Distrib. Comput. (2011) 23:341–358
DOI 10.1007/s00446-010-0123-3

# Verification of consensus algorithms using satisfiability solving

**Tatsuhiro Tsuchiya · André Schiper**

Received: 13 July 2009 / Accepted: 11 November 2010 / Published online: 27 November 2010


**Abstract** *Consensus* is at the heart of fault-tolerant distributed computing systems. Much research has been devoted to developing algorithms for this particular problem. This paper presents a semi-automatic verification approach for asynchronous consensus algorithms, aiming at facilitating their development. Our approach uses model checking, a widely practiced verification method based on state traversal. The challenge here is that the state space of these algorithms is huge, often infinite, thus making model checking infeasible. The proposed approach addresses this difficulty by reducing the verification problem to small model checking problems that involve only single phases of algorithm execution. Because a phase consists of a small, finite number of rounds, bounded model checking, a technique using satisfiability solving, can be effectively used to solve these problems. The proposed approach allows us to model check several consensus algorithms up to around 10 processes.

**Keywords** Consensus · Model checking · Fault-tolerant distributed algorithms · Formal verification

## 1 Introduction

*Consensus* is the problem of getting processes to agree on the same decision in spite of faults. Consensus is central to the construction of fault-tolerant distributed systems. For example, atomic broadcast, which is at the core of state machine replication, can be implemented as a sequence of consensus instances [5]. Other services, such as view synchrony and membership, can also be constructed using consensus [19,33]. Because of its importance, much research has been being devoted to developing new algorithms for this problem.

In this paper we study the verification of consensus algorithms for non-synchronous distributed systems, aiming at facilitating the development of such algorithms. Specifically, we propose a semi-automatic verification approach based on model checking techniques. *Model checking* is a popular method for formally verifying state transition systems with state traversal. The main advantage of using this method is that it allows the fully automatic analysis of designs. It is now possible to model check systems with 100 million states or more, thanks to the rapid progress in related algorithms.

However, model checking of asynchronous consensus algorithms is still far from straightforward. As these algorithms have runs of unbounded length, have unbounded queues or unbounded sets of messages in transit, they induce an infinite state space, making even latest model checkers powerless. This problem can be partially alleviated by adopting a round-based model, which is an abstract model built on top of the message-passing model. In particular, we adopt a round-based model called the *Heard-Of (HO) model* [9,10]. In the conventional model, events, such as message send or receive, are arbitrarily interleaved. That is, any event causes a different system state, thus easily causing a state space explosion. In the HO model (as in other round-based models), on the other hand, an algorithm is modeled as a sequence of rounds and therefore has a more regular structure. In each round, every process sends messages, then receives messages from other processes, and finally makes a local state transition. As a global state transition occurs only once per each round, this model allows us to abstract away a large number

T. Tsuchiya (✉)
Osaka University, 1–5 Yamadaoka, Suita, Osaka 565-0871, Japan
e-mail: t-tutiya@ist.osaka-u.ac.jp

A. Schiper
École Polytechnique Fédérale de Lausanne (EPFL),
1015 Lausanne, Switzerland
e-mail: andre.schiper@epfl.ch

of intermediate states, thus making the process of design and analysis of algorithms much easier.

This coarse abstraction rests on the ground that most of the existing consensus algorithms are structured in *communication-closed* rounds: that is, processes react solely to messages sent for the round that they are currently executing. In non-synchronous settings, communication-closedness is ensured by buffering early messages and discarding late messages. In [6] it is proven that under the assumption of communication-closedness, the lockstep semantics of the HO model and the interleaving semantics are equivalent with respect to the correctness of consensus algorithms.

However, as will be shown later, adopting a round model is not sufficient for model checking purposes: even with state-of-the art model checkers, the system size that can be directly model checked is rather small, typically three or four processes.

We address this scalability problem as follows: In our approach the verification problem is divided into several problems that can be solved by model checking. Importantly, these model checking problems can be solved by only analyzing single phases—that is, a small number of consecutive rounds—of a consensus algorithm. Because of this property, *bonded model checking* [13] can be very effectively used. As the name suggests, bounded model checking searches state transitions of bounded length. This restriction allows the model checking problem to be reduced to the *satisfiability problem* for a formula, which is the problem of determining whether or not values can be assigned to variables such that the formula evaluates to true. Bounded model checking can often be used only for finding defects near the initial states; but in our case this problem is avoided because it is already known that the depth of the search space exactly matches the number of rounds in a phase.

In reducing the verification problem to the set of bounded model checking problems, it is necessary to use approximations of some state sets, because otherwise it would not be possible to examine all behaviors only by checking single phases. Specifically we need to approximate the set of reachable states, as well as the set of *univalent* states—the states from which only one value may be decided. Practical success of our approach depends on how accurate these approximations are. In case studies we show that it is not difficult to derive, from a given algorithm, the approximations accurate enough to make the verification process successful.

Through the case studies, we also show that the proposed approach doubles the size of systems that can be verified, compared to the approach in which the whole state space is model checked. This improvement cannot be achieved without a significant reduction in the time and memory space used for verification, because the state space increases exponentially in the number of processes.

### 1.1 Related work

Some attempts have been reported to model check asynchronous consensus algorithms. As stated above, if a conventional message-passing model is assumed, then one has to deal with an infinitely huge state space with fine details and thus model checking can only be used for limited purposes. For example, TLC, the TLA+ model checker, was used to find errors in the TLA+ specifications of some *Paxos* algorithms [18,27]. The models that were model checked consisted of two or three processes and a small number of rounds [26]. In [38], an approach to automatic discovery of consensus algorithms is proposed. This approach depends on a procedure that determines if a given decision rule satisfies the required safety properties of a single phase of a "full-information exchange" consensus algorithm. This procedure cannot be used for liveness verification or to verify an entire consensus algorithm.

In [20], a synchronous consensus algorithm is model checked for three processes. Studies on model checking of shared memory-based randomized consensus algorithms can be found in [11,24].

Model checking is also applied to the algorithms for Byzantine agreement, which is a similar problem to consensus [2,23]. In [2] a synchronous system model is assumed. In [23] an asynchronous algorithm is verified but manual proof is used in combination with model checking.

In our previous work [36], we adopt the HO model and model check several asynchronous consensus algorithms with the NuSMV model checker [12]. The approach there is to build a finite state model that represents the whole state space of a given algorithm using an abstraction technique. The model generated is in turn verified by traditional model checking. As we show, the performance improvement of the technique proposed in this paper with respect to [36] is significant.

Adopting the HO model can facilitate not only model checking but also *theorem proving*. In [7] a formal proof is presented for the *LastVoting* algorithm, a variant of *Paxos* in the HO model. This proof is "at least five times shorter" than the one for a similar consensus algorithm based on the conventional message-passing model.

None of these previous studies, including ours, uses bounded model checking. It is partly because of its inherent limitation, namely, only depth-bounded state search is possible. In the model checking community, the extension of bounded model checking to "unbounded" model checking has been intensively researched recently. Combining induction and bounded model checking is one such approach [34]. Our verification approach has some similarity with it. Especially the procedure for proving an invariant, which we will explain in Sect. 5.1, can be viewed as a variation of $k$-induction [32], a generalization of induction.

### 1.2 Roadmap

This paper is structured as follows: Section 2 describes the HO model and the consensus problem. Sections 3 and 4 describe our proposed model checking techniques for verification of safety and liveness, respectively. Section 5 proposes automatic procedures for validating two important assumptions used in the safety and liveness verification. Section 6 shows the results of case studies. Section 7 concludes the paper.

## 2 The consensus problem

We start with an overview of the models that have been considered for solving consensus. Then we focus on the round model that we use in the paper.

### 2.1 Models for solving consensus

Consensus is solvable in a synchronous system, but this leads to inefficient solutions. This fact, and the FLP impossibility result [16], have led to focus on models stronger than the asynchronous system model, and weaker than the synchronous system model. One such model is the partially synchronous system model [15]; another model is the asynchronous system model augmented with failure detectors [5].

We consider here the partially synchronous system model, and, as in [15], we consider an abstraction on top of the system model for expressing consensus algorithms: a round-based model. The reason for this choice has been explained in Sect. 1, where we point out the problems of model checking of asynchronous algorithms. It should be noted that a consensus algorithm expressed in the asynchronous system augmented with failure detectors can, in general, always be translated into a round-based consensus algorithm. Therefore, focusing on a round-based model does not restrict the scope of the paper.

### 2.2 Round-based model

In a round-based model, the computation consists of rounds of message exchange. In each round $r$, each process $p$ sends a (possibly empty) message according to a sending function $S_p^r$ to every process, and, at the end of round $r$, computes a new state according to a state transition function $T_p^r$: message reception is thus implicit. The state transition function takes as input the set of messages received in round $r$ (a message sent in round $r$ can only be received in round $r$) and the current process state. Since the message reception is implicit, waiting for messages is not handled at the application level. The issue is handled by the layer that implements the round model. In [15] the round layer considered provides the following guarantee: there exists a round $GSR > 0$ such that for all rounds $r \geq GSR$, all messages sent in round $r$ by correct processes to correct processes are received in round $r$. An algorithm that ensures this property in a partially synchronous system is given in [15]. Another algorithm is proposed in [22].

The benefit of adding predicates to a round-based model has been shown by Gafni: predicates can be used to systematically characterize the synchrony properties extracted from the underlying system model and thus allow unification of synchrony and asynchrony [17]. We use here the notation introduced by the Heard-Of (HO) model [10]. If $\Pi$ is the set of processes, $HO(p, r) \subseteq \Pi$ denotes the set of processes from which $p$ receives a message in round $r$: $HO(p, r)$ is the "heard of" set of $p$ in round $r$. If $q \notin HO(p, r)$ while $q$ sent (or was supposed to send) a message to $p$ in round $r$, then a *transmission fault* occurred. This can be due to the asynchrony of communication or process, or to a process or link failure. The exact reason is not relevant. Predicates over the HO sets are called *communication predicates*. For example, an asynchronous system with reliable links and at most $f$ crash failures can be represented by the following communication predicate:

$$\forall p \in \Pi, \quad \forall r > 0 : |HO(p, r)| \geq n - f$$

For any round $r$, its kernel is defined as the set of processes $K(r) = \bigcap_{p \in \Pi} HO(p, r)$. With this notation, a synchronous system with reliable links and at most $f$ crash failures can be represented by the following communication predicate:

$$\left| \bigcap_{r>0} K(r) \right| \geq n - f \wedge \forall p \in \Pi, \quad \forall r > 0 : HO(p, r+1) \subseteq K(r)$$

The consensus problem is therefore solved by a pair "round-based algorithm + communication predicate". The solution applies to a partially synchronous system whenever the communication predicate is implementable in such a system. We come back to this issue later in Sect. 2.4.

The round-based model can naturally be extended to accommodate coordinator-based algorithms, by letting a communication predicate deal with not only HO sets but also with coordinators. This extended notion of a communication predicate is called a *communication-coordinator predicate*.

A process is usually coordinator for a sequence of rounds, and this sequence of rounds is called a *phase*. We denote by $k$ the number of rounds that compose a single phase. Let $Coord(p, \phi) \in \Pi$ denote the coordinator of process $p$ in phase $\phi$. We assume that $p$ knows its coordinator $Coord(p, \phi)$ in phase $\phi$ and that the coordinator does not change during that phase. The domain of a communication-coordinator predicate is the collection of $HO(p, r)$ and $Coord(p, \phi)$, for all $p \in \Pi, r > 0, \phi > 0$. The sending function and the state transition function are now represented

as $S_p^r(s_p, Coord(p,\phi))$ and $T_p^r(Msg, s_p, Coord(p,\phi))$, where $\phi$ is the phase that round $r$ belongs to.

As in the case for HO sets, the property of coordinators is represented by a communication-coordinator predicate. For example, the property that all processes will eventually have an identical coordinator is represented by the predicate: $\exists\phi > 0, \exists co \in \Pi, \forall p \in \Pi : co = Coord(p,\phi)$. Note that the procedure for selecting coordinators is outside of the consensus algorithm and is abstracted away in the HO model: processes may use an external device/oracle or may adopt the rotating coordinator strategy.

Since the correctness of an algorithm falls into safety or liveness, we consider a communication-coordinator predicate $\mathcal{P}$ of the form ($\triangleq$ means *is defined to equal*)

$$\mathcal{P} \triangleq \mathcal{P}^{safe} \wedge \mathcal{P}^{live},$$

where $\mathcal{P}^{safe}$ is used for satisfying safety, whereas $\mathcal{P}^{live}$ is used for satisfying liveness in conjunction with $\mathcal{P}^{safe}$.

We assume that $\mathcal{P}^{safe}$ and $\mathcal{P}^{live}$ are of the following form:

$$\begin{aligned} \mathcal{P}^{safe} &\triangleq \forall\phi > 0 : P^{safe}(\phi) \\ \mathcal{P}^{live} &\triangleq \exists\phi > 0 : P^{sync}(\phi) \end{aligned}$$

where $P^{safe}(\phi)$ and $P^{sync}(\phi)$ are predicates over HO sets and coordinators in phase $\phi$. The latter form means that only a single "good" phase is required to satisfy termination. We will explain the reason for this assumption in Sect. 2.4. We also assume that $P^{safe}(\phi) \wedge P^{sync}(\phi)$ is not the constant predicate FALSE.[1] This excludes the uninteresting case where the system has no possible behavior.

### 2.3 Consensus

*Consensus* is the problem of getting all processes to agree on the same decision. Each process is assumed to have a proposed value at the beginning and is required to eventually decide on a value proposed by some process. We consider the following specification:

- *Integrity*: Any decision value is the proposed value of some process.
- *Agreement*: No two different values are decided.
- *Termination*: All processes eventually decide.[2]

It is easy to see that integrity and agreement are safety properties, whereas termination is a liveness property. For most consensus algorithms, integrity is trivially satisfied; thus we limit our discussion to the verification of agreement and termination. Note that the termination property requires all processes to decide. This is because there is no notion of faulty processes in the HO model,[3] and therefore a process is never exempted from deciding. There is no harm in doing so, since we can always represent a process $p$ that crashes as a process that is no more heard of by any other process (but still hears the other non-crashed processes). Requiring such a process $p$ to decide has no impact on the rest of the computation.

### 2.4 The *LastVoting* (paxos) algorithm

We present now the *LastVoting* algorithm (Fig. 1) that is used as a running example throughout the paper [10]. *LastVoting* can be viewed as an HO model-version of *Paxos* [25]. It is also close to the $\Diamond\mathcal{S}$ consensus algorithm by Chandra and Toueg [5].

In *LastVoting* a phase consists of four rounds. In the first round (round $4\phi - 3$), coordinators collect the current estimate $x_p$ and the timestamp $ts_p$ from processes. If a coordinator obtains these values from a majority of processes, then it picks up the estimate that is associated with the greatest timestamp and sets $vote_p$ equal to that estimate. In the second round (round $4\phi - 2$) the coordinator broadcasts $vote_p$ to all processes. If a process $p$ receives this value, then it updates timestamp $t_p$ to the current phase number $\phi$ and then votes for that value by replying ack to the coordinator in the third round (round $4\phi - 1$). If the coordinator obtains a majority of votes, then it again broadcasts the value of $vote_p$ in the fourth round (round $4\phi$). If a process receives this value, then it decides on the value.

A sufficient condition for the *LastVoting* algorithm to solve consensus is specified by the communication-coordinator predicate $\mathcal{P} \triangleq \mathcal{P}^{safe} \wedge \mathcal{P}^{live}$ such that:[4]

$$\begin{aligned} P^{safe}(\phi) &\triangleq \texttt{TRUE} \\ \mathcal{P}^{safe} &\triangleq \forall\phi > 0 : P^{safe}(\phi) \\ &= \texttt{TRUE} \\ P^{sync}(\phi) &\triangleq \exists co \in \Pi, \forall p \in \Pi : \\ &\quad \wedge\ co = Coord(p,\phi) \\ &\quad \wedge\ |HO(co, 4\phi - 3)| > n/2 \\ &\quad \wedge\ |HO(co, 4\phi - 1)| > n/2 \\ &\quad \wedge\ co \in HO(p, 4\phi - 2) \\ &\quad \wedge\ co \in HO(p, 4\phi) \\ \mathcal{P}^{live} &\triangleq \exists\phi > 0 : P^{sync}(\phi) \end{aligned} \tag{1}$$

[1] As stated in Sect. 4, this assumption is required by termination verification.

[2] As in much of the literature on consensus, we do not require processes to halt. The problem of ensuring halting can be treated separately as the problem of reliably broadcasting a decision value, which in practice can be done by repeated sending.

[3] However, all messages sent by a process may be lost, which is indistinguishable for other processes.

[4] Adopting the notation of TLA$^+$, we write a conjunction or disjunction as a list of formulas bulleted by $\wedge$ or $\vee$.

**Fig. 1** The *LastVoting ( Paxos)* algorithm

1: **Initialization:**
2: $x_p \in Val$, initially $v_p$
{*Estimate for the decision.* $v_p$ *is the proposed value of* $p$.}
3: $vote_p \in Val \cup \{?\}$, initially ?
{$Val$ *is the set of values that may be proposed.*}
4: $commit_p$ a Boolean, initially `false`
5: $ready_p$ a Boolean, initially `false`
6: $ts_p \in \mathbb{N}$, initially 0
{*Timestamp.* $\mathbb{N}$ *is the set of non-negative integers.*}

7: **Round** $r = 4\phi - 3$**:**
8: $S_p^r$ :
9: send $\langle x_p , ts_p \rangle$ to $Coord(p, \phi)$

10: $T_p^r$ :
11: **if** $p = Coord(p, \phi)$ **and** number of $\langle \nu , \theta \rangle$ received $> n/2$ **then**
12: let $\overline{\theta}$ be the largest $\theta$ from $\langle \nu , \theta \rangle$ received
13: $vote_p :=$ one $\nu$ such that $\langle \nu , \overline{\theta} \rangle$ is received
14: $commit_p :=$ `true`

15: **Round** $r = 4\phi - 2$**:**
16: $S_p^r$ :
17: **if** $p = Coord(p, \phi)$ **and** $commit_p$ **then**
18: send $\langle vote_p \rangle$ to all processes

19: $T_p^r$ :
20: **if** received $\langle v \rangle$ from $Coord(p, \phi)$ **then**
21: $x_p := v$ ; $ts_p := \phi$

22: **Round** $r = 4\phi - 1$**:**
23: $S_p^r$ :
24: **if** $ts_p = \phi$ **then**
25: send $\langle ack \rangle$ to $Coord(p, \phi)$

26: $T_p^r$ :
27: **if** $p = Coord(p, \phi)$ **and** number of $\langle ack \rangle$ received $> n/2$ **then**
28: $ready_p :=$ `true`

29: **Round** $r = 4\phi$**:**
30: $S_p^r$ :
31: **if** $p = Coord(p, \phi)$ **and** $ready_p$ **then**
32: send $\langle vote_p \rangle$ to all processes

33: $T_p^r$ :
34: **if** received $\langle v \rangle$ from $Coord(p, \phi)$ **then**
35: DECIDE$(v)$
36: **if** $p = Coord(p, \phi)$ **then**
37: $ready_p :=$ `false`
38: $commit_p :=$ `false`

Recall that we require all processes to decide, as stated in Sect. 2.3. Hence Formula (1) requires every process to agree on the coordinator $co$.

For the *LastVoting* algorithm, agreement can never be violated no matter how bad the HO set is; that is the algorithm is always safe, even in completely asynchronous runs. Hence $\mathcal{P}^{safe}$ is the constant predicate TRUE.

To meet liveness, on the other hand, some synchrony condition must be assumed, since there is no deterministic consensus algorithm in a pure asynchronous system [16]. The above $P^{sync}(\phi)$ specifies a synchronous phase $\phi$ where: all processes agree on the same coordinator $co$; $co$ can hear from a majority of processes in the first and third rounds of that phase; and every process can hear from $co$ in the second and fourth rounds. If such a phase $\phi$ occurs, then all processes will make a decision in that phase. The predicate $\mathcal{P}^{live}$ states that such a synchronous phase will eventually occur. Based on the results of [15], it is easy to show that this predicate is implementable in the partially synchronous model provided that a majority of processes eventually behave correctly, that is, suffer no faults after some point in time. As stated in Sect. 2.2, the algorithm presented in [15] guarantees that in any round $r > GSR$, all message sent by correct processes to correct processes are received in that round. This, and the majority of correct processes assumption, ensure rounds of the HO model where all processes can hear from a majority of processes. In these rounds a unique coordinator can be chosen by, for example, selecting the process with the highest ID from those that can be heard of.

Many of the existing consensus algorithms follow this template; that is, only a single phase is necessary to reach consensus once the system has been stabilized.[5] Accordingly, the proposed approach verifies a restricted version of termination, namely:

[5] In Sect. 7, we mention a possible extension of our approach to deal with algorithms that do no have this property.

– *Termination*: All processes decide in phase $\phi$ if $P^{sync}(\phi)$ holds.

To distinguish this property from the original definition of termination, we call it *simultaneous termination*. Clearly simultaneous termination implies termination.

## 3 Verification of agreement

In this section, we present our approach to verifying agreement for a pair of a consensus algorithm and a communication-coordinator predicate $\mathcal{P}^{safe} \triangleq \forall \phi > 0 : P^{safe}(\phi)$. The verification of termination is discussed in Sect. 4.

Our reasoning consists of two levels. Section 3.1 presents the phase-level reasoning, which shows that agreement verification can be accomplished by examining only single phases of algorithm execution. Section 3.2 then describes how model checking can be used to analyze the single phases at the round level.

### 3.1 Phase level analysis

At the upper-level of our reasoning, we perform a phase-wise analysis, rather than round-wise. We define a *configuration* as a $n$+1-tuple consisting of the states of the $n$ processes and the phase number. Let $\mathcal{C}$ be the set of all possible configurations; that is,

$$\mathcal{C} \triangleq \mathcal{S}_1 \times \cdots \times \mathcal{S}_n \times \mathbb{N}^+$$

where $\mathcal{S}_p$ is the set of possible states of a process $p$ and $\mathbb{N}^+$ is the set of positive integers. Given a configuration $c = \langle s_1, \ldots, s_n, \phi \rangle \in \mathcal{C}$, we denote by $\phi(c)$ the phase number $\phi$ of $c$. It should be noted that the state $s_p$ of a process $p$ is a value assignment to the variables of $p$. Hence any set of configurations can be represented by a predicate over the process variables of all processes and $\phi$; that is, the predicate represents the set of configurations for which it evaluates to true. We therefore use the notions of a set of configurations and of such a predicate interchangeably.

Let $Val$ be the set of values that may be proposed. We define a ternary relation $R \subseteq \mathcal{C} \times 2^{Val} \times \mathcal{C}$ as follows: $\langle c, d, c' \rangle \in R$ iff the algorithm and $P^{safe}(\phi(c))$ permit a phase such that: (1) $c$ is the configuration at the beginning of that phase (phase $\phi(c)$), (2) $d$ is the set of all values decided in the phase, and (3) $c'$ is the configuration at the beginning of the next phase (phase $\phi(c')$). By definition $\phi(c) + 1 = \phi(c')$ if $\langle c, d, c' \rangle \in R$.

Let $Init$ be the set of the configurations that can occur at the beginning of phase 1. We define a *run* as an infinite sequence $c_1 d_1 c_2 d_2 \cdots (c_i \in \mathcal{C}, d_i \in 2^{Val})$ such that $c_1 \in Init$ and $\langle c_i, d_i, c_{i+1} \rangle \in R$ for all $i \geq 1$. We let $Run$ denote the set of all runs. Let $Reachable$ be the set of all configurations that can occur in a run; that is, $Reachable \triangleq \{c \mid \exists c_1 d_1 c_2 d_2 \cdots \in Run, \exists i \geq 1 : c = c_i\}$. We say that a configuration $c$ is *reachable* iff $c \in Reachable$. Agreement holds iff:

$$\forall c_1 d_1 c_2 d_2 \cdots \in Run : \left| \bigcup_{i>0} d_i \right| \leq 1 \tag{2}$$

In determining whether Formula (2) holds or not, our verification approach does not individually explore every infinite run. Rather, it collectively examines all phases that can occur in a run. The notion of *univalence* plays here a crucial role. A configuration is said to be *univalent* if there is only one value that can be decided from that configuration [16]. If the configuration is univalent and $v$ is the only value that can be decided, then the configuration is said to be $v$-*valent*. Formally:

**Definition 1** A configuration $c_i$ is $v$-*valent* iff $\bigcup_{j \geq i} d_j = \emptyset$ or $\bigcup_{j \geq i} d_j = \{v\}$ holds for every sequence $c_i d_i c_{i+1} d_{i+1} \cdots$ such that $\forall j \geq i : \langle c_j, d_j, c_{j+1} \rangle \in R$.

By definition, if $c$ is a $v$-valent configuration, then the configuration $c'$ in the next phase, i.e., $c'$ such that $\langle c, d, c' \rangle \in R$, is also $v$-valent. That is, in a run $c_1 d_1 c_2 d_2 \ldots$, if $c_i$ is $v$-valent, then every configuration $c_j$ such that $j > i$ is also $v$-valent.

Agreement holds if, whenever some processes decide in a phase, the decided values are the same (say $v$), and the configuration at the beginning of the next phase is $v$-valent. Formally, agreement holds if:

$$\forall c \in Reachable : \forall \langle c, d, c' \rangle \in R :$$
$$d = \emptyset \vee \exists v : (d = \{v\} \wedge c' \text{is } v\text{-valent})$$

In words, $\langle c, d, c' \rangle$ corresponds to any phase that can occur, $d = \emptyset$ means that no decision is made in that phase, and $\exists v : (d = \{v\} \wedge c' \text{is } v\text{-valent})$ means that a single value $v$ is decided in that phase and that the next phase starts with a $v$-valent configuration (and so does any successive phase).

It should be noted that this formula only refers to individual phase-level transitions from the configurations in $Reachable$, rather than to runs. This property is critical for reducing the verification problem to a model checking problem of single phases. However, it is impractical to directly check this formula, because roughly speaking, obtaining $Reachable$ is as hard as examining all runs.

The key here is to use, instead of $Reachable$, its over-approximation. An over-approximation of the set of reachable states is usually referred to as an *invariant*.

**Definition 2** A set of configurations is an *invariant* iff it contains all reachable configurations.

In other words, an invariant is a predicate that is true in all reachable configurations.

In the agreement verification, we assume that we already have an invariant $Inv$ and a predicate $U(v)$ that specifies

(a subset of) $v$-valent reachable configurations. Indeed, we will have to validate $Inv$ and $U(v)$. Formally they must satisfy the following properties.

**Assumption 1** ($Inv$) A set $Inv$ of configurations is an invariant; that is, $Reachable \subseteq Inv$.

**Assumption 2** ($U(v)$) For $v \in Val$, $U(v)$ is a set of configurations such that any configuration $c \in U(v)$ is either (i) reachable and $v$-valent or (ii) unreachable.

*Example 1* In *LastVoting* the variables $commit_p$ and $ready_p$ are initially set to `false` (lines 4, 5), updated only when a process $p$ is a coordinator (lines 14, 28, 37, 38) and always reset to `false` at the end of every phase (lines 37, 38). The value of the timestamp $ts_p$ is at most $\phi - 1$ when a phase $\phi$ starts. Hence the following predicate is always true at the beginning of every phase $\phi$:

$$\begin{aligned} Inv \triangleq\ & \forall p \in \Pi : \\ & \wedge\ commit_p = \texttt{false} \\ & \wedge\ ready_p = \texttt{false} \\ & \wedge\ ts_p < \phi \end{aligned}$$

*Example 2* In many consensus algorithms, a majority quorum of processes is used to "lock" a decision value. This is also true of the *LastVoting* algorithm. A reachable configuration is $v$-valent if a majority of processes have the same estimate $v$ for the decision value and have greater timestamps than the other processes. Thus we have:

$$\begin{aligned} U(v) \triangleq\ & \exists Q \subseteq \Pi : \\ & \wedge\ |Q| > n/2 \\ & \wedge\ \forall p \in Q : (x_p = v \wedge \forall q \in \Pi \backslash Q : ts_p > ts_q) \end{aligned}$$

Given $Inv$ and $U(v)$, we can use the following theorem to verify the algorithm against agreement.

**Theorem 1** *Agreement holds if:*

$$\begin{aligned} &\forall c \in Inv : \forall \langle c, d, c' \rangle \in R : \\ &\ d = \emptyset \vee \exists v : (d = \{v\} \wedge c' \in U(v)) \end{aligned} \tag{3}$$

*Proof* Let $c_1 d_1 c_2 d_2 \cdots$ be any run. We prove that (3) implies (2) by showing that $|\bigcup_{i>0} d_i| \leq 1$. Since every $c_i$ is reachable and thus contained in $Inv$, (3) implies that for any $\langle c_i, d_i, c_{i+1} \rangle$, either $d_i = \emptyset$ or $\exists v : (d_i = \{v\} \wedge c_{i+1} \in U(v))$.

If $d_i = \emptyset$ for all $i \geq 1$, then $|\bigcup_{i>0} d_i| = 0$ trivially holds. Now suppose that some nonempty $d_i$ exists. Let $d_l$ be the first such nonempty $d_i$; that is, $d_l \neq \emptyset$ and $d_1 = d_2 = \cdots = d_{l-1} = \emptyset$. Then (3) implies $d_l = \{v\}$ for some $v$ and $c_{l+1} \in U(v)$. Because $c_{l+1}$ is reachable, Assumption 2 implies that it is $v$-valent. Since $c_{l+1}$ is $v$-valent, any $c_i$ such that $i > l + 1$ is also $v$-valent and thus $d_i = \{v\}$ or $d_i = \emptyset$ for any $i \geq l + 1$. As a result, $\bigcup_{i>0} d_i = \{v\}$. □

This theorem leads directly to the following verification steps:

Given: Algorithm, $n$, $P^{safe}(\phi)$, $Inv$, $U(v)$.
Step A1: Check if (3) holds or not. If it holds, then agreement holds.
Step A2: If (3) does not hold, then further analysis is needed, because in this case (i) the consensus algorithm is incorrect with respect to $P^{safe}(\phi)$ or (ii) $Inv$ and $U(v)$ are too large or too small, respectively. Case (i) can be further divided into two cases: the algorithm is indeed incorrect or $P^{safe}(\phi)$ is too weak.

Our verification approach is sound but not complete; that is, the algorithm under verification is proven to be correct only when it is correct, but there is no guarantee that a conclusive answer is always obtained. This limitation is inevitable because we rely on imperfect information about the algorithm, represented by $Inv$ and $U(v)$. Thus the practical success of our approach largely depends on these predicates. Through case studies, we will show that it is not difficult to derive appropriate $Inv$ and $U(v)$ from consensus algorithms.

Theoretically, on the other hand, if agreement is indeed satisfied, then some $Inv$ and $U(v)$ always exist such that (3) holds. That is, there always exist $Inv$ and $U(v)$ that are accurate enough to make the procedure complete, provided that they are expressible in the logic we use for model checking.

**Theorem 2** *Suppose that $Inv = Reachable$ and that $U(v)$ contains every state that is both reachable and $v$-valent. Then agreement holds if and only if (3) holds.*

*Proof* Since the proof of Theorem 1 directly applies to the "if" part, it suffices to prove the "only-if" part. Suppose that (3) does not hold. Then there is some $\langle c, d, c' \rangle \in R$ such that: $c$ is reachable; $d \neq \emptyset$; $\forall v : (d \neq \{v\} \vee c' \notin U(v))$. Hence we have two cases to consider: (i) $|d| > 1$ or (ii) $d = \{v\}$ and $c' \notin U(v)$ for some $v$. In case (i), agreement is trivially violated. In case (ii), since $c'$ is reachable and $U(v)$ contains every reachable $v$-valent state, $c'$ is not $v$-valent, meaning that a run exists in which both $v$ and another value $v'$ different from $v$ are decided. □

### 3.2 Model checking of single phases

This section shows how *model checking* can be used to determine if Formula (3) holds or not. Model checking is the process of exploring a state transition system to determine whether or not a given property holds. Since our problem involves only single phases, we only need to consider $k$ consecutive state transitions of the consensus algorithm, where $k$ is the number of rounds per phase (see Sect. 2.2).

The behavior of the system in a single phase can be represented as a tuple $\langle c^1 \mathbf{ho}^1 \mathbf{dv}^1 c^2 \mathbf{ho}^2 \mathbf{dv}^2 \cdots c^k \mathbf{ho}^k \mathbf{dv}^k c^{k+1}, \mathbf{Coord} \rangle$, where

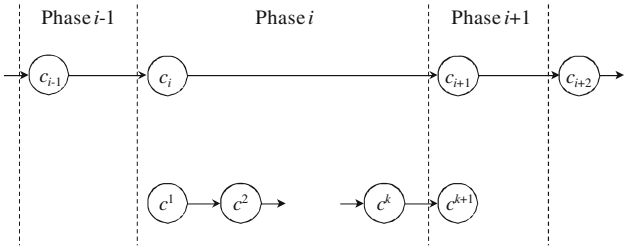


**Fig. 2** Transitions of configurations at the phase level (*top*) and at the round level (*bottom*)

- $c^i$ $(1 \leq i \leq k)$ is the configuration at the beginning of the $i$-th round of the phase, while $c^{k+1}$ is the configuration at the beginning of the first round of the next phase (see Fig. 2). Hence $\phi(c^1) = \phi(c^2) = \cdots = \phi(c^k) = \phi(c^{k+1}) - 1$.
- $\mathbf{ho}^i$ $(1 \leq i \leq k)$ is a collection of $n$ HO sets—one per process—in the $i$-th round.
- $\mathbf{dv}^i \triangleq \langle dv_1^i, \ldots, dv_n^i \rangle$, where $dv_p^i \in Val \cup \{?\}$ is the value decided by each process $p$ in the $i$-th round. If a process $p$ does not decide in the round, then $dv_p^i = ?$
- **Coord** is a collection of $n$ coordinators—one per process—in the phase.

We call such a tuple a *one-phase execution* iff it is consistent with the consensus algorithm and $P^{safe}(\phi(c^1))$.[6] By definition, $\langle c, d, c' \rangle \in R$ iff there is at least one one-phase execution $\langle c^1 \mathbf{ho}^1 \mathbf{dv}^1 \cdots c^k \mathbf{ho}^k \mathbf{dv}^k c^{k+1}, \mathbf{Coord} \rangle$ such that $c = c^1$, $c' = c^{k+1}$, and $d = \left( \bigcup_{p \in \Pi, 1 \leq i \leq k} \{dv_p^i\} \right) \setminus \{?\}$.

Our model checking problem is described as follows: Determine if, for all one-phase executions $\langle c^1 \mathbf{ho}^1 \mathbf{dv}^1 \cdots c^k \mathbf{ho}^k \mathbf{dv}^k c^{k+1}, \mathbf{Coord} \rangle$ such that $c^1 \in Inv$, the following condition holds:

$$d = \emptyset \ \vee \ \exists v : (d = \{v\} \ \wedge \ c^{k+1} \in U(v)) \tag{4}$$

where $d = \left( \bigcup_{p \in \Pi, 1 \leq i \leq k} \{dv_p^i\} \right) \setminus \{?\}$.

Clearly (3) holds iff (4) holds for all one-phase executions that start with a configuration in $Inv$.

This model checking problem only concerns exactly $k$ consecutive transitions. Because of this, *bounded model checking* [13] can be most effectively used to solve it. The idea of bounded model checking is to reduce the model checking problem to the *satisfiability problem* for a formula in some logic.

In order to check (4) with this model checking technique, we proceed as follows. We construct three mathematical formulas:

- $X$, which exactly represents all one-phase executions. That is, $X$ evaluates to true iff the variable values represent a one-phase execution.
- $INV$, which specifies that $c^1 \in Inv$.
- $Agr$, which specifies that (4) holds.

These formulas are built from variables that represent the elements of a one-phase execution.

Then we check the satisfiability of:

$$X \wedge INV \wedge \neg Agr \tag{5}$$

This formula can only be satisfied by a value assignment corresponding to a one-phase execution that (i) starts from $Inv$ *and* (ii) for which (4) does not hold. Therefore every one-phase execution that starts from $Inv$ meets (4) iff Formula (5) is unsatisfiable.

Step A1 and Step A2 can now be replaced with Step B1 and Step B2, respectively.

Step B1: Check the satisfiability of Formula (5). If no satisfying value assignment exists, then every one-phase execution starting from a configuration in $Inv$ satisfies (4), meaning that (3) holds. As a result, it is guaranteed that agreement holds.

Step B2: On the other hand if there is a satisfying assignment, further analysis is needed to obtain a conclusive answer (see Step A2).

Basically these formulas are specific to the algorithm and/or the communication-coordinator predicate. In the rest of the section we show how to construct these formulas with *LastVoting* as a running example.

### 3.2.1 Logic and variables

To ensure that the satisfiability of the formula is decidable, we use a logic that allows boolean combinations of linear (in)equalities and boolean expressions with integer and boolean variables. A quantifier is allowed only when it is an existential quantifier that can be moved to the front of the formula without changing the meaning of the formula. A *Satisfiability Modulo Theories* (SMT) solver, such as Yices [14], can be used to check the satisfiability of formulas in this logic.

To make the formula concise, we use the IF- THEN- ELSE and tuple ($\langle \cdots \rangle$) constructs and several "macros". These macros include:

- CARDINALITY$(x \in \langle e_1, e_2, \cdots \rangle : P(x))$ : The number of $e_i$ such that $P(e_i) = \texttt{true}$.
- CHOOSE$(x \in \langle e_1, e_2, \cdots \rangle : P(x))$ : Some $e_i$ such that $P(e_i) = \texttt{true}$; $\perp$ if no such $e_i$ exists.

[6] Here whether $c^1$ is reachable or not is irrelevant. In other words, a one-phase execution specifies how the system would behave in a phase, provided that the phase begins with $c^1$.

Before checking the satisfiability, these macros are translated into the raw logic. The macro CARDINALITY is translated into a term that has an integer value. A term that uses the macro CHOOSE, e.g., $y = \text{CHOOSE}(x \in \langle e_1, e_2, \cdots \rangle : P(x))$, is translated as follows:

$$(P(y) \wedge (y = e_1 \vee y = e_2 \vee \cdots)) \\ \vee (y = \bot \wedge \neg P(e_1) \wedge \neg P(e_2) \wedge \cdots)$$

This is added as a conjunct to the formula involving that term.

Formula (5) is constructed from variables that represent the elements of a one-phase execution, namely, $c^i (1 \leq i \leq k+1)$, $\mathbf{ho}^i$, $\mathbf{dv}^i (1 \leq i \leq k)$, and **Coord**.

For the *LastVoting* algorithm, the variables involved are as follows: The configuration $c^i$ is represented by: $x_p^i$, $vote_p^i, ts_p^i, commit_p^i, ready_p^i$ for $p \in \Pi$ and $\phi^i$. The collection of HO sets $\mathbf{ho}^i$ is represented by boolean variables $ho_{p,q}^i$ for $p, q \in \Pi$ such that $ho_{p,q}^i = \texttt{true}$ iff $p$ hears $q$ in the $i$-th round. The collection of decision values $\mathbf{dv}^i$ is represented by $dv_p^i$ for $p \in \Pi$. The collection of coordinators **Coord** is represented by $Coord_p$ for $p \in \Pi$, where $Coord_p$ specifies $p$'s coordinator.

Since the logic of the formula only allows integer and boolean variables, it is necessary to convert the domain of these variables into the integer domain, unless they are of boolean type. We do this by mapping $Val$ to $[1..\infty]$ and ? to 0. As a result, the domain $Dom$ of the variables is represented as follows:

$$\begin{aligned} &Dom \triangleq \\ &\wedge \bigwedge_{p \in \Pi, 1 \leq i \leq k+1} (x_p^i \geq 1 \wedge vote_p^i \geq 0 \wedge ts_p^i \geq 0) \\ &\wedge \bigwedge_{1 \leq i \leq k+1} \phi^i \geq 1 \\ &\wedge \bigwedge_{p \in \Pi, 1 \leq i \leq k} dv_p^i \geq 0 \\ &\wedge \bigwedge_{p \in \Pi} 1 \leq Coord_p^i \leq n \end{aligned}$$

Special values are represented by distinct constant integers. Such values include: $\bot$, $empty$, and $ack$. The value $\bot$ is used to represent an invalid value. $empty$ and $ack$ represent an empty message and acknowledge message, respectively. One could map these symbols to any integers, unless a variable value simultaneously has two meanings. We map $\bot, empty, ack$ to $-1, -2, 1$, respectively.

### 3.2.2 Formula X

$X$ is composed as follows:

$$X \triangleq Dom \wedge T^1 \wedge T^2 \wedge \cdots \wedge T^k \wedge Safe$$

where

- $T^i$ is a mathematical representation of the $i$-th round of the algorithm.
- $Safe$ is a mathematical representation of HO sets and all possible coordinators that are permitted by $P^{safe}(\phi)$.

Formula $T^i$ evaluates to true iff $c^i$ and $c^{i+1}$—the configurations at the beginning of the $i$-th and $i$+1-th rounds—are consistent with the algorithm under verification. For example, the third round of *LastVoting* (round $4\phi - 1$) is represented as follows:

$$\begin{aligned} &SndMsg_{p,q}^3 \triangleq \\ &\quad \text{IF } ts_p^3 = \phi^3 \wedge q = Coord_p \text{ THEN } ack \text{ ELSE } empty \\ &RcvMsg_{p,q}^3 \triangleq \\ &\quad \text{IF } ho_{p,q}^3 = \texttt{true} \text{ THEN } SndMsg_{q,p}^3 \text{ ELSE } \bot \\ &T_p^3 \triangleq \\ &\quad \text{IF} \\ &\quad \wedge p = Coord_p \\ &\quad \wedge \text{CARDINALITY}(Msg \in \langle RcvMsg_{p,1}^3, \ldots, \\ &\qquad RcvMsg_{p,n}^3 \rangle : Msg = ack) \geq \lceil \tfrac{n+1}{2} \rceil \\ &\quad \text{THEN} \\ &\quad \wedge ready_p^4 = \texttt{true} \\ &\quad \wedge \langle x_p^4, vote_p^4, commit_p^4, ts_p^4 \rangle \\ &\qquad = \langle x_p^3, vote_p^3, commit_p^3, ts_p^3 \rangle \\ &\quad \wedge dv_p^3 = ? \\ &\quad \text{ELSE} \\ &\quad \wedge \langle x_p^4, vote_p^4, ready_p^4, commit_p^4, ts_p^4 \rangle \\ &\qquad = \langle x_p^3, vote_p^3, ready_p^3, commit_p^3, ts_p^3 \rangle \\ &\quad \wedge dv_p^3 = ? \\ &T^3 \triangleq T_1^3 \wedge \cdots \wedge T_n^3 \wedge \phi^4 = \phi^3 \end{aligned}$$

$SndMsg_{p,q}$ and $RcvMsg_{p,q}$ express the message sent by $p$ to $q$ and the message received by $p$ from $q$, respectively. Here $empty$ and $\bot$ have different meanings; $\bot$ means that the sender process is not heard of (no message received), whereas $empty$ means that a message is received but it is empty. $T_p^3$ represents the state transition of process $p$ at round 3. It specifies that if $p$ itself is $p$'s coordinator and receives acks from a majority of processes, then it sets $ready_p$ to `true`; $p$ does not change other variables and makes no decision (specified as $dv_p^3 = ?$). Formula $T^3$ is a conjunction of $T_p^3$'s and the term $\phi^4 = \phi^3$. The term $\phi^4 = \phi^3$ states that $c^3$ and $c^4$ are in the same phase.

The *LastVoting* algorithm requires no condition on HO sets and coordinators to satisfy safety; that is, $P^{safe}(\phi) =$ `TRUE` is sufficient. In this case we simply have:

$$Safe \triangleq \texttt{true}$$

### 3.2.3 Formula INV

$INV$ specifies that $c^1 \in Inv$. For example, consider $Inv$ shown in Example 1. In this case we have:

$$INV \triangleq \bigwedge_{p\in\Pi} \wedge commit_p^1 = \texttt{false} \wedge ready_p^1 = \texttt{false} \wedge ts_p^1 < \phi^1$$

### 3.2.4 Formula Agr

*Agr* specifies that (4) holds. For example, consider *Agr* shown in Example 2. In this case we have:

$$\bar{v} \triangleq \text{CHOOSE}(v \in \langle dv_1^1, \cdots, dv_n^k\rangle : v \neq ?)$$
$$\bar{U}' \triangleq \bigvee_{Q\subseteq\Pi:|Q|>n/2} \bigwedge_{p\in Q} \left( x_p^{k+1}{=}\bar{v} \wedge \bigwedge_{q\in\Pi\setminus Q} ts_p^{k+1} > ts_q^{k+1} \right)$$
$$Agr \triangleq \vee\ \bar{v} = \bot \vee\ \bar{v} \neq \bot \wedge \bigwedge_{p\in\Pi, 1\leq i\leq k} (dv_p^i = ? \vee dv_p^i = \bar{v}) \wedge \bar{U}'$$

Note that formula *Arg*, except the $\bar{U}'$ part, is independent of the algorithm (if we ignore the dependency on $k$).

Value $\bar{v}$ is one of the values decided in the phase. Formula $\bar{U}'$ represents whether $U(\bar{v})$ holds or not for configuration $c^{k+1}(= c^5)$. Value $\bar{v}$ is $\bot$ if no decision has been made. Hence $\bar{v} = \bot$ holds iff $d = \emptyset$, where $d$ is the set of all decided values. In *Agr*, the term $\bigwedge_{p\in\Pi, 1\leq i\leq k}(...)$ represents that $d = \emptyset$ or $d = \{\bar{v}\}$. Thus the last disjunct represents that $d = \{\bar{v}\}$ and $c^{k+1} \in U(\bar{v})$. The verification of Formula (5) for *LastVoting* is discussed in Sect. 6.

## 4 Verification of termination

This section presents a method for verifying a pair (algorithm, communication-coordinator predicate $\mathcal{P} \triangleq \mathcal{P}^{safe} \wedge \mathcal{P}^{live}$) against termination. As stated in Sect. 2, we assume that $\mathcal{P}^{live}$, the condition for termination, is of form:

$$\mathcal{P}^{live} \triangleq \exists\phi > 0 : P^{sync}(\phi)$$

The proposed method verifies simultaneous termination, that is, whether all processes decide in a phase $\phi$ such that $P^{sync}(\phi)$ is true. Clearly, simultaneous termination implies termination.

Let $R^{sync} \subseteq \mathcal{C} \times 2^{\Pi} \times \mathcal{C}$ be a ternary relation such that $\langle c, \pi, c'\rangle \in R^{sync}$ iff a one-phase execution from $c$ to $c'$ exists such that $\pi$ is the set of processes that decide and $P^{sync}(\phi(c))$ holds for that one-phase execution. Hence simultaneous termination is satisfied iff:[7]

$$\forall c \in Reachable : (\forall\langle c, \pi, c'\rangle \in R^{sync} : \pi = \Pi) \quad (6)$$

[7] Note that we safely exclude the exceptional case where no $\langle c, \pi, c'\rangle \in R^{sync}$ exists for some $c$, because $P^{safe}(\phi) \wedge P^{sync}(\phi)$ is not the constant predicate FALSE.

**Theorem 3** *Termination holds if:*

$$\forall c \in Inv : (\forall\langle c, \pi, c'\rangle \in R^{sync} : \pi = \Pi) \quad (7)$$

*Proof* Because $Reachable \subseteq Inv$ (Assumption 1), (7) implies (6), that is, simultaneous termination. □

Whether (7) holds or not can be determined using bounded model checking, as was done for agreement verification. Note that $\langle c, \pi, c'\rangle \in R^{sync}$ corresponds to one or several one-phase executions $\langle c^1\mathbf{ho}^1\mathbf{dv}^1 \cdots c^k\mathbf{ho}^k\mathbf{dv}^k c^{k+1}, \mathbf{Coord}\rangle$ such that $c = c^1$, $c' = c^{k+1}$, and $\pi = \{p \mid \exists i, 1 \leq i \leq k : dv_p^i \neq ?\}$. Hence the problem needed to be solved by model checking is to determine if, for all one-phase executions such that:

- $c^1 \in Inv$; and
- $P^{sync}(\phi(c^1))$ holds for the HO sets, $\mathbf{ho}^1, \mathbf{ho}^2, \ldots, \mathbf{ho}^k$, and the coordinators, **Coord**,

the following condition (8) holds:

$$\forall p \in \Pi, \exists i, 1 \leq i \leq k : dv_p^i \neq ? \quad (8)$$

In words, (8) states that every process decides in some round of a phase. By definition, (7) holds iff (8) holds for all the above one-phase executions.

This decision problem is reduced to the satisfiability problem. The formula to be checked is:

$$X \wedge INV \wedge Sync \wedge \neg Term$$

where:

- *Sync* represents $P^{sync}(\phi(c^1))$. Formula *Sync* is satisfied iff $P^{sync}(\phi(c^1))$ holds for the HO sets and the coordinators represented by $ho_{p,q}^i$ and $Coord_p$.
- *Term* is satisfied iff (8) holds. We have:

  $$Term \triangleq \bigwedge_{p\in\Pi} \bigvee_{1\leq i\leq k} dv_p^i \neq ?$$

  Note that *Term* is independent of the algorithm, except in that it depends on $k$.

The formula $X \wedge INV \wedge Sync \wedge \neg Term$ can be satisfied by, and only by a one-phase execution that (i) starts from a configuration in *Inv*, (ii) satisfies $P^{sync}(\phi)$, *and* (iii) for which (8) does not hold. Therefore every one-phase execution examined satisfies (8) iff no satisfying assignment exists. As a result, termination can be verified as follows:

Given: Algorithm, $n$, $P^{safe}(\phi)$, $P^{sync}(\phi)$, *Inv*.
Step C1: Check the satisfiability of $X \wedge INV \wedge Sync \wedge \neg Term$. If no satisfying value assignment exists, then termination is guaranteed.

Step C2: If the formula is satisfiable, then it means that (i) the algorithm is incorrect with respect to $P^{sync}(\phi)$ or (ii) $Inv$ is too large. Case (i) can be further divided into three cases: the algorithm is indeed incorrect; the algorithm satisfies termination but not simultaneous termination; or $P^{sync}(\phi)$ is too weak. Further analysis is required to obtain a conclusive answer.

*Example 3* Consider $P^{sync}(\phi)$ for *LastVoting (Formula (1))*. We have:

$$
\begin{aligned}
&Sync \triangleq \\
&\bigvee_{p\in\Pi} \\
&\quad \wedge\ p = Coord_1 = \cdots = Coord_n \\
&\quad \wedge\ \text{CARDINALITY}(ho \in \langle ho^1_{p,1}, \ldots, ho^1_{p,n}\rangle : \\
&\qquad ho = \texttt{true}) \geq \lceil \tfrac{n+1}{2} \rceil \\
&\quad \wedge\ \text{CARDINALITY}(ho \in \langle ho^3_{p,1}, \ldots, ho^3_{p,n}\rangle : \\
&\qquad ho = \texttt{true}) \geq \lceil \tfrac{n+1}{2} \rceil \\
&\quad \wedge\ \bigwedge_{q\in\Pi} ho^2_{q,p} = \texttt{true} \\
&\quad \wedge\ \bigwedge_{q\in\Pi} ho^4_{q,p} = \texttt{true}
\end{aligned}
$$

Note that if $Inv$ is equal to $Reachable$, then the above procedure can determine exactly whether simultaneous termination holds or not. This follows from the fact that the necessary and sufficient condition for simultaneous termination is (6), which is identical to (7) except $Inv$ replaced with $Reachable$.

## 5 Validating $Inv$ and $U(v)$

So far we have assumed that $Inv$ (see Example 1) and $U(v)$ (see Example 2) satisfy Assumption 1, respectively Assumption 2 (see Sect. 3.1). Here we present automatic procedures that can check that $Inv$ and $U(v)$ indeed satisfy the corresponding assumptions. We show that with some minor modifications, the model checking technique described in the previous sections can be used for this purpose.

### 5.1 Validating $Inv$

Here we discuss the validation of Assumption 1, i.e., the invariance of $Inv$. The validation uses the following theorem.

**Theorem 4** *Suppose that $Inv$ is a set of configurations. $Inv$ is an invariant if the following two conditions hold:*

$$Init \subseteq Inv \tag{9}$$

$$\forall c \in Inv : (\forall \langle c, d, c'\rangle \in R : c' \in Inv) \tag{10}$$

*Proof* By induction we show that for any $c_1 d_1 c_2 d_2 \cdots \in Run$, $c_i \in Inv$ for any $i \geq 1$. By definition of $Run$, $c_1 \in Init$. By (9) $c_1 \in Inv$. Suppose that $c_i \in Inv$ for some $i \geq 1$. Then $c_{i+1} \in Inv$ by (10). □

In the formal verification literature, such an invariant that can be proven by induction is often referred to as an *inductive invariant*. The use of bounded model checking for proving an inductive invariant is studied in, for example, [32,34]. The method presented in this section follows the same approach; that is, it determines whether $Inv$ is an inductive invariant or not.

#### 5.1.1 Test of (9) (Base Case)

Testing (9) can be done by searching for a configuration that is in $Init$ but not in $Inv$. This can be conducted by checking the satisfiability of the following formula:

$$Dom \wedge INIT \wedge \neg INV$$

where $INIT$ specifies that $c^1 \in Init$.

Since $\neg INV$ is satisfied iff $c^1 \notin Inv$, the above formula is true iff $c^1 \in Init \backslash Inv$. Hence condition (9) holds iff no satisfying assignment exists.

*Example 4* Consider the *LastVoting* algorithm for example. In this case we have:

$$
\begin{aligned}
INIT \triangleq\ &\wedge \bigwedge_{p\in\Pi} \\
&\wedge\ vote^1_p = ? \\
&\wedge\ commit^1_p = \texttt{false} \\
&\wedge\ ready^1_p = \texttt{false} \\
&\wedge\ ts^1_p = 0 \\
&\wedge\ \phi^1 = 1
\end{aligned}
$$

#### 5.1.2 Test of (10) (Inductive Step)

The problem of testing whether (10) holds or not is equivalent to determining if for all one-phase executions $\langle c^1 \mathbf{ho}^1 \mathbf{dv}^1 \cdots c^k \mathbf{ho}^k \mathbf{dv}^k c^{k+1}, \mathbf{Coord}\rangle$ we have:

$$c^1 \in Inv \longrightarrow c^{k+1} \in Inv$$

This problem can be reduced to the satisfiability problem of the following formula:

$$X \wedge INV \wedge \neg INV'$$

where $INV'$ is satisfied iff $c^{k+1} \in Inv$. Thus if there is no satisfying solution, then every one-phase execution starting from a configuration in $Inv$ ends with another configuration in $Inv$, i.e., (10) holds.

$INV'$ is almost identical to $INV$, except that every first-round version variable $var^1$ in $INV$ is now replaced with its $k$+1-th version $var^{k+1}$.

The validation of Assumption 1 ($Inv$) for *LastVoting* is discussed in Sect. 6.

### 5.2 Validating $U(v)$ with Assumption 1

Here we discuss the validation of $U(v)$, i.e., the fact that $U(v)$ represents $v$-valent configurations. We assume that $Inv$ correctly represents an invariant; that is, $Reachable \subseteq Inv$ (Assumption 1).

**Theorem 5** *Suppose that $v \in Val$ and that $U(v)$ is a set of configurations. Any $c \in U(v)$ is either reachable and $v$-valent or unreachable if:*

$$\begin{array}{l} \forall c \in Inv : \forall \langle c, d, c' \rangle \in R : \forall v \in Val : \\ \quad c \in U(v) \longrightarrow (d = \emptyset \vee d = \{v\}) \wedge c' \in U(v) \end{array} \tag{11}$$

*Proof* It suffices to show that when (11) holds, for any $c_i$ ($i \geq 1$) in any run $c_1 d_1 c_2 d_2 \cdots \in Run$, if $c_i \in U(v)$, then $c_i$ is $v$-valent. Since $Reachable \subseteq Inv$, $c_i \in Inv$ for any $i \geq 1$. If $c_i \in U(v)$, then because of (11), $d_i = \emptyset$ or $d_i = \{v\}$ and $c_{i+1} \in U(v)$. By induction, for any $j \geq i$, $d_j = \emptyset$ or $d_j = \{v\}$. Hence $c_i$ is $v$-valent. □

Again, bounded model checking can be used to check if (11) holds. The problem to be solved is to determine whether every one-phase execution $\langle c^1 \mathbf{ho}^1 \mathbf{dv}^1 \cdots c^k \mathbf{ho}^k \mathbf{dv}^k c^{k+1}, \mathbf{Coord} \rangle$ such that $c^1 \in Inv$ satisfies the following condition for any $v \in Val$:

$$c^1 \in U(v) \longrightarrow (d = \emptyset \vee d = \{v\}) \wedge c^{k+1} \in U(v) \tag{12}$$

where $d = \left( \bigcup_{p \in \Pi, 1 \leq i \leq k} \{dv_p^i\} \right) \setminus \{?\}$.

The problem is reduced to the satisfiability problem of:

$$X \wedge INV \wedge \exists \hat{v} : (\hat{v} \in Val \wedge \neg Univ(\hat{v}))$$

where $\hat{v}$ is a newly introduced variable and $\neg Univ(\hat{v})$ is true iff (12) does *not* hold when $v = \hat{v} \in Val$. If $X \wedge INV \wedge \exists \hat{v} : (\hat{v} \in Val \wedge \neg Univ(\hat{v}))$ is unsatisfiable, then $U(v)$ satisfies Assumption 2, because in that case (12) holds for any $v \in Val$ and for any one-phase execution that starts with a configuration in $Inv$. Note that this use of a quantifier is allowed by the logic of the formula and that the term $\hat{v} \in Val$ is represented as $\hat{v} \geq 1$ (see Sect. 3.2.1).

Formula $Univ(\hat{v})$ is constructed as follows:

$$\begin{array}{l} Univ(\hat{v}) \triangleq \\ \quad \hat{U}(\hat{v}) \longrightarrow \bigwedge_{p \in \Pi, 1 \leq i \leq k} (dv_p^i = ? \vee dv_p^i = \hat{v}) \ \wedge \ \hat{U}'(\hat{v}) \end{array}$$

where $\hat{U}(\hat{v})$ and $\hat{U}'(\hat{v})$ represent $c^1 \in U(\hat{v})$ and $c^{k+1} \in U(\hat{v})$, respectively. The term $\bigwedge_{p \in \Pi, 1 \leq i \leq k}(..)$ represents $d = \emptyset$ or $d = \{\hat{v}\}$. This term is independent of the algorithm, except in that it depends on $k$.

```
1: Initialization:
2:   x_p ∈ Val, initially v_p                {v_p is the proposed value of p.}
3:   vote_p ∈ Val ∪ {?}, initially ?
4:   voteToSend_p a Boolean, initially false
5:   ts_p ∈ ℕ, initially 0

6: Round r = 3φ − 2:
7:   S_p^r :
8:     send ⟨x_p , ts_p, Coord(p, φ)⟩ to all processes

9:   T_p^r :
10:    if (φ = 1) and #⟨∗, ∗, ∗⟩ ≥ n − α then
11:      if n − α messages received are equal to ⟨x̄, ∗, ∗⟩ then
12:        DECIDE(x̄)
13:    if p = Coord(p, φ) and
         #⟨∗, ∗, p⟩ received > max(n/2, 2α) then
14:      if the messages received are all equal to ⟨∗, 0, ∗⟩
           and, except at most α, are equal to ⟨x̄, 0, ∗⟩ then
15:        vote_p := x̄
16:      else
17:        let θ̄ be the largest θ from ⟨∗, θ, ∗⟩ received
18:        vote_p := one x̄ such that ⟨x̄, θ̄, ∗⟩ is received
19:      voteToSend_p := true

20: Round r = 3φ − 1:
21:   S_p^r :
22:     if p = Coord(p, φ) and voteToSend_p then
23:       send ⟨vote_p⟩ to all processes

24:   T_p^r :
25:     if received ⟨v⟩ from Coord(p, φ) then
26:       x_p := v ; ts_p := φ

27: Round r = 3φ:
28:   S_p^r :
29:     if ts_p = φ then
30:       send ⟨ack, x_p⟩ to all processes

31:   T_p^r :
32:     if ∃v s.t. # ⟨ack, v⟩ received > n/2 then
33:       DECIDE(v)
34:     voteToSend_p := false
```

**Fig. 3** The *Hybrid-1*($\alpha$) algorithm ($\alpha \leq \lfloor n/4 \rfloor$)

*Example 5* For $U(v)$ shown in Example 2, we have:

$$\hat{U}(\hat{v}) \triangleq \bigvee_{Q \subseteq \Pi : |Q| > n/2} \ \bigwedge_{p \in Q} \left( x_p^1 = \hat{v} \wedge \bigwedge_{q \in \Pi \setminus Q} ts_p^1 > ts_q^1 \right)$$

$$\hat{U}'(\hat{v}) \triangleq \bigvee_{Q \subseteq \Pi : |Q| > n/2} \ \bigwedge_{p \in Q} \left( x_p^{k+1} = \hat{v} \wedge \bigwedge_{q \in \Pi \setminus Q} ts_p^{k+1} > ts_q^{k+1} \right)$$

The validation of Assumption 2 ($U(v)$) for *LastVoting* is discussed in Sect. 6.

## 6 Case studies

### 6.1 Algorithms verified

In this section we present the results of applying the proposed approach to four consensus algorithms:

- *LastVoting* (*Paxos*) [10], see Fig. 1.
- *Hybrid-1*($\alpha$) [8], see Fig. 3.
- *CoordUniformVoting* [10], see Fig. 4.
- Simplified *CoordUniformVoting* [10], see Fig. 5.

The conditions for safety and liveness are summarized in Table 1. $Inv$ and $U(v)$ are shown in Table 2.

```
1: Initialization:
2:    x_p ∈ Val, initially v_p                {v_p is the proposed value of p.}
3:    vote_p ∈ Val ∪ {?}, initially ?

4: Round r = 3φ − 2:
5:    S_p^r :
6:       if p = Coord_p(φ) then
7:          send ⟨ x_p ⟩ to all processes

8:    T_p^r :
9:       if some message ⟨ v ⟩ is received from Coord(p, φ) then
10:         x_p := v

11: Round r = 3φ − 1:
12:   S_p^r :
13:      send ⟨ x_p ⟩ to all processes

14:   T_p^r :
15:      if all the values received are equal to v then
16:         vote_p := v

17: Round r = 3φ:
18:   S_p^r :
19:      send ⟨ vote_p ⟩ to all processes

20:   T_p^r :
21:      if at least one ⟨ v ⟩ with v ≠? is received then
22:         x_p := v
23:      if all the messages received are equal to ⟨ v ⟩ with v ≠? then
24:         DECIDE(v)
25:      vote_p := ?
```

**Fig. 4** The *CoordUniformVoting* algorithm

```
1: Initialization:
2:    x_p ∈ Val, initially v_p                {v_p is the proposed value of p.}
3:    vote_p ∈ Val ∪ {?}, initially ?

4: Round r = 2φ − 1:
5:    S_p^r :
6:       if p = Coord(p, φ) then
7:          send ⟨x_p⟩ to all processes

8:    T_p^r :
9:       if received ⟨v⟩ from Coord(p, φ) then
10:         vote_p := v

11: Round r = 2φ:
12:   S_p^r :
13:      send ⟨vote_p⟩ to all processes

14:   T_p^r :
15:      if at least one ⟨v⟩ with v ≠ ? is received then
16:         x_p := v
17:      if all messages received are equal to ⟨v⟩ with v ≠ ? then
18:         DECIDE(v)
19:      vote_p := ?
```

**Fig. 5** Simplified *CoordUniformVoting* algorithm

### 6.1.1 LastVoting

For *LastVoting*, the conditions for safety and liveness are explained in Sect. 2.4, while $Inv$ and $U(v)$ are explained in Examples 1 and 2 in Sect. 3.1.

### 6.1.2 Hybrid-1(α)

The *Hybrid-1(α)* algorithm is an improved version of the *Fast Paxos* algorithm [27]. Unlike the other three algorithms, *Hybrid-1*($\alpha$) has the design parameter $\alpha$. This algorithm is always safe if $\alpha \leq \lfloor n/4 \rfloor$.

*Hybrid-1(α)* combines a fast phase and an ordinary phase of *Fast Paxos* into the same phase. In the first round of Phase 1, if a process has received the same estimate from $n - \alpha$ processes, then the process can immediately decide (line 3). To prevent processes from deciding different values in later rounds (line 3), if $n - \alpha$ or more processes have the same proposed value, then it must be guaranteed that no process having a different estimate will be allowed to update its timestamp. That is, when at least $n - \alpha$ processes have the same estimate $v$ and the remaining processes have a timestamp equal to zero, the system must be $v$-valent. The first disjunct of the $U(v)$ states this formally. The second disjunct is the same as the $U(v)$ for *LastVoting*.

For this algorithm, we perform a parameterized verification to verify all possible values for $\alpha$ as follows: We introduce an integer variable into the formula $X$ to represent $\alpha$, instead of substituting a specific constant for it. At the same time, we add to $X$ the terms $\alpha \geq 0$ and $n \geq 4\alpha$ as conjuncts. These conjuncts guarantee that $\alpha$ stays within the permissible range.

### 6.1.3 CoordUniformVoting

The *CoordUniformVoting* algorithm can be regarded as a deterministic and coordinator-based version of the Ben-Or algorithm [1].

In the first round, every process $p$ that considers itself to be the coordinator broadcasts $x_p$, which is the estimate of the decision value. If a process receives a value from its coordinator, then it adopts the value as its own estimate. In the second round, every process $p$ broadcasts $x_p$. Then, $p$ votes for value $v$ if all received values are equal to $v$; otherwise, the process does not cast a vote. In the third round, the cast is actually performed: $p$ sends the value of $vote_p$, which is either $v$ or ?, to all. If $p$ receives at least one vote, then $p$ adopts it as its estimate. If every received message is the vote for $v$, then $p$ decides $v$.

Unlike the first two algorithms, *CoordUniformVoting* requires some synchrony condition to satisfy its safety. Specifically, as shown in Table 1, it requires that the following predicate hold for every phase $\phi$:

$$P^{safe}(\phi) \triangleq \forall p, q \in \Pi, \forall r, 0 \leq r \leq 2 : HO(p, 3\phi - r) \cap HO(q, 3\phi - r) \neq \emptyset$$

$P^{safe}(\phi)$ states that any pair of processes can hear from at least one common process. This is implementable in an asynchronous system with at most $f = \lfloor \frac{n-1}{2} \rfloor$ crash failures with reliable links (see the communication predicate for this system shown in Sect. 2.2).

The $P^{safe}(\phi)$ predicate and the condition of the **if** statement at line 4 ensure that no two processes cast different

**Table 1** Conditions for safety and liveness ($\mathcal{P}^{safe}$ and $\mathcal{P}^{live}$)

| | |
|---|---|
| (a) *LastVoting* | |
| Condition for safety | None (TRUE) |
| Condition for liveness | $\exists \phi > 0, \exists co \in \Pi, \forall p \in \Pi :$<br>$\wedge\ co = Coord(p, \phi))$<br>$\wedge\ \lvert HO(co, 4\phi - 3)\rvert > n/2$<br>$\wedge\ \lvert HO(co, 4\phi - 1)\rvert > n/2$<br>$\wedge\ co \in HO(p, 4\phi - 2)$<br>$\wedge\ co \in HO(p, 4\phi)$ |
| (b) *Hybrid-1*($\alpha$) | |
| Condition for safety | None (TRUE) |
| Condition for liveness | $\exists \phi > 0, \exists co \in \Pi, \forall p \in \Pi, \forall r, 0 \leq r \leq 2 :$<br>$\wedge\ \lvert HO(p, 3\phi - r)\rvert > \max(n/2, 2\alpha)$<br>$\wedge\ co = Coord(p, \phi)$<br>$\wedge\ co \in HO(p, 3\phi - r)$ |
| (c) *CoordUniformVoting* | |
| Condition for safety | $\forall \phi > 0, \forall p, q \in \Pi, \forall r, 0 \leq r \leq 2 :$<br>$HO(p, 3\phi - r) \cap HO(q, 3\phi - r) \neq \emptyset$ |
| Condition for liveness | $\exists \phi > 0, \exists co \in \Pi, \forall p \in \Pi :$<br>$co = Coord(p, \phi)\ \wedge\ co \in HO(p, 3\phi - 2)$ |
| (d) Simplified *CoordUniformVoting* | |
| Condition for safety | $\forall \phi > 0, \forall p, q \in \Pi, \forall r, 0 \leq r \leq 1 :$<br>$\wedge\ HO(p, 2\phi - r) \cap HO(q, 2\phi - r) \neq \emptyset$<br>$\wedge\ Coord(p, \phi) = Coord(q, \phi)$ |
| Condition for liveness | $\exists \phi > 0, \exists co \in \Pi, \forall p \in \Pi :$<br>$co = Coord(p, \phi)\ \wedge\ co \in HO(p, 2\phi - 1)$ |

**Table 2** $Inv$ and $U(v)$

| | |
|---|---|
| (a) *LastVoting* | |
| $Inv$ | $\forall p \in \Pi :$<br>$\wedge\ commit_p = \texttt{false}$<br>$\wedge\ ready_p = \texttt{false}$<br>$\wedge\ ts_p < \phi$ |
| $U(v)$ | $\exists Q \subseteq \Pi :$<br>$\wedge\ \lvert Q\rvert > n/2$<br>$\wedge\ \forall p \in Q : (x_p = v \wedge \forall q \in \Pi \backslash Q : ts_p > ts_q)$ |
| (b) *Hybrid-1*($\alpha$) | |
| $Inv$ | $\forall p \in \Pi : voteToSend_p = \texttt{false} \wedge ts_p < \phi$<br>$\vee\ \exists Q \subseteq \Pi :$<br>$\wedge\ \lvert Q\rvert \geq n - \alpha$<br>$\wedge\ \forall p \in Q : (x_p = v) \wedge \forall p \in \Pi \backslash Q : (ts_p = 0)$ |
| $U(v)$ | $\vee\ \exists Q \subseteq \Pi :$<br>$\wedge\ \lvert Q\rvert > n/2$<br>$\wedge\ \forall p \in Q : (x_p = v \wedge \forall q \in \Pi \backslash Q : ts_p > ts_q)$ |
| (c) *CoordUniformVoting* | |
| $Inv$ | $\forall p \in \Pi : vote_p = ?$ |
| $U(v)$ | $\forall p \in \Pi : x_p = v$ |
| (d) Simplified *CoordUniformVoting* | |
| $Inv$ | $\forall p \in \Pi : vote_p = ?$ |
| $U(v)$ | $\forall p \in \Pi : x_p = v$ |

**Table 3** Execution time (h: mm:ss)

| | $n=4$ | $n=5$ | $n=6$ | $n=7$ | $n=8$ | $n=9$ | $n=10$ | $n=11$ | $n=12$ | $n=13$ | $n=14$ | $n=15$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (a) *LastVoting* | | | | | | | | | | | | |
| Agreement | 0:01 | 0:02 | 0:07 | 0:27 | 1:27 | 4:53 | 21:44 | > 5 h | > 5 h | > 5 h | > 5 h | > 5 h |
| Termination | 0:00 | 0:01 | 0:01 | 0:04 | 0:20 | 0:28 | 2:06 | 11:52 | 29:11 | 3:19:56 | 4:04:29 | > 5 h |
| $Inv$ (9) | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 |
| $Inv$ (10) | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:04 | 0:00 | 0:03 | 0:06 | 0:03 |
| $U(v)$ | 0:01 | 0:09 | 0:24 | 1:49 | 3:04 | 2:05:21 | > 5 h | > 5 h | > 5 h | > 5 h | > 5 h | > 5 h |
| (b) *Hybrid-1*($\alpha$) | | | | | | | | | | | | |
| Agreement | 0:11 | 1:01 | 5:54 | 1:25:14 | >5 h | >5 h | >5 h | >5 h | >5 h | >5 h | >5 h | >5 h |
| Termination | 0:00 | 0:00 | 0:01 | 0:01 | 0:10 | 13:05 | 1:56 | 5:48 | 10:22 | 14:20 | 47:31 | > 5 h |
| $Inv$ (9) | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 |
| $Inv$ (10) | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:01 | 0:02 | 0:03 | 0:04 | 0:11 | 0:18 | 0:22 |
| $U(v)$ | 0:09 | 1:01 | 9:38 | 1:10:03 | > 5 h | > 5 h | > 5 h | > 5 h | > 5 h | > 5 h | > 5 h | > 5 h |
| (c) *CoordUniformVoting* | | | | | | | | | | | | |
| Agreement | 0:01 | 0:02 | 0:06 | 0:20 | 0:55 | 2:34 | 6:05 | 13:20 | 42:54 | 3:03:18 | > 5 h | > 5 h |
| Termination | 0:02 | 0:02 | 0:03 | 0:05 | 0:06 | 0:09 | 0:14 | 0:22 | 0:27 | 0:38 | 1:20 | 1:46 |
| $Inv$ (9) | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 |
| $Inv$ (10) | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 |
| $U(v)$ | 0:00 | 0:00 | 0:01 | 0:01 | 0:02 | 0:07 | 0:09 | 0:10 | 0:22 | 0:33 | 0:37 | 1:09 |
| (d) Simplified *CoordUniformVoting* | | | | | | | | | | | | |
| Agreement | 0:01 | 0:03 | 0:06 | 0:21 | 1:02 | 2:53 | 10:19 | 29:33 | 1:31:51 | > 5 h | > 5 h | > 5 h |
| Termination | 0:00 | 0:00 | 0:01 | 0:04 | 0:05 | 0:09 | 0:17 | 0:35 | 0:51 | 1:17 | 1:37 | 2:10 |
| $Inv$ (9) | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 |
| $Inv$ (10) | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 | 0:00 |
| $U(v)$ | 0:00 | 0:00 | 0:01 | 0:01 | 0:03 | 0:05 | 0:06 | 0:17 | 0:17 | 0:21 | 0:54 | 0:59 |

votes. In the third round, $P^{safe}(\phi)$ also guarantees that if some process decides a value, then all processes will adopt the same value. This in turn guarantees that only the same decision is possible in any later phase.

Recall that the condition for safety is represented as the formula $Safe$ in our model checking procedures (see Sect. 3.2.2). For *LastVoting* and $Hybrid - 1(\alpha)$, $Safe$ simply equals `true`, whereas for *CoordUniformVoting* we have:

$$Safe \triangleq \bigwedge_{\substack{p_1,p_2\in\Pi: p_1<p_2 \\ i:1\le i\le 3}} \bigvee_{q\in\Pi} (ho^i_{p_1,q} = \texttt{true} \wedge ho^i_{p_2,q} = \texttt{true})$$

The condition for termination shown in Table 1c specifies that all processes can hear from the same coordinator in the first round. The $Inv$ in Table 2c is derived from the fact that the variable $vote_p$ is always initialized to ? in every phase (line 4). Once a decision has been made, all processes have to have the same estimate, equal to the decision value. The $U(v)$ in Table 2c represents this observation.

#### 6.1.4 Simplified CoordUniformVoting

The Simplified *CoordUniformVoting* algorithm can be viewed as the Mostéfaoui-Raynal algorithm [31] expressed in the HO model. This algorithm requires that only a single coordinator exist, in addition to the condition that any two processes hear from at least one common process. The single coordinator ensures that all votes are uniform. Termination is intended to be satisfied in a phase where in its first round, all the processes agree on the coordinator and can hear from it. These conditions are formally represented in Table 1d. The $Inv$ and $U(v)$ in Table 2d are the same as for *CoordUniformVoting*.

### 6.2 Experiments

We conducted, up to $n = 15$, the five kinds of checks, namely agreement, termination, the base case of $Inv$ (Formula (9)), the inductive step of $Inv$ (Formula (10)), and $U(v)$. The experiments were performed on a Linux workstation with an Intel Xeon processor 2.2GHz and 4Gbyte memory. We used the Yices [14] satisfiability solver. Table 3 shows the

**Table 4** The maximum number of processes for which agreement and termination were proven

| Algorithm | Agreement | Termination |
|---|---|---|
| *LastVoting* | $n = 9$ | $n = 14$ |
| *Hybrid-1*($\alpha$) | $n = 7$ | $n = 14$ |
| *CoordUniformVoting* | $n = 13$ | $n = 15$ |
| Simplified *CoordUniformVoting* | $n = 12$ | $n = 15$ |

**Table 5** Traditional approach on *LastVoting* with different model checkers

| Model checker | Agreement | | Termination | |
|---|---|---|---|---|
| | $n$ | Time | $n$ | Time |
| NuSMV [12] | $n = 4$ | 2:47 | $n = 3$ | 0:41 |
| SPIN [21]* | $n = 3$ | 48:42 | – | |
| ALV [4] | $n = 3$ | 32:01 | – | |

* The rotating coordinator paradigm is assumed.

execution time required for these checks.[8] Notation "> 5h" indicates that the check was not completed within 5 h. The execution time increases as $n$ increases, although fluctuating in some cases. For example, the execution time for the termination check of *Hybrid-1(α)* decreases when $n$ increases from $n = 9$ to $n = 10$ (Table 3b). Such fluctuations can be explained by the fact that the Yices SMT solver uses a variety of heuristics to prune the search space and thus a larger state space does not always mean a longer execution time. This fact also explains why *CoordUniformVoting* sometimes has a smaller verification time than its simplified version.

No satisfiable solution was found in any of the checks. This means, up to $n$ for which model checking was successfully completed, that the $Inv$ and $U(v)$ presented in Table 2 satisfy Assumptions 1 and 2, respectively, and that both agreement and termination hold. Table 4 summarizes the maximum number of processes for which agreement and termination were proven. Note that agreement is proven when the four checks, namely, agreement, the base case and induction step for $Inv$, and $U(v)$ are completed and all cases are unsatisfiable, whereas proving termination involves termination check and the two checks for $Inv$.

### 6.3 Traditional approach

For comparison, we evaluated a different approach in which the whole state space of an algorithm is explored with an existing model checker. Specifically we verified *LastVoting* with three model checkers: NuSMV [12], SPIN [21], and ALV [4].

SMV [28] and SPIN are probably the two best known model checkers. NuSMV is one of the latest implementations of SMV. Although SMV and SPIN can only deal with finite state systems, the abstraction technique proposed in [36] allows one to model check the whole state space with these model checkers. In [36] we show how NuSMV can be used to model check HO model-based consensus algorithms.

In model checking with SPIN we made an extensive optimization to reduce the state space. Specifically we completely removed the information on HO sets from the state space, by specifying all possible behaviors as non-deterministic ones. The details can be found in [29]. This optimization does not work for SMV because HO sets are already compactly represented by the data structure used in SMV (specifically, binary decision diagrams).

ALV is a model checker that can analyze infinite state systems with unbounded integer variables by means of Presburger arithmetic [3]. It does not require any finite state abstraction. We made the same optimization as with SPIN to avoid explicit representation of HO sets.

Table 5 presents the maximum number of processes that each of the model checkers was able to handle without running out of memory or time (5 h) together with the time needed to model check the largest model.

Termination is only verified by NuSMV, because for SPIN and ALV, the above optimization completely abstracts away HO sets and thus the termination property cannot be formally specified. In spite of the extensive optimization, however, SPIN could not complete the verification of *LastVoting* even for $n = 3$ because of memory exhaustion. The execution time of SPIN presented in Table 5 is that required to model check a rotating coordinator-based version of the algorithm. This variant has a significantly smaller state space, because the coordinator is determined completely deterministically.

Comparing Tables 4 and 5 one can clearly see that the proposed approach scales much better than the approach using existing model checkers. This improvement can be explained by the fact that our approach can avoid explosive growth of the search space by limiting it to single phases.

[8] All the files used in the experiment are available at: http://www-ise4.ist.osaka-u.ac.jp/~t-tutiya/dc2010/

## 7 Conclusion

We proposed a verification approach for asynchronous consensus algorithms. Using the notions of invariant and univalence, we reduced agreement and termination verification to the problem of model checking only single phases of the algorithm. This unique property of the model checking problem allowed us to effectively use bounded model checking. As case studies we applied the proposed approach to four consensus algorithms and mechanically verified that they satisfy agreement and termination up to around 10 processes. Comparing the performance of the traditional model checking approach showed that the performance improvement is significant.

The cases tested in the case studies in Sect. 6 were all successful, in that the formula was proven to be unsatisfiable, meaning that the property verified indeed holds. As stated in Sects. 3.1 and 4, this is not always the case and the proof process can fail if either the algorithm and/or its associated communication-coordinator predicate are incorrect or the invariant $Inv$ and/or the approximation of the univalent state set, $U(v)$, are inaccurate.

In a follow-up paper [30] we report the results of applying the proposed approach to an algorithm that is seeded with a design fault. This artificial fault was easily identified, using the value assignment to the variables that the SMT solver outputs. When the proof process fails, the SMT solver outputs the satisfying value assignment to show how the checked formula evaluates to true. This assignment represents an instance of the one-phase executions that falsify the given property, and thus one can easily analyze the problematic behavior of the system.

On the other hand, we have little experience about the cases where the proof process fails because of the inaccuracy of $Inv$ or $U(v)$. We believe that in such a case the one-phase execution that the SMT solver outputs can also serve as a vital clue to finding the cause of the failure. Refinement techniques for state approximations, such as *invariant strengthening* [32], may be useful in that case. We leave the application of such techniques to our context for future work.

There are several other future research directions. For example, the verification of algorithms that implement rounds of the HO model deserves further study. An example of related work in this line is [35], where we model check such an algorithm. Further studies are also needed to improve the applicability of the proposed verification approach. For example, our approach cannot be directly used for termination verification if the algorithm requires more than one synchronous phase to have all processes decide. A simple way to deal with such algorithms could be to model check two or more consecutive phases, instead of a single phase. However, this extension cannot be used to verify if termination is satisfied when synchronous phases occur sporadically. Extensions to handle a larger class of algorithms, including these, are of technical interest.

**Acknowledgments** The authors wish to thank Takahiro Minamikawa for providing us the SPIN model of *LastVoting* and the anonymous referees for their valuable comments. Tatsuhiro Tsuchiya was supported in part by the Grant-in-Aid for Young Scientists (B) from the Japanese Ministry of Education, Science, Sports and Culture under grant number 20700026. André Schiper was supported in part by the Swiss National Science Foundation under grant number 200021-111701 and by Hasler Foundation under grant number 2070. This paper is a revised and expanded version of a paper presented at the 22nd International Symposium on Distributed Computing (DISC 2008), September 2008 [37].

## References

1. Ben-Or, M.: Another advantage of free choice: Completely asynchronous agreement protocols (extended abstract). In: Proc. Second ACM Symp. on Principles of Distributed Computing (PODC-2), pp. 27–30 (1983)
2. Bokor, P., Pataricza, A., Serafini, M., Suri, N.: Model checking of distributed dependable protocols using semantic property preserving abstractions. Tech. Rep. TR-TUD-DEEDS-09-01-2007, TU Darmstadt (2007)
3. Bultan, T., Gerber, R., Pugh, W.: Model-checking concurrent systems with unbounded integer variables: Symbolic representations, approximations, and experimental results. ACM Trans. Program. Lang. Syst. **21**(4), 747–789 (1999). doi:10.1145/325478.325480
4. Bultan, T., Yavuz-Kahveci, T.: Action language verifier. In: Proc. 16th IEEE Int'l Conf. on Automated Software Engineering (ASE '01), pp. 382–386. San Diego, CA, USA (2001)
5. Chandra, T.D., Toueg, S.: Unreliable failure detectors for reliable distributed systems. J. ACM **43**(2), 225–267 (1996). doi:10.1145/226643.226647
6. Chaouch-Saad, M., Charron-Bost, B., Merz, S. : A reduction theorem for the verification of round-based distributed algorithms. In: Bournez, O., Potapov, I. (eds.) Reachability Problems '09, Lecture Notes in Computer Science, vol. 5797, pp. 93–106. Springer, Palaiseau (2009)
7. Charron-Bost, B., Merz, S.: Formal verification of a Consensus algorithm in the Heard-Of model. Int. J. Softw. Inform. **3**(2–3), 273–303 (2009)
8. Charron-Bost, B., Schiper, A.: Improving Fast Paxos: Being optimistic with no overhead. In: Proc. of 12th Pacific Rim International Symposium on Dependable Computing (PRDC'06), pp. 287–295. IEEE CS Press, Riverside, CA, USA (2006)
9. Charron-Bost, B., Schiper, A.: Harmful dogmas in fault tolerant distributed computing. SIGACT News **38**(1), 53–61 (2007)
10. Charron-Bost, B., Schiper, A.: The heard-of model: Computing in distributed systems with Benign failures. Distrib. Comput. **22**(1), 49–71 (2009)
11. Cheung, L.: Randomized wait-free consensus using an atomicity assumption. In: Proc. 9th International Conference on Principles of Distributed Systems (OPODIS'05), LNCS, vol. 3974, pp. 47–60. Springer, Pisa, Italy (2005)
12. Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV 2: An open-source tool for symbolic model checking. In: Proc. of 14th Conf. on Computer Aided Verification (CAV 2002), LNCS, vol. 2404. Springer, Copenhagen, Denmark (2002)

13. Clarke, E., Biere, A., Raimi, R., Zhu, Y.: Bounded model checking using satisfiability solving. Formal Methods Syst. Des. **19**(1), 7–34 (2001). doi:10.1023/A:1011276507260
14. Dutertre, B., de Moura, L.M.: A fast linear-arithmetic solver for DPLL(T). In: Proc. of 18th Conf. on Computer Aided Verification (CAV 2006), LNCS, vol. 4144, pp. 81–94. Springer, Seattle, USA (2006)
15. Dwork, C., Lynch, N., Stockmeyer, L.: Consensus in the presence of partial synchrony. J. ACM **35**(2), 288–323 (1988). doi:10.1145/42282.42283
16. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. J. ACM **32**(2), 374–382 (1985). doi:10.1145/3149.214121
17. Gafni, E.: Round-by-round fault detectors: Unifying synchrony and asynchrony. In: Proc. 17th ACM Symp. on Principles of Distributed Computing (PODC-17), pp. 143–152. ACM Press, New York, NY, USA (1998). doi:10.1145/277697.277724
18. Gafni, E., Lamport, L.: Disk Paxos. Distrib. Comput. **16**(1), 1–20 (2003). doi:10.1007/s00446-002-0070-8
19. Guerraoui, R., Schiper, A.: The generic consensus service. IEEE Trans. Softw. Eng. **27**(1), 29–41 (2001). doi:10.1109/32.895986
20. Hendriks, M.: Model checking the time to reach agreement. In: Pettersson, P., Yi, W. (eds.) 3rd International Conference on the Formal Modeling and Analysis of Timed Systems (FORMATS'05), LNCS, vol. 3829, pp. 98–111. Springer (2005)
21. Holzmann, G.J.: The model checker SPIN. IEEE Trans. Softw. Eng. **23**(5), 279–295 (1997). doi:10.1109/32.588521
22. Hutle, M., Schiper, A.: Communication predicates: A high-level abstraction for coping with transient and dynamic faults. In: Proc. Int'l Conf. on Dependable Systems and Network (DSN 2007). IEEE CS Press, Edinburgh, UK (2007). Full version in technical report LSR-REPORT-2006-006, EPFL, pp. 92–101 (2006)
23. Kwiatkowska, M.Z., Norman, G.: Verifying randomized Byzantine agreement. In: Proc. 22nd IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems (FORTE '02), LNCS 2529, pp. 194–209. Springer, Houston, USA (2002)
24. Kwiatkowska, M.Z., Norman, G., Segala, R.: Automated verification of a randomized distributed consensus protocol using Cadence SMV and PRISM. In: Proc. of 13th Conf. on Computer Aided Verification (CAV 2001), LNCS, vol. 2102, pp. 194–206. Springer, Paris, France (2001)
25. Lamport, L.: The part-time parliament. ACM Trans. Comput. Syst. **16**(2), 133–169 (1998). doi:10.1145/279227.279229
26. Lamport, L.: Personal Communication (2006)
27. Lamport, L.: Fast Paxos. Distrib. Comput. **19**(2), 79–103 (2006)
28. McMillan,K.L.: Symbolic Model Checking. Kluwer Academic Publishers, Norwell, MA (1993)
29. Minamikawa, T., Tsuchiya, T., Kikuno, T.: Language and tool support for model checking of fault-tolerant distributed algorithms. In: Proc. of 14th Pacific Rim International Symposium on Dependable Computing (PRDC'08), pp. 40–47. IEEE CS Press, Taipei, Taiwan (2008)
30. Minamikawa, T., Tsuchiya, T., Kikuno, T.: Towards automated verification of distributed consensus protocols. In: Proc. of 16th Asia-Pacific Software Engineering Conference (APSEC 2009), pp. 499–506. IEEE CS Press, Penang, Malaysia (2009)
31. Mostéfaoui, A., Raynal, M.: Solving consensus using Chandra-Toueg's unreliable failure detectors: A general quorum-based approach. In: Proceedings of the 13th International Symposium on Distributed Computing, pp. 49–63. Springer, London, UK (1999)
32. de Moura, L., Rueß, H., Sorea, M.: Bounded model checking and induction: From refutation to verification. In: Proc. of 15th Conf. on Computer Aided Verification (CAV 2002), LNCS, vol. 2725, pp. 14–26 (2003)
33. Schiper, A., Toueg, S.: From set membership to group membership: A separation of concerns. IEEE Trans. Dependable Secur. Comput. (TDSC) **3**(1), 2–12 (2006)
34. Sheeran, M., Singh, S., Stålmarck, G.: Checking safety properties using induction and a sat-solver. In: Proceedings of the Third International Conference on Formal Methods in Computer-Aided Design (FMCAD '00), LNCS, vol. 1954, pp. 108–125. Springer, London, UK (2000)
35. Tsuchiya, T., Schiper, A.: An automatic real-time analysis of the time to reach consensus. In: Proc. of 13th Pacific Rim International Symposium on Dependable Computing (PRDC 2007), pp. 53–60 (2007)
36. Tsuchiya, T., Schiper, A.: Model checking of consensus algorithms. In: Proc. 26th Symp. on Reliable Distributed Systems (SRDS), pp. 137–148. Beijing, China (2007)
37. Tsuchiya, T., Schiper, A.: Using bounded model checking to verify consensus algorithms. In: 22nd International Symposium on Distributed Computing (DISC 2008), LNCS, vol. 5218, pp. 466–480. Springer (2008)
38. Zieliński, P.: Automatic verification and discovery of Byzantine consensus protocols. In: Proc. Int'l Conf. on Dependable Systems and Network (DSN 2007), pp. 72–81. IEEE CS Press, Edinburgh, UK (2007)