Modeling Consensus in a Process Calculus[⋆]

Uwe Nestmann[1], Rachele Fuzzati[1], and Massimo Merro[2]

[1] EPFL, Switzerland
[2] University of Verona, Italy

Abstract. We give a process calculus model that formalizes a well-known algorithm (introduced by Chandra and Toueg) solving consensus in the presence of a particular class of failure detectors ($\Diamond\mathcal{S}$); we use our model to formally prove that the algorithm satisfies its specification.

1 Introduction & Summary

This paper serves the following purposes: (1) to report on the first *formal* proof known to us of a Consensus algorithm developed by Chandra and Toueg using a particular style of failure detectors [CT96]; (2) to demonstrate the feasibility of using process calculi to carry out solid proofs for such algorithms; (3) to report on an operational semantics model for failure detectors that is easier to use in proofs than the original one based on so-called failure patterns.

Distributed Consensus In the field of Distributed Algorithms, a widely-used computation model is based on *asynchronous* communication between a fixed number n of connected processes, where *no timing assumptions* can be made. Moreover, processes are subject to *crash-failure*: once crashed, they do not recover. The Distributed Consensus problem is well-known in this field: initially, each process proposes some value; eventually, all processes who do not happen to crash shall agree on one of the proposed values. More precisely, Consensus is specified by the following three properties on possible runs of a system.

Termination: Every correct process (eventually) decides some value.
Validity: If a process decides v, then v was proposed by some process.
Agreement: No two correct processes decide differently.

Here, a process is called *correct* in a given run, if it does not crash *in this run*. An important impossibility result states that Consensus cannot be solved in the aforementioned computation model when even a single process may fail [FLP85]. Since this impossibility result, several refinements of the computation model have been developed to overcome it. One of them is the addition of *unreliable failure detectors* (FD), i.e., modules attached to each process that can be locally queried to find out whether another process is currently locally suspected to

[⋆] Appeared in the Proceedings of CONCUR 2003, LNCS 2761 (© Springer-Verlag). Supported by the Swiss National Science Foundation, grant No. 21-67715.02, the Hasler Foundation, grant No. DICS 1825, an EPFL start-up grant, and the EU FET-GC project PEPITO. The full version is available at: http://lamp.epfl.ch/~uwe/

have crashed [CT96, CHT96]. FDs are unreliable in that they may have wrong suspicions, they may disagree among themselves, and they may change their suspicions at any time. To become useful, the behavior of FDs is constrained by abstract reliability properties about (i) the guaranteed suspicion of crashed processes, and (ii) the guaranteed non-suspicion of correct processes. Obviously, due to the run-based definition of correctness of processes, also these constraints are expressed over runs. A number of different combinations of FD-constraints were proposed in [CT96], one pair of which is commonly referred to as $\Diamond\mathcal{S}$:

Strong Completeness (SC): Eventually every process that crashes is permanently suspected by (the FD of) every correct process.

Eventual Weak Accuracy (EWA): There is a time after which some correct process is never suspected by (the FD of) any correct process.

Chandra and Toueg also provide an algorithm—using pseudo-code, without formal semantics—in the context of FDs satisfying the reliability constraints of $\Diamond\mathcal{S}$. The algorithm solves Consensus under the condition that a majority $\lceil\frac{n+1}{2}\rceil$ of processes are correct. It proceeds in *rounds* and is based on the *rotating coordinator* paradigm: for each round number, a single process is predetermined to play a coordinator role, while all other processes in this round play the role of participants. Each of the n processes counts rounds locally and knows at any time, who is the coordinator of its current round. Note that, due to asynchrony, any such system may easily reach states, in which all processes are in different rounds. Each round proceeds in four phases, in which (1) each participant sends to the coordinator of its current round its current estimate of the consensus value stamped with the round number at which it adopted this estimate; (2) the coordinator waits for sufficiently many estimates to arrive, selects one of those with the highest stamp; this is the round proposal that is distributed to the participants; (3) each participant either waits for the coordinator's round-proposal or, if this is currently permitted by its local FD, suspects the coordinator—in both cases, participants then send (positive or negative) acknowledgments to the coordinator and proceed to the next round; (4) the coordinator waits for sufficiently many acknowledgments; if they are all positive it proceeds to the decision, otherwise it proceeds to the next round. "Deciding on a value" means to send the value to all processes using Reliable Broadcast (RB). The reception of an RB-message is called RB-*delivery*; processes may RB-deliver independent of their current round and phase. On RB-delivery, a process "officially" decides on the broadcast value. Note that also the broadcast-initiator must perform RB-delivery. Since RB satisfies a termination property, every non-crashed process will eventually receive the broadcast messages.

Intuitively the algorithm works because coordinators always wait for a majority of messages before they proceed (which is why, to ensure the computation is non-blocking, strictly less than the majority are allowed to crash). Once a majority of processes have positively acknowledged in the same round, the coordinator's proposal of that round is said to be *locked*: if ever "after" another majority positively acknowledges, it will be for the very same value, thus satisfying Agreement. If some coordinator manages to get these acknowledgments and survives until RB-delivery, the algorithm also satisfies Termination. The interest in having a FD with $\Diamond\mathcal{S}$ is the guarantee (EWA) that eventually there will be a correct process that is never again suspected, thus will be positively acknowledged when playing the coordinator. $\Diamond\mathcal{S}$ also gives the guarantee (SC) that such a process will indeed be able to reach a round in which it plays the coordinator role. More detailed proofs of termination, validity, and agreement, are given in natural language and found in [CT96]. We found them reasonable, but hard to follow and very hard to formally verify.

Our first main criticism is that the pseudo-code does not have a formal semantics. Thus, there is no well-defined way to generate system runs, which are the base of the FD and Consensus properties. To tackle this problem, many years of research on concurrency theory provide us with a variety of decent formalisms that only need to be extended to also model failures and their detection.

Our second main criticism is more subtle. Some proofs of properties over runs make heavy reference to the concept of rounds, e.g., using induction on round numbers, although the relation between runs and asynchronous rounds is never clarified. This is problematic! Typically, such an induction starts with the smallest round in which some property X holds, e.g., in which a majority has positively acknowledged. In a given run, to find this starting point one may take the initial state and search from there for the first state in which X holds for some round. However, this procedure is not correct. It may well be that at a *later state* of the run, X holds for a *smaller round*! Accordingly, when the induction proceeds to a higher round, it might go backwards in time along a system run. Therefore, the concept of time—and of iteration along a run—is not fully compatible with the concept of asynchronous rounds. The solution, rather implicit in [CT96], is to consider runs as a whole, ignoring *when* events happen, just noting *that* they happened. In other words, we should pick a sufficiently advanced state of a given run (for example the last one in a finite run), and then find an appropriately abstract way to *reason about its possible past.* Summing up, the proofs would profit much from a global view on system states and their past that provides us with precise information about what processes *have been* in which round in the past, and what they precisely did when they were there.

Our Approach We provide a process calculus setting that faithfully captures the asynchronous process model. We equip this model with an operational control over crash-failure and FD properties (§2). However, instead of $\Diamond\mathcal{S}$, for which the algorithm was designed, we use the following FD [CHT96]:

Eventual Perpetual Uniform Trust (Ω) There is a time after which all the (correct) processes always trust the same correct process.

The FDs Ω and $\Diamond\mathcal{S}$ are equivalent in the sense that one can be used to implement the other, and vice versa. Although Ω was introduced only to simplify the minimality proofs of [CHT96], it turns out to be more natural to develop our operational model for it rather than for $\Diamond\mathcal{S}$. (Briefly, instead of keeping track of loads of useless unreliable suspicion information, Ω only requires to model small amounts of reliable trust information.) We then model the Consensus algorithm as a term in this calculus (§3), allowing us in principle to analyze its properties over runs generated by its local-view formal operational semantics. However, we do not do this as one might expect by iteration along system runs, showing the preservation of invariants. Instead, in order to formally deal with the round abstraction, we develop a global-view matrix-like representation of reachable states that contains the complete history of message sent "up to now" (§4). Also for this abstraction, we provide a formal semantics, and we use it instead of the local-view semantics to prove the Consensus properties (§5). The key justification for this approach is a very tight formal operational correspondence proof between the local-view process semantics and the global-view matrix semantics. It exploits slightly non-standard process calculus technology (see the full paper).

Contributions One novelty is the operational modeling of FD properties.

However, the essential novelty is the formal global-view matrix representation of the reachable states of a Consensus system that formally captures the round abstraction. It allowed us to bridge the gap between the local-view code and semantics describing the algorithm on the one hand, and the round-based reasoning that enables comprehensible structured proofs on the other hand.

Another contribution is that some proofs of [CT96], especially for Agreement, can now be considered as being formalized. Instead of trying to directly formalize Termination, we came up with different proof ideas for it (Theorem 2).

Conclusion The matrix semantics provides us with a tractable way to perform a formal analysis of this past, according to when and which messages have been sent in the various earlier rounds.

We use process calculus and operational semantics to justify proofs via global views that are based on the abstraction of rounds. In fact, this round-based global view of a system acts as a vehicle for many proofs about distributed algorithms, while to our knowledge it has never been formally justified and thus remained rather vague. Thus, we expect that our contribution will not only be valuable for this particular verification exercise, but also generally improve the understanding of distributed algorithms in asynchronous systems.

Related work We are only aware of a formal model and verification of *Randomized* Consensus using probabilistic I/O-automata [PSL00].

Future work Apart from this application-oriented work, we have also modeled the other failure detectors of [CT96]. We are currently working on the formal comparison of our representation to theirs. This work is independent of the language used to describe the algorithms that make use of failure detectors.

It would also be interesting to study extensions of our operational semantics setting for failure detectors towards more dynamic mobile systems.

Acknowledgments We thank Sergio Mena, André Schiper, Pawel Wojciechowski, and Rachid Guerraoui for discussions on the Consensus problem; James Leifer, Peter Sewell, Holger Hermanns for discussions on proof techniques; Daniel Bünzli and Aoife Hegarty for improving the presentation; and the anonymous reviewers.

2 The Process Calculus Model

We use a simple *distributed asynchronous value-passing* process calculus; name-passing is not needed for static process groups. We use an extension with *named sites* inspired by Berger and Honda [BH00], but unlike them we do not have to model message loss. Our notion of sites also resembles the locations of Dπ [RH01] and the Nomadic pi calculus [WS00]. For convenience, we also employ site-local purely signaling synchronous actions. We do not need the usual restriction operator, because we are going to study only internal transitions.

$$\begin{array}{rcl} v & ::= & x \;\big|\; i \;\big|\; \mathsf{t} \;\big|\; \mathsf{f} \;\big|\; \mathsf{f}(\tilde{v}) \;\big|\; \ldots \\ M & ::= & \overline{a}\langle\tilde{v}\rangle \\ \alpha & ::= & a(\tilde{x}) \;\big|\; \tau \;\big|\; \mathrm{susp}_j \;\big|\; a \;\big|\; \overline{a} \\ G & ::= & G{+}G \;\big|\; \alpha.P \;\big|\; \mathbf{0} \\ P & ::= & P|P \;\big|\; \mathsf{Y}\langle\tilde{v}\rangle \;\big|\; M \;\big|\; G \;\big|\; \mathsf{if}\ v\ \mathsf{then}\ P\ \mathsf{else}\ P \\ N & ::= & N|N \;\big|\; i[P] \;\big|\; M \end{array}$$

where process constants Y are associated with defining equations $\mathsf{Y}(\tilde{x}) := P$, which also gives us recursion. $\mathbf{I} \subseteq \mathbf{V}$ is a set of site identifiers (metavariables i, j, k, n), for which we simply take a subset of the natural numbers **Nat** equipped with standard operations like equality and modulo. $\{\mathsf{t}, \mathsf{f}\} \subseteq \mathbf{V}$ is the set of boolean values. The set $\mathbf{V}$ of value expressions (metavariable v) contains various operations on sets and lists, like addition, extraction, arity, and comparison. We also use a function eval that performs the deterministic evaluation of value expressions. By abuse of notation, we use all value metavariables (and x) also as input variables. Names $\mathbf{N}$ (metavariable a) are different from values ($\mathbf{N} \cap \mathbf{V} = \emptyset$).

We use $\mathbb{G}$, $\mathbb{P}$, and $\mathbb{N}$, to refer to the sets of terms generated by the respective non-terminal symbols for guards G, local processes P, and networks N. Sites $i[P]$ are named and may be syntactically distributed over terms; sometimes, we refer to them as *processes.* The interpretation of all operators is standard [BH00]. For actions susp_j, see the explanation and formal semantics later on. We include both synchronous signals $(a, \overline{a})$ and asynchronous messages M with matching receivers; for simplicity, we do not introduce separate syntactic categories for respective channels. As usual, parallel composition is associative and commutative; with finite indexing sets I we use $\prod_{i\in I} P_i$ as abbreviation for the arbitrarily ordered and nested parallel composition of the P_i, and similar for $\prod_{i\in I} N_i$.

Structural Equivalence The relation $\Lleftarrow\!\!\!\equiv\!\!\!\Rrightarrow$ is defined as the smallest equivalence relation generated by the laws of the commutative monoids $(\mathbb{G}, +, \mathbf{0})$, $(\mathbb{P}, |, \mathbf{0})$, and $(\mathbb{N}, |, \mathbf{0})$, the law $i[P_1] \,|\, i[P_2] \Lleftarrow\!\!\!\equiv\!\!\!\Rrightarrow i[P_1|P_2]$ that defines the scope of sites, the straightforward laws induced by evaluation of value expressions:

- $\mathsf{if}\ v\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2 \Lleftarrow\!\!\!\equiv\!\!\!\Rrightarrow P_1$ if $\mathrm{eval}(v) = \mathsf{t}$,
- $\mathsf{if}\ v\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2 \Lleftarrow\!\!\!\equiv\!\!\!\Rrightarrow P_2$ if $\mathrm{eval}(v) = \mathsf{f}$,
- $\overline{a}\langle\tilde{v}\rangle \Lleftarrow\!\!\!\equiv\!\!\!\Rrightarrow \overline{a}\langle\mathrm{eval}(\tilde{v})\rangle$, $\mathsf{Y}\langle\tilde{v}\rangle \Lleftarrow\!\!\!\equiv\!\!\!\Rrightarrow \mathsf{Y}\langle\mathrm{eval}(\tilde{v})\rangle$;
- $\mathsf{Y}\langle\tilde{v}\rangle \Lleftarrow\!\!\!\equiv\!\!\!\Rrightarrow P\{^{\tilde{v}}\!/\!_{\tilde{x}}\}$ if $\mathsf{Y}(\tilde{x}) := P$,

$$(\textsc{tau}) \quad i[\,\tau.P+G\,] \xrightarrow{\tau@i} i[\,P\,] \qquad (\textsc{suspect?}) \quad i[\,\mathrm{susp}_j\,.P+G\,] \xrightarrow{\mathrm{susp}_j@i} i[\,P\,]$$

$$(\textsc{com}) \quad i[\;\overline{a}.P_1+G_1 \mid a.P_2+G_2\;] \xrightarrow{\tau@i} i[\;P_1 \mid P_2\;]$$

$$(\textsc{snd}) \quad i[\,M\,] \xrightarrow{\tau@i} M \qquad (\textsc{rcv}) \quad \overline{a}\langle\tilde{v}\rangle \mid i[\,a(\tilde{x}).P+G\,] \xrightarrow{\tau@i} i[\,P\{{}^{\tilde{v}}\!/\!_{\tilde{x}}\}\,]$$

$$(\textsc{str}) \ \frac{N \Lleftarrow\!\!\Rrightarrow \hat{N} \qquad \hat{N} \xrightarrow{\mu@i} \hat{N}' \qquad \hat{N}' \Lleftarrow\!\!\Rrightarrow N'}{N \xrightarrow{\mu@i} N'} \qquad (\textsc{par}) \ \frac{N_1 \xrightarrow{\mu@i} N_1'}{N_1|N_2 \xrightarrow{\mu@i} N_1'|N_2}$$

Table 1. Network Transitions

$$(\textsc{trust}) \ \frac{i \notin \mathsf{TI} \cup \mathsf{C}}{(\mathsf{TI},\mathsf{C}) \ \rightarrow \ (\mathsf{TI} \cup \{i\},\mathsf{C})} \qquad (\textsc{crash}) \ \frac{i \notin \mathsf{TI} \cup \mathsf{C} \qquad |\mathsf{C}| \leq \lfloor \frac{n-1}{2} \rfloor}{(\mathsf{TI},\mathsf{C}) \ \rightarrow \ (\mathsf{TI},\mathsf{C} \cup \{i\})}$$

Table 2. Environment Transitions

and that is preserved within non-prefix contexts. The inclusion of conditional resolution and recursion unfolding within structural equivalence is to allow us to have the transition relation defined below to deal exclusively with interactions. However, an unconstrained use of $\Lleftarrow\!\!\Rrightarrow$ quickly leads to problems when applying equivalence laws in an unintended direction. Thus, for proofs, we replace the relation $\Lleftarrow\!\!\Rrightarrow$ and the rule (STR) of Table 1 with a directed (*normalized)* version.

Network Transitions Transitions on networks are generated by the laws in Table 1. Each transition $\mu@i$ is labeled by the action $\mu \in \{\tau, \mathrm{susp}_j\}$ and the site identifier i indicating the site required for the action. The communication of asynchronous messages takes two steps: once they are sent, i.e., appear at top-level on a site, they need to leave the sender site (SND) into the buffering "ether"; once in the ether, they may be received by a process on the target site (RCV).

Without definition (see the full paper for details), let $\xrightarrow{\mu@i}_{\mathrm{n}}$ denote the *normalized* transition relation that we get when using a directed structural relation; this relation is defined on the subset of *normalized* network terms $\mathbb{N}^{\mathrm{n}}$.

Environment Transitions By adding an environment component Γ to networks, we model both failures and their detection, as well as "trust" in the sense of Ω. Environments $\Gamma := (\mathsf{TI},\mathsf{C})$ contain (i) information about sites $i \in \mathsf{TI} \subseteq \mathbf{I}$ that have become *trusted* forever and *immortal*, so they can no longer be suspected nor crash, and (ii) information about sites $i \in \mathsf{C} \subseteq \mathbf{I}$ that have already crashed.

Environments are updated according to the rules in Table 2. Rule (TRUST) models the instant at which (according to Ω) processes become trusted—in our model they also become immortal: they will be "correct" in every possible future.

$$(\text{DETECT})\ \frac{\Gamma \ \rightarrow \Gamma'}{\Gamma \vdash N \ \rightarrow \Gamma' \vdash N} \qquad (\text{ACT})\ \frac{i \notin \mathsf{C} \qquad N \xrightarrow{\tau@i} N'}{(\mathsf{TI},\mathsf{C}) \vdash N \ \rightarrow (\mathsf{TI},\mathsf{C}) \vdash N'}$$

$$(\text{SUSPECT!})\ \frac{j \neq i \notin \mathsf{C} \qquad N \xrightarrow{\text{susp}_j@i} N' \qquad j \notin \mathsf{TI}}{(\mathsf{TI},\mathsf{C}) \vdash N \ \rightarrow (\mathsf{TI},\mathsf{C}) \vdash N'}$$

Table 3. System Transitions

Rule (CRASH) keeps track of processes that crash and is subject to an upper bound: for instance, the Consensus algorithm of [CT96] is supposed to work correctly only under the constraint that at most $\lfloor\frac{n-1}{2}\rfloor$ processes may crash.

System Transitions Configurations are pairs of the form $\Gamma \vdash N$. Their transitions come either from the environment Γ (DETECT), modeling the unconstrained occurrence of its transitions, or they come from the network N. In this case, the environment must explicitly permit the network actions. Rule (ACT) guarantees that only non-crashed sites may act. Rule (SUSPECT!) provides the model for suspicions: a site j may only be suspected by a process on another (different) non-crashed site i and—which is crucial—the *suspected site must not be trusted.* Note that suspicions in this model are "very unreliable" since every non-trusted site may be suspected from within any non-crashed site at any time.

Runs FD properties are based on the notion of *run.* In our language, runs are complete (in)finite sequences of transitions (denoted by $\rightarrow^*$) starting in some initial configuration $(\emptyset,\emptyset) \vdash N$. According to [CT96], a process is called *correct* in a given run, if it does not crash in that run. There is a close relation between this notion and the environment information in states of system runs.

Lemma 1 (Correctness in System Runs).

1. *If R is the run $(\emptyset,\emptyset)\vdash N_0 \ \rightarrow^* (\mathsf{TI},\mathsf{C})\vdash N \not\rightarrow$ then:*
   - *$i \in \mathsf{TI}$ iff i is correct in R; $i \in \mathsf{C}$ iff i is not correct in R.*
   - *$|\mathsf{TI}| \geq n-\lfloor\frac{n-1}{2}\rfloor$, $|\mathsf{C}| \leq \lfloor\frac{n-1}{2}\rfloor$, and $\mathsf{TI} \uplus \mathsf{C} = \{1..,n\}$.*
2. *If R is the run $(\emptyset,\emptyset)\vdash N_0 \ \rightarrow^* (\mathsf{TI},\mathsf{C})\vdash N \ \rightarrow^{*/\omega}$ then:*
   - *If $i \in \mathsf{TI}$, then i is correct in R. If $i \in \mathsf{C}$, then i is not correct in R.*
   - *$|\mathsf{TI}| \geq n-\lfloor\frac{n-1}{2}\rfloor$, $|\mathsf{C}| \leq \lfloor\frac{n-1}{2}\rfloor$, and $\mathsf{TI} \uplus \mathsf{C} \subseteq \{1..,n\}$.*

Proof (Sketch). By the rules of Table 2 and rule (SUSPECT!) in Table 3. □

For finite runs, Lemma 1(1) states that in final states all decisions concerning "life" and "death" are taken. For intermediate states of infinite runs, Lemma 1(2) provides us with only partial but nevertheless reliable information.

Our operational representation of the FD Ω consists of two parts: (i) the above rule (SUSPECT!), and (ii) a condition on runs that at least one site must eventually become trusted and immortal (for the current run) such that it cannot be suspected afterwards and will turn out to be correct.

Definition 1 (Ω-Runs). *Let R be a run starting in $(\emptyset, \emptyset) \vdash N_0$.*
*R is called Ω-*run *if $(\emptyset, \emptyset) \vdash N_0 \ \longrightarrow^* \ (\mathsf{TI}, \mathsf{C}) \vdash N$ is a prefix of R with $\mathsf{TI} \neq \emptyset$.*

The condition $\mathsf{TI} \neq \emptyset$ means that, for at least one transition in the run R, the rule (TRUST) must have been applied. In Ω-runs, it is sufficient to check a syntactic condition on states that guarantees the absence of subsequent unpermitted suspicions. In contrast, the original FD model requires to carefully check that after some hypothetical (not syntactically indicated) time all occurrences of suspicion steps do not address a particular process that happens to be correct in this run by analyzing every single step of the run. Thus, our operational FD model considerably simplifies the analysis of runs. The formal comparison of operational models and the original history-based models is ongoing work, in which we also address the remaining failure detector classes introduced in [CT96].

3 Solving Consensus with Ω-Detection

Table 4 shows the Consensus algorithm of [CT96] represented as the process calculus term $\mathit{Consensus}_{(v_1..,v_n)}$. When no confusion is possible, we may omit the initial values $(v_1.., v_n)$. We use the notation $\mathsf{Y}_i^{\tilde{v}}$ as an abbreviation for both $\mathsf{Y}_i(i, \tilde{v})$ and $\mathsf{Y}_i\langle i, \tilde{v}\rangle$, so the subscript is part of the constant while the superscripts represent formal/actual parameters. The subscript must, in fact, also be considered part of the parameters, because we will access it in the body, but since we never change this parameter, we omit it in the abbreviation.

Let n be the number of processes, and $\mathrm{crd}(r) := ((r{-}1) \bmod n){+}1$ denote the coordinator of round r. $\mathsf{Y}_i^{r,v,s,L}$ represents participant i in round r with current estimate v dating back to round s, and a list L of messages previously received from other participants (see below). Y_i itself ranges over $\mathsf{P1}_i, \mathsf{P2}_i, \mathsf{P4}_i, \mathsf{R}_i, \mathsf{Z}_i$ for $i = \mathrm{crd}(r)$, and over $\mathsf{P1}_i, \mathsf{P3}_i, \mathsf{R}_i$ for $i \neq \mathrm{crd}(r)$. D_i is part of the RB-protocol: it is the component that "decides" and re-broadcasts on RB-delivery.

All protocol participants are interconnected; we use separate channel names $(c1_i, c2_i, c3_i)$ for the messages sent in the first three phases, and further channel names for broadcasting (b_i) and announcing decisions (decide_i). For convenience, we use site-indexed channel names, but note that the indices i are only virtual: they are considered to be part of the indivisible channel name. In addition to these $5{*}n$ asynchronous channels, we use n synchronous channels $(\mathit{undecided}_i)$, also "indexed". We use the latter to conveniently avoid fairness conditions on runs concerning the reception of the otherwise asynchronous signals. We include some redundant information (gray-shaded in Table 4) within messages—especially about the sender and receiver identification—such that we can easily and uniquely distinguish messages. We also add some τ-steps, which are only there to facilitate the presentation of some of the proofs.

Behaviors In the 1st phase, we $(\mathsf{P1}_i^{r,v,s,L})$ send our current estimate and depending on whether we are coordinator of our round, we move to phase 2 or 3.

In the 2nd phase, we $(\mathsf{P2}_i^{r,v,s,L})$ wait for sufficiently many 3rd-phase estimate messages for our current round r. Once we have them, we determine the best

$$\mathit{Consensus}_{(v_1..,v_n)} \overset{\mathrm{def}}{=} \prod_{i=1}^{n} i\Big[\, \mathsf{P1}_i^{1,v_i,0,\emptyset} \,\big|\, \mathsf{D}_i \,\Big]$$

$$\mathsf{P1}_i^{r,v,s,L} \overset{\mathrm{def}}{=} \overline{c1_{\mathrm{crd}(r)}}\langle i,r,v,s\rangle \;\mid\; \textsf{if } i{=}\mathrm{crd}(r) \textsf{ then } \mathsf{P2}_i^{r,v,s,L} \textsf{ else } \mathsf{P3}_i^{r,v,s,L}$$

$$\begin{aligned}\mathsf{P2}_i^{r,v,s,L} \overset{\mathrm{def}}{=}\; & \textsf{if } |L_1^r| < \lceil \tfrac{n+1}{2} \rceil \\ & \textsf{then } c1_i\,(\tilde{x})\,.\,\mathsf{P2}_i^{r,v,s,(1,\tilde{x})::L} \\ & \textsf{else } \tau.\Big(\prod_{i\neq k=1}^{n} \overline{c2_k}\langle k,r,\mathrm{best}(L_1^r)\rangle \;\big|\; \mathsf{P4}_i^{r,\mathrm{best}(L_1^r),r,L} \Big)\end{aligned}$$

$$\begin{aligned}\mathsf{P3}_i^{r,v,s,L} \overset{\mathrm{def}}{=}\; & \textsf{if } L_2^r = \emptyset \\ & \textsf{then } \Big(c2_i\,(\tilde{x})\,.\,\mathsf{P3}_i^{r,v,s,(2,\tilde{x})::L} \;+\; \mathrm{susp}_{\mathrm{crd}(r)}\,.\,\big(\, \overline{c3_{\mathrm{crd}(r)}}\langle i,r,\mathsf{f}\rangle \mid \mathsf{R}_i^{r,v,s,L} \,\big)\Big) \\ & \textsf{else } \tau.\big(\, \overline{c3_{\mathrm{crd}(r)}}\langle i,r,\mathsf{t}\rangle \mid \mathsf{R}_i^{r,\mathrm{val}(L_2^r),r,L} \,\big)\end{aligned}$$

$$\begin{aligned}\mathsf{P4}_i^{r,v,s,L} \overset{\mathrm{def}}{=}\; & \textsf{if } |L_3^r| < \lceil \tfrac{n+1}{2} \rceil - 1 \\ & \textsf{then } c3_i\,(\tilde{x})\,.\,\mathsf{P4}_i^{r,v,s,(3,\tilde{x})::L} \\ & \textsf{else if } \bigwedge_{l\in L_3^r} \mathrm{bool}(l) \textsf{ then } \tau.\big(\prod_{k=1}^{n} \overline{b_k}\langle i,k,1,r,v\rangle \mid \mathsf{Z}_i^{r,v,s,L} \,\big) \textsf{ else } \mathsf{R}_i^{r,v,r,L}\end{aligned}$$

$$\mathsf{Z}_i^{r,v,s,L} \overset{\mathrm{def}}{=} \mathbf{0}$$

$$\mathsf{R}_i^{r,v,s,L} \overset{\mathrm{def}}{=} \mathit{undecided}_i\,.\,\mathsf{P1}_i^{r+1,v,s,L}$$

$$\mathsf{D}_i \overset{\mathrm{def}}{=} \overline{\mathit{undecided}_i}\,.\,\mathsf{D}_i \;+\; b_i\,(j,\cdot,m,r,v)\,.\,\Big(\overline{\mathit{decide}_i}\langle j,i,m,r,v\rangle \;\big|\; \prod_{k=1}^{n} \overline{b_k}\langle i,k,2,r,v\rangle \Big)$$

Table 4. Consensus

one among them (see below), and we impose *its* value as the one to adopt in the round r by sending it to everybody else. (As a slight optimization of [CT96], we do not send the proposal to ourselves, and also we do not send an acknowledgment to ourselves, assuming that we agree with our own proposal.) Remembering the just proposed value, we then move to phase 4.

In the 3rd phase, we ($\mathsf{P3}_i^{r,v,s,L}$) are waiting for the proposal from the coordinator of our current round r. As soon as it arrives, we positively acknowledge it, and (try to) restart and move to the next round. As long as it has not yet arrived, we may also have the possibility to suspect the coordinator in order to move on; in this case, we continue with our old value and stamp.

In the 4th phase, we ($\mathsf{P4}_i^{r,v,s,L}$) wait for sufficiently many 3rd-phase acknowledgment messages for our current round r. Once we have them, we check whether they are all positive. If yes, then we launch reliable broadcast by sending our decision value v on all b_k; it becomes reliable only through the definition of D_k on the receiver side of the b_k. If no, then we simply try to restart.

If we ($\mathsf{R}_i^{r,v,s,L}$) want to restart, we must get the explicit permission from our broadcast controller process D_i along the local synchronous channel $\mathit{undecided}_i$. This permission will never again be given as soon as we (at site i) have "delivered", i.e., received the broadcast along b_i and subsequently have decided.

When halting a coordinator, we do not just let it become $\mathbf{0}$ or disappear, but use a specific constant Z_i to denote the final state. The reason is that we can keep accessibly within the term the final information of halted processes ($\mathsf{Z}_i^{r,v,s,L}$), which would otherwise disappear as well.

Data Structures The parameter $L{\in}\mathbb{L}$ is a heterogeneous list of elements in $\mathbb{L}_1$ for 1st-phase messages, $\mathbb{L}_2$ for 2nd-phase messages, and $\mathbb{L}_3$ for 3rd-phase messages. By L_1, L_2, L_3, we denote the various homogeneous sublists of L for the corresponding phases. By $|L|$, we denote the length of a list L. By $l{::}L$, we denote the addition of element l to L. For each homogeneous type of sublist, we provide some more notation. For convenience, we allow ourselves to use component access via "logical" names rather than "physical" projections. For example, in all types, one component represents a round number. By $L^r := \{\, l{\in}L \mid \mathrm{round}(l){=}r \,\}$, we extract all elements of list L that apparently belong to round r. Similarly, the function $\mathrm{val}(l)$ extracts the value field of list element l.

Elements of $\mathbb{L}_1$ ($\{1\}{\times}I{\times}\mathbb{N}{\times}\mathbf{V}{\times}\mathbb{N}$), like 1st-phase messages, consist of a site identifier ($\in I$), a round number ($\in\mathbb{N}$), an estimate value ($\in\mathbf{V}$), and a stamp ($\in\mathbb{N}$). Let $L{\in}\mathbb{L}_1^*$. By $\mathrm{max\_s}(L) := \max\{\,\mathrm{stamp}(l) \mid l{\in}L\,\}$ we extract the maximal stamp occurring in the elements of L. By $\mathrm{best}(L) := \mathrm{val}(\mathrm{min\_i}\{\, l{\in}L \mid \mathrm{stamp}(l){=}\mathrm{max\_s}(L)\,\})$, we extract among all the elements of L that have the highest stamp the one element that is smallest with respect to the site identifier, and return the value of it.

Elements of $\mathbb{L}_2$ ($\{2\}{\times}I{\times}\mathbb{N}{\times}\mathbf{V}$), like 2nd-phase messages, consist of a site identifier ($\in I$), a round number ($\in\mathbb{N}$), and an estimated value ($\in\mathbf{V}$).

Elements of $\mathbb{L}_3$ ($\{3\}{\times}I{\times}\mathbb{N}{\times}\mathbb{B}$), like 3rd-phase messages, consist of a sender site identifier ($\in I$), a round number ($\in\mathbb{N}$), and a boolean value ($\in\mathbb{B}$). Let $l{\in}\mathbb{L}_3$. By $\mathrm{bool}(l)$, we extract the boolean component of list element l.

4 A Global Message-Oriented View: Matrices

By analysis of Chandra and Toueg's proofs of the Consensus properties [CT96], we observe that they become feasible *only* if we manage to argue formally and globally about the contributions of processes to individual rounds. To this aim, we design an alternative representation of the reachable state of *Consensus*: message *matrices* $\mathbb{M}$. In fact, matrices contains precisely the same information as terms: we can freely move between the two representations via formal mappings:

$$\begin{tikzcd}\mathbb{N} \arrow[r, shift left, "\mathcal{M}[\![\;]\!]\ \text{using}\ \mathcal{E}_t()"] & \mathbb{M} \arrow[l, shift left, "\mathcal{N}[\![\;]\!]\ \text{using}\ \mathcal{E}^{-1}()"]\end{tikzcd}$$

It is for this tight connection that we augmented the definition of *Consensus* in Table 4 with book-keeping data, never forgetting any message ever received.

$\mathcal{M}[\![\;]\!]$: From Networks to Matrices With any state reachable starting from *Consensus*, we associate a matrix structure containing all the asynchronous messages

$M := \mathcal{E}_{\mathrm{M}}^{-1}(\mathbf{x})$	$l := \mathcal{E}_{\mathrm{L}}^{-1}(\mathbf{x})$	snd	rcv	rnd	$\mathcal{E}_t(M) =: \mathbf{x} := \mathcal{E}_t(l)$	tag($\mathbf{x}$)
$\overline{c1_{\mathrm{crd}(r)}}\langle i,r,v,s\rangle$	$(1,i,r,v,s)$	i	crd(r)	r	$(i,r)\overset{1}{\mapsto}(v,s,t)$	t
$\overline{c2_i}\langle i,r,v\rangle$	$(2,i,r,v)$	crd(r)	i	r	$(i,r)\overset{2}{\mapsto}(v,t)$	t
$\overline{c3_{\mathrm{crd}(r)}}\langle i,r,z\rangle$	$(3,i,r,z)$	i	crd(r)	r	$(i,r)\overset{3}{\mapsto}(z,t)$	t
$\overline{b_i}\langle j,i,m,r,v\rangle$		j	i	r	$(i,j,m)\overset{\mathrm{b}}{\mapsto}(r,v,t)$	t
$\overline{decide_i}\langle j,i,m,r,v\rangle$		i	–	r	$(i)\overset{\mathrm{d}}{\mapsto}(j,m,r,v,t)$	t

Table 5. From Messages M to Matrix Entries $\mathbf{x}$ ... and back

that have been sent "up to now", organized according to the round in which they were sent. For the 1st-, 2nd-, and 3rd-phase messages, the resulting structure is a specific kind of two-dimensional matrix (see column six of Table 5): one dimension for *process ids* (variable i ranging from 1 to n), one dimension for *round numbers* (variable r ranging unboundedly over natural numbers starting at 1). For broadcast- and decision-messages, which may only eventually occur for a single round per process, the format is slightly different.

For each message, we distinguish three *transmission states*:

- being *sent*, but not yet having left the sender site ($\surd$)
- being *in transit*, i.e., having left the sender site, but not yet arrived ($\surd\!\!\surd$)
- being *received*, i.e., appearing in the list L ($\surd\!\!\surd\!\!\surd$)

We usually let t range over the elements of the ordered set $\{\surd < \surd\!\!\surd < \surd\!\!\surd\!\!\surd\}$. For d-entries, aka: decision messages, there is no receiver and thus always $t \neq \surd\!\!\surd\!\!\surd$.

Networks can be mapped into matrices because our process representation memorizes the required information on past messages ($\surd\!\!\surd\!\!\surd$) in the state parameters L; messages that are sent and not yet received ($\surd, \surd\!\!\surd$) can be analyzed "directly" from the respective system state component. Table 5 lists the various entry types of matrices, and how they correspond to the formats found in networks, namely messages M and list entries $l{\in}L$. For better orientation, we include columns snd and rcv that indicate the respective sender and receiver.

We may view a matrix $\mathfrak{M}$ as the heterogeneous superposition of five homogeneous parts. Each part can be regarded either as a set of elements $\mathfrak{M}.\mathbf{x}$ as in column six of Table 5, or as a function according to the domain of x, ranging over $\{\mathfrak{M}.1_i^r, \mathfrak{M}.2_i^r, \mathfrak{M}.3_i^r, \mathfrak{M}.\mathrm{b}_{ij}^m, \mathfrak{M}.\mathrm{d}_i\}$; we use $\top$ and $\bot$ to denote defined and undefined images. Matrix update $\mathfrak{M}\{\mathrm{x} := \tilde{v}\}$ is overriding.

Matrix Semantics The initial matrix of *Consensus* is denoted by

$$\mathfrak{Consensus}_{(v_1..,v_n)} := \mathcal{M}[\![\, Consensus_{(v_1..,v_n)} \,]\!] = \emptyset\{\, \forall i : 1_i^1 := (v_i, 0, \surd)\,\}$$

In order to simulate the behavior of networks at the level of matrices, we propose an operational semantics that manipulates matrices precisely mimicking the behavior of their corresponding networks. As with networks, the rules in Tables 2

Rounds		Processes				
Number	**Phase**	**1**	**2**	**3**	**4**	**5**
1	1	$(v_1,0,\surd)$	$(v_2,0,\surd\!\surd)$	$(v_3,0,\surd\!\surd\!\surd)$	$(v_4,0,\surd\!\surd\!\surd)$	$(v_5,0,\surd\!\surd\!\surd)$
	2	—	$(v_3,\surd)$	$(v_3,\surd)$	$(v_3,\surd\!\surd)$	$(v_3,\surd\!\surd\!\surd)$
	3	—		$(f,\surd\!\surd\!\surd)$	$(f,\surd\!\surd\!\surd)$	$(t,\surd\!\surd)$
2	1	$(v_3,1,\surd)$		$(v_3,0,\surd\!\surd)$	$(v_4,0,\surd)$	$(v_3,1,\surd\!\surd)$
	2		—			
	3		—	$(f,\surd\!\surd)$	$(f,\surd)$	$(f,\surd\!\surd)$
3	1			$(v_3,0,\surd\!\surd\!\surd)$	$(v_4,0,\surd\!\surd\!\surd)$	$(v_3,1,\surd\!\surd\!\surd)$
	2	$(v_3,\surd\!\surd)$	$(v_3,\surd)$	—	$(v_3,\surd\!\surd)$	$(v_3,\surd\!\surd\!\surd)$
	3			—	$(f,\surd\!\surd\!\surd)$	$(f,\surd\!\surd\!\surd)$
4	1			$(v_3,3,\surd)$	$(v_4,0,\surd\!\surd\!\surd)$	$(v_3,1,\surd\!\surd)$
	2				—	
	3			$(f,\surd)$	—	

Table 6. Example Matrix

and 3, where networks and their transitions are replaced by matrices and their (equally labeled) transitions, allow us to completely separate the treatment of behavior in the context of crashes from the description of the behavior of messages in the matrix. The rules are given in the full paper. Here, we just look at an example of a matrix for $n = 5$ (Table 6) that is reachable by using the matrix semantics. For instance, to be a valid matrix, coordinators can only have proceeded to the next round if they received ($\surd\!\surd\!\surd$) a majority−1 of 3rd-phase messages; c.f. the coordinators of rounds 1 and 3. Also, participants proceed with the value of the previous round if they *nack* (f), or with the proposed value of the previous coordinator if they *ack* (t); c.f. process 5 in its rounds 2 and 4.

Some transitions that are enabled from within the example matrix are: messages with tag $\surd$ may be released to the network and get tag $\surd\!\surd$; process 4 may receive 1st-phase messages from process 5, from either round 2 or 4. Many other requirements like these are represented by the 12 rules of the matrix semantics.

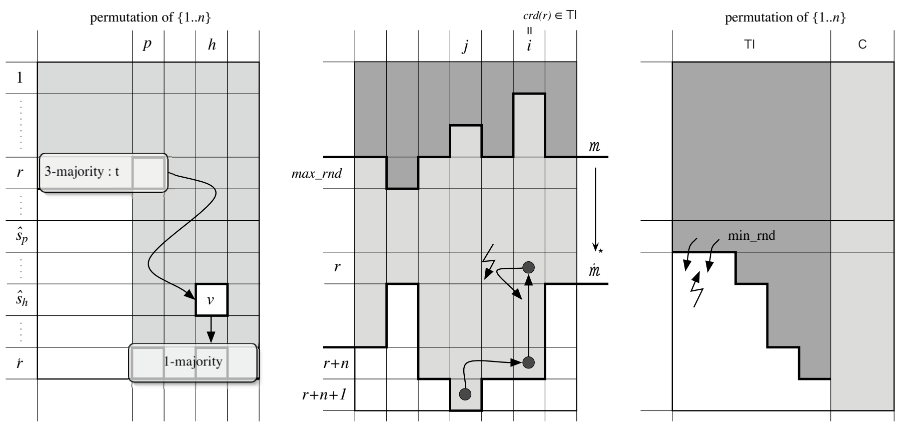
$\mathcal{N}[\![\ ]\!]$: From Matrices to Networks We only note here that the presence of all previously sent messages, distinguishing all their transmission states, allows us to uniquely reconstruct the term counterpart, i.e., for every site i we may uniquely determine its phase $\mathrm{phs}_i(\mathfrak{M}){:=}\mathsf{Y}_i \in \{\, \mathsf{P1}_i, \mathsf{P2}_i, \mathsf{P3}_i, \mathsf{P4}_i, \mathsf{R}_i, \mathsf{Z}_i \,\}$ with accompanying parameters $r{:=}\,\mathrm{rnd}_i(\mathfrak{M}), v, s, L$ and its decision state $\mathrm{dec}_i(\mathfrak{M}) \in \{\, \mathsf{D}_i, \mathbf{0} \,\}$.

The matrix semantics mimics the network semantics very closely.

Proposition 1 (Operational Correspondence). *Let Consensus*$^{\mathrm{n}} \longrightarrow_{\mathrm{n}}^{*} N$.

1. *If* $N \xrightarrow{\mu@i}_{\mathrm{n}} N'$, *then* $\mathcal{M}[\![\, N \,]\!] \xrightarrow{\mu@i} \mathcal{M}[\![\, N' \,]\!]$.
2. *If* $\mathcal{M}[\![\, N \,]\!] \xrightarrow{\mu@i} \mathfrak{M}$, *then* $N \xrightarrow{\mu@i}_{\mathrm{n}} \Lleftarrow\!\!\!\!\!\equiv \mathcal{N}[\![\, \mathfrak{M} \,]\!]$.

Normalized network runs can then straightforwardly be translated step-by-step into matrix runs using $\mathcal{M}[\![\ ]\!]$, and vice versa using $\mathcal{N}[\![\ ]\!]$. If a network run is infinite, then its corresponding matrix run is infinite as well. Or, conversely, if a corresponding matrix run is finite, then the original network run must have



(I) Pre-Agreement (II) Ω-Finiteness (III) Termination

Table 7. Matrix Proofs

been finite as well. Furthermore, since we produce system runs—where the distributed algorithm is embedded into our failure-sensitive environments—with either networks or matrices, the correspondence carries over also to the system level. Therefore, we may use the matrix semantics instead of the original network semantics to reason about the Consensus algorithm and its properties.

5 Properties of the Algorithm: Consensus

In this section, we prove the three required Consensus properties—validity, agreement, and termination—using the matrix structures. As the graphical sketches in Table 7 show, we heavily exploit the fact that the matrix abstraction allows us to analyze message patterns that have been sent in the past. We do not need to know precisely in which order all the messages have been sent, but we do need to have some information about the order in which they *cannot* have been sent. Our formal matrix semantics provides us with precisely this kind of information.

We conclude this section by transferring the results back to networks.

Validity From the definition, every decided value has initially been proposed.

Proposition 2 (Validity). *Let* $\mathfrak{Consensus}_{(v_1..,v_n)} \rightarrow^* \mathfrak{M}$.
If $\mathfrak{M}.\mathrm{d}_i = (j, m, r, v, t)$*, then there is* $k \in \{\, 1..,n \,\}$ *with* $v = v_k$.

Agreement We call $\mathrm{val}^r(\mathfrak{M})$ the value that the coordinator of round r in $\mathfrak{M}$ tried to impose in its second phase; it may be undefined. In the Introduction, we said that a value gets *locked* as soon as enough processes have, in the same round, positively acknowledged to the coordinator of this round. This condition translates into matrix terminology, as follows:

Definition 2. *A value v is called* locked *for round r in matrix $\mathfrak{M}$, written $\mathfrak{M} \overset{r}{\mapsto} v$, if $\#\{\, j \mid \mathfrak{M}.3^r_j = (\mathsf{t}, \cdot) \,\} \geq \lceil \frac{n+1}{2} \rceil - 1$.*

Note the convenience of the matrix abstraction to access the messages that were sent in the past, without having to look at the run leading to the current state. Now, if $\mathfrak{M} \overset{r}{\mapsto} v$ then $v = \mathrm{val}^r(\mathfrak{M})$. Also, broadcast is always for a locked value.

Lemma 2. *If $\mathfrak{M}.\mathrm{b}^m_{ij} = (r, v, \cdot)$, then $\mathfrak{M} \overset{r}{\mapsto} v$.*

Lemma 3. *If $\mathfrak{M} \overset{r}{\mapsto} v_1$ and $\mathfrak{M} \overset{r}{\mapsto} v_2$, then $v_1 = v_2$.*

The key idea is to compare lockings in two different rounds.

Proposition 3 (Pre-Agreement). *If $\mathfrak{M} \overset{r_1}{\mapsto} v_1$ and $\mathfrak{M} \overset{r_2}{\mapsto} v_2$, then $v_1 = v_2$.*

Note that both lockings have already happened in the past of $\mathfrak{M}$.

Proof (Sketch). Suppose that $\mathfrak{M} \overset{r}{\mapsto} v$, so $v = \mathrm{val}^r(\mathfrak{M})$. We prove by course-of-value induction that for all $\hat{r} > r$, if $\mathfrak{M} \overset{\hat{r}}{\mapsto} \hat{v}$, then $v = \hat{v}$.

First, in both rounds r and $\hat{r}$, a majority is responsible for the locking. In Table 7(I), we make explicit (by permutation) that there is a process p that belongs to both majorities. Then, let h be the process that won the first phase of round $\hat{r}$ in that $\mathrm{crd}(\hat{r})$ chose h's estimate as its round-proposal. Using the matrix semantics, we identify the rounds $\hat{s}_p$ and $\hat{s}_h$, in which p and h acknowledged the estimate that they still believe in at round $\hat{r}$. By a number of of auxiliary lemmas on matrices we conclude that $r \leq \hat{s}_p \leq \hat{s}_h < \hat{r}$.

Now, if $r = \hat{r}$, then trivially $\mathrm{val}^r(\mathfrak{M}) = \mathrm{val}^{\hat{r}}(\mathfrak{M})$ (Lemma 3).

If $\hat{r} > r$ then, by induction, we have $\mathrm{val}^r(\mathfrak{M}) = \mathrm{val}^{\hat{s}_h}(\mathfrak{M})$, and since h preserves the value it adopted in $\hat{s}_h$ until it reaches $\hat{r}$, where it "wins", also $\mathrm{val}^{\hat{s}_h}(\mathfrak{M}) = v = \mathrm{val}^{\hat{r}}(\mathfrak{M})$, we conclude $\mathrm{val}^r(\mathfrak{M}) = \mathrm{val}^{\hat{r}}(\mathfrak{M})$. □

Theorem 1 (Agreement). *If $\mathfrak{M}.\mathrm{d}_i = (\cdot, \cdot, \cdot, v_i, \cdot)$ and $\mathfrak{M}.\mathrm{d}_j = (\cdot, \cdot, \cdot, v_j, \cdot)$, then $v_i = v_j$.*

Proof (Sketch). If $\mathfrak{M}.\mathrm{d}_i = (k_i, m_i, r_i, v_i, \cdot)$, then by the only matrix rule to generate d_i-entries there must (have) be(en) r_i with $\mathfrak{M}.\mathrm{b}^{m_i}_{ik_i} = (r_i, v_i, \cdot)$. Analogously for j: if $\mathfrak{M}.\mathrm{d}_j = (k_j, m_j, r_j, v_j, \cdot)$, there must (have) be(en) r_j with $\mathfrak{M}.\mathrm{b}^{m_j}_{jk_j} = (r_j, {}_j\, , \cdot)$. By Lemma 2, both $\mathfrak{M} \overset{r_i}{\mapsto} v_i$ and $\mathfrak{M} \overset{r_j}{\mapsto} v_j$. By Proposition 3, we conclude $v_i = v_j$. □

Termination In an infinite run, every round is reached.

Lemma 4 (Infinity). *Let R denote an infinite system run of $\mathfrak{Consensus}$. Then, for all $r > 0$, there is a prefix of R of the form*

$$(\emptyset, \emptyset) \vdash \mathfrak{Consensus} \ \longrightarrow^* \ \Gamma \vdash \mathfrak{M}$$

where $\mathfrak{M}.1^r_i = \top$ for some i.

Proof (Sketch). By combinatorics on the number of steps per round. □

Theorem 2 (Ω-Finiteness). *All Ω-runs of $\mathfrak{Consensus}$ are finite.*

Proof (Sketch). Assume, by contradiction, to have an infinite Ω-run. The bold line $\mathfrak{M}$ in Table 7(II), marks the global state at instant t, when process i becomes $\in \mathsf{TI}$. Call max_rnd the greatest round at time t, and $r > max\_rnd$ the first round in which $i = \mathrm{crd}(r)$. Since the run is infinite, with Lemma 4 there is a time $\hat{t} > t$, where we reach state $\hat{\mathfrak{M}}$, where round $r{+}n{+}1$ is populated by some j. Since $i \in \mathsf{TI}$, j can reach $r{+}n{+}1$ only by positively acknowledging i in round $r{+}n$. So i was in $r{+}n$, therefore also in r. Since i was in r and has gone further, it has been suspected. But here we get a contradiction because in round r already $i \in \mathsf{TI}$ and no process was allowed to suspect it, while the matrix $\mathfrak{M}$ evolved into $\hat{\mathfrak{M}}$. So, no Ω-run can be infinite. □

Theorem 3 (Termination). *All Ω-runs of $\mathfrak{Consensus}$ are of the form*

$$(\emptyset,\emptyset)\vdash \mathfrak{Consensus} \ \rightarrow^* \ (\mathsf{TI},\mathsf{C})\vdash \mathfrak{M} \not\rightarrow$$

with $\mathsf{TI} \uplus \mathsf{C} = \{\,1..,n\,\}$ *and* $i \in \mathsf{TI} \neq \emptyset$ *implies that* $\mathfrak{M}.\mathrm{d}_i = \top$.

Proof (Sketch). We first show that if there was $i \in \mathsf{TI}$ with $\mathfrak{M}.\mathrm{d}_i = \bot$, then actually $\mathfrak{M}.\mathrm{d}_j = \bot$ for all $j \in \mathsf{TI}$. Since $\mathfrak{M} \not\rightarrow$, we may thus call all processes in $j \in \mathsf{TI}$ as being in deadlock. Then, we proceed by contradiction. We concentrate on the non-empty set $Min \subseteq \mathsf{TI}$ of processes in the currently minimal round. The contradiction arises, as in Table 7(III), by using the matrix semantics to show that Min must be empty, otherwise contradicting that $\mathfrak{M} \not\rightarrow$. □

Back to the Process Calculus With Table 5, we observe that the definedness of an entry $\mathfrak{M}.\mathrm{d}_i = \top$ corresponds to a message $\overline{decide_i}\langle\cdots\rangle$ having been sent. Therefore, and with the operational correspondence (Proposition 1), which closely resembles strong bisimulation, all the previous results carry over to networks.

References

[BH00] M. Berger and K. Honda. The Two-Phase Commitment Protocol in an Extended pi-Calculus. In L. Aceto and B. Victor, eds, *Proceedings of EXPRESS '00*, volume 39.1 of *ENTCS*. Elsevier Science Publishers, 2000.

[CHT96] T. D. Chandra, V. Hadzilacos and S. Toueg. The Weakest Failure Detector for Solving Consensus. *Journal of the ACM*, 43(4):685–722, 1996.

[CT96] T. D. Chandra and S. Toueg. Unreliable Failure Detectors for Reliable Distributed Systems. *Journal of the ACM*, 43(2):225–267, 1996.

[FLP85] M. J. Fisher, N. Lynch and M. Patterson. Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM*, 32(2):374–382, 1985.

[PSL00] A. Pogosyants, R. Segala and N. Lynch. Verification of the Randomized Consensus Algorithm of Aspnes and Herlihy: a Case Study. *Distributed Computing*, 13(3):155–186, 2000.

[RH01] J. Riely and M. Hennessy. Distributed Processes and Location Failures. *Theoretical Computer Science*, 226:693–735, 2001.

[WS00] P. Wojciechowski and P. Sewell. Nomadic Pict: Language and Infrastructure Design for Mobile Agents. *IEEE Concurrency*, 8(2):42–52, 2000.