States in Process Calculi

Christoph Wagner   Uwe Nestmann

Technische Universität Berlin, Germany

{christoph.wagner|uwe.nestmann}@tu-berlin.de

Formal reasoning about distributed algorithms (like Consensus) typically requires to analyze global states in a traditional state-based style. This is in contrast to the traditional action-based reasoning of process calculi. Nevertheless, we use domain-specific variants of the latter, as they are convenient modeling languages in which the local code of processes can be programmed explicitly, with the local state information usually managed via parameter lists of process constants. However, domain-specific process calculi are often equipped with (unlabeled) reduction semantics, building upon a rich and convenient notion of structural congruence. Unfortunately, the price for this convenience is that the analysis is cumbersome: the set of reachable states is modulo structural congruence, and the processes' state information is very hard to identify. We extract from congruence classes of reachable states individual state-informative representatives that we supply with a proper formal semantics. As a result, we can now freely switch between the process calculus terms and their representatives, and we can use the stateful representatives to perform assertional reasoning on process calculus models.

1   Introduction

Many articles have been written about the pros and cons of using either action-based (aka: behavioral) or state-based (aka: assertional) formalism for the purpose of verification. This paper can be added to the list of these papers, but here we will not argue in favor of one of the two kinds [3], and we will also not design a new formalism to integrate both styles [6]. We rather deliver support when using an action-based formalism—here: process calculi—to verify the correctness of distributed algorithms, where the reasoning is traditionally rather state-based [11]. In this context, it is instructive to look at Lamport's case study [10], where he outlines that assertional reasoning not only applies to sequential and even concurrent algorithms, but also to distributed algorithms [11]. Several of his remarks still hold just like 30 years ago, in particular concerning the requirements and aspects of the assertional method: 1. The communication medium needs to be represented explicitly, ideally itself as a process. 2. The concept of a global state should be explicitly available. 3. The method involves reasoning about the entire network of processes at once.

We try to profit as much as possible from both worlds. From process calculi, we take the advantage of exploiting formally defined semantics for a tiny modeling language that is close to a reasonable programming language. From distributed computing, we take the assertional proof methods that are based on the availability of complete global state information to verify the correctness of global invariants. We study the possibility of putting them together such that the assertional proof techniques are explicitly tied to the process calculus code.

A domain-specific process calculus for the particular class of fault-tolerant distributed consensus algorithms should be equipped with (a) an explicit notion of distribution, node failure and failure-detection, (b) the underlying communication medium in form of respective primitives, (c) a value language comprising expressions (constructors and selectors) for the required base types and containers (like lists), and (d) some control-flow commands like conditionals.

Next to the syntax, we also need to supply adequate semantics such as rules to evaluate conditionals, rules to compute the values of expressions, rules to check for the current state of a failure detector so that we know whether we should rather wait a bit longer for a process to react, or just proceed. For some or most of these enhancements, we can typically choose whether we want their behavior be modeled as explicit steps in a transition system, or whether we hide it in respective rules of a structural congruence relation so that they do not "cost" a transition step; we then speak of structural evaluation. There are advantages for both. In essence, as we focus on the interplay between independent distributed components, we most often find the guideline that local computation—which does not change the local state—is hidden within the structural congruence, while interaction with the other components via the interprocess communication or some global failure detection device is modeled as transitions.

With the calculus at hand, one typically uses process constants to define the behavior at the various program points; in a sense, these constants act as labels that we can use to implement recursive behavior. For example, we will later on write an algorithm that starts in the state

$$\prod_{i=1}^{n} i\left[\mathsf{B}(v_1,\ldots,v_n)\right]$$

where the notation $i[-]$ indicates that the included process description is running at location i, with the behavior B being a constant that is here invoked with its initial set of parameters $(v_1,\ldots,v_n)$. This hints at a crucial underlying idea: *The modeling philosophy, when using a process calculus for this type of algorithm, is such that we simulate the local state of a process via the list of parameters when invoking a constant that captures its behavior during the intended phase of the algorithm.* At a later phase, process i is going to be reinstated in a different phase, with a different list of parameters, for example $\mathsf{B}(v_1',\ldots,v_n')$.

This simulation of local state information becomes problematic deriving its transition system. The substitution of a process constant by its defining expression (just like a procedure call) takes place within the laws of a structural congruence relation $\equiv$ as part of the rule

$$\frac{C \equiv C_1 \longrightarrow C_2 \equiv C'}{C \longrightarrow C'}$$

As a result, the next reachable configuration C' may be in a shape where we do no longer see the parameter lists of local processes. We might then argue that procedure calls should not be handled by the structural congruence, but rather at the cost of a transition step. However, then we would still lose the structure immediately. We may instead require that after every step, we *must* end up in a form consisting of just process constants. This could be enforced syntactically, or imposed as a modeling discipline. But then another problem occurs when the behavior of such a process first does some local computation, like evaluating a conditional (without changing its local state!), and depending on this evaluation proceeds with behaviors again conveniently described by other constants, which also do only carry out local computations. Where should the structural evaluation stop?

Our approach to tackle this dilemma is based on a guideline to identify process constants that are essential to understand the local state of a process. This requires an intuitively deep understanding of the algorithm, just like when one is looking for invariants. Referring to the character of processes as reactive components (sometimes also called event handlers), we should identify the moments when processes wait for messages to arrive. Thus structural evaluation should proceed to the point where such a moment is reached and then stop with the process constant, where such a message arrival is immediately expected. A similar moment of waiting is reached when we require to detect a failure before computation may proceed.

Using this observation, we then design a *standard form* for the global state of the algorithm, in which all and only the possible "waiting states" are listed by means of their respective process constants—including their explicit lists of parameters representing the local state information—together with the parallel composition of messages that are underway, i.e. sent but not yet received, possibly listed according to their type. So, a standard form for an algorithm with two possible waiting states $(\mathsf{W}_1, \mathsf{W}_2)$ per process may be, as a first approximation, represented as

$$\left(\prod_{i \in I_1} \mathsf{W}_1(v_1^1, \ldots, v_n^1) \mid \prod_{i \in I_2} \mathsf{W}_2(v_1^2, \ldots, v_n^2) \right) \mid M$$

with $\{I_1, I_2\}$ being a partition of the set of process indices with I_k containing the indices of those processes that are currently in state W_k, and M being the composition of current messages. To make such forms unique, we may have to impose an order on the listing of the various components and message types. Note that this idea is, in a very simple form, already present in Milner's Scheduler example, as found in [13].

We could now try to recover a standard form after each computation step, modulo structural congruence. (Milner preferred to work with transitions up to strong bisimilarity.) Instead of trying to guide structural evaluation in order to precisely hit such standard forms, we prefer to extract all the parameters of such standard forms and cast them into a new mathematical structure, basically a sizable tuple that we then use as our substitute representation of the global state. We devise a mapping that extracts exactly one representative of such a structure from each structural congruence class that is reachable when running the algorithm. The expansion of these representatives back to a process term will then precisely provide the intended standard forms. As the representatives in the new structure carry the complete state information, we define an explicit operational semantics that produces a new representative after each transition, precisely mimicking the transitions of the expanded process term. In fact, we design it as a 1-1-correspondence, just like a strong bisimulation between process terms and their stateful representatives. The goal is then clear. For any assertional proof of properties of the algorithm, we use the stateful representatives instead of the process terms from which we initially extracted them.

Contributions   Previously [9], we presented a rather detailed analysis of a Consensus algorithm known from the literature [1]. There (and in the related Phd thesis by Kühnrich [8]), we already introduced the idea of standard forms, but they were not uniquely defined, and we did not design an explicit operational semantics for them. Instead, we focused on the overall proof methodology that integrated the assertional invariant-based reasoning with the definition of a bisimulation relation for the overall correctness proof. In contrast, in this paper, while we revisit the same algorithm, the new contributions are the precise definition of the standard forms, their uniqueness, the definition of an explicit operational semantics for their stateful representatives, and the proof of 1-1-correspondence with the underlying process calculus semantics. In the Appendix, we supply some proofs on the correctness of the chosen algorithm, now carried out based on the explicit semantics of the stateful representatives. While the standard forms need to be reinvented for each algorithm, the principles for finding them and also the overall methodology are reusable.

2   Distributed Consensus

In this chapter we introduce a distributed process calculus for fault tolerant systems. Fault tolerant systems are systems that are able to proceed their work even in the case of failures. In every fault tolerant

system there is a limit to how many failures can occur until the system ceases its operation. The limit is directly linked to the design of the system and the nature of the failure. Failures can occur in every part of the system, e. g. message transfer, on channels, or processes. We restrict our interest to permanent crashes of processes. To detect crashes we need a tool, which we denote as failure detector. We concentrate on unreliable failure detectors as described in [2].

2.1 Unreliable Failure Detectors

The following ideas and definitions are taken mostly verbatim from [2].

A distributed system consists of a set of n processes, $\Pi = \{p_1, \ldots, p_n\}$. Every pair of processes is connected by a reliable communication channel. For simplicity, the existence of a discrete global clock is assumed, with the range $\mathscr{T}$ of the clock's ticks being the set of natural numbers. The function $F : \mathscr{T} \to 2^\Pi$ describes a *failure pattern*, while $F(t)$ denotes the set of processes that have crashed through time $t \in \mathscr{T}$. The set of crashed processes is described by $crashed(F) = \bigcup_{t \in \mathscr{T}} F(t)$, where the set of correct processes is described by $correct(F) = \Pi - crashed(F)$. $p \in crashed(F)$ says p crashes in F and $p \in correct(F)$ says p is correct in F. Note, only failure patterns F such that at least one process is correct are considered, i. e. $correct(F) \neq \emptyset$.

Each failure detector module outputs the set of processes that it currently suspects to have crashed. The function $H : \Pi \times \mathscr{T} \to 2^\Pi$ is called a *failure detector history*, where $H(p,t)$ denotes the value of the failure detector model of process p at time t. $q \in H(p,t)$ says that p suspects q at time t in H.

Informally, a failure detector $\mathscr{D}$ provides (possibly incorrect) information about the failure pattern F that occurs in an execution. Formally, failure detector $\mathscr{D}$ is a function that maps each failure pattern F to a set of failure detector histories $\mathscr{D}(F)$. This is the set of all failure detector histories that could occur in executions with failure pattern F and failure detector $\mathscr{D}$. We use two different failure detectors that satisfy the following completeness and accuracy properties.

- *Strong completeness*: Eventually every process that crashes is permanently suspected by every correct process.
  $\forall F. \forall H \in \mathscr{D}(F). \exists t \in \mathscr{T}. \forall p \in crashed(F). \forall q \in correct(F). \forall t' \geq t. \ p \in H(q,t')$
- *Strong accuracy*: No process is suspected before it crashes.
  $\forall F. \forall H \in \mathscr{D}(F). \forall t \in \mathscr{T}, \forall p,q \in \Pi - F(t). \ p \notin H(q,t)$
- *Weak accuracy*: Some correct process is never suspected.
  $\forall F. \forall H \in \mathscr{D}(F). \exists p \in correct(F). \forall t \in \mathscr{T}. \forall q \in \Pi - F(t). \ p \notin H(q,t)$

A failure detector is called *perfect* if it satisfies strong completeness and strong accuracy, with the set of all these failure detectors being denoted by $\mathscr{P}$. A failure detector is called *strong* if it satisfies strong completeness and weak accuracy, with the set of all these failure detectors being denoted by $\mathscr{S}$.

2.2 Syntax of the Calculus

We introduce a tailor-made calculus [9] to model Distributed Consensus. It is based on process calculi like the well-known CCS or π-calculus [12, 14]. The calculus is an adapted version of the calculus introduced by Francalanza and Hennessy in [4]. Most of the following definitions are taken verbatim from [9, 8].

The syntax of the calculus is shown in Table 1, it consists of four layers: data values, guarded processes, processes, and networks. The existence of a countably infinite set of channel, variable, and function names $\mathbf{A} = \{a, b, c, \ldots\}$ and a finite set of location names $\mathscr{L}oc$ that contains the special name $\star$ is assumed. The definitions are mostly standard. $\mathscr{S}(|k|).P$ is the process that behaves like P if the process

DATA VALUES **V**	v	$::=$	$\bot, 0, 1, 2, 3, \ldots \mid (v, v) \mid \{v, \ldots, v\}$
VARIABLE PATTERN	X	$::=$	$x \mid (X, X),$ with $x \in \mathbf{A}$
EXPRESSIONS	e	$::=$	$v \mid X \mid (e, e) \mid \mathtt{f}(e),$ with $\mathtt{f} \in \mathbf{A}$
GUARDED PROCESSES **G**	G	$::=$	$0 \mid \overline{c}\langle e \rangle.P \mid c(X).P \mid \mathscr{S}(\!|k|\!).P \mid \mathscr{P}(\!|k|\!).P \mid G + G$
			$\mid$ **if** e **then** G **else** G
PROCESSES **P**	P, Q	$::=$	$\tau.P \mid G \mid K(e) \mid P \parallel P$
NETWORKS **N**	M, N	$::=$	$0 \mid \ell[P] \mid N \parallel N \mid N \setminus a$
PROCESS EQUATIONS			$D \overset{\text{def}}{=} \{K_j(X) = P_j\}_{j \in J}$ a finite set of process definitions

Table 1: Syntax

k is suspected to have crashed and $\mathscr{P}(\!|k|\!).P$ behaves like P if the process k has crashed. $K(X)$ denotes a parametrized process constant, which is defined with respect to a finite set of process equations D of the form $\{K_j(X) = P_j\}_{j \in J}$.

Definition 2.1 (Configurations). Configurations C have either of the two forms $(\mathscr{L}, n) \triangleright_{\bot} M$ or $(\mathscr{L}, n) \triangleright_{\text{ti}} M$, where $\mathscr{L} \subseteq \mathscr{L}oc \setminus \{\star\}$ is a finite set of locations, $n \in \mathbb{N}$ is the number of processes that can crash and M is a network. The location $\text{ti} \in \mathscr{L}oc$, $\text{ti} \neq \star$ is called a *trusted immortal* [15]; it cannot be suspected and it never crashes. We define **C** as the set of all configurations.

We define the projection:

$$\mathrm{live}(\ell, (\mathscr{L}, n)) = \begin{cases} \text{true} & \text{,if } \ell \in \mathscr{L} \vee \ell = \star \\ \text{false} & \text{,else} \end{cases}$$

Hence $\mathscr{L} \subseteq \mathscr{L}oc$ is the set of live processes and n denotes the number of processes that are allowed to crash. Accordingly $\mathscr{L}oc \setminus \mathscr{L}$ denotes the set of crashed processes. Let $[\![e]\!]$ denote the evaluation of expression e, defined in the standard way.

The substitution of value v for a variable pattern X in expression e or process P is written $e\{v/X\}$ and $P\{v/X\}$ respectively. The operator $\mathrm{fn}(\cdot)$ defined on processes and networks is defined as usual. Notice that only data values can be substituted for names and that all variables of the pattern X must be free in P. We write $\overline{c}\langle e \rangle$ for $\overline{c}\langle e \rangle.0$ and $c.P$ for $c(x).P$, $x \notin \mathrm{fn}(P)$ and $\overline{c}$ for $\overline{c}\langle \bot \rangle$. Moreover we use $a@i(x).P \overset{\text{def}}{=} a(x).P + \mathscr{S}(\!|i|\!).P\{\bot/x\}$, i.e. either there is communication on channel a or location i is suspected. We proceed with P in both cases with respect to the given substitution.

2.3 Semantics of the Calculus

Network evaluation defines some rules to reduce configurations.

Definition 2.2. Let $>$ be the evaluation relation defined on configurations (assuming $\mathit{live}(\ell, \Gamma)$ every-where, except explicitly stated otherwise), closed under restriction, parallel composition, and the follow-

ing rules:

$$\Gamma \rhd_{\text{ti}} \ell\,[P \parallel Q] \qquad\qquad > \Gamma \rhd_{\text{ti}} \ell\,[P] \parallel \ell\,[Q] \tag{E1}$$

$$\Gamma \rhd_{\text{ti}} \ell\,[0] \qquad\qquad > \Gamma \rhd_{\text{ti}} 0 \tag{E2}$$

$$\Gamma \rhd_{\text{ti}} \ell\,[P] \qquad\qquad > \Gamma \rhd_{\text{ti}} 0, \ \ \neg\,\text{live}(\ell, \Gamma) \tag{E3}$$

$$\Gamma \rhd_{\text{ti}} 0 \parallel N \qquad\qquad > \Gamma \rhd_{\text{ti}} N \tag{E4}$$

$$\Gamma \rhd_{\text{ti}} N \parallel 0 \qquad\qquad > \Gamma \rhd_{\text{ti}} N \tag{E5}$$

$$\Gamma \rhd_{\text{ti}} \ell\,[\overline{c}\langle e\rangle.P] \qquad\qquad > \Gamma \rhd_{\text{ti}} \ell\,[\overline{c}\langle \llbracket e\rrbracket\rangle.P]$$

$$\Gamma \rhd_{\text{ti}} \ell\,[K(e)] \qquad\qquad > \Gamma \rhd_{\text{ti}} \ell\,[P\{\llbracket e\rrbracket/X\}], \ \ (K(X) \overset{\text{def}}{=} P) \in D$$

$$\Gamma \rhd_{\text{ti}} \ell\,[\textbf{if } e \textbf{ then } P \textbf{ else } Q] > \Gamma \rhd_{\text{ti}} \ell\,[P], \ \ \llbracket e\rrbracket > 0$$

$$\Gamma \rhd_{\text{ti}} \ell\,[\textbf{if } e \textbf{ then } P \textbf{ else } Q] > \Gamma \rhd_{\text{ti}} \ell\,[Q], \ \ \llbracket e\rrbracket = 0.$$

Let $C >^* C'$ denote the maximal evaluation of an arbitrary configuration C into C' respecting the rules above, i.e. $\forall C, C' \in \mathbf{C}.\ C >^* C' \overset{\text{def}}{=} C > \cdots > C' \wedge \nexists C''.\ C' > C''$.

We took this definition from [9] and added (similarly to [8]) the rules (E1) to (E5) to it. We removed transitivity to ease the proof of local confluence in Lemma 3.7, because otherwise we would not only have to prove diamonds but also triangles. Note that structural congruence contains the reflexive and transitive closure of $>$. As an example consider $\Gamma \rhd_{\text{ti}} \ell\,[\textbf{if } 1 \textbf{ then } (\overline{c}\langle e\rangle.P \parallel Q) \textbf{ else } R] >$ $\Gamma \rhd_{\text{ti}} \ell\,[\overline{c}\langle e\rangle.P \parallel Q]$. Without Rule (E1) we cannot evaluate this term any further. With this Rule we further evaluate $\Gamma \rhd_{\text{ti}} \ell\,[\overline{c}\langle e\rangle.P \parallel Q] > \Gamma \rhd_{\text{ti}} \ell\,[\overline{c}\langle e\rangle.P] \parallel \ell\,[Q] > \Gamma \rhd_{\text{ti}} \ell\,[\overline{c}\langle \llbracket e\rrbracket\rangle.P] \parallel \ell\,[Q]$. Intuitively the Rule (E1) allows to simplify the syntactic representation of a term as far as possible. This significantly simplifies the definition of the semantics in Section 4. Rules (E2) to (E5) remove dead processes from configurations.

Note that in [9] the Rule E1 is added to structural congruence instead of network evaluation. We find this version to be more intuitive and handy, because it allows us to omit structural congruence in some following proofs. Note that this decision does not influence the meaning of structural congruence.

Definition 2.3. Structural congruence $\equiv\, \subseteq \mathbf{C} \times \mathbf{C}$ is the least equivalence relation containing $>$, satisfying commutative monoid laws for $(\mathbf{N}, \parallel, 0)$ closed under restriction and parallel composition.

Actions $\alpha \in \text{Act}$ are of the form $\alpha ::= \tau \mid \overline{c}v \mid cv$. The transition relation $\longrightarrow\, \subseteq \mathbf{C} \times \text{Act} \times \mathbf{C}$ is the smallest relation satisfying the rules of Table 2. Rule (TI) non-deterministically selects a trusted immortal. This rule has to be applied initially because all other rules require the trusted immortal set. (Stop) allows agents to cease execution. (Susp) and (PSusp) model (perfect) suspicion of agents. (SumL) and (SumR) allow to reduce the left or right side of a sum. Similar (Par) and (Res) allow to reduce an agent within parallel composition or restriction. (Com), (Tau), (Snd), and (Rcv) model communication, internal steps, sending, and receiving. Finally, (Red) models steps modulo structural congruence.

Let $\Longrightarrow$ denote the reflexive and transitive closure of $\overset{\tau}{\longrightarrow}$ and let $\overset{\alpha}{\Longrightarrow}$ abbreviate $\Longrightarrow \overset{\alpha}{\longrightarrow} \Longrightarrow$ if $\alpha \neq \tau$ and $\Longrightarrow$ else. Let Act^* be the set of finite sequences of elements in $\text{Act} \setminus \{\tau\}$. We define $\overset{\sigma}{\Longrightarrow}$ for some sequence $\sigma = \alpha_1, \ldots, \alpha_n$ with $\alpha_{1 \leq i \leq n} \in \text{Act}$ as $\overset{\alpha_1}{\Longrightarrow} \cdots \overset{\alpha_n}{\Longrightarrow}$ for $n > 0$ and else as $\Longrightarrow$. We use the standard definition of weak bisimulation and weak bisimulation up to techniques as described e.g. in [16].

2.4 Case Study

Distributed Consensus is the following well-known problem: there is a fixed number n of agents each initially propose a value v_i, $1 \leq i \leq n$; then, eventually, the agents must agree on a common value

$$\text{(TI)}\quad \dfrac{\mathsf{ti} \in \mathscr{L} \setminus \{\star\}}{(\mathscr{L},n) \triangleright M \xrightarrow{\tau} (\mathscr{L},n) \triangleright_{\mathsf{ti}} M} \qquad\qquad \text{(Stop)}\quad \dfrac{\ell \neq \mathsf{ti} \wedge \ell \in \mathscr{L}}{(\mathscr{L},n+1) \triangleright_{\mathsf{ti}} M \xrightarrow{\tau} (\mathscr{L}\setminus\{\ell\},n) \triangleright_{\mathsf{ti}} M}$$

$$\text{(PSusp)}\quad \dfrac{\mathrm{live}(\ell,\Gamma) \wedge \neg\,\mathrm{live}(k,\Gamma)}{\Gamma \triangleright_{\mathsf{ti}} \ell[\mathscr{P}(|k|).P] \xrightarrow{\tau} \Gamma \triangleright_{\mathsf{ti}} \ell[P]} \qquad \text{(Susp)}\quad \dfrac{\mathrm{live}(\ell,\Gamma) \wedge k \neq \mathsf{ti} \wedge k \neq \ell}{\Gamma \triangleright_{\mathsf{ti}} \ell[\mathscr{S}(|k|).P] \xrightarrow{\tau} \Gamma \triangleright_{\mathsf{ti}} \ell[P]}$$

$$\text{(Tau)}\quad \dfrac{\mathrm{live}(\ell,\Gamma)}{\Gamma \triangleright_{\mathsf{ti}} \ell[\tau.P] \xrightarrow{\tau} \Gamma \triangleright_{\mathsf{ti}} \ell[P]} \qquad \text{(SumL)}\quad \dfrac{\mathrm{live}(l,\Gamma) \wedge \Gamma \triangleright_{\mathsf{ti}} \ell[G_1] \xrightarrow{\alpha} \Gamma' \triangleright_{\mathsf{ti}} \ell[P]}{\Gamma \triangleright_{\mathsf{ti}} \ell[G_1+G_2] \xrightarrow{\alpha} \Gamma' \triangleright_{\mathsf{ti}} \ell[P]}$$

$$\text{(Par)}\quad \dfrac{\Gamma \triangleright_{\mathsf{ti}} M \xrightarrow{\alpha} \Gamma' \triangleright_{\mathsf{ti}} M'}{\Gamma \triangleright_{\mathsf{ti}} M \parallel N \xrightarrow{\alpha} \Gamma' \triangleright_{\mathsf{ti}} M' \parallel N} \qquad \text{(SumR)}\quad \dfrac{\mathrm{live}(l,\Gamma) \wedge \Gamma \triangleright_{\mathsf{ti}} \ell[G_2] \xrightarrow{\alpha} \Gamma' \triangleright_{\mathsf{ti}} \ell[P]}{\Gamma \triangleright_{\mathsf{ti}} \ell[G_1+G_2] \xrightarrow{\alpha} \Gamma' \triangleright_{\mathsf{ti}} \ell[P]}$$

$$\text{(Snd)}\quad \dfrac{\mathrm{live}(\ell,\Gamma)}{\Gamma \triangleright_{\mathsf{ti}} \ell[\overline{c}\langle v\rangle] \xrightarrow{\overline{c}v} \Gamma \triangleright_{\mathsf{ti}} 0} \qquad \text{(Rcv)}\quad \dfrac{\mathrm{live}(\ell,\Gamma)}{\Gamma \triangleright_{\mathsf{ti}} \ell[c(X).P] \xrightarrow{cv} \Gamma \triangleright_{\mathsf{ti}} \ell[P\{v/X\}]}$$

$$\text{(Com)}\quad \dfrac{\Gamma \triangleright_{\mathsf{ti}} M \xrightarrow{\alpha} \Gamma \triangleright_{\mathsf{ti}} M' \quad \Gamma \triangleright_{\mathsf{ti}} N \xrightarrow{\overline{\alpha}} \Gamma \triangleright_{\mathsf{ti}} N'}{\Gamma \triangleright_{\mathsf{ti}} M \parallel N \xrightarrow{\tau} \Gamma \triangleright_{\mathsf{ti}} M' \parallel N'}, \ \alpha,\overline{\alpha} \neq \tau$$

$$\text{(Red)}\quad \dfrac{C \Rightarrow C_1 \xrightarrow{\alpha} C_2 \Rightarrow C'}{C \xrightarrow{\alpha} C'} \qquad \text{(Res)}\quad \dfrac{\Gamma \triangleright_{\mathsf{ti}} M \xrightarrow{\alpha} \Gamma' \triangleright_{\mathsf{ti}} M'}{\Gamma \triangleright_{\mathsf{ti}} M \setminus a \xrightarrow{\alpha} \Gamma' \triangleright_{\mathsf{ti}} M' \setminus a}, \ \alpha \neq \overline{a}v, av$$

Table 2: Structural Operational Semantics

$v_i \in \{v_1,\ldots,v_n\}$. The model in which we study this problem consists of asynchronously communicating agents that are vulnerable to crash failures. Each agent is furthermore equipped with a failure detector, which can detect whether other agents have stopped or not. The precise specification of the problem comprises three properties with temporal logic flavor:

Termination: Every live agent eventually decides some value.
Agreement: No two agents decide differently.
Validity: If an agent decides value v, then v was proposed by some agent.

Table 3 presents an algorithm by Chandra and Toueg [2] that is supposed to solve Distributed Consensus in the context of failure detector $\mathscr{S}$. It is based on three phases. In the following we give a short description of the algorithm. The system consists of n agents, which are identified by numbers, and the wrapper, i. e. $\mathscr{L}oc \stackrel{\text{def}}{=} \{\star,1,\ldots,n\}$. Every agent p has a knowledge vector V_p initially only containing its own proposed value v_p, i. e. $V_p(p) = v_p$. Unknown values are represented by $\bot$.

Validity is shown by an invariant. To show termination and agreement, we have to prove following bisimulation:

$$(\mathscr{L}oc,n-1) \triangleright_{\bot} (\text{System} \parallel \text{Wrapper}) \setminus R \approx (\mathscr{L}oc,0) \triangleright_{\mathsf{ti}} \star [\overline{ok}]$$

Phase 1 consists of $n-1$ rounds. In every round agent p broadcasts to every other agent a message Δ_p, initially containing the knowledge of the own proposed value. p then collects all messages Δ_q from every agent q in the system or suspects q to have crashed, i. e. $\mathscr{S}(|q|)$. If a message is received containing the proposed value of agent q which is not known by p it is added to the knowledge of p. After agent p processed all messages, it builds the message to send in the next round by taking all new learned values into this message.
Phase 2 is used to synchronize knowledge. Every agent p sends its knowledge vector V_p to every other agent. Then p collects for every agent q either V_q or suspects q. If p collected a message such that the message contains no knowledge about agent q, then p also removes knowledge about this agent, i. e. p sets $V_p(q) = \bot$.

```
 1: Pseudo code for agent p          16:        with Δ'(q) ≠ ⊥ then
 2: Vₚ ← ⊥̃, Vₚ(p) ← vₚ               17:           Vₚ(q) ← Δ'(q)
 3: Δₚ ← Vₚ, Mₚ ← ∅                   18:           Δₚ(q) ← Δ'(q)
 4:                                   19:
 5: Phase 1:                          20: Phase 2:
 6: for all rₚ ← 1 to n−1 do          21: send P2(Vₚ) to all
 7:    send P1(p,rₚ,Δₚ) to all        22: block until
 8:    Δₚ ← ⊥̃                         23:    for all 1 ≤ q ≤ n do
 9:    block until                    24:        receive m = P2(V)
10:       for all 1 ≤ q ≤ n           25:        Mₚ ← Mₚ ∪ {m}
11:          receive m = P1(q,rₚ,Δ)   26:        or suspect 𝒮(|q|)
12:          Mₚ ← Mₚ ∪ {m}            27: for all q ← 1 to n do
13:          or suspect 𝒮(|q|)        28:    if ∃V' ∈ Mₚ : V'(q) = ⊥
14:       for all q ← 1 to n do       29:        then Vₚ(q) ← ⊥
15:          if Vₚ(q) = ⊥ and ∃Δ' ∈ Mₚ 30:
                                      31: Phase 3:
                                      32: decide = min {q | Vₚ(q) ≠ ⊥}
```

Table 3: Distributed Consensus [2]

Phase 3. In Phase 3 every agent decides on a value $v = \min\{q \mid V_p(q) \neq \bot\}$.

We use an adapted version of the implementation of this algorithm as described in [8], see Table 4. The implementation is adapted such that the whole knowledge vector is no longer sent over channel c in Phase 3. This change does not affect the correctness of the algorithm because this vector is never used by the wrapper. We also changed the wrapper code. In the implementation of [8], the wrapper reduced to 0 upon receiving an invalid value v. Invalid has the meaning that the wrapper already learned a value v some agent p decided on and then received a value v' from some other agent q with $V \neq V'$. Because it might come in handy to know which process decided on that wrong value, we changed the wrapper in a way such that it has an additional parameter which represents whether it can receive messages or not. Initially the wrapper is allowed to receive messages but sets the parameter to 0 upon receiving an invalid value. Observe that $\mathsf{Wrap}(\cdot,\cdot,0) \approx 0$. Since these are the only changes, the adapted version of the implementation has the same behavior as the original version.

Definition 2.4. Each agent is uniquely identified by a number $p \in \Pi$, where $\Pi = \{1,\dots,n\}$ is the set of agents, and a proposed value v_p. The tuple U contains all initially proposed values, i.e. $U = (v_1,\dots,v_n)$.

We use $\bot$ for unknown values and define the domain of values D as $\mathbb{N} \cup \{\bot\}$ where $\mathbb{N} = \{1,2,3,\dots\}$ and let $\leq_{nat}$ be the usual ordering relation on the natural numbers. The ordering $\leq \subseteq D \times D$ is the least relation containing $\leq_{nat}$ with the additional requirements $\bot \leq \bot$ and $\bot \leq i$ for all $i = 1,\dots,n$. An n-vector is a map from set $\{1,\dots,n\}$ to the set D and $\tilde{\bot}$ is the n-vector $(\bot,\dots,\bot)$. We occasionally regard vectors as ordered lists of values. The ordering $\leq$ is extended point-wise to n-dimensional vectors and we write $V_1 \leq V_2$ when vector V_2 is greater than or equal to V_1. We say a vector V is valid if $V \leq U$, where U is the vector of Definition 2.4. The initial vector V_i^0 of agent i contains only its proposed value v_i at position $V_i^0(i)$, i.e. $V_i^0(i) = (U(i),0)$ and $V_i^0(j) = (\bot,0)$ for all $j \in \Pi \setminus \{i\}$. The replay vector is initialized in a similar way, i.e. $\Delta_i^0(i) = U(i)$ and $\Delta_i^0(j) = \bot$ for all $j \in \Pi \setminus \{i\}$. The initial configuration is a term in the calculus of the form $(\mathscr{L}oc, n-1) \triangleright_\bot \left(\prod_{i=1}^n i \left[\mathsf{P1}_i(1, I_i^0, I_i, \emptyset) \right] \parallel \star [\mathsf{Wrap}(1, \bot, 1)] \right) \setminus R$ for some $n \in \mathbb{N}$, $I_i^0(j) = v_i^0$ if $j = i$ and $\bot^0$ else, and $I_i(j) = v_i$ if $j = i$ and $\bot$ else. Let $\mathbf{C_0}$ denote the set of all initial configurations. To simplify the following consideration we do not regard initial

1: System $\overset{\text{def}}{=\!=}$
2: $\left(\prod_{i=1}^{n} i \left[\mathsf{P1}_i(1, I_i^0, I_i, \emptyset) \right] \right)$

3: $\mathsf{P1}_p(r, V, \Delta, M) \overset{\text{def}}{=\!=}$
4:    **if** $(r < n)$ **then**
5:      $\prod_{1 \leq i \leq n} \overline{a_{p,i,r}} \langle \Delta \rangle \parallel \mathsf{C1}_p(r, V, M, 1)$
6:    **else**
7:      $\mathsf{P2}_p(V, M)$

8: $\mathsf{C1}_p(r, V, M, i) \overset{\text{def}}{=\!=}$
9:    **if** $i \leq n$ **then**
10:      $a_{i,p,r} @ i(\Delta).\mathsf{C1}_p\Big(r, V, M + (\Delta, r, i), i+1 \Big)$
11:    **else**
12:      $\mathsf{P1}_p\Big(r+1, \mathtt{update_k}(r, M, V),$
13:                 $\mathtt{update_r}(r, M, V), M \Big)$

14: $\mathsf{P2}_p(V, M) \overset{\text{def}}{=\!=}$
15:    $\prod_{1 \leq i \leq n} \overline{b_{p,i}} \langle V \rangle \parallel \mathsf{C2}_p(V, M, 1)$

16: $\mathsf{C2}_p(V, M, i) \overset{\text{def}}{=\!=}$
17:    **if** $i \leq n$ **then**
18:      $b_{i,p} @ i(V').\mathsf{C2}_p(V, M + (V', i), i+1)$
19:    **else**
20:      $\mathsf{P3}_p(\mathtt{correct}(M, V))$

21: $\mathsf{P3}_p(V) \overset{\text{def}}{=\!=}$
22:    $\overline{c_p} \langle \mathtt{getfst}(V) \rangle$

23: $\mathsf{Wrap}(i, v, b) \overset{\text{def}}{=\!=}$
24:    **if** $(b == 1)$ **then**
25:      **if** $(1 \leq i \leq n)$ **then**
26:        $\mathscr{P}(|i|).\mathsf{Wrap}(i+1, v, 1) +$
27:        $c_i(v').$
28:          **if** $((v == \bot \wedge v' \mathrel{!=} \bot) \vee v == v')$ **then**
29:        $\mathsf{Wrap}(i+1, v', 1)$ **else** $\overline{\mathsf{Wrap}(i, v, 0)}$
30:      **else if** $(i == n+1)$ **then** $\overline{ok}$
31:    **else** $\mathsf{Wrap}(i, v, 0)$

Table 4: Encoding of the Algorithm in Table 3

configurations as reachable configurations, but only configurations that are reachable from some initial configuration and in which the trusted immortal is set. Let $\mathbf{C_R}$ denote the set of reachable terms, i.e. $\mathbf{C_R} = \left\{ \Gamma \triangleright_{\mathsf{ti}} M \mid \exists C \in \mathbf{C_0}.\ \exists \sigma \in \mathrm{Act}^*.\ C \overset{\sigma}{\Longrightarrow} \Gamma \triangleright_{\mathsf{ti}} M \right\}$.

3   Stateful Representatives

A stateful representative represents the global state of the system. We observe that the system consists of messages in transit or agents wanting to receive messages. To describe the state of the system we need variables that represent these messages and agents. Thus we obtain the sets of all messages of Phase 1, i.e. Ω_1, Phase 2, i.e. Ω_2, and Phase 3, i.e. Ω_3. We also get the sets of all agents wanting to receive messages of Phase 1, i.e. Θ_1, and Phase 2, i.e. Θ_2. These sets do not contain a single value like an agent identifier but rather a tuple of related informations, e.g. Ω_1 contains tuples (p, i, r, Δ). These tuples provide the information that there is a message from agent p to agent i in Round r containing the value Δ, we also know that this message belongs to Phase 1 because the tuple is contained in Ω_1.

Definition 3.1. [Stateful Representative] A standard form of a reachable configuration C is a configuration of the form in Table 5 that is structural congruent to C, where $\mathrm{live}(p, \boxed{\Gamma})$ for all occurrences of p. The stateful representative of a standard form is the vector given by all boxed elements, i.e. a stateful representative has the form:

$$(\Gamma, \mathsf{ti}, \Omega_1, \Omega_2, \Omega_3, \Theta_1, \Theta_2, \mathsf{j}, \mathsf{w}, \mathsf{b})$$

$$
\Gamma \rhd_{\mathsf{ti}} \left(\begin{array}{l}
\displaystyle\prod_{(p,i,r)\in \Pi_1^{\mathrm{out}}} p\left[\overline{a_{p,i,r}}\langle \Delta_{p,r}\rangle\right] \parallel \\[2mm]
\displaystyle\prod_{(p,i)\in \Pi_2^{\mathrm{out}}} p\left[\overline{b_{p,i}}\langle V_p^{\mathsf{P2}}\rangle\right] \parallel \\[2mm]
\displaystyle\prod_{p\in \Pi_3^{\mathrm{out}}} p\left[\overline{c_p}\langle v_p^{\mathsf{P3}}\rangle\right] \parallel \\[2mm]
\displaystyle\prod_{(p,r)\in \Pi_1^{\mathrm{col}}} p\left[\mathsf{C1}_p\left(r, V_p^{\mathsf{P1}}, M_p^{\mathsf{P1}}, i_p^{\mathsf{P1}}\right)\right] \parallel \\[2mm]
\displaystyle\prod_{p\in \Pi_2^{\mathrm{col}}} p\left[\mathsf{C2}_p\left(V_p^{\mathsf{P2}}, M_p^{\mathsf{P2}}, i_p^{\mathsf{P2}}\right)\right] \parallel \\[2mm]
\star\left[\mathsf{Wrap}(\mathsf{j},\mathsf{w},\mathsf{b})\right]
\end{array} \right) \setminus R
$$

where $i_p^{\mathsf{P1}} \leq n, i_p^{\mathsf{P2}} \leq n, \mathsf{j} \leq n, r < n$, and $\mathsf{b} \in \{0,1\}$.

Table 5: Standard Form for Reachable Configurations

with

$$
\Omega_1 = \left\{ (p,i,r,\Delta) \mid (p,i,r) \in \Pi_1^{\mathrm{out}} \wedge \Delta = \Delta_{p,r} \right\}
$$

$$
\Omega_2 = \left\{ (p,i,V) \mid (p,i) \in \Pi_2^{\mathrm{out}} \wedge V = V_p^{\mathsf{P2}} \right\}
$$

$$
\Omega_3 = \left\{ (p,v) \mid p \in \Pi_3^{\mathrm{out}} \wedge v = v_p^{\mathsf{P3}} \right\}
$$

$$
\Theta_1 = \left\{ (p,r,V,M,i) \mid (p,r) \in \Pi_1^{\mathrm{col}} \wedge V = V_p^{\mathsf{P1}} \wedge M = M_p^{\mathsf{P1}} \wedge i = i_p^{\mathsf{P1}} \right\}
$$

$$
\Theta_2 = \left\{ (p,V,M,i) \mid p \in \Pi_2^{\mathrm{col}} \wedge V = V_p^{\mathsf{P2}} \wedge M = M_p^{\mathsf{P2}} \wedge i = i_p^{\mathsf{P2}} \right\}
$$

Let $\mathbf{C_N}$ denote the set of standard forms. Hence $\mathbf{C} \supsetneq \mathbf{C_R} \supsetneq \mathbf{C_N}$.

By Definition 3.1 every standard form represents a reachable configuration. We prove that for each reachable configuration there is a standard form, and hence a stateful representative.

Lemma 3.2. *For every reachable configuration there is a standard form.*

We prove this lemma by induction over the number of steps necessary to reach a configuration from an initial configuration.

Definition 3.3 (NF). Let $\mathrm{NF} : \mathbf{C_R} \to \mathbf{C_N}$ be a function defined by

$$
\mathrm{NF}(C) = \mathrm{rw}(\mathrm{order}(\mathrm{identify}(C')))
$$

where $C >^* C'$, i. e. identify is applied to a fully evaluated configuration.

The auxiliary function identify is given by:

$$
\mathrm{identify}(\Gamma \rhd_{\mathsf{ti}} N \setminus R) = \Gamma \rhd_{\mathsf{ti}} \mathrm{identify}(N) \setminus R
$$

$$
\mathrm{identify}(N \parallel M) = \mathrm{identify}(N) \parallel \mathrm{identify}(M)
$$

$$
\mathrm{identify}(\ell[P]) = \ell[\mathrm{translate}(P)]
$$

By Definition 2.2, a fully evaluated configuration is a parallel composition of terms of the form as visualized in Table 4 Lines 5, 10, 15, 18, 22, 26–29, 30, and 31. With the function identify, translate is applied to each of these subterms. For the following terms

$$Ph1 \stackrel{\text{def}}{=} a_{i,p,r}@i(\Delta).C1_p\Big(r,V,M+(\Delta,r,i),i+1\Big)$$

$$Ph2 \stackrel{\text{def}}{=} b_{i,p}@i(V').C2_p(V,M+(V',i),i+1)$$

$$W \stackrel{\text{def}}{=} \mathscr{P}(|i|).\mathsf{Wrap}(i+1,v) +$$
$$c_i(v').$$
$$\textbf{if } ((v == \bot \wedge v' \mathrel{!=} \bot) \vee v == v') \textbf{ then}$$
$$\mathsf{Wrap}(i+1,v',1) \textbf{ else } \mathsf{Wrap}(i,v,0)$$

we define translate as:

$$\text{translate}(Ph1) = C1_p(r,V,M,i)$$
$$\text{translate}(Ph2) = C2_p(V,M,i)$$
$$\text{translate}(W) = \mathsf{Wrap}(i,v,1)$$
$$\text{translate}(\overline{ok}) = \mathsf{Wrap}(0,v,1)$$
$$\text{translate}(P) = P, \text{ otherwise}$$

The function order orders the subterms of the configuration into a standard form as visualized in Table 5. First all outputs on channel a are moved to the beginning of the standard form followed by all outputs on b and c. Then the input guarded terms on channel a and b, denoted as $C1_p(\cdot,\cdot,\cdot,\cdot)$ respectively $C2_p(\cdot,\cdot,\cdot)$, followed by the Wrapper. Hence the function order can be implemented using the structural congruence rules for commutativity and associativity of the parallel operator.

We define rw as:

$$\text{rw}(C) = \begin{cases} \Gamma \rhd_{\text{ti}} (M \parallel \star[\mathsf{Wrap}(0,\bot,1)]) \setminus R & \text{, if } C = \Gamma \rhd_{\text{ti}} M \setminus R \text{ and} \\ & \qquad M \text{ does not contain the location } \star \\ C & \text{, else} \end{cases}$$

If the wrapper was reduced to 0, the function restores the wrapper to obtain a standard form as visualized in Table 5.

Note that restriction is moved outwards by the definition above. Note as well, that there does not exist an inverse function because NF is not injective. However for each configuration C passed to NF we can compute an inverse to the standard form $NF(C)$ which is structural congruent to C. Therefore we will now define such a function returning a representative for the inverse being denoted by NF^{-1}.

Definition 3.4 (NF^{-1})**.** Let $NF^{-1} : \mathbf{C_N} \to \mathbf{C_R}$ be a function defined by

$$NF^{-1}(C) = \text{identify}'(C)$$

The auxiliary function identify$'$ is given by:

$$\text{identify}'(\Gamma \rhd_{\text{ti}} N \setminus R) = \Gamma \rhd_{\text{ti}} \text{identify}'(N) \setminus R$$
$$\text{identify}'(N \parallel M) = \text{identify}'(N) \parallel \text{identify}'(M)$$
$$\text{identify}'(\ell[P]) = \ell\left[\text{translate}^{-1}(P)\right]$$

with translate^{-1} being the inverse of the function translate given in the definition of NF.

Lemma 3.5. *For every reachable configuration, the function* NF *returns the corresponding standard form, i. e.:* $\forall C \in \mathbf{C_R}. \, C \equiv \mathrm{NF}(C) \wedge \mathrm{NF}(C) \in \mathbf{C_N}$.

Accordingly we show that the function NF^{-1} returns for every standard form a structural congruent configuration. Note that $\mathrm{NF}(\mathrm{NF}^{-1}(N)) = N$ and $\mathrm{NF}^{-1}(\mathrm{NF}(C)) \equiv C$.

Lemma 3.6. *For every standard form N,* $\mathrm{NF}^{-1}(N)$ *returns a structural congruent configuration, i. e.:* $\forall N \in \mathbf{C_N}. \, N \equiv \mathrm{NF}^{-1}(N) \wedge \mathrm{NF}^{-1}(N) \in \mathbf{C_R}$

In the following we show that the operator $>^*$ returns a fixed point, which is a necessary condition for the uniqueness of stateful representatives.

Lemma 3.7 (Confluence)**.** *The operator* $>^*$ *returns a fixed point, i. e.:*

$$\forall C_1, C_2 \in \mathbf{C_R}. \, C_1 > C_2 \text{ implies } \exists! C' \in \mathbf{C_R}. \, C_1 >^* C' \wedge C_2 >^* C'$$

Based on the lemma above we conclude that $\mathrm{NF}(C)$ returns a unique standard form for every reachable configuration.

Lemma 3.8. *Let C be a reachable configuration with* $\mathrm{NF}(C)$ *as the corresponding standard form. For every other configuration* C' *with* $C \equiv C'$ *the following holds:* $\mathrm{NF}(C) = \mathrm{NF}(C')$.

Finally we show that the definition of stateful representatives is unambiguous.

Lemma 3.9. *Let* $C \in \mathbf{C_R}$ *be a reachable configuration with the standard form N, then* $\mathrm{NF}(C) = N$.

Now we can proof that for every configuration C that has a standard form and a step C to C' then there also exists a standard form for C', hence C and C' have stateful representatives.

Theorem 3.10. *If* $C \xrightarrow{\tau} C'$ *and C has a standard form, then* C' *also has a standard form.*

4 A Semantics for Stateful Representatives

Building a Semantics for Stateful Representatives we can exploit the Stateful Representative being a global state of the algorithm. So we can directly derive a Semantics on Stateful Representatives out of the algorithm itself—by doing so we can ignore the actual implementation. The only obstacle is to make sure the resulting Stateful Representative is unique. As shown in Section 3 the Stateful Representative is a reordered version of the maximally evaluated calculus term. Thus we have to keep that in mind while creating the rules.

For example in Phase 1 of the Algorithm we have to distinguish between 3 cases: (i) transition to the next Round, (ii) transition to the next Phase, (iii) else. Additionally we have to distinguish between message reception and suspicion of the other process. The rules that model suspicion can directly be derived from the rules that model message reception because we have to handle received values differently.

The next three rules describe the behavior of a receiving process in Phase 1 ($(q, r) \in \Pi_1^{\mathrm{col}}$) with the sending process not being suspected ($\mathrm{live}(p, \Gamma)$). We distinguish between three cases according to Line 4 and Line 9:

SR3: The receiving agent is in the last round of Phase 1 receiving a message from the last agent ($p = n \wedge r = n - 1$).

SR2: The receiving agent is not in the last round of Phase 1 receiving a message from the last agent ($p = n \wedge r < n - 1$).

SR1: Else ($p < n \wedge r < n$).

$\exists p,q \in \mathscr{L}oc.\ \text{live}(p,\Gamma) \wedge \text{live}(q,\Gamma).\ \exists r \in \mathbb{N}.\ p < n \wedge r < n.$ **Rule** SR1
$(p,q,r) \in \Pi_1^{\text{out}} \wedge (q,r) \in \Pi_1^{\text{col}} \wedge i_q^{\text{P1}} = p.$

$$
\begin{array}{rcll}
\widehat{\Pi}_1^{\text{out}} & = & \Pi_1^{\text{out}} \setminus \{(p,q,r)\} & \text{remove the message received} \\
\wedge\quad \widehat{i}_q^{\text{P1}} & = & i_q^{\text{P1}} + 1 & \text{receive from the next process} \\
\wedge\quad \widehat{M}_q^{\text{P1}} & = & M_q^{\text{P1}} \cup \{(\Delta_{p,r},r,p)\} & \text{add the received information to local messages}
\end{array}
$$

$\exists p,q \in \mathscr{L}oc.\ \text{live}(p,\Gamma) \wedge \text{live}(q,\Gamma).\ \exists r \in \mathbb{N}.\ p = n \wedge r < n-1.$ **Rule** SR2
$(p,q,r) \in \Pi_1^{\text{out}} \wedge (q,r) \in \Pi_1^{\text{col}} \wedge i_q^{\text{P1}} = p.$

$$
\begin{array}{rcll}
\widehat{\Pi}_1^{\text{out}} & = & \bigcup_{j=1}^{n}\{(q,j,r+1)\} \cup (\Pi_1^{\text{out}} \setminus \{(p,q,r)\}) & \text{additionally send messages to all other proc} \\
\wedge\quad \widehat{i}_q^{\text{P1}} & = & 1 & \text{receive from the first process} \\
\wedge\quad \widehat{M}_q^{\text{P1}} & = & M_q^{\text{P1}} \cup \{(\Delta_{p,r},r,p)\} & \\
\wedge\quad \widehat{V}_q^{\text{P1}} & = & \texttt{update}_{\texttt{k}}(r,\widehat{M}_q^{\text{P1}},V_q^{\text{P1}}) & \text{update knowledge} \\
\wedge\quad \widehat{\Delta}_{q,r+1} & = & \texttt{update}_{\texttt{r}}(r,\widehat{M}_q^{\text{P1}},V_q^{\text{P1}}) & \text{update newly received knowledge} \\
\wedge\quad \widehat{\Pi}_1^{\text{col}} & = & \big(\Pi_1^{\text{col}} \setminus \{(q,r)\}\big) \cup \{(q,r+1)\} & \text{collect messages of the next round}
\end{array}
$$

The change to $\widehat{i}_q^{\text{P1}}$ may be omitted in the following rule because no according triple is added to the set $\widehat{\Pi}_1^{\text{out}}$. Because $\widehat{\Pi}_1^{\text{col}} = \Pi_1^{\text{col}} \setminus \{(q,r)\}$ we may also omit the change to $\widehat{i}_q^{\text{P1}}$. Both changes have no effect on the derived stateful representative.

$\exists p,q \in \mathscr{L}oc.\ \text{live}(p,\Gamma) \wedge \text{live}(q,\Gamma).\ \exists r \in \mathbb{N}.\ p = n \wedge r = n-1.$ **Rule** SR3
$(p,q,r) \in \Pi_1^{\text{out}} \wedge (q,r) \in \Pi_1^{\text{col}} \wedge i_q^{\text{P1}} = p.$

$$
\begin{array}{rcll}
\widehat{\Pi}_1^{\text{out}} & = & \Pi_1^{\text{out}} \setminus \{(p,q,r)\} & \\
\wedge\quad \widehat{i}_q^{\text{P1}} & = & i_q^{\text{P1}} + 1 & \\
\wedge\quad \widehat{\Pi}_2^{\text{out}} & = & \bigcup_{j=1}^{n}\{(q,j)\} \cup \Pi_2^{\text{out}} & \text{send messages in Phase 2} \\
\wedge\quad \widehat{M}_q^{\text{P1}} & = & M_q^{\text{P1}} \cup \{(\Delta_{p,r},r,p)\} & \\
\wedge\quad \widehat{V}_q^{\text{P1}} & = & \texttt{update}_{\texttt{k}}(r,\widehat{M}_q^{\text{P1}},V_q^{\text{P1}}) & \\
\wedge\quad \widehat{\Delta}_{q,r+1} & = & \texttt{update}_{\texttt{r}}(r,\widehat{M}_q^{\text{P1}},V_q^{\text{P1}}) & \\
\wedge\quad \widehat{\Pi}_1^{\text{col}} & = & \Pi_1^{\text{col}} \setminus \{(q,r)\} & \text{stop receiving messages in Phase 1} \\
\wedge\quad \widehat{\Pi}_2^{\text{col}} & = & \Pi_2^{\text{col}} \cup \{q\} & \text{receives messages in Phase 2} \\
\wedge\quad \widehat{i}_q^{\text{P2}} & = & 1 & \\
\wedge\quad \widehat{V}_q^{\text{P2}} & = & \widehat{V}_q^{\text{P1}} & \text{transfer knowledge to Phase 2} \\
\wedge\quad \widehat{M}_q^{\text{P2}} & = & \widehat{M}_q^{\text{P1}} & \text{transfer knowledge to Phase 2}
\end{array}
$$

The Rules SR4 to SR6 are nearly the same as Rules SR1 to SR3. They model suspicion of the sending process, thus the only differences in the Rules are: (i) the message in transit does not get removed, (ii) the value that should be received gets replaced by $\bot$.

The next rule applies if some agent has crashed, i. e. the process and all of its messages gets removed from the system.

$\exists p \in \mathscr{L}oc.\ p \neq \text{ti} \wedge \text{live}(p,\Gamma).$ **Rule** SR7

$$
\begin{array}{rcl}
(\widehat{\mathscr{L}},\widehat{n}) & = & (\mathscr{L} \setminus \{p\}, n-1) \\
\wedge\quad \widehat{\Pi}_1^{\text{out}} & = & \Pi_1^{\text{out}} \setminus \{(p,i,r) \mid \exists i \in \mathscr{L}oc, r \in \mathbb{N}.\ (p,i,r) \in \Pi_1^{\text{out}}\} \\
\wedge\quad \widehat{\Pi}_2^{\text{out}} & = & \Pi_2^{\text{out}} \setminus \{(p,i) \mid \exists i \in \mathscr{L}oc.\ (p,i) \in \Pi_2^{\text{out}}\} \\
\wedge\quad \widehat{\Pi}_3^{\text{out}} & = & \Pi_3^{\text{out}} \setminus \{p\} \\
\wedge\quad \widehat{\Pi}_1^{\text{col}} & = & \Pi_1^{\text{col}} \setminus \{(p,r) \mid \exists r \in \mathbb{N}.\ (p,r) \in \Pi_1^{\text{col}}\} \\
\wedge\quad \widehat{\Pi}_2^{\text{col}} & = & \Pi_2^{\text{col}} \setminus \{p\}
\end{array}
$$

$$N \rightsquigarrow \mathrm{NF}^{-1}(N)$$

$$\forall \qquad \exists$$

$$N' \rightsquigarrow \mathrm{NF}^{-1}(N')$$

Figure 1: Soundness

$$\mathrm{NF}(C) \leftsquigarrow C$$

$$\exists \qquad \forall$$

$$\mathrm{NF}(C') \leftsquigarrow C'$$
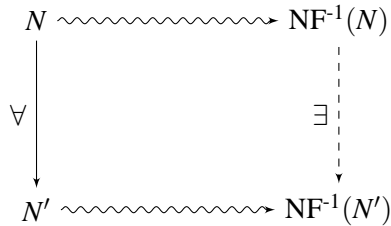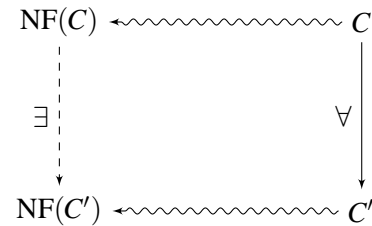
Figure 2: Completeness

4.1   Correctness

To obtain a 1-1-correspondence, we prove our semantics to be equivalent to the calculus semantics restricted to standard forms, see Figure 1 and 2. Therefore we show two main properties which we call *soundness* and *completeness*.

Soundness. We call our semantics to be sound, if whenever a standard form N can do a step to N', the according configuration C can do a step to C' such that N' is the standard form to C'.

Completeness. We call our semantics to be complete, if for every step of a configuration C to C', the according standard form N can do a step to N' such that N' is the according standard form to C'.

By showing these properties we are able to solely use the semantics of stateful representatives to show properties of the implementation itself.

Being able to focus on stateful representatives enables us to tame the state space, because by Definition 3.1 a standard form is structural congruent to a configuration and by Lemma 3.8 every two structurally equivalent configurations have the same standard form. Hence a stateful representative is a representative of an equivalence class of configurations.

The semantics is sound regarding to the calculus if every step in the stateful representative semantics can be emulated by the calculus semantics, see Figure 1.

Theorem 4.1 (Soundness). $\forall N, N' \in \mathbf{C_N}.\ N \longrightarrow N'$ *implies* $\exists C, C' \in \mathbf{C_R}.\ \mathrm{NF}^{-1}(N) = C \wedge \mathrm{NF}^{-1}(N') = C' \wedge C \xrightarrow{\tau} C'.$

To show soundness we have to provide a proof tree using the calculus semantics for each rule of the stateful representative semantics. By proving soundness we observe that we obtain 7 proof trees, two for each phase and one for process crashes.

To prove completeness we have to show that every transition being possible in the calculus beginning from the initial setting can be simulated by our stateful representative semantics, see Figure 2. For simplicity in the following we always assume live for the sending and receiving agent in a communication step. In the case of suspicion we assume live for the receiving agent and neither live nor ¬live for the sending agent. Note that neither the wrapper nor the trusted immortal can crash.

Theorem 4.2 (Completeness). $\forall C, C' \in \mathbf{C_R}.\ C \xrightarrow{\tau} C'$ *implies* $\exists N, N' \in \mathbf{C_N}.\ N = \mathrm{NF}(C) \wedge N' = \mathrm{NF}(C') \wedge N \longrightarrow N'.$

To prove this Theorem we use Lemmata 3.2, 3.5, and 3.8 to be able to only observe standard forms instead of every reachable configuration. Analyzing the standard form we observe that the only possible steps are either communication or process crashes. For each of these cases we show that there is a corresponding rule in the stateful representative semantics, e. g. communication in Phase 1 corresponds to Rules SR1 to SR3. We show by case differentiation that for each communication step in Phase 1 exactly one of these rules can be applied to obtain the correct stateful representative. The proof for the other cases is similar.

Soundness and correctness allows us to freely switch between both semantics. Thus we can now use state based reasoning techniques like invariants as well as action based techniques like bisimulation at the same time.

5   Conclusion

The main ideas have been explained in the Introduction. We may finally emphasize that we could have started the whole verification exercise from scratch using a state machine approach, for example using the ASM formalism or also TLA+. Actually, we have done such work ourselves, as in [5] as the basis for fully formal verification [7] within the Isabelle proof checking environment, but we are still convinced that the initial design of the algorithm using language primitives to "code" the local behaviors for the individual processes is of good value. After all, it is then very easy for programmers to implement the algorithm on that basis. In comparison, this would be rather indirect when starting from a state machine approach.

References

[1] T. D. Chandra, V. Hadzilacos & S. Toueg (1996): *The weakest failure detector for solving consensus*. *J. ACM* 43(4), pp. 685–722, doi:10.1145/234533.234549.

[2] T. D. Chandra & S. Toueg (1996): *Unreliable failure detectors for reliable distributed systems*. *J. ACM* 43(2), pp. 225–267, doi:10.1145/226643.226647.

[3] R. De Nicola & F. Vaandrager (1990): *Action versus state based logics for transition systems*. In: *Semantics of Systems of Concurrent Processes*, *LNCS* 469, pp. 407–419, doi:10.1007/3-540-53479-2_17.

[4] A. Francalanza & M. Hennessy (2007): *A Fault Tolerance Bisimulation Proof for Consensus (Extended Abstract)*. In: *ESOP*, *LNCS* 4421, pp. 395–410, doi:10.1007/978-3-540-71316-6_27.

[5] R. Fuzzati, M. Merro & U. Nestmann (2007): *Distributed Consensus, revisited*. *Acta Informatica*, pp. 377–425, doi:10.1007/s00236-007-0052-1.

[6] H. Hansen, H. Virtanen & A. Valmari (2003): *Merging State-Based and Action-Based Verification*. In: *ACSD*, pp. 150–156, doi:10.1109/CSD.2003.1207709.

[7] P. Küfner, U. Nestmann & C. Rickmann (2012): *Formal Verification of Distributed Algorithms - From Pseudo Code to Checked Proofs*. In: *IFIP TCS*, *LNCS* 7604, pp. 209–224, doi:10.1007/978-3-642-33475-7_15.

[8] M. Kühnrich (2011): *Approaches to Formal Verification*. Ph.D. thesis, Aalborg University.

[9] M. Kühnrich & U. Nestmann (2009): *On Process-Algebraic Proof Methods for Fault Tolerant Distributed Systems*. In: *FORTE*, *LNCS* 5522, pp. 198–212, doi:10.1007/978-3-642-02138-1_13.

[10] L. Lamport (1982): *An Assertional Correctness Proof of a Distributed Algorithm*. *Sci. Comput. Program.* 2(3), pp. 175–206, doi:10.1016/0167-6423(83)90014-X.

[11] N. Lynch (1996): *Distributed Algorithms*. Kaufmann Publishers.

[12] R. Milner (1980): *A calculus of communicating systems*. Springer, doi:10.1007/3-540-10235-3.

[13] R. Milner (1999): *Communicating and mobile systems: the π-calculus*. Cambridge University Press.

[14] R. Milner, J. Parrow & D. Walker (1992): *A calculus of mobile processes, I & II*. *Information and Computation* 100(1), pp. 1–77, doi:10.1016/0890-5401(92)90008-4.

[15] U. Nestmann & R. Fuzzati (2003): *Unreliable Failure Detectors via Operational Semantics*. In: *ASIAN*, *LNCS* 2896, pp. 54–71, doi:10.1007/978-3-540-40965-6_5.

[16] D. Sangiorgi & D. Walker (2001): *The π-calculus: A Theory of Mobile Processes*. Cambridge University Press.