A Reduction Theorem for the Verification of Round-Based Distributed Algorithms*

Mouna Chaouch-Saad[1], Bernadette Charron-Bost[2], and Stephan Merz[3]

[1] Faculté des Sciences, Tunis, Tunisia, Mouna.Saad@fst.rnu.tn
[2] CNRS & LIX, Palaiseau, France, charron@lix.polytechnique.fr
[3] INRIA Nancy & LORIA, Nancy, France, Stephan.Merz@loria.fr

Abstract. We consider the verification of algorithms expressed in the Heard-Of Model, a round-based computational model for fault-tolerant distributed computing. Rounds in this model are communication-closed, and we show that every execution recording individual events corresponds to a coarser-grained execution based on global rounds such that the local views of all processes are identical in the two executions. This result helps us to substantially mitigate state-space explosion and verify Consensus algorithms using standard model checking techniques.

1 Introduction

Distributed algorithms are often quite subtle, both in the way they operate and in the assumptions they make. Formal verification is therefore crucial in distributed computing. Unfortunately, due to their asynchronous nature, distributed algorithms almost invariably give rise to state-space explosion, severely limiting the applicability of model checking techniques. This is particularly true for fault-tolerant algorithms whose correctness relies on elaborate failure hypotheses.

The key to overcome this problem is to make use of the inherently non-sequential nature of distributed executions and to exploit the causality relation [4] between events of the execution in order to reduce the number of executions that have to be analyzed. In this paper we study reductions that hold for distributed algorithms that are structured in *rounds*: each process first sends messages and receives messages sent for the round, and finally makes a local state transition. More specifically, Charron-Bost and Schiper [1] recently proposed the Heard-Of (HO) model, a round-based model for fault-tolerant distributed computing. We formally prove that for verifying interesting properties for algorithms in this model, it suffices to model executions as infinite sequences of global rounds. Moreover, rounds in the HO model are communication closed, hence the medium of communication can be considered as empty at the end of each round, and the overall state can be represented just by the collection of local states for each process. These two observations induce reductions that go beyond well-known techniques of partial-order reduction [3, 11], and that can

* We gratefully acknowledge support by CMCU (Comité Mixte de Coopération Universitaire) project DEFI Utique.

indeed justify reductions from infinite-state to finite-state models. We validate our approach by verifying finite instances of some of the Consensus algorithms proposed in [1], using a standard explicit-state model checker.

The paper is organised as follows: Section 2 provides a short introduction to the HO model and defines executions. Section 3 proves the reduction theorem that establishes a close correspondence between the two representations of executions. We present in Section 4 some experiments on model checking Consensus algorithms in the HO model. Section 5 discusses related work and concludes.

2 The Heard-Of Model for Distributed Algorithms

Computations in the HO model are organized in rounds, in which each process exchanges messages, takes a step, and then proceeds to the next round. Without any specific synchronization assumptions, processes execute rounds at their own pace. In particular, the difference between the numbers of rounds that two different processes are executing at any given moment may be arbitrarily large. Rounds are communication-closed layers in the terminology of Elrad and Francez [2]: messages are valid only for the round they were sent in. Thus the model generalizes the classical notion of synchronized rounds developed for synchronous systems [7].[4]

2.1 A Round-Based Computational Model

We suppose that we have a finite, non-empty set Π of process identifiers[5] 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 w.l.o.g. that each process sends some message to every process in Π, in each round. We denote the cardinality of Π by $N > 0$, let $\perp \notin M$ be a placeholder indicating that no message has (yet) been received, and write $M_\perp = M \cup \{\perp\}$. To each p in Π, we associate a *process specification* $Proc_p = (\Sigma_p, s_{0,p}, S_p, T_p)$ whose components are the following:

- Σ_p is the set of p's *states*, and $s_{0,p} \in \Sigma_p$ is the *initial state* of process p,
- $S_p : \mathbb{N} \times \Sigma_p \times \Pi \to M$ is the *message sending function* such that $S_p(r, s, q)$ denotes the message that p sends to q at round r, given the state s of p, and
- $T_p : \mathbb{N} \times \Sigma_p \times M_\perp^\Pi \to \Sigma_p$ is the *next-state function*: $T_p(r, s, \mu)$ yields the successor state of process p at round r, given its current state s and the partial vector $\mu = (\mu_q)_{q \in \Pi}$ of messages where μ_q indicates the message that p received from q at round r, or $\perp$ if no message was received.

The collection of process specifications $Proc_p$ is called an *algorithm* on Π.

[4] Communication-closedness can be ensured in asynchronous settings by buffering messages which are early, and by discarding messages which are late.

[5] When there is no risk of confusion, we simply speak of *processes* in Π.

2.2 Executions of 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. It then waits for messages to arrive for round r before it executes a state transition according to the next-state function T_p, based on its current state and the vector of messages received, and starts a new round. The *heard-of set* $HO(p, r)$ for p at round r is the set of processes from which p receives a message at round r.

Formally, we define executions with respect to a given *HO collection*

$$HO : \Pi \times \mathbb{N} \to 2^{\Pi}$$

that specifies, for each $p \in \Pi$ and round $r \in \mathbb{N}$, the heard-of set $HO(p, r)$. Process p proceeds to round $r+1$ when it has received messages from the processes in $HO(p, r)$. We make HO collections an explicit parameter of the definition of executions because algorithms are unlikely to work under completely arbitrary HO collections. 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* $\mathcal{P} \subseteq (\Pi \times \mathbb{N} \to 2^{\Pi})$, and the correctness of an algorithm is asserted relative to a certain communication predicate $\mathcal{P}$. As discussed in [1], 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 sets, and the harder it will be to achieve coordination among processes in the corresponding failure model.

Fine-grained executions. We define two models of execution, whose relationship will be explored in Section 3. The *fine-grained model* represents events of individual processes and the way they interleave, and so faithfully models the asynchronous execution of distributed algorithms. A *configuration* of an algorithm is a tuple $(rd, st, sent, rcv, msgs)$:

- rd, st, $sent$ and rcv are arrays indexed by processes where $rd(p) \in \mathbb{N}$, $st(p) \in \Sigma_p$, $sent \subseteq \Pi$, and $rcv(p) \in M_{\perp}^{\Pi}$ denote, for process p, its current round, its local state, the set of processes to which p has sent messages in the current round, and the partial vector of messages received;
- $msgs \subseteq \Pi \times \mathbb{N} \times \Pi \times M$ represents the messages in transit: $(p, r, q, m) \in msgs$ if p sent message m at round r to q, but q has not yet received m.

The algorithm starts in the *initial configuration* c where $c.rd(p) = 0$, $c.st(p) = s_{0,p}$, $sent(p) = \emptyset$, and $c.rcv(p) = (q \in \Pi \mapsto \perp)$ for all $p \in \Pi$, and where no messages are in transit, i.e. $c.msgs = \emptyset$.

Configuration c' is a *successor configuration* of c if one of the following cases holds:

– Transition (c, c') represents process p *sending a message* to process q:

$$\begin{array}{lll} q \in \Pi \setminus c.sent(p), & c'.sent = c.sent\big(p := c.sent(p) \cup \{q\}\big), & \\ c'.rd = c.rd, & c'.st = c.st, & c'.rcv = c.rcv, \\ c'.msgs = c.msgs \cup \big\{\big(p, c.rd(p), q, S_p(c.rd(p), c.st(p), q)\big)\big\} & & \end{array}$$

The transition is enabled if p has not yet sent a message to q during its current round. The effect of the transition is to add the message (computed according to function S_p) to the set of messages in transit and to record the fact that the message has been sent in the *sent* field of configuration c' for process p.

– Transition (c, c') represents a *message reception*: there exist $p, q \in \Pi$ and $m \in M$ such that

$$\begin{array}{ll} q \in HO(p, c.rd(p)), & (q, c.rd(p), p, m) \in c.msgs, \\ c'.msgs = c.msgs \setminus \{(q, c.rd(p), p, m)\}, & c'.rd = c.rd, \quad c'.st = c.st, \\ c'.rcv = c.rcv\big(p := c.rcv(p)(q := m)\big), & c'.sent = c.sent. \end{array}$$

The transition is enabled if q is a member of p's heard-of set for p's current round and message m is in transit from q to p for that round. The effect of the transition is to transfer the message from the set of messages in transit to the vector of messages received by p, while the rounds, process states, and *sent* fields remain unchanged.

– Transition (c, c') is a *local transition* of some process $p \in \Pi$:

$$\begin{array}{ll} c.sent(p) = \Pi, & \operatorname{dom} c.rcv(p) = HO(p, c.rd(p)), \\ c'.rd = c.rd\big(p := c.rd(p) + 1\big), & \\ c'.st = c.st\big(p := T_p(c.rd(p), c.st(p), c.rcv(p))\big), & \\ c'.sent = c.sent(p := \emptyset), & c'.rcv = c.rcv\big(p := (q \in \Pi \mapsto \bot)\big), \\ c'.msgs = c.msgs & \end{array}$$

where $\operatorname{dom} c.rcv(p)$ denotes the set $\{q \in \Pi : c.rcv(p, q) \neq \bot\}$.[6] A local transition of p is enabled when p has sent messages for the current round to all processes and has received messages from precisely the processes specified by the HO collection for its current round. The configuration c' is obtained by incrementing the round number of process p, updating its local state according to the next-state function T_p, and resetting the *sent* and *rcv* fields for process p.

A *fine-grained execution* is an ω-sequence $c_0 c_1 \ldots$ of configurations such that c_0 is the initial configuration, c_{i+1} is a successor configuration of c_i for all $i \in \mathbb{N}$, and for each $p \in \Pi$ there are infinitely many $i \in \mathbb{N}$ such that (c_i, c_{i+1}) is a local transition of 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 also implies the existence of sufficiently many transitions of type message sending and reception.

[6] We identify a function $f : A \times B \to C$ and its "curried" version $f_c : A \to (B \to C)$.

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. Thus, a *coarse-grained execution* is an ω-sequence $\sigma_0\sigma_1\ldots$ where each σ_i is an array of local states $\sigma_i(p) \in \Sigma_p$ indexed by $p \in \Pi$, such that

- $\sigma_0(p) = s_{0,p}$ is the initial state of p, for all $p \in \Pi$, and
- at every step, *all* processes make a transition according to their next-state function and the HO collection: for all $p \in \Pi$ and all $r \in \mathbb{N}$,

$$\begin{array}{l} \sigma_{r+1}(p) = T_p\big(r, \sigma_r(p), rcvd(p,r)\big) \\ \text{where} \quad rcvd(p,r) = \big(q \in \Pi \mapsto \left\{ \begin{array}{ll} S_q(r, \sigma_r(q), p) & \text{if } q \in HO(p,r) \\ \bot & \text{otherwise} \end{array} \right\}\big). \end{array}$$

In words, the state $\sigma_{r+1}(p)$ is computed according to the next-state function T_p (at the current round r) from the state $\sigma_r(p)$, and the vector of messages that p receives at round r according to the HO collection. A step of a coarse-grained execution encapsulates a move by each process; because messages can be received only in the rounds for which they have been sent, there is no need to represent messages in transit.

Variations. We made some choices in the above definitions. For example, all message sending transitions for a process in a given round could be grouped into a single transition, and possibly even combined with the local transitions, yielding an intermediate granularity of executions. Another alternative would be to define executions without fixing the HO collection in advance. Instead, a local transition in the fine-grained model could occur at any point after the process has sent all messages for the current round. In the coarse-grained model, the HO sets $HO(p,r)$ would be chosen non-deterministically. Indeed, the representation of HO algorithms in TLA$^+$ presented in Section 4 is defined in such a way.

Charron-Bost and Schiper [1] define a variant of HO algorithms called *coordinated* HO algorithms, whose message-sending functions S_p and transition relations T_p depend on an additional parameter indicating the process that p believes to be the coordinator of the current round. Correspondingly, executions are defined in this variant with respect to a HO collection as well as an assignment of coordinators $Coord(p,r) \in \Pi$ per process and round.

The reduction theorem presented in the following section can be adapted to any of these alternative definitions. It also extends to non-deterministic settings where each process has a set of possible initial states and a next-state relation instead of a next-state function. The only essential requirement is that processes react only to messages intended for the round they are currently executing.

3 A Reduction Theorem for HO Algorithms

We now present our main theorem, which asserts, informally, that in the HO model, the fine-grained and coarse-grained execution semantics are indistinguishable from the point of view of any process.

3.1 Relating the Two Models of Execution

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 local states that q assumes in ρ. More precisely, for a fine-grained execution $\xi = c_0 c_1 \ldots$, we define

$$\xi^q = c_0.st(q)\ c_1.st(q)\ \ldots.$$

For a coarse-grained execution $\sigma = \sigma_0\sigma_1 \ldots$, the q-view is simply

$$\sigma^q = \sigma_0(q)\,\sigma_1(q)\ \ldots$$

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* [5], 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 executions do not generate any more local views of an algorithm than coarse-grained ones.

Theorem 1. *For any fine-grained execution $\xi = c_0 c_1 \ldots$ of an HO algorithm for some HO collection $\big(HO(p,r)\big)_{p\in\Pi, r\in\mathbb{N}}$, there exists a coarse-grained execution σ of the same algorithm for the same HO collection such that $\sigma \approx \xi$.*

Proof (sketch). Given execution $\xi = c_0 c_1 \ldots$ and some process $p \in \Pi$, let $\ell_0^p = 0$ and for $n > 0$, $\ell_n^p = k+1$ if (c_k, c_{k+1}) is the n-th local transition of p in ξ; remember that every process p performs infinitely many local transitions in a fine-grained execution. By the definition of fine-grained executions, round numbers and local states of p change only during local transitions. It follows that $c_i.rd(p) = c_{\ell_n^p}.rd(p) = n$ and $c_i.st(p) = c_{\ell_n^p}.st(p)$ for all $n \in \mathbb{N}$ and all $\ell_n^p \le i < \ell_{n+1}^p$.

We will now show that the sequence $\sigma = \sigma_0\sigma_1 \ldots$ defined by

$$\sigma_n = \big(p \in \Pi \mapsto c_{\ell_n^p}.st(p)\big)$$

is a coarse-grained execution of the same algorithm for the given HO collection HO. By the observations above, this definition of σ ensures that $\sigma^p \simeq \xi^p$ for all $p \in \Pi$, and therefore $\sigma \approx \xi$.

To show the initialization condition, it suffices to observe that

$$\sigma_0(p) = c_{\ell_0^p}.st(p) = c_0.st(p) = s_{0,p}$$

is the initial state for all $p \in \Pi$. It remains to show that for all $p \in \Pi$ and $n \in \mathbb{N}$, we have $\sigma_{n+1}(p) = T_p\big(n, \sigma_n(p), rcvd(p,n)\big)$.

By induction on the definition of fine-grained executions, it is easy to verify the following invariants, for all $n, r \in \mathbb{N}$, $m \in M$, and $p, q \in \Pi$:

- If $(p, r, q, m) \in c_n.msgs$ then $m = S_p(r, c_{\ell_r^p}.st(p), q)$: any message for round r in transit from p to q was computed according to p's send function for round r, based on p's local state at (the beginning of) round r.

– If $r = c_n.rd(q)$ and $\bot \neq m = c_n.rcv(q,p)$ then $m = S_p(r, c_{\ell_r^p}.st(p), q)$: any message received by q from p for round r was computed according to p's send function for round r, based on p's local state at (the beginning of) round r.

For $n \in \mathbb{N}$, consider now the transition of process p from round n to round $n+1$ in execution ξ, i.e. the transition $(c_{\ell_{n+1}^p - 1}, c_{\ell_{n+1}^p})$. For simplicity of notation, we write $c = c_{\ell_{n+1}^p - 1}$ and $c' = c_{\ell_{n+1}^p}$. From the observations about round numbers and local states we know that $c.rd(p) = n$, $c.st(p) = c_{\ell_n^p}.st(p)$, and $c'.rd(p) = n+1$. Since (c, c') is a local transition of p, we have $\operatorname{dom} c.rcv(p) = HO(p,n)$, and in particular $c.rcv(p,q) = \bot$ iff $q \in \Pi \setminus HO(p,n)$. Moreover, the second invariant above implies that

$$c.rcv(p,q) = S_q(n, c_{\ell_n^q}.st(q), p)$$

for all $q \in HO(p,n)$. Altogether this means $c.rcv(p) = rcvd(p,n)$.

Using the fact that $c'.st(p) = T_p\big(c.rd(p), c.st(p), c.rcv(p)\big)$ and rewriting with the above equalities, we obtain that

$$\sigma_{n+1}(p) = c'.st(p) = T_p\big(n, \sigma_n(p), rcvd(p,n)\big),$$

which completes the proof. □

We note in passing that the converse of Theorem 1 is true almost trivially.

Theorem 2. *For any coarse-grained execution σ of an HO algorithm for some HO collection $\big(HO(p,r)\big)_{p\in\Pi, r\in\mathbb{N}}$, there exists a fine-grained execution ξ of the same algorithm for the same HO collection such that $\xi \approx \sigma$.*

Proof (sketch). Given a coarse-grained execution σ, it is easy to construct a corresponding fine-grained execution where processes execute rounds in lock-step, first sending all messages, then receiving the messages according to the HO sets $HO(p,r)$ and finally performing their respective local transitions. □

3.2 Application: Verification of Local Properties

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. More 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 $\rho_2 \models P$.[7]

Corollary 3. *If P is a local property and $\sigma \models P$ holds for all coarse-grained executions σ of an algorithm, then $\xi \models P$ also holds for all fine-grained executions ξ of the same algorithm.*

[7] As usual, $\rho \models P$ means that P is satisfied by execution ρ.

Proof. Let ξ be some fine-grained execution (over some HO collection), then Theorem 1 yields a coarse-grained execution σ (over the same HO collection) such that $\sigma \approx \xi$. By assumption, we must have $\sigma \models P$, and since P is local, this implies $\xi \models P$. □

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, and because fewer (types of) transitions must be considered.

Corollary 3 is useful in practice if typical correctness properties are indeed local. Observe that local properties must be stuttering invariant [8], by the definition of local equivalence $\approx$ of executions. Moreover, their satisfaction should not depend on the specific interleaving of process transitions. As a trivial example for a non-local property, suppose that each process $p \in \Pi$ maintains a counter of its current round in the variable rnd_p. Then any coarse-grained execution by definition satisfies the LTL formula

$$\bigwedge_{p,q \in \Pi} \Box(rnd_p = rnd_q) \tag{1}$$

asserting that all processes execute the same round at any moment; this formula obviously does not hold for fine-grained executions.

In the following we 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.[8] We assume that the set of (flexible) 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 $\mathcal{V}_p$.

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 simulaneity of local states of different processes, or assert temporal relations between states of processes, and the formula (1) is a typical example since variables of different processes appear in the scope of a temporal operator.

3.3 Consensus as a Local Property

We argue that local properties express many interesting correctness properties of distributed algorithms. As a concrete and important example, consider the specification of the Consensus problem [7]. 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

[8] LTL-X formulas are stuttering invariant [8]; our criterion carries over to the logic TLA considered in Section 4 because LTL-X is a sublogic of TLA.

$decide_p$, 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.

The Consensus problem is specified by the conjunction of the following formulas of LTL-X, which are all local according to the criterion introduced in Section 3.2.

Integrity. The integrity property asserts that decision values must be among the initial values (possibly of some other process). This property is expressed by the following first-order combination of p-formulas:

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

Irrevocability. A process that has decided must never change its decision value. This property is expressed by the following p-formula, for all $p \in \Pi$:

$$\forall v : v \neq null \Rightarrow \Box\big(decide_p = v \Rightarrow \Box(decide_p = v)\big)$$

Agreement. The core correctness property of Consensus algorithms requires that if any two processes decide, they decide on the same value. Again, this can be expressed as a first-order combination of p-formulas:

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

Termination. The preceding properties are all safety properties. The final property required by Consensus is that all (non-faulty) processes eventually decide. Because the HO model does not flag processes as being faulty [1], this property is simply expressed by the following p-formula, for all $p \in \Pi$:

$$\Diamond(decide_p \neq null).$$

4 Model Checking HO Algorithms

We validate the effectiveness of our reduction-based approach to verification by verifying finite instances of Consensus algorithms in the HO model.

Exploiting Corollary 3, we model coarse-grained executions of HO algorithms in TLA$^+$ [6]. We instantiate this generic model for some of the Consensus algorithms that are discussed in [1], and use the TLA$^+$ model checker TLC [12] for verification. In this work, we do not aim at utmost efficiency, but prefer the high level of abstraction offered by TLA$^+$ that lets us obtain readable models, close to the mathematical description of HO algorithms in Section 2.

However, model checking even finite instances of these algorithms would be impossible in the fine-grained execution model: the model would have to include round numbers and therefore be infinite-state. Even if we artificially imposed bounds on round numbers (abandoning the verification of liveness properties), state explosion would make verification impractical.

MODULE *HeardOf*

EXTENDS *Naturals*
CONSTANTS $Proc, State, Msg, roundsPerPhase, Start(\_), Send(\_,\_,\_,\_), Trans(\_,\_,\_,\_)$
VARIABLES *round, state, heardof*

$$
\begin{array}{lcl}
Init & \triangleq & \land\ round = 0 \\
 & & \land\ state = [p \in Proc \mapsto Start(p)] \\
 & & \land\ heardof = [p \in Proc \mapsto \{\}] \\
Step(HO) & \triangleq & \text{LET}\ rcvd(p) \triangleq \{\langle q, Send(q, round, state[q], p)\rangle : q \in HO[p]\} \\
 & & \text{IN}\ \ \land\ round' = (round + 1)\%roundsPerPhase \\
 & & \quad\ \land\ state' = [p \in Proc \mapsto Trans(p, round, state[p], rcvd(p))] \\
 & & \quad\ \land\ heardof' = HO \\
Next & \triangleq & \exists HO \in [Proc \to \text{SUBSET}\ Proc] : Step(HO) \\
vars & \triangleq & \langle round, state, heardof\rangle \\
NoSplit(HO) & \triangleq & \forall p, q \in Proc : HO[p] \cap HO[q] \neq \{\} \\
NextNoSplit & \triangleq & \exists HO \in [Proc \to \text{SUBSET}\ Proc] : NoSplit(HO) \land Step(HO) \\
Uniform(HO) & \triangleq & \exists S \in \text{SUBSET}\ Proc : S \neq \{\} \land HO = [q \in Proc \mapsto S]
\end{array}
$$

Fig. 1. Generic TLA$^+$ module for HO algorithms.

4.1 A Generic TLA$^+$ Model for HO Algorithms

We begin by introducing a generic representation of coarse-grained executions of HO algorithms in TLA$^+$. It is similar to the semantic presentation in Section 2.2, except for two differences that help us obtain finite-state models. The first difference concerns round numbers, which are formally a parameter of the functions S_p and T_p. Many actual algorithms do not refer to the absolute round number, but are organized in *phases*, where a phase consists of a fixed finite number of rounds. Therefore, the functions S_p and T_p only depend on the current round number relative to the phase number, and it suffices to count rounds modulo the number of rounds per phase. The second difference, already indicated at the end of Section 2.2, is to choose assignments of HO sets to processes non-deterministically for each step of the algorithm instead of fixing them in advance.

A generic TLA$^+$ module that represents HO algorithms appears in Fig. 1. It begins by importing the standard TLA$^+$ module for arithmetic over natural numbers and then declares the constant and variable parameters of the module: the sets *Proc*, *State* and *Msg* represent processes, process states, and messages. Parameter *roundsPerPhase* indicates the number of rounds per phase. The parameters *Start*, *Send*, and *Trans* will be instantianted to specify the behavior of concrete algorithms: for each $p \in Proc$, the predicate $Start(p)$ characterizes the initial state of p, $Send(p, r, s, q)$ yields the message that process p sends to process q at round r of a phase, given p's current local state s, and $Trans(p, r, s, rcvd)$ computes the next state of p at round r, given p's local state

s and the partial vector *rcvd* of messages received, which we represent as a set of pairs $\langle q, m \rangle$ indicating that m was received from q.

A round of the system is represented by three variables: *round* indicates the number of the current step modulo *roundsPerPhase*, *state* is an array[9] of local states per process, and *heardof* records the HO assignment of the preceding transition (its initial value is chosen arbitrarily). This auxiliary variable serves to express communication predicates.

With this understanding, the initialization and next-state predicates closely follow the definition of coarse-grained executions in Section 2.2. The predicates *NoSplit* and *Uniform* are two examples of formulas serving to express communication predicates. Action *NextNoSplit* defines a variant of the next-state relation that enforces non-split rounds. The remaining communication predicates appearing in [1] can be defined in a similar way.

4.2 Modeling and Verifying Concrete HO Algorithms in TLA$^+$

Charron-Bost and Schiper [1] propose several Consensus algorithms in the HO model. As an example of how these can be encoded in our TLA$^+$ framework, a specification of their *OneThirdRule* algorithm appears in Fig. 2. The module declares the constant parameter N (the number of processes) and *defines* the sets *Proc* and *Msg*: we arbitrarily specify that each process $p \in 1..N$ proposes $10 * p$ as its initial value. Next, the module defines the remaining constant parameters of module *HeardOf*. Phases of *OneThirdRule* consist of only one round. Process states are represented as records with two fields x and *decide*, whose initialization is obvious. At each round, each process sends its current x field to all processes. The next-state function is defined as follows: if a process has received messages from more than 2/3 of all processes, it updates its x field to the smallest most frequently received value (cf. definition of *min*). If it has received some value v from more than 2/3 of all processes, then it also updates its *decide* field to v.

Charron-Bost and Schiper show that the algorithm *OneThirdRule* always achieves the integrity, irrevocability, and agreement properties, and that it guarantees termination for runs that eventually execute some uniform round for "sufficiently large" heard-of sets. We express these properties in TLA and use TLC to verify these theorems. Observe that we have expressed the correctness properties in module *OneThirdRule* using different, but equivalent formulas than those given in Section 3.3. In particular, TLC checks *Validity*, *Agreement*, and *Irrevocability* as state and transition invariants while computing the state space, which is more efficient than verifying arbitrary temporal formulas. Because the concept of local properties is a semantic one, it is independent of the particular syntactic formulation of the property, and we can apply Corollary 3 to deduce that the formulas are also satisfied by fine-grained executions.

Figure 3 gives the number of generated and distinct states and the running time of TLC for verifying these properties, as well as for the somewhat more complicated *UniformVoting* algorithm that is encoded in TLA$^+$ in a similar

[9] TLA$^+$ uses square brackets to denote functions.

MODULE *OneThirdRule*

EXTENDS *Naturals, FiniteSet*
CONSTANTS N
VARIABLES *round, state, heardof*

$$
\begin{array}{ll}
roundsPerPhase & \triangleq\ 1 \\
Proc & \triangleq\ 1\,..\,N \\
InitValue(p) & \triangleq\ 10 * p \\
Value & \triangleq\ \{InitValue(p) : p \in Proc\} \\
Msg & \triangleq\ Value \\
null & \triangleq\ 0 \\
ValueOrNull & \triangleq\ Value \cup \{null\}
\end{array}
$$

$$
\begin{array}{ll}
State & \triangleq\ [x : Value, decide : ValueOrNull] \\
Init(p) & \triangleq\ [x \mapsto InitValue(p), decide \mapsto null] \\
Send(p, r, s, q) & \triangleq\ s.x \\
Trans(p, r, s, rcvd) & \triangleq \\
\quad \text{IF } Cardinality(rcvd) > (2 * N) \div 3 & \\
\quad \text{THEN LET } Freq(v) \triangleq Cardinality(\{q \in Proc : \langle q, v\rangle \in rcvd\}) & \\
\qquad MFR(v) \triangleq \forall w \in Value : Freq(w) \leq Freq(v) & \\
\qquad min \triangleq \text{CHOOSE } v \in Value : MFR(v) \wedge (\forall w \in Value : MFR(w) \Rightarrow v \leq w) & \\
\qquad willDecide \triangleq \exists v \in Value : Freq(v) > (2 * N) \div 3 & \\
\qquad \text{IN } [x \mapsto min, & \\
\qquad\quad decide \mapsto \text{IF } willDecide \text{ THEN CHOOSE } v \in Value : Freq(v) > (2 * N) \div 3 & \\
\qquad\qquad\qquad \text{ELSE } s.decide] & \\
\quad \text{ELSE } s &
\end{array}
$$

INSTANCE *HeardOf*

$$
\begin{array}{ll}
Safety & \triangleq\ Init \wedge \Box[Next]_{vars} \\
Liveness & \triangleq\ \Diamond(Uniform(heardof) \wedge Cardinality(heardof) > (2 * N) \div 3) \\
Integrity & \triangleq\ \forall p \in Proc : \Box\big(state[p].decide \in \{null\} \cup \{InitValue(p) : p \in Proc\}\big) \\
Irrevocability & \triangleq\ \forall p \in Proc : \Box[state[p].decide = null]_{state[p].decide} \\
Agreement & \triangleq\ \forall p, q \in Proc : \Box(state[p].decide \neq null \wedge state[q].decide \neq null \\
 & \qquad\qquad \Rightarrow\ state[p].decide = state[q].decide) \\
Termination & \triangleq\ \forall p \in Proc : \Diamond(state[p].decide \neq null)
\end{array}
$$

THEOREM $Safety \Rightarrow Integrity \wedge Irrevocability \wedge Agreement$
THEOREM $Safety \wedge Liveness \Rightarrow Termination$

Fig. 2. TLA$^+$ specification of the algorithm *OneThirdRule*.

	OneThirdRule		UniformVoting	
	$N = 3$	$N = 4$	$N = 3$	$N = 4$
states	5633	9,830,401	21351	15,865,770
distinct	11	150	122	887
time (s)	1.87	939	13.8	1330

Fig. 3. Results of verification with TLC.

way. Measurements were taken on an Intel® 2.16GHz Core Duo® laptop with 2GB RAM running Mac OSX 10.5. It is apparent that the non-deterministic choice of a collection of heard-of sets at every transition induces a combinatorial explosion in the number of successor states that are generated, although many of them are identical (cf. the low number of distinct states generated by the model checker). As mentioned above, we regard these results just as an indication of the feasibility of the approach; there is ample room for improvement using standard optimization techniques or symbolic model checking. In particular, we did not apply symmetry reduction, except for identifying states that differ only in the value of the auxiliary variable *heardof*.

5 Conclusion

The main contribution of this paper is the precise statement and the proof of a reduction theorem for algorithms expressed in the HO model. The key ingredient for obtaining the reduction theorem is the fact that the HO model relies on communication-closed rounds. For this reason, the local transition (of a fine-grained execution) in which process p passes from round r to round $r + 1$ is causally independent of all transitions of processes at rounds $r' > r$. Hence, executions can be rearranged into coarse-grained executions whose unit of atomicity is that of global rounds of all processes, without changing the local observations of any process.

As a corollary to the reduction theorem, we obtain a method for applying standard model checking algorithms for the verification of local properties of distributed algorithms, that is, properties whose satisfaction only depends on local views of processes. We have shown that this class of properties contains the correctness properties of Consensus algorithms. Specifically, we have been able to verify (finite instance of) some Consensus algorithms proposed in [1], which would be impossible in a standard, fine-grained representation of executions.

Tsuchiya and Schiper [9, 10] applied symbolic and bounded model checkers to verify Consensus algorithms in the HO model over coarse-grained runs. However, they do not explain why the verification of coarse-grained models is sufficient. Our contribution can be understood as a formal justification of their models; we also delimit the applicability of the approach by introducing the notion of local properties.

In future work, we intend to further validate this approach by verifying more distributed algorithms. We are also interested in syntactic criteria (beyond the

basic one presented in Section 3.3) for determining if a temporal formula expresses a local property. A longer-term goal would be to have model checkers apply this kind of reduction automatically whenever the user attempts to verify a local property of a distributed algorithm.

References

1. B. Charron-Bost and A. Schiper. The Heard-Of model: Computing in distributed systems with benign failures. *Distributed Computing*, 2009. To appear.
2. T. Elrad and N. Francez. Decomposition of distributed programs into communication-closed layers. *Science of Computer Programming*, 2(3), Apr. 1982.
3. P. Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems. An Approach to the State-Explosion Problem*, volume 1032 of *Lecture Notes in Computer Science*. Springer, 1996.
4. L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.
5. 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.
6. L. Lamport. *Specifying Systems*. Addison-Wesley, Boston, Mass., 2002.
7. N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.
8. D. Peled and T. Wilke. Stutter-invariant temporal properties are expressible without the next-time operator. *Inf. Proc. Letters*, 63(5):243–246, 1997.
9. 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 Computer Society.
10. T. Tsuchiya and A. Schiper. Using bounded model checking to verify consensus algorithms. In G. Taubenfeld, editor, *22nd Intl. Symp. Distributed Computing (DISC 2008)*, volume 5218 of *Lecture Notes in Computer Science*, pages 466–480, Arcachon, France, 2008. Springer.
11. A. Valmari. The state explosion problem. In *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*, pages 429–528. Springer, 1998.
12. Y. Yu, P. Manolios, and L. Lamport. Model checking TLA+ specifications. In L. Pierre and T. Kropf, editors, *Correct Hardware Design and Verification Methods (CHARME'99)*, volume 1703 of *Lecture Notes in Computer Science*, pages 54–66, Bad Herrenalb, Germany, 1999. Springer.