A Fault Tolerance Bisimulation Proof For Consensus (Extended Abstract)

Adrian Francalanza[1] and Matthew Hennessy[2]

[1] Imperial College, London SW7 2BZ, England, adrianf@doc.ic.ac.uk
[2] University of Sussex, Brighton BN1 9RH, England, matthewh@sussex.ac.uk

Abstract. The possibility of partial failure occuring at any stage of computation complicates rigorous formal treatment of distributed algorithms. We propose a methodology for formalising and proving the correctness of distributed algorithms which alleviates this complexity. The methodology uses fault-tolerance bisimulation proof techniques to split the analysis into two phases, that is a failure-free phase and a failure phase, permitting separation of concerns. We design a minimal partial-failure calculus, develop a corresponding bisimulation theory for it and express a consensus algorithm in the calculus. We then use the consensus example and the calculus theory to demonstrate the benefits of our methodology.

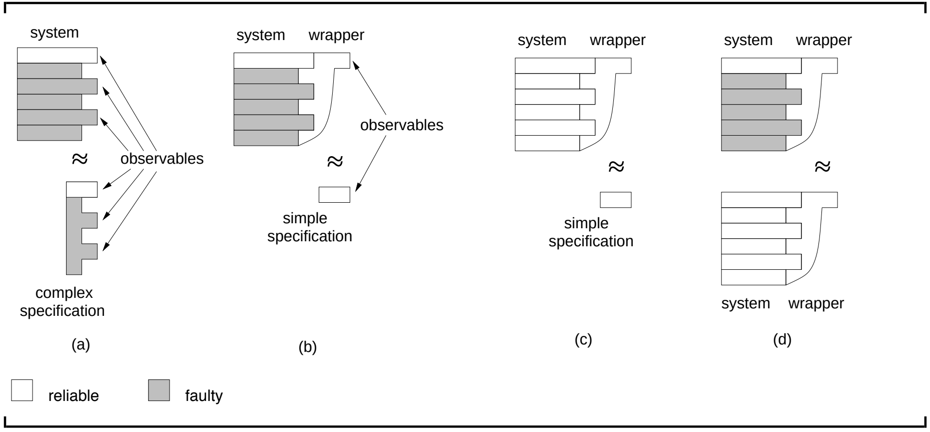
1 Introduction

The areas of Distributed Systems and Process Calculi are two (major) areas in Computer Science addressing the same problems but ”speak(ing) different languages” [14]. In particular, seminal work in Distributed Systems, such as [2, 11] present algorithms in semi-formal pseudo-code and correctness proofs of an informal algorithmic nature. The understandable reluctance to apply the rigorous theory of process calculi to formal proofs for standard distributed algorithms stems from the complexity and sheer size of the resulting formal descriptions and proofs. This problem is accentuated when failures are considered, which typically occur at any point during computation and can potentially affect execution. More specifically, in a process calculus with formal semantics based on labelled transition systems (lts), and a related bisimulation equivalence $\approx$, correctness proofs compare the behaviour of the distributed algorithm, described in the base calculus, to a correctness specification, also defined in the base calculus, using $\approx$; see Table 1(a). The required witness bisimulation relations resulting from this general approach turn out to be substantial, even for the simplest of algorithms and specifications. Even worse, partial failure tends to obfuscate the simplicity of the correctness specification while enlarging the state space of the bisimulations.

To tame such complexity, attempts at formalising distributed algorithm proofs have made use of mechanised theorem provers [8] or translations into tailor-made abstract interpretations [14]. In spite of their effectiveness, such tools and techniques tend to obscure the natural structure of the proofs of correctness, because they either still produce monolithic proofs, which are hard to digest, or else depart from the source formal language in which the algorithm is expressed.

We propose a prescriptive methodology to formally prove correctness of distributed algorithms which fine tunes well-studied bisimulation techniques to a partial failure

Table 1. *Correctness proofs using fault-tolerant bisimulation techniques*



setting. The methodology is based on a common assumption that some processes are assumed to be reliable, thus immortal. Failure can affect behaviour in two ways: either *directly*, when the process itself fails, or *indirectly*, when a process depends on internal interaction with a secondary process which in turn fails. The methodology limits observations to reliable processes *only*, which are *only* indirectly affected by failure. Using wrapper code around the algorithm being analysed, we reformulate the equivalence described earlier into a comparison between the re-packaged algorithm and a *simpler* specification that does not include unreliable processes (Table 1(b)). The wrappers can be *dedicated*, testing for separate correctness criteria. We can therefore decompose generic catch-all specifications into separate simpler specifications, which are easier to formulate, understand and verify against the expected behaviour.

This reformulation carries more advantages than merely decomposing the specification and shifting the complexity of the equivalence from the specification side to the algorithm side in the form of wrappers. A specification that does not include unreliable processes permits *separation of concerns* through *fault-tolerance* bisimulation techniques [7]. These techniques allow us to decompose our reformulated equivalence statement into two sub-statements. In the first we temporarily ignore failures and test for *basic correctness*: we use ”standard” bisimulations to compare the specification with the behaviour of the repackaged algorithm in a *failure-free* setting (Table 1(c)). In the second sub-statement we test for fault-tolerance and *correctness preservation*: we compare the behaviour of the repackaged algorithm in the failure-free setting with the repackaged algorithm itself in the failure setting (Table 1(d)). We argue that the fault-tolerance reformulation is a natural way to tackle such a proof, dividing it into two sub-proofs, which can be checked independently. The reformulation however carries further advantages. For a start, the first equivalence is considerably easier to prove, and can be treated as a vetting test before attempting the more involving second proof. Moreover, when proving the second equivalence statement, which compares the same code but under different conditions, we can exploit the common structure on both sides of the equivalence to construct the required witness bisimulation.

Our proposed methodology goes one step further and uses (permanent) failure to reduce the size of witness bisimulations in two ways. First, we note that while permanent failure may induce abnormal behaviour in the live code, it also *eliminates* transitions from dead code. Thus, by developing appropriate abstractions to represent dead code, we can greatly reduce the size of witness bisimulations. Second, we note that distributed algorithms tolerate failure (and preserve the expected behaviour) through the use of *redundancy* which is usually introduced in the form of *symmetrical replicated code*. As a result, correctness bisimulations for such algorithms are characterised by a considerable number of transitions that are similar in structure. This, in turn, gives us scope for identifying a subset of these transitions which are *confluent* and developing up-to techniques that abstract over these confluent moves. The number of replication patterns are arguably bounded and are reused throughout a substantial number of fault-tolerant algorithms, which means that we expect these up-to techniques to be applicable, at least in part, to a range of fault-tolerant distributed algorithm. More importantly however, they identify the (non-confluent) transitions that really matter, making the bisimulation proofs easier to describe and understand.

The remaining text is structured as follows. In Section 2 we introduce our language, a *partial-failure calculus*. In Section 3 we express a consensus algorithm in our calculus, realising the long considered view of consensus as a fault-tolerance problem [4]; we also show how to express the correctness of the algorithm in terms of basic correctness and correctness preservation equivalences. In Section 4 we develop up-to techniques for our algorithm and in Section 5 we give its proof of correctness.

2 Language

Our *partial-failure* calculus is inspired by [15] and consists of processes from a subset of CCS[12], distributed across a number of failing locations. We assume a set $\textsc{Act}$ of communicating actions equipped with a bijective function $\bar{\cdot}$; for every name $a \in \textsc{Act}$ we have a complement $\bar{a} \in \textsc{Act}$; α ranges over strong actions, defined as $\textsc{Act} \cup \{\tau\}$, including the distinguished silent action τ. We also assume a distinct set Locs of locations l, k which includes the immortal location $\star$.

Processes, defined in Table 2, can be guarded by an action, composed using choice, composed in parallel or scoped. As in [15], only actions can be scoped (not locations). By contrast to [15], we here simplify the calculus and disallow process constants and replication (thus no recursion and infinite computation) and migration of processes (thus no change in failure dependencies). Another important departure from [15] is that instead of *ping* we use a guarding construct $\mathsf{susp}\ l.P$, already introduced in [5], which *tests* for the status of l and releases P once it (correctly) suspects that l is dead; the construct captures the intuition that failure detection is separate from the actual failure, and can be delayed. Systems, also defined in Table 2, are *located* processes composed in parallel with channel scoping. We view our calculus as a *partial-failure* calculus rather than a *distributed* calculus as it permits action synchronisations across locations. This implies a tighter synchronisation assumption between locations, which in our calculus merely embody *units of failure*. Nevertheless, distributed choices are still disallowed because their implementation is problematic in a dynamic partial-failure setting.

Table 2. *Syntax*

Processes								
$P,\ Q ::= \alpha.P$	*(guard)*	$\mid\ \mathbf{0}$	*(inert)*	$\mid\ P+Q$	*(choice)*	$\mid\ (\nu a)P$	*(scoping)*
$\mid\ P\vert Q$	*(fork)*	$\mid\ \mathsf{susp}\ k.P$	*(failure detector)*				
Systems								
$M,\ N ::= l[\![P]\!]$	*(located)*	$\mid\ N\vert M$	*(parallel)*	$\mid\ (\nu\, a)N$	*(scoping)*		

Table 3. *Reduction Rules*

Assuming $l \in \mathcal{L}$, $n \geq 0$

(Act)
$$\frac{}{\langle \mathcal{L}, n\rangle \triangleright l[\![\alpha.P]\!] \xrightarrow{\alpha} l[\![P]\!]}$$

(Susp)
$$\frac{}{\langle \mathcal{L}, n\rangle \triangleright l[\![\mathsf{susp}\ k.P]\!] \xrightarrow{\tau} l[\![P]\!]}\ k \notin \mathcal{L}$$

(Halt)
$$\frac{}{\langle \mathcal{L}, n{+}1\rangle \triangleright M \xrightarrow{\tau} \langle \mathcal{L}{-}l, n\rangle \triangleright M}$$

(Fork)
$$\frac{}{\langle \mathcal{L}, n\rangle \triangleright l[\![P|Q]\!] \quad\xrightarrow{\tau}\quad l[\![P]\!]|l[\![Q]\!]}$$

(New)
$$\frac{}{\langle \mathcal{L}, n\rangle \triangleright l[\![(\nu a)P]\!] \quad\xrightarrow{\tau}\quad (\nu a)l[\![P]\!]}$$

(Sum)
$$\frac{\langle \mathcal{L}, n\rangle \triangleright l[\![P_i]\!] \quad\xrightarrow{\alpha}\quad l[\![P]\!]}{\langle \mathcal{L}, n\rangle \triangleright l[\![\sum_{i\in I} P_i]\!] \quad\xrightarrow{\alpha}\quad l[\![P]\!]}$$

(Rest)
$$\frac{\langle \mathcal{L}, n\rangle \triangleright M \quad\xrightarrow{\alpha}\quad \langle \mathcal{L}', n'\rangle \triangleright M'}{\langle \mathcal{L}, n\rangle \triangleright (\nu a)M \quad\xrightarrow{\alpha}\quad \langle \mathcal{L}', n'\rangle \triangleright (\nu a)M'}\ \alpha \notin \{a, \bar{a}\}$$

(Par)
$$\frac{\langle \mathcal{L}, n\rangle \triangleright M \quad\xrightarrow{\alpha}\quad \langle \mathcal{L}', n'\rangle \triangleright M'}{\begin{array}{l}\langle \mathcal{L}, n\rangle \triangleright M|N \quad\xrightarrow{\alpha}\quad \langle \mathcal{L}', n'\rangle \triangleright M'|N \\ \langle \mathcal{L}, n\rangle \triangleright N|M \quad\xrightarrow{\alpha}\quad \langle \mathcal{L}', n'\rangle \triangleright N|M'\end{array}}$$

(Com)
$$\frac{\langle \mathcal{L}, n\rangle \triangleright M \xrightarrow{\alpha} M' \qquad \langle \mathcal{L}, n\rangle \triangleright N \xrightarrow{\bar{\alpha}} N'}{\begin{array}{c}\langle \mathcal{L}, n\rangle \triangleright M|N \quad\xrightarrow{\tau}\quad M'|N' \\ \langle \mathcal{L}, n\rangle \triangleright N|M \quad\xrightarrow{\tau}\quad N'|M'\end{array}}$$

Notation: We denote a series of parallel processes $P_1|\ldots|P_n$ as $\prod_{i\in I} P_i$ and a series of choices $P_1 + \ldots + P_n$ as $\sum_{i\in I} P_i$ for $I = \{1, \ldots, n\}$. We omit the final $\mathbf{0}$ term in processes, writing $a.\mathbf{0}$ as a. We also denote the located inactive process $l[\![\mathbf{0}]\!]$ as simply $\mathbf{0}$ and omit location information for processes located at the immortal location. Thus, at system level, we write $M \,|\, P \,|\, \mathbf{0}$ to denote $M \,|\, \star\, [\![P]\!] \,|\, l[\![\mathbf{0}]\!]$.

Operational Semantics: We define a *liveset*, $\mathcal{L}$, as a set of locations, $\{l_1, \ldots, l_n\}$ denoting the locations that are alive; we usually omit mention of the special location $\star$, which is assumed to be in every $\mathcal{L}$. A system M subject to a liveset, $\mathcal{L}$, and a bounded number of dynamic failures, n, is called a configuration, and is denoted as $\langle \mathcal{L}, n\rangle \triangleright M$. Intuitively it denotes a system M that is running on the network (state) $\mathcal{L}$ where at most n locations from $\mathcal{L}$ may fail. Transitions are defined between tuples of configurations as

$$\langle \mathcal{L}, n\rangle \triangleright M \quad\xrightarrow{\alpha}\quad \langle \mathcal{L}', n'\rangle \triangleright M' \tag{1}$$

by the rules in Table 3. To improve readability, we abbreviate (1) to $\langle \mathcal{L}, n\rangle \triangleright M \quad\xrightarrow{\alpha}\quad M'$ whenever the state of the network $\langle \mathcal{L}, n\rangle$ does not change in the residual configuration. The rules in Table 3 are standard located CCS rules, with the exception of (Susp) describing perfect failure detection [2], and (Halt) describing dynamic failure [7]. All rules assume $l \in \mathcal{L}$ and $n \geq 0$.

Example 1. In (2) below, the system $\star[\![a.P + \mathsf{susp}\ l.P]\!]$ is in some sense *fault tolerant* up to 1 failure occuring in $\mathcal{L}$. Although $a.P$ depends on the liveness of l to proceed as P, the summand $\mathsf{susp}\ l.P$ also produces the same continuation P whenever l is dead. In order to verify this we have three cases to consider: (a) if $l \notin \mathcal{L}$ then $\mathsf{susp}\ l.P$ will trigger and produce $\star[\![P]\!]$; (b) if $l \in \mathcal{L}$ and $n = 0$, then l can never die and $a.P$ will always synchronise with $l[\![\bar{a}]\!]$ and continue as $\star[\![P]\!]$; (c) if $l \in \mathcal{L}$ and $n \neq 0$ then if l dies before the synchronisation on a occurs, we have case (a), otherwise we have case (b).

$$\langle \mathcal{L}, n\rangle \triangleright (\nu a)\ \ l[\![\bar{a}]\!] \mid \star\ [\![a.P + \mathsf{susp}\ l.P]\!] \qquad (2)$$

The equivalence relation chosen for our partial-failure calculus is *(weak) bisimulation equivalence*, based on weak matching moves $\overset{\hat{\alpha}}{\Longrightarrow}$ denoting $\xrightarrow{\tau}^{*}\xrightarrow{\alpha}\xrightarrow{\tau}^{*}$ if $\alpha \in \{a, \bar{a}\}$ and $\xrightarrow{\tau}^{*}$ if $\alpha = \tau$.

Definition 1 (Weak bisimulation equivalence). *Denoted as $\approx$, is the largest relation over configurations such that if $\langle \mathcal{L}_1, n_1\rangle \triangleright M_1 \approx \langle \mathcal{L}_2, n_2\rangle \triangleright M_2$ then*

- $\langle \mathcal{L}_1, n_1\rangle \triangleright M_1 \xrightarrow{\alpha} \langle \mathcal{L}_1', n_1'\rangle \triangleright M_1'$ *implies* $\langle \mathcal{L}_2, n_2\rangle \triangleright M_2 \overset{\hat{\alpha}}{\Longrightarrow} \langle \mathcal{L}_2', n_2'\rangle \triangleright M_2'$ *such that* $\langle \mathcal{L}_1', n_1'\rangle \triangleright M_1' \approx \langle \mathcal{L}_2', n_2'\rangle \triangleright M_2'$
- $\langle \mathcal{L}_2, n_2\rangle \triangleright M_2 \xrightarrow{\alpha} \langle \mathcal{L}_2', n_2'\rangle \triangleright M_2'$ *implies* $\langle \mathcal{L}_1, n_1\rangle \triangleright M_1 \overset{\hat{\alpha}}{\Longrightarrow} \langle \mathcal{L}_1', n_1'\rangle \triangleright M_1'$ *such that* $\langle \mathcal{L}_1', n_1'\rangle \triangleright M_1' \approx \langle \mathcal{L}_2', n_2'\rangle \triangleright N_2'$

Assuming that $\mathbf{loc}(M)$ is a function returning the set of all location names used in M (together with $\star$), then system M is said to be executing in a *failure-free* setting if it is subject to the network $\langle \mathbf{loc}(M), 0\rangle$. Based on this intuition and our notion of equivalence, we can give a formal definition for fault-tolerant systems.

Definition 2 (Fault Tolerance). *A system M is fault tolerant up to n faults whenever*

$$\langle \mathbf{loc}(M), 0\rangle \triangleright M \quad \approx \quad \langle \mathbf{loc}(M), n\rangle \triangleright M$$

Our chosen definitions are not arbitrary. Definition 1 is sound with respect to a standard contextual equivalence called reduction barbed congruence [10]. Definition 2 is sound with respect to dynamic fault-tolerance up-to n faults defined in [7], using fault inducing contexts. The adaptation of these definitions to our calculus and the proof of the corresponding soundness statements will appear in the full version of the paper.

Example 2. Using Definitions 1 and 2, we can now show that (2) above is fault tolerant up to 1 fault by giving a witness bisimulation relation satisfying

$$\langle \{l\}, 0\rangle \triangleright (\nu a)\ l[\![\bar{a}]\!] \mid \star\ [\![a.P + \mathsf{susp}\ l.P]\!] \quad \approx \quad \langle \{l\}, 1\rangle \triangleright (\nu a)\ l[\![\bar{a}]\!] \mid \star\ [\![a.P + \mathsf{susp}\ l.P]\!]$$

3 Consensus

Despite its limitations (no infinite computation), our calculus is expressive enough to describe a number of (non-recursive) standard distributed algorithms in the presence

Table 4. *The Rotating Co-ordinator Algorithm for Participant i*

```
1    $x_i$ := input;
2    for $r$ := 1 to $n$ do {  if $r = i$ then broadcast $x_i$;
3                              if alive($p_r$) then $x_i$ := input_from_broadcast };
4    output $x_i$;
```

of dynamic failure. As an example we describe the rotating co-ordinator algorithm [16], solving a specific instance of consensus using *strong* failure detectors ($\mathcal{S}$ [2]); the pseudo-code description is reproduced in Table 4. The algorithm consists of n parallel, independently failing participants, ordered and named 1 to n, each *inputting* a value v from a set of values V and then *deciding* by outputting a value $v' \in V$. Each participant executes the code in Table 4, going through n rounds (the loop on lines 2 and 3) and changing the broadcasting co-ordinator to participant i for round $r = i$. The correctness criteria for *consensus* is defined by the following three conditions [11, pg. 101]:

Termination: All non-failing participants must eventually decide.
Agreement: No two participants decide on different values.
Validity: If all participants are given the same value $v \in V$ as input, then v is the only possible decision value[3].

To attain consensus with $n-1$ dynamic failures, the algorithm needs to be fault-tolerant with respect to two error conditions, namely *Decision Blocking* (when a participant may be waiting forever for a value to be broadcast from a crashed co-ordinator) and *Corrupted Broadcast* (when co-ordinator may broadcast its values to a *subset* of the participants before crashing). The code in Table 4 overcomes decision blocking by using a failure detector to determine the state of the co-ordinator (alive(p_r)) and overcomes the possibility of $(n-1)$ corrupted broadcasts by repeating the broadcast for n rounds.

We give a precise description of the rotating co-ordinator algorithm as the system C, given in Table 5. Without loss of generality, we assume that the decision set is simply $V = \{true, false\}$ and have n participants located at *independently failing* locations $l_1 \dots l_n$. The process $\mathsf{P}^x_{i,r}$, for $x \in \{true, false\}$, denotes the i^{th} participant, at round r, with current estimate x. It is defined in terms of two parallel processes, $\mathsf{B}^x_{i,r}$ for *broadcasting* the current value at round r, and $\mathsf{R}^x_{i,r}$ for *receiving* the new value at round r. As in Table 4, broadcast is only allowed if $i = r$ and otherwise it acts as the inert process. On the other hand, the receiver at round r awaits synchronisation on $\mathit{true}_{i,r}$ or $\mathit{false}_{i,r}$ and updates the estimate for round $(r+1)$ accordingly. At the same time, the receiver guards this distributed synchronisation with $\mathsf{susp}\ l_r.\mathsf{P}^x_{i,r+1}$ to prevent decision blocking in case l_r, the location of the participant currently broadcasting, fails. Estimates for round r can only come from the participant at l_r and thus all actions $\mathit{true}_{i,r}$ and $\mathit{false}_{i,r}$ are scoped in C. Every participant can be arbitrarily initialised as $\mathsf{P}^{true}_{i,1}$ or $\mathsf{P}^{false}_{i,1}$ through the free actions prop^{true}_i and prop^{false}_i respectively. Finally every participant decides at round $(n+1)$ to either report true, executing dec^{true}_i, or report false, executing dec^{false}_i.

We can also give a precise description of the consensus correctness requirements in our calculus. As stated in the Introduction, we repackage our algorithm as a fault-

[3] When $|V| = 2$ this implies a stronger notion of validity: any decision value for any participant is the initial value of some process.

Table 5. *Rotating Co-ordinator Algorithm in our Partial-Failure Calculus*

(Consensus)

$$\mathsf{C} \overset{\text{def}}{=} \left(\nu^{n}_{i,r=1}\, true_{i,r}, false_{i,r}\right) \prod_{i=1}^{n} l_i[\![prop_i^{true}.\mathsf{P}^{true}_{i,1} + prop_i^{false}.\mathsf{P}^{false}_{i,1}]\!]$$

(Participant)

$$\mathsf{P}^{x}_{i,r} \overset{\text{def}}{=} \mathsf{R}^{x}_{i,r} \mid \mathsf{B}^{x}_{i,r} \quad x \in \{true, false\},\ r \leq n$$

$$\mathsf{P}^{x}_{i,n+1} \overset{\text{def}}{=} \overline{dec_i^{x}} \quad x \in \{true, false\}$$

(Broadcast)

$$\mathsf{B}^{x}_{i,r} \overset{\text{def}}{=} \prod_{j=1}^{n} \overline{x_{j,r}} \quad x \in \{true, false\},\ r = i$$

$$\mathsf{B}^{x}_{i,r} \overset{\text{def}}{=} \mathbf{0} \quad x \in \{true, false\},\ r \neq i$$

(Recieve)

$$\mathsf{R}^{x}_{i,r} \overset{\text{def}}{=} true_{i,r}.\mathsf{P}^{true}_{i,r+1} + false_{i,r}.\mathsf{P}^{false}_{i,r+1} + \mathsf{susp}\, l_r.\mathsf{P}^{x}_{i,r+1}$$

tolerant system where any interactions with observers occur through wrapper code residing at the immortal location $\star$; this allows us to decompose our proof into the basic correctness and correctness preservation phases, as in Table 1(c) and (d).

Table 6 defines the *wrapper code* which, when put in parallel with C of Table 5, provides separate *testing scenarios* for the algorithm. We have two forms of initialization code: I^{gen} arbitrarily initialises every participant to either *true* or *false* after the action *start* whereas I^{true} and I^{false} initialise *all* participants to just *true*, or just *false* respectively. We also have two processes for evaluating the values decided upon: A^{gen}_1 checks that all the participants 1 to n agreed on a value (either *true* or *false*) or else crashed, producing the action $\overline{ok}$ if the test is successful; A^{true}_1 and A^{false}_1 check that all participants have agreed on the specific value *true*, and *false* respectively, or crashed.

Definition 3 (Consensus). *Let $\mathcal{L}_n$ denote $\{l_1, \ldots, l_n, \star\}$, and $(\tilde{m})$ stand for the actions $prop_i^{true}$, $prop_i^{false}$, dec_i^{true}, dec_i^{false} for $1 \leq i \leq n$. Then C satisfies consensus whenever*

Strong Basic Agreement: $\langle \mathcal{L}_n, 0\rangle \triangleright (\nu\tilde{m})(C \mid I^{gen} \mid A^{gen}_1) \quad \approx \quad \langle \emptyset, 0\rangle \triangleright start.\overline{ok}$

Basic Validity:

$$\langle \mathcal{L}_n, 0\rangle \triangleright (\nu\tilde{m})(C \mid I^{true} \mid A^{true}_1) \approx \langle \emptyset, 0\rangle \triangleright start.\overline{ok}$$
$$\langle \mathcal{L}_n, 0\rangle \triangleright (\nu\tilde{m})(C \mid I^{false} \mid A^{false}_1) \approx \langle \emptyset, 0\rangle \triangleright start.\overline{ok}$$

and moreover

Strong ft-Agreement: $\langle \mathcal{L}_n, 0\rangle \triangleright (\nu\tilde{m})(C \mid I^{gen} \mid A^{gen}_1) \quad \approx \quad \langle \mathcal{L}_n, n-1\rangle \triangleright (\nu\tilde{m})(C \mid I^{gen} \mid A^{gen}_1)$

ft-Validity:

$$\langle \mathcal{L}_n, 0\rangle \triangleright (\nu\tilde{m})(C \mid I^{true} \mid A^{true}_1) \approx \langle \mathcal{L}_n, n-1\rangle \triangleright (\nu\tilde{m})(C \mid I^{true} \mid A^{true}_1)$$
$$\langle \mathcal{L}_n, 0\rangle \triangleright (\nu\tilde{m})(C \mid I^{false} \mid A^{false}_1) \approx \langle \mathcal{L}_n, n-1\rangle \triangleright (\nu\tilde{m})(C \mid I^{false} \mid A^{false}_1)$$

In Definition 3 *strong agreement* subsumes the agreement and termination conditions: it composes C with I^{gen} and A^{gen}_1. *Validity* uses more specific wrappers, and composes C first with $\mathsf{I}^{true} \mid \mathsf{A}^{true}_1$ and then with $\mathsf{I}^{false} \mid \mathsf{A}^{false}_1$. Scoping the actions $prop_i^{true}$, $prop_i^{false}$, dec_i^{true} and dec_i^{false} in each test case limits external interaction to the non-failing actions *start* and $\overline{ok}$ at $\star$, the immortal location. This allows Definition 3 to divide *consensus* conditions into basic correctness and correctness preservation conditions. For example

Strong Agreement: $\langle \mathcal{L}_n, (n-1)\rangle \triangleright (\nu\tilde{m})(C \mid \mathsf{I}^{gen} \mid \mathsf{A}^{gen}_1) \quad \approx \quad \langle \emptyset, 0\rangle \triangleright start.\overline{ok}$

follows from **Strong Basic Agreement**, **Strong ft-Agreement** and transitivity of $\approx$.

Table 6. *Consensus Wrappers*

(Initialisation)		
$\mathsf{I}^x \overset{\text{def}}{=} \mathit{start}.\prod_{i=1}^{n} \overline{\mathit{prop}_i^x}$	$\mathsf{I}^{gen} \overset{\text{def}}{=} \mathit{start}.\prod_{i=1}^{n}(\overline{\mathit{prop}_i^{true}} + \overline{\mathit{prop}_i^{false}})$	$x \in \{\mathit{true}, \mathit{false}\}$
(Agreement)		
$\mathsf{A}_i^x \overset{\text{def}}{=} \mathit{dec}_i^x.\mathsf{A}_{i+1}^x + \mathsf{susp}\ l_i.\mathsf{A}_{i+1}^x$	$\mathsf{A}_{n+1}^x \overset{\text{def}}{=} \overline{\mathit{ok}}$	$x \in \{\mathit{true}, \mathit{false}\},\ i \leq n$
$\mathsf{A}_i^{gen} \overset{\text{def}}{=} \mathit{dec}_i^{true}.\mathsf{A}_{i+1}^{true} + \mathit{dec}_i^{false}.\mathsf{A}_{i+1}^{false} + \mathsf{susp}\ l_i.\mathsf{A}_{i+1}^{gen}$		$i \leq n$

4 Up-to Techniques in the Presence of Failure

Definition 3 expresses consensus in terms of six bisimulations. The main complication in proving these bisimulations lies in the large amount of internal actions that need to be considered. A large number of these internal actions are regular in structure (processes executing symmetric transitions at different locations and at different rounds) and most of these transitions are *confluent*; they do not affect the set of transitions that can be taken, either now or in the future. In the fault-tolerance bisimulations, we also have an extensive amount of *dead code*, that is code at dead locations or code that is forever blocked because it can only be released by actions at dead locations. Here we develop up-to bisimulation techniques that abstract over confluent moves and dead code.

We define a structural equivalence relation over configurations as the least relation satisfying the rules in Table 7. Even though this equivalence is normally defined over systems, we exploit the state of the network $\langle \mathcal{L}, n \rangle$ to define a stronger relation. Apart from the first six rules and the last two (contextual) rules, all of which are fairly standard, we also have new rules such as (s-Dead), adopted from [7], equating any code at dead locations, irrespective of its form. The network information is also used to define the new structural rule (gc-Susp), identifying suspicions that can never trigger because the location tested for can never fail; it is alive and no more failures can be induced. Also new is (gc-Act) which identifies action branches that can never trigger because there is no corresponding co-action within the action scope.[4] Our structural equivalence is a strong bisimulation.

Lemma 1 (≡ is a strong bisimulation).

$$\begin{tikzcd}
\langle \mathcal{L}, n \rangle \triangleright N \arrow[d, "\alpha"'] & \equiv & \langle \mathcal{L}, n \rangle \triangleright M & \mathit{implies} & \langle \mathcal{L}, n \rangle \triangleright N \arrow[d, "\alpha"'] & \equiv & \langle \mathcal{L}, n \rangle \triangleright M \arrow[d, "\alpha"'] \\
\langle \mathcal{L}', m' \rangle \triangleright N' & & & & \langle \mathcal{L}', m' \rangle \triangleright N' & \equiv & \langle \mathcal{L}', m' \rangle \triangleright M'
\end{tikzcd}$$

We now identify a number of τ-actions, referred to as β-actions or β-moves, and show that they are confluent. These silent β-actions are denoted as

$$\langle \mathcal{L}, n \rangle \triangleright N \overset{\tau}{\longmapsto}_{\beta} \langle \mathcal{L}, n \rangle \triangleright M$$

[4] We purposefully use the naming convention (gc-) for certain structural rules that are generally applied in one direction rather than the other to ”garbage collect” redundant dead code.

Table 7. *Structural Equivalence Rules*

(s-Scomm)	$\langle\mathcal{L},n\rangle \triangleright l[\![P+Q]\!] \equiv \langle\mathcal{L},n\rangle \triangleright l[\![Q+P]\!]$	
(s-Sassoc)	$\langle\mathcal{L},n\rangle \triangleright l[\![(P+Q)+R]\!] \equiv \langle\mathcal{L},n\rangle \triangleright l[\![P+(Q+R)]\!]$	
(s-inert)	$\langle\mathcal{L},n\rangle \triangleright l[\![P+\mathbf{0}]\!] \equiv \langle\mathcal{L},n\rangle \triangleright l[\![P]\!]$	
(s-Pcomm)	$\langle\mathcal{L},n\rangle \triangleright N \mid M \equiv \langle\mathcal{L},n\rangle \triangleright M \mid N$	
(s-Passoc)	$\langle\mathcal{L},n\rangle \triangleright (N \mid M) \mid M' \equiv \langle\mathcal{L},n\rangle \triangleright N \mid (M \mid M')$	
(gc-Inert)	$\langle\mathcal{L},n\rangle \triangleright M \mid l[\![\mathbf{0}]\!] \equiv \langle\mathcal{L},n\rangle \triangleright M$	
(s-Extr)	$\langle\mathcal{L},n\rangle \triangleright (\nu a)(M \mid N) \equiv \langle\mathcal{L},n\rangle \triangleright M \mid (\nu a)N$	$a \notin \mathbf{fn}(M)$
(gc-Scope)	$\langle\mathcal{L},n\rangle \triangleright (\nu a)M \equiv \langle\mathcal{L},n\rangle \triangleright M$	$a \notin \mathbf{fn}(M)$
(gc-Act)	$\langle\mathcal{L},n\rangle \triangleright (\nu a)l[\![\alpha.P + \sum_i P_i]\!] \equiv \langle\mathcal{L},n\rangle \triangleright (\nu a)l[\![\sum_i P_i]\!]$	$\alpha \in \{a, \bar{a}\}$
(gc-Susp)	$\langle\mathcal{L},0\rangle \triangleright l[\![\mathsf{susp}\ k.P + \sum_i P_i]\!] \equiv \langle\mathcal{L},0\rangle \triangleright l[\![\sum_i P_i]\!]$	$k \in \mathcal{L}$
(s-Dead)	$\langle\mathcal{L},n\rangle \triangleright l[\![P]\!] \equiv \langle\mathcal{L},n\rangle \triangleright l[\![Q]\!]$	$l \notin \mathcal{L}$

(s-Rest)

$$\frac{\langle\mathcal{L},n\rangle \triangleright M \equiv \langle\mathcal{L},n\rangle \triangleright N}{\langle\mathcal{L},n\rangle \triangleright (\nu a)M \equiv \langle\mathcal{L},n\rangle \triangleright (\nu a)N}$$

(Par)

$$\frac{\langle\mathcal{L},n\rangle \triangleright M \equiv \langle\mathcal{L},n\rangle \triangleright M'}{\begin{array}{c}\langle\mathcal{L},n\rangle \triangleright M|N \equiv \langle\mathcal{L},n\rangle \triangleright M'|N \\ \langle\mathcal{L},n\rangle \triangleright N|M \equiv \langle\mathcal{L},n\rangle \triangleright N|M'\end{array}}$$

and defined in Table 8. We then develop up-to bisimulation techniques that abstract from matching configurations related by β-moves. The details differ considerably from [7] because we use different constructs like choice and failure detection, and allow distributed synchronisation across locations. Apart from the standard local rules (BNew) and (BFork), and the context rules (BRest) and (BPar), Table 8 has three new rules dealing with synchronisations. (BLin) states that distribution does not interfere with a scoped linear synchronisation, as long as *we cannot induce more dynamic failures*, that is $n = 0$. (BLoc) states that a *local* scoped linear synchronisation is always a β-move. Finally, (BFTol) states that a distributed scoped linear synchronisation is a β-move if it is *asynchronous from one end* and the co-synchronisation at the other end is *guarded by a* susp *with the same continuation*; these conditions make τ-move in (BFTol), in a sense, *fault-tolerant* as we have already seen in (2). We prove a special form of confluence for our β-moves.

Lemma 2 (Confluence of β-moves). $\overset{\tau}{\longmapsto}_\beta$ *observes the diamond property:*

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

where $\mathcal{R}$ *is* $\overset{\tau}{\longmapsto}_\beta$ *or* $\equiv$, *or else* $\alpha = \tau$ *and* $\langle\mathcal{L},n\rangle \triangleright M = \langle\mathcal{L}',n'\rangle \triangleright N'$

Note the use of the non-standard $\mathcal{R}$ to close the diamond instead of $\overset{\tau}{\longmapsto}_\beta$ in this Lemma. It allows for the special case when the code causing the β-move crashes. In this case, we only require that resulting pair are structurally equivalent, using (s-Dead).

We defined a modified bisimulation relation from Definition 1 where the conditions for the matching residuals are relaxed; instead of demanding that they are again related in $\approx$ we allow approximate matching through $\equiv$ and $\overset{\tau}{\longmapsto}{}^*_\beta$.

Table 8. *Transition Rules for β-moves*

Assuming $l \in \mathcal{L}$, $n \geq 0$

(BLin)

$$\frac{}{\langle \mathcal{L}, 0\rangle \triangleright (\nu a)(l[\![\bar{a}.P]\!] \mid k[\![a.Q]\!]) \ \xrightarrow{\tau}_{\beta}\ \langle \mathcal{L}, 0\rangle \triangleright (\nu a)(l[\![P]\!] \mid k[\![Q]\!])}\ l, k \in \mathcal{L}$$

(BLoc)

$$\frac{}{\langle \mathcal{L}, n\rangle \triangleright (\nu a)(l[\![\bar{a}.P]\!] \mid l[\![a.Q]\!]) \ \xrightarrow{\tau}_{\beta}\ \langle \mathcal{L}, n\rangle \triangleright (\nu a)(l[\![P]\!] \mid l[\![Q]\!])}$$

(BFTol)

$$\frac{}{\langle \mathcal{L}, n\rangle \triangleright (\nu a)(l[\![\bar{a}]\!] \mid k[\![a.P + \mathsf{susp}\ l.P]\!]) \ \xrightarrow{\tau}_{\beta}\ \langle \mathcal{L}, n\rangle \triangleright (\nu a)k[\![P]\!]}\ l, k \in \mathcal{L}$$

(BNew)

$$\frac{}{\langle \mathcal{L}, n\rangle \triangleright l[\![(\nu a)P]\!] \xrightarrow{\tau}_{\beta} \langle \mathcal{L}, n\rangle \triangleright (\nu a)l[\![P]\!]}\ l \in \mathcal{L}$$

(BRest)

$$\frac{\langle \mathcal{L}, n\rangle \triangleright M \xrightarrow{\tau}_{\beta} \langle \mathcal{L}, n\rangle \triangleright M'}{\langle \mathcal{L}, n\rangle \triangleright (\nu a)M \xrightarrow{\tau}_{\beta} \langle \mathcal{L}, n\rangle \triangleright (\nu a)M'}$$

(BFork)

$$\frac{}{\langle \mathcal{L}, n\rangle \triangleright l[\![P|Q]\!] \xrightarrow{\tau}_{\beta} \langle \mathcal{L}, n\rangle \triangleright l[\![P]\!]|l[\![Q]\!]}\ l \in \mathcal{L}$$

(BPar)

$$\frac{\langle \mathcal{L}, n\rangle \triangleright M \xrightarrow{\tau}_{\beta} \langle \mathcal{L}, n\rangle \triangleright M'}{\begin{array}{c}\langle \mathcal{L}, n\rangle \triangleright M|N \xrightarrow{\tau}_{\beta} \langle \mathcal{L}, n\rangle \triangleright M'|N \\ \langle \mathcal{L}, n\rangle \triangleright N|M \xrightarrow{\tau}_{\beta} \langle \mathcal{L}, n\rangle \triangleright N|M'\end{array}}$$

Definition 4 (β-transfer property). *A relation $\mathcal{R}$ over configurations satisfies the β-transfer property if*

$$\begin{tikzcd}
\langle \mathcal{L}, n\rangle \triangleright N \arrow[d, "\alpha"'] \arrow[r, phantom, "\mathcal{R}"] & \langle \mathcal{L}, n\rangle \triangleright M & \textit{implies} & \langle \mathcal{L}, n\rangle \triangleright N \arrow[d, "\alpha"'] \arrow[r, phantom, "\mathcal{R}"] & \langle \mathcal{L}, n\rangle \triangleright M \arrow[d, "\alpha"'] \\
\langle \mathcal{L}', n'\rangle \triangleright N' & & & \langle \mathcal{L}', n'\rangle \triangleright N' \arrow[r, phantom, "\mathcal{A}_l \circ \mathcal{R} \circ \mathcal{A}_r"] & \langle \mathcal{L}', n'\rangle \triangleright M'
\end{tikzcd}$$

where $\mathcal{A}_l$ *is* $\equiv \circ \xrightarrow{\tau}{}^{*}_{\beta}$ *and* $\mathcal{A}_r$ *is* $\approx$

Definition 5 (Bisimulation up-to-β). *A relation $\mathcal{R}$ over configurations is a bisimulation up-to-β if it and its inverse $\mathcal{R}^{-1}$ satisfy the β-transfer property.*

Before we can use bisimulations up-to-β, we need to show they are sound with respect to Definition 1. This soundness proof uses the results of Lemma 3.

Lemma 3 ($\xrightarrow{\tau}{}^{*}_{\beta}$ implies $\approx$). $\langle \mathcal{L}, n\rangle \triangleright N \ \xrightarrow{\tau}{}^{*}_{\beta}\ \langle \mathcal{L}, n\rangle \triangleright M$ *implies* $\langle \mathcal{L}, n\rangle \triangleright N \approx \langle \mathcal{L}, n\rangle \triangleright M$.

Theorem 1 (Soundness of bisimulations up-to-β). *If* $\langle \mathcal{L}, n\rangle \triangleright N\, \mathcal{R}\, \langle \mathcal{L}', m\rangle \triangleright M$, *where* $\mathcal{R}$ *is a bisimulation up-to-β, then* $\langle \mathcal{L}, n\rangle \triangleright N \approx \langle \mathcal{L}', m\rangle \triangleright M$.

Example 3. Suppose $l, k \in \mathcal{L}$. Then we can show that

$$\langle \mathcal{L}, n\rangle \triangleright (\nu a, b)\ l[\![\bar{a}]\!] \mid k[\![a.P + b.Q + \mathsf{susp}\ l.P]\!] \ \approx\ \langle \mathcal{L}, n\rangle \triangleright (\nu a, b)k[\![P]\!] \qquad (3)$$

To see this first note that using (s-Extr), (s-Scomm), (s-Sassoc), (gc-Act) and (s-Extr) again we can tighten the scope of νb, garbage collect the branch guarded by b and then scope extrude νb again to obtain

$$\langle \mathcal{L}, n\rangle \triangleright (\nu a, b)\; l\llbracket \bar{a} \rrbracket \,|\, k\llbracket a.P + b.Q + \mathsf{susp}\; l.P \rrbracket \equiv \langle \mathcal{L}, n\rangle \triangleright (\nu a, b)\; l\llbracket \bar{a} \rrbracket \,|\, k\llbracket a.P + \mathsf{susp}\; l.P \rrbracket$$

An application of (BFTol) gives

$$\langle \mathcal{L}, n\rangle \triangleright (\nu a, b)\; l\llbracket \bar{a} \rrbracket \,|\, k\llbracket a.P + \mathsf{susp}\; l.P \rrbracket \quad \overset{\tau}{\longmapsto}_{\beta} \quad \langle \mathcal{L}, n\rangle \triangleright (\nu a, b)\; k\llbracket P \rrbracket$$

and now (3) follows from Lemma 1 and Lemma 3.

5 Consensus Satisfaction Proof

Using Theorem 1, we just need to give witness bisimulations up-to β-moves satisfying the bisimulations set out in Definition 3. In the following witness bisimulations, we use the letters t, f, p and d for the action names true, false, prop and dec, respectively. Our presentation makes use of sets of integers I_i partitioning the set $\{1 \ldots n\}$; the partition predicate is:

$$\mathsf{part}_1^n(I_1, \ldots, I_k) \quad \overset{\text{def}}{=} \quad I_1 \cup \ldots \cup I_k = \{1 \ldots n\} \;\text{ and }\; \forall i, j \in \{1 \ldots k\}\; I_i \cap I_j = \emptyset$$

We also denote the smallest number in a partition I_i as $I_{i\,min}$ and the largest number in a partition I_i that is smaller that any element in any other partition I_j as $I^{+}_{i\,min}$.

We first prove the basic (failure-free) equivalences. We here only give the witness bisimulation for Strong Basic Agreement; the two witness bisimulations required for Basic Validity are similar but simpler. We assume $\tilde{m} = \prod_{i,r=1}^{n} t_{i,r}, f_{i,r}, p_i^t, p_i^f, d_i^t, d_i^f$ and use A, I, $\mathcal{L}_n$ and $\emptyset$ as shorthand for A_1^{gen}, I^{gen}, $\langle\{1 \ldots n\}, 0\rangle$ and $\langle\emptyset, 0\rangle$ respectively. We also partition $\{1 \ldots n\}$ into three sets: I denotes the set of *uninitialised* participants, whereas J and H denote initialised participants with current estimates t and f respectively; when we do not use partition H, participants in J all have either estimate t or f. We also use the process definition $N_i \overset{\text{def}}{=} l_i\llbracket p_i^t.\mathsf{P}_{i,1}^t + p_i^f.\mathsf{P}_{i,1}^f \rrbracket \mid \overline{p_i^t} + \overline{p_i^f}$ for *non-initialised* participant i.

$$\left\{\begin{array}{lc|l}
1) & \Big\langle \mathcal{L}_n \triangleright (\nu\tilde{m})\Big(C \,|\, \mathsf{I}^{gen} \,|\, \mathsf{A}_1^{gen}\Big),\; \emptyset \triangleright \mathit{start}.\overline{ok} \Big\rangle & \\[2ex]
2) & \left\langle \mathcal{L}_n \triangleright (\nu\tilde{m})\left(\mathsf{A} \,|\, \prod\limits_{i\in I} N_i \,|\, \prod\limits_{j\in J} l_j\llbracket \mathsf{R}_{j,1}^t \rrbracket \,|\, \prod\limits_{h\in H} l_h\llbracket \mathsf{R}_{h,1}^f \rrbracket\right),\; \emptyset \triangleright \overline{ok} \right\rangle & \begin{array}{l}\mathsf{part}_1^n(I,J,H)\\ \text{and } I_{min} = 1\end{array} \\[3ex]
3) & \left\langle \mathcal{L}_n \triangleright (\nu\tilde{m})\left(\mathsf{A} \,|\, \prod\limits_{i\in I}\left(N_i \,|\, \prod\limits_{r=1}^{I_{min}-1} l_r\llbracket \overline{x_{i,r}} \rrbracket\right) \,|\, \prod\limits_{j\in J} l_j\llbracket \mathsf{R}_{j,I_{min}}^x \rrbracket\right),\; \emptyset \triangleright \overline{ok} \right\rangle & \begin{array}{l}\mathsf{part}_1^n(I,J)\\ \text{and } I_{min} \neq 1\\ \text{and } x \in \{t, f\}\end{array} \\[3ex]
4) & \Big\langle \mathcal{L}_n \triangleright \overline{ok},\; \emptyset \triangleright \overline{ok} \Big\rangle &
\end{array}\right\}$$

In the above up-to β witness bisimulation case (2) represents the states where participants have different estimates at round $r = 1$ because the broadcaster at round 1 has not been initialised yet. Case (3) represents participants in agreement for rounds $r \geq 2$, but blocked because the co-ordinator participant for round r has not been initialised. We note that in case (3), uninitialised participants $i \in I$ include the broadcasted values from previous rounds that are yet to be consumed by them once they are initialised.

We highlight the salient aspect of the above bisimulation relation: apart from the initialisation τ-moves, *all* the remaining τ-transitions turn out to be β-moves; they are instances of (BLin) (modulo $\equiv$). We illustrate this through a walk-through of the main transitions:

- If we are in (2) and the j^{th} participant in the left configuration is initialised (through a τ action) with $x \in \{t, f\}$ then
  - if $j \neq 1$ the participant proceeds to round 1 with estimate x and joins set J or H accordingly. We match this action by the empty move and remain in case (2).
  - if $j = 1$ the participant proceeds to round 1 and acts as the co-ordinator, broadcasting x. For all participants $j \in J$ or $h \in H$, broadcast synchronisation turns out to be a β-move using (BLin), and (gc-Act) and (gc-Susp), among other rules, to garbage collect inactive branches as in Example 3. At this point all initialised participants agree on the broadcasted value x at round 2, and proceed through the next rounds using β-moves, still agreeing on x, until they block again on the next I_{min}. We match this action with the empty action and progress to case (3).
- If we are in (3) and the i^{th} participant is initialised then
  - if $i \neq I_{min}$ then the right configuration performs an empty move and we remain in case (3), abstracting away from the β-moves of participant i consuming all the broadcasts to reach round I_{min} with estimate x.
  - if $i = I_{min}$ then the matching move is similar but with two further sub-cases
    - If $I = \{i\}$ then all participant would have agreed on x, the first broadcasted value and we progress to case (4) through a series of β-moves.
    - If $|I| \geq 2$ then all participants $j \in J$ progress to the round of the next minimum uninitialised participant $(I/\{i\})_{min}$, and remain in case (3).

The witness bisimulation for Strong ft-Agreement up to $(n-1)$ faults is given below; we leave similar but simpler witness bisimulations for ft-Validity to the interested reader. We carry over all the shorthand notation used for the failure-free witness bisimulation together with some more: the operation $\ddot{x}$ denotes value inverse for $x \in \{t, f\}$, and is defined as $\ddot{t} = f$ and $\ddot{f} = t$; for $K \subset \{1 \ldots n\}$, $\mathcal{L}_n^K$ denotes the network state $\langle \mathcal{L}_n/\{l_k \mid k \in K\}, n - |K| \rangle$; $B(i, x)_j^{j+n}$ denotes the sequence of broadcasts of x for participant i from rounds j up to $j + n$, that is $\prod_{r=j}^{j+n} l_r[\![\bar{x}_{i,r}]\!]$.

The salient aspect of our correctness preservation witness bisimulations is that they automatically bring to the fore the mechanisms that enable the algorithm to overcome decision blocking and corrupted broadcast. Through the use of the β-moves (BLoc), in the case of participant initialisation in ft-Validity, and (BFTol), in the case of broadcast communications where a participant receives the same estimate it currently holds, our witness bisimulations abstract over superfluous transitions. This means that the only non-confluent τ-moves remaining are those for participant initialisation, in the case of Strong ft-Agreement, those that crash participants and those where the broadcasted value and the current participant estimate differ. The latter two kinds are the core transitions that embody corrupted broadcast, when the broadcaster crashes, and lead towards the eventual agreement, when not interfered with by failure. The up-to-β level of abstraction also makes the overall structure of the bisimulation proof reflect move closely the reasoning needed in a careful, informal proof of correctness.

$$
\left\{
\begin{array}{lll}
1) & \langle \mathcal{L}_n \triangleright (\nu\tilde{m})\mathsf{A} \mid \mathsf{I} \mid \mathsf{C},\ \mathcal{L}_n^K \triangleright (\nu\tilde{m})\mathsf{A} \mid \mathsf{I} \mid \mathsf{C}\rangle & |K \subseteq \{1\dots n\} \\[2ex]
2) & \begin{array}{l}
\Big\langle \mathcal{L}_n \triangleright (\nu\tilde{m})\Big(\mathsf{A} \mid \prod\limits_{i\in I} N_i \mid \prod\limits_{j\in J} l_j\llbracket \mathsf{R}^t_{j,1}\rrbracket \mid \prod\limits_{h\in H} l_h\llbracket \mathsf{R}^f_{h,1}\rrbracket \mid \prod\limits_{k\in K} l_k\llbracket P_k\rrbracket\Big) \\
\quad , \mathcal{L}_n^K \triangleright (\nu\tilde{m})\Big(\mathsf{A} \mid \prod\limits_{i\in I} N_i \mid \prod\limits_{j\in J} l_j\llbracket \mathsf{R}^t_{j,1}\rrbracket \mid \prod\limits_{h\in H} l_h\llbracket \mathsf{R}^f_{h,1}\rrbracket\Big)\Big\rangle
\end{array}
& \begin{array}{|l}
\mathsf{part}^n_1(I, J, H, K) \\
\text{and } I_{min} = 1
\end{array} \\[4ex]
3) & \begin{array}{l}
\Big\langle \mathcal{L}_n \triangleright (\nu\tilde{m})\left(\begin{array}{l}
\mathsf{A} \mid \prod\limits_{i\in I}\Big(N_i \mid B(i,x)_1^{I_{min}-1}\Big) \mid \prod\limits_{j\in J} l_j\llbracket \mathsf{R}^x_{j,I_{min}}\rrbracket \\
\mid \prod\limits_{h\in H} l_h\llbracket \mathsf{R}^x_{h,I_{min}}\rrbracket \mid \prod\limits_{k\in K} l_k\llbracket P_k\rrbracket
\end{array}\right) \\
\quad , \mathcal{L}_n^K \triangleright (\nu\tilde{m})\Big(\mathsf{A} \mid \prod\limits_{i\in I} N_i \mid \prod\limits_{j\in J} l_j\llbracket \mathsf{R}^y_{j,I_{min}}\rrbracket \mid \prod\limits_{h\in H} l_h\llbracket \mathsf{R}^{\ddot{y}}_{h,I_{min}}\rrbracket\Big)\Big\rangle
\end{array}
& \begin{array}{|l}
\mathsf{part}^n_1(I, J, H, K) \\
\text{and } 1 \neq I_{min} < J_{min} \\
\text{and } I_{min} < H_{min} \\
\text{and } x, y \in \{t, f\} \\
\text{and } \{1, \dots, (I_{min}-1)\} \subseteq K
\end{array} \\[6ex]
4) & \begin{array}{l}
\Big\langle \mathcal{L}_n \triangleright (\nu\tilde{m})\left(\begin{array}{l}
\mathsf{A} \mid \prod\limits_{i\in I}\Big(N_i \mid B(i,x)_1^{I_{min}-1}\Big) \mid \prod\limits_{j\in J} l_j\llbracket \mathsf{R}^x_{j,I_{min}}\rrbracket \\
\mid \prod\limits_{h\in H} l_h\llbracket \mathsf{R}^x_{h,I_{min}}\rrbracket \mid \prod\limits_{k\in K} l_k\llbracket P_k\rrbracket
\end{array}\right) \\
\quad , \mathcal{L}_n^K \triangleright (\nu\tilde{m})\left(\begin{array}{l}
\mathsf{A} \mid \prod\limits_{i\in I}\Big(N_i \mid B(i,y)_{J_{min}}^{J^+_{min}}\Big) \mid \prod\limits_{j\in J} l_j\llbracket \mathsf{R}^y_{j,I_{min}}\rrbracket \\
\mid \prod\limits_{h\in H}\Big(l_h\llbracket \mathsf{R}^{\ddot{y}}_{h,J_{min}}\rrbracket \mid B(i,y)_{J_{min}}^{J^+_{min}}\Big)
\end{array}\right)\Big\rangle
\end{array}
& \begin{array}{|l}
\mathsf{part}^n_1(I, J, H, K) \\
\text{and } J_{min} < I_{min} < H_{min} \\
\text{and } x, y \in \{t, f\} \\
\text{and } |J|, |I| \geq 1
\end{array} \\[6ex]
5) & \begin{array}{l}
\Big\langle \mathcal{L}_n \triangleright (\nu\tilde{m})\left(\begin{array}{l}
\mathsf{A} \mid \prod\limits_{i\in I}\Big(N_i \mid B(i,x)_1^{I_{min}-1}\Big) \mid \prod\limits_{j\in J} l_j\llbracket \mathsf{R}^x_{j,I_{min}}\rrbracket \\
\mid \prod\limits_{h\in H} l_h\llbracket \mathsf{R}^x_{h,I_{min}}\rrbracket \mid \prod\limits_{k\in K} l_k\llbracket P_k\rrbracket
\end{array}\right) \\
\quad , \mathcal{L}_n^K \triangleright (\nu\tilde{m})\left(\begin{array}{l}
\mathsf{A} \mid \prod\limits_{i\in I}\Big(N_i \mid B(i,y)_{J_{min}}^{J^+_{min}}\Big) \mid \prod\limits_{j\in J} l_j\llbracket \mathsf{R}^y_{j,H_{min}}\rrbracket \\
\mid \prod\limits_{h\in H}\Big(l_h\llbracket \mathsf{R}^{\ddot{y}}_{h,J_{min}}\rrbracket \mid B(i,y)_{J_{min}}^{J^+_{min}}\Big)
\end{array}\right)\Big\rangle
\end{array}
& \begin{array}{|l}
\mathsf{part}^n_1(I, J, H, K) \\
\text{and } J_{min} < H_{min} < I_{min} \\
\text{and } x, y \in \{t, f\} \\
\text{and } |J|, |H|, |I| \geq 1
\end{array} \\[6ex]
6) & \begin{array}{l}
\Big\langle \mathcal{L}_n \triangleright (\nu\tilde{m})\Big(\mathsf{A} \mid \prod\limits_{j\in J} l_j\llbracket \mathsf{R}^x_{j,I_{min}}\rrbracket \mid \prod\limits_{h\in H} l_h\llbracket \mathsf{R}^x_{h,I_{min}}\rrbracket \mid \prod\limits_{k\in K} l_k\llbracket P_k\rrbracket\Big) \\
\quad , \mathcal{L}_n^K \triangleright (\nu\tilde{m})\Big(\mathsf{A} \mid \prod\limits_{j\in J} l_j\llbracket \mathsf{R}^y_{j,H_{min}}\rrbracket \mid \prod\limits_{h\in H}\Big(l_h\llbracket \mathsf{R}^{\ddot{y}}_{h,J_{min}}\rrbracket \mid B(i,y)_{J_{min}}^{J^+_{min}}\Big)\Big)\Big\rangle
\end{array}
& \begin{array}{|l}
\mathsf{part}^n_1(J, H, K) \\
\text{and } J_{min} < H_{min} \\
\text{and } x, y \in \{t, f\} \\
\text{and } |J|, |H| \geq 1
\end{array} \\[4ex]
7) & \mathcal{L}_n \triangleright \overline{ok},\ \ \mathcal{L}_n^K \triangleright \overline{ok} &
\end{array}
\right\}
$$

Characterised by the non-confluent transitions (participant initialisation, participant crashing and broadcasts where the value broadcasted and the current participant estimate differ), the witness bisimulation partitions the n participants into 4 mutually exclusive sets: I denotes the participants that are yet *uninitialised*, K denotes the participants that have *crashed*, J denotes the participants with estimate x, the value being broadcasted at the current round and H denote the participants with current estimate $\ddot{x}$ differing from broadcasted value x at the current round.

Based on these participant partitions, the witness bisimulation describes the following cases for bisimilar pairs: in (2) no broadcast has yet occured because the first co-ordinator is still uninitialised; (3) is a similar case where the *live* participant with the lowest index i is uninitialised (all the participants $< i$ have crashed); (4) describes the case when the live participant with the lowest index j is initialised with x, and all initialised participants with estimate x are blocked because I_{min} is yet to be initialised; (5) is similar to case (4), only that participants with estimate x that is being broadcasted are blocked on an *initialised* participant from partition H with estimate $\ddot{y}$ which still needs to consume a broadcast (and change its estimate); (6) is a special case of (5) where there are no uninitialised participants. Thus, in this last case, (6), we map live blocked participants in a dynamic failure setting to unblocked participants in a failure free setting at the final round n.

We note that witness bisimulation shows that even though agreement is reached in both failure-free (left) and dynamic failure (right) sides, each side may agree on different values at round $(n+1)$. More specifically in the failure-free setting agreement is reached on the value to which the first participant is initialised; this is not necessarily the case in dynamic failure setting. We also note that the witness bisimulation is uncluttered from crashed code through the structural rule (s-Dead). Thus, in every bisimilar pair, it maps the corresponding live code in a left (failure-free) configuration, irrespective of its state, to the inert process **0**, on the right. We overview the main transitions of the important (enumerated) stages in this relation, that is for stages (3), (4), (5) and (6):

Stage (3)**:** If participant $i \in I$ is initialised, then we go to stage (4) or (5), depending on the value y it is initialised to and whether $(I/\{i\})^{+}_{min} < J_{min}, H_{min}$. If participant $i \in I$ crashes, then if $(I/\{i\})^{+}_{min} < J_{min}, H_{min}$ we remain in (3) else go to stage (4) or (5).

Stage (4)**:** If participant $j \in J$ crashes, then if $J_{min} = J^{+}_{min}$ we go to stage (3), otherwise we remain in (4). If participant $i \in I$ is initialised we have a number of cases: if it is initialised to y or it is initialised to $\ddot{y}$ and $(i \neq I_{min})$ then

- we remain in (4) if $|I| \neq 1$.
- we go to (6) if $|I| = 1$.
- we go to (7) if $(|H| = 0 \wedge |I| = 1)$.

Else, if the I_{min} is initialised to $\ddot{y}$, then

- we go to (5) if $|I| \neq 1$, swapping J for H and vice-versa.
- we go to (6) if $|I| = 1$, again swapping J for H and vice-versa.
- we go to (7) if $(|H| = 0 \wedge |I| = 1)$

Similarly, if participant I_{min} crashes, then depending on the next smallest participant every $j \in J$ blocks on, we can either remain in (4) or transition to stage (5) if $|I| \neq 1$, stage (6) if $|I| = 1$ or stage (7) if $(|H| = 0 \wedge |I| = 1)$. Finally, if participant $h \in H$ consumes the broadcasts or crashes, we still remain in stage (4), potentially making $|H| = 0$.

Stage (5)**:** If participant $j \in J$ crashes, then if $J_{min} \neq J^{+}_{min}$ we remain in (5), otherwise we transition to stage (4) where H is swapped for J (and vice-versa). If participant

$h \in H$ accepts the broadcast or crashes, we remain in (5) or transition back to (4), depending on whether $H_{min} = H^{+}_{min}$. If participant $i \in I$ is initialised, we still remain in (5) whereas if $i \in I$ crashes, we remain in (5) or transition to (6) if $|I| = 1$.

Stage (6)**:** If participant $j \in J$ crashes, then if $|J| = 1$ we reach agreement and go to stage (7), otherwise we remain in (6), possibly swapping participants $h \in H$ for participants $j \in J$. If participant $h \in H$ accepts the broadcast or crashes, we transition to stage (7) if $|H| = 1$ or remain in (6).

All the above transitions are matched by the empty transition on the failure-free side, except those transitions that involve initialising participants: In this case we match the transition by initialising the corresponding participant in the failure-free setting.

6 Conclusion

We have designed a *partial-failure* process calculus in which distributed algorithms can be formally described and analysed. We have also developed up-to techniques in this calculus by identifying novel confluent moves involving the choice and perfect failure detection operator, together with a stronger structural equivalence abstracting over dead code. Most importantly however, we have proposed a methodology for formally proving the correctness of distributed algorithms in the presence of failure using fault-tolerance bisimulation techniques. We have shown how this methodology can alleviate the burned of exhibiting such formal proofs by giving, what to our knowledge is, the first bisimulation-based proof of Consensus with perfect failure detectors. Moreover, the decomposition of the proof into basic correctness and correctness preservation equivalences permits separation of concerns and leads to a better understanding of the role and weight of each action in the studied algorithm.

Future Work: There are various possible extension to our calculus. We can weaken our failure detectors to $\diamond\mathcal{S}$, [2], by enhancing our network representation with two livesets, suspectable and non-suspectable, similar to the techniques used in [14, 13]. We can also introduce recursive computation, which would allow us to study consensus solving algorithms with no static bounds on the number of rounds. Such a study would require more sophisticated reasoning about termination; work such as [3, 17] should shed more light on this complication. Independent of the calculus, we plan to validate our proposed methodology by applying it to a range of fault-tolerant distributed algorithms expressed in various calculi; examples of such algorithms include those in [11, 16].

Related Work: The confluence of certain τ-steps has long been known as a useful technique in the management of bisimulations, [9]. See [8] for particularly good examples of where they have significantly decreased the size of witness bisimulations. We have extended the concept, by considering confluence up to a particularly strong form of structural equivalence which enables useful garbage collections to be carried out in fault-tolerance proofs, by virtue of the presence of dead locations.

The closest to our work is [14], where the correctness of a consensus solving algorithm for a more complex setting which uses $\diamond\mathcal{S}$ failure detectors is formalised using a process calculus. However, their proof methods differ from ours: they give a translation from the calculus encoding of the algorithm into an abstract interpretation and then perform correctness analysis on the abstract interpretation. Results similar to ours are also presented in [1]; there the atomicity of the 2-phase commit protocol is encoded and proved correct using a process calculus with persistence and transient failure; bisimulations are used to obtain algebraic laws which are then used to prove atomicity.

Acknowledgments: We would like to thank the referees for their incisive comments on a preliminary version of this paper.

References

1. Martin Berger and Kohei Honda. The two-phase commitment protocol in an extended pi-calculus. *Electr. Notes Theor. Comput. Sci.*, 39(1), 2000.
2. Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, March 1996.
3. Yuxin Deng and Davide Sangiorgi. Ensuring termination by typability. In *IFIP TCS*, pages 619–632, 2004.
4. Michael J. Fischer. The consensus problem in unreliable distributed systems (a brief survey). In *Proceedings of the 1983 International FCT-Conference on Fundamentals of Computation Theory*, pages 127–140. Springer-Verlag, 1983.
5. Cedric Fournet, Georges Gonthier, Jean Jaques Levy, and Remy Didier. A calculus of mobile agents. *CONCUR 96*, LNCS 1119:406–421, August 1996.
6. 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.
7. 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.
8. J. F. Groote and M. P. A. Sellink. Confluence for process verification. *Theor. Comput. Sci.*, 170(1-2):47–81, 1996.
9. 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.
10. K. Honda and N. Yoshida. On reduction-based process semantics. *Theoretical Computer Science*, 152(2):437–486, 1995.
11. Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.
12. R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.
13. Uwe Nestmann and Rachele Fuzzati. Unreliable failure detectors via operational semantics. In *ASIAN*, pages 54–71, 2003.
14. Uwe Nestmann, Rachele Fuzzati, and Massimo Merro. Modeling consensus in a process calculus. In *CONCUR: 14th International Conference on Concurrency Theory*. LNCS, Springer-Verlag, 2003.
15. James Riely and Matthew Hennessy. Distributed processes and location failures. *Theoretical Computer Science*, 226:693–735, 2001.
16. Gerard Tel. *Introduction to distributed algorithms*. Cambridge University Press, New York, NY, USA, 1994.
17. Nobuko Yoshida, Martin Berger, and Kohei Honda. Strong normalisation in the pi-calculus. *Inf. Comput.*, 191(2):145–202, 2004.