Formal Verification of Consensus Algorithms Tolerating Malicious Faults

Bernadette Charron-Bost[1], Henri Debrat[2], and Stephan Merz[2]

[1] CNRS & LIX, Palaiseau, France, charron@lix.polytechnique.fr
[2] INRIA Nancy & LORIA, Nancy, France, {Henri.Debrat,Stephan.Merz}@loria.fr

Abstract. Consensus is the paradigmatic problem in fault-tolerant distributed computing: it requires network nodes that communicate by message passing to agree on common value even in the presence of (benign or malicious) faults. Several algorithms for solving Consensus exist, but few of them have been rigorously verified, much less so formally. The Heard-Of model proposes a simple, unifying framework for defining distributed algorithms in the presence of communication faults. Algorithms proceed in communication-closed rounds, and assumptions on the faults tolerated by the algorithm are stated abstractly in the form of communication predicates. Extending previous work on the case of benign faults, our approach relies on the fact that properties such as Consensus can be verified over a coarse-grained, round-based representation of executions. We have encoded the Heard-Of model in the interactive proof assistant Isabelle/HOL and have used this encoding to formally verify three Consensus algorithms based on synchronous and asynchronous assumptions. Our proofs give some new insights into the correctness of the algorithms, in particular with respect to transient faults.

1 Introduction

Fault-tolerant distributed computing is the art of making separate computing nodes cooperate for achieving a common objective, even in the presence of faults. In particular, the Consensus problem assumes that every node initially proposes some value, and requires that nodes eventually choose a common value among the proposed ones. Fault-tolerant distributed algorithms are often subtle, both in their operational design and in the assumptions they make, including the underlying model of communication and the kinds and numbers of faults they tolerate. *Benign* faults prevent processes from receiving expected messages, but do not affect the contents of messages received; these may be caused e.g. by process crashes or link breaks. *Malicious* faults are more severe as they are aimed to model any type of (process and link) malfunctioning, including corrupted process states and corrupted messages. It is well known [8] that Consensus is unsolvable in a fully asynchronous model of communication if at least one node may fail by crashing, but that it can be solved in partially synchronous models [6] even in the presence of malicious faults.

Given the subtle differences between communication and fault models, it is all too easy to make erroneous claims about what algorithms actually achieve, and formal statements and proofs about algorithms appear crucial for comparing them. Surprisingly, few rigorous correctness proofs exist in the literature, and even fewer of them have been fully checked with the help of formal verification methods and tools [18,13]. We believe that this lack of formal analysis is largely due to the absence of widely accepted frameworks in which models of computation and faults can be expressed and compared.

Charron-Bost and Schiper [5] proposed the Heard-Of (HO) model as a simple, unifying framework for defining distributed algorithms that operate in the presence of benign communication faults. In this model, computations are structured in rounds: during every round, each process first sends messages, then receives messages from other processes, and finally makes a local state transition. Rounds are communication-closed layers in the sense that processes receive messages solely sent at the round they currently execute. The round-based structure of the HO model greatly facilitates the design and understanding of distributed algorithms: an algorithm is simply specified by defining a message sending function and a next-state function, for every process and round. This operational description is then complemented by imposing communication predicates on executions, which restrict the kinds of faults that the algorithm tolerates. It has been shown [5] that common communication and fault models can be represented within the HO model. In previous work [3,4] we have proved that important correctness properties such as Consensus can be verified over a “coarse-grained” model of executions in which rounds are executed atomically, and have applied this result for formally verifying algorithms in the HO model, using model checking and interactive theorem proving. Independently, Tsuchiya and Schiper [19,20] also proposed the use of symbolic model checking for verifying HO algorithms.

In the meantime, the HO model has been augmented for supporting malicious communication faults [2], and it is this extension that we refer to when we speak of the HO model in the following. Extending [4], we show in this paper that the reduction theorem that allows us to consider only round-based executions remains valid in the presence of malicious faults. We present an encoding of the HO model in the interactive theorem prover Isabelle/HOL [16]. We have used this encoding to formally prove the correctness of three algorithms for achieving Consensus in the presence of malicious faults: the $\mathcal{U}_{T,E,\alpha}$ and $\mathcal{A}_{T,E,\alpha}$ algorithms from [2], and the well-known *Exponential Information Gathering* (EIGByz_f) algorithm [1,15]. The overall approach to verification is quite similar for all three algorithms. In particular, the EIGByz_f algorithm could be transposed “as is” in the HO model, although it was originally introduced in a traditional model that caters for faults of processes. We show that EIGByz_f tolerates certain transient faults, which could not be expressed by the original model. The precise, yet abstract representation of hypotheses on allowed faults in the HO model lets us not only express such faults but also analyze precisely to which property each hypothesis contributes. The full Isabelle theories are available on the Web[3].

The paper is structured as follows: Section 2 reviews the HO model, formally defines fine-grained and coarse-grained executions, states the fundamental re-

[3] `http://www.loria.fr/~debrat/`

duction theorem, and formally defines the Consensus problem. Our encoding of the HO model in Isabelle is described in Section 3. Its application to the verification of the three algorithms we consider appears in Section 4. Section 5 discusses related work and concludes the article.

2 The Heard-Of Model for Distributed Algorithms

Computations in the HO model are composed of rounds, in which processes exchange messages, take a step, and then proceed to the next round. In the parlance of Elrad and Francez [7], each round is a communication-closed layer in the sense that any message sent in a round can be received only in that round. Communication faults are abstractly represented in the HO model by means of *heard-of sets* (HO) that indicate which links are alive, thus capturing message omissions, and *safe heard-of sets* (SHO) that indicate which links are safe, thus capturing message corruption.

2.1 A round-based computational model

We suppose that we have a finite, nonempty set Π of process identifiers (or simply *processes*) and a set of messages M. By including a designated *empty message* in M that processes use to indicate absence of useful information, we may assume that each process sends a message to every process in Π, in each round. We denote the cardinality of Π by $N > 0$, let $\bot \notin M$ be a placeholder indicating that no message has been received, and write $M_\bot = M \cup \{\bot\}$. To each p in Π, we associate a *process specification* $Proc_p = (States_p, Init_p, S_p, T_p)$ whose components are the following:

- $States_p$ is a set of p's *states*, and $Init_p \subseteq States_p$ is a nonempty subset of *initial states* of process p,
- for each integer $r \in \mathbb{N}$, a *message-sending function* $S_p^r : States_p \times \Pi \to M$;
- for each integer $r \in \mathbb{N}$, a *next-state function* $T_p^r : States_p \times M_\bot^\Pi \to States_p$.

The next-state function T_p^r takes as its arguments the current state of process p and a mapping from (sender) processes to messages or $\bot$, and returns the next state of p.[4] In particular, the HO model is built on the assumption of *point-to-point* communications, and the next-state function definition is such that in its second argument, received messages are indexed by Π. The collection of process specifications $Proc_p$ is called an *algorithm* on Π.

As an example of an HO algorithm, a specification of the $\mathcal{U}_{T,E,\alpha}$ algorithm introduced in [2], appears as Algorithm 1. We consider a nonempty set V, with a specific element $v_0 \in V$; the two values ? and *null* are assumed not to be in V. Each process p maintains three variables x_p, $vote_p$, and $decide_p$ initialized

[4] For notational simplicity, we assume here that T_p^r is a function, but all results carry over to the case of next-state relations (i.e., non-deterministic processes) [3], which are accommodated by our Isabelle representation.

Algorithm 1 The $\mathcal{U}_{T,E,\alpha}$ algorithm

```
1: Initialization:
2:    $x_p \in V$; initially $x_p = v_p$                    { $v_p$ is the initial value of $p$ }
3:    $vote_p \in V \cup \{?\}$; initially $vote_p = ?$
4:    $decide_p \in V \cup \{null\}$; initially $decide_p = null$

5: Round $r = 2\phi$
6:    $S_p^r$ : send $\langle x_p \rangle$ to all processes
7:    $T_p^r$ : if received $> T$ values equal to $v$ with $v \in V$ then $vote_p := v$

8: Round $r = 2\phi + 1$
9:    $S_p^r$ : send $\langle vote_p \rangle$ to all processes
10:   $T_p^r$ : if received $> \alpha$ messages with value $v \in V$ then $x_p := v$ else $x_p := v_0$
11:         if received $> E$ messages with value $v \in V$ then $decide_p := v$
12:         $vote_p := ?$
```

to some value in V, ?, and *null*, respectively. At each round r, every process p sends x_p or $vote_p$ to all, depending whether r is odd or even. Then, provided that p receives sufficiently many messages with the same value in V, p updates x_p, $vote_p$, or $decide_p$. The algorithm thus involves three threshold values T, E, and α, which are basic parameters of how it executes.

2.2 Executing HO algorithms

Each process of an HO algorithm executes an infinite sequence of rounds, which are numbered consecutively, starting with round 0. At the beginning of each round r, process p first emits messages to all processes, computed according to the message sending function S_p^r. It then waits for messages to arrive for round r before it executes a state transition according to the next-state function T_p^r, based on its current state and the messages it has just received, and starts a new round.

We define executions with respect to a given collection of initial states (one per process) and a given *receive history* $\mu : \Pi \times \mathbb{N} \times \Pi \to M_\perp$ that specifies, for each pair p, q of processes and each round $r \in \mathbb{N}$, the message $\mu(p, r, q)$ that p receives from q at round r. The initial states, and the receive history determine, for each $p \in \Pi$, the sequence of p's states. Then we define, for each $p \in \Pi$ and round $r \in \mathbb{N}$, the *heard-of set*

$$HO(p, r) = \{q \in \Pi : \mu(p, r, q) \neq \perp\},$$

and the *safe heard-of set*

$$SHO(p, r) = \{q \in \Pi : \mu(p, r, q) = S_q^r(s_q, p)\}$$

where s_q is q's state at the beginning of round r. Both sets specify the discrepancy between *what should be sent* and *what is actually received*. As for the benign case [5], we make no assumption on the reason why $\mu(p, r, q)$ may be different from $S_q^r(s_q, p)$: it may be due to an incorrect sending by q, an incorrect reception by p, or due to the corruption by the link. Obviously, $SHO(p, r) \subseteq HO(p, r)$, and

$HO(p,r) \setminus SHO(p,r)$ is the set of processes q whose messages for p in round r are corrupted.

Assumptions on the underlying system model and communication network, such as the degree of synchronism and the failure model, are formally expressed by *communication predicates* $\mathcal{P} \subseteq (\Pi \times \mathbb{N} \to 2^{\Pi} \times 2^{\Pi})$, and the correctness of an algorithm is asserted relative to a certain communication predicate $\mathcal{P}$. Note that communication predicates may refer to the (S)HO sets at different rounds and can therefore express assumptions about transient faults. As discussed in [5], standard failure models with various degrees of synchronism can be represented in this way: the weaker the communication predicate is, the more freedom the system has to provide heard-of and safe heard-of sets, and the harder it will be to achieve coordination among processes in the corresponding failure model. As an example, the following communication predicate guarantees that no process receives more than α corrupted messages in any round, but that every process receives more than β correct messages at each round:

$$\mathcal{P}_{\alpha,\beta} :: \forall p \in \Pi, \forall r \in \mathbb{N} : |\,HO(p,r) \setminus SHO(p,r)\,| \leq \alpha \ \wedge \ |\,SHO(p,r)\,| > \beta$$

It is worth noting that, with $\alpha = 0$ and $\beta = -1$, only benign faults may occur, i.e., all received messages carry the expected content:

$$\mathcal{P}_{benign} :: \forall p \in \Pi, \forall r \in \mathbb{N} : HO(p,r) = SHO(p,r)$$

2.3 Two models of executions

We define two models of execution, whose relationship will be explored further: the *fine-grained model* and the *coarse-grained model*. Both are based on the notion of (global) *configuration* which is a tuple of process and channel states, one per component. The state of any component c in configuration σ is denoted $\sigma(c)$. An initial configuration is one in which the state of each process p is in $Init_p$, and the state of each channel is the empty set. The two models differ in the nature of the atomic steps which take a configuration to the next one.

Fine-grained executions. Each process p can execute three types of atomic actions that may change the state of p itself and the state of the channels incident on p: the sending of a message, the reception of a message, or an internal action. Only internal actions modify the process state, and process states at the end of round r do not depend on the order in which messages are received at round r. An *event* e consists of the execution of a single action by a process.

In the (classical) fine-grained model of execution, configuration σ' is a *successor configuration* of σ if there exists some event e that takes σ to σ'. By the definition of process specification, the pair (σ, σ') determines a unique event e, and we say that (σ, σ') *corresponds to* e.

A *fine-grained execution* of an algorithm is then defined to be an ω-sequence $\sigma_0\sigma_1\ldots$ of configurations where σ_0 is an initial configuration, σ_{i+1} is a successor configuration of σ_i for all $i \in \mathbb{N}$, and for each $p \in \Pi$ there are infinitely many

$i \in \mathbb{N}$ such that (σ_i, σ_{i+1}) corresponds to some event by p. The last condition specifies a condition of (local) progress for each process; since p can execute a local transition ending round r only if it has sent messages to all processes and has received messages from all $q \in HO(p, r)$, this condition implies the existence of sufficiently many transitions of type message sending and reception. Obviously, each fine-grained execution defines a unique receive history.

Coarse-grained executions. We now define an execution model of HO algorithms that is based on the much coarser abstraction where entire rounds are the unit of atomicity. A *coarse-grained execution* is an ω-sequence $\sigma_0\sigma_1\ldots$ of configurations such that

- σ_0 is an initial configuration, and
- at every step, *all* processes make a transition according to their next-state function and messages they have received: there exists a receive history μ such that for all $p \in \Pi$ and all $r \in \mathbb{N}$,

$$\sigma_{r+1}(p) = T_p^r\big(\sigma_r(p), \mu_p^r\big) \quad \text{where} \quad \mu_p^r = \big(q \in \Pi \mapsto \mu(p, r, q)\big).$$

In words, the state $\sigma_{r+1}(p)$ is computed according to the next-state function T_p^r from the state $\sigma_r(p)$, and the messages that p receives at round r. A step of a coarse-grained execution thus encapsulates a move by each process. Channels are considered empty in each configuration σ_r of such a round-by-round execution since messages can be received only in the rounds for which they have been sent.

2.4 A reduction theorem

We now present a basic theorem, which asserts that in our model, the fine-grained and coarse-grained execution semantics are indistinguishable from the point of view of any process. Given a (fine-grained or coarse-grained) execution ρ and a process $q \in \Pi$, we define the *q-view* ρ^q of ρ for process q as the sequence of q's local states in ρ. More precisely, for a fine-grained or a coarse-grained execution $\rho = \rho_0\rho_1\ldots$, the q-view is simply

$$\rho^q = \rho_0(q)\, \rho_1(q)\, \ldots$$

Any two executions ρ_1 and ρ_2 can be compared with respect to the views that they generate for the processes in Π. We say that two executions ρ_1 and ρ_2 are *q-equivalent* (for $q \in \Pi$) if $\rho_1^q \simeq \rho_2^q$ where $\simeq$ denotes *stuttering equivalence* [12], i.e. if their q-views agree up to finite repetitions of states. We call ρ_1 and ρ_2 *locally equivalent*, written $\rho_1 \approx \rho_2$, if they are q-equivalent for all $q \in \Pi$.

The following theorem asserts that fine-grained and coarse-grained executions generate the same set of local views.

Theorem 1. *For any fine-grained execution ξ of an HO algorithm, there exists a coarse-grained execution σ of the same algorithm for the same receive history such that $\sigma \approx \xi$, and vice-versa.*

The proof of this theorem given in [4] for benign faults extends to the more general context of malicious faults since it is based on some commutativity properties (among events) which do not depend on the fault model.

Theorem 1 can be used to verify linear-time properties of HO algorithms that are expressed in terms of local views of processes, and that are insensitive to specific interleavings. Formally, we say that a property P is *local* if for any (coarse- or fine-grained) executions ρ_1 and ρ_2 such that $\rho_1 \approx \rho_2$ we have $\rho_1 \models P$ iff $\rho_2 \models P$, i.e., ρ_1 satisfies P iff ρ_2 does. As an immediate consequence of Theorem 1, we obtain the following corollary:

Corollary 2. *If P is a local property, then $\sigma \models P$ holds for all coarse-grained executions σ of an algorithm if and only if $\xi \models P$ also holds for all fine-grained executions ξ of the same algorithm.*

Having to verify a given property just for all coarse-grained executions represents a significant reduction because coarse-grained executions afford a simpler representation of the system state (channels are all empty), and because fewer interleavings of events and fewer (types of) transitions must be considered.

We now indicate a sufficient syntactic criterion for determining when a formula of LTL-X, i.e., linear-time temporal logic without the next-time operator expresses a local property.[5] We assume that the set of state variables that appear in formulas is of the form $\mathcal{V} = \bigcup_{p \in \Pi} \mathcal{V}_p$ where $\mathcal{V}_p \cap \mathcal{V}_q = \emptyset$ for different processes $p \neq q$, and such that any state $s \in \Sigma_p$ of a process $p \in \Pi$ uniquely determines the values of the variables in $\mathcal{V}_p$.

We say that a formula φ is a *p-formula*, for $p \in \Pi$, if it contains only state variables from $\mathcal{V}_p$. It is easy to see that p-formulas are local properties, as are first-order combinations of p-formulas, for possibly different processes $p \in \Pi$. However, temporal combinations of p-formulas are in general not local because they can express the simultaneity of local states of different processes, or assert temporal relations between states of processes [3].

2.5 The Consensus problem

In this paper, we concentrate on the well-known agreement problem, called *Consensus*, regarded as the fundamental problem that must be solved to implement a fault-tolerant system by replication. We assume that the state variables $\mathcal{V}_p$ include variables x_p and $decide_p$. The intuitive idea is that at the beginning of an execution the variable x_p holds the initial value of process p. Variable $decide_p$, initially *null*, represents the decision taken by process p in the sense that $decide_p$ is updated to the value $v \neq null$ when process p decides value v.

Consensus is specified as the conjunction of the following formulas of LTL-X, which are all local according to the criterion introduced in Section 2.4.

Integrity. Any decision value must be among the initial values.

$$\forall v : v \neq null \wedge \left(\bigvee_{p \in \Pi} \Diamond(decide_p = v) \right) \Rightarrow \bigvee_{q \in \Pi} x_q = v.$$

[5] LTL-X formulas are stuttering invariant [17].

Irrevocability. A process that has decided must never change its decision value.

$$\forall v : v \neq \mathit{null} \Rightarrow \Box\big(\mathit{decide}_p = v \Rightarrow \Box(\mathit{decide}_p = v)\big)$$

Agreement. The agreement property requires that if any two processes decide, they decide on the same value.

$$\begin{aligned}\forall v, w : \quad & v \neq \mathit{null} \wedge w \neq \mathit{null} \\ & \wedge \bigvee_{p,q \in \Pi} \big(\Diamond(\mathit{decide}_p = v) \wedge \Diamond(\mathit{decide}_q = w)\big) \\ \Rightarrow\ & v = w.\end{aligned}$$

Termination. The preceding properties are all safety properties; the sole liveness property requires that all processes eventually decide.

$$\Diamond(\mathit{decide}_p \neq \mathit{null}).$$

Contrary to classical approaches, the HO model does not flag processes as being faulty [5], and the above Consensus specification makes no exception: all processes must decide the same initial value of some process. Such a strong specification is not trivially unsolvable. Indeed, since there is no deviation from the next-state functions, processes may not take arbitrary steps, such as deciding arbitrary values. In the following, we formally prove that three HO algorithms solve the above strong Consensus specification under suitable communication predicates, thus demonstrating how the algorithms prevent every process from being contaminated by corrupted messages.

3 Representing the Heard-Of model in Isabelle

The uniform presentation of algorithms in the HO model by message-sending and next-state functions, and of system models by communication predicates, is attractive for formal verification, and the ability to verify these algorithms over coarse-grained executions significantly reduces the state space. Indeed, several algorithms solving Consensus under benign faults that were presented in [5] have been verified (for fixed-size instances) using model checking techniques [19,20,3]. Malicious faults, however, may introduce an infinite number of arbitrary values, making model checking prohibitive. We now describe our encoding of the HO model in the interactive proof assistant Isabelle/HOL [16], which allows us to verify arbitrary instances of algorithms.

3.1 Representing Algorithms and Communication Predicates

In the Isabelle model, the set Π of processes is represented by a type variable $'\mathit{proc}$. We will constrain $'\mathit{proc}$ below so that it can only be instantiated by types with finitely many values. Similarly, the type variables $'\mathit{pst}$ and $'\mathit{msg}$ serve to represent the sets of local process states and messages, and corresponding concrete types will be defined for particular algorithms. Assignments of HO (or SHO) sets to processes are of type

type_synonym $'proc\ HO\ =\ 'proc \rightarrow 'proc\ set$,

i.e., functions from processes to sets of processes. The computational model is represented using Isabelle's *locale* mechanism: models of concrete algorithms are obtained as instances of the locale, whereas generic properties of the HO model can be proved within the locale and will be inherited by every instance.

```
locale $SHOAlgorithm$ =
fixes
  $initState :: [('proc :: finite), 'pst] \rightarrow bool$ and
  $sendMsg :: [nat, 'proc, 'proc, 'pst] \rightarrow 'msg$ and
  $nextState :: [nat, 'proc, 'pst, ('proc \rightharpoonup 'msg), 'pst] \rightarrow bool$ and
  $commPerRd :: ['proc\ HO, 'proc\ HO] \rightarrow bool$
  $commGlobal :: [nat \rightarrow 'proc\ HO, nat \rightarrow 'proc\ HO] \rightarrow bool$
```

The interface of the Isabelle locale representing HO algorithms is shown above. It takes five parameters: *initState* represents a predicate (boolean function) such that *initState* $p\ s$ is true iff s is an initial state of process p. (In Isabelle/HOL, function application is denoted by juxtaposition.) Similarly, the parameters *sendMsg* and *nextState* formally represent the message-sending and next-state functions S_p^r and T_p^r. For convenience, the communication predicate associated with the algorithm is split into a predicate *commPerRd*, which is evaluated at every round, and a predicate *commGlobal*, evaluated globally over ω-sequences of HO and SHO collections (cf. the definition of *SHORun* below).

3.2 Defining coarse-grained executions

By Theorem 1, it is enough to verify Consensus algorithms over coarse-grained executions only, and we represent just these in Isabelle. As explained in Section 2.3, a coarse-grained execution is an ω-sequence of configurations, each of which is a function of type $'proc \rightarrow 'pst$. Since channels are empty in every configuration of a coarse-grained execution, they need not be modeled.

In an initial configuration, every process is in an initial state:

definition *initConfig* **where** $initConfig\ cfg\ \equiv\ \forall p.\ initState\ p\ (cfg\ p)$.

Configuration cfg' is a possible successor of configuration cfg at round r of an execution, given assignments HO and SHO of HO (resp., SHO) sets if for every process p there exists a vector μ of incoming messages compatible with HO and SHO such that the states of p before and after the transition and the message vector μ satisfy the *nextState* predicate.

definition *nextConfig* **where** $nextConfig\ r\ cfg\ HO\ SHO\ cfg'\ \equiv$
  $\forall p.\ \exists m \in msgsVectors\ r\ p\ cfg\ HO\ SHO.\ nextState\ r\ p\ (cfg\ p)\ m\ (cfg'\ p)$

where the set of possible message vectors is defined as

definition *msgsVectors* **where** $msgsVectors\ r\ p\ cfg\ HO\ SHO \equiv$
  $\{m.\ (\forall q.\ q \in SHO\ p\ \longleftrightarrow\ m\ q\ =\ Some\ (sendMsg\ r\ q\ p\ (cfg\ q))) \wedge$
    $(\forall q.\ q \in HO\ p\ \longleftrightarrow\ m\ q\ \neq\ None)\}$

In words, vector m is compatible with HO and SHO if for all processes q in p's HO set, $m\ q \neq None$,[6] and moreover, for q in p's SHO set, $m\ q$ equals the message that q sent to p for the current round according to the *sendMsg* function. Because the value $m\ q$ is unconstrained for processes $q \in (HO\ p) \setminus (SHO\ p)$, any type-correct value may be received from these processes.

We now define a predicate characterizing executions of an HO algorithm, relative to collections *HOs* and *SHOs*, as infinite sequences of configurations $c_0 c_1 \ldots$ where c_0 is an initial configuration, for all r, configuration c_{r+1} is a successor of c_r, and the Heard-Of collections satisfy the communication predicate.

```
definition SHORun where SHORun rho HOs SHOs ≡
    initConfig (rho 0)
  ∧ ∀r. nextConfig r (rho r) (HOs r) (SHOs r) (rho (Suc r))
  ∧ ∀r. commPerRd (HOs r) (SHOs r)
  ∧ commGlobal HOs SHOs
```

4 Verifying Concrete Algorithms

We outline how different Consensus algorithms can be represented and verified as instances of the locale *SHOAlgorithm* introduced previously.

4.1 Modeling and verifying non-synchronous algorithms in Isabelle

Biely et al. [2] introduce two non-synchronous Consensus algorithms tolerating malicious faults: the $\mathcal{U}_{T,E,\alpha}$ algorithm introduced in Section 2.1, and a one-round algorithm called $\mathcal{A}_{T,E,\alpha}$. We instantiate the generic Isabelle locale *SHOAlgorithm* for these algorithms and verify their correctness.

Figure 1 shows the representation of $\mathcal{U}_{T,E,\alpha}$ in Isabelle. We begin by declaring an anonymous type *Proc* of processes that is assumed to be finite. We then introduce the parameters T, E and α and indicate the assumed relations between them. Process states are represented as a record *pstate*, and messages are similarly represented as a data type *msg*. The definitions of the initial state predicate and the message-sending function are straightforward. Observe that the x field of initial states is left unconstrained, hence the initial value of processes may be any type-correct value. The definition of the next-state relation is split into two cases depending on the round number being even or odd.

The communication predicate for the $\mathcal{U}_{T,E,\alpha}$ algorithm, as specified in [2], is defined as the conjunction of the two following predicates:

```
definition Ute_commPerRd where Ute_commPerRd HO SHO ≡
  ∀p.   card ((HO p) \ (SHO p)) ≤ alpha
      ∧ card (SHO p) > N + 2 * alpha − E − 1
      ∧ card (SHO p) > T
```

[6] Isabelle's *None* corresponds to the pseudo-value $\bot \notin M$ introduced in Section 2.

```
typedecl Proc
axiomatization where procFinite : finite (UNIV :: Proc set)
abbreviation N ≡ card (UNIV :: Proc set) – cardinality of the set of processes
axiomatization T :: nat and E :: nat and α :: nat where
  E − α > N ÷ 2 and T − α > N ÷ 2 and E < N and T < N
consts defaultv :: ′val
record ′val pstate =
  x :: ′val
  vote :: ′val option
  decide :: ′val option
datatype ′val msg =
    Val ′val
  | Vote ′val option
definition step where step r ≡ r mod 2
definition initState where
initState p st ≡ vote st = None ∧ decide st = None
definition sendMsg where
sendMsg r ≡ if step r = 0 then Val(x st) else Vote(vote st)
definition next0 where
next0 r p st msgs st′ ≡
  (∃v. card{q. msgs q = Some (Val v)} > T ∧ st′ = st(|vote := Some v|))
∨ ((¬∃v. card{q. msgs q = Some (Val v)} > T) ∧ st′ = st(|vote := None|))
definition next1 where
next1 r p st msgs st′ ≡
  vote st′ = None
∧ (∃v. card {q. msgs q = Some (Vote (Some v))} > α ∧ x st′ = v) ∨
  ((¬∃v. card {q. msgs q = Some (Vote (Some v))} > α) ∧ x st′ = defaultv)
∧ (∃v. card {q. msgs q = Some (Vote (Some v))} > E ∧ decide st′ = Some v) ∨
  ((¬∃v. card {q. msgs q = Some (Vote (Some v))} > E) ∧ decide st′ = decide st)
definition nextState where nextState r ≡ if step r = 0 then next0 r else next1 r
```

Fig. 1. Isabelle representation of the $\mathcal{U}_{T,E,\alpha}$ algorithm.

```
definition phase where phase r ≡ r div 2
definition Ute_commGlobal where
Ute_commGlobal HOs SHOs ≡ ∀r. ∃φ > phase r. ∃r′. ∃π. ∀p.
    r′ = 2 ∗ φ + 1
  ∧ π = HOs r′ p ∧ π = SHOs r′ p
  ∧ card (SHOs (r′ + 1) p) > T ∧ card (SHOs (r′ + 2) p) > E
```

The "round-by-round" predicate *Ute_commPerRd* is just the predicate $\mathcal{P}_{\alpha,\beta}$ introduced in Section 2.2 for $\beta = \max(N + 2\alpha - E - 1, T)$. It ensures the safety properties of the algorithm. The "global" predicate *Ute_commGlobal* is used to prove termination. It requires that there are infinitely many phases ϕ such that

(1) the HO and SHO processes for all processes are identical in the second step of phase ϕ and (2) the cardinality of the SHO sets for all processes exceeds T (resp., E) in the first (resp., second) step of the subsequent phase.

Finally, we declare $\mathcal{U}_{T,E,\alpha}$ to be an instance of the generic locale for SHO algorithms described in Section 3. This is achieved by the following Isabelle command, which instantiates the parameters of the locale *SHOAlgorithm* by the operators defined for the $\mathcal{U}_{T,E,\alpha}$ algorithm.

> **interpretation** *SHOAlgorithm*
> *initState sendMsg nextStateUte_commPerRd Ute_commGlobal*
> **by** *unfold_locales*

We have used Isabelle to formally prove the correctness of $\mathcal{U}_{T,E,\alpha}$ (for an arbitrary number of processes). The proof is based on the informal proof given in [2], which we have split into a sequence of lemmas. Our main contribution is that we have been able to carry out a formal proof of a non-synchronous algorithm tolerating malicious faults with reasonable effort (the overall size of the verbose Isar proof script is under 900 lines, including comments). This would not have been possible without the high level of abstraction provided by the HO model. Based on the machine-checked proof, we can confidently assert the correctness of $\mathcal{U}_{T,E,\alpha}$.

Theorem 3. *The $\mathcal{U}_{T,E,\alpha}$ algorithm solves Consensus under the communication predicate specified by Ute_commPerRd and Ute_commGlobal.*

The $\mathcal{A}_{T,E,\alpha}$ algorithm, introduced together with $\mathcal{U}_{T,E,\alpha}$ in [2], is represented in Isabelle in an analogous way. It is a one-round HO algorithm in which a decision is taken immediately if a sufficient number of identical messages is received.

$\mathcal{U}_{T,E,\alpha}$ and $\mathcal{A}_{T,E,\alpha}$ differ in the algorithmic structure and rely on different communication predicates. In particular, $\mathcal{A}_{T,E,\alpha}$ has a simpler "round-by-round" predicate but a more elaborate "global" predicate:

> **definition** *Ate_commPerRd* **where**
> $\mathit{Ate\_commPerRd}\ \mathit{HO}\ \mathit{SHO}\ \equiv\ \forall p.\ \mathit{card}\ ((\mathit{HO}\ p) \setminus (\mathit{SHO}\ p)) \leq \alpha$
>
> **definition** *Ate_commGlobal* **where**
> $\mathit{Ate\_commGlobal}\ \mathit{HOs}\ \mathit{SHOs} \equiv$

$$
\begin{aligned}
& \forall r\ p.\ \exists r' > r.\ \mathit{card}\ (\mathit{HOs}\ r'\ p) > T \\
\wedge\ & \forall r\ p.\ \exists r' > r.\ \mathit{card}\ (\mathit{SHOs}\ r'\ p) > E \\
\wedge\ & \forall r.\ \exists r' > r.\ \exists \pi_1\ \pi_2.\ \mathit{card}\ \pi_1 > E - \alpha\ \wedge\ \mathit{card}\ \pi_2 > T\ \wedge \\
& \qquad\qquad \forall p \in \pi_1.\ \mathit{HOs}\ r'\ p = \pi_2 \wedge \mathit{SHOs}\ r'\ p = \pi_2
\end{aligned}
$$

These two predicates require that:

- the number of corrupted messages in each round is never greater than α,
- T and E thresholds are passed infinitely often, for every process,
- infinitely often, there exists a sufficiently big set π_1 of processes whose HO and SHO sets are all identical to some set π_2 that passes the T threshold.

We have again formally proved in Isabelle the correctness of the $\mathcal{A}_{T,E,\alpha}$ algorithm under this communication predicate. Despite the differences in the algorithms and the predicates, the effort required for carrying out the two proofs is quite comparable.

4.2 Verifying a synchronous algorithm

Our third case study is the well-known EIGByz_f [1,15] algorithm, which decides after $f+1$ rounds and is designed for synchronous system models. Encoding EIGByz_f in the HO model is straightforward. We have proved in Isabelle that the algorithm solves Consensus under the communication predicate defined by the round-by-round predicate $\mathcal{R}(r)$ and the global predicate $\mathcal{G}$, defined as

$$\mathcal{R}(r) :: \left| \bigcap_{p \in \Pi} \mathit{SHO}(p,r) \right| > \frac{N+f}{2} \qquad \mathcal{G} :: \left| \bigcap_{p \in \Pi, r \in \mathbb{N}} \mathit{SHO}(p,r) \right| \geq N-f.$$

EIGByz_f was designed for synchronous systems with reliable links and at most f faulty processes. In such a system, every process receives the correct message from at least the non-faulty processes at every round, and therefore the predicate $\mathcal{G}$ is satisfied. The standard correctness proof for EIGByz_f [1,15] assumes that $N > 3f$, and therefore $N - f > \frac{N+f}{2}$. Since moreover, for any $r \in \mathbb{N}$, we obviously have

$$\left(\bigcap_{p \in \Pi, r' \in \mathbb{N}} \mathit{SHO}(p,r') \right) \subseteq \left(\bigcap_{p \in \Pi} \mathit{SHO}(p,r) \right),$$

it follows that any execution of EIGByz_f where $N > 3f$ also satisfies $\forall r : \mathcal{R}(r)$. The standard correctness hypotheses thus imply our communication predicates.

However, our proof shows that EIGByz_f can indeed tolerate more transient faults than the standard bound can express. For example, consider the case where $N = 5$ and $f = 2$. Our predicates are satisfied in executions where two processes exhibit transient faults, but never fail simultaneously. Indeed, in such an execution, every process receives four correct messages at every round r, hence $\mathcal{R}(r)$ holds. Also, $\mathcal{G}$ is satisfied because there are three processes from which every process receives the correct messages at all rounds. By our correctness proof, it follows that EIGByz_f then achieves Consensus, unlike what one could expect from the standard correctness predicate. This observation underlines the interest of expressing assumptions about transient faults, as in the HO model.

Finally, it is worth noting that, unlike $\mathcal{U}_{T,E,\alpha}$ and $\mathcal{A}_{T,E,\alpha}$, no assumption on the sets $\mathit{HO}(p,r) \setminus \mathit{SHO}(p,r)$ is ever required for the correctness of EIGByz_f: our predicates for EIGByz_f are expressed in terms of the SHO sets only. In other words, the conditions ensuring the correctness of EIGByz_f only specify how links must be both safe and live. However, contrasting with $\mathcal{U}_{T,E,\alpha}$ and $\mathcal{A}_{T,E,\alpha}$, which are correct under round-by-round conditions, EIGByz_f requires a global predicate on the sequence of rounds (namely, $\mathcal{G}$). Such global predicates on just the safe heard-of sets actually correspond to what is classically called the "synchronous approach".

5 Related and Future Work

Despite the crucial need for rigorous correctness proofs, especially in the context of fault-tolerance, few distributed algorithms have been formally verified. Moreover, formal verification of these algorithms mostly concerns benign faults (e.g., [19,20,3]) or even assumes that no fault may occur (e.g., [9,10]). We are aware of a few contributions addressing formal verification of distributed algorithms in the context of malicious faults, but all of them consider perfectly synchronous systems (e.g., [14,18]), with the exception of recent work by Lamport [13]. Lamport gives a formal safety proof of a variant on the Paxos algorithm that tolerates Byzantine faults (i.e., processes may deviate from their transition function). This algorithm, like most non-synchronous Consensus algorithms designed to tolerate malicious faults, assumes that processes can *authenticate* their communications, for example based on the use of *digital signatures*. A digital signature for process p is an extra information that p can add to any of its outgoing messages in order to prove that the message really originated at p, even if it has been relayed by several other processes. This informal description actually refers to the semantics of messages, and as pointed out by Lynch [15], no formal definition of malicious faults with authentication has ever been given. We therefore contend that relying on properties of authenticated messages represents a gap in the proof of an algorithm that uses them. Since neither $\mathcal{A}_{T,E,\alpha}$, $\mathcal{U}_{T,E,\alpha}$, nor *EIGByz* need authentication, we have been able to formally verify each of these Consensus algorithms in the context of malicious (communication) faults.

The Heard-Of model, in which we have carried out our work, lets us describe different algorithms, designed for different communication and fault models, in a uniform way. We verified three Consensus algorithms ($\mathcal{U}_{T,E,\alpha}$, $\mathcal{A}_{T,E,\alpha}$ and EIGByz_f) that tolerate malicious faults in our encoding of the HO model in the interactive proof assistant Isabelle/HOL, and we are confident that other algorithms can be verified with similar effort. Our proofs are at least an order of magnitude shorter than proofs for comparable algorithms under benign faults, such as the correctness proof for DiskPaxos [11] in Isabelle/HOL. This difference is essentially due to the higher level of abstraction gained through the use of the HO model, which allows us to consider only coarse-grained executions.

In future work, we would like to extend the framework to also cover malicious (Byzantine) transition faults. Although transition faults are indistinguishable from malicious communication faults to other processes in the network, the definition of Consensus has to be adapted, since no requirements can be placed on faulty processes. We are also interested in the representation of and formal reasoning about properties such as authentication, atomic broadcast or weak-interactive consistency in the HO model.

References

1. A. Bar-noy, D. Dolev, C. Dwork, and H. R. Strong. Shifting gears: Changing algorithms on the fly to expedite Byzantine agreement. In *Information and Computation*, pages 42–51, 1987.

2. M. Biely, J. Widder, B. Charron-Bost, A. Gaillard, M. Hutle, and A. Schiper. Tolerating corrupted communication. In *Proc. 26th Annual ACM Symposium on Principles of Distributed Computing*, PODC '07, pages 244–253, New York, NY, USA, 2007. ACM.
3. M. Chaouch-Saad, B. Charron-Bost, and S. Merz. A reduction theorem for the verification of round-based distributed algorithms. In O. Bournez, editor, *3rd Workshop on Reachability Problems (RP'09)*, volume 5797 of *LNCS*, pages 93–106, Palaiseau, France, 2009. Springer.
4. B. Charron-Bost and S. Merz. Formal verification of a Consensus algorithm in the Heard-Of model. *Int. J. Software and Informatics*, 3(2-3):273–303, 2009.
5. B. Charron-Bost and A. Schiper. The Heard-Of model: Computing in distributed systems with benign failures. *Distributed Computing*, 2009.
6. C. Dwork, N. A. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, Apr. 1988.
7. T. Elrad and N. Francez. Decomposition of distributed programs into communication-closed layers. *Science Comp. Prog.*, 2(3), Apr. 1982.
8. M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, Apr. 1985.
9. C. Georgiou, N. A. Lynch, P. Mavrommatis, and J. A. Tauber. Automated implementation of complex distributed algorithms specified in the IOA language. *Intl. J. Software Tools for Technology Transfer*, 11(2):153–171, 2009.
10. W. H. Hesselink. The verified incremental design of a distributed spanning tree algorithm: Extended abstract. *Formal Asp. Comput.*, 11(1):45–55, 1999.
11. M. Jaskelioff and S. Merz. Proving the correctness of Disk Paxos. Archive of Formal Proofs, `http://afp.sourceforge.net/entries/DiskPaxos.shtml`, 2005.
12. L. Lamport. What good is temporal logic? In R. E. A. Mason, editor, *Information Processing 83: Proceedings of the IFIP 9th World Congress*, pages 657–668, Paris, Sept. 1983. IFIP, North-Holland.
13. L. Lamport. Byzantining Paxos by refinement. Technical report, Microsoft Research, Dec. 2010.
14. L. Lamport and S. Merz. Specifying and verifying fault-tolerant systems. In *3rd Intl. Symp. Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'94)*, volume 863 of *LNCS*, pages 41–76, Lübeck, Germany, 1994. Springer.
15. N. A. Lynch. *Distributed Algorithms.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.
16. T. Nipkow, L. Paulson, and M. Wenzel. *Isabelle/HOL. A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.
17. D. Peled and T. Wilke. Stutter-invariant temporal properties are expressible without the next-time operator. *Inf. Proc. Letters*, 63(5):243–246, 1997.
18. U. Schmid, B. Weiss, and J. M. Rushby. Formally verified byzantine agreement in presence of link faults. In *22nd Intl. Conf. Distributed Computing Systems (ICDCS'02)*, pages 608–616, Vienna, Austria, 2002. IEEE Comp. Society.
19. T. Tsuchiya and A. Schiper. Model checking of consensus algorithms. In *26th IEEE Symp. Reliable Distributed Systems (SRDS 2007)*, pages 137–148, Beijing, China, 2007. IEEE Comp. Society.
20. T. Tsuchiya and A. Schiper. Using bounded model checking to verify consensus algorithms. In G. Taubenfeld, editor, *22nd Intl. Symp. Dist. Comp. (DISC 2008)*, volume 5218 of *LNCS*, pages 466–480, Arcachon, France, 2008. Springer.