A Theory for Observational Fault Tolerance

Adrian Francalanza [a], Matthew Hennessy [b]

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

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

Key words:
fault tolerance, distributed calculi, observational equivalences, labelled transition systems, bisimulations, up-to techniques

1 Introduction

One reason for the study of programs in the presence of *faults*, i.e. defects at the lowest level of abstractions [2], is to be able to construct more *dependable* systems, meaning systems exhibiting a high probability of *behaving* according to their *specification* [18]. System dependability is often expressed through attributes like maintainability, availability, safety and *reliability*, the latter of which is defined as a measure of the *continuous delivery* of *correct behaviour*, [18]. There are a number of approaches for achieving system dependability in the presence of faults, ranging from fault removal, fault prevention and *fault tolerance*.

The fault tolerant approach to system dependability consist of various techniques that employ *redundancy* to prevent faults from generating *failure*, i.e. abnormal behaviour caused by faults [2]. Two forms of redundancy are *space redundancy* (*replication*), i.e. using several copies of the same system components, and *time redundancy*, i.e. performing the same chunk of computation more than once. Redundancy can also be managed in various ways: certain fault tolerant techniques are based on *fault detection* which subsequently trigger *fault recovery*; other techniques do not use fault detection and still attain fault masking - these, however, tend to be more expensive in terms of redundancy usage (time redundancy). If enough redundancy is used and it is managed appropriately, this can lead to *fault masking*, where the specified behaviour is preserved without noticeable glitches.

Fault tolerance is of particular relevance in distributed computing. Distribution yields a natural notion of *partial failure*, whereby faults affect a *subset* of the computation. Partial failure, in turn, gives scope for introducing redundancy as *replication*, distributed across independently failing entities such as locations. In general, the higher the replication, the greater the potential for fault tolerance. Nevertheless, fault tolerance also depends on how replicas are managed. One classification, due to [18], identifies three classes, namely *active replication* (all replicas are invoked for every operation), *passive replication* (operations are invoked on primary replicas and secondary replicas are updated in batches at checkpoints), and *lazy replication* (a hybrid of the previous two approaches, exploiting the separation between write and read operations).

In this paper we address fault tolerance in a distributed setting, focusing on simple examples using *stateless* (read-only) replicas which are invoked only once. This simplification obviates the need for additional machinery to sequence multiple requests (in the case of active replication) or synchronise the state of replicas (in the case of passive replication); as a result management techniques based on lazy replication simply collapse into passive replication category. Nevertheless, these simple examples still capture the essence of the concepts we choose to study.We code these examples in a simplified version of Dπ [10] with failing locations [4], a distributed version of the standard π-calculus [16], where the locations that host processes model closely physical network nodes.

Example 1 (Fault Tolerant Servers) *Consider the systems* server_i, *three server implementations accepting client requests on channel* req *with two arguments,* x *being the value to process and* y *being the reply channel on which the answer is returned. Requests are forwarded to internal databases, denoted by the scoped channel* $data$, *distributed and replicated across the auxiliary locations* k_1, k_2 *and* k_3. *A database looks up the mapping of the value* x *using some unspecified function* $f(-)$ *and returns the answer,* $f(x)$, *back on port* y. *When multiple (database) replicas are used, as in* server_2 *and* server_3, *requests are sent to all replicas in an arbitrary fashion, without the use of failure detection, and multiple answers are synchronised at* l *on the scoped channel* $sync$, *returning the first answer received on* y.

$$\mathsf{server}_1 \Leftarrow (\nu\, data)\left(\begin{array}{l} l[\![req?(x,y).\mathsf{go}\, k_1.data!\langle x,y,l\rangle]\!] \\ \mid k_1[\![data?(x,y,z).\mathsf{go}\, z.y!\langle f(x)\rangle]\!] \end{array}\right)$$

$$\mathsf{server}_2 \Leftarrow (\nu\, data)\left(\begin{array}{l} l\left[\!\!\left[req?(x,y).(\nu sync)\left(\begin{array}{l} \mathsf{go}\, k_1.data!\langle x, sync, l\rangle \\ \mid \mathsf{go}\, k_2.data!\langle x, sync, l\rangle \\ \mid sync?(x).y!\langle x\rangle \end{array}\right)\right]\!\!\right] \\ \mid k_1[\![data?(x,y,z).\mathsf{go}\, z.y!\langle f(x)\rangle]\!] \\ \mid k_2[\![data?(x,y,z).\mathsf{go}\, z.y!\langle f(x)\rangle]\!] \end{array}\right)$$

$$\mathsf{server}_3 \Leftarrow (\nu\, data)\left(\begin{array}{l} l\left[\!\!\left[req?(x,y).(\nu sync)\left(\begin{array}{l} \mathsf{go}\, k_1.data!\langle x, sync, l\rangle \\ \mid \mathsf{go}\, k_2.data!\langle x, sync, l\rangle \\ \mid \mathsf{go}\, k_3.data!\langle x, sync, l\rangle \\ \mid sync?(x).y!\langle x\rangle \end{array}\right)\right]\!\!\right] \\ \mid k_1[\![data?(x,y,z).\mathsf{go}\, z.y!\langle f(x)\rangle]\!] \\ \mid k_2[\![data?(x,y,z).\mathsf{go}\, z.y!\langle f(x)\rangle]\!] \\ \mid k_3[\![data?(x,y,z).\mathsf{go}\, z.y!\langle f(x)\rangle]\!] \end{array}\right)$$

The theory developed in [4] enables us to *differentiate* between these systems, based on the different behaviour observed when composed with systems such as

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

in a setting where locations may fail; in the definition of client, *ret* is the name of a reply channel, and v is some value appropriate to the unspecified function $f(-)$. Here we go one step further, allowing us to *quantify*, in some sense, the difference between these systems. Intuitively, if locations k_1, k_2 and k_3 can fail in fail-stop fashion[17] and observations are limited to location l only, then server_2 seems to be more *fault tolerant* than server_1. In fact observers limited to l, such as client, cannot observe changes in behaviour in server_2 when *at most 1* location from k_1, k_2 and k_3 fails. Similarly, server_3 is more *fault tolerant* than server_1 and server_2 because the composite system

$$\mathsf{server}_3 \mid \mathsf{client}$$

preserves its behaviour at l *up to 2* faults occurring at any of k_1, k_2 and k_3.

In this paper we give a formal definition of when a system is deemed to be fault tolerant up to n-faults, which coincides with this intuition. As in [4] we need to consider systems M, running on a network, which we will represent as $\Gamma \triangleright M$, where Γ is some representation of the current state of the network. Then we will say that M is fault-tolerant up to n faults, when running on network Γ, if

$$\Gamma \triangleright M \cong F^n[\Gamma \triangleright M] \tag{1}$$

where $F^n[\;]$ is some context which induces at most n faults on Γ, and $\cong$ is some behavioural equivalence between system descriptions.

A key aspect of this behavioural equivalence is the implicit separation between *reliable* locations, which are assumed not to fail, and *unreliable* locations, which may fail. In the above example, l is reliable while k_1, k_2 and k_3 are assumed unreliable, thus subject to failure. Furthermore, it is essential that observers not have access to these unreliable locations, at any time during a computation. The general intuition is that we shield users from unreliable resources, thereby ensuring that no user code fails. But another important reason, which is more specific to this work, is that if observers are allowed to access unreliable locations then the proposed (1) above would no longer capture the intuitive notion of fault-tolerance up to n faults. For instance, we would no longer have

$$\Gamma \triangleright \mathsf{server}_2 \cong F^1[\Gamma \triangleright \mathsf{server}_2]$$

An observer with access to any of the locations k_1, k_2, k_3 would be able to detect possible failures in $F^1[\Gamma \triangleright \mathsf{server}_2]$, not present in $\Gamma \triangleright \mathsf{server}_2$, and thus discriminate between the two configurations.

We enforce this separation between reliable, observable locations and unreliable, unobservable locations using a simple type system in which reliable locations are represented as *public* values, and unreliable locations are represented as *confined*. In particular the typing system ensures that confined values, that is the unreliable locations, never become available at public locations.

In the second part of the paper we develop co-inductive proof techniques for proving system fault tolerance, that is establishing identities of the form (1) above; this can be seen as a continuation of the work in [4]. One novel aspect of the current paper is the use of *extended configurations* which have the form

$$\langle \Gamma, n \rangle \triangleright M \tag{2}$$

Here, the network Γ is bounded by the number n, denoting the maximum number of faults that Γ *can still incur* at unreliable locations. This extra network information allows us to define transitions which model the effect of the fault contexts on the network state in (1). More importantly however, it gives us more control over our proofs. For instance, it allows us to express how many unreliable locations may still fail, without committing ourselves to stating precisely which of these locations will fail, as is the case with fault contexts in (1). In addition, when we reach an extended configuration where $n = 0$ in (2) above, we can treat unreliable locations as immortal (reliable) since the extended configuration failure bound prohibits further unreliable locations from failing. All this turns out to alleviate some of the burden of constructing our proofs for fault tolerance.

The rest of the paper is organised as follows. Section 2 formally defines the language we use, DπLoc, together with a reduction semantics. It also contains the type system for enforcing the separation between public and confined locations. With this reduction semantics we can adapt the standard notion of *reduction barbed congruence*, [11,8], to DπLoc. But because of our type system, we are assured that the resulting behavioural equivalence $\cong$ reflects the fact that observations can only be made at public locations. In Section 3 we give our formal definitions of fault-tolerance, which relies on considering public locations as reliable and confined locations as unreliable. More specifically, we give two versions of (1) above, called

Table 1. *Syntax of typed DπF*

Types				
$\mathtt{U},\mathtt{W} ::= \mathtt{ch}_{\mathtt{c}}\langle\tilde{\mathtt{U}}\rangle \mid \mathtt{P} \mid \mathtt{loc}_{\mathtt{V}}$	*(stateless types)*	$\mathtt{V} ::= \mathtt{p} \mid \mathtt{c}$	*(visibility)*	
$\mathtt{P} ::= \mathtt{ch}_{\mathtt{p}}\langle\tilde{\mathtt{P}}\rangle \mid \mathtt{loc}_{\mathtt{p}}$	*(public types)*			
$\mathtt{T},\mathtt{R} ::= \mathtt{ch}_{\mathtt{c}}\langle\tilde{\mathtt{U}}\rangle \mid \mathtt{P} \mid \mathtt{loc}^{\mathtt{S}}_{\mathtt{V}}$	*(stateful types)*	$\mathtt{S} ::= \mathtt{a} \mid \mathtt{d}$	*(liveness status)*	
Processes				
$P, Q ::= u!\langle V\rangle.P$	*(output)*	$\mid u?(X).P$	*(input)*	
$\mid$ if $v = u$ then P else Q	*(matching)*	$\mid * u?(X).P$	*(replicated input)*	
$\mid (\nu\, n : \mathtt{T})P$	*(name declaration)*	$\mid$ go $u.P$	*(migration)*	
$\mid \mathbf{0}$	*(inertion)*	$\mid P \vert Q$	*(fork)*	
$\mid$ ping $u.P$ else Q	*(status testing)*			
Systems				
$M, N, O ::= l[\![P]\!]$	*(located process)*	$\mid N \vert M$	*(parallel)*	
$\mid (\nu\, n : \mathtt{T})N$	*(hiding)*			

static and *dynamic* fault tolerance, motivating the difference between the two via examples. Proof techniques for establishing fault tolerance are given in Section 4 where we give a complete co-inductive characterisation of $\cong$ using labelled actions. In Section 5 we refine these proof techniques for the more demanding fault tolerant definition, *dynamic fault tolerance*, using co-inductive definitions over extended configurations. We also develop useful up-to techniques for presenting witness bisimulations for extended configurations. Section 6 concludes by outlining the main contributions of the paper and discussing future and related work.

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 our language, DπLoc, is a variation of Dπ [10] and is given in Table 1. The main syntactic category is that of *systems*, ranged over by M, N: these are essentially a collection of *located processes*, or *agents*, composed in parallel where location and channel names may be scoped to a subset of agents. The syntax for processes, P, Q, is an extension of that in Dπ. There is input and output on channels; in the latter V represents a tuple of identifiers, while in the former X is tuple of variables, to be interpreted as a pattern. There are also the standard constructs for parallel composition, replicated input, local declarations, a test for equality between identifiers, migration and a zero process. The only addition to the original Dπ is $\mathsf{ping}\ k.P\ \mathsf{else}\ Q$, which tests for the *liveness status* of k in the style of [1,15] and branches to P if k is alive and Q otherwise. For these terms we assume the standard notions of *free* and *bound* occurrences of both names and variables, together with the associated concepts of α-conversion and *substitution*. We also assume that systems are *closed*, that is they have no free variable occurrences. Note that all of the examples discussed in Section 1 are valid system level terms in DπLoc. But it is worth emphasising that when we write definitions of the form

$$sys \Leftarrow S$$

the identifier *sys* is not part of our language; such definitions merely introduce convenient abbreviations for system level terms; in the above *sys* is simply an abbreviation for the term S.

Types: The original Dπ [10] comes endowed with a rich type system used to enforce access control policies. This is ignored in the current paper as it addresses issues which are orthogonal to our concerns. Instead, as explained in the Introduction, we use a simple type system for enforcing visibility constraints on values. The two main type categories, channels and locations, are decorated by visibility annotations, giving $\mathtt{ch}_{\mathtt{V}}\langle\tilde{\mathtt{U}}\rangle$ and $\mathtt{loc}_{\mathtt{V}}$, where $\mathtt{V}$ may either be $\mathtt{p}$, public, or $\mathtt{c}$, confined. In Table 1 these are called *stateless* types, and are ranged over by $\mathtt{U}$.

The essential constraint enforced by the typing system is that public channels, that is channels whose visibility is $\mathtt{p}$, can only be used to transmit *purely public* values; we use $\mathtt{P}$ to range over the types of these values. The type system also enforces a secondary constraint, namely that all confined locations are free and their number is fixed throughout execution. Stated otherwise, it prohibits scoped confined locations and the creation of fresh confined locations. The reason for the primary constraint is to prohibit observers, which are restricted to public channels/locations, from gaining access to confined channel/location names through interaction. The reason for the secondary restriction is that our definitions of fault tolerance depend on complete knowledge of the unreliable (confined) locations at the beginning of the analysis; we revisit this point later in Section 3.

However there is a complication. As explained in [4], the reduction semantics is considerably simplified if we also allow types to record the *liveness status* of a location, whether it is alive or dead. Thus we introduce two further annotations, $\mathtt{a}$

and $\mathtt{d}$, which can be added to the locations types, giving the forms $\mathtt{loc}_{\mathtt{V}}^{\mathtt{a}}$ and $\mathtt{loc}_{\mathtt{V}}^{\mathtt{d}}$. This gives a new class of types, the *stateful* types, ranged over by $\mathtt{T}$, $\mathtt{R}$. It should be emphasised however that although these stateful types are used in the typing system, typeability does not depend in any way on these *liveness status* annotations used on the location types.

A type environment Γ is a partial function from names to *stateful* types whose domain is finite. However when type-checking we wish to ignore the *liveness status* annotation on types. Consequently we define judgements of the form

$$\Gamma \vdash n : \mathtt{U} \tag{3}$$

where $\mathtt{U}$ is a stateless type. The returned type $\mathtt{U}$ is obtained simply by dropping the any *liveness status* annotation in $\Gamma(n)$.

Example 2 (Public Types) *We can have the type assignments*

$$\Gamma \vdash a : \mathtt{ch}_{\mathrm{p}}\langle \mathtt{ch}_{\mathrm{p}}\langle \mathtt{ch}_{\mathrm{p}}\langle\rangle\rangle\rangle \text{ and } \Gamma \vdash a : \mathtt{ch}_{\mathrm{c}}\langle \mathtt{ch}_{\mathrm{p}}\langle \mathtt{ch}_{\mathrm{p}}\langle\rangle\rangle\rangle$$

We can assign channel names to public channel types that communicate public values (first assignment) or to confined channel types that communicate arbitrary values (second assignment). However, we cannot have

$$\Gamma \vdash a : \mathtt{ch}_{\mathrm{p}}\langle \mathtt{ch}_{\mathrm{c}}\langle \mathtt{ch}_{\mathrm{p}}\langle\rangle\rangle\rangle$$

because the object type of a public channel type must also be a public type. Similarly, we cannot have the type assignment

$$\Gamma \vdash a : \mathtt{ch}_{\mathrm{c}}\langle \mathtt{ch}_{\mathrm{p}}\langle \mathtt{ch}_{\mathrm{c}}\langle\rangle\rangle\rangle$$

because, even though public values can be communicated on confined channels, the public channel communicated does not constitute a valid *public type since its object type is a confined channel.*

Our typing judgements take the form

$$\Gamma \vdash N$$

and are defined by the rules in Table 2, which use an extended form of type environment, Σ; these, in addition to names, also map *variables* to *stateless* types. The rules are standard value passing ones designed to ensure that the values transmitted respect the declared object type of the channels on which they are communicated. The rule (t-out) uses the obvious generalisation of the judgement (3) to values V, while in (t-rest) the standard notation $\Gamma, n : \mathtt{T}$ is used to denote the new environment obtained by extending the function Γ so that it now maps n to the type $\mathtt{T}$. But inherent in the use of this notation is that n is new to **dom**(Γ). Similar notation is used in (t-nw). The only non-standard feature in the typing rules of Table 2 is the

Table 2. *Typing rules for typed DπLoc*

Processes

(t-out)
$$\frac{\Sigma \vdash u : \mathtt{ch}_{\mathtt{V}}\langle\tilde{\mathtt{U}}\rangle \quad \Sigma \vdash V : \tilde{\mathtt{U}} \quad \Sigma \vdash P}{\Sigma \vdash u!\langle V\rangle.P}$$

(t-in-rep)
$$\frac{\Sigma \vdash u : \mathtt{ch}_{\mathtt{V}}\langle\tilde{\mathtt{U}}\rangle \quad \Sigma, X : \tilde{\mathtt{U}} \vdash P}{\Sigma \vdash u?(X).P \qquad \Sigma \vdash *u?(X).P}$$

(t-nw)
$$\frac{\mathtt{T} \neq \mathtt{loc_c} \quad \Sigma, n : \mathtt{T} \vdash P}{\Sigma \vdash (\nu\, n : \mathtt{T})P}$$

(t-cond)
$$\frac{\Sigma \vdash u : \mathtt{U},\ v : \mathtt{U} \quad \Sigma \vdash P,\ Q}{\Sigma \vdash \text{if } u = v \text{ then } P \text{ else } Q}$$

(t-fork)
$$\frac{\Sigma \vdash P,\ Q}{\Sigma \vdash P|Q}$$

(t-axiom)
$$\frac{}{\Sigma \vdash \mathbf{0}}$$

(t-go)
$$\frac{\Sigma \vdash u : \mathtt{loc_V} \quad \Sigma \vdash P}{\Sigma \vdash \text{go } u.P}$$

(t-ping)
$$\frac{\Sigma \vdash u : \mathtt{loc_V} \quad \Sigma \vdash P,\ Q}{\Sigma \vdash \text{ping } u.P \text{ else } Q}$$

Systems

(t-rest)
$$\frac{\mathtt{T} \neq \mathtt{loc_c} \quad \Gamma, n : \mathtt{T} \vdash N}{\Gamma \vdash (\nu\, n : \mathtt{T})N}$$

(t-par)
$$\frac{\Gamma \vdash N,\ M}{\Gamma \vdash N|M}$$

(t-proc)
$$\frac{\Gamma \vdash l : \mathtt{loc_V} \quad \Gamma \vdash P}{\Gamma \vdash l[\![P]\!]}$$

condition $\mathtt{T} \neq \mathtt{loc_c}$ in (t-nw) and (t-rest). This additional condition precludes the creation of *new confined* locations and the existence of *scoped confined* locations, thereby guaranteeing the secondary restriction of the type system discussed earlier, namely that all confined locations are free.

Example 3 (Type-checking Systems) *Let Γ_e denote the environment*

$$\Gamma_e = \begin{cases} l : \mathtt{loc}^{\mathtt{a}}_{\mathtt{p}},\ k_1 : \mathtt{loc}^{\mathtt{a}}_{\mathtt{c}},\ k_2 : \mathtt{loc}^{\mathtt{a}}_{\mathtt{c}},\ k_3 : \mathtt{loc}^{\mathtt{a}}_{\mathtt{c}}, \\ req : \mathtt{ch_p}\langle \mathtt{P}_1, \mathtt{ch_p}\langle \mathtt{P}_1\rangle\rangle,\ ret : \mathtt{ch_p}\langle \mathtt{P}_1\rangle,\ v : \mathtt{P}_1,\ a : \mathtt{ch_p}\langle \mathtt{P}_2\rangle \end{cases}$$

where $\mathtt{P}_1$ and $\mathtt{P}_2$ are arbitrary public types. Then one can check

$$\Gamma_e \vdash \mathit{server}_i$$

for $i = 1 \ldots 3$, where server$_i$ is defined in the Introduction, provided the locally declared channels data and sync are declared at the types $\mathtt{ch_V}\langle \mathtt{T}, \mathtt{ch_p}\langle \mathtt{T}\rangle, \mathtt{loc_p}\rangle$ and $\mathtt{ch_V}\langle \mathtt{T}\rangle$ respectively, with arbitrary visibility $\mathtt{V}$.

Consider the alternative server

$$\mathit{serverBad} \Leftarrow \mathit{server}_1 \mid l[\![a!\langle k_1\rangle]\!]$$

which acts as server_1 *but also communicates the confined location* k_1 *on channel* a *at* l*. It is easy to see that our type system rejects* $\mathsf{serverBad}$*, that is*

$$\Gamma_e \nvdash \mathsf{serverBad}$$

because it outputs a confined value, k_1*, on a public channel* a*. We know that the object type of the public channel* a *must be public,* $\mathtt{P}_2$*, which cannot be matched with the confined type of the value outputted* k_1 *when applying the typing rule (t-out).*

Consider another server

$$\mathsf{serverBad2} \;\Leftarrow\; (\nu\, k_1 : \mathtt{loc}^{\mathtt{a}}_{\mathtt{c}})\, \mathsf{server}_1$$

If we consider the slightly modified type environment

$$\Gamma'_e = \Gamma_e \backslash k_1 : \mathtt{loc}^{\mathtt{a}}_{\mathtt{c}}$$

removing the type assignment to k_1 *from* Γ_e*, then our type system also rejects* $\mathsf{serverBad2}$*, that is*

$$\Gamma'_e \nvdash \mathsf{serverBad2}$$

because the confined location k_1 *is scoped.*

The main property required of our type system is that, in some sense, typing is preserved by substitution of values for variables, provided their types are in agreement:

Lemma 4 (Substitution) *If* $\Sigma, x\!:\!\mathtt{U} \vdash P$ *and* $\Sigma \vdash v\!:\!\mathtt{U}$*, then* $\Sigma \vdash P\{^{v}\!/\!_{x}\}$

PROOF. By induction on the derivation of $\Sigma, x\!:\!\mathtt{U} \vdash P$.

Reduction Semantics:

Definition 5 *The pair* $\Gamma \triangleright M$ *is called a* configuration *if all the free names in* M *have a type assigned to them in* Γ*, that is* $\mathbf{fn}(M) \subseteq \mathbf{dom}(\Gamma)$*. It is called a* valid *configuration if in addition* $\Gamma \vdash M$*.*

Reductions then take the form of a binary relation over configurations

$$\Gamma \triangleright N \quad \longrightarrow \quad \Gamma' \triangleright N' \tag{4}$$

defined in terms of the reduction rules in Table 3, whereby systems reduce with respect to the *liveness status* of the locations in Γ; here we should emphasise that the reductions depend in no way on the type information in Γ, other than the liveness annotations on location types.

Table 3. *Reduction Rules for DπLoc*

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

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

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

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

(r-eq)
$$\frac{}{\Gamma \triangleright l[\![\text{if } u\!=\!u \text{ then } P \text{ else } Q]\!] \longrightarrow \Gamma \triangleright l[\![P]\!]}$$

(r-neq)
$$\frac{}{\Gamma \triangleright l[\![\text{if } u\!=\!v \text{ then } P \text{ else } Q]\!] \longrightarrow \Gamma \triangleright l[\![Q]\!]}\; u \neq v$$

(r-go)
$$\frac{}{\Gamma \triangleright l[\![\text{go } k.P]\!] \longrightarrow \Gamma \triangleright k[\![P]\!]}\; \Gamma \vdash k : \textbf{alive}$$

(r-ngo)
$$\frac{}{\Gamma \triangleright l[\![\text{go } k.P]\!] \longrightarrow \Gamma \triangleright k[\![\mathbf{0}]\!]}\; \Gamma \nvdash k : \textbf{alive}$$

(r-ping)
$$\frac{}{\Gamma \triangleright l[\![\text{ping } k.P \text{ else } Q]\!] \longrightarrow \Gamma \triangleright l[\![P]\!]}\; \Gamma \vdash k : \textbf{alive}$$

(r-nping)
$$\frac{}{\Gamma \triangleright l[\![\text{ping } k.P \text{ else } Q]\!] \longrightarrow \Gamma \triangleright l[\![Q]\!]}\; \Gamma \nvdash k : \textbf{alive}$$

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

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

(r-ctxt-rest)
$$\frac{\Gamma,\; n : \mathtt{T} \triangleright N \;\longrightarrow\; \Gamma',\; n : \mathtt{T} \triangleright M}{\Gamma \triangleright (\nu\, n : \mathtt{T})N \;\longrightarrow\; \Gamma' \triangleright (\nu\, n : \mathtt{T})M}$$

(r-ctxt-par)
$$\frac{\Gamma \triangleright N \;\longrightarrow\; \Gamma' \triangleright N'}{\begin{array}{l}\Gamma \triangleright N|M \;\longrightarrow\; \Gamma' \triangleright N'|M \\ \Gamma \triangleright M|N \;\longrightarrow\; \Gamma' \triangleright M|N'\end{array}}$$

In these rules, in order to emphasise the meaning and abstract away from visibility type information, we write
$$\Gamma \vdash l : \textbf{alive}$$
instead of $\Gamma \vdash l : \mathtt{loc}^{\mathtt{a}}_{\mathtt{V}}$; we also write $\Gamma \nvdash l : \textbf{alive}$ instead of $\Gamma \vdash l : \mathtt{loc}^{\mathtt{d}}_{\mathtt{V}}$. Thus all reduction rules assume the location where the code is executing is alive. Moreover, (r-go), (r-ngo), (r-ping) and (r-nping) reduce according to the status of the remote

Table 4. *Structural Rules for DπLoc*

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

Table 5. *Error reductions in DπLoc*

(e-out)

$$\frac{\exists v_i \in V \;\; \text{such that} \;\; \Gamma \vdash v_i\!:\!\mathtt{loc_c} \text{ or } \Gamma \vdash v_i\!:\!\mathtt{ch_c}\langle\tilde{\mathtt{W}}\rangle}{\Gamma \triangleright l[\![a!\langle V\rangle.P]\!] \longrightarrow_{\mathbf{err}}}$$

(e-scope)

$$\frac{}{\Gamma \triangleright (\nu\, l\!:\!\mathtt{loc^S_c})M \longrightarrow_{\mathbf{err}}}$$

(e-par)

$$\frac{\Gamma \triangleright M \longrightarrow_{\mathbf{err}}}{\begin{array}{c}\Gamma \triangleright M \,|\, N \longrightarrow_{\mathbf{err}}\\ \Gamma \triangleright N \,|\, M \longrightarrow_{\mathbf{err}}\end{array}}$$

(e-rest)

$$\frac{\Gamma,\; n : \mathtt{T} \triangleright M \longrightarrow_{\mathbf{err}}}{\Gamma \triangleright (\nu\, n\!:\!\mathtt{T})M \longrightarrow_{\mathbf{err}}}$$

location concerned. All the remaining rules are standard, including the use of a *structural equivalence* $\equiv$ between systems; see [4] for more details. The attentive reader should have noted that when using the rules in Table 3, whenever $\Gamma \triangleright N \longrightarrow \Gamma' \triangleright N'$ it can be deduced that Γ' always coincides with Γ. Even though this is certainly true, in later sections we will introduce reductions that change network status of the reduct $\Gamma' \triangleright N'$.

Proposition 6 (Subject Reduction) *If* $\Gamma \vdash M$ *and* $\Gamma \triangleright M \longrightarrow \Gamma \triangleright N$*, then* $\Gamma \vdash N$.

PROOF. By induction on the derivation of $\Gamma \triangleright M \longrightarrow \Gamma \triangleright N$. As usual the main difficulty occurs with the communication rules, (r-comm) and (r-rep), where the Substitution lemma, Lemma 4 is used. Treatment of (r-str) also requires us to prove that typeability is preserved by the structural equivalence.

We can also show that the type system does indeed fulfill its intended purpose. More specifically, in well-typed systems

- confined values will never be made public.

- confined locations are never scoped.

In Table 5 we formalise these notions of runtime errors, writing

$$\Gamma \triangleright N \longrightarrow_{\mathbf{err}}$$

to mean that, intuitively, N is either about to export some confined value on a public channel, (e-out), or currently holds a scoped confined location, (e-scope) - the remaining two rules in Table 5 are standard contextual rules. We show that such errors can never occur in a valid environment.

Proposition 7 (Type Safety) *Suppose $\Gamma \triangleright N$ is a valid configuration, that is $\Gamma \vdash N$. If $\Gamma \triangleright N \longrightarrow^{*} \Gamma \triangleright N'$ then $\Gamma \triangleright N' \not\longrightarrow_{\mathbf{err}}$*

PROOF. From Subject Reduction, Proposition 6, we know that $\Gamma \vdash N'$. It is also straightforward to show, using the rules in Figure 5, that $\Gamma \triangleright M \longrightarrow_{\mathbf{err}}$ implies $\Gamma \nvdash M$, from which the result follows.

In the remainder of the paper we will confine our attention to *valid* configurations; from the two previous propositions we know that they are preserved under reductions and they do not give rise to runtime errors.

Behavioural equivalence: As the appropriate semantic equivalence for DπLoc, we propose a contextual equivalence based on the standard notion of *reduction barbed congruence*, [11,8], which is adapted to the presence of configurations. More importantly, as explained in the Introduction, we refine it further so as to ensure that observations can only be made at public locations. This restriction is enforced using our type system.

For any Γ let $\mathbf{pub}(\Gamma)$ be the environment obtained by restricting to names which are assigned public types. Then we write $\Gamma \vdash_{\mathsf{obs}} O$, meaning intuitively that O is a valid observer with respect to Γ, whenever $\mathbf{pub}(\Gamma) \vdash O$.

Example 8 (Type System prevents Errors) *Referring back to Example 3, we have already seen that*

$$\Gamma_e \vdash \mathsf{server}_i$$

for $i = 1 \ldots 3$ but we can also check that the client*, defined in the Introduction, is also a valid observer with respect to $\mathbf{pub}(\Gamma_e)$ which translates to*

$$\mathbf{pub}(\Gamma_e) = l : \mathtt{loc}^{\mathtt{a}}_{\mathtt{p}},\ req : \mathtt{ch}_{\mathtt{p}}\langle \mathtt{P}_1, \mathtt{ch}_{\mathtt{p}}\langle \mathtt{P}_1 \rangle\rangle,\ ret : \mathtt{ch}_{\mathtt{p}}\langle \mathtt{P}_1 \rangle,\ v : \mathtt{P}_1,\ a : \mathtt{ch}_{\mathtt{p}}\langle \mathtt{P}_2 \rangle$$

We can thus show

$$\Gamma_e \vdash_{obs} \mathsf{client}$$

On the other hand consider

$$\textsf{obseverBad} \;\Leftarrow\; l[\![\textsf{go}\, k_1.\textsf{go}\, l.ok!\langle\rangle]\!]$$

Intuitively this should not be considered a valid observer because it uses the confined value k_1, and indeed we have

$$\Gamma_e \nvdash_{obs} \textsf{observerBad}$$

By prohibiting their use of any *confined values, valid observers are not only forced to be located to public locations, but also constrained to migrate to public locations only.*

Our typing system ensures that a valid observer can never obtain confined values under any sequence of interactions within a well-formed configuration. Consider again the alternative server serverBad*, already discussed in Example 3. If we compose* serverBad *in parallel with the valid observer*

$$\textsf{observerGood} \;\Leftarrow\; l[\![a?(x).\textsf{go}\, x.\textsf{go}\, l.ok!\langle\rangle]\!]$$

then one reduction step involving the communication of k_1 on channel a yields

$$\Gamma_e \triangleright \textsf{serverBad} \mid \textsf{observerGood} \;\longrightarrow\; \Gamma_e \triangleright \textsf{server}_1 \mid \textsf{observerBad}$$

Here the valid observer observerGood *reduces to the invalid observer* observerBad*, which has obtained knowledge of the confined location k_1.*

Our type system ensures that this never happens because, although we have

$$\Gamma_e \vdash_{obs} \textsf{observerGood}$$

our type system rejects serverBad *(Example 3).*

It is convenient to define our behavioural equivalence so that it can relate arbitrary configurations, which now are assumed to be valid; however, we would expect equivalent configurations to have the same *public interface*. We also only expect it to be preserved by composition with valid observers. This leads to the following definition.

Definition 9 (o-Contextual) *A relation over $\mathcal{R}$ configurations is called o-Contextual if, whenever $\Gamma \triangleright M \;\mathcal{R}\; \Gamma' \triangleright N$*

Interfaces: $\mathbf{pub}(\Gamma) = \mathbf{pub}(\Gamma')$

Parallel Observers: $\left.\begin{array}{l}\Gamma \triangleright M \mid O \;\mathcal{R}\; \Gamma' \triangleright N \mid O \\ \Gamma \triangleright O \mid M \;\mathcal{R}\; \Gamma' \triangleright O \mid N\end{array}\right\}$ *whenever $\Gamma \vdash_{obs} O$.*

Fresh extensions: $\Gamma, n :: \mathrm{P} \triangleright M \;\mathcal{R}\; \Gamma', n :: \mathrm{P} \triangleright N.$ [1]

[1] Recall that this implies n is fresh to both Γ and Γ'.

Since we want to limit observation to public resources, we restrict the standard notion of a barb and limit it to *public channels* at *public*, *live locations*.

Definition 10 (o-Barb) *$\Gamma \triangleright N \Downarrow^{o}_{a@l}$ denotes an o-observable barb by configuration $\Gamma \triangleright N$, on channel a at location l. This is true when $\Gamma \triangleright N \longrightarrow^{*} \Gamma \triangleright N'$ for some N' such that*

$$N' \equiv (\nu\, \tilde{n} : \tilde{\mathrm{T}}) M | l[\![a!\langle V \rangle.Q]\!]$$

where $\Gamma \vdash l : \mathtt{loc}^{\mathtt{a}}_{\mathtt{p}},\ a : \mathtt{ch}_{\mathtt{p}}\langle \tilde{\mathrm{P}} \rangle$. We say a relation over configurations preserves barbs *if:*

$$\left.\begin{array}{l} \Gamma \triangleright M \ \mathcal{R}\ \Gamma' \triangleright N \\ \Gamma \triangleright M \Downarrow^{o}_{a@l} \end{array}\right\} \textit{implies } \Gamma \triangleright N \Downarrow^{o}_{a@l}$$

The next definition is standard (see [8]), but is added for completeness.

Definition 11 (Reduction closure) *A relation over $\mathcal{R}$ configurations is* reduction-closed *whenever*

$$\left.\begin{array}{l} \Gamma_M \triangleright M \ \mathcal{R}\ \Gamma_N \triangleright N \\ \Gamma_M \triangleright M \longrightarrow \Gamma'_M \triangleright M' \end{array}\right\} \textit{implies } \Gamma_N \triangleright N \longrightarrow^{*} \Gamma'_N \triangleright N'$$

for some configuration $\Gamma'_N \triangleright N'$ such that $\Gamma'_M \triangleright M' \ \mathcal{R}\ \Gamma'_N \triangleright N'$.

Combining these we obtain our touchstone equivalence for DπLoc:

Definition 12 (Reduction barbed congruence) *Let $\cong$ be the largest relation between configurations which is:*

- *o-contextual*
- *reduction-closed*
- *preserves o-barbs*

Example 13 (Equivalent Configurations with Restricted View) *Our definition of reduction barbed congruence allows us to limit observations to certain locations, thereby allowing a* partial view *of a system. For instance, even though the equivalences defined in [4] could discriminate between server_1, server_2 and server_3 running on a network without failure, we can now say that these servers are observationally equivalent if observations are limited to location l. More specifically, it turns out that*

$$\Gamma_e \triangleright \mathsf{server}_i \cong \Gamma_e \triangleright \mathsf{server}_j \quad \textit{for } i, j \in \{1, 2, 3\}$$

where Γ_e, previously defined in Example 3, is limiting observations to the only public location, l; the locations k_1, k_2 and k_3 are confined in Γ_e.

3 Defining Fault Tolerance

We now give contextual definitions of fault-tolerance in the style of (1), outlined in the Introduction. We use the touchstone behavioural equivalence, Definition 12, to compare failure-free and failure-induced configurations. We also quantify over all possible fault contexts, special contexts that induce faults. The specific form of these fault contexts embody our assumptions about faults, which in turn determine the nature of our fault tolerance definitions.

Our definitions of fault tolerance are based on the clear separation of locations into two disjoint sets: *reliable* locations and *unreliable* locations. Reliable locations are assumed to be immortal, whereas unreliable locations may be dead already (permanently in fail-stop fashion[17]) or may die in future. An important assumption of our fault tolerance definitions is that the separation between reliable and unreliable locations happens once, *at the start of the analysis*, based on the location information at that moment; intuitively, we pick a subset from the *free* locations which we assume to be unreliable. Since scoped locations are *not* known to us at this stage, we cannot tag them as unreliable, which is why the type system in Section 2 precluded scoped unreliable(confined) locations. Once unreliable locations are determined, we take a prescriptive approach and force observers to reside at reliable locations only, thereby ensuring that they never incur failure *directly*. To put it more succinctly, observations never fail.

Nevertheless, an observer can still be affected by failure *indirectly*. This happens when an observer interacts with system code residing at public locations whose behaviour *depends*, in some way or another, on code residing at unreliable locations. In such a setting, fault tolerance would be a property of the system code at public locations, which *preserves* its behaviour up to a *certain level* of fault, even when the unreliable code it depends on (residing at unreliable locations) fails. If we map public locations to reliable locations and confined locations as unreliable, then the framework developed in Section 2 fits our requirements for such a definition; the type system also ensures that this clear separation between reliable and unreliable code is preserved during interaction, by ensuring that an observer will never receive an unreliable location, when it is expecting a reliable one. In the remainder of the document we will interchangeably use the terms reliable and unreliable for public and confined types respectively.

Our first notion of *n*-fault-tolerance, formalising the intuition behind (1), is when the faulting context induces at most *n* location failures *prior to the execution of the system*. Of course, these failures must only be induced on locations which are confined, based on the prior assumption that public locations are reliable, and thus immortal. The implicit assumptions behind our first fault tolerance definition are that:

- either the unreliable locations have always been dead and no more failure will occur in future.
- or the frequency of failure occurrence, which often happens in bursts, is much lower than that of reduction steps, within a give period of time. Thus we can assume that computations will not be interleaved by further failures.

Formally, we define the operation $\Gamma - l$ as:

$$\Gamma - l \overset{\text{def}}{=} \begin{cases} \Gamma', l : \mathtt{loc}^{\mathsf{d}}_{\mathsf{c}} & \text{if } \Gamma = \Gamma', l : \mathtt{loc}^{\mathsf{a}}_{\mathsf{c}} \\ \Gamma & \text{otherwise} \end{cases}$$

Definition 14 (Static Fault Tolerance) *For any set of location names $\tilde{l}$ let $F^{\tilde{l}}_{S}(-)$ be the function which maps any network Γ to $\Gamma - \tilde{l}$; that is the environment obtained by ensuring that the status of any confined l_i in Γ is* dead. *We say $F^{\tilde{l}}_{S}(-)$ is a* valid static n-fault context *if the size of $\tilde{l}$ is* at most *n. A configuration $\Gamma \triangleright N$ is static n-fault tolerant if for every valid static n-fault context $F^{\tilde{l}}_{S}$*

$$\Gamma \triangleright N \cong F^{\tilde{l}}_{S}(\Gamma) \triangleright N$$

With this formal definition we can now examine the systems server_i, using the Γ_e defined in Example 3.

Example 15 (Static Fault Tolerance) *We can formally check that $\Gamma_e \triangleright \mathsf{server}_1$ is not static 1-fault tolerant because by considering the fault context $F^{k_1}_{S}$ we can show that*

$$\Gamma_e \triangleright \mathsf{server}_1 \ncong F^{k_1}_{S}(\Gamma_e) \triangleright \mathsf{server}_1$$

Similarly we can show that $\Gamma_e \triangleright \mathsf{server}_2$ is not 2-fault tolerant, by considering its behaviour in the static 2-fault tolerant context F^{k_1,k_2}_{S}. More specifically here let client denote the system

$$l[\![req!\langle v, ret\rangle.ret?(x).ok!\langle x\rangle]\!]$$

Then, assuming Γ_e has a suitable type for the channel ok, the configuration

$$\Gamma_e \triangleright \mathsf{server}_2 \,|\, client$$

can perform the barb ok@l, whereas with the configuration

$$\Gamma_e - \{k_1, k_2\} \triangleright \mathsf{server}_2 \,|\, client$$

this is not possible.

We can also examine other systems which employ passive *replication. The system $\mathsf{sPassive}$ defined below uses two identical replicas of the distributed database at k_1 and k_2, but treats the replica at k_1 as the* primary *replica and the one at k_2 as* *a* secondary *(backup) replica. Once again, the type of the scoped channel data is* $\mathtt{T} = \mathtt{ch_c}\langle \mathtt{P}, \mathtt{ch_p}\langle \mathtt{P}\rangle, \mathtt{loc_p}\rangle$*, where we recall that* $\mathtt{P}$ *denotes a public type.*

$$\mathit{sPassive} \Leftarrow (\nu\, \mathit{data}:\mathtt{T}) \left(\begin{array}{l} l\left[\!\!\left[\begin{array}{l} \mathit{serv}?(x,y).\mathsf{ping}\; k_1.\mathsf{go}\; k_1.\mathit{data}!\langle x,y,l\rangle \\ \qquad\qquad\qquad\qquad \mathsf{else}\;\mathsf{go}\; k_2.\mathit{data}!\langle x,y,l\rangle \end{array} \right]\!\!\right] \\ \mid k_1[\![\mathit{data}?(x,y,z).\mathsf{go}\, z\,.y!\langle f(x)\rangle]\!] \\ \mid k_2[\![\mathit{data}?(x,y,z).\mathsf{go}\, z\,.y!\langle f(x)\rangle]\!] \end{array} \right)$$

The coordinating interface at l uses the ping construct to detect failures *in the primary replica: if* k_1 *is alive, the request is sent to the primary replica and the secondary replica at* k_2 *is* not invoked*; if, on the other hand, the primary replica is dead, then the passive replica at* k_2 *is promoted to a primary replica and the request is sent to it. This implementation saves on* time redundancy *since, for any request, only one replica is invoked.*

It turns out that $\Gamma_e \triangleright \mathit{sPassive}$ *is static 1-fault tolerant, as are* $\Gamma_e \triangleright \mathit{server}_2$ *and* $\Gamma_e \triangleright \mathit{server}_3$*. The latter,* $\Gamma_e \triangleright \mathit{server}_3$*, is also static* 2*-fault tolerant. However, establishing positive results is problematic because the definition of* $\cong$ *quantifies over all valid observers and over all possible static n-fault contexts. The problems associated with the quantification over all valid observers is addressed in the next section, when we give a co-inductive characterisation of* $\cong$*.*

Our second notion of *n*-fault tolerance is based on faults that may occur asynchronously *at any stage during* the execution of a system. This translates to a weaker assumption about the periodicity of faults than that underlying Definition 14. Also, this second fault tolerance definition does not assume any dependency between faults. It only assumes an upper-bound of faults and that faults are permanent.

To formalise this notion we to extend the language DπLoc, by introducing a new process called kill. Then the new system $l[\![\mathsf{kill}]\!]$ simply asynchronously kills location l. The reduction semantics, and typing rule, for this new construct is given in Table 6. We use DπLoc$_e$ to denote the extended language, and note that because of the typing rule only confined locations can be killed. In particular this means that if O is an observer, that is $\Gamma \vdash_{\mathsf{obs}} O$, then O does not have the power to kill any locations. The net effect is that reduction barbed congruence, $M \cong N$, only compares the systems M, N from DπLoc$_e$, in contexts which have no power to kill locations in M or N, although these systems themselves may have this power.

Definition 16 (Dynamic Fault Tolerance) *For any set of locations* $\tilde{l}$ *let* $F_D^{\tilde{l}}(-)$ *denote the function which maps the system M to* $M \mid l_1[\![\mathsf{kill}]\!] \mid \ldots \mid l_n[\![\mathsf{kill}]\!]$ *for any* $l_i \in \tilde{l}$*. Such a function is said to be a valid dynamic n-fault context if the size of* $\tilde{l}$ *is at most n. A configuration* $\Gamma \triangleright N$ *is dynamic n-fault tolerant if for every valid dynamic*

Table 6. *The kill construct*

(t-kill)
$$\frac{\Gamma \vdash l : \mathtt{loc_c}}{\Gamma \vdash l[\![\mathsf{kill}]\!]}$$

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

n-fault context

$$\Gamma \triangleright M \cong \Gamma \triangleright F_D^{\tilde{l}}(M)$$

Example 17 (Dynamic Fault Tolerance) *As we shall see later on, it turns out that $\Gamma_e \triangleright \mathsf{server}_2$ and $\Gamma_e \triangleright \mathsf{server}_3$ are both dynamic fault tolerant up to 1 fault; $\Gamma_e \triangleright \mathsf{server}_3$ is also dynamic 2-fault tolerant. We however note that, contrary to the static case, $\Gamma_e \triangleright \mathsf{sPassive}$ is* ***not*** *dynamic 1-fault tolerant. This can be proved using $F_D^{k_1}(-)$ and showing that*

$$\Gamma_e \triangleright \mathsf{sPassive} \not\cong \Gamma_e \triangleright \mathsf{sPassive} \,|\, k_1[\![\mathsf{kill}]\!]$$

The equivalence does not hold because k_1 may fail after *sPassive tests for its status. In this case, the backup database at k_2 is never queried and thus an answer will never reach l.*

However, this is not the case for any passive replication server with two replicas. For instance, we can consider sMonitor, defined as

$$\mathsf{sMonitor} \Leftarrow (\nu\, data : \mathtt{T}) \left(\begin{array}{l} l \left[\!\!\left[serv?(x,y).(\nu\, sync : \mathtt{R}) \left(\begin{array}{l} \mathsf{go}\, k_1.data!\langle x, sync, l\rangle \\ |\; \mathsf{mntr}\, k_1.\mathsf{go}\, k_2.data!\langle x, sync, l\rangle \\ |\; sync?(z).y!\langle z\rangle \end{array} \right) \right]\!\!\right] \\ |\; k_1[\![data?(x,y,z).\mathsf{go}\, z\ .y!\langle f(x)\rangle]\!] \\ |\; k_2[\![data?(x,y,z).\mathsf{go}\, z\ .y!\langle f(x)\rangle]\!] \end{array} \right)$$

where again, $\mathtt{T} = \mathtt{ch}\langle \mathtt{P}, \mathtt{ch_p}\langle \mathtt{P}\rangle, \mathtt{loc_p}\rangle$ and the type of the synchronisation channel sync is $\mathtt{R} = \mathtt{ch_c}\langle \mathtt{P}\rangle$. This passive replication server still treats the database at k_1 as the primary replica and the database at k_2 as the secondary replica. However, instead of a single *ping test on the primary replica at k_1, it uses a* monitor *process for failure detection*

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

The monitor process $\mathsf{mntr}\, k.P$ repeatedly *tests the status of the monitored location (k) and continues as P only when k becomes dead. Due to the asynchrony across locations, in sMonitor there are cases when we still receive two database answers at l (the queried database at k_1 may first return an answer and then fail). At this point the server interface detects the failure and queries the backup at k_2 which, in turn,* *returns a second answer. sMonitor solves this problem by synchronising multiple answers from replicas with the channel sync, similar to server_2 and server_3 in Example 1.*

It turns out that $\Gamma_e \triangleright \mathsf{sMonitor}$ is also dynamic 1-fault tolerant, but as in the case of $\Gamma_e \triangleright \mathsf{server}_2$ and $\Gamma_e \triangleright \mathsf{server}_3$, such a positive result is hard to show because $\cong$ quantifies over all possible observers.

4 Proof Techniques for Fault Tolerance

We define a labelled transition system (lts) for DπLoc$_e$, which consists of a collection of transitions over (closed) configurations, $\Gamma \triangleright N \xrightarrow{\mu} \Gamma' \triangleright N'$, where μ can be any of the following:

- internal action, τ
- output action, $(\tilde{n} : \tilde{\mathtt{T}})l : a!\langle V\rangle$
- input action, $(\tilde{n} : \tilde{\mathtt{T}})l : a?(V)$

where the names $\tilde{n}$ bind a subset of the names in V in both the input and output transitions. Bound names in output labels denote scope extruded names whereas bound names in input labels denote fresh names introduced by the (implicit) observer. These three transitions are defined inductively by the rules given in Table 7 and Table 8, inspired by [9,8,4], but with a number of differences.

In accordance with Definition 10 (observable barbs) and Definitions 9 (valid observers), (l-in) and (l-out) restrict external communication to *public* channels at *public* locations, where the notation $\Gamma \vdash_{\mathsf{obs}} l$ and $\Gamma \vdash_{\mathsf{obs}} a$ denote $\Gamma \vdash l : \mathtt{loc}^{\mathtt{p}}$ and $\Gamma \vdash a : \mathtt{ch}_{\mathtt{p}}\langle\tilde{\mathtt{P}}\rangle$ respectively, for some types $\tilde{\mathtt{P}}$. Furthermore, in (l-in) we require that the types of the values received, V, match the object type of channel a; since a is public and configurations are well-typed, this also implies that V are public values defined in Γ. More prosaically, the object type of the input channel is $\tilde{\mathtt{P}}$, and by Lemma 4, we know the reduct is still well-typed. The restriction on the rule for output transitions, together with the assumption that all configurations are well-typed, also means that in (l-open) we only scope extrude public values. Contrary to [4], the lts does not allow external killing of locations (through the label $\mathsf{kill} : l$) since public locations are reliable and never fail. Finally, the transition rule for internal communication, (l-par-comm), uses an overloaded function $\uparrow(-)$ for inferring input/output capabilities of the sub-systems: when applied to types, $\uparrow(\mathtt{T})$ transforms all the type tags to public ($\mathtt{p}$); when applied to environments, $\uparrow(\Gamma)$ changes all the

Table 7. *Operational Rules(1) for Typed DπLoc*

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

(l-in)
$$\frac{}{\Gamma \triangleright l[\![a?(X).P]\!] \xrightarrow{l:a?(V)} \Gamma \triangleright l[\![P\{^{V}/_{X}\}]\!]}\ \Gamma \vdash_{\text{obs}} l,\ \Gamma \vdash a : \mathtt{ch_P}\langle\tilde{\mathtt{P}}\rangle,\ V : \tilde{\mathtt{P}}$$

(l-in-rep)
$$\frac{}{\Gamma \triangleright l[\![*a?(X).P]\!] \xrightarrow{\tau} \Gamma \triangleright l[\![a?(X).(P| * a?(X).P)]\!]}$$

(l-out)
$$\frac{}{\Gamma \triangleright l[\![a!\langle V\rangle.P]\!] \xrightarrow{l:a!\langle V\rangle} \Gamma \triangleright l[\![P]\!]}\ \Gamma \vdash_{\text{obs}} l, a$$

(l-fork)
$$\frac{}{\Gamma \triangleright l[\![P|Q]\!] \xrightarrow{\tau} \Gamma \triangleright l[\![P]\!] \mid l[\![Q]\!]}$$

(l-eq)
$$\frac{}{\Gamma \triangleright l[\![\text{if } u = u \text{ then } P \text{ else } Q]\!] \xrightarrow{\tau} \Gamma \triangleright l[\![P]\!]}$$

(l-neq)
$$\frac{}{\Gamma \triangleright l[\![\text{if } u = v \text{ then } P \text{ else } Q]\!] \xrightarrow{\tau} \Gamma \triangleright l[\![Q]\!]}\ u \neq v$$

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

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

(l-go)
$$\frac{}{\Gamma \triangleright l[\![\text{go } k.P]\!] \xrightarrow{\tau} \Gamma \triangleright k[\![P]\!]}\ \Gamma \vdash k : \textbf{alive}$$

(l-ngo)
$$\frac{}{\Gamma \triangleright l[\![\text{go } k.P]\!] \xrightarrow{\tau} \Gamma \triangleright k[\![\mathbf{0}]\!]}\ \Gamma \nvdash k : \textbf{alive}$$

(l-ping)
$$\frac{}{\Gamma \triangleright l[\![\text{ping } k.P \text{ else } Q]\!] \xrightarrow{\tau} \Gamma \triangleright l[\![P]\!]}\ \Gamma \vdash k : \textbf{alive}$$

(l-nping)
$$\frac{}{\Gamma \triangleright l[\![\text{ping } k.P \text{ else } Q]\!] \xrightarrow{\tau} \Gamma \triangleright l[\![Q]\!]}\ \Gamma \nvdash k : \textbf{alive}$$

types to public types in the same manner. The definitions for these operations are

$$\uparrow(\mathtt{T}_1, \ldots, \mathtt{T}_n) \overset{\text{def}}{=} (\uparrow(\mathtt{T}_1), \ldots, \uparrow(\mathtt{T}_n)) \qquad \uparrow(\mathtt{T}) \overset{\text{def}}{=} \begin{cases} \mathtt{ch_P}\langle\uparrow(\tilde{\mathtt{R}})\rangle & \text{if } \mathtt{T} = \mathtt{ch_V}\langle\tilde{\mathtt{R}}\rangle \\ \mathtt{loc^S_P} & \text{if } \mathtt{T} = \mathtt{loc^S_V} \end{cases}$$
$$\uparrow(\Gamma, n : \mathtt{T}) \overset{\text{def}}{=} \uparrow(\Gamma), n : \uparrow(\mathtt{T})$$

Table 8. *Operational Rules (2) for Typed DπLoc*

(l-open)

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

(l-weak)

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

(l-rest)

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

(l-par-ctxt)

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

(l-par-comm)

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

All the remaining rules are simplified versions of the corresponding rules in [4].

Using the lts of actions we can now define, in the standard manner, *weak bisimulation equivalence* over configurations. Our definition uses the standard notation for weak actions; we use $\Longrightarrow$ as a shorthand for the reflexive transitive closure on silent transitions $\xrightarrow{\tau}{}^*$. Hence, $\overset{\mu}{\Longrightarrow}$ denotes $\Longrightarrow\xrightarrow{\mu}\Longrightarrow$, and $\overset{\widehat{\mu}}{\Longrightarrow}$ denotes $\xrightarrow{\tau}{}^*$ if $\mu = \tau$ and $\overset{\mu}{\Longrightarrow}$ otherwise.

Definition 18 (Weak bisimulation equivalence) *A relation $\mathcal{R}$ over configurations is called a* bisimulation *if whenever $\Gamma_M \triangleright M \;\mathcal{R}\; \Gamma_N \triangleright N$, then*

- $\Gamma_M \triangleright M \xrightarrow{\mu} \Gamma'_M \triangleright M'$ *implies* $\Gamma_N \triangleright N \overset{\hat{\mu}}{\Longrightarrow} \Gamma'_N \triangleright N'$ *such that* $\Gamma'_M \triangleright M' \;\mathcal{R}\; \Gamma'_N \triangleright N'$
- $\Gamma_N \triangleright N \xrightarrow{\mu} \Gamma'_N \triangleright N'$ *implies* $\Gamma_M \triangleright M \overset{\hat{\mu}}{\Longrightarrow} \Gamma'_M \triangleright M'$ *such that* $\Gamma'_M \triangleright M' \;\mathcal{R}\; \Gamma'_N \triangleright N'$

Weak bisimulation equivalence, denoted by $\approx$, is taken to be the largest bisimulation.

Theorem 19 (Full Abstraction) *Suppose* $\mathbf{pub}(\Gamma) = \mathbf{pub}(\Gamma')$. *Then for any DπLoc$_e$ configurations* $\Gamma \triangleright M$, $\Gamma' \triangleright N$*:*

$$\Gamma \triangleright M \;\cong\; \Gamma' \triangleright N \quad \textit{if and only if} \quad \Gamma \triangleright M \approx \Gamma' \triangleright N$$

PROOF. *(Outline)*To prove

$$\Gamma \triangleright M \approx \Gamma' \triangleright N \quad \text{implies} \quad \Gamma \triangleright M \cong \Gamma' \triangleright N$$

we show that $\approx$ satisfies all the defining properties of $\cong$ (Definition 12). The most involved task is showing that $\approx$ is *o-Contextual* (Definition 9). This has already been done in [4] for more complex contexts. Following the approach there, we inductively define a relation $\mathcal{R}$ as the least relation over configurations satisfying:

$$\mathcal{R} = \left\{ \begin{array}{l|l} \langle \Gamma_1 \triangleright M_1,\ \Gamma_2 \triangleright M_2 \rangle & \Gamma_1 \triangleright M_1 \approx \Gamma_2 \triangleright M_2,\ \ \mathbf{pub}(\Gamma_1) = \mathbf{pub}(\Gamma_2) \\ \begin{array}{l} \langle \Gamma_1 \triangleright M_1|O,\ \Gamma_2 \triangleright M_2|O \rangle \\ \langle \Gamma_1 \triangleright O|M_1,\ \Gamma_2 \triangleright O|M_2 \rangle \end{array} & \Gamma_1 \triangleright M_1 \,\mathcal{R}\, \Gamma_2 \triangleright M_2 \text{ and } \Gamma_1 \vdash_{\mathsf{obs}} O \\ \langle \Gamma_1, n{:}\mathtt{T} \triangleright M_1,\ \Gamma_2, n{:}\mathtt{T} \triangleright M_2 \rangle & \begin{array}{l} \Gamma_1 \triangleright M_1 \,\mathcal{R}\, \Gamma_2 \triangleright M_2, \\ n \text{ is fresh in } \Gamma_1, \Gamma_2 \end{array} \\ \langle \Gamma_1 \triangleright (\nu\, n{:}\mathtt{T})M_1,\ \Gamma_2 \triangleright (\nu\, n{:}\mathtt{U})M_2 \rangle & \Gamma_1, n{:}\mathtt{T} \triangleright M_1 \,\mathcal{R}\, \Gamma_2, n{:}\mathtt{U} \triangleright M_2 \end{array} \right\}$$

In $\mathcal{R}$ we add an extra clause from those given in Definition 9, namely the last one for name scoping. We then show that $\mathcal{R}$ is a bisimulation; since $\approx$ is the largest possible bisimulation it follows that $\mathcal{R} \subseteq \approx$. Because of the definition of $\mathcal{R}$ it then follows that $\approx$, when confined to configurations with the same public interface, is o-Contextual.

The proof for the converse,

$$\Gamma \triangleright M \cong \Gamma' \triangleright N \quad \text{implies} \quad \Gamma \triangleright M \approx \Gamma' \triangleright N$$

relies on the notion of *definability*, that is, for every action, relative to a type environment Γ, there is an observer which uses the public knowledge of Γ to completely characterise the effect of that action. In our case, we only need to prove definability for input/output actions, which has already been done for a more complex setting in [4]. For instance the context which characterises the input transition labeled by $(\tilde{n} : \tilde{\mathtt{T}})l : a?(V)$ would be

$$[-] \mid O$$

where O is the system $(\nu\, \tilde{n} : \tilde{\mathtt{T}})l[\![a!\langle V \rangle.\mathsf{go}\ k_0.\mathit{eureka}!\langle\rangle]\!]$ and k_0 and *eureka* are fresh public location and channel names, respectively. Because of the restrictions on the manner in which transitions can be inferred, we are assured that O is allowed as an observer, that is $\Gamma \vdash_{\mathsf{obs}} 0$.

Theorem 19 allows us to prove *positive* fault tolerance results by giving a bisimulation for every reduction barbed congruent pair required by Definitions 14 and 16.

Example 20 (Proving Static Fault Tolerance) *To show that $\Gamma_e \triangleright \mathsf{sPassive}$, defined earlier in Example 15, is static 1-fault tolerant we needed to show that $\mathsf{sPassive}$ preserves the same behaviour under any static 1-fault contexts. Now, by the definition of the operation $\Gamma - l$, we know that the only l we need to consider are cases where l is confined in Γ; otherwise $\Gamma - l = \Gamma$ and the relation we have to prove would be a simple case of the identity relation. For our specific case, since Γ_e has only 3 confined locations, we only need to consider three static 1-fault contexts, and by Theorem 19, showing that $\Gamma_e \triangleright \mathsf{sPassive}$ is static 1-fault tolerant boils down to constructing 3 witness bisimulations to show*

$$\Gamma \triangleright \mathit{sPassive} \cong (\Gamma - k_1) \triangleright \mathit{sPassive}$$
$$\Gamma \triangleright \mathit{sPassive} \cong (\Gamma - k_2) \triangleright \mathit{sPassive}$$
$$\Gamma \triangleright \mathit{sPassive} \cong (\Gamma - k_3) \triangleright \mathit{sPassive}$$

Here we give the witness relation for the most involved case, for k_1, and leave the other simpler cases for the interested reader. The witness relation is $\mathcal{R}$ defined as

$$\mathcal{R} \stackrel{\text{def}}{=} \{\langle \Gamma \triangleright \mathit{sPassive}, \Gamma - k_1 \triangleright \mathit{sPassive}\rangle\} \cup \left(\bigcup_{n,m \in \textsc{Names}} \mathcal{R}'(n,m) \right)$$

$$\mathcal{R}'(x,y) \stackrel{\text{def}}{=} \left\{ \begin{array}{ll} \Gamma \triangleright (\nu d) l[\![\mathit{Png}(x,y)]\!] \mid R_1 \mid R_2 & , \Gamma - k_1 \triangleright (\nu d) l[\![\mathit{Png}(x,y)]\!] \mid R_1 \mid R_2 \\ \Gamma \triangleright (\nu d) l[\![Q_1(x,y)]\!] \mid R_1 \mid R_2 & , \Gamma - k_1 \triangleright (\nu d) l[\![Q_2(x,y)]\!] \mid R_1 \mid R_2 \\ \Gamma \triangleright (\nu d) k_1[\![d!\langle x,y,l\rangle]\!] \mid R_1 \mid R_2 & , \Gamma - k_1 \triangleright (\nu d) k_2[\![d!\langle x,y,l\rangle]\!] \mid R_1 \mid R_2 \\ \Gamma \triangleright (\nu d) k_1[\![\mathit{go}\, l.y!\langle f(x)\rangle]\!] \mid R_2 & , \Gamma - k_1 \triangleright (\nu d) R_1 \mid k_2[\![\mathit{go}\, l.y!\langle f(x)\rangle]\!] \\ \Gamma \triangleright (\nu d) l[\![y!\langle f(x)\rangle]\!] \mid R_2 & , \Gamma - k_1 \triangleright (\nu d) R_1 \mid l[\![y!\langle f(x)\rangle]\!] \\ \Gamma \triangleright (\nu d) R_2 & , \Gamma - k_1 \triangleright (\nu d) R_1 \end{array} \right\}$$

where d stands for data and

$$\begin{aligned} \mathit{Png}(x,y) &\Leftarrow \mathit{ping}\, k_1.Q_1(x,y)\ \mathit{else}\ Q_2(x,y) \\ Q_i(x,y) &\Leftarrow \mathit{go}\, k_i.d!\langle x,y,l\rangle \\ R_i &\Leftarrow k_i[\![d?(x,y,z).\mathit{go}\, z.y!\langle f(x)\rangle]\!] \end{aligned}$$

$\mathcal{R}$ is the union of all the relations $\mathcal{R}'(n,m)$ where n, m denote the possible names for the value and return channel that are received on channel serv.

To facilitate our presentation, the general form of every $\mathcal{R}'(n,m)$ is described through $\mathcal{R}'(x,y)$, a relation between configurations having two free variables, x and y; each $\mathcal{R}'(n,m)$ is obtained by instantiating x and y for n and m respectively. In a similar fashion, $\mathcal{R}'(x,y)$ uses convenient abbreviations for processes, such as $\mathsf{Png}(x,y)$ - a shorthand for a process with free variables x and y. In $\mathcal{R}'(n,m)$ this shorthand denotes the closed process $\mathsf{Png}(n,m)$.

Every $\mathcal{R}'(n,m)$ relates $\Gamma \triangleright \mathsf{sPassive}$ and $(\Gamma - k_1) \triangleright \mathsf{sPassive}$ and captures the essential mechanism of how $(\Gamma - k_1) \triangleright \mathsf{sPassive}$ uses redundancy to preserve the same observable behaviour of $\Gamma \triangleright \mathsf{sPassive}$. In this mapping, all the requests are serviced by the primary replica at k_1 in $\Gamma \triangleright \mathsf{sPassive}$, whereas they are serviced by the secondary replica at k_2 in $(\Gamma - k_1) \triangleright \mathsf{sPassive}$.

5 Generic Techniques for Dynamic Fault Tolerance

In spite of the benefits gained from proof techniques developed in Section 4, proving positive fault tolerance results still entails a lot of unnecessary repeated work. This problem is mainly due to Definition 14 and Definition 16, which quantify over all *fault contexts*. The universal quantification of fault contexts can generally be bounded, as in Example 20, through the n index of the fault contexts (indicating the maximum failure to induce) and by the number of unreliable locations defined in the environment, which limits the witness bisimulations we need to construct. Despite such bounds on fault contexts, we are still required to perform much redundant work. For instance, to prove that server_3 is 2-fault tolerant, we need to provide 6 bisimulation relations[2], one for every different case in

$$\Gamma \triangleright \mathsf{server}_3 \;\cong\; \Gamma \triangleright \mathsf{server}_3 | k_i[\![\mathsf{kill}]\!] | k_j[\![\mathsf{kill}]\!] \qquad \text{for } i, j \in \{1, 2, 3\}$$

A closer inspection of the required relations reveals that there is a lot of overlap between them. For instance, in the witness bisimulation where $i = 1, j = 2$ and in the witness bisimulation for $i = 1, j = 3$, in each case part of our analysis requires us to consider the behavior of server_3 under a setting where k_1 dies first, leading to a large number of bisimilar tuples which are common to both witness bisimulations. These overlapping states would be automatically circumvented if we require a single relation that is somewhat the merging of all of these separate relations.

Hence, in this section we reformulate our fault tolerance definition for dynamic fault tolerance (the most demanding) to reflect such a merging of relations; a similar

[2] The cases where the number n is less than 2 (in our case $n = 1$) is handled by the instance where both i and j are the same location; it is not hard to show that $\Gamma \triangleright M | k_i[\![\mathsf{kill}]\!] | k_i[\![\mathsf{kill}]\!] \approx \Gamma \triangleright M | k_i[\![\mathsf{kill}]\!]$ for $i \in \{1, 2, 3\}$ since a location cannot be killed twice.

Table 9. *Fail Silent Transition Rule DπLoc*

(l-fail)

$$\frac{}{\langle \Gamma, n\rangle \triangleright N \xrightarrow{\tau} \langle \Gamma - k, n-1\rangle \triangleright N}\ \Gamma \vdash k{:}\mathtt{loc}^{\mathtt{a}}_{\mathtt{c}}$$

definition for the static case should be amenable to similar treatment. We start by defining *extended configurations*, which have the form

$$\langle \Gamma, n\rangle \triangleright M$$

where M is a system from DπLoc and $\Gamma \triangleright M$ is a (valid) configuration. Intuitively, the extended configuration above denotes a system M, without any sub-systems of the form $l[\![\mathsf{kill}]\!]$, that is running on the network Γ, where at most n unreliable locations may fail. The additional network information in the form of a fault bound gives us an *upper limit* on the unreliable locations that *may still fail*. It provides a more succinct way of expressing dynamic failure, without recurring to the fault inducing code of the form $l[\![\mathsf{kill}]\!]$. More specifically, it allows us to express how many unreliable locations may still fail, in line with Definition 16, without committing ourselves as to which of these locations will fail, as is the case when using fault contexts. This leads to an alternative definition for dynamic fault tolerance that is easier to work with.

The network fault upper bound gives us further advantages. For instance it gives us more control when, after a possibly empty sequence of transitions, we reach configurations with $n = 0$; it less obvious to discern this from systems containing asynchronous kills $l[\![\mathsf{kill}]\!]$. In these extreme cases, we can treat code at unreliable locations as reliable, since the network failure upper-bound guarantees that none of these will fail, thereby simplifying our analysis.

We define transitions between tuples of extended configurations as

$$\langle \Gamma, n\rangle \triangleright M \xrightarrow{\mu} \langle \Gamma', n'\rangle \triangleright M' \tag{5}$$

in terms of all the transition rules given in Tables 7 and 8, with the exception of (l-kill), which is replaced by the new transition (l-fail) defined in Table 9, describing dynamic failure. Even though the transitions in Table 7 and Table 8 are defined on configurations, they can be applied to extended configurations in the obvious way. For example, the previous transition (l-out) applied to extended configurations now reads

(l-out)

$$\frac{}{\langle \Gamma, n\rangle \triangleright l[\![a!\langle V\rangle.P]\!] \xrightarrow{l:a!\langle V\rangle} \langle \Gamma, n\rangle \triangleright l[\![P]\!]}\ \Gamma \vdash_{\mathsf{obs}} l, a$$

We note that, for all transitions adapted from Tables 7 and 8, the network upper-bound does not change from the source to the target configuration.

Our previous configurations $\Gamma \triangleright M$ can be viewed as a simple instance of extended configurations of the form $\langle \Gamma, 0 \rangle \triangleright M$ where the maximum number dynamic failures that may occur at unreliable locations is 0. Also, using transitions defined over extended configurations (5), we smoothly carry over the previous definition of bisimulation, Definition 18, to extended configurations. We next give an alternative (co-inductive) definition of dynamic n-fault tolerance, based on extended configurations.

Definition 21 (Co-inductive Dynamic Fault Tolerance) *A configuration $\Gamma \triangleright N$ is co-inductive n (-dynamic) fault tolerant if*

$$\langle \Gamma, 0 \rangle \triangleright M \approx \langle \Gamma, n \rangle \triangleright M$$

Before we can use Definition 21 to prove dynamic fault tolerance, we need to show that the new definition is sound with respect to our previous definition of dynamic fault tolerance, Definition 16. This proof requires a lemma stating the correspondence between actions in configurations and actions in extended configurations.

Lemma 22 (Actions for Configurations and Extended Configurations) *Suppose M is a DπLoc system. Then for every $n \geq 0$,*

(1) $\Gamma \triangleright M \xrightarrow{\mu} \Gamma \triangleright M'$ *if and only if* $\langle \Gamma, n \rangle \triangleright M \xrightarrow{\mu} \langle \Gamma, n \rangle \triangleright M'$
(2) $\Gamma \triangleright M | l[\![\mathit{kill}]\!] \xrightarrow{\tau} \Gamma - l \triangleright M | l[\![\mathbf{0}]\!]$ *if and only if* $\langle \Gamma, n+1 \rangle \triangleright M \xrightarrow{\tau} \langle \Gamma - l, n \rangle \triangleright M.$

PROOF. The first statement is proved by induction the derivations of $\Gamma \triangleright M \xrightarrow{\mu} \Gamma \triangleright M'$ and $\langle \Gamma, n \rangle \triangleright M \xrightarrow{\mu} \langle \Gamma, n \rangle \triangleright M'$. The second is a simple analysis of the transitions involved. Note that here, because $\Gamma \triangleright M | l[\![\mathsf{kill}]\!]$ is assumed to be a configuration, we are assured that $\Gamma \vdash l : \mathtt{loc}^{\mathsf{a}}_{\mathsf{c}}$. See (t-kill) in Table 6.

Theorem 23 (Soundness of Co-inductive Dynamic Fault Tolerance)

$$\langle \Gamma, 0 \rangle \triangleright M_1 \approx \langle \Gamma, n \rangle \triangleright M_2 \quad \textit{implies} \quad \begin{cases} \textit{for any dynamic n-fault context } F^{\tilde{l}}_{D} \\ \Gamma \triangleright M_1 \cong \Gamma \triangleright F^{\tilde{l}}_{D}(M_2) \end{cases}$$

PROOF. Let $\mathcal{R}_n$ be a relation parameterised by a number n and defined as

$$\mathcal{R}_n \stackrel{\text{def}}{=} \left\{ \Gamma_1 \triangleright M_1 \;, \Gamma_2 \triangleright M_2 \,|\, \underbrace{l_i[\![\mathsf{kill}]\!] | \ldots | l_j[\![\mathsf{kill}]\!]}_{m} \;\middle|\; \begin{array}{l} \langle \Gamma_1, 0 \rangle \triangleright M_1 \approx \langle \Gamma_2, m \rangle \triangleright M_2 \\ \text{and } 0 \leq m \leq n \end{array} \right\}$$

We proceed by showing that $\mathcal{R}_n$ is a bisimulation over DπLoc configurations, up to structural equivalence; that is

$$\mathcal{R}_n \quad \subseteq \quad \approx \tag{6}$$

The required soundness result then follows because if

$$\langle \Gamma, 0 \rangle \triangleright M \approx \langle \Gamma, n \rangle \triangleright M$$

then by (6) and the definition of $\mathcal{R}_n$ we know that for every dynamic n-fault context $F_D^{\tilde{l}}(-)$, we also have

$$\Gamma \triangleright M \approx \Gamma \triangleright F_D^{\tilde{l}}(M)$$

Finally, by Theorem 19 we obtain

$$\Gamma \triangleright M \cong \Gamma \triangleright F_D^{\tilde{l}}(M)$$

which by Definition 16 means that $\Gamma \triangleright M$ is dynamic n-fault tolerant.

In the proof of (6), we focus on matching the actions of the right hand side configuration in $\mathcal{R}_n$; we leave the simpler case, that is matching the actions of the left hand side, to the interested reader. We thus assume

$$\Gamma_1 \triangleright M_1 \, \mathcal{R}_n \, \Gamma_2 \triangleright M_2 \mid \underbrace{l_i[\![\mathsf{kill}]\!] | \ldots | l_j[\![\mathsf{kill}]\!]}_{m} \tag{7}$$

for some $0 \leq m \leq n$ and we have

$$\Gamma_2 \triangleright M_2 \mid l_i[\![\mathsf{kill}]\!] | \ldots | l_j[\![\mathsf{kill}]\!] \xrightarrow{\mu} \Gamma_2' \triangleright M_2' \tag{8}$$

We have to show that

$$\Gamma_1 \triangleright M_1 \overset{\widehat{\mu}}{\Longrightarrow} \Gamma_1 \triangleright M_1' \;\text{ such that } \Gamma_1 \triangleright M_1' \, \mathcal{R}_n \, \Gamma_2' \triangleright M_2'$$

From the structure of $l_i[\![\mathsf{kill}]\!] | \ldots | l_j[\![\mathsf{kill}]\!]$, we deduce that there can be no interaction between M_2 and $l_i[\![\mathsf{kill}]\!] | \ldots | l_j[\![\mathsf{kill}]\!]$ and, by (l-par), we conclude that this action can be caused by either of the following actions:

(a) $\Gamma_2 \triangleright M_2 \xrightarrow{\mu} \Gamma_2 \triangleright M_2''$ where $M_2' \equiv M_2'' \mid l_i[\![\mathsf{kill}]\!] | \ldots | l_j[\![\mathsf{kill}]\!]$

(b) $\Gamma_2 \triangleright l_i[\![\mathsf{kill}]\!] | \ldots | l_j[\![\mathsf{kill}]\!] \xrightarrow{\mu} \Gamma_2 - l_k \triangleright l_i[\![\mathsf{kill}]\!] | \ldots | l_k[\![\mathbf{0}]\!] | \ldots | l_j[\![\mathsf{kill}]\!]$ where μ must be τ and $M_2' \equiv M_2 \mid \underbrace{l_i'[\![\mathsf{kill}]\!] | \ldots | l_j'[\![\mathsf{kill}]\!]}_{m-1}$

(a) In this case, from (7) and the definition of $\mathcal{R}_n$ we know

$$\langle \Gamma_1, 0 \rangle \triangleright M_1 \approx \langle \Gamma_2, m \rangle \triangleright M_2 \tag{9}$$

Also, by (a) and Lemma 22(1) we also have

$$\langle \Gamma_2, m \rangle \triangleright M_2 \xrightarrow{\mu} \langle \Gamma_2, m \rangle \triangleright M_2'' \tag{10}$$

From (9) we know that (10) can be matched by

$$\begin{aligned} &\langle\Gamma_1, 0\rangle \triangleright M_1 \overset{\widehat{\mu}}{\Longrightarrow} \langle\Gamma_1, 0\rangle \triangleright M_1' && (11)\\ \text{where}\quad &\langle\Gamma_1, 0\rangle \triangleright M_1' \;\approx\; \langle\Gamma_2, m\rangle \triangleright M_2'' && (12)\end{aligned}$$

From (11) and Lemma 22(1) we deduce

$$\Gamma_1 \triangleright M_1 \overset{\widehat{\mu}}{\Longrightarrow} \Gamma_1 \triangleright M_1'$$

and from (12) and the definition of $\mathcal{R}_n$ we also know

$$\Gamma_1 \triangleright M_1' \,\mathcal{R}_n\, \Gamma_2 \triangleright M_2'' \mid l_i[\![\mathsf{kill}]\!] | \ldots | l_j[\![\mathsf{kill}]\!]$$

(b) In this case, once again from (7) and the definition of $\mathcal{R}_n$ we know

$$\langle\Gamma_1, 0\rangle \triangleright M_1 \approx \langle\Gamma_2, m\rangle \triangleright M_2 \qquad (13)$$

Using (8) and Lemma 22(2) we can derive

$$\langle\Gamma_2, m\rangle \triangleright M_2 \xrightarrow{\tau} \langle\Gamma_2 - l_k, m-1\rangle \triangleright M_2 \qquad (14)$$

From (13) we know (14) can be matched by

$$\begin{aligned} &\langle\Gamma_1, 0\rangle \triangleright M_1 \overset{\widehat{\tau}}{\Longrightarrow} \langle\Gamma_1, 0\rangle \triangleright M_1' && (15)\\ \text{where}\quad &\langle\Gamma_1, 0\rangle \triangleright M_1' \;\approx\; \langle\Gamma_2 - l_k, m-1\rangle \triangleright M_2 && (16)\end{aligned}$$

From (15) and Lemma 22(1) we obtain

$$\Gamma_1 \triangleright M_1 \overset{\widehat{\tau}}{\Longrightarrow} \Gamma_1 \triangleright M_1'$$

and from (16) we get the required pairing

$$\Gamma_1 \triangleright M_1' \,\mathcal{R}_n\, \Gamma_2 - l_k \triangleright M_2 | \underbrace{l_i'[\![\mathsf{kill}]\!] | \ldots | l_j'[\![\mathsf{kill}]\!]}_{m-1}$$

With Theorem 23, we can now give a single witness bisimulation to show the dynamic fault tolerance of a configuration. However, a considerable number of transitions in these witness bisimulations turn out to be *confluent* silent transitions, meaning that they do not affect the set of transitions we can undertake in our bisimulations, now or in the future. One consequence of this fact is that reduction via such confluent moves produces bisimilar configurations. We thus develop up-to bisimulation techniques that abstract over such moves. This alleviates the burden of exhibiting our witness bisimulations and allows us to focus on the transitions that really matter.

Table 10. *β-Transition Rules (1) for Typed DπLoc*

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

(b-in-rep)

$$\frac{}{\langle\Gamma, n\rangle \triangleright l[\![*a?(X).P]\!] \overset{\tau}{\longmapsto}_\beta \langle\Gamma, n\rangle \triangleright l[\![a?(X).(P| * a?(Y).P\{^Y\!/\!_X\})]\!]}$$

(b-eq)

$$\frac{}{\langle\Gamma, n\rangle \triangleright l[\![\text{if } u\!=\!u \text{ then } P \text{ else } Q]\!] \overset{\tau}{\longmapsto}_\beta \langle\Gamma, n\rangle \triangleright l[\![P]\!]}$$

(b-neq)

$$\frac{}{\langle\Gamma, n\rangle \triangleright l[\![\text{if } u\!=\!v \text{ then } P \text{ else } Q]\!] \overset{\tau}{\longmapsto}_\beta \langle\Gamma, n\rangle \triangleright l[\![Q]\!]}\; u \neq v$$

(b-fork)

$$\frac{}{\langle\Gamma, n\rangle \triangleright l[\![P|Q]\!] \overset{\tau}{\longmapsto}_\beta \langle\Gamma, n\rangle \triangleright l[\![P]\!] \mid l[\![Q]\!]}$$

(b-new)

$$\frac{}{\langle\Gamma, n\rangle \triangleright l[\![(\nu\, n\!:\!\mathtt{T})P]\!] \overset{\tau}{\longmapsto}_\beta \langle\Gamma, n\rangle \triangleright (\nu\, n\!:\!\mathtt{T}) l[\![P]\!]}$$

(b-par)

$$\frac{\langle\Gamma, n\rangle \triangleright N \overset{\tau}{\longmapsto}_\beta \langle\Gamma', n'\rangle \triangleright N'}{\begin{array}{c}\langle\Gamma, n\rangle \triangleright N|M \overset{\tau}{\longmapsto}_\beta \langle\Gamma', n'\rangle \triangleright N'|M \\ \langle\Gamma, n\rangle \triangleright M|N \overset{\tau}{\longmapsto}_\beta \langle\Gamma', n'\rangle \triangleright M|N'\end{array}}$$

(b-rest)

$$\frac{\langle\Gamma, m\!:\!\mathtt{T}, n\rangle \triangleright N \overset{\tau}{\longmapsto}_\beta \langle\Gamma', m\!:\!\mathtt{T}, n'\rangle \triangleright N'}{\langle\Gamma, n\rangle \triangleright (\nu\, m\!:\!\mathtt{T})N \overset{\tau}{\longmapsto}_\beta \langle\Gamma', n'\rangle \triangleright (\nu\, m\!:\!\mathtt{T})N'}$$

Based on [3], we denote β-actions or β-moves as

$$\langle\Gamma, n\rangle \triangleright N \overset{\tau}{\longmapsto}_\beta \langle\Gamma', n\rangle \triangleright N'$$

These β-transitions are defined in Table 10 and Table 11. Our situation is more complicated than that in [3] because we also have to deal with failure. While we directly inherit local rules, such as (b-eq) and (b-fork), and context rules, such as (b-rest) and (b-par), we do not carry over distributed silent transitions such as code migration across locations. Instead, we here identify three sub-cases when migration is a β-move, that is

- when we are *migrating to* a location that is *dead*, (b-ngo).

Table 11. *β-Transition Rules (2) for Typed DπLoc*

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

(b-ngo)
$$\frac{}{\langle \Gamma, n\rangle \triangleright l[\![\mathsf{go}\ k.P]\!] \overset{\tau}{\longmapsto}_\beta \langle \Gamma, n\rangle \triangleright k[\![\mathbf{0}]\!]}\ \Gamma \nvdash k : \textbf{alive}$$

(b-go-pub)
$$\frac{}{\langle \Gamma, n\rangle \triangleright l[\![\mathsf{go}\ k.P]\!] \overset{\tau}{\longmapsto}_\beta \langle \Gamma, n\rangle \triangleright k[\![P]\!]}\ \Gamma \vdash_{\mathsf{obs}} l, \Gamma \vdash k : \textbf{alive}$$

(b-go-ff)
$$\frac{}{\langle \Gamma, 0\rangle \triangleright l[\![\mathsf{go}\ k.P]\!] \overset{\tau}{\longmapsto}_\beta \langle \Gamma, 0\rangle \triangleright k[\![P]\!]}\ \Gamma \vdash k : \textbf{alive}$$

(b-nping)
$$\frac{}{\langle \Gamma, n\rangle \triangleright l[\![\mathsf{ping}\ k.P\ \mathsf{else}\ Q]\!] \overset{\tau}{\longmapsto}_\beta \langle \Gamma, n\rangle \triangleright l[\![Q]\!]}\ \Gamma \nvdash k : \textbf{alive}$$

(b-ping-pub)
$$\frac{}{\langle \Gamma, n\rangle \triangleright l[\![\mathsf{ping}\ k.P\ \mathsf{else}\ Q]\!] \overset{\tau}{\longmapsto}_\beta \langle \Gamma, n\rangle \triangleright l[\![P]\!]}\ \Gamma \vdash_{\mathsf{obs}} k$$

(b-ping-ff)
$$\frac{}{\langle \Gamma, 0\rangle \triangleright l[\![\mathsf{ping}\ k.P\ \mathsf{else}\ Q]\!] \overset{\tau}{\longmapsto}_\beta \langle \Gamma, 0\rangle \triangleright l[\![P]\!]}\ \Gamma \vdash k : \textbf{alive}$$

- when we are *migrating from* a public location (thus immortal) to another live location (b-go-pub).
- when both the source and destination locations are alive and we cannot induce further dynamic failures, (b-go-ff).

Migration across locations is generally not a confluent move because it can be interfered with by failure. More specifically, the source location may fail before the code migrates, killing the code that could otherwise exhibit observable behaviour at the destination location. However, migrations to a dead location k, (b-ngo), are confluent because they all reduce to the system $k[\![\mathbf{0}]\!]$ which has no further transitions. Migrations from an immortal location, (b-go-pub), are confluent because failure can only affect the destination location; if the code migrates before the destination location fails then it crashes at the destination; if it migrates after the destination fails

Table 12. *Structural Equivalence Rules for Typed DπLoc Configurations*

(bs-comm)	$\langle\Gamma, n\rangle \triangleright N\vert M$	$\equiv_f$	$\langle\Gamma, n\rangle \triangleright M\vert N$	
(bs-assoc)	$\langle\Gamma, n\rangle \triangleright (N\vert M)\vert M'$	$\equiv_f$	$\langle\Gamma, n\rangle \triangleright N\vert(M\vert M')$	
(bs-unit)	$\langle\Gamma, n\rangle \triangleright N\vert l[\![\mathbf{0}]\!]$	$\equiv_f$	$\langle\Gamma, n\rangle \triangleright N$	
(bs-extr)	$\langle\Gamma, n\rangle \triangleright (\nu\, m{:}\mathtt{T})(N\vert M)$	$\equiv_f$	$\langle\Gamma, n\rangle \triangleright N\vert(\nu\, m{:}\mathtt{T})M$	$m \notin \mathbf{fn}(N)$
(bs-flip)	$\langle\Gamma, n\rangle \triangleright (\nu\, m_1{:}\mathtt{T})(\nu\, m_2{:}\mathtt{U})N$	$\equiv_f$	$\langle\Gamma, n\rangle \triangleright (\nu\, m_2{:}\mathtt{U})(\nu\, m_1{:}\mathtt{T})N$	
(bs-inact)	$\langle\Gamma, n\rangle \triangleright (\nu\, m{:}\mathtt{T})N$	$\equiv_f$	$\langle\Gamma, n\rangle \triangleright N$	$m \notin \mathbf{fn}(N)$
(bs-dead)	$\langle\Gamma, n\rangle \triangleright l[\![P]\!]$	$\equiv_f$	$\langle\Gamma, n\rangle \triangleright l[\![Q]\!]$	$\Gamma \nvdash l : \mathbf{alive}$

then the case is similar to that of (b-ngo).[3] Finally, if we cannot induce more failures, as is the case of (b-go-ff), then trivially we cannot interfere with migration between two live locations.

Similarly, pinging is generally not confluent because the location tested for may change its status and affect the branching of the ping's transition. However there are specific cases where it is a β-move, namely

- when the location tested is alive and no more failures can occur, (b-ping-ff).
- when the location tested is dead, (b-nping).
- when the location tested is public, and therefore immortal, (b-ping-pub).

In all three cases, the location tested for cannot change its status before or after the ping.

Even though a setting with failure requires us to analyse many more states than in a failure-free setting and limits the use of β-moves, we can exploit the permanent nature of the failure assumed in our model to define a stronger structural equivalence than the one defined earlier in Table 4. The new structural equivalence, denoted as $\equiv_f$, is strengthened by defining it over *extended configurations* instead of over systems. It is the least relation satisfying the rules in Table 12 and closed under the obvious generalisation of the operations of parallel composition and name restriction to extended configurations. Taking advantage of the *network status*, we can add a new structural rule, (bs-dead), which allows us to equate dead code, that is code residing at (permanently) dead locations.

Example 24 (Stronger Structural Equivalence) *Using $\equiv_f$, we can now equate the arbitrary systems $l[\![P]\!]$ and $k[\![Q]\!]$ running over the network $\langle\Gamma, n\rangle$ when both l and*

[3] The additional condition on the liveness of the destination location is extra but excludes cases when (b-ngo) can be applied instead.

k are dead in Γ. The derivation is as follows:

$$\begin{aligned}
\langle \Gamma, n\rangle \triangleright l[\![P]\!] &\equiv_f \langle \Gamma, n\rangle \triangleright l[\![\mathbf{0}]\!] && \textit{(bs-dead)}\\
&\equiv_f \langle \Gamma, n\rangle \triangleright l[\![\mathbf{0}]\!] \mid k[\![\mathbf{0}]\!] && \textit{(bs-unit)}\\
&\equiv_f \langle \Gamma, n\rangle \triangleright k[\![\mathbf{0}]\!] && \textit{(bs-unit)}\\
&\equiv_f \langle \Gamma, n\rangle \triangleright k[\![Q]\!] && \textit{(bs-dead)}
\end{aligned}$$

As with the standard structural equivalence, one can show that $\equiv_f$ is a strong bisimulation:

Lemma 25 ($\equiv_f$ is a strong bisimulation)

$$\begin{tikzcd}
\langle \Gamma, n\rangle \triangleright N \arrow[d, "\mu"'] \arrow[r, phantom, "\equiv_f"] & \langle \Gamma, n\rangle \triangleright M \arrow[r, phantom, "\textit{implies}"] & \langle \Gamma, n\rangle \triangleright N \arrow[d, "\mu"'] \arrow[r, phantom, "\equiv_f"] & \langle \Gamma, n\rangle \triangleright M \arrow[d, "\mu"'] \\
\langle \Gamma', n'\rangle \triangleright N' & & \langle \Gamma', n'\rangle \triangleright N' \arrow[r, phantom, "\equiv_f"] & \langle \Gamma', n'\rangle \triangleright M'
\end{tikzcd}$$

PROOF. A long but straightforward induction on the proof of $\langle \Gamma, n\rangle \triangleright N \equiv_f \langle \Gamma, n\rangle \triangleright M$.

Lemma 26 (Commutativity of $\equiv_f$ and $\overset{\tau}{\longmapsto}_\beta$) $\equiv_f \circ \overset{\tau}{\longmapsto}_\beta$ *implies* $\overset{\tau}{\longmapsto}_\beta \circ \equiv_f$

PROOF. If $\langle \Gamma, n\rangle \triangleright N \equiv_f \circ \overset{\tau}{\longmapsto}_\beta \langle \Gamma, n\rangle \triangleright M$ then we know that there is some N' such that

$$\langle \Gamma, n\rangle \triangleright N \;\equiv_f\; \langle \Gamma, n\rangle \triangleright N' \tag{17}$$
$$\langle \Gamma, n\rangle \triangleright N' \overset{\tau}{\longmapsto}_\beta \langle \Gamma, n\rangle \triangleright M \tag{18}$$

Now, mimicking the proof strategy of the previous lemma, we can use induction on the proof of (17) to find a matching transition $\langle \Gamma, n\rangle \triangleright N \overset{\tau}{\longmapsto}_\beta \langle \Gamma, n\rangle \triangleright M'$ such that $\langle \Gamma, n\rangle \triangleright M' \equiv_f \langle \Gamma, n\rangle \triangleright M$. The existence of this M' ensures that $\langle \Gamma, n\rangle \triangleright N \overset{\tau}{\longmapsto}_\beta \circ \equiv_f \langle \Gamma, n\rangle \triangleright M$.

Lemma 27 (Confluence of β-moves) $\overset{\tau}{\longmapsto}_\beta$ *observes the following diamond property:*

$$\begin{tikzcd}
\langle \Gamma, n\rangle \triangleright N \arrow[r, mapsto, "\tau", "\beta"'] \arrow[d, "\mu"'] & \langle \Gamma, n\rangle \triangleright M \\
\langle \Gamma', n'\rangle \triangleright N' &
\end{tikzcd}$$

implies either μ is τ and $\langle\Gamma, n\rangle \triangleright M = \langle\Gamma', n'\rangle \triangleright N'$ or else

$$\begin{tikzcd}
\langle\Gamma, n\rangle \triangleright N \arrow[r, mapsto, "\tau", "\beta"'] \arrow[d, "\mu"'] & \langle\Gamma, n\rangle \triangleright M \arrow[d, "\mu"'] \\
\langle\Gamma', n'\rangle \triangleright N' \arrow[r, phantom, "\mathcal{R}"] & \langle\Gamma', n'\rangle \triangleright M'
\end{tikzcd}$$

where $\mathcal{R}$ is the relation $\left(\overset{\tau}{\longmapsto}_\beta \cup \equiv_f \cup \overset{\tau}{\longmapsto}_\beta \circ \equiv_f\right)$

PROOF. The proof proceeds by induction on the structure of N and then by case analysis of the different types of μ and induction on the derivation of the β-move.

As examples, we consider two of the more interesting cases. First we consider the case where the relation $\mathcal{R}$ required to complete the confluence diamond is a case of $\equiv_f$. The second case is an instance where $\mathcal{R}$ is $\overset{\tau}{\longmapsto}_\beta \circ \equiv_f$.

(1) Consider the case where

$$N = l[\![\mathsf{if}\ n\!=\!n\ \mathsf{then}\ Q_1\ \mathsf{else}\ Q_2]\!] \text{ for some } n,\ Q_1,\ Q_2 \tag{19}$$
$$\mu = \tau \text{ and } n' = n - 1 \quad \text{(a confined location was killed)} \tag{20}$$

By case analysis and (19), we know that the β-move is the local reduction (b-eq) and thus

$$M = l[\![Q_1]\!] \tag{21}$$

From (20) we know the last rule used to derive the other action is (l-fail) and thus, using (19), we also derive

$$N' = l[\![\mathsf{if}\ n\!=\!n\ \mathsf{then}\ Q_1\ \mathsf{else}\ Q_2]\!]$$
$$\Gamma' = \Gamma - k' \text{ for some confined location } k' \text{ where } \Gamma \vdash k' : \textbf{alive}$$

We focus on the case where $k' = l$ and leave the case when $k' \neq l$ to the interested reader. On one side, using (21) we can produce

$$\langle\Gamma, n\rangle \triangleright l[\![Q_1]\!] \xrightarrow{\tau} \langle\Gamma - l, n - 1\rangle \triangleright l[\![Q_1]\!]$$

But on the other side we cannot produce a matching β-move because l, the location the name matching β-move is performed, is dead in $\Gamma - l$. However, the two reducts differ only with respect to dead code and we can use $\mathcal{R} =\equiv_f$ and (bs-dead) to get

$$\langle\Gamma - l, n - 1\rangle \triangleright l[\![\mathsf{if}\ n\!=\!n\ \mathsf{then}\ Q_1\ \mathsf{else}\ Q_2]\!] \ \equiv_f\ \langle\Gamma - l, n - 1\rangle \triangleright l[\![Q_1]\!]$$

(2) Consider a second case where

$$N = l[\![\mathsf{go}\ k.P]\!] \tag{22}$$
$$\mu = \tau \text{ and } n' = n - 1 \text{ (a confined location was dynamically killed)} \tag{23}$$

and moreover

l is public (immortal) in Γ (24)
k is confined (unreliable) but still alive in Γ (25)

Using (22), (24), (25) and case analysis we know that the β-move was derived using (b-go-pub), and thus we obtain

$$M = k[\![P]\!] \tag{26}$$

From (23) we know the last rule used to derive the other action is (l-fail) and thus, using (22), we also obtain

$N' = l[\![\mathsf{go}\ k.P]\!]$
$\Gamma' = \Gamma - k'$ for some confined location k' where $\Gamma \vdash k' : \textbf{alive}$

We focus on the case where $k' = k$ and leave the case when $k' \neq k$ to the interested reader. On one side, using (26) we can produce

$$\langle\Gamma, n\rangle \triangleright k[\![P]\!] \xrightarrow{\tau} \langle\Gamma - k, n-1\rangle \triangleright k[\![P]\!]$$

But on the other side we cannot produce a matching β-move using the same β-rule (b-go-pub) because k, the destination of the migration, is dead in $\Gamma - k$. Instead, we can use an alternative β-move, this time using (b-ngo) to obtain

$$\langle\Gamma - k, n-1\rangle \triangleright l[\![\mathsf{go}\ k.P]\!] \overset{\tau}{\longmapsto}_\beta \langle\Gamma - k, n-1\rangle \triangleright k[\![\mathbf{0}]\!]$$

and use the case when $\mathcal{R} = (\overset{\tau}{\longmapsto}_\beta \circ \equiv_f)$ to relate the two reducts, which differ only with respect to dead code. More precisely, we use (bs-dead) once again to get

$$\langle\Gamma - k, n-1\rangle \triangleright k[\![P]\!] \equiv_f \langle\Gamma - k, n-1\rangle \triangleright k[\![\mathbf{0}]\!]$$

Example 28 *Here we illustrate the fact that not all distributed migrations are confluent. Consider the configuration*

$$\langle\Gamma, n\rangle \triangleright k[\![\mathit{go}\ l.a!\langle\rangle]\!]$$

where

- *we can induce more dynamic failures, $n \geq 1$*
- *k, the source location of the migrating code, is alive but unreliable, $\Gamma \vdash k : \mathtt{loc}^{\mathtt{a}}_{\mathtt{c}}$*
- *l, the destination location is alive, $\Gamma \vdash l : \textbf{alive}$.*

Here, contrary to [3], the silent migration transition,

$$\Gamma \triangleright k[\![\mathit{go}\ l.a!\langle\rangle]\!] \xrightarrow{\tau} \Gamma \triangleright l[\![a!\langle\rangle]\!]$$

can not *be a β-move, even if we abstract over dead code. The problem occurs when we consider the transition killing k, and obtaining*

$$\Gamma \triangleright k[\![go\ l.a!\langle\rangle]\!] \xrightarrow{\tau} \langle \Gamma - k, n - 1\rangle \triangleright k[\![go\ l.a!\langle\rangle]\!]$$

Here we can never complete the diamond diagram for these two transitions, as required in Lemma 27.

Lemma 29 (Confluence over Weak moves)

$$\begin{tikzcd}
\langle\Gamma, n\rangle \triangleright N \arrow[r, mapsto, "\tau", "\beta\,*"'] \arrow[d, Rightarrow, "\hat{\mu}"'] & \langle\Gamma, n\rangle \triangleright M & \text{implies} & \langle\Gamma, n\rangle \triangleright N \arrow[rr, mapsto, "\tau", "\beta\,*"'] \arrow[d, Rightarrow, "\hat{\mu}"'] & & \langle\Gamma, n\rangle \triangleright M \arrow[d, Rightarrow, "\hat{\mu}"'] \\
\langle\Gamma', n'\rangle \triangleright N' & & & \langle\Gamma', n'\rangle \triangleright N' \arrow[r, mapsto, "\tau", "\beta\,*"'] & \circ \equiv_f & \langle\Gamma', n'\rangle \triangleright M'
\end{tikzcd}$$

where the length of the derivation of $\langle\Gamma, n\rangle \triangleright N \overset{\hat{\mu}}{\Longrightarrow} \langle\Gamma', n'\rangle \triangleright N'$ *is of the* at most *that of* $\langle\Gamma, n\rangle \triangleright M \overset{\hat{\mu}}{\Longrightarrow} \langle\Gamma', n'\rangle \triangleright M'$.

PROOF. The proof is by induction on the length of derivation, using Lemma 27, Corollary 26 and Lemma 25.

Proposition 30 *Suppose* $\langle\Gamma, n\rangle \triangleright N \overset{\tau}{\longmapsto}{}^{*}_{\beta} \langle\Gamma, n\rangle \triangleright M$. *Then* $\langle\Gamma, n\rangle \triangleright N \approx \langle\Gamma, n\rangle \triangleright M$.

PROOF. We define $\mathcal{R}$ as

$$\mathcal{R} = \left\{ \begin{array}{l} \langle\Gamma, n\rangle \triangleright N\ , \langle\Gamma, m\rangle \triangleright M\ \ |\langle\Gamma, n\rangle \triangleright N \overset{\tau}{\longmapsto}_{\beta} \langle\Gamma, m\rangle \triangleright M \\ \langle\Gamma, n\rangle \triangleright N\ , \langle\Gamma, m\rangle \triangleright M\ \ |\langle\Gamma, n\rangle \triangleright N \equiv_f \langle\Gamma, m\rangle \triangleright M \end{array} \right\}$$

Using Lemma 27 and Lemma 25 it is easy to show that $\mathcal{R}$ a bisimulation. Then by transitivity of $\approx$ we obtain the required result.

Definition 31 (Bisimulation up-to β-moves) *A relation $\mathcal{R}$ over configurations is called a* bisimulation up-to β-moves, *if whenever* $\langle\Gamma_N, n\rangle \triangleright N\ \ \mathcal{R}\ \ \langle\Gamma_M, m\rangle \triangleright M$ *then*

- $\langle\Gamma_N, n\rangle \triangleright N \xrightarrow{\mu} \langle\Gamma'_N, n'\rangle \triangleright N'$ *implies* $\langle\Gamma_M, m\rangle \triangleright M \overset{\hat{\mu}}{\Longrightarrow} \langle\Gamma'_M, m'\rangle \triangleright M'$ *such that* $\langle\Gamma'_N, n'\rangle \triangleright N' \overset{\tau}{\longmapsto}{}^{*}_{\beta} \circ \equiv_f \circ \mathcal{R} \circ \approx \langle\Gamma'_M, m'\rangle \triangleright M'$
- $\langle\Gamma_M, m\rangle \triangleright M \xrightarrow{\mu} \langle\Gamma'_M, m'\rangle \triangleright M'$ *implies* $\langle\Gamma, n\rangle \triangleright N \overset{\hat{\mu}}{\Longrightarrow} \langle\Gamma'_N, n'\rangle \triangleright N'$ *such that* $\langle\Gamma'_M, m'\rangle \triangleright M' \overset{\tau}{\longmapsto}{}^{*}_{\beta} \circ \equiv_f \circ \mathcal{R} \circ \approx \langle\Gamma'_N, n'\rangle \triangleright N'$

We use $\approx_\beta$ to denote the largest such relation.

Definition 31 provides us with a powerful method for approximating bisimulations. In a bisimulation up-to-β-moves an action $\langle \Gamma_N, n\rangle \triangleright N \xrightarrow{\mu} \langle \Gamma'_N, n'\rangle \triangleright N'$ can be matched by a weak matching action $\langle \Gamma_M, m\rangle \triangleright M \overset{\hat{\mu}}{\Longrightarrow} \langle \Gamma'_M, m'\rangle \triangleright M'$ such that up-to β-derivatives of $\langle \Gamma'_N, n'\rangle \triangleright N'$ modulo structural equivalence on the one side, and up-to bisimilarity on the other side, the pairs $\langle \Gamma'_N, n'\rangle \triangleright N'$ and $\langle \Gamma'_M, m'\rangle \triangleright M'$ are once more related. Intuitively then, in such a relation a configuration can represent all the configurations to which it can evolve using β-moves. in order to justify the use of these approximate bisimulations we need the following result:

Lemma 32 *Suppose* $\langle \Gamma_1, n_1\rangle \triangleright M_1 \approx_\beta \langle \Gamma_2, n_2\rangle \triangleright M_2$ *and* $\langle \Gamma_1, n_1\rangle \triangleright M_1 \overset{\widehat{\mu}}{\Longrightarrow} \langle \Gamma'_1, n'_1\rangle \triangleright M'_1$. *Then* $\langle \Gamma_2, n_2\rangle \triangleright M_2 \overset{\widehat{\mu}}{\Longrightarrow} \langle \Gamma'_2, n'_2\rangle \triangleright M'_2$, *where* $\langle \Gamma'_1, n'_1\rangle \triangleright M'_1 \approx \circ \approx_\beta \circ \approx \langle \Gamma'_2, n'_2\rangle \triangleright M'_2$.

PROOF. We proceed by induction on the length of $\langle \Gamma_1, n_1\rangle \triangleright M_1 \overset{\widehat{\mu}}{\Longrightarrow} \langle \Gamma'_1, n'_1\rangle \triangleright M'_1$. The base case, when the length is zero and $\Gamma_1 \triangleright M_1 = \Gamma'_1 \triangleright M'_1$ is trivial. There are two inductive cases. Here we focus on one case where

$$\langle \Gamma_1, n_1\rangle \triangleright M_1 \xrightarrow{\tau} \langle \Gamma^1_1, n^1_1\rangle \triangleright M^1_1 \overset{\widehat{\mu}}{\Longrightarrow} \langle \Gamma'_1, n'_1\rangle \triangleright M'_1 \tag{27}$$

and leave the other (similar) case for the interested reader.

By the definition of $\approx_\beta$, Definition 31, there exists $\langle \Gamma^1_2, n^1_2\rangle \triangleright M^1_2$ such that $\langle \Gamma_2, n_2\rangle \triangleright M_2 \xrightarrow{\tau}{}^* \langle \Gamma^1_2, n^1_2\rangle \triangleright M^1_2$ and

$$\langle \Gamma^1_1, n^1_1\rangle \triangleright M^1_1 \overset{\tau}{\longmapsto}{}^*_\beta \equiv \circ \approx_\beta \circ \approx \langle \Gamma^1_2, n^1_2\rangle \triangleright M^1_2 \tag{28}$$

By (27) and the expansion of (28) we have the following diagram to complete, for some Γ^1_1, Γ^2_2, M^2_1, M^2_2.

$$\begin{tikzcd}
\langle \Gamma^1_1, n^1_1\rangle \triangleright M^1_1 \arrow[r, mapsto, "\tau\ *"] \arrow[d, Rightarrow, "\widehat{\mu}"'] & \circ \equiv_f \langle \Gamma^1_1, n^1_1\rangle \triangleright M^2_1 \approx_\beta & \langle \Gamma^2_2, n^2_2\rangle \triangleright M^2_2 & \approx & \langle \Gamma^1_2, n^1_2\rangle \triangleright M^1_2 \\
\langle \Gamma'_1, n'_1\rangle \triangleright M'_1 & & & &
\end{tikzcd}$$

We immediately fill the first part of the diagram, using Lemma 29 and Lemma 25, to get the following:

$$\begin{tikzcd}
\langle \Gamma^1_1, n^1_1\rangle \triangleright M^1_1 \arrow[r, mapsto, "\tau\ *", "\beta"'] \arrow[d, Rightarrow, "\widehat{\mu}"'] & \circ \equiv_f \langle \Gamma^1_1, n^1_1\rangle \triangleright M^2_1 \approx_\beta \arrow[d, Rightarrow, "\widehat{\mu}"'] & \langle \Gamma^2_2, n^2_2\rangle \triangleright M^2_2 & \approx & \langle \Gamma^1_2, n^1_2\rangle \triangleright M^1_2 \\
\langle \Gamma'_1, n'_1\rangle \triangleright M'_1 \arrow[r, mapsto, "\tau\ *", "\beta"'] & \circ \equiv_f \langle \Gamma^1_1, n^1_1\rangle \triangleright M^3_1 & & &
\end{tikzcd}$$

By our inductive hypothesis we fill in the third part where $\mathcal{R} = \approx \circ \approx_\beta \circ \approx$

$$\begin{tikzcd}
\langle \Gamma_1^1, n_1^1 \rangle \triangleright M_1^1 \arrow[r, phantom, "\overset{\tau}{\longmapsto}{}^{*}_{\beta} \circ \equiv_f"] \arrow[d, Rightarrow, "\widehat{\mu}"'] & \langle \Gamma_1^1, n_1^1 \rangle \triangleright M_1^2 \arrow[r, phantom, "\approx_\beta"] \arrow[d, Rightarrow, "\widehat{\mu}"'] & \langle \Gamma_2^2, n_2^2 \rangle \triangleright M_2^2 \arrow[r, phantom, "\approx"] \arrow[d, Rightarrow, "\widehat{\mu}"'] & \langle \Gamma_2^1, n_2^1 \rangle \triangleright M_2^1 \\
\langle \Gamma_1', n_1' \rangle \triangleright M_1' \arrow[r, phantom, "\overset{\tau}{\longmapsto}{}^{*}_{\beta} \circ \equiv_f"] & \langle \Gamma_1^1, n_1^1 \rangle \triangleright M_1^3 \arrow[r, phantom, "\mathcal{R}"] & \langle \Gamma_2^3, n_2^3 \rangle \triangleright M_2^3 &
\end{tikzcd}$$

And finally we complete the diagram by the definition of $\approx$

$$\begin{tikzcd}
\langle \Gamma_1^1, n_1^1 \rangle \triangleright M_1^1 \arrow[r, phantom, "\overset{\tau}{\longmapsto}{}^{*}_{\beta} \circ \equiv_f"] \arrow[d, Rightarrow, "\widehat{\mu}"'] & \langle \Gamma_1^1, n_1^1 \rangle \triangleright M_1^2 \arrow[r, phantom, "\approx_\beta"] \arrow[d, Rightarrow, "\widehat{\mu}"'] & \langle \Gamma_2^2, n_2^2 \rangle \triangleright M_2^2 \arrow[r, phantom, "\approx"] \arrow[d, Rightarrow, "\widehat{\mu}"'] & \langle \Gamma_2^1, n_2^1 \rangle \triangleright M_2^1 \arrow[d, Rightarrow, "\widehat{\mu}"'] \\
\langle \Gamma_1', n_1' \rangle \triangleright M_1' \arrow[r, phantom, "\overset{\tau}{\longmapsto}{}^{*}_{\beta} \circ \equiv_f"] & \langle \Gamma_1^1, n_1^1 \rangle \triangleright M_1^3 \arrow[r, phantom, "\mathcal{R}"] & \langle \Gamma_2^3, n_2^3 \rangle \triangleright M_2^3 \arrow[r, phantom, "\approx"] & \langle \Gamma_2', n_2' \rangle \triangleright M_2'
\end{tikzcd}$$

The required result follows from the above completed diagram and the fact that $\overset{\tau}{\longmapsto}{}^{*}_{\beta} \subseteq \approx$, from Proposition 30, and $\equiv_f \subseteq \sim \subseteq \approx$, from Lemma 25.

Proposition 33 (Soundness of bisimulations up-to-β-moves)

$$\langle \Gamma, n \rangle \triangleright N \approx_\beta \langle \Gamma', m \rangle \triangleright M \;\; \textit{implies} \;\; \langle \Gamma, n \rangle \triangleright N \approx \langle \Gamma', m \rangle \triangleright M$$

PROOF. We prove the proposition by defining the relation $\mathcal{R}$ as

$$\mathcal{R} = \left\{ \langle \Gamma, n \rangle \triangleright N \;, \langle \Gamma', m \rangle \triangleright M \quad \middle| \langle \Gamma, n \rangle \triangleright N \approx \circ \approx_\beta \circ \approx \langle \Gamma', m \rangle \triangleright M \right\}$$

and showing that it is a bisimulation. The result then follows, since $\approx \subseteq \mathcal{R}$.

Assume $\langle \Gamma_1, n_1 \rangle \triangleright M_1 \xrightarrow{\mu} \langle \Gamma_1', n_1' \rangle \triangleright M_1'$. By our definition of $\mathcal{R}$, there exists $\langle \Gamma_1^1, n_1^1 \rangle \triangleright M_1^1$ and $\langle \Gamma_2', n_2' \rangle \triangleright M_2'$ such that

$$\langle \Gamma_1, n \rangle \triangleright M_1 \approx \langle \Gamma_1^1, n_1^1 \rangle \triangleright M_1^1 \tag{29}$$
$$\langle \Gamma_1^1, n_1^1 \rangle \triangleright M_1^1 \approx_\beta \langle \Gamma_2', n_2' \rangle \triangleright M_2' \tag{30}$$
$$\langle \Gamma_2', n_2' \rangle \triangleright M_2' \approx \langle \Gamma_2, n_2 \rangle \triangleright M_2 \tag{31}$$

From (29) and the definition of bisimulation we know

$$\langle \Gamma_1^1, n_1^1 \rangle \triangleright M_1^1 \overset{\widehat{\mu}}{\Longrightarrow} \langle \Gamma_1^2, n_1^2 \rangle \triangleright M_1^2 \quad \text{such that } \langle \Gamma_1', n_1' \rangle \triangleright M_1' \approx \langle \Gamma_1^2, n_1^2 \rangle \triangleright M_1^2 \tag{32}$$

By Lemma 32, (32), and (30) we know

$$\langle\Gamma_2', n_2'\rangle \triangleright M_2' \overset{\widehat{\mu}}{\Longrightarrow} \langle\Gamma_2^1, n_2^1\rangle \triangleright M_2^1 \text{ such that}$$
$$\langle\Gamma_1^2, n_1^2\rangle \triangleright M_1^2 \approx \circ \approx_\beta \circ \approx \langle\Gamma_2^1, n_2^1\rangle \triangleright M_2^1 \tag{33}$$

and by (33) and (31) we also conclude

$$\langle\Gamma_2, n_2\rangle \triangleright M_2 \overset{\widehat{\mu}}{\Longrightarrow} \langle\Gamma_2^2, n_2^2\rangle \triangleright M_2^2 \text{ such that } \langle\Gamma_2^1, n_2^1\rangle \triangleright M_2^1 \approx \langle\Gamma_2^2, n_2^2\rangle \triangleright M_2^2 \tag{34}$$

which is our matching move, where $\langle\Gamma_1', n_1'\rangle \triangleright M_1' \ \mathcal{R}\ \langle\Gamma_2^2, n_2^2\rangle \triangleright M_2^2$ by (32), (33), (34) and the transitivity of $\approx$.

As a result of Theorem 23, in order to prove that server_2 is dynamically 1-fault tolerant, we can use Definition 21 and give a *single* witness bisimulation relation satisfying

$$\langle\Gamma_e, 0\rangle \triangleright \mathsf{server}_2 \approx \langle\Gamma_e, 1\rangle \triangleright \mathsf{server}_2,$$

as opposed to three separate relations otherwise required by Definition 16. But now, because of Proposition 33 we can go one step further and limit ourselves to a single witness bisimulation up-to β-moves. This approach is taken in the following, final, example.

Example 34 (Proving Dynamic Fault Tolerance) *Consider the relation $\mathcal{R}$ over extended configurations, defined by*

$$\mathcal{R} \stackrel{\text{def}}{=} \{\langle\Gamma \triangleright \mathit{server}_2,\ \Gamma \triangleright \mathit{server}_2\rangle\} \cup \left(\bigcup_{m,m' \in \textsc{Names}} \mathcal{R}'(m, m') \right)$$

$\mathcal{R}$ is the union of all the relations $\mathcal{R}'(m, m')$ where we substitute the variables x, y in $\mathcal{R}'(x, y)$ by names $m, m' \in$ $\textsc{Names}$. For clarity, the presentation of $\mathcal{R}'(x, y)$:

- *omits type information associated with scoped names data and sync*
- *uses the shorthand ñ for data, sync and Γ for Γ_e defined earlier; Γ is also used as an abbreviation for $\langle\Gamma, n\rangle$ whenever $n = 0$*
- *uses the following process definitions from Example 20:*

$$S(y) \Leftarrow \mathit{sync}?(x).y!\langle x\rangle$$
$$R_i \Leftarrow k_i[\![\mathit{data}?(x, y, z).\mathit{go}\ z\ .y!\langle f(x)\rangle]\!]$$

The mapping of the intermediary states in $\mathcal{R}'(x, y)$ is based on the separation of the sub-systems making up server_2 (and its derivatives) into two classes, based on their dependencies on the unreliable locations k_1, k_2 and k_3:

Independent: *sub-systems whose behaviour is not affected by the state of k_i for $i =$ 1..3. An example of such code is the located process $l[\![\mathit{sync}?(x).y!\langle x\rangle]\!]$, denoted as $l[\![S(y)]\!]$ above.*

Dependent: *sub-systems whose behaviour depends on the state of k_i for $i = 1..3$. Examples of such sub-systems are*

- *located processes that intend to* go to *k_i, such as the queries sent to the database replica* $\mathsf{go}\, k_i.d!\langle x, y, l\rangle$.
- *processes that* reside at *k_i, such as the database replicas themselves, denoted as R_i above and its derivative $k_i[\![\mathsf{go}\, l\, .y!\langle f(x)\rangle]\!]$.*
- *located processes that have* migrated from *k_i, such as replies from these replicas, $l[\![y!\langle f(x)\rangle]\!]$.*

In server_2 there are sub-systems dependent on k_1 and k_2 but not on k_3. To relate sub-systems dependent on unreliable locations, $\mathcal{R}'(x, y)$ uses three asymmetric relations ranging over systems:

- *$\mathcal{R}_i^{Id}(x, y)$ is a (quasi) identity relation; when we define the actual relation we explain why it is not exactly the identity.*
- *$\mathcal{R}_i^{\mathbf{0}}(x, y)$ maps left systems depending on k_i to the null process at k_i on the right, $k_i[\![\mathbf{0}]\!]$. We use this mapping when k_i is dead, exploiting the structural equivalence rule* (bs-dead)*.*
- *$\mathcal{R}_i^{\geq}(x, y)$, maps left systems depending on k_i to the null process at l on the right, $l[\![\mathbf{0}]\!]$. We use this mapping when in order to reach a bisimilar state, the replica at k_i must have been successfully queried and the answer must have been successfully returned and consumed by $l[\![\mathsf{S}(y)]\!]$.*

In $\mathcal{R}'(x, y)$, the dependent sub-systems are related with these three sub-relations depending on two factors: (1) the state of the respective k_i they depend on (2) whether the global system is in a position to output an answer back to the observer. More specifically:

(1) As long as k_i is alive in the right configuration, then the sub-systems depending on k_i is related to its corresponding sub-system in the left hand configuration using $\mathcal{R}_i^{Id}(x, y)$ for $i = 1..2$.

(2) When k_i for $i = 1..2$ dies, then we refer to the second criteria, that is whether the global system is ready or not to return an answer back to the observer, derived from the fact that $l[\![\mathsf{S}(y)]\!]$ has not yet reduced:

   (a) If the global system is not *ready to output an answer back to the observer, then we relate the sub-systems depending on k_i using $\mathcal{R}_1^{\mathbf{0}}(x, y)$. We note that here we make use of β-moves such as those using* (b-ngo) *and structural rules such as* (bs-dead) *from Table 11 and Table 12 respectively. To map dead code and code migrating to dead locations to $k_i[\![\mathbf{0}]\!]$. The other sub-system depending on the other unreliable location (k_j for $j \neq i$) is still related using $\mathcal{R}_i^{Id}(x, y)$.*

   (b) If the global system is ready to output an answer back to the observer ($l[\![y!\langle f(x)\rangle]\!]$), or has already done so, then we relate the sub-systems depending on the dead location k_i using $\mathcal{R}_1^{\mathbf{0}}(x, y)$. The difference from the previous case lies in the mapping used for the sub-systems depending on

k_j, the other unreliable location. Here we have two further sub-cases:

(i) If k_i died before *servicing the query, that is before returning an answer $\mathsf{sync}!\langle f(x)\rangle$ back to l, then the only way we can output an answer back to the observer is through the complete servicing of the other replica at k_j. Thus we map the sub-system depending on k_j in the left configuration to the corresponding sub-system on the right using using $\mathcal{R}^{\geq}_{2}(x,y)$.*

(ii) If k_i died after *servicing the query, we simply match the sub-systems depending on the other unreliable location k_j using the identity relation $\mathcal{R}^{Id}_{i}(x,y)$ as before.*

For clarity, the presentation of $\mathcal{R}'(x,y)$ is partitioned into three groups of clauses, each containing 4 clauses each.

- *The first group describes the cases where the configurations are* not *ready to output back an answer to the observer (case 2(a)).*
- *The second group describes the cases where the configurations are ready to output back an answer to the observer (case 2(b)).*
- *The third group describes the cases where the configurations have already outputted back an answer to the observer.*

We note that, in contrast to Example 20, the three sub-relations below abstract away from mapping the sub-system reduct

$$Q_i(x,y) \;\Leftarrow\; l[\![\mathsf{go}\, k_i.\mathsf{data}!\langle x,y,l\rangle]\!]$$

Since we are only required to give a bisimulation up-to β, the β-rule (b-go-pub) allows us to automatically abstract away from such an intermediary process in $\mathcal{R}'(x,y)$, since the migration source location l is public, thus immortal. Similarly, the two sub-relations $\mathcal{R}^{\mathbf{0}}_{i}(x,y)$ and $\mathcal{R}^{\geq}_{i}(x,y)$ abstract away from mapping tuples like

$$\langle k_i[\![\mathsf{go}\, l\,.y!\langle f(x)\rangle]\!], k_i[\![\mathbf{0}]\!]\rangle \text{ and } \langle k_i[\![\mathsf{go}\, l\,.y!\langle f(x)\rangle]\!], l[\![\mathbf{0}]\!]\rangle$$

respectively. Since the left hand configuration executes in a (dynamic) failure-free setting, we can apply the β-rule (b-go-ff) to abstract away over the intermediary process $k_i[\![\mathsf{go}\, l\,.y!\langle f(x)\rangle]\!]$. For the same reason, $\mathcal{R}^{Id}_{i}(x,y)$ is not exactly the identity relation; the β-rule (b-go-ff) allows us to abstract away from $k_i[\![\mathsf{go}\, l\,.y!\langle f(x)\rangle]\!]$ and instead we have the pair

$$\langle l[\![y!\langle f(x)\rangle]\!], k_i[\![\mathsf{go}\, l\,.y!\langle f(x)\rangle]\!]\rangle$$

We note however that for cases when the potential dynamic failure is also 0 in the right configuration (this happens after we induce one failure), then we can perform the same abstraction on the right hand side, omit the above pair and obtain an identity relation. More specifically, in $\mathcal{R}'(x,y)$, this happens for the 2nd, 3rd, 4th, 8th and 12th clauses.

$$
\mathcal{R}'(x,y) \stackrel{\text{def}}{=} \left\{ \begin{array}{lll}
\Gamma \triangleright (\nu\tilde{n})\begin{pmatrix} l[\![S(y)]\!] \\ \mid M_1 \mid M_2 \end{pmatrix} & , \langle \Gamma, 1\rangle \triangleright (\nu\tilde{n})\begin{pmatrix} l[\![S(y)]\!] \\ \mid N_1 \mid N_2 \end{pmatrix} & \left| \begin{array}{l} \langle M_1, N_1\rangle \in \mathcal{R}_1^{Id}(x,s) \\ \langle M_2, N_2\rangle \in \mathcal{R}_2^{Id}(x,s) \end{array} \right. \\
\Gamma \triangleright (\nu\tilde{n})\begin{pmatrix} l[\![S(y)]\!] \\ \mid M_1 \mid M_2 \end{pmatrix} & , \Gamma - k_1 \triangleright (\nu\tilde{n})\begin{pmatrix} l[\![S(y)]\!] \\ \mid N_1 \mid N_2 \end{pmatrix} & \left| \begin{array}{l} \langle M_1, N_1\rangle \in \mathcal{R}_1^{\geq}(x,s) \\ \langle M_2, N_2\rangle \in \mathcal{R}_2^{Id}(x,s) \end{array} \right. \\
\Gamma \triangleright (\nu\tilde{n})\begin{pmatrix} l[\![S(y)]\!] \\ \mid M_1 \mid M_2 \end{pmatrix} & , \Gamma - k_2 \triangleright (\nu\tilde{n})\begin{pmatrix} l[\![S(y)]\!] \\ \mid N_1 \mid N_2 \end{pmatrix} & \left| \begin{array}{l} \langle M_1, N_1\rangle \in \mathcal{R}_1^{Id}(x,s) \\ \langle M_2, N_2\rangle \in \mathcal{R}_2^{\geq}(x,s) \end{array} \right. \\
\Gamma \triangleright (\nu\tilde{n})\begin{pmatrix} l[\![S(y)]\!] \\ \mid M_1 \mid M_2 \end{pmatrix} & , \Gamma - k_3 \triangleright (\nu\tilde{n})\begin{pmatrix} l[\![S(y)]\!] \\ \mid N_1 \mid N_2 \end{pmatrix} & \left| \begin{array}{l} \langle M_1, N_1\rangle \in \mathcal{R}_1^{Id}(x,s) \\ \langle M_2, N_2\rangle \in \mathcal{R}_2^{Id}(x,s) \end{array} \right. \\
\\
\Gamma \triangleright (\nu\tilde{n})\begin{pmatrix} l[\![y!\langle f(x)\rangle]\!] \\ \mid M_1 \mid M_2 \end{pmatrix} & , \langle \Gamma, 1\rangle \triangleright (\nu\tilde{n})\begin{pmatrix} l[\![y!\langle f(x)\rangle]\!] \\ \mid N_1 \mid N_2 \end{pmatrix} & \left| \begin{array}{l} \langle M_1, N_1\rangle \in \mathcal{R}_1^{Id}(x,s) \\ \langle M_2, N_2\rangle \in \mathcal{R}_2^{Id}(x,s) \end{array} \right. \\
\Gamma \triangleright (\nu\tilde{n})\begin{pmatrix} l[\![y!\langle f(x)\rangle]\!] \\ \mid M_1 \mid M_2 \end{pmatrix} & , \Gamma - k_1 \triangleright (\nu\tilde{n})\begin{pmatrix} l[\![y!\langle f(x)\rangle]\!] \\ \mid N_1 \mid N_2 \end{pmatrix} & \left| \begin{array}{l} \langle M_1, N_1\rangle \in \mathcal{R}_1^{\mathbf{0}}(x,s) \\ \langle M_2, N_2\rangle \in \mathcal{R}_2^{\geq}(x,s) \end{array} \right. \\
\Gamma \triangleright (\nu\tilde{n})\begin{pmatrix} l[\![y!\langle f(x)\rangle]\!] \\ \mid M_1 \mid M_2 \end{pmatrix} & , \Gamma - k_2 \triangleright (\nu\tilde{n})\begin{pmatrix} l[\![y!\langle f(x)\rangle]\!] \\ \mid N_1 \mid N_2 \end{pmatrix} & \left| \begin{array}{l} \langle M_1, N_1\rangle \in \mathcal{R}_1^{\geq}(x,s) \\ \langle M_2, N_2\rangle \in \mathcal{R}_2^{\mathbf{0}}(x,s) \end{array} \right. \\
\Gamma \triangleright (\nu\tilde{n})\begin{pmatrix} l[\![y!\langle f(x)\rangle]\!] \\ \mid M_1 \mid M_2 \end{pmatrix} & , \Gamma - k_3 \triangleright (\nu\tilde{n})\begin{pmatrix} l[\![y!\langle f(x)\rangle]\!] \\ \mid N_1 \mid N_2 \end{pmatrix} & \left| \begin{array}{l} \langle M_1, N_1\rangle \in \mathcal{R}_1^{Id}(x,s) \\ \langle M_2, N_2\rangle \in \mathcal{R}_2^{Id}(x,s) \end{array} \right. \\
\\
\Gamma \triangleright (\nu\tilde{n})\Big(M_1 \mid M_2 \Big) & , \langle \Gamma, 1\rangle \triangleright (\nu\tilde{n})\Big(N_1 \mid N_2 \Big) & \left| \begin{array}{l} \langle M_1, N_1\rangle \in \mathcal{R}_1^{Id}(x,s) \\ \langle M_2, N_2\rangle \in \mathcal{R}_2^{Id}(x,s) \end{array} \right. \\
\Gamma \triangleright (\nu\tilde{n})\Big(M_1 \mid M_2 \Big) & , \Gamma - k_1 \triangleright (\nu\tilde{n})\Big(N_1 \mid N_2 \Big) & \left| \begin{array}{l} \langle M_1, N_1\rangle \in \mathcal{R}_1^{\mathbf{0}}(x,s) \\ \langle M_2, N_2\rangle \in \mathcal{R}_2^{\geq}(x,s) \end{array} \right. \\
\Gamma \triangleright (\nu\tilde{n})\Big(M_1 \mid M_2 \Big) & , \Gamma - k_2 \triangleright (\nu\tilde{n})\Big(N_1 \mid N_2 \Big) & \left| \begin{array}{l} \langle M_1, N_1\rangle \in \mathcal{R}_1^{\geq}(x,s) \\ \langle M_2, N_2\rangle \in \mathcal{R}_2^{\mathbf{0}}(x,s) \end{array} \right. \\
\Gamma \triangleright (\nu\tilde{n})\Big(M_1 \mid M_2 \Big) & , \Gamma - k_3 \triangleright (\nu\tilde{n})\Big(N_1 \mid N_2 \Big) & \left| \begin{array}{l} \langle M_1, N_1\rangle \in \mathcal{R}_1^{Id}(x,s) \\ \langle M_2, N_2\rangle \in \mathcal{R}_2^{Id}(x,s) \end{array} \right.
\end{array} \right\}
$$

$$\mathcal{R}^{\mathbf{0}}_i(x,y) \overset{\text{def}}{=} \left\{ \begin{pmatrix} k_i[\![d!\langle x,y,l\rangle]\!] \mid R_i & , k_i[\![\mathbf{0}]\!] \\ l[\![y!\langle f(x)\rangle]\!] & , k_i[\![\mathbf{0}]\!] \\ l[\![\mathbf{0}]\!] & , k_i[\![\mathbf{0}]\!] \end{pmatrix} \right\} \quad \mathcal{R}^{\geq}_i(x,y) \overset{\text{def}}{=} \left\{ \begin{pmatrix} k_i[\![d!\langle x,y,l\rangle]\!] \mid R_i & , l[\![\mathbf{0}]\!] \\ l[\![y!\langle f(x)\rangle]\!] & , l[\![\mathbf{0}]\!] \\ l[\![\mathbf{0}]\!] & , l[\![\mathbf{0}]\!] \end{pmatrix} \right\}$$

$$\mathcal{R}^{Id}_i(x,y) \overset{\text{def}}{=} \left\{ \begin{pmatrix} k_i[\![d!\langle x,y,l\rangle]\!] \mid R_i & , k_i[\![d!\langle x,y,l\rangle]\!] \mid R_i \\ l[\![y!\langle f(x)\rangle]\!] & , k_i[\![go\ l\,.y!\langle f(x)\rangle]\!] \\ l[\![y!\langle f(x)\rangle]\!] & , l[\![y!\langle f(x)\rangle]\!] \\ l[\![\mathbf{0}]\!] & , l[\![\mathbf{0}]\!] \end{pmatrix} \right\}$$

To elucidate the above presentation, we consider a number of possible transitions in $\mathcal{R}'(x,y)$ as an example. Assume we are one of the states described by the first clause in $\mathcal{R}'(x,y)$

$$\Gamma \triangleright (\nu\tilde{n})\begin{pmatrix} l[\![S(y)]\!] \\ \mid M_1 \mid M_2 \end{pmatrix}, \langle \Gamma, 1\rangle \triangleright (\nu\tilde{n})\begin{pmatrix} l[\![S(y)]\!] \\ \mid N_1 \mid N_2 \end{pmatrix} \quad \begin{array}{l} \textit{where} \\ \langle M_1, N_1\rangle \in \mathcal{R}^{Id}_1(x,s) \\ \langle M_2, N_2\rangle \in \mathcal{R}^{Id}_2(x,s) \end{array} \tag{35}$$

If in either configuration (left or right), we accept an answer from any replica and $l[\![S(y)]\!]$ goes to $l[\![y!\langle f(x)\rangle]\!]$, then we can match this with an identical transition and go to a state described by the 4th *clause*

$$\Gamma \triangleright (\nu\tilde{n})\begin{pmatrix} l[\![y!\langle f(x)\rangle]\!] \\ \mid M'_1 \mid M'_2 \end{pmatrix}, \langle \Gamma, 1\rangle \triangleright (\nu\tilde{n})\begin{pmatrix} l[\![y!\langle f(x)\rangle]\!] \\ \mid N'_1 \mid N'_2 \end{pmatrix} \quad \begin{array}{l} \textit{where} \\ \langle M'_1, N'_1\rangle \in \mathcal{R}^{Id}_1(x,s) \\ \langle M'_2, N'_2\rangle \in \mathcal{R}^{Id}_2(x,s) \end{array}$$

If on the other hand, from (35) *the right configuration performs a τ-move and injects a dynamic fault at k_1 (the case for k_2 is dual), we transition to a state described by the* 2nd *clause*

$$\Gamma \triangleright (\nu\tilde{n})\begin{pmatrix} l[\![S(y)]\!] \\ \mid M_1 \mid M_2 \end{pmatrix}, \Gamma - k_1 \triangleright (\nu\tilde{n})\begin{pmatrix} l[\![S(y)]\!] \\ \mid N_1 \mid N_2 \end{pmatrix} \quad \begin{array}{l} \textit{where} \\ \langle M_1, N_1\rangle \in \mathcal{R}^{\mathbf{0}}_1(x,s) \\ \langle M_2, N_2\rangle \in \mathcal{R}^{Id}_2(x,s) \end{array} \tag{36}$$

At this point, any actions by M_2 or N_2 are mapped by the identical action on the opposite side, while still remaining in a state described by the 2nd *clause of the relation. If however in* (36)*, M_1 is involved in an action, then we have two cases:*

- *If the action involving M_1 causes $l[\![S(y)]\!]$ to reduce to $l[\![y!\langle f(x)\rangle]\!]$ while reducing to M_1' itself, then we transition to a state described by the* 5^{th} *clause*

$$\Gamma \triangleright (\nu\tilde{n})\begin{pmatrix} l[\![y!\langle f(x)\rangle]\!] \\ \mid M_1' \mid M_2 \end{pmatrix}, \Gamma - k_1 \triangleright (\nu\tilde{n})\begin{pmatrix} l[\![y!\langle f(x)\rangle]\!] \\ \mid N_1 \mid N_2' \end{pmatrix} \quad \begin{array}{l} \textit{where} \\ \langle M_1', N_1\rangle \in \mathcal{R}_1^{\mathbf{0}}(x,s) \\ \langle M_2, N_2'\rangle \in \mathcal{R}_2^{\geq}(x,s) \end{array}$$

  where on the right hand side, N_2 has compensate for the inactive N_1 and match the move by weakly reducing to N_2', interacting with its respective $l[\![S(y)]\!]$ so that it reduces it to $l[\![y!\langle f(x)\rangle]\!]$. We highlight the fact that this internal interaction cannot be done by N_1 since k_1 is dead.
- *Otherwise, if $l[\![S(y)]\!]$ is not affected, we match the silent move from M_1 with the empty move on the right hand side.*

6 Conclusions and Related Work

This paper is a revised and extended version of the conference presentation [5]. We adopted a subset of [4] and developed a theory for system fault tolerance in the presence of fail-stop node failure. We formalised two definitions for fault tolerance based on the well studied concept of observational equivalence. The first definition assumes a static network state whereby the faults have already been induced; the second definition assumes that faults may be induced dynamically at any stage of the computation. Subsequently, we developed sound proof techniques with respect to these definitions which enable us to give tractable proofs to show the fault tolerance of systems; we gave two example proofs using these proof techniques.

Future Work The immediate next step is to apply the theory to a wider spectrum of examples, namely systems using replicas with state and system employing fault tolerance techniques such as lazy replication: we postulate that the existing theory should suffice. Other forms of fault contexts that embody different assumptions about failures, such as fault contexts with dependencies between faults, could be explored. Another avenue worth considering is extending the theory to deal with link failure and the interplay between node and link failure [4]. In the long run, we plan to develop of a compositional theory of fault tolerance, enabling the construction of fault tolerant systems from smaller component sub-systems. For all these cases, this work should provide a good starting point.

Related Work To the best of our knowledge, Prasad's thesis [14] is the closest work to ours, addressing fault tolerance for process calculi. Even though similar concepts such as redundancy (called ”duplication”) and failure-free execution are identified, the setting and development of Prasad differs considerably form ours. In essence, three new operators (”displace”, ”audit” and ”checkpoint”) are introduced in a variant of CCS; equational laws for terms using these operators are then developed so that algebraic manipulation can be used to show that terms in this calculus are, in some sense, fault tolerant with respect to their specification.

The use of confluence of certain τ-steps as a useful technique for the management of large bisimulations is not new. It has been already studied extensively in [13,7]. See [6] for particularly good examples of where they have significantly decreased the size of witness bisimulations. Elsewhere, Nestmann *et al.* [12] have explored various other ways of using bounds in the environment to govern permissible failures.

References

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

[2] Flavin Christian. Understanding fault tolerant distributed systems. *Communications of the ACM*, 34(2):56–78, February 1991.

[3] Alberto Ciaffaglione, Matthew Hennessy, and Julian Rathke. Proof methodologies for behavioural equivalence in Dπ. Technical Report 03/2005, University of Sussex, 2005.

[4] 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.

[5] 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*. Springer, 2006.

[6] J. F. Groote and M. P. A. Sellink. Confluence for process verification. *Theor. Comput. Sci.*, 170(1-2):47–81, 1996.

[7] Jan Friso Groote and Jaco van de Pol. State space reduction using partial tau-confluence. In *Mathematical Foundations of Computer Science*, pages 383–393, 2000.

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

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

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

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

[12] Uwe Nestmann and Rachele Fuzzati. Unreliable Failure Detectors via Operational Semantics. In *ASIAN '03*, Lecture Notes in Computer Science, pages 54–71, 2003.

[13] Anna Philippou and David Walker. On confluence in the pi-calculus. In *ICALP '97: Proceedings of the 24th International Colloquium on Automata, Languages and Programming*, pages 314–324, London, UK, 1997. Springer-Verlag.

[14] K. V. S. Prasad. *Combinators and Bisimulation Proofs for Restartable Systems*. PhD thesis, Department of Computer Science, University of Edinburgh, December 1987.

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

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

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

[18] Paulo Verissimo and Luis Rodrigues. *Distributed Systems for System Architects*. Kluwer Academic Publishers, 2001.