examples can be drawn from ad-hoc networks, embedded systems and generic routing software; see [24] for some examples. In these settings, the software typically *discovers* new parts of the neighbouring computing environment at runtime, but this often does not entail the accessibility of this environment. This separation between discovery and eventual accessibility is naturally expressed in our calculus. Our framework also handles the reverse, that is, remote resources that are known and accessible, but eventually become inaccessible through failure; in such a setting, we study the power of network observation mechanisms used to *update* the knowledge of the changes at the current underlying network caused by failure, enabling the discovery of alternative routes to remote resources.

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* 26th *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] R. De Nicola, D. Gorla, and R. Pugliese. Global computing in a dynamic network of tuple spaces. In J.M. Jacquet and G.P. Picco, editors, *Proc. of 7th International Conference on Coordination Models and Languages (COORDINATION 2005)*, volume 3454 of *LNCS*, pages 157–172. Springer, 2005.

[7] R. De Nicola, D. Gorla, and R. Pugliese. Global computing in a dynamic network of tuple spaces. *Science of Computer Programming*, 64(2):187–204, 2007.

[8] Rocco De Nicola, Daniele Gorla, and Rosario Pugliese. Basic observables for a calculus for global computing. In L. Caires et al., editor, *Proc. of 32nd International Colloquium on Automata, Languages and Programming (ICALP 2005)*, volume 3580 of *LNCS*, pages 1226–1238. Springer, 2005.

[9] Rocco De Nicola, Daniele Gorla, and Rosario Pugliese. Basic observables for a calculus for global computing. *Information and Computation*, 205(10):1491–1525, 2007.

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

[11] Adrian Francalanza. *A Study of Failure in a Distributed Pi-calculus*. PhD thesis, University of Sussex, 2006.

[12] Adrian Francalanza and Matthew Hennessy. A theory of system behaviour in the presence of node and link failures. In *CONCUR*, volume 3653 of *Lecture Notes in Computer Science*, pages 368–382. Springer, 2005.

[13] Adrian Francalanza and Matthew Hennessy. A theory of system fault tolerance. In L. Aceto and A. Ingolfsdottir, editors, *Proc. of 9th Intern. Conf. on Foundations of Software Science and Computation Structures (FoSSaCS'06)*, volume 3921 of *LNCS*, pages 16–31. Springer, 2006.

[14] Adrian Francalanza and Matthew Hennessy. A fault tolerance bisimulation proof for consensus. In Rocco De Nicola, editor, *16th European Symposium on Programming (ESOP'07)*, volume 4421 of *LNCS*, pages 395–410. Springer, March 2007.

[15] Rob. van Glabbeek and Ursula Goltz. Equivalence notions for concurrent systems and refinement of actions (extended abstract). In A. Kreczmar and G. Mirkowska, editors, Proceedings 14th 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.

[16] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. *The Java Language Specification Second Edition*. Addison-Wesley, Boston, Mass., 2000.

[17] Matthew Hennessy. *A Distributed Pi-calculus*. Cambridge University Press, 2007.

[18] 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.

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

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

[21] Kohei Honda and Martin Berger. The two-phase commitment protocol in an extended pi-calculus. In Luca Aceto and Björn Victor, editors, *EXPRESS00: 7th International Workshop on Expressiveness in Concurrency*, volume 39, pages 105–130, Amsterdam, The Netherlands, 2000. Elsevier.

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

[23] Martin W. Murhammer, Orcun Atakan, Stefan Bretz, Larry R. Pugh, Kazunari Suzuki, and David H. Wood. *TCP/IP Tutorial and Technical Overview*. IBM Redbooks. International Technical Support Organization, 8 edition, December 2006.

[24] Sebastian Nanz and Chris Hankin. A framework for security analysis of mobile wireless networks. *Theoretical Computer Science*, 367(1-2):203–227, 2006.

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

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

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

A Auxilliary Definitions

We here define the standard notions of free/bound variable and names for DπF. We recall from Section 2 that V and X denote respectively tuples of identifiers $(u_1, \ldots, u_n)$ and tuples of distinct variables $(x_1, \ldots, x_n)$. We define the functions **names**(V) and **vars**(V) which extract names and variables respectively from identifier tuples:

$$\mathbf{names}((u_1, \ldots, u_n)) \stackrel{\text{def}}{=} \{n \mid n = u_i\} \qquad \mathbf{vars}((u_1, \ldots, u_n)) \stackrel{\text{def}}{=} \{x \mid x = u_i\}$$

We also abuse notation and use $\{X\}$ to mean

$$\{(x_1, \ldots, x_n)\} \stackrel{\text{def}}{=} \{x \mid x = x_i\}$$

We overload the functions for free name $\mathbf{fn}(-)$ and for free variables $\mathbf{fv}(-)$ to range over types, open processes and open systems as follows:

$$\mathbf{fn}(\mathsf{ch}) \stackrel{\text{def}}{=} \emptyset \qquad \mathbf{fv}(\mathsf{ch}) \stackrel{\text{def}}{=} \emptyset$$

$$\mathbf{fn}(\mathsf{loc}[\mathsf{S}, \mathsf{C}]) \stackrel{\text{def}}{=} \{n \mid n \in \mathsf{C}\} \qquad \mathbf{fv}(\mathsf{loc}[\mathsf{S}, \mathsf{C}]) \stackrel{\text{def}}{=} \{x \mid x \in \mathsf{C}\}$$

$$\mathbf{fn}(u!\langle V\rangle.P) \stackrel{\text{def}}{=} \mathbf{fn}(P) \cup \mathbf{names}(V) \cup \begin{cases} \{u\} & \text{if } u \in \textsc{Names} \\ \emptyset & \text{othewise} \end{cases}$$

$$\mathbf{fv}(u!\langle V\rangle.P) \stackrel{\text{def}}{=} \mathbf{fv}(P) \cup \mathbf{vars}(V) \cup \begin{cases} \{u\} & \text{if } u \in \textsc{Vars} \\ \emptyset & \text{otherwise} \end{cases}$$

$$\mathbf{fn}(u?(X).P) \stackrel{\text{def}}{=} \mathbf{fn}(P) \cup \begin{cases} \{u\} & \text{if } u \in \textsc{Names} \\ \emptyset & \text{othewise} \end{cases}$$

$$\mathbf{fv}(u?(X).P) \stackrel{\text{def}}{=} \left(\mathbf{fv}(P) \cup \begin{cases} \{u\} & \text{if } u \in \textsc{Vars} \\ \emptyset & \text{otherwise} \end{cases} \right) \setminus \{X\}$$

$$\mathbf{fn}(*P) \stackrel{\text{def}}{=} \mathbf{fn}(P) \qquad \mathbf{fv}(*P) \stackrel{\text{def}}{=} \mathbf{fv}(P)$$

$$\mathbf{fn}(P|Q) \stackrel{\text{def}}{=} \mathbf{fn}(P) \cup \mathbf{fn}(Q) \qquad \mathbf{fv}(P|Q) \stackrel{\text{def}}{=} \mathbf{fv}(P) \cup \mathbf{fv}(Q)$$

$$\mathbf{fn}((\nu\, n\!:\!\mathrm{T})P) \stackrel{\text{def}}{=} \mathbf{fn}(\mathrm{T}) \cup \mathbf{fn}(P) \setminus \{n\} \qquad \mathbf{fv}((\nu\, n\!:\!\mathrm{T})P) \stackrel{\text{def}}{=} \mathbf{fv}(\mathrm{T}) \cup \mathbf{fv}(P)$$

$$\mathbf{fn}(\mathbf{0}) \stackrel{\text{def}}{=} \emptyset \qquad \mathbf{fv}(\mathbf{0}) \stackrel{\text{def}}{=} \emptyset$$

$$\mathbf{fn}(\mathsf{go}\ u.P) \stackrel{\text{def}}{=} \mathbf{fn}(P) \cup \begin{cases} \{u\} & \text{if } u \in \textsc{Names} \\ \emptyset & \text{othewise} \end{cases}$$

$$\mathbf{fv}(\mathsf{go}\ u.P) \stackrel{\text{def}}{=} \mathbf{fv}(P) \cup \begin{cases} \{u\} & \text{if } u \in \textsc{Vars} \\ \emptyset & \text{otherwise} \end{cases}$$

$$\mathbf{fn}(\textsf{kill}) \overset{\text{def}}{=} \emptyset \qquad\qquad \mathbf{fv}(\textsf{kill}) \overset{\text{def}}{=} \emptyset$$

$$\mathbf{fn}(\textsf{break}\ u) \overset{\text{def}}{=} \begin{cases} \{u\} & \text{if } u \in \textsc{Names} \\ \emptyset & \text{othewise} \end{cases}$$

$$\mathbf{fv}(\textsf{break}\ u) \overset{\text{def}}{=} \begin{cases} \{u\} & \text{if } u \in \textsc{Vars} \\ \emptyset & \text{otherwise} \end{cases}$$

$$\mathbf{fn}(\textsf{if}\ u = v\ \textsf{then}\ P\ \textsf{else}\ Q) \overset{\text{def}}{=} \mathbf{fn}(P) \cup \mathbf{fn}(Q) \cup \{n \mid n \in \{u, v\}\}$$

$$\mathbf{fv}(\textsf{if}\ u = v\ \textsf{then}\ P\ \textsf{else}\ Q) \overset{\text{def}}{=} \mathbf{fv}(P) \cup \mathbf{fv}(Q) \cup \{x \mid x \in \{u, v\}\}$$

$$\mathbf{fn}(\textsf{ping}\ u.P\ \textsf{else}\ Q) \overset{\text{def}}{=} \mathbf{fn}(P) \cup \mathbf{fn}(Q) \cup \begin{cases} \{u\} & \text{if } u \in \textsc{Names} \\ \emptyset & \text{othewise} \end{cases}$$

$$\mathbf{fv}(\textsf{ping}\ u.P\ \textsf{else}\ Q) \overset{\text{def}}{=} \mathbf{fv}(P) \cup \mathbf{fv}(Q) \cup \begin{cases} \{u\} & \text{if } u \in \textsc{Vars} \\ \emptyset & \text{otherwise} \end{cases}$$

$$\mathbf{fn}(l[\![P]\!]) \overset{\text{def}}{=} \mathbf{fn}(P) \cup \{l\} \qquad\qquad \mathbf{fv}(l[\![P]\!]) \overset{\text{def}}{=} \mathbf{fv}(P)$$

$$\mathbf{fn}(N|M) \overset{\text{def}}{=} \mathbf{fn}(N) \cup \mathbf{fn}(M) \qquad\qquad \mathbf{fv}(N|M) \overset{\text{def}}{=} \mathbf{fv}(N) \cup \mathbf{fv}(M)$$

$$\mathbf{fn}((\nu\, n : \mathtt{T})M) \overset{\text{def}}{=} \mathbf{fn}(\mathtt{T}) \cup \mathbf{fn}(M) \setminus \{n\} \qquad \mathbf{fv}((\nu\, n : \mathtt{T})M) \overset{\text{def}}{=} \mathbf{fv}(\mathtt{T}) \cup \mathbf{fv}(M)$$

We also overload the bound name and bound variables functions, $\mathbf{bn}(-)$ and $\mathbf{bv}(-)$, to range over open processes and systems:

$$\mathbf{bn}((\nu\, n : \mathtt{T})P) \overset{\text{def}}{=} \{n\} \cup \mathbf{bn}(P)$$

$$\mathbf{bn}(\mathbf{0}) = \mathbf{bn}(\textsf{kill}) = \mathbf{bn}(\textsf{break}\ u) \overset{\text{def}}{=} \emptyset$$

$$\mathbf{bn}(u!\langle V\rangle.P) = \mathbf{bn}(u?(X).P) = \mathbf{bn}(*P) = \mathbf{bn}(\textsf{go}\ u.P) \overset{\text{def}}{=} \mathbf{bn}(P)$$

$$\mathbf{bn}(P|Q) = \mathbf{bn}(\textsf{if}\ u = v\ \textsf{then}\ P\ \textsf{else}\ Q) = \mathbf{bn}(\textsf{ping}\ u.P\ \textsf{else}\ Q) \overset{\text{def}}{=} \mathbf{bn}(P) \cup \mathbf{bn}(Q)$$

$$\mathbf{bv}(u?(X).P) \overset{\text{def}}{=} \{X\} \cup \mathbf{bv}(P)$$

$$\mathbf{bv}(\mathbf{0}) = \mathbf{bv}(\textsf{kill}) = \mathbf{bv}(\textsf{break}\ u) \overset{\text{def}}{=} \emptyset$$

$$\mathbf{bv}(u!\langle V\rangle.P) = \mathbf{bv}(*P) = \mathbf{bv}(\textsf{go}\ u.P) = \mathbf{bv}((\nu\, n : \mathtt{T})P) \overset{\text{def}}{=} \mathbf{bv}(P)$$

$$\mathbf{bv}(P|Q) = \mathbf{bv}(\textsf{if}\ u = v\ \textsf{then}\ P\ \textsf{else}\ Q) = \mathbf{bv}(\textsf{ping}\ u.P\ \textsf{else}\ Q) \overset{\text{def}}{=} \mathbf{bv}(P) \cup \mathbf{bv}(Q)$$

$$
\begin{aligned}
\mathbf{bn}(l[\![P]\!]) &\overset{\text{def}}{=} \mathbf{bn}(P) & \mathbf{bv}(l[\![P]\!]) &\overset{\text{def}}{=} \mathbf{bv}(P)\\
\mathbf{bn}((\nu\, n\colon \mathtt{T})P) &\overset{\text{def}}{=} \{n\} \cup \mathbf{bn}(P) & \mathbf{bv}((\nu\, n\colon \mathtt{T})P) &\overset{\text{def}}{=} \mathbf{bv}(P)\\
\mathbf{bn}(M|N) &\overset{\text{def}}{=} \mathbf{bn}(M) \cup \mathbf{bn}(N) & \mathbf{bv}(M|N) &\overset{\text{def}}{=} \mathbf{bv}(M) \cup \mathbf{bv}(N)
\end{aligned}
$$