A Theory of System Behaviour in the Presence of Node and Link Failures (Extended Abstract)

Adrian Francalanza[1] and Matthew Hennessy[1]

University of Sussex, Falmer Brighton BN1 9RH, England,
{adrianf,matthewh}@sussex.ac.uk

Abstract. We develop a behavioural theory of distributed programs (systems) 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 node 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.

1 Introduction

It is generally accepted that *partial failures* are one of the principal factors 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 and programming languages have arisen in the literature to model the behaviour of distributed programs in the presence of failures, and to study the correctness of algorithms is such a setting. The purpose of this paper is to:

- invent 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 [12], a simple distributed version of the standard π-calculus [15, 18], 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 Dπ abstract server implementations as motivation:

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

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

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

The three systems server, servD and servD2Rt implement a lookup server that accepts a single request for a lookup on channel *req* at location l with two arguments, x being the value to be looked up and y being the return channel on which to return the required information. A typical client for these servers would have the following form, sending the name l as the value to be looked up and *ret* as the return channel:

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

Every server forwards the request to an internal database hidden from the client, denote by the scoped channel *data*, which looks up 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; by contrast, servD and 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; 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.[1]. 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, servDmay block and not service 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 [10].

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, being software executing in a distributed manner

[1] Here the construct go $l,\ k.P$ is shorthand for go l.go k.P

over Σ. Here Σ records the set of nodes in the network, their *status*, that is whether they are *alive* or *dead*, and their *connectivity*, that is the set of (symmetric) links between these nodes. On the other hand, N will be more or less a standard system description from Dπ, augmented with a conditional construct for reacting to network failures. We believe that this results in a succinct but expressive framework, in which many of the phenomena associated with practical distributed settings, such as routing algorithms and ad-hoc network discover, can be examined.

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

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

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 break 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, δ, 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, but at the same time, cannot determine the state of such nodes and the code executing at them because they are not able to *reach* them. This may happen either because the newly discovered nodes are completely disconnected or else 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.

We choose to develop the theory in terms of the calculus itself, despite the widely held 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. We believe that a calculus with partial connection between nodes is very natural in itself since WANs are often *not* a clique. This calculus also gives rise to an interesting theory of partial views that we believe deserves to be investigated in its own right. In addition, we would also like to explore the interplay between node and link failure and their respective observation from the software's point of view. With this in mind, we anticipate that any such encoding would be cumbersome to use and the corresponding theory of partial views would be too complicated to develop. Moreover, it is unlikely that this resultant theory would be fully abstract, due to the fact that any encoding would typically decomposes atomic reductions such as migration into sub-reductions, which in turn affects the resulting bisimulation equivalence; see [9].

The paper is organised as follows: Section 2 introduces DπF and the reduction semantics. In Section 3 we present an initial definition of actions for DπF, based on the

Table 1. *Syntax of typed DπF*

Types

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

Processes

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

Systems

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

general approach of [11]. The resulting bisimulation equivalence can be used to demonstrate equivalencies 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 show that the resulting equivalence is *fully abstract* with respect to an intuitive form of *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. The complete proofs, elaborate discussions and extensive examples may be found in the corresponding technical report [8].

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 Figure 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*, $(\nu\, n : \mathtt{T})N$. Although we could employ the full power of the type system for Dπ [10], for simplicity, we use a very simple notion of type and adapt it the purpose at hand. Thus, if n is used as a channel in N, then $\mathtt{T}$ is simply $\mathtt{ch}$; however if it is a location then $\mathtt{T} = \mathtt{loc}_{\mathtt{A}}[\mathtt{L}]$ 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, $\{l_1, \ldots, l_n\}$.

The syntax for agents, P, Q, is an extension of that in Dπ. There is input and output on channels; here V is a tuple of identifiers, and X a tuple of 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. We also introduce a ping conditional construct, $l[\![\mathsf{ping}\, k.P\lceil Q\rceil]\!]$, in the style of [2, 1, 17], branching to $l[\![P]\!]$ or $l[\![Q]\!]$ depending on the *accessibility* of k from l. Finally we have two new constructs to simulate failures; $l[\![\mathsf{kill}]\!]$ 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 account the effects of *dynamic* failures.

In this extended abstract, we will assume the standard notions of *free* and *bound* occurrences of both names and variables, together with the associated concepts of α-conversion and *substitution*. Furthermore, we will assume that all system terms are *closed*, that is they have no free occurrences of variables.

Reduction semantics: This takes the form of a binary relation

$$\Delta \triangleright N \longrightarrow \Delta' \triangleright N' \tag{2}$$

where Δ and Δ' are representations of the state of the network. Intuitively this must record the set of locations in existence, whether they are alive or dead, and the existence of any links between them.

Definition 1 (Network representation). *We first introduce some notation to represent the* links *in a network. A binary relation $\mathcal{L}$ over locations is called a* link set *if it is:*

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

The latter property allows the smooth handling of the degenerate case of a process moving from a site l to l itself. Also, for any linkset $\mathcal{L}$, we let $\mathbf{dom}(\mathcal{L})$ denote its domain: that is the collection of locations l such that $\langle l, l\rangle \in \mathcal{L}$. In the sequel, we also use the abbreviation $l \leftrightarrow k$ in link sets to denote the pairs $\langle l, l\rangle$, $\langle k, k\rangle$, $\langle l, k\rangle$, $\langle k, l\rangle$.

A network representation *Δ is any triple*[2] *$\langle \mathcal{N}, \mathcal{D}, \mathcal{L}\rangle$ where*

- *$\mathcal{N}$ is a set of names, as before; we now use $\mathbf{loc}(\mathcal{N})$ to represent the subset of $\mathcal{N}$ which are locations*
- *$\mathcal{D} \subseteq \mathbf{loc}(\mathcal{N})$ represents the set of dead locations, as before.*
- *$\mathcal{L} \subseteq \mathbf{loc}(\mathcal{N}) \times \mathbf{loc}(\mathcal{N})$ represents the set of connections between locations*

So we may take Δ and Δ' in (2) above to be simple network representations. Formally we call pairs $\Delta \triangleright N$ *configurations*, whenever every free name in N occurs in the name component of Δ, and we define reductions to take place between such configurations. Since not all nodes are interconnected, the reduction semantics is based on the notions of *accessibility* and *reachability* between nodes: k is accessible from l in Δ, denoted as $\Delta \vdash k \leftarrow l$, if and only if k is alive and there is a (direct) live link between l and k; a node k is *reachable* from l in Δ, denotes as $\Delta \vdash k \leftrightsquigarrow l$, if there exists a *chain of live links* between the two nodes, where *every intermediate node is alive*. We refer the reader to the Appendix for the formal definitions.

For convenience, the rules governing these reductions are given in the three separate figures. These rely on certain notation for checking the state of nodes and links in a network and of updating the network; once again, we refer the reader to the Appendix for the formal definitions of the notation used.

[2] In this definition, we only represent live links in Δ and omit dead links; it turns out that the latter are never used. Nevertheless, the representation can easily be extended to represent dead links by adding another linkset, say $\overline{\mathcal{L}}$.

Table 2. *Local Reduction Rules for DπF*

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

(r-comm)

$$\frac{}{\Delta \triangleright l[\![a!\langle V\rangle.P]\!] \mid l[\![a?(X).Q]\!] \quad\longrightarrow\quad \Delta \triangleright l[\![P]\!] \mid l[\![Q\{^{V}\!/\!_{X}\}]\!]}$$

(r-rep)

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

(r-fork)

$$\frac{}{\Delta \triangleright l[\![P|Q]\!] \quad\longrightarrow\quad \Delta \triangleright l[\![P]\!] \mid l[\![Q]\!]}$$

(r-eq)

$$\frac{}{\Delta \triangleright l[\![\mathsf{if}\ u = u.P\lceil Q\rceil]\!] \longrightarrow \Delta \triangleright l[\![P]\!]}$$

(r-neq)

$$\frac{}{\Delta \triangleright l[\![\mathsf{if}\ u = v.P\lceil Q\rceil]\!] \longrightarrow \Delta \triangleright l[\![Q]\!]}\ u \neq v$$

Table 3. *Network Reduction Rules for DπF*

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

(r-go)

$$\frac{}{\Delta \triangleright l[\![\mathsf{go}\ k.P]\!] \longrightarrow \Delta \triangleright k[\![P]\!]}\ \Delta \vdash k \leftarrow l$$

(r-ngo)

$$\frac{}{\Delta \triangleright l[\![\mathsf{go}\ k.P]\!] \longrightarrow \Delta \triangleright k[\![\mathbf{0}]\!]}\ \Delta \nvdash k \leftarrow l$$

(r-ping)

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

(r-nping)

$$\frac{}{\Delta \triangleright l[\![\mathsf{ping}\ k.P\lceil Q\rceil]\!] \longrightarrow \Delta \triangleright l[\![Q]\!]}\ \Delta \nvdash k \leftarrow l$$

(r-kill)

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

(r-brk)

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

(r-newc)

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

(r-newl)

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

The first set of rules, in Figure 2, give the standard rules for (local) communication, and the management of replication, matching and parallelism, and are derived from the corresponding rules for Dπ in [12]. But note that they are all depend on the requirement that l, the location of the activity, is currently alive; this is the intent of the predicate $\Delta \vdash l : \textbf{alive}$.

The second set, in Figure 3, is more interesting. Rules (r-go) and (r-ngo) state that a migration is successful depending on the accessibility of the destination. Similarly, (r-ping) and (r-nping) are subject to the same condition for the respective branchings. Note that $l[\![\mathsf{ping}\ k.P\lceil Q\rceil]\!]$ yields *partial information* about the state of the underlying network: 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 breaking of the link $l \leftrightarrow k$, or both. The rules (r-kill), (r-brk) make the obvious changes to the current network; $\Delta - l$ means changing l to be a dead site in Δ, while $\Delta - l \leftrightarrow k$ means breaking the link between l and k. Finally (r-newc) and (r-newl) regulates the generation of new names;

Table 4. *Contextual Reduction Rules for DπF*

(r-str)

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

(r-ctxt-rest)

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

(r-ctxt-par)

$$\frac{\Delta \triangleright N \quad \longrightarrow \quad \Delta' \triangleright N'}{\begin{array}{l} \Delta \triangleright N|M \quad \longrightarrow \quad \Delta' \triangleright N'|M \\ \Delta \triangleright M|N \quad \longrightarrow \quad \Delta' \triangleright M|N' \end{array}} \; \Delta \vdash M$$

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

for example, (r-newl) launches a new location with a declared type $\mathtt{loc}_{\mathtt{S}}[\mathtt{C}]$ using the function $\mathsf{inst}(\mathtt{loc}_{\mathtt{S}}[\mathtt{C}], l, \Delta)$. Intuitively, this functions returns the location type $\mathtt{loc}_{\mathtt{S}}[\mathtt{D}]$, where the set of locations $\mathtt{D}$, is the subset of locations in $\mathtt{C}\cup\{l\}$ which are *reachable* from l. We refer the reader to the technical report, [8], for an example explaining how this function works.

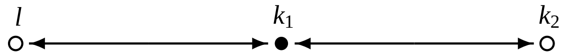
Finally, in Figure 4 we have an adaptation of the standard *contextual* rules, which allow the basic reductions to occur in *evaluation contexts*. The rule (r-str) allows reductions up to a structural equivalence, in the standard manner, using the identities in Figure 5. The only non-trivial identities in Figure 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 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 $\mathtt{U}$.

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 1 (Syntax). Let Δ represent the network $\langle\{l, a\}; \emptyset; \{l \leftrightarrow l\}\rangle$ consisting of a channel a and a live node l and M_1 the system

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

Here M_1 generates 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 network by



where the nodes ∘ and • denote live and dead nodes respectively. Note that the same network could be denoted by the system N_1

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

Note also that the two systems are structurally equivalent, $M_1 \equiv N_1$, through (s-flip-2). As a notational abbreviation, in all future example we will omit the status annotation, a, in location declarations; so for example system N_1 would be given as

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

3 A Labelled transition system

In this section we give a labelled transition system for the language, in which the labelled actions are intended to mimic the possible interactions between a system and an observer; it is natural to assume that both share the same underlying network. However this first example demontrates that our representation of this joint network is no longer sufficient, if we want to faithfully record the effect interactions have on systems because they may lead to a discrepancy between the *system network view* and the *observer network view*.

Example 2 (Observer's Network view). Let Δ and M_1 be defined as in Example 1. An observer O at site l, such as $l[\![a?(x).P(x)]\!]$, can gain knowledge of the new location k_2, thereby evolving to $l[\![P(k_2)]\!]$. But even though it is in possession of the name k_2, it's knowledge of the state of the underlying network is no longer represented by Δ, and there is now a mismatch between the observes view of the network, and the systems view. The system view is now $\Delta' = \langle\{a, l, k_2\}; \emptyset; \{l \leftrightarrow l,\ k_2 \leftrightarrow k_2\}\rangle$, that is Δ augmented by the scope extrusion of the *live* node k_2 linked to a private (dead) node k_1, which is, in turn, linked to l. But the observer's view is quite different: the node l is accessible to the observer, since it has code running there; nevertheless, even though the observer knows about k_2 at l in $P(k_2)$, it does not have enough information to *reach* k_2 from l. As a result, it has no means how to determine k_2's state in terms of its status and connections nor interact with any code at k_2. This means that the representation of the observers view, requires a new kind of annotation, for nodes such as k_2 which are known, but not accessible

Stated otherwise, in order to give an lts semantics, we need to refine our representations of networks.

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

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

(l-out)

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

(l-in)

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

(l-in-rep)

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

(l-fork)

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

(l-eq)

$$\frac{}{\Sigma \triangleright l[\![\text{if } u = u.P\lceil Q\rceil]\!] \xrightarrow{\tau} \Sigma \triangleright l[\![P]\!]}$$

(l-neq)

$$\frac{}{\Sigma \triangleright l[\![\text{if } u = v.P\lceil Q\rceil]\!] \xrightarrow{\tau} \Sigma \triangleright l[\![Q]\!]}\ u \neq v$$

Definition 2 (Effective network representations). *An* effective network representation *Σ is a triple $\langle \mathcal{N}, \mathcal{O}, \mathcal{H}\rangle$, where:*

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

We also assume three consistency requirements: (i) $\textbf{dom}(\mathcal{O}) \subseteq \textbf{loc}(\mathcal{N})$*,* (ii) $\textbf{dom}(\mathcal{H}) \subseteq \textbf{loc}(\mathcal{N})$ *and* (iii) $\textbf{dom}(\mathcal{O}) \cap \textbf{dom}(\mathcal{H}) = \emptyset$*.*

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

With this refined notion, we can now represent the observers view of Example 2 as $\mathcal{N} = \{l, k_2\}$, $\mathcal{O} = \{l \leftrightarrow l\}$ and $\mathcal{H} = \{k_2 \leftrightarrow k_2\}$. In the sequel, we will use *configurations* of the form $\Sigma \triangleright N$, where Σ is a network representation, and N satisfies the obvious consistency constraints with respect to it.

We now define a labelled transition system for DπF, which consists of a collection of actions over configurations, $\Sigma \triangleright N \xrightarrow{\mu} \Sigma' \triangleright N'$, defined by the transition rules in Figures 6, 7 and 8, where μ can be an internal action, τ, a bound input, $(\tilde{n} : \tilde{\mathrm{T}})l : a?(V)$

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)

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

(l-halt)

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

(l-disc)

$$\frac{}{\Sigma \triangleright N \xrightarrow{l \nleftrightarrow k} \Sigma - (l \leftrightarrow k) \triangleright N}\ \Sigma \vdash_{\mathsf{obs}} l \leftrightarrow k$$

(l-go)

$$\frac{}{\Delta \triangleright l[\![\mathsf{go}\ k.P]\!] \xrightarrow{\tau} \Delta \triangleright k[\![P]\!]}\ \Delta \vdash k \leftarrow l$$

(l-ngo)

$$\frac{}{\Delta \triangleright l[\![\mathsf{go}\ k.P]\!] \xrightarrow{\tau} \Delta \triangleright k[\![\mathbf{0}]\!]}\ \Delta \nvdash k \leftarrow l$$

(l-ping)

$$\frac{}{\Delta \triangleright l[\![\mathsf{ping}\ k.P\lceil Q \rceil]\!] \xrightarrow{\tau} \Delta \triangleright l[\![P]\!]}\ \Delta \vdash k \leftarrow l$$

(l-nping)

$$\frac{}{\Delta \triangleright l[\![\mathsf{ping}\ k.P\lceil Q \rceil]\!] \xrightarrow{\tau} \Delta \triangleright l[\![Q]\!]}\ \Delta \nvdash k \leftarrow l$$

(l-newc)

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

(l-newl)

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

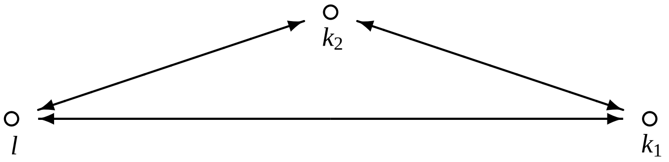
or bound output, $(\tilde{n} : \tilde{\mathtt{T}})l : a!\langle V \rangle$, adopted from [11, 10], or the new labels, kill $: l$ and $l \nleftrightarrow k$, denoting external location killing and link breaking respectively. In this extended abstract we refrain from commenting on the definition of these actions and refer the full paper, [8]. We only highlight the fact that the transition rules introducing external actions such as (l-out), (l-in), (l-halt) and (l-disc) are subject to judgements of the form $\Sigma \vdash_{\mathsf{obs}} l : \textbf{alive}$, requiring that l is alive and *accessible by the observer*.

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

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

to mean that there is a (weak) bisimulation between the configurations $\Sigma \triangleright M$ and $\Sigma \triangleright N$

Example 3 (Server Implementations Revisited). Consider the network:



formally represented as $\Sigma = \langle \mathcal{N}, \mathcal{O}, \mathcal{H} \rangle$, where $\mathcal{N} = \{l, k_1, k_2\}$, $\mathcal{O} = \{l \leftrightarrow k_1,\ l \leftrightarrow k_2,\ k_1 \leftrightarrow k_2\}$ and $\mathcal{H} = \emptyset$. If we assume that the three server implementations presented earlier in the Introdcuction were running over Σ, we are able to formally argue that

$$\Sigma \models \mathsf{server} \not\approx_{int} \mathsf{servD} \not\approx_{int} \mathsf{servD2Rt}$$

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

(l-open)

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

(l-weak)

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

(l-rest-typ)

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

(l-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)

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

(l-par-comm)

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

To see this, it is sufficient to examine the behaviour of these systems subsequent to an actions such as $\xrightarrow{k \not\leftrightarrow k_1}$ and $\xrightarrow{\mathsf{kill}:k_1}$.

One can also use the lts to establish positive results. For example, for $\Sigma_{l,k} = \langle\{l, k\}, \{l \leftrightarrow k\}, \emptyset\rangle$, one can prove

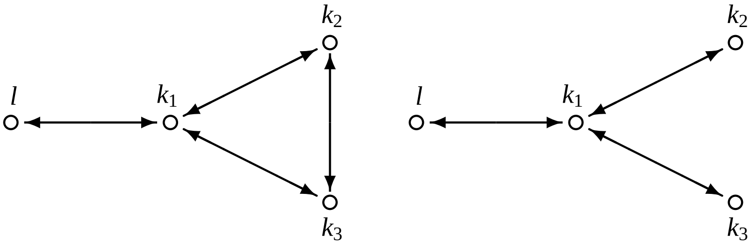
$$\Sigma_{l,k} \models l[\![\mathsf{ping}\ k.\ a!\langle\rangle\ulcorner\mathbf{0}\urcorner]\!] \approx_{int} k[\![\mathsf{go}\ l.a!\langle\rangle]\!]$$

Nevertheless, we can argue, at least informally, that this notion of equivalence is too *discriminating* and the lts labels too *intentional*, because we distinguish between configurations where the differences in behaviour are difficult to observe. Problems arise when there is an interplay between *hidden* nodes, links and dead nodes.

Example 4 (Inaccessible Network State). Let Σ be the network in which there is only one node, l, which is alive and consider the two systems

$$\begin{aligned} M_2 &\Leftarrow (\nu\, k_1 : \{l\})\, (\nu\, k_2 : \{k_1\})\, (\nu\, k_3 : \{k_1, k_2\})\ l[\![a!\langle k_2, k_3\rangle.P]\!] \\ N_2 &\Leftarrow (\nu\, k_1 : \{l\})\, (\nu\, k_2 : \{k_1\})\, (\nu\, k_3 : \{k_1\})\ l[\![a!\langle k_2, k_3\rangle.P]\!] \end{aligned}$$

When M_2 and N_2 are running on Σ, the code $l[\![a!\langle k_2, k_3\rangle.P]\!]$, present in both M_2 and N_2, is effectively running on the following respective networks, due to the newly declared locations:



Using our lts, we determine that $\Sigma \models M_2 \not\approx_{int} N_2$ because the configurations give rise to *different* output actions:

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

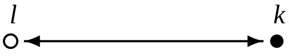
$$\Sigma \triangleright N_2 \xrightarrow{(k_2:\emptyset,\ k_3:\emptyset)l:a!\langle k_2,k_3\rangle} \Sigma + k_2:\emptyset\ + k_3:\emptyset \triangleright (\nu\, k_1:\{l,k_2,k_3\})\, l[\![P]\!]$$

The difference lies in the type at which the location k_3 is exported: M_2 exports k_3 connected to k_2 whereas in N_2 exports a completely disconnected k_3.

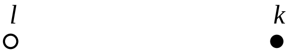
However, if k_1 does not occur in P, then k_1 can never be scope extruded to the observer and thus k_2 and k_3 will remain inaccessible in both systems. This means that the presence (or absence) of the link $k_2 \leftrightarrow k_3$ can never be verified by the observer and thus there should be no observable difference between M_2 and N_2 running on Σ.

Example 5 (Interplay between Node and Link Failure). We consider the following three configurations together with the depiction of the respective networks over which the common located process $l[\![a!\langle k\rangle.P]\!]$ is running:

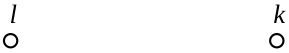
$$M_3^1 \Leftarrow \langle\{l,a\},\{l_1 \leftrightarrow l_1\},\emptyset\rangle \triangleright (\nu\, k:\mathtt{loc}_{\mathtt{d}}[\{l\}])l[\![a!\langle k\rangle.P]\!] \quad :$$



$$M_3^2 \Leftarrow \langle\{l,a\},\{l_1 \leftrightarrow l_1\},\emptyset\rangle \triangleright (\nu\, k:\mathtt{loc}_{\mathtt{d}}[\emptyset])l[\![a!\langle k\rangle.P]\!] \quad :$$



$$M_3^3 \Leftarrow \langle\{l,a\},\{l_1 \leftrightarrow l_1\},\emptyset\rangle \triangleright (\nu\, k:\mathtt{loc}_{\mathtt{a}}[\emptyset])l[\![a!\langle k\rangle.P]\!] \quad :$$



Intuitively, no observer can distinguish between these three configurations; even though some observer might obtain the scoped name k by inputting on channel a at l, it cannot determine the difference in the state of network. From rule (l-nping), we conclude that any attempt to ping k from l will yield the negative branch. However, such an observation does not give the observer enough information about whether it was caused by a node fault at k, a link fault between l and k or both. As a result, we would like to equate all three configuration. However, our lts specifies that all three configurations perform the output with different scope extrusion labels, namely:

$$\langle\{l\},\{l \leftrightarrow l\},\emptyset\rangle \triangleright M_3^1 \xrightarrow{(k:\mathtt{loc}_{\mathtt{d}}[\{l\}])l:a!\langle k\rangle} \langle\{l\},\{l \leftrightarrow l\},\emptyset\rangle \triangleright l[\![P]\!]$$

$$\langle\{l\},\{l \leftrightarrow l\},\emptyset\rangle \triangleright M_3^2 \xrightarrow{(k:\mathtt{loc}_{\mathtt{d}}[\emptyset])l:a!\langle k\rangle} \langle\{l\},\{l \leftrightarrow l\},\emptyset\rangle \triangleright l[\![P]\!]$$

$$\langle\{l\},\{l \leftrightarrow l\},\emptyset\rangle \triangleright M_3^3 \xrightarrow{(k:\mathtt{loc}_{\mathtt{a}}[\emptyset])l:a!\langle k\rangle} \langle\{l\},\{l \leftrightarrow l\},\{k \leftrightarrow k\}\rangle \triangleright l[\![P]\!]$$

and as a result, these configurations are differentiated by $\approx_{int}$.

4 Reduction barbed congruence

The fundamental problem with the lts of the previous section is that when new locations are scope extruded, the associated information, coded in the types at which they are exported, is too detailed. The current actions carry too much *internal* information and hence, we need a revised form of action, which carry just the right amount of information.

However, before we plunge into our revision, it is best to have yardstick with respect to which we can calibrate the appropriateness of the revised labelled actions, and the resulting bisimulation equivalence. We adapt a well-known formulation of contextual equivalence to DπF, [13, 11], called *reduction barbed congruence*. This relies on the notion of a *barb*, a collection of primitive observations which can be made on systems. Let us write $\Sigma \triangleright N \Downarrow_{a@l}$ to mean that an output on channel a at an accessible location l can be observed. Then, we would expect all reasonable behavioural equivalencies to preserve these barbs. But the key idea in the definition is to use a notion of *contextual* relation over configurations, in which the contexts only have access to the *observable* part of the network.

Definition 3 (Contextual Relations). *A relation $\mathcal{R}$ over configurations is* contextual *if:*

(Parallel Systems)

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

(Network Extensions)

- $\Sigma \triangleright M \,\mathcal{R}\, \Sigma' \triangleright N$ *and* $\Sigma \vdash_{obs} \mathtt{T},\ \Sigma' \vdash_{obs} \mathtt{T},\ n$ *fresh* *implies* $\Pi + n : \mathtt{T} \models M \,\mathcal{R}\, N$

where $\Sigma \vdash_{obs} O$ and $\Sigma \vdash_{obs} \mathtt{T}$ restrict the observer O and connections of location types to accessible locations only.

Definition 4 (Reduction barbed congruence). *Let $\cong$ be the largest relation between configurations which is* contextual, *preserves* barbs *and is* reduction-closed.

Note that, apriori, this definition allows us to compare configurations which have different networks. However, it turns out that whenever $\Sigma \triangleright M \cong \Sigma' \triangleright N$, the external parts of Σ and Σ' must coincide. In the sequel, we abbreviate $\Sigma \triangleright M \cong \Sigma \triangleright N$, the cases where both networks are identical, to $\Sigma \models M \cong N$.

We now outline a revision of our labelled actions with the property that the resulting bisimulation equivalence coincides with the yardstick relation, $\cong$. The idea is to reuse the same actions but to simply change the types at which bound names appear. Currently, these are of the form $\mathtt{T} = \mathtt{ch}$ or $\mathtt{loc_S[C]}$, where the latter indicates the status of a location and its connectivity. We change these types to new types of the form $\mathtt{L}, \mathtt{K} = \{l_1 \leftrightarrow k_1, \ldots, l_i \leftrightarrow k_i\}$ where $\mathtt{L}, \mathtt{K}$ are linksets. these represent the new live nodes and links, which are made accessible to observers by the extrusion of the new location. Alternatively, this is the information which is added to the observable part of the network representation, Σ_O, as a result of the action.

The formal definition is given in Figure 9, which is expressed in terms of a function $\mathsf{lnk}(n : \mathtt{T}, \Sigma)$, the definition of which is relegated to the Appendix. Intuitively, if n is

Table 9. *The derived lts for DπF*

(l-deriv-1)

$$\frac{\Sigma \triangleright N \xrightarrow{\mu} \Sigma' \triangleright N'}{\Sigma \triangleright N \xmapsto{\mu} \Sigma' \triangleright N'}\ \mu \in \{\tau, \mathsf{kill} : l, l \nleftrightarrow k\}$$

(l-deriv-2)

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

(l-deriv-3)

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

a channel ($\mathtt{T} = \mathtt{ch}$) or a dead location ($\mathtt{T} = \mathtt{loc_d}[\mathtt{L}]$), $\mathsf{lnk}(n : \mathtt{T}, \Sigma)$ returns the empty link set $\emptyset$. Otherwise, when it is a live location ($\mathtt{T} = \mathtt{loc_a}[\mathtt{C}]$), it constructs the linkset denoting the nodes and links that are made accessible by the addition of the new location $n : \mathtt{loc_a}[\mathtt{C}]$ to the network Σ.

These revised actions give rise to a new bisimulation equivalence over configurations, $\approx$, and we use

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

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

Example 6 (Derived bisimulations). Recall that, in Example 4, we had different actions for $\Sigma \triangleright M_2$ and $\Sigma \triangleright N_2$ because $\Sigma \triangleright M_2$ exported k_3 with a link to k_2 and $\Sigma \triangleright N_2$ did not. However, Σ contains only one accessible node, l, and extending it with the completely disconnected new node k_2 does not increase the set of accessible nodes, Σ_O. Furthermore, increasing $\Sigma + k_2 : \emptyset$ with a new node k_3, linked to the inaccessible k_2 (in the case of $\Sigma \triangleright M_2$) or completely disconnected (in the case of $\Sigma \triangleright N_2$), also leads to no increase in the accessible nodes. Correspondingly, the calculations of $\mathsf{lnk}(k_2 : \emptyset, \Sigma)$ and $\mathsf{lnk}(k_3 : \{k_2\}, \Sigma + k_2 : \emptyset)$ both lead to the empty linkset type. Formally, we get the same derived actions

$$\Sigma \triangleright M_2 \xmapsto{(k_2:\emptyset, k_3:\emptyset)l:a!\langle k_2, k_3\rangle} \Sigma + k_2 : \emptyset\ + k_3 : \{k_2\} \triangleright (\nu\, k_1 : \{l, k_2, k_3\})\, l[\![P]\!]$$
$$\Sigma \triangleright N_2 \xmapsto{(k_2:\emptyset, k_3:\emptyset)l:a!\langle k_2, k_3\rangle} \Sigma + k_2 : \emptyset\ + k_3 : \emptyset \triangleright (\nu\, k_1 : \{l, k_2, k_3\})\, l[\![P]\!]$$

Furthermore, if P contains no occurrence of k_1, we can go on to show $\Sigma \models M \approx N$.

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

$$\Sigma + k_2 : \emptyset\ + k_3 : \{k_2\} \triangleright (\nu\, k_1 : \{l, k_2, k_3\})\, l[\![P]\!] \xmapsto{(k_1:\mathtt{L})l:a!\langle k_1\rangle} \ldots$$
$$\Sigma + k_2 : \emptyset\ + k_3 : \emptyset \triangleright (\nu\, k_1 : \{l, k_2, k_3\})\, l[\![P]\!] \xmapsto{(k_1:\mathtt{K})l:a!\langle k_1\rangle} \ldots$$

where $\mathtt{L}/\mathtt{K} = \{k_2 \leftrightarrow k_3\}$. More specifically, $\mathtt{L}$ and $\mathtt{K}$ hold information directly related to k_1 such as $k_1 \leftrightarrow l$ together with information related to previously inaccessible nodes such as $k_2 \leftrightarrow k_3$, which has now become accessible as a result of exporting k_1. The first derived action $(k_1 : \mathtt{L})l : a!\langle k_1\rangle$ thus exports the extra (previously hidden) information $k_2 \leftrightarrow k_3$ in $\mathtt{L}$ and based on this discrepancy, we have $\Sigma \models M_2 \not\approx N_2$

Revisiting Example 5, the three different actions of M_3^1, M_3^2 and M_3^3 now converge to the same action $M_3^i \xmapsto{(k:\emptyset)l:a!\langle k\rangle} \ldots \triangleright l[\![P]\!]$, hence $\Sigma \models M_3^1 \approx M_3^2 \approx M_3^3$.

The main result of this paper can now be stated:

Theorem 1. *In DπF, $\Sigma \models M \approx N$ if and only if $\Sigma \models M \cong N$*

Proof. (Outline) In one direction, this involves showing that $\approx$ as a relation over configurations satisfies the defining properties of *reduction barbed congruence*. The main problem here is to show that $\approx$ is contextual, and in particular that $\Sigma \models M \approx N$ implies $\Sigma \models M|O \approx N|O$ for every O which only has access to the external (accessible) part of Σ. This in turn involves developing *Decomposition* and *Composition* lemmas for derived actions from configurations of the form $\Sigma \triangleright M|O$. The overall structure of the proof is similar to the corresponding result in [10], Proposition 12, but the details are more complicated because of the presence of the network. We therefore relegate to the Appendix the formal statement of these lemmas and refer to the full paper, [8], for an elaborate presentation of the proofs.

The essential part of the converse is to show *Definability*, that is for every derived action, relative to a network Σ, there is an observer which only uses the external knowledge of Σ to completely characterises the effect of that action. These observers have already been constructed for simpler languages such as π-calculus, in [11], and Dπ, in [10]. Here the novelty is to be able to characterise the observable effect that actions have on a network. But it turns out that for every Σ we can define an observer O_{Σ} which when run on an arbitrary network Σ' can determine whether the external or accessible part of Σ' coincides with that of Σ using a process called *verNetStatus*, which we also include in the Appendix. The complete proof is included in the full paper, [8].

5 Conclusions and Related Work

We have presented a simple extension of Dπ, in which there is an explicit representation of the state of the underlying network on which processes execute. Our main result is a *fully-abstract* bisimulation equivalence with which we can reason about the behaviour of distributed processes in the presence of specific network configurations with dead nodes and partial connections and also dynamic network failures. To the best of our knowledge, this is the first time system behaviour in the presence of *link* failure and *permanent* partial accessibility of nodes has ever been investigated. It is also the first time the interplay between node and link failure and their respective program observation has been investigated in a process calculus setting.

Application and Future Work: Our work is best viewed as a well-founded framework from which numerous variations could be considered such as unidirectional links, ping constructs that are *eventually* correct, transient failure and persistent code. In our more immediate research, we intend to use our present results to develop a theory of *fault-tolerance* and to apply it to example systems from the literature such as [5].

As it currently stands, we believe our work lends itself well to the study of distributed software that 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. In these settings, the software typically *discovers* new parts of the neighbouring network at runtime and *updates* its knowledge of the current underlying network with changes caused by failure.

Related Work: There have been a number of studies on process behaviour in the presence of *permanent node failure* only, amongst which [17], which was 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. The closest work to the study of link failure is [6], where distributed Linda-like programs are studied in the presence of connect and disconnect software primitives that dynamically change the accessibility of locations. The connect construct employed is however very powerful and can connect any two disconnected sites; this obviates the need for observer restricted views, thereby simplifying immensely the theory. Elsewhere, permanent location failure with hierarchical dependencies have been studied by Fournet, Gonthier, Levy and Remy in [7]. Berger [3] was the first to study a π-calculus extension that models *transient* location failure with persistent code and communication failures, while Nestmann, Merro and Fuzzatti [16] employ a tailor made process calculus to express standard results in distributed systems, such as [5].

References

1. Roberto M. Amadio. An asynchronous model of locality, failure, and process mobility. In D. Garlan and D. Le Métayer, editors, *Proceedings of the 2nd International Conference on Coordination Languages and Models (COORDINATION'97)*, volume 1282, pages 374–391, Berlin, Germany, 1997. Springer-Verlag.
2. Roberto M. Amadio and Sanjiva Prasad. Localities and failures. *FSTTCS: Foundations of Software Technology and Theoretical Computer Science*, 14, 1994.
3. Martin Berger. Basic theory of reduction congruence for two timed asynchronous π-calculi. In *Proc. CONCUR'04*, 2004.
4. Luca Cardelli. Wide area computation. In *Proceedings of* 26^{th} *ICALP*, Lecture Notes in Computer Science, pages 10–24. Springer-Verlag, 1999.
5. Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, March 1996.
6. Rocco De Nicola, Daniele Gorla, and Rosario Pugliese. Basic observables for a calulus for global computing. Technical report, Universita di Firenze, 2004.
7. Cedric Fournet, Georges Gonthier, Jean Jaques Levy, and Remy Didier. A calculus of mobile agents. *CONCUR 96*, LNCS 1119:406–421, August 1996.
8. Adrian Francalanza and Matthew Hennessy. Location and link failure in a distributed π-calculus. Technical report, University of Sussex, 2005.
9. R.J. van Glabbeek and U. Goltz. Equivalence notions for concurrent systems and refinement of actions (extended abstract). In A. Kreczmar and G. Mirkowska, editors, Proceedings 14^{th} Symposium on *Mathematical Foundations of Computer Science,* MFCS '89, Porąbka-Kozubnik, Poland, August/September 1989, volume 379 of *lncs*, pages 237–248. Springer-Verlag, 1989.
10. 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.

11. Matthew Hennessy and Julian Rathke. Typed behavioural equivalences for processes in the presence of subtyping. *Mathematical Structures in Computer Science*, 14:651–684, 2004.
12. Matthew Hennessy and James Riely. Resource access control in systems of mobile agents. *Information and Computation*, 173:82–120, 2002.
13. K. Honda and N. Yoshida. On reduction-based process semantics. *Theoretical Computer Science*, 152(2):437–486, 1995.
14. R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.
15. Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, parts I and II. *Information and Computation*, 1992.
16. Nestmann, Fuzzati, and Merro. Modeling consensus in a process calculus. In *CONCUR: 14th International Conference on Concurrency Theory*. LNCS, Springer-Verlag, 2003.
17. James Riely and Matthew Hennessy. Distributed processes and location failures. *Theoretical Computer Science*, 226:693–735, 2001.
18. Davide Sangiorgi and David Walker. *The π-calculus*. Cambridge University Press, 2001.

A DπF Notation

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

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

For DπF we have two kinds of network representations, ranged over by $\varDelta$ and $\varSigma$. We define the following operations on them:

$$
\begin{array}{rcll}
\varDelta - l & \overset{\text{def}}{=} & \langle \varDelta_{\mathcal{N}},\ \varDelta_{\mathcal{D}} \cup \{l\},\ \varDelta_{\mathcal{L}}\rangle & (\textit{location killing}) \\
\varSigma - l & \overset{\text{def}}{=} & \langle \varSigma_{\mathcal{N}},\ \varSigma_{\mathcal{O}}/\{l\},\ \varSigma_{\mathcal{L}}/\{l\}\rangle & (\textit{location killing}) \\
 & & & \\
\varDelta - l \leftrightarrow k & \overset{\text{def}}{=} & \langle \varDelta_{\mathcal{N}},\ \varDelta_{\mathcal{D}},\ \varDelta_{\mathcal{L}}/\{\langle l, k\rangle, \langle k, l\rangle\}\rangle & (\textit{link breaking}) \\
\varSigma - l \leftrightarrow k & \overset{\text{def}}{=} & \langle \varSigma_{\mathcal{N}},\ \varSigma_{\mathcal{O}}/\{\langle l, k\rangle, \langle k, l\rangle\},\ \varSigma_{\mathcal{L}}/\{\langle l, k\rangle, \langle k, l\rangle\}\rangle & (\textit{link breaking}) \\
 & & & \\
\varDelta + a : \mathtt{ch} & \overset{\text{def}}{=} & \langle \varDelta_{\mathcal{N}} \cup \{a\},\ \varDelta_{\mathcal{D}},\ \varSigma_{\mathcal{L}}\rangle & (\textit{adding a channel}) \\
\varSigma + a : \mathtt{ch} & \overset{\text{def}}{=} & \langle \varSigma_{\mathcal{N}} \cup \{a\},\ \varSigma_{\mathcal{O}},\ \varSigma_{\mathcal{H}}\rangle & (\textit{adding a channel}) \\
 & & & \\
\varDelta + l : \mathtt{loc_d}[\mathtt{C}] & \overset{\text{def}}{=} & \langle \varDelta_{\mathcal{N}} \cup \{l\},\ \varDelta_{\mathcal{D}} \cup \{l\},\ \varSigma_{\mathcal{L}} \cup l \leftrightarrow \mathtt{C}\rangle & (\textit{adding a location}) \\
\varDelta + l : \mathtt{loc_a}[\mathtt{C}] & \overset{\text{def}}{=} & \langle \varDelta_{\mathcal{N}} \cup \{l\},\ \varDelta_{\mathcal{D}},\ \varSigma_{\mathcal{L}} \cup l \leftrightarrow \mathtt{C}\rangle & \\
\varSigma + l : \mathtt{loc_d}[\mathtt{C}] & \overset{\text{def}}{=} & \langle \varSigma_{\mathcal{N}} \cup \{l\},\ \varSigma_{\mathcal{O}},\ \varSigma_{\mathcal{H}}\rangle & (\textit{adding a location}) \\
\varSigma + l : \mathtt{loc_a}[\mathtt{C}] & \overset{\text{def}}{=} & &
\end{array}
$$

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

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

$$
\begin{array}{rcll}
\varSigma(\varDelta) & \overset{\text{def}}{=} & \langle \varDelta_{\mathcal{N}},\ \varDelta_{\mathcal{L}}/\varDelta_{\mathcal{D}},\ \emptyset\rangle & (\textit{from } \varDelta \textit{ to } \varSigma) \\
\varDelta(\varSigma) & \overset{\text{def}}{=} & \langle \varSigma_{\mathcal{N}},\ (\mathbf{loc}(\varSigma_{\mathcal{N}})/\mathbf{dom}(\varSigma_{\mathcal{O}} \cup \varSigma_{\mathcal{H}})),\ \varSigma_{\mathcal{O}} \cup \varSigma_{\mathcal{H}}\rangle & (\textit{from } \varSigma \textit{ to } \varDelta) \\
\mathcal{I}(\varSigma) & \overset{\text{def}}{=} & \langle \varSigma_{\mathcal{N}},\ \varSigma_{\mathcal{O}}\rangle & (\textit{observer knowledge}) \\
\mathcal{I}(\varDelta) & \overset{\text{def}}{=} & \mathcal{I}(\varSigma(\varDelta)) &
\end{array}
$$

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

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

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

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

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

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

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

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

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

B Main Lemmas and Propositions

B.1 Lemmas and Propositions to prove $\Sigma \models M \approx N$ implies $\Sigma \models M \cong N$

Lemma 1 (Composition).

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

Proof. (Outline) The proof progresses by extracting the necessary structure of the systems M, N and the network Σ to be able to re-compose them using rules such as (l-par-ctxt), (l-par-comm) and (l-rest)

Lemma 2 (Decomposition). *Suppose* $\Sigma \triangleright M|N \xmapsto{\mu} \Sigma' \triangleright M'$ *where* $\Sigma \vdash_{obs} M$ *or* $\Sigma \vdash_{obs} N$ *. 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*
   - $\Sigma \triangleright M \xmapsto{(\tilde{n}:\tilde{\mathtt{L}})l:a!\langle V\rangle} \Sigma'' \triangleright M''$ *and* $\Sigma \triangleright N \xmapsto{(\tilde{n}:\tilde{\mathtt{K}})l:a?(V)} \Sigma''' \triangleright N'$
   - $\Sigma \triangleright M \xmapsto{(\tilde{n}:\tilde{\mathtt{K}})l:a?(V)} \Sigma'' \triangleright M''$ *and* $\Sigma \triangleright N \xmapsto{(\tilde{n}:\tilde{\mathtt{L}})l:a!\langle V\rangle} \Sigma''' \triangleright N'$

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

Proof. (Outline) The proof progressed by induction on the derivation of $\Sigma \triangleright M|N \xmapsto{\mu} \Sigma' \triangleright M'$.

Proposition 1 (Contextuality of Behavioural 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* $\Sigma_{1..2} \vdash_{obs} O, \mathtt{T}$ *and* n *fresh in* $\Sigma_{1..2}$ *we have:*

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

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

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

and showing that $\mathcal{R} \subseteq \approx$; since $\approx$ is the biggest possible relation, this would mean that it is contextual.

B.2 Lemmas and Propositions to prove $\Sigma \models M \cong N$ implies $\Sigma \models M \approx N$

Lemma 3 (Observable Network). *Consider the process definition $verNetStatus_k^{\langle \mathcal{N},\mathcal{O}\rangle}(x)$ that is intended to be running at a location k, connected to all observable locations in a network Σ. It returns an output on the parameterised channel x if and only if $\mathcal{N} = \Sigma_{\mathcal{N}}$ and $\mathcal{O} = \Sigma_{\mathcal{O}}$.*

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

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

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

Assume that for arbitrary network representation Σ, Σ_+ stands for

$$\Sigma + k_0 : \mathtt{loc}_{\mathtt{a}}[\mathbf{dom}(\Sigma_{\mathcal{O}})] + \textsc{succ} : \mathtt{ch}$$

Then,

$$\Sigma_+ \triangleright k_0[\![verNetStatus_{k_0}^{\langle \mathcal{N},\mathcal{O}\rangle}(\textsc{succ})]\!] \longrightarrow^* \Sigma_+ \triangleright k_0[\![\textsc{succ}!\langle\rangle]\!] \ \ \textit{iff}\ \ \mathcal{N} = \Sigma_{\mathcal{N}} \textit{ and } \mathcal{O} = \Sigma_{\mathcal{O}}$$

Proof. (Outline) We prove this lemma by contradiction. We analyse all the possible cases why $\mathcal{N} = \Sigma_{\mathcal{N}}$ and $\mathcal{O} = \Sigma_{\mathcal{O}}$ and then show that for each of these cases,

$$\Sigma_+ \triangleright k_0[\![verStat_{k_0}^{\langle \mathcal{N},\mathcal{O}\rangle}(\textsc{succ})]\!] \not\longrightarrow^* \Sigma_+ \triangleright k_0[\![\textsc{succ}!\langle\rangle]\!]$$

Proposition 2 (Definability). *Assume that for an arbitrary network representation Σ, the network Σ_+ denotes:*

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

where k_0, $\textsc{succ}$ and $\textsc{fail}$ are fresh to $\Sigma_{\mathcal{N}}$. Thus, for every external action μ and network representation Σ, every non-empty finite set of names Nm where $\Sigma_{\mathcal{N}} \subseteq Nm$, 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 $\Sigma_{\mathcal{O}}$, there exists a system $T^{\mu}(Nm, \textsc{succ}, \textsc{fail}, k_0)$ with the property that $\Sigma_+ \vdash_{obs} T^{\mu}(Nm, \textsc{succ}, \textsc{fail}, k_0)$, such that:

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

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

Proof. (Outline) We have to prove that the above two clauses are true for all of the four external actions. If μ is one of the two non-standard actions, $\mathsf{kill} : l$ and $l \nleftrightarrow k$, the test required are:

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

and

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

respectively. If μ is the bound input action $(\tilde{n} : \tilde{\mathtt{L}})l : a?(V)$, where $\tilde{\mathtt{L}} = \mathsf{lnk}(\tilde{n} : \tilde{\mathtt{T}}, \Sigma)$ for some $\tilde{\mathtt{T}}$, the required system is

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

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

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

such that

$$verNwStatus_{k_0}^{\langle N,O\rangle}(x_1 \ldots x_m, y) \Leftarrow (\nu\, k' : \mathtt{T}_{k'})\mathsf{go}\ k'.(\nu d)\begin{pmatrix} verNetStatus_{k'}^{\langle N,O\rangle + (x_1..x_m : \tilde{\mathtt{K}})}(d) \\ \mid d?().\mathsf{go}\ k_0.y!\langle k'\rangle \end{pmatrix}$$

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

For the sake of presentation ,we assume that the first $v_1 \ldots v_m$ 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 construct for arbitrary ordering of bound names in V using the same principles used for this test. We also use the conditional $\mathsf{if}\ x \notin Nm.P$ as an abbreviation for the obvious nested negative comparisons between x and each name in Nm.