Model Checking Round-Based Distributed Algorithms

Xin An
Shandong University
School of Computer Science and Technology
Jinan, 250101 China

Jun Pang
University of Luxembourg
Computer Science and Communications
6, rue Richard Coudenhove-Kalergi
L-1359 Luxembourg

***Abstract*—In the field of distributed computing, there are many round-based algorithms to solve fundamental problems, such as leader election and distributed consensus. Due to the nature of these algorithms, round numbers are unbounded and can increase infinitely during executions of the algorithms. This can lead to the state space explosion problem when verifying correctness of these algorithms using model checking.**

In this paper, we present a general idea of investigating the bounded distance of round numbers in round-based algorithms. We can manage to transform their state spaces into finite by maintaining such relations in a proper way, and thus make automatic verification of these algorithms possible. We apply this idea to several algorithms and present their verification results in the model checker Spin.

I. Introduction

Today computer-based systems, e.g., telecommunication systems, banking systems, airline reservation systems, highway and air traffic control systems are widely used in our daily life. Most of these systems are critically dependent on their underlying distributed algorithms to achieve specific goals such as distributed agreement and fault tolerance. Clearly, it is important for the algorithms to run correctly and efficiently. Failures of these systems can be potentially disastrous, and cause the loss of a huge amount of money or even human lives. However, since the settings in which these algorithms operate are complicated, design of such algorithms is error-prone and can be an extremely difficult and complex task. Usually, specifications of correctness arguments of these algorithms are given at an informal level. It is insufficient to convince all the readers and provide high degree of confidence of their correctness under all circumstances. If one wants to verify the correctness of some proofs, she/he has to prove substantial parts or entire sub-results, for which only informal arguments were given. This has been observed in a recent paper [1] on formal reasoning about the correctness of a distributed consensus algorithm [2].

Model checking is an automatic technique for verifying finite state concurrent systems. It verifies a desired property over a given system through an *exhaustive exploration* of all the states reachable by the system. Due to the finiteness of the state space, the search always terminates. Techniques such as symbolic representation, symmetry reduction and abstraction, have been developed to deal with the *state explosion* problem and enhance the scalability of model checking. It is widely agreed that the proper use of model checking gives more confidence in the correctness of computer-based systems. In the last several decades, it has been used successfully in practice to verify complex sequential circuit designs and communication protocols.

In the field of distributed computing, for asynchronous anonymous networks there exist no deterministic solutions for many fundamental problems, including leader election and distributed consensus. Additional techniques such as randomization, failure detector are introduced to circumvent these problems. An apparent characteristic of these solutions is the involvement of round numbers. Due to the nature of these algorithms, the round numbers are unbounded and can increase infinitely during executions of the algorithms. This gives rise to the state space explosion problem of model checking and thus makes the direct use of model checking to verify correctness of these algorithms infeasible. However, instead of having an explicit representation of round numbers, by investigating their relations in the studied algorithms and maintaining such relations in a proper way, we can manage to transform their state spaces into finite and make automatic verification of these algorithms possible. In this paper, we aim to model check several round-based distributed algorithms based on this idea.

The rest of the paper is structured as follows: In Section II, we present our main verification idea to circumvent the state space explosion problem when model checking round-based distributed algorithms in depth. We apply our approach to the probabilistic Itai-Rodeh algorithm [3], [4] and its one variation, which is newly introduced in this paper, for leader election in Section III, a probabilistic consensus algorithm proposed by Bracha and Toueg [5] and a rotating coordinator algorithm with failure detector [6, pp. 154] in Section IV, respectively. Their verification results in Spin [7] are summarized in Section V. In the end, we briefly discuss related work in Section VI and conclude this paper with future research directions in Section VII.

II. Our Approach

The presence of round numbers in most round-based distributed algorithms causes the state space explosion problem, which makes it impossible to verify them directly using model checking. To have an automatic verification of these algorithms, we need to transform their state space into finite, that is to have a finite representation of round numbers. (For some algorithms, we also need to have a finite representation for message channels. This typically follows when we have a finite representation for round numbers.) Meanwhile, we need to keep all possible behaviors of the original algorithms. We can achieve this goal by the following steps:

STEP 1: We need to investigate and find relations of round numbers. That is to find the inherent bounded distance on round numbers of the involved processes in the algorithms. Such a bounded distance implies that only a finite number of round numbers are needed during the executions of such algorithms, even though the round numbers are unbounded. This provides the theoretical foundation for us to have a finite representation of unbounded round numbers.

STEP 2: We need to represent unbounded round numbers by keeping their (relative) relations. Based on the first step, we can use modulus (of the found distance) to model round numbers. Moreover, as all algorithms with unbounded round numbers involve comparisons of round numbers, we also need to keep their relations. In our transformation, we usually have to trace the latest round number (by a global variable) to indicate their relative order.[1]

STEP 3: We need to make sure that our transformations still capture all possible behaviors in the original algorithms. Usually, by representing unbounded round numbers in a finite way, some conflicts of round numbers may arise, especially in asynchronous networks where messages may stay for an arbitrary long time. These messages in previous rounds need to be marked as special at an appropriate time, so that they would be recognized by the involved processes in a same way as in the algorithm's original specification.

Once we have done these steps properly, we can model these algorithms with bounded round numbers and apply model checking for the verification of round-based distributed algorithms directly. We illustrate our approach (focusing on STEP 1) to a number of algorithms.

III. LEADER ELECTION ALGORITHMS

In this section, we consider an *asynchronous*, *anonymous*, *unidirectional* ring consisting of n processes $p_1, \ldots, p_n$. Processes communicate asynchronously by message passing over reliable channels with capacity n.

Leader election is the problem of electing a unique leader in a network, in the sense that the leader knows that it has been elected and the other processes know that they have not been elected. We concentrate on model checking of the Itai-Rodeh algorithm [3], [4] as well as a new leader election algorithm based on it. We describe both algorithms in Section III-A. Then we investigate the relation of round numbers of each algorithm in Section III-B.

[1]Note that this global variable does not play a role in the studied distributed algorithm, it is only used in our transformed model for the aim of automatic verification.

A. The algorithms

In an anonymous network, processes do not carry an identity. Angluin [8] show that there does not exist a terminating algorithm for electing a leader in an asynchronous anonymous network. The Itai-Rodeh algorithm [3], [4] is the first probabilistic solution for leader election in anonymous unidirectional rings. In this algorithm, each process selects a *random identity* from a finite set and sends out a message carrying its identity. A *hop counter* is included in each message, which helps processes to recognize its own messages (by checking whether the hop counter equals the ring size n). A *bit* is also included for each message to indicate whether some other processes with the same identity exist. The bit is dirtied when it passes a process that is not its originator but has the same identity. When some process receives its own message, it either becomes the leader (if the bit is clean), or selects a new identity and starts the next election round (if the bit is dirty). In next election round, only processes that shared the largest identity are *active*. All other processes have been made *passive* by the receipt of a message with an identity larger than their own.

Algorithm 1 The Itai-Rodeh algorithm

```
• Initially, all processes are active, and each process $p_i$ randomly selects its identity $id_i \in \{1, \ldots, k\}$ and sends the message $(id_i, 1, 1, true)$.

• Upon receipt of a message $(id, round, hop, bit)$, a passive process $p_i$ ($state_i = passive$) passes on the message, increasing the counter *hop* by one; an active process $p_i$ ($state_i = active$) behaves according to the following steps:
  - if $hop = n$ and $bit = true$, then $p_i$ becomes the leader ($state'_i = leader$);
  - if $hop = n$ and $bit = false$, then $p_i$ selects a new random identity $id'_i \in \{1, \ldots, k\}$, moves to the next round ($round'_i = round_i + 1$), and sends the message $(id'_i, round'_i, 1, true)$;
  - if $(round, id) = (round_i, id_i)$ and $hop < n$, then $p_i$ passes on the message $(id, round, hop + 1, false)$;
  - if $(round, id) > (round_i, id_i)$ (where $(round, id)$ and $(round_i, id_i)$ are compared lexicographically), then $p_i$ becomes passive ($state'_i = passive$) and passes on the message $(id, round, hop + 1, bit)$;
  - if $(round, id) < (round_i, id_i)$, then $p_i$ purges the message.
```

The Itai-Rodeh algorithm makes no assumptions about channel behavior, except *fair scheduling* meaning that in each infinite execution sequence, every sent message eventually arrives at its destination. Since the channels are *non-FIFO*, an old message, that has been overtaken by other messages in the ring, could in principle result in a situation where no leader is elected [9]. To avoid this problem, the algorithm proceeds in successive rounds, each process and message is supplied with a *round number*, which initially starts at one and is augmented at each new election round. Thus an old message can be recognized and ignored. Due to the use of round numbers, the Itai-Rodeh algorithm has an infinite state space. Algorithm 1 describes the algorithm in details. Each process p_i maintains three parameters: its identity id_i, its current state $state_i$ (which ranges over active or passive or leader) and its current round number $round_i$. The messages are of this form: (id, $round$, hop, bit).

Fokkink and Pang [9] present two variations of the Itai-Rodeh algorithm with *FIFO* channels, in which round numbers are not needed any more. Their second algorithm is based on the observation that when some process p_i detects an identity clash, which means it receives a message with its own identity and hop count smaller than n, it is not necessary for p_i to wait for its own message. Instead, p_i can immediately select a new random identity and start a new election round. Similarly, we apply this idea to the original Itai-Rodeh algorithm and develop a new leader election algorithm for anonymous unidirectional rings. The algorithm (see Algorithm 2) makes the same assumptions as the Itai-Rodeh algorithm, but omits all the occurrences of dirty bit.

Algorithm 2 The Itai-Rodeh algorithm without dirty bit

```
• Initially, all processes are active, and each process $p_i$ randomly selects its identity $id_i \in \{1, \dots, k\}$ and sends the message $(id_i, 1, 1)$.

• Upon receipt of a message $(id, round, hop)$, a passive process $p_i$ $(state_i = passive)$ passes on the message, increasing the counter $hop$ by one; an active process $p_i$ $(state_i = active)$ behaves according to the following steps:
    - if $hop = n$, then $p_i$ becomes the leader ($state'_i = leader$);
    - if $(round, id) = (round_i, id_i)$, then $p_i$ selects a new random identity $id'_i \in \{1, \dots, k\}$, moves to the next round ($round'_i = round_i + 1$), and sends the message $(id'_i, round'_i, 1)$;
    - if $(round, id) > (round_i, id_i)$ (where $(round, id)$ and $(round_i, id_i)$ are compared lexicographically), then $p_i$ becomes passive ($state'_i = passive$) and passes on the message $(id, round, hop + 1)$;
    - if $(round, id) < (round_i, id_i)$, then $p_i$ purges the message.
```

B. Finding relations of round numbers

In the Itai-Rodeh algorithm, round numbers are used to distinguish the processes as well as their messages in different election rounds. Processes who have lost the election in previous election rounds have become passive and only forward received messages. According to the algorithm, an active process can enter into the next election round if and only if its own message has visited all the other processes ($hop = n$) and comes back with a dirtied bit ($bit = false$), which ensures that only those processes with the same largest identity (and round number) remain active. Intuitively, this implies that during the executions of the Itai-Rodeh algorithm, the distance of round numbers (of active processes) is at most one. Similar to the Itai-Rodeh algorithm, we find that the distance of round numbers of active processes in Algorithm 2 is bounded by $n - 1$. It is because active processes may enter next round as soon as possible, and the ring is of size n. We formulate the results as Theorems 1 and 2.

Theorem 1. *The distance of round numbers of active processes in Algorithm 1 is at most* 1.

Theorem 2. *The distance of round numbers of active processes in Algorithm 2 is at most* $n - 1$.

IV. Consensus algorithms

In this section, we consider an *asynchronous*, *fully connected* network consisting of n processes $p_1, \dots, p_n$. The communication between processes is *asynchronous* by message passing over reliable channels. A process may *crash* during the executions.

Consensus is the problem to, despite of the occurrences of failures, get all correct processes in a system to uniformly decide on some value. Initially, each process has an initial value 0 or 1 and must eventually decide on some value satisfying the following properties: *Termination* – all correct processes must eventually decide some value; *Agreement* – the decision of each correct process must be identical; *Validity* – the value that has been decided must have been proposed by some process. To verify consensus algorithms, all the three properties need to be checked. However, validity is usually trivially satisfied for most consensus algorithms [10]. Instead of checking whether the other two properties (agreement and termination) hold, we focus on the property that *eventually some process decides*. This is based on the fact that this property usually implies the correctness of agreement and termination. By focusing on this property which guarantees the correctness of the algorithm, we can reduce all the behaviors after some process has decided in our built models of this algorithm.

The impossibility result [11], namely no asynchronous, deterministic, 1-crash-robust consensus algorithm exists, has led to weaker formulations and stronger models in order to solve the consensus problem. The former technique includes the introduction of randomization techniques, the probabilistic consensus algorithm proposed by Bracha and Toueg in [5] is of this kind. The latter includes the involvement of synchrony and (unreliable) failure detector [12], [2], one example is a rotating coordinator algorithm with the failure detector $\diamond S$ [6, pp.514]. The maximal number of faulty processes that can be handled by an algorithm is called the *resilience* of the algorithm, and is denoted by t.

For aforementioned two algorithms, we apply our approach (Section II) to deal with the problem with unbounded round numbers (and unbounded message channels), which causes state space explosion in model checking.

A. A crash-robust consensus algorithm

Bracha and Toueg propose a probabilistic consensus algorithm [5] and prove that under the assumption of *fair* *scheduling* their algorithm terminates with probability 1. An upper bound on the resilience $t < n/2$ is also given. The algorithm operates in *rounds*. In each round k, correct process p_i, which has not decided (i.e., $y_p = b$), sends a message to all processes (including itself) and awaits the receipt of $n - t$ messages of the same round. The message includes p_i's value, its current round k, as well as a weight. The *weight* is the number of votes received for the vote in the previous round (1 in the first round). The message with a weight bigger than $n/2$ for some vote is called a *witness*. If p_i receives a witness in round k, then p_i changes its vote to the witness's; otherwise p_i chooses the vote that is the majority of the received votes. A decision is made when p_i receives more than t witnesses in some round, and in order to make other processes decide, p_i still needs to send witness for the next two rounds. Algorithm 3 taken from [6, pp.455] describes the details.

As operating in the asynchronous network model, processes may receive messages with different rounds and need to process these messages in the appropriate round. In the algorithm, the messages from previous rounds are ignored and from future rounds are buffered in message channels. Moreover, different processes may take different subsets of all the messages of a same round into account.

As there are non-terminating paths (the algorithm is ensured to terminate with probability 1), round numbers can increase infinitely. Different from the Itai-Rodeh algorithm, we cannot find the inherent bounded distance of round numbers for the algorithm. Instead, we focus on the cases where only one process is allowed to crash. Considering related works of verifying consensus algorithms that only small cases are possible (typically only 3 or 4 nodes) to be verified [10], we believe that model checking these 1-crashed cases is still an interesting study.

For 1-crashed cases of the algorithm, each correct process needs $n-1$ messages in each round to proceed. If some correct process p_i lags in some round r_i, then from round $r_i + 1$ on, all other correct processes have to collect all votes except the one from p_i to proceed. At the end of round r_i+1, they would choose the same vote (say v) and enter round $r_i + 2$. In round $r_i + 2$, if p_i does not catch up, they would receive the same $n - 1$ number of votes v and start round $r_i + 3$ with sending out witness. Then in round r_i+3, some process would decide. Thus, the distance of correct processes is at most 3. The other source for state space explosion in consensus algorithms is unbounded message channels. Since each process sends out messages in each round, the size of message channels are also bounded by $n+3(n-1) = 4n-3$ (due to the bounded distance of round numbers).

Theorem 3. *Before some process decides, the distance of round numbers of correct processes for the 1-crash cases of Algorithm 3 is at most* 3.

Proposition 1. *The size of message channels necessary for each correct process in the 1-crash cases of Algorithm 3 is* $4n - 3$.

Algorithm 3 A crash-robust consensus algorithm

```
var value_p          : $(0,1)$    init $x_p$
    round_p          : integer    init $0$
    weight_p         : integer    init $1$
    msgs_p$[0,1]$     : integer    init $0$
    witness_p$[0,1]$  : integer    init $0$

while $y_p = b$ do
  (*Reset counts*)
  witness_p[0], witness_p[1], msgs_p[0], msgs_p[1] := $0,0,0,0$;
  shout $\langle \textbf{vote}, round_p, value_p, weight_p \rangle$;
  while $msgs_p[0] + msgs_p[1] < n - t$ do
    receive $\langle vote, r, v, w \rangle$;
    if $r > round_p$ then
      send $\langle \textbf{vote}, r, v, w \rangle$ to $p$
    else if $r = round_p$ then
      $msgs_p[v] := msgs_p[v] + 1$;
      if $w > n/2$ then
        $witness_p[v] := witness_p[v] + 1$
      end if
    else
      skip
    end if
  end while
  (*Choose new value: vote and weight in next round*)
  if $witness_p[0] > 0$ then
    $value_p := 0$
  else if $witness_p[1] > 0$ then
    $value_p := 1$
  else if $msgs_p[0] > msgs_p[1]$ then
    $value_p = 0$
  else
    $value_p = 1$;
  end if
  $weight_p := msgs_p[value_p]$
  (*Decide if more than $t$ witnesses*)
  if $witness_p[value_p] > t$ then
    $y_p := value_p$
  end if
  $round_p := round_p + 1$
  (*Help other processes decide*)
end while
shout $\langle \textbf{vote}, round_p, value_p, n - t \rangle$
shout $\langle \textbf{vote}, round_p + 1, value_p, n - t \rangle$
```

The property that eventually some process decides implies agreement and termination is captured by the following lemma [6, Lemma 14.21].

Lemma 1. *For Algorithm 3, if a process decides then all correct processes decide for the same value, and at most two rounds later.*

B. A rotating coordinator algorithm

In this section, we look at a rotating coordinator consensus algorithm with a failure detector $\diamond S$ and $t < n/3$ [6, pp.514]. The $\diamond S$ satisfies two properties: *Strong Completeness* – eventually all crashed processes are detected and *Eventually weak accuracy* – there exists a time after which some correct process is never suspected.

This algorithm operates in rounds and is based on the rotating coordinator paradigm [2]. In round k, all processes know that the coordinator p_c is the one with process number $k \mid_n +1$ and send the messages including their values (x) and round (r) to the coordinator p_c. The coordinator p_c waits until $n - t$ messages are received. Then, the coordinator p_c evaluates the outcome of this round k according to the majority of received values and broadcasts the outcome. The outcome consists of the chosen value (v) of this round, the round number as well as a bit (d) used to indicate unanimousness of the received votes. A process p_i which has sent its value to p_c waits either to receive the outcome from the coordinator or to suspect the coordinator. In the first case, p_i collects the outcome of this round and proceeds to the next round. In the second case, p_i just moves to the next round. A decision is taken when p_i receives an outcome from unanimous votes. The process continues its activity after a decision until all correct processes decide. Algorithm 4 taken from [6, pp.514] describes the details.

Algorithm 4 A rotating coordinator consensus algorithm

```
$x_i := input$;
$r := 0$;

**while** true **do**
    (*Start new round and compute coordinator*)
    $r := r + 1$; $c := r \mid_n +1$;
    (*Phase 1: all processes send value to coordinator*)
    send $\langle \textbf{value}, x_i, r \rangle$ to $P_c$;
    (*Phase 2: coordinator evaluates outcome*)
    **if** $i = c$ **then**
        wait until $n - t$ messages $\langle \textbf{value}, v_j, r \rangle$ are received;
        $v :=$ majority of received values;
        $d := (\forall j : v_j = v)$;
        (*range over received messages*)
        **forall** $j$ **do** send $\langle \textbf{outcome}, d, v, r \rangle$ to $p_j$
    **end if**
    (*Phase 3: evaluate the round*)
    **if** collect $\langle \textbf{outcome}, d, v, r \rangle$ from $p_c$ **then**
        $x_i := v$;
        **if** $d$ **then**
            decide(v)
        **end if**
    **end if**
**end while**
```

Due to the fact that properties of $\diamond S$ only hold eventually, round numbers in this algorithm can increase infinitely. Similar to the algorithm in Section IV-A, we focus on the cases where only one process is allowed to crash. In our discussion, we assume that each outcome evaluated by the current coordinator is always received by itself in that round.

Similarly, if some correct process p_i lags in round r_i, then when all correct processes except p_i play the coordinator in some round r_j which is from $r_i + 1$ to $r_i + n$, they have to collect all their votes and would have chosen the same value. Then if p_i does not catch up, when anyone of these correct processes plays coordinator again from round $r_i + n + 1$ to $r+2n$, it would receive the unanimous $n-1$ values and decide. Correspondingly, with bounded round numbers, the size of the message channels are also bounded.

Theorem 4. *Before some process decides, the distance of round numbers of correct processes for the 1-crash cases of Algorithm 4 is at most $2n$.*

Proposition 2. *The size of the message channels necessary for each correct process in the 1-crash cases of Algorithm 4 is $2n - 1$.*

The property that eventually some process decides implies agreement and termination is described in Lemma 2.

Lemma 2. *In 1-crash cases of Algorithm 4, if a process decides then all correct processes decide for the same value, and at most $2n$ rounds later.*

V. Model Checking Results in Spin

We present verification results of the four algorithms in the model checker Spin [7] (with the options "weak fairness", "use partial order reduction" and "use compression") based on the relations of round numbers we have found in Sections III and IV. Tables I-IV mainly contain information about size of the models (number of states and transitions) and memory/time consumption. Experiments are performed either on a computer with a CPU of 2.26 GHz and 2Gb main memory (for cases with memory consumption $\leq 2G$) or on a powerful machine (of a cluster) with CPU of 3.4 GHz and 32Gb main memory (for cases with memory consumption greater than 2G). Due to space limitation, details about our models in Spin are left out and can be found in an extended version of the paper [13].

A. Verifying the leader election algorithms

Based on the identified distances of round numbers (Theorems 1 and 2), we are able to have a finite representation of round numbers for both algorithms in Section III.

In our Spin models, instead of increasing round number by one each time when entering next election round, an active process proceeds with changing its round number by $round = (round + 1) \mid_2$ for Algorithm 1 and $round = (round + 1) \mid_n$ for Algorithm 2, respectively. Furthermore, we need to maintain the relations of round numbers to track which round is the latest election round. In our Spin models, we use a global variable to indicate the latest election round. It adjusts its value by monitoring the round numbers of all active processes. Moreover, due to the fact that the communication channels are *non-FIFO*, even though some processes have become passive their messages may still stay in the channels. Therefore, when we transform the algorithm using bounded round numbers we need to recognize the delayed messages as well. For this purpose, we use a dedicated procedure to mark these messages as *old* at the end of each election round (when all active processes have entered into next election round). The desirable property of leader election is specified as an LTL formula in the form of "$\neg \Box\Diamond p \Rightarrow \Diamond\Box\, q$". The formula says that if the situation where more than one active processes have the same (largest) identity (captured by p) does not appear infinitely often[2] then an unique leader (captured by q) will be eventually elected. In Tables I and II, the first three columns give the number of processes (n), the number of identities (m), and the size of the message channels (s).

TABLE I
VERIFICATION RESULTS OF ALGORITHM 1

n	m	s	#states	#trans.	mem.(Mb)	time(s)
2	2	2	$2.3*10^3$	$2.3*10^4$	2.6	0.06
3	2	3	$6.3*10^5$	$1.3*10^7$	42.3	37.4
3	3	3	$2.0*10^6$	$4.4*10^7$	129.1	131
4	2	4	$9.2*10^7$	$1.5*10^9$	10,479.1	8,770

TABLE II
VERIFICATION RESULTS OF ALGORITHM 2

n	m	s	#states	#trans.	mem.(Mb)	time(s)
2	2	2	$1.4*10^3$	$1.7*10^4$	5.2	0.04
3	2	3	$5.3*10^5$	$1.5*10^7$	48.0	42.9
3	3	3	$2.0*10^6$	$6.4*10^7$	172.5	180
4	2	4	$2.7*10^8$	$1.1*10^{10}$	20,567.1	51,600

B. Verifying the consensus algorithms

With theoretical results in Section IV, we are able to have finite representations of unbounded round numbers (and message channels) for Algorithms 3 and 4.

In our Spin models, instead of increasing round number by one each time when entering next election round, each process proceeds with changing its round number by $round = (round+1)\mid_4$ for Algorithm 3 and $round = (round+1)\mid_{3n}$ for Algorithm 4, respectively. Since both algorithms operate in rounds, in each round all processes either broadcast their votes or send their votes to the coordinator. Each process in Algorithm 3 or the coordinator in Algorithm 4 proceeds with those messages of the same round number as its own. Thus, in our Spin model we simply define the message channels dedicated to a fixed number of different rounds, they are shared by all processes. Each process sends messages to the corresponding channels according to its current round number and proceeds the messages in the corresponding channel of the same round. Implementing channels in this way simplifies the models and gives rise to smaller state spaces. For Algorithm 3, to make sure that processes may take different subset of messages into account, we model that each process randomly chooses $n-t$ messages from the channels. For Algorithm 4, we model the failure detector $\diamond S$ by two global variables to indicate when it has become *strongly complete* and/or *weakly accurate*. These two variables will be used by processes when they suspect another process to be crashed.

[2]Note that the Itai-Rodeh algorithm is a Las Vegas algorithm terminating with probability 1.0.

The property "eventually some process decides" for Algorithm 3 is specified in a similar way as for the leader election algorithms, where we have to rule out the possibility for the situation where no "progress" is made towards a decision to appear infinitely often. While for Algorithm 4, it is an LTL formula "$\Diamond\Box\, q$", the predicate q captures that some process has decided. In Table III the algorithm takes any initial inputs to the processes, while due to the complexity of Algorithm 4 we need to fix the inputs for the processes (we only take the isomorphically distinct cases).

TABLE III
VERIFICATION RESULTS OF ALGORITHM 3

n	#states	#trans.	mem.(Mb)	time(s)
3	$1.1*10^5$	$7.9*10^5$	9.6	2.0
4	$3.4*10^6$	$4.1*10^7$	174.1	113
5	$1.8*10^7$	$5.5*10^8$	1,413.3	1,600

TABLE IV
VERIFICATION RESULTS OF ALGORITHM 4

n	inputs	#states	#trans.	mem.(Mb)	time(s)
4	1000	$4.0*10^6$	$1.6*10^8$	329.8	516
4	0100	$7.1*10^5$	$3.6*10^7$	21.3	31
4	0010	$2.2*10^6$	$1.1*10^8$	73.9	98.2
4	0001	$5.2*10^6$	$2.8*10^8$	249.0	242
4	1100	$6.1*10^7$	$2.6*10^9$	7,909.6	12,800
4	1010	$5.2*10^7$	$2.2*10^9$	7,398.1	11,000
4	1001	$6.0*10^7$	$2.6*10^9$	7,839.9	11,900

VI. RELATED WORK

Two variations of the Itai-Rodeh algorithm are proposed by Fokkink and Pang [9] for anonymous unidirectional rings with FIFO channels. Both of them are finite-state as round numbers are omitted. They have model checked their algorithms with up to 5 processes in PRISM. Faragó model checks several variations of the Itai-Rodeh algorithm with FIFO channels in Spin [14], including the above two [9], by either putting upper bounds on round numbers or having round numbers modulo a fixed integer to make state spaces finite. He has managed to check algorithms with up to seven processes. Since channels are FIFO, round numbers are actually not needed [9]. Hence, Faragó does not give a way to solve the inherent infinite state space problem when model checking these algorithms. Differently, we identify bounds on distance of round numbers of active processes and use it to achieve a finite representation of round numbers.

The papers on combining model checking and other proof techniques for verifying consensus algorithms and on model checking consensus algorithms with fixed numbers of processes and rounds have already been discussed by Tsuchiya and Schiper [10], [15]. Tsuchiya and Schiper present a first (fully) automatic verification of consensus algorithms using NuSMV without imposing any restrictions on the number of rounds [10]. They reduce agreement and termination verification to the problem of model checking the algorithm with a global round, and use bounded model checking to effectively verify consensus problems [15]. The reason why they can reduce their verification is justified by the reduction theorem of Chaouch-Saad, Charron-Bost and Merz [16]. Chaouch-Saad, Charron-Bost and Merz [16] model check the *OneThirdRule* consensus algorithm after applying the reduction theorem. However, the consensus algorithms they discuss are all based on the Heard-Of model. Moreover, the reduction theorem [16] has the assumption of communication-closedness and can only be applied to the verification of local properties. [3] We concentrate on automatic verification, but our work is still different from theirs. Their verification is largely due to the high abstraction provided by their network model. Thus, when model checking Heard-Of model-based consensus algorithms, one no longer has to explicitly consider messages buffered in the channels. Due to the difference in network models, we have to model message channels explicitly. Thus, our models are more complicated and have an additional cause for state space explosion. This requires us to find bounds on message channel size as well. The other difference is the way on how to deal with unbounded round numbers. They focus on either maintaining relative order of pairs of round numbers [10] or a global round of the checked algorithm [15], [16], while we aim to find bounds on distance of round numbers. Our approach is not limited to consensus problems, it can be also applied to other algorithms such as leader election.

VII. Conclusion

We have presented a general idea of studying the relations of round numbers in round-based distributed algorithms, which normally have infinite state spaces. Based on the identified relations (bounded distance), it is possible to transform state spaces of the algorithms under verification into finite, and thus automatic verification of these algorithms is made possible. We have tested our idea on the Itai-Rodeh algorithm and a newly proposed algorithm for leader election and two (different types of) algorithms for distributed consensus. Their model checking results in Spin look quite promising.

In future, we want to look at other distributed algorithms including recently developed algorithms for mobile ad hoc networks, as we strongly believe that model checking is a crucial automatic technique to guarantee correctness of complicated systems and algorithms. Another research direction is to combine the presented verification idea with other approaches dealing with state space explosion to make verification of distributed algorithms more scalable.

[3]Consensus can be formulated as a local property, while it is not the case for leader election.

Acknowledgment

We want to thank Wan Fokkink for helpful discussions.

References

[1] R. Fuzzati, M. Merro, and U. Nestmann, "Distributed consensus, revisited," *Acta Informatica*, vol. 44, no. 6, pp. 377–425, 2007.

[2] T. D. Chandra and S. Toueg, "Unreliable failure detectors for reliable distributed systems," *Journal of the ACM*, vol. 43, no. 2, pp. 225–267, 1996.

[3] A. Itai and M. Rodeh, "Symmetry breaking in distributive networks," in *Proc. 22nd Annual IEEE Symposium on Foundations of Computer Science*, 1981, pp. 150–158.

[4] ——, "Symmetry breaking in distributed networks," *Information and Computation*, vol. 88, no. 1, pp. 60–87, 1990.

[5] G. Bracha and S. Toueg, "Asynchronous consensus and broadcast protocols," *Journal of the ACM*, vol. 32, no. 4, pp. 824–840, 1985.

[6] G. Tel, *Introduction to Distributed Algorithms*. Cambridge University Press, 2000.

[7] G. J. Holzmann, *The Spin Model Checker: Primer and Reference Manual*. Addison-Wesley, 2003.

[8] D. Angluin, "Local and global properties in networks of processors (extended abstract)," in *Proc. 12th Annual ACM Symposium on Theory of Computing*, 1980, pp. 82–93.

[9] W. J. Fokkink and J. Pang, "Variations on Itai-Rodeh leader election for anonymous rings and their analysis in PRISM," *Journal of Universal Computer Science*, vol. 12, no. 8, pp. 981–1006, 2006.

[10] T. Tsuchiya and A. Schiper, "Model checking of consensus algorithms," in *Proc. 26th IEEE International Symposium on Reliable Distributed Systems*, 2007, pp. 137–148.

[11] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *Journal of the ACM*, vol. 32, no. 2, pp. 374–382, 1985.

[12] T. D. Chandra, V. Hadzilacos, and S. Toueg, "The weakest failure detector for solving consensus," in *Proc. 11th Annual ACM symposium on Principles of Distributed Computing*, 1992, pp. 147–158.

[13] X. An, "Model checking round-based distributed algorithms," Master's thesis, University of Luxembourg, 2009.

[14] D. Faragó, "Model checking of randomized leader election algorithms," Master's thesis, Universität Karlsruhe, 2007.

[15] T. Tsuchiya and A. Schiper, "Using bounded model checking to verify consensus algorithms," in *Proc. 22nd International Symposium on Distributed Computing*, 2008, pp. 466–480.

[16] M. Chaouch-Saad, B. Charron-Bost, and S. Merz, "A reduction theorem for the verification of round-based distributed algorithms," in *Proc. LIX Colloquium Reachability Problems*, ser. Lecture Notes in Computer Science, vol. 5797. Springer, 2009, pp. 93–106.

Appendix

Proof of Theorem 1

Proof: We apply induction to prove this theorem. We use δ to represent the maximal distance between any two round numbers of active processes, $\delta = max\{|r_i - r_j|\}, \forall i, j \in \{1, ..., n\}$ and processes p_i, p_j are active.

Basis: Prior to the first arrival of a message, all the processes are active and in the same round. Thus the theorem trivially holds.

Induction step: By induction, we assume that the theorem holds before some message arrives at some process. That is $\delta <= 1$ so far. When a message arrives at a passive process, it's simply forwarded. Assume a message m with parameters $(id, round, hop, bit)$ arrives at an active process p_i with identity id_i and round number $round_i$. According to the algorithm, there are the following cases.

Case 1: $hop = n$ and $bit = true$. Then p_i becomes the leader, and no process changes its round number. The distance remains unchanged.

Case 2: $hop = n$ and $bit = false$. In this situation, p_i would increase its round number. There are two cases: $\delta = 0$ or $\delta = 1$. For the first case, all the active processes have the same round number r_i. After p_i increases its round number, $\delta = 1$. In the second case, the set of round numbers of active processes before p_i receives the message has two possibilities $r_i, r_i + 1$ or $r_i, r_i - 1$. For the first subcase, when p_i increases its round number to $r_i + 1$, the theorem still holds. We look at the second subcase. When p$_i$ receives its own message again, it means its message with r_i have visited all the processes and made all active processes with round number smaller than r_i passive. Thus at this moment there exists no active processes with round number $r_i - 1$. It implies that the second subcase never occurs.

Case 3: $hop < N$ and $(round, id) = (round_i, id_i)$. Then p_i just dirties the bit and passes on the message. The distance δ remains unchanged.

Case 4: $(round, id) > (round_i, id_i)$. Then p_i would become passive and be eliminated from the set of round numbers of active processes. If $\delta = 0$, all the active processes have the same round r_i. If $\delta = 1$, some active processes are in round r_i and the others in round either $r_i + 1$ or $r_i - 1$. For all these cases, when an active process with r_i becomes passive, the theorem still holds.

Case 5: if $(round, id) < (round_i, id_i)$. Then p_i simply purges the message. The distance δ remains unchanged. ■

Proof of Theorem 2

Proof: We apply induction to prove this theorem. We use δ to represent the maximal distance of round numbers of active processes. It's defined as in the proof of Theorem 1.

BASIS: Prior to the first arrival of a message, all the processes are active and in the same round. Thus the theorem trivially holds.

INDUCTION STEP: By induction, we assume that the theorem holds before some message arrives at some process. That is $\delta <= n-1$ so far. When a message arrives at a passive process, it's simply forwarded. Assume a message m with parameters $(id, round, hop)$ arrives at an active process p_i with identity id_i and round number $round_i$. If $hop = n$, then p_i becomes the leader and its round number remains unchanged. Suppose $hop < n$, we consider the the following cases according to Algorithm 2.

Case 1: $(round, id) = (round_i, id_i)$. Then p_i would increase its round number. By induction, δ is either smaller than $n - 1$ or equal to $n - 1$. If it is smaller than $n - 1$, then with some active process increases its round number by one, δ is either still smaller than $n - 1$ or equal to $n - 1$. Thus, the theorem holds. If δ is equal to $n-1$, then there must exist two different active processes in the $round$ r_{max} and r_{min} respectively, and $r_{max} - r_{min} = n - 1$. There are two subcases: $r_i < r_{max}$ and $r_i = r_{max}$. For the first subcase, after p_i increases its round number from r_i to $r_i + 1$, $r_i + 1$ is maximally equal to r_{max}. δ is still $n - 1$. For the second subcase, if p_i increases its round number by one, then the distance would exceed $n - 1$. However, the second subcase never occurs. We prove it by contradiction. If this subcase exists, it means that p_i with round $r_i = r_{max}$ has just received a message with the same round number and identity. As $hop < n$, this message is from some other process. Thus there must exist some other active process p_j with the same round $r_j = r_{max}$. Suppose active process p_k is in the minimal round r_{min}. Then p_k has never received some message with larger round number than r_{min}, otherwise it would be made passive. Then the next process p_t following the direction of the ring from p_k can only have received messages with round numbers no larger than r_{min}. Thus, the process p_t is either passive or active and maximally in round $r_{min} + 1$. According to the same reason, the following w-th processes are maximally in round $r_{min} + w$ if active. The ring size is n. Thus w is maximally $n - 1$. Therefore, only the $(n - 1)$-th process has the possibility in round $r_{min} + n - 1$ (that is r_{max}). That contradicts with the subcase that another process with r_{max} exists. Therefore, the second case never occurs.

Case 2: $(round, id) > (round_i, id_i)$. Then p_i would become passive and be eliminated from the set of round numbers of active processes. The theorem still holds.

Case 3: $(round, id) < (round_i, id_i)$. Then p_i simply purges the message. The theorem still holds. ■

Proof of Theroem 3

Proof: We prove it by showing that (1) if the distance of round numbers is more than 3, then some process must have decided, and (2) there exists a scenario that the distance is 3 without any process deciding.

Proof of (1). Let p_i, p_j be two different active processes with round numbers k and $k+(3+d)$, respectively ($d \geq 1$). We need to prove that in this situation some process must have decided. The situation implies that p_j has come through round $k + 1$, $k + 2$ and $k + 3$ without the engagement of p_i. As for 1-crash cases, $n - 1$ messages are needed to proceed. It implies that p_j have received all the messages in these rounds from all the processes except for p_i. It means that all the other processes are at least in round $k + 3$ and have come through the rounds $k + 1$ and $k + 2$ by receiving the set of messages from all the processes except p_i. Because they have received the same set of messages, at the end of round $k+1$ all the processes except for p_i must have chosen the same vote. At round $k+2$, all the processes could have only received the same $n - 1$ vote, thus they must have chosen the vote with weight $n - 1$. Therefore, in round $k + 3$, p_j received $n - 1$ same votes with weight $n - 1$, all of which are witnesses. According to the algorithm, p_j must have decided in this round.

Proof of (2). We give a scenario that the distance of round numbers is 3 and no process decides. Let p_i, p_j be two different processes and all processes are in round k. n is an even integer and $n = 2t$. s processes have the value 1 including

p_i and the others have 0. So in round k all processes are able to choose 1 or 0 with weight s according to the set of messages received. p_i stays in round k after sending its vote to all processes. At the end of round k, s processes choose value 0 and the other $s-1$ choose 1, all with weight s. In round $k+1$, all these processes except for p_i receive all these messages with weight s. And they all choose 0 with weight s and have no witness. In round $k+2$, all the processes receive the same $n-1$ messages with the same value 0 and weight s. As no witness appears in this round, they all enter round $k+3$. ■

Proof of Theorem 4

Proof: We will prove it by proving that (1) if the distance is larger than $2n$, then some process must have decided and (2) there exists a scenario that the distance is $2n$ before some process decides.

Proof of (1). Let two active processes p_i, p_j be in the round k and $k+(2n+d)$ respectively and $d \geq 1$. We prove that in this situation, some process must have decided. The situation implies that p_j has come through all the rounds from k to $k+2n$ without the engagement of p_i. As only one process may crash, according to the algorithm in each round the coordinator needs to get $n-1$ messages to proceed. It implies that all the coordinators from $k+1$ to $k+2n$ must have received the messages of all the processes except for p_i. In the algorithm, all the processes take turns to play the coordinator. So from $k+1$ to $k+2n, p_j$ must have played the coordinator twice. Suppose p_j plays the coordinator for the second time in round r_j, then $k+2n \geq r_j \geq k+n+1$. Because $n-1$ messages are needed for the coordinator to proceed, p_j must have received all the messages except p_i in round r_j, which means all the other processes are in round r_j or larger. Then during rounds $k+1$ to $k+n$ they have all played the coordinator once and received the same set of messages without the message from p_i. As they run the same local algorithm, then after round $k+n$ all the other processes except p_i have chosen the same value. Therefore, when p_j played the coordinator for the second time in round r_j which is between $k+n+1$ and $k+2n$, it received $n-1$ messages with the same value and decided in that round. So in the round $k+2n+d$, p_j must have decided.

Proof of (2). We give a scenario that the distance is $2n$ before some process decides. Let p_i, p_j be two different processes with the same value 0. p_i is in the round k and the coordinator of this round is p_j. n is an even integer. Half of the processes have value 0 and the other half have 1. p_j receives $n/2$ messages with the value 0 and $n/2-1$ messages with 1. According to the algorithm p_j keeps its value in round k. From round $k+1$ to $k+(n-1)$, each process except p_j plays coordinator once and chooses 1 as it receives at least $n/2$ messages with 1. In round $k+n$, p_j plays the coordinator again and changes its value to 1. Then p_j proceeds with suspecting all the other processes and increasing its round number until $k+2n$ in which it plays the coordinator again. At this moment, we have two different active processes with round number k and $k+2n$. ■

Proof of Lemma 2

Proof: Suppose some process p_i decides value v at round k. Then according to the algorithm, p_i receives the outcome from the coordinator p_c of this round with bit d set to 1. Other correct processes can either receive the outcome from p_c and decide in this round or suspect p_c and enter next round. And from round $k+1$ to $k+n$, each process will play the coordinator once. When those processes that have not decided so far play the coordinator, they may either receive $N-t$ messages with the same value v and decide or set its value to v (According to Lemma 3, it can receive at most t messages with different value from v and $n-2t$ messages with value v. Considering $t < n/3$, such that $n-2t > t$). At the end of round $k+n$ all processes have the same value v. If there still exist processes that haven't decided, they will decide value v when playing coordinator next time in round between $k+n+1$ and $k+2n$. Therefore, when some process decides, all the other correct processes will decide in maximally $2n$ rounds with the same value. ■

Lemma 3. ***[6, pp.514]*** *If at the beginning of a round $k \geq N-t$ processes have value v, then at least k processes have v at the end of that round.*