View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Infoscience - École polytechnique fédérale de Lausanne

Using Bounded Model Checking to Verify Consensus Algorithms

Tatsuhiro Tsuchiya*
Osaka University
1-5 Yamadaoka, Suita, 565-0871 Japan
t-tutiya@ist.osaka-u.ac.jp

André Schiper†
École Polytechnique Fédérale de Lausanne (EPFL)
1015 Lausanne, Switzerland
andre.schiper@epfl.ch

Abstract

This paper presents an approach to automatic verification of asynchronous round-based consensus algorithms. We use model checking, a widely practiced verification method; but its application to asynchronous distributed algorithms is difficult because the state space of these algorithms is often infinite. 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 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 some consensus algorithms up to around 10 processes.

1. Introduction

Model checking, a method for formally verifying state transition systems, has now become popular, because it allows the fully automatic analysis of designs. This paper presents an approach to model checking of *asynchronous consensus algorithms*. *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 [3]. Other services, such as view synchrony and membership, can also be constructed using consensus [16, 25]. Because of the importance, much research has been being devoted to developing new algorithms for this problem.

Model checking of asynchronous consensus algorithms is difficult, because these algorithms usually induce an infinite state space, making model checking infeasible. Sources of infinite state spaces include unbounded round numbers and unbounded message channels. In our previous work [26], we succeeded in model checking several asynchronous consensus algorithms by adopting a round-based computation model, called the *Heard-Of (HO) model* [6], and by devising a finite abstraction of unbounded round numbers. The scalability, however, still needs to be addressed, because the system size that can be model checked is rather small, typically three or four processes.

In this paper we present a different approach for the verification of asynchronous consensus algorithms. Our approach divides the verification problem into several problems that can be solved by model checking. Importantly, these model checking problems can be solved by only analyzing single phases of the execution of the consensus algorithm. Because of this property of the problems, *bonded model checking [9]* can be very effectively used. Through case studies, we show that the time and memory space required for verification can be significantly reduced, resulting in increasing the size of systems that can be verified.

Related Work Several attempts have been reported to model check consensus algorithms. The TLA specifications of some *Paxos* algorithms [15, 23] were debugged with the aid of TLC, the TLA model checker. The models that were model checked consisted of two or three processes and a small number of rounds [22]. In [27], an approach to automatic discovery of consensus algorithms was 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 [17], a synchronous consensus algorithm was model checked for three processes. Studies on model checking of shared memory-based randomized consensus algorithms can be found in [7, 20]. Model checking of Byzantine agreement protocols was studied in [1] and [19].

*Supported in part by the Grant-in-Aid from MEXT of Japan (no. 20700026). This work began when the first author was visiting EPFL with support from Scientist Exchange Program between JSPS and SNSF.

†Research funded by the Swiss National Science Foundation under grant number 200021-111701 and Hasler Foundation under grant number 2070.

A synchronous system model was assumed in [1]. In [19] an asynchronous algorithm was verified but manual proof was used with combination of model checking.

The verification approach proposed in this paper can be viewed as a variation of *k-induction*. Although in different contexts, combining bounded model checking and k-induction was studied in, for example, [10].

Roadmap This paper is structured as follows: Section 2 describes the HO model and the consensus problem. Section 3 and Section 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. Consensus in the Heard-Of Model

The HO model [6] is a communication-closed round model that generalizes the asynchronous round model by Dwork et al. [12] with some features of [14] and [24]. Let $\Pi = \{1, 2, \cdots, n\}$ be the set of processes. An algorithm proceeds in phases, each of which consists of $k\ (\geq 1)$ rounds.[1] An algorithm comprises, for each process $p$and each round r, a sending function S_p^r and a transition functionT_p^r. In each round r, every process p sends messages according to $S_p^r(s_p)$, where s_p is the state of p. Then, p makes a state transition according to $T_p^r(Msg, s_p)$, where Msg is the collection of all messages that p has received in round r.

In the HO model both synchrony degree and faults are represented in the form of *transmission faults*. We denote by $HO(p, r)$ $(\subseteq \Pi)$ 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. A transmission fault is a situation where $q \notin HO(p, r)$ while q sent (or was supposed to send) a message to p in round r. Transmission faults can occur if messages missed a round due to the asynchrony of communication and processing, or if a process or a link is faulty.

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. In the HO model, consensus is specified by the following three conditions:

Integrity Any decision value is the proposed value of some process.

Agreement No two processes decide differently.

Termination All processes eventually decide.

[1] In [3] and [21], a round is decomposed in phases. "Round" and "phase" are swapped here to use the classical terminology [12].

Algorithm 1 The *LastVoting* (*Paxos*) algorithm [6]

1: **Initialization:**
2: $x_p \in Val$, initially v_p {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 {$\mathbb{N}$ *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 θ from $\langle \nu\ ,\ \theta \rangle$ received
13: $vote_p :=$ one ν 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`

For most consensus algorithms, integrity is trivially satisfied; thus we limit our discussion to the verification of agreement and termination. Note that in the HO model the termination property requires all processes to decide. Discussion of the reason for this specification can be found in [5, 6].

The above computation model can naturally be extended to represent coordinator-based algorithms. Let $Coord(p, \phi)$ denote the coordinator process of process p in phase ϕ. 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 ϕ is the phase that round r belongs to. The *LastVoting* algorithm (Algorithm 1) is an example of a coordinator-based consensus algorithm [6]. This algorithm can be viewed as an HO model-version of *Paxos* [21]. *LastVoting* is also close to the $\diamond\mathcal{S}$ consensus algorithm [3].

Since there is no deterministic consensus algorithm in a pure asynchronous system, some synchrony condition must be assumed to solve the problem. In the HO model such a condition is represented as a predicate over the collections of HO sets $(HO(p,r))_{p\in\Pi,r>0}$ and of coordinators $(Coord(p,\phi))_{p\in\Pi,\phi>0}$. For example, the following predicate specifies a sufficient condition for the *LastVoting* algorithm to solve consensus:

$$\begin{array}{l}\exists\phi_0>0, \exists co\in\Pi, \forall p\in\Pi: \\ \quad (co = Coord(p,\phi_0)) \\ \quad \wedge\ (|HO(co,4\phi_0-3)|>n/2) \\ \quad \wedge\ (|HO(co,4\phi_0-1)|>n/2) \\ \quad \wedge\ (co\in HO(p,4\phi_0-2)) \wedge (co\in HO(p,4\phi_0))\end{array} \quad (1)$$

In words, phase ϕ_0 is a synchronous phase 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 ϕ_0 occurs, then all processes will make a decision at the end of this phase. This condition is required only for termination. Agreement can never be violated no matter how bad the HO set is. For simplicity, in the paper we limit our discussion to verification of such algorithms — algorithms that are always safe, even in completely asynchronous runs.

3. Verification of Agreement

In this section, we present our approach to the verification of agreement. The verification of termination is discussed in Section 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,

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

where $\mathcal{S}_p$ is a set of states of a process p and $\mathbb{N}^+$ is a set of positive integers. Given a configuration $c = (s_1, \cdots, s_n, \phi) \in \mathcal{C}$, we denote by $\phi(c)$ the phase number ϕ of c. It should be noted that the state of a process is a value assignment to the variables of the process. Hence

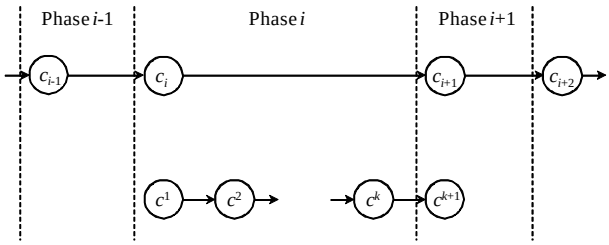

Figure 1. Transitions of configurations at the phase level (top) and at the round level (bottom).

any set of configurations can be represented by a predicate over the process variables of all processes and ϕ; that is, the predicate represents a 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: $(c,d,c') \in R$ iff the system can transit from the configuration c at the beginning of phase $\phi(c)$ to the next configuration c' at the beginning of the next phase $\phi(c')$ while deciding the values in d during phase $\phi(c)$. By definition $\phi(c)+1 = \phi(c')$ if $(c,d,c') \in R$.

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

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

The key feature of our verification approach is that it can determine whether Formula (2) holds or not without exploring all runs. In doing this, the notion of *univalence* plays a crucial role. A configuration is said to be *univalent* if there is only one value that can be decided from this configuration [13]. 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 ***(Univalence)*** 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 $\forall j \geq i : (c_j, d_j, c_{j+1}) \in R$.

By definition, any configuration next to a v-valent configuration is also v-valent. That is, we have:

Lemma 1 If c is a v-valent configuration, then any c' such that $(c, d, c') \in R$ is v-valent.

In the proposed verification approach, we assume that some property, represented by $U(v)$ and Inv, holds on the algorithm under verification. $U(v)$ is shown below. The assumption for Inv will be described later. Indeed, we will have to validate $U(v)$ and Inv.

Assumption 1 ($U(v)$): For any $v \in Val$, $U(v)$ is a set of configurations such that if $c \in U(v) \cap Reachable$, then c is v-valent. In other words, any configuration in $U(v)$ is either (i) reachable and v-valent or (ii) unreachable.

Example 1 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:

$$U(v) := \\ \exists Q \subseteq \Pi : \left\{ \begin{array}{l} |Q| > n/2 \wedge \\ \forall p \in Q : (x_p = v \wedge \forall q \in \Pi \backslash Q : ts_p > ts_q) \end{array} \right.$$

The following theorem states that 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 in $U(v)$.

Theorem 1 Agreement holds if:

$$\begin{array}{l} \forall c \in Reachable : \forall (c, d, c') \in R : \\ \quad d = \emptyset \vee \exists v \in Val : (d = \{v\} \wedge c' \in U(v)) \end{array} \tag{3}$$

Proof We show that for any $c_1 d_1 c_2 d_2 \cdots \in Run$, (3) implies that for any $l \geq 1$, either (i) $\bigcup_{0<i\leq l} d_i = \emptyset$ or (ii) for some $v \in Val$, $\bigcup_{0<i\leq l} d_i = \{v\}$ and $c_l \in U(v)$, meaning that (2) holds. The proof is by induction on l.

Base case: (3) implies that either $d_1 = \emptyset$ or $d_1 = \{v\} \wedge c_2 \in U(v)$ for some $v \in Val$.

Inductive step: Suppose that the above (i) or (ii) holds for some $l \geq 1$. If (i) holds for l, then $d_{l+1} = \emptyset$ or $d_{l+1} = \{v\} \wedge c_{l+1} \in U(v)$ for some $v \in Val$. Hence (i) or (ii) holds for $l + 1$. If (ii) holds for l, then $d_{l+1} = \emptyset$ or $d_{l+1} = \{v\}$ since $c_l \in U(v)$. Also $c_{l+1} \in U(v)$ by Lemma 1. Thus (ii) also holds for $l + 1$. □

It should be noted that Formula (3) only refers to individual phase-level transitions from $Reachable$, rather than to runs. This property is critical for reducing the verification problem to a model checking problem of single phases. However, directly checking this formula would do little good, because roughly speaking, obtaining $Reachable$ is as hard as examining all runs.

The key here is that $Reachable$ can be substituted by its *over-approximation*. An over-approximation of the set of reachable states is usually referred to as an *invariant*.

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

We assume that an invariant Inv is available:

Assumption 2 (Inv): A set Inv of configurations is an invariant; that is, $Reachable \subseteq Inv$.

Example 2 It is easy to see that for *LastVoting*, the following predicate is always true at the beginning of every phase ϕ:

$$Inv := \forall p \in \Pi : \left\{ \begin{array}{l} commit_p = \texttt{false} \\ \wedge\, ready_p = \texttt{false} \wedge\ ts_p < \phi \end{array} \right.$$

Theorem 2 Agreement holds if:

$$\begin{array}{l} \forall c \in Inv : \forall (c, d, c') \in R : \\ \quad d = \emptyset \vee \exists v \in Val : (d = \{v\} \wedge c' \in U(v)) \end{array} \tag{4}$$

Proof Because $Reachable \subseteq Inv$, (4) implies (3). By Theorem 1, (3) implies agreement. □

This theorem leads directly to the following verification steps:

Step A1: Check if (4) holds or not.

Step A2: If (4) holds, then agreement holds. Otherwise, further analysis is needed because in this case (i) the consensus algorithm is incorrect or (ii) $U(v)$ or Inv are too small or too large, respectively.

3.2. Model Checking of Single Phases

This section shows how *model checking* can be used to determine if Formula (4) 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 Section 2).

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

- c^i ($1 \leq i \leq k$) is the configuration at the beginning of the i-th round of the phase, i.e., round $k * (\Phi - 1) + i$, while c^{k+1} corresponds to the first round of the next phase (see Figure 1). Hence $\Phi = \phi(c^1) = \phi(c^2) = \cdots = \phi(c^k)$ and $\Phi + 1 = \phi(c^{k+1})$.
- $ho^i = (HO(1, k(\Phi-1)+i)), \cdots, HO(n, k(\Phi-1)+i)))$ is a collection of n HO sets – one per process – in the i-th round.
- $dv^i = (dv^i_1, \cdots, dv^i_n)$, where $dv^i_p \in Val \cup \{?\}$ for any $p \in \Pi$, is the collection of values decided by each process in the i-th round. If a process p does not decide in the round, then $dv^i_p = ?$.

- $Coord = (Coord(1,\Phi), \cdots, Coord(n,\Phi))$ is a collection of n coordinators – one per process – in phase Φ.

We call such a tuple a *one-phase execution* iff it is consistent with the given consensus algorithm.[2] By definition, $(c,d,c') \in R$ iff there is a one-phase execution $(c^1 ho^1 dv^1 \cdots c^k ho^k dv^k c^{k+1}, Coord)$ such that $c = c^1$, $c' = c^{k+1}$, and $d = \big(\bigcup_{p\in\Pi, 1\le i\le k}\{dv_p^i\}\big)\backslash\{?\}$.

We denote by X the set of all one-phase executions that need to be examined to check (4): Since we are concerned with the phase that starts with a configuration in Inv, a one-phase execution $(c^1 ho^1 dv^1 \cdots c^k ho^k dv^k c^{k+1}, Coord)$ is in X iff $c^1 \in Inv$.

Our model checking problem is described as follows: Given (i) an algorithm to be verified, (ii) $U(v)$, and (iii) Inv, determine if the following condition holds for all $(c^1 ho^1 dv^1 \cdots c^k ho^k dv^k c^{k+1}, Coord) \in X$:

$$d = \emptyset \vee \exists v \in Val : (d = \{v\} \wedge c^{k+1} \in U(v)) \quad (5)$$

where $d = \big(\bigcup_{p\in\Pi, 1\le i\le k}\{dv_p^i\}\big)\backslash\{?\}$.

Clearly (4) holds iff (5) holds for all one-phase executions in X.

This model checking problem is unique in that it only concerns exactly k consecutive transitions. Because of this, *bounded model checking* [9] can be most effectively used to solve this problem. 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 set of constraints in some logic. The satisfiability problem is the problem of determining whether or not values can be assigned to variables such that all constraints evaluate to true. Bounded model checking can often be used only for finding defects near the initial states; but in our case it can allow complete verification because it is already known that the depth of the search space is exactly k.

Most of the work on bounded model checking is based on propositional logic. In contrast, in our case the constraints are boolean combinations of linear (in)equalities and boolean expressions with integer and boolean variables. The variables involved in the constraints are:

- Variables that represent the values of the process variables at each configuration c^i $(1 \le i \le k+1)$. If the process variable is *boolean*, so is the corresponding representing variable; otherwise the representing variable is *integer*.
- An integer variable Φ, which represents the phase number; that is, $\Phi = \phi(c^1) = \phi(c^2) = \cdots = \phi(c^k)$.
- Boolean variables $ho_{p,q}^i (p,q \in \Pi, 1 \le i \le k)$, which represent whether p hears of q in the i-th round. That is, $q \in HO(p, k(\Phi-1)+i)$ iff $ho_{p,q}^i = \texttt{true}$.
- Integer variables $dv_p^i (p,q \in \Pi, 1 \le i \le k)$, which represent the value that is decided by p in the i-th round.
- Integer variables $Coord_p (p \in \Pi)$, which represent the coordinator of p in phase Φ.

It should be noted that any one-phase execution is uniquely represented as a value assignment to these variables.

Example 3 For the *LastVoting* algorithm, the variables involved in model checking are: $x_p^i, vote_p^i, ts_p^i$ (integer) and $commit_p^i$, $ready_p^i$ (boolean) for $p \in \Pi, 1 \le i \le 5$; Φ (integer); $ho_{p,q}^i$ (boolean) for $p,q \in \Pi, 1 \le i \le 4$; dv_p^i (integer) for $p \in \Pi, 1 \le i \le 4$; $Coord_p$ (integer) for $p \in \Pi$.

In order to check (5), we proceed as follows. We consider two set of constraints on the above variables:

- $\mathcal{X}$, which represents all one-phase executions in X. That is, $\mathcal{X}$ represents exactly the value assignments to variables corresponding to a one-phase execution in X.
- $\overline{\mathcal{U}}$, which represents the value assignments to variables that correspond to a one-phase execution in X that does *not* meet (5).

$\mathcal{X}$ is derived from the consensus algorithm and Inv, while $\overline{\mathcal{U}}$ is derived from $U(v)$. Note that $\mathcal{X}$ and $\overline{\mathcal{U}}$ can only be simultaneously satisfied by a value assignment corresponding to a one-phase execution that (i) belongs to X *and* (ii) for which (5) does not hold. Therefore every one-phase execution in X meets (5) iff $\mathcal{X} \cup \overline{\mathcal{U}}$ is unsatisfiable.

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

Step B1: Check the satisfiability of:

$$\mathcal{X} \cup \overline{\mathcal{U}}$$

This check can be done by an off-the-shelf *Satisfiability Modulo Theories* (SMT) solver, such as Yices [11].

Step B2: If no satisfying value assignment exists, then every one-phase execution in X satisfies (5), meaning that (4) holds. As a result, it is guaranteed that agreement holds. On the other hand if there is a satisfying assignment, further analysis is needed to obtain a conclusive answer (see Step A2).

We now show how to construct $\mathcal{X}$ and $\overline{\mathcal{U}}$.

Constraints $\mathcal{X}$: $\mathcal{X}$ is composed as follows:

$$\mathcal{X} := Dom \cup \mathcal{T}^1 \cup \mathcal{T}^2 \cup \cdots \cup \mathcal{T}^k \cup \mathcal{I},$$

where Dom, $\mathcal{T}^i$ and $\mathcal{I}$ are as follows:

[2] 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.

Dom**:** Dom restricts the domains of the variables. We represent the set Val of possible proposed values by the set of all positive integers and ? by zero. For the *LastVoting* algorithm, Dom consists of the following constraints:

- $\forall p \in \Pi, \forall i, 1 \leq i \leq 5$: $x_p^i > 0 \wedge vote_p^i \geq 0 \wedge ts_p^i \geq 0$
- $\Phi > 0$
- $\forall p \in \Pi, \forall i, 1 \leq i \leq 4$: $dv_p^i \geq 0$
- $\forall p \in \Pi : 1 \leq Coord_p \leq n$

$\mathcal{T}^i$**:** $\mathcal{T}^i$ represents the i-th round of the algorithm; $\mathcal{T}^i$ is satisfiable iff the system's states – represented by the variable values at the beginning of the ith and $i{+}1$-th rounds – are consistent with the algorithm under verification. For example:

$$
\begin{array}{l}
\mathcal{T}^3 := \\
\quad \forall p \in \Pi : \\
\qquad x_p^3 = x_p^4 \\
\quad \wedge \quad vote_p^3 = vote_p^4 \\
\quad \wedge \quad ts_p^3 = ts_p^4 \\
\quad \wedge \quad commit_p^3 = commit_p^4 \\
\quad \wedge \quad ite(Coord_p = p \wedge \\
\qquad \bigvee_{Q \in Maj} \bigwedge_{q \in Q} (Coord_q = p \wedge ts_q^3 = \Phi \wedge ho_{p,q}^3 = \texttt{true}), \\
\qquad ready_p^4 = \texttt{true}, ready_p^3 = ready_p^4) \\
\quad \wedge \quad dv_p^3 = 0
\end{array}
$$

where $Maj \equiv \{Q \mid Q \subseteq \Pi, |Q| > n/2\}$. The first four terms express that process variables x_p, $vote_p$, ts_p and $commit_p$ do not change in the third round. The ite term[3] represents how variable $ready_p$ is updated: It specifies that $ready_p$ will be updated to `true` if p considers itself the coordinator and receives messages from a majority of processes who consider the coordinator to be p (see lines 24–27 of Algorithm 1). The last term expresses that no decision is made in the third round.

$\mathcal{I}$**:** $\mathcal{I}$ enforces that $c^1 \in Inv$. For example, consider Inv shown in Example 2 that refers to phase Φ. In this case we have:

$$
\mathcal{I} := \forall p \in \Pi : \left\{ \begin{array}{l} commit_p^1 = \texttt{false} \\ \wedge\, ready_p^1 = \texttt{false} \wedge\ ts_p^1 < \Phi \end{array} \right.
$$

Constraints $\overline{\mathcal{U}}$**:** The negation of (5) is:

$$d \neq \emptyset \wedge \forall v \in Val : \neg(d = \{v\} \wedge c^{k+1} \in U(v))$$

where $d = \left(\bigcup_{p \in \Pi, 1 \leq i \leq k} \{dv_p^i\}\right) \backslash \{?\}$. If $v \notin d$, then $d \neq \{v\}$, which allows us to replace Val with d:

$$d \neq \emptyset \wedge \forall v \in d : \neg(d = \{v\} \wedge c^{k+1} \in U(v))$$

[3] $ite(a, b, c) = b$ if $a = \texttt{true}$; $ite(a, b, c) = c$, otherwise.

$\overline{\mathcal{U}}$ consists of constraints that represent the above formula. For example, consider $U(v)$ given in Example 1. Then $\overline{\mathcal{U}}$ consists of the conjunction of the following constraints:

- $\bigvee_{p \in \Pi, 1 \leq i \leq k} dv_p^i \neq 0$
- $\forall p \in \Pi, \forall i, 1 \leq i \leq k$:

$$
\begin{array}{l}
(dv_p^i \neq 0) \longrightarrow \\
\neg \left(\begin{array}{l} \bigwedge_{q \in \Pi, 1 \leq j \leq k} (dv_q^j = dv_p^i \vee dv_q^j = 0) \\ \wedge \bigvee_{Q \in Maj} \bigwedge_{q \in Q} \left(\begin{array}{l} x_q^{k+1} = dv_p^i \\ \wedge \bigwedge_{q' \in \Pi \backslash Q} ts_q^{k+1} > ts_{q'}^{k+1} \end{array} \right) \end{array} \right)
\end{array}
$$

Basically, the second constraint means that some process has decided ($dv_p^i \neq 0$) while the configuration is not univalent ($\neg(...)$).

4. Verification of Termination

As stated in Section 2, in the context of the HO model, the condition for termination is specified by a predicate over the collections of HO sets and coordinators.

In this section we consider a condition of the form:

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

where $P^{sync}(\phi)$ is a predicate over the HO sets and the coordinators in phase ϕ such that:

- $P^{sync}(\phi)$ is invariant under phase changes; that is, for any $\phi, \phi' > 0$, we have $P^{sync}(\phi) = P^{sync}(\phi')$ if the HO sets and the coordinators in phases ϕ and ϕ' are identical. Because of this property we henceforth denote $P^{sync}(\phi)$ as P^{sync}.
- P^{sync} is not the constant `false`.

For example, Condition (1) in Section 2 is of this form. The verification method described here determines whether the given algorithm always terminates in a phase where P^{sync} holds.

Let $R^{sync} \subseteq \mathcal{C} \times 2^{\Pi} \times \mathcal{C}$ be a ternary relation such that $(c, \pi, c') \in R^{sync}$ iff whenever phase $\phi(c)$ meets P^{sync}, a one-phase execution from c to c' is possible in which π is the set of processes that decide. Hence termination is satisfied if:[4]

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

Theorem 3 Termination holds if:

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

[4] Note that we exclude the exceptional case where no $(c, \pi, c') \in R^{sync}$ exists for some c, because P^{sync} is not the constant `false`.

Proof Because $Reachable \subseteq Inv$ (Assumption 2), (7) implies (6). □

Whether (7) holds or not can be determined using bounded model checking, as was done for agreement verification: Let X^{sync} be the set of all one-phase executions that start with a configuration in Inv, and can occur if P^{sync} holds for the phase. That is, $(c^1 ho^1 dv^1 \cdots c^k ho^k dv^k c^{k+1}, Coord) \in X^{sync}$ iff:

- $c^1 \in Inv$ (hence $X^{sync} \subseteq X$); and
- P^{sync} holds for the HO sets, $ho^1, ho^2, \cdots, ho^k$, and the coordinators, $Coord$.

Hence $(c, \pi, c') \in R^{sync}$ iff there is a one-phase execution $(c^1 ho^1 dv^1 \cdots c^k ho^k dv^k c^{k+1}, Coord) \in X^{sync}$ such that $c = c^1$, $c' = c^{k+1}$, and $\pi = \{p \mid \exists i, 1 \leq i \leq k : dv_p^i \neq ?\}$.

The problem we want to solve is to determine if the following condition (8) holds for all $(c^1 ho^1 dv^1 \cdots c^k ho^k dv^k c^{k+1}, Coord) \in X^{sync}$:

$$\forall p \in \Pi, \exists i, 1 \leq i \leq k : dv_p^i \neq ? \tag{8}$$

In words, (8) states that every process decides in some round of a phase where P^{sync} holds. By definition, (7) holds iff (8) holds for all executions in X^{sync}.

The problem of deciding whether all one-phase executions in X^{sync} satisfy (8) is reduced to the satisfiability problem, as was done in Section 3.2. The constraints to be checked are:

$$\mathcal{X} \cup Sync \cup \overline{Term}$$

where:

- $Sync$ represents P^{sync}. $Sync$ consists of constraints over ho_p^i and $Coord_p$ ($p \in \Pi, 1 \leq i \leq k$) and is satisfied iff P^{sync} holds for the HO sets and the coordinators represented by ho_p^i and $Coord_p$. As a result, $\mathcal{X} \cup Sync$ represents X^{sync}.
- $\overline{Term}$ is satisfied by a value assignment corresponding to a one-phase execution iff it does *not* meet (8); that is, some process exists that does not decide in the execution. $\overline{Term}$ is composed of only a single constraint:

$$\overline{Term} := \bigvee_{p \in \Pi} \bigwedge_{1 \leq i \leq k} dv_p^i = 0$$

The constraints $\mathcal{X} \cup Sync \cup \overline{Term}$ can be simultaneously satisfied by, and only by, a value assignment corresponding to a one-phase execution that (i) is in X^{sync} *and* (ii) for which (8) does not hold. Therefore every one-phase execution in X^{sync} satisfies (8) iff no satisfying assignment exists. As a result, termination can be verified as follows:

Step C1: Check the satisfiability of $\mathcal{X} \cup Sync \cup \overline{Term}$.

Step C2: If no satisfying value assignment exists, then every one-phase execution in X^{sync} satisfies (8), meaning that (4) holds. It is therefore guaranteed that termination holds. On the other hand, if the constraints are satisfiable, then it means that (i) the algorithm is incorrect or (ii) Inv is too large. In this case further analysis is required to obtain a conclusive answer.

Example 4 Consider condition (1) for termination of *LastVoting*. We have:

$$Sync := \bigvee_{p \in \Pi} \left(\begin{array}{ll} & p = Coord_1 = \cdots = Coord_n \\ \wedge & \bigvee_{Q \in Maj} \bigwedge_{q \in Q} ho_{p,q}^1 = \texttt{true} \\ \wedge & \bigvee_{Q \in Maj} \bigwedge_{q \in Q} ho_{p,q}^3 = \texttt{true} \\ \wedge & \bigwedge_{q \in \Pi} ho_{q,p}^2 = \texttt{true} \\ \wedge & \bigwedge_{q \in \Pi} ho_{q,p}^4 = \texttt{true} \end{array} \right)$$

5. Validating Inv and $U(v)$

So far we have assumed that $U(v)$ (see Example 1) and Inv (see Example 2) satisfy Assumption 1, respectively Assumption 2 (see Section 3.1). Here we present automatic procedures that can check that $U(v)$ and Inv 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 remove Assumption 2; that is, it is not known whether or not Inv is an invariant.

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

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

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

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). □

Test of (9) Testing (9) can be done by searching a configuration that is in $Init$ but not in Inv. This can be conducted by checking the satisfiability of the following constraints on a configuration c^1:

$$Dom \cup INIT \cup \overline{\mathcal{I}}$$

where:

- $INIT$ specifies that $c^1 \in Init$.
- $\overline{\mathcal{I}}$ specifies that $c^1 \notin Inv$. $\overline{\mathcal{I}}$ is the negation of $\mathcal{I}$, see Section 3.2.

Condition (9) holds iff no satisfying assignment exists.

Test of (10) Testing whether (10) holds or not can be done by determining if for all $(c^1 ho^1 dv^1 \cdots c^k ho^k dv^k c^{k+1}, Coord) \in X$:

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

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

$$\mathcal{X} \cup \overline{\mathcal{I}}'$$

where $\overline{\mathcal{I}}'$ is satisfied iff $c^{k+1} \notin Inv$. Thus if there is no satisfying solution, then every one-round execution in X ends with a configuration in Inv, i.e., (10) holds.

$\overline{\mathcal{I}}'$ is almost identical to $\overline{\mathcal{I}}$, except that (i) every first-round version variable var^1 in $\overline{\mathcal{I}}$ is now replaced with its $k+1$-th version var^{k+1} and (ii) Φ is replaced with $\Phi + 1$.

Example 5 Consider the *LastVoting* algorithm for example. In this case we have:

$$INIT := \bigwedge_{p \in \Pi} \left(\begin{array}{l} vote_p^1 = ? \ \wedge \ commit_p^1 = \texttt{false} \\ \wedge ready_p^1 = \texttt{false} \ \wedge \ ts_p^1 = 0 \end{array} \right) \wedge \Phi = 1$$

For the Inv given in Example 2 that refers to phase Φ, we have:

$$\overline{\mathcal{I}} := \neg \bigwedge_{p \in \Pi} \left(\begin{array}{l} commit_p^1 = \texttt{false} \\ \wedge ready_p^1 = \texttt{false} \wedge ts_p^1 < \Phi \end{array} \right)$$

$$\overline{\mathcal{I}}' := \neg \bigwedge_{p \in \Pi} \left(\begin{array}{l} commit_p^{k+1} = \texttt{false} \\ \wedge ready_p^{k+1} = \texttt{false} \wedge ts_p^{k+1} < \Phi + 1 \end{array} \right)$$

5.2. Validating $U(v)$ with Assumption 2

Here we remove Assumption 1; that is, it is not known that $U(v)$ represents v-valent configurations. However, we assume that Inv correctly represents an invariant; that is, $Reachable \subseteq Inv$ (Assumption 2).

Theorem 5 Suppose that $v \in Val$ and $U(v)$ is a set of configurations. Any $c \in U(v) \cap Reachable$ is v-valent if:

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

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 for all $v \in Val$. The problem to be solved is to determine whether or not for any one-phase execution $(c^1 ho^1 dv^1 \cdots c^k ho^k dv^k c^{k+1}, Coord) \in X$, the following condition holds:

$$\begin{array}{l} \forall v \in Val : \\ \quad c^1 \in U(v) \longrightarrow (d = \emptyset \vee d = \{v\}) \wedge c^{k+1} \in U(v) \end{array} \tag{12}$$

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

The problem is reduced to the satisfiability problem of:

$$\mathcal{X} \cup \overline{\mathcal{V}}$$

where $\overline{\mathcal{V}}$ is satisfied by a one-phase execution in X iff it does *not* meet (12). Therefore if $\mathcal{X} \cup \overline{\mathcal{V}}$ is unsatisfiable, then (12) holds for any execution in X, ensuring that any $c \in U(v) \cap Reachable$ is v-valent.

The variables that occur in $\overline{\mathcal{V}}$ are: process variables at c^1 and c^{k+1}; Φ; dv_p^i for all $p \in \Pi, i, 1 \leq i \leq k$; and an auxiliary integer variable V that represents v in (12).

Example 6 For $U(v)$ shown in Example 1, we have:

$$\begin{array}{l} \overline{\mathcal{V}} := (V > 0) \wedge \\ \neg \left(\begin{array}{l} \bigvee_{Q \in Maj} \left(\bigwedge_{p \in Q} x_p^1 = V \wedge \bigwedge_{q \in \Pi \backslash Q} ts_p^1 > ts_q^1 \right) \longrightarrow \\ \quad \bigwedge_{p \in \Pi, 1 \leq i \leq k} (dv_p^i = V \vee dv_p^i = 0) \\ \quad \wedge \bigvee_{Q \in Maj} \bigwedge_{p \in Q} \left(x_p^{k+1} = V \wedge \bigwedge_{q \in \Pi \backslash Q} ts_p^{k+1} > ts_q^{k+1} \right) \end{array} \right) \end{array}$$

6. Case Studies

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

- *LastVoting* (*Paxos*) [6], see Algorithm 1.
- *Hybrid-1(α)*, an improved version of the *Fast Paxos* algorithm [23], presented in [4], see Algorithm 2.

For *LastVoting*, the condition for termination is specified by (1) in Section 2, while $U(v)$ and Inv are given in Examples 1 and 2.

Similarly to *LastVoting*, *Hybrid-1(α)* is always safe if $\alpha \leq \lfloor n/4 \rfloor$. The condition for termination is specified in

Table 1. Properties for *Hybrid-1(α)*

(a) Condition for Liveness

$$\exists \phi > 0, \exists co \in \Pi, \forall p \in \Pi : \begin{cases} |HO(p,r)| > \max(n/2, 2\alpha) \wedge co = Coord(p,\phi) \\ \wedge \quad \forall i, 0 \leq i \leq 2 : co \in HO(p, 3\phi - i) \end{cases}$$

(b) $U(v)$ and Inv

$$U(v) := \exists Q \subseteq \Pi : \begin{cases} |Q| \geq n - \alpha \wedge \\ \forall p \in Q : (x_p = v) \wedge \forall p \in \Pi \backslash Q : (ts_p = 0) \end{cases} \vee \exists Q \subseteq \Pi : \begin{cases} |Q| > n/2 \wedge \\ \forall p \in Q : (x_p = v \wedge \forall q \in \Pi \backslash Q : ts_p > ts_q) \end{cases}$$

$$Inv := \forall p \in \Pi : voteToSend_p = \texttt{false} \wedge ts_p < \phi$$

Table 1(a); $U(v)$ and Inv are given in Table 1(b). *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 12). To prevent processes from deciding different values in later rounds (line 33), 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 term of the $U(v)$ in Table 1(b) states this formally.

Experiments: The experiments were performed on a Linux workstation with an Intel Xeon processor 2.2GHz and 4Gbyte memory. We used the Yices [11] satisfiability solver. For both algorithms, we conducted the four kinds of checks, namely agreement, termination, $U(v)$, and Inv, up to $n = 11$. Table 2 shows the execution time required for these checks. Notation "t.o." (timeout) indicates that the check was not completed within 5 hours.

No satisfiable solution was found in any of the checks. Therefore Table 2 shows that for the two algorithms (i) $U(v)$ and Inv meet Assumption 1 up to $n = 8$, Assumption 2 up to $n = 11$, and (ii) agreement and termination are guaranteed up to $n = 8$ and $n = 11$, respectively.

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* against the agreement property

Algorithm 2 The *Hybrid-1(α)* algorithm ($\alpha \leq \lfloor n/4 \rfloor$)

```
1: Initialization:
2:     x_p ∈ Val, initially v_p   {v_p is the initial 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, p⟩
                 and, except at most α, are equal to ⟨x̄, 0, p⟩ then
15:                vote_p := x̄
16:            else
17:                let θ̄ be the largest θ from ⟨−, θ, p⟩ received
18:                vote_p := one x̄ such that ⟨x̄, θ̄, p⟩ 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
```

with three model checkers: NuSMV [8], SPIN [18], and ALV [2].

SMV and SPIN are 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 [26] allows one to model check the whole state space with these model checkers. 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 nondeterministic ones. This optimization does not work for SMV because HO sets are already compactly represented by the data structure used in SMV. ALV is a well known model checker that can analyze infinite state systems with unbounded integer variables. It does not require any finite state abstraction, while the same optimization as with SPIN was performed to avoid explicit representation of HO sets.

Table 3 presents the maximum number of processes that each of the model checkers was able to handle without running out of memory or time (5 hours) together with the time

Table 2. Execution time (in seconds)

	LastVoting								*Hybrid-1*($\lfloor n/4 \rfloor$)							
n	4	5	6	7	8	9	10	11	4	5	6	7	8	9	10	11
agreement	0.1	0.3	0.6	5.0	10	139	312	11066	0.1	0.8	2.2	57	539	11848	t.o.	t.o.
termination	0.0	0.1	0.1	0.5	1.0	3.6	7.6	32	0.0	0.1	0.3	1.1	3.2	13	28	121
$U(v)$	0.1	0.6	1.7	104	2356	t.o.	t.o.	t.o.	0.3	2.3	8.2	142	16650	t.o.	t.o.	t.o.
Inv	0.0	0.1	0.1	0.4	0.9	2.7	5.2	29	0.0	0.1	0.3	0.8	2.7	10	26	92

Table 3. Traditional approach on *LastVoting* with different model checkers

Model Checker	n	Time (sec)
NuSMV [8]	4	167
SPIN [18]	3	2922
ALV [2]	3	1921

needed to model check the largest model. Comparing Tables 2 and 3 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.

7. Conclusion

We proposed a novel approach to automatic verification of asynchronous consensus algorithms. Using the notions of univalence and invariant, 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 two consensus algorithms and mechanically verified that they satisfy agreement up to 8 processes and termination up to 11 processes. Comparing the performance of the traditional model checking approach showed that the performance improvement was significant.

Acknowledgments The SPIN model for *LastVoting* was provided by Takahiro Minamikawa at Osaka University.

References

[1] P. Bokor, A. Pataricza, M. Serafini, and N. Suri. Model checking of distributed dependable protocols using semantic property preserving abstractions. Technical Report TR-TUD-DEEDS-09-01-2007, TU Darmstadt, Sept. 2007.

[2] T. Bultan and T. Yavuz-Kahveci. Action language verifier. In *Proc. 16th IEEE Int'l Conf. on Automated Software Engineering (ASE '01)*, pages 382–386, San Diego, CA, USA, 2001.

[3] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of ACM*, 43(2):225–267, 1996.

[4] B. Charron-Bost and A. Schiper. Improving Fast Paxos: Being optimistic with no overhead. In *Proc. of 12th Pacific Rim International Symposium on Dependable Computing (PRDC'06)*, pages 287–295, Riverside, CA, USA, Dec. 2006. IEEE CS Press.

[5] B. Charron-Bost and A. Schiper. Harmful dogmas in fault tolerant distributed computing. *SIGACT News*, 38(1):53–61, Mar. 2007.

[6] B. Charron-Bost and A. Schiper. The Heard-Of Model: Computing in Distributed Systems with Benign Failures. Technical Report LSR-REPORT-2007-001, EPFL, 2007.

[7] L. Cheung. Randomized wait-free consensus using an atomicity assumption. In *Proc. 9th International Conference on Principles of Distributed Systems (OPODIS'05)*, volume 3974 of *LNCS*, pages 47–60, Pisa, Italy, Dec. 2005. Springer.

[8] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV 2: An opensource tool for symbolic model checking. In *Proc. of 14th Conf. on Computer Aided Verification (CAV 2002)*, volume 2404 of *LNCS*, Copenhagen, Denmark, July 2002. Springer.

[9] E. Clarke, A. Biere, R. Raimi, and Y. Zhu. Bounded model checking using satisfiability solving. *Formal Methods in System Design*, 19(1):7–34, 2001.

[10] L. de Moura, H. Rueß, and M. Sorea. Bounded model checking and induction: From refutation to verification. In *Proc. of 15th Conf. on Computer Aided Verification (CAV 2002)*, volume 2725 of *LNCS*, pages 14–26, July 2003.

[11] B. Dutertre and L. M. de Moura. A fast linear-arithmetic solver for DPLL(T). In *Proc. of 18th Conf. on Computer Aided Verification (CAV 2006)*, volume 4144 of *LNCS*, pages 81–94, Seattle, USA, Aug. 2006. Springer.

[12] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of ACM*, 35(2):288–323, 1988.

[13] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of ACM*, 32(2):374–382, 1985.

[14] E. Gafni. Round-by-round fault detectors: Unifying synchrony and asynchrony. In *Proc. 17th ACM Symp. on Principles of Distributed Computing (PODC-17)*, pages 143–152, New York, NY, USA, 1998. ACM Press.

[15] E. Gafni and L. Lamport. Disk Paxos. *Distributed Computing*, 16(1):1–20, 2003.

[16] R. Guerraoui and A. Schiper. The generic consensus service. *IEEE Trans. on Software Engineering*, 27(1):29–41, 2001.

[17] M. Hendriks. Model checking the time to reach agreement. In P. Pettersson and W. Yi, editors, *3rd International Conference on the Formal Modeling and Analysis of Timed Systems (FORMATS'05)*, volume 3829 of *LNCS*, pages 98–111. Springer–Verlag, 2005.

[18] G. J. Holzmann. The model checker SPIN. *IEEE Trans. on Software Engineering*, 23(5):279–295, 1997.

[19] M. Z. Kwiatkowska and G. Norman. 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*, pages 194–209, Houston, USA, 2002. Springer-Verlag.

[20] M. Z. Kwiatkowska, G. Norman, and R. Segala. Automated verification of a randomized distributed consensus protocol using Cadence SMV and PRISM. In *Proc. of 13th Conf. on Computer Aided Verification (CAV 2001)*, volume 2102 of *LNCS*, pages 194–206, Paris, France, July 2001. Springer.

[21] L. Lamport. The part-time parliament. *ACM Trans. on Computer Systems*, 16(2):133–169, May 1998.

[22] L. Lamport. Personal Communication, 2006.

[23] L. Lamport. Fast Paxos. *Distributed Computing*, 19(2):79–103, Oct. 2006.

[24] N. Santoro and P. Widmayer. Time is not a healer. In *Proceedings of the 6th Annual Symposium on Theoretical Aspects of Computer Science (STACS'89)*, volume 346 of *LNCS*, pages 304–313, Paderborn, Germany, Feb. 1989. Springer.

[25] A. Schiper and S. Toueg. From Set Membership to Group Membership: A Separation of Concerns. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 3(1):2–12, Jan.-March 2006.

[26] T. Tsuchiya and A. Schiper. Model checking of consensus algorithms. In *Proc. 26th Symp. on Reliable Distributed Systems (SRDS)*, pages 137–148, Beijing, China, Oct. 2007.

[27] P. Zieliński. Automatic verification and discovery of Byzantine consensus protocols. In *Proc. Int'l Conf. on Dependable Systems and Network (DSN 2007)*, pages 72–81, Edinburgh, UK, June 2007. IEEE CS Press.