Journal of Universal Computer Science, vol. 12, no. 8 (2006), 981-1006
submitted: 22/12/05, accepted: 22/8/06, appeared: 28/8/06 © J.UCS

Variations on Itai-Rodeh Leader Election for Anonymous Rings and their Analysis in PRISM

Wan Fokkink
(Vrije Universiteit, Section Theoretical Computer Science
CWI, Embedded Systems Group
Amsterdam, The Netherlands
wanf@cs.vu.nl)
Jun Pang
(Universität Oldenburg, Safety-Critical Embedded Systems
Oldenburg, Germany
jun.pang@informatik.uni-oldenburg.de)

Abstract: We present two probabilistic leader election algorithms for anonymous unidirectional rings with FIFO channels, based on an algorithm from Itai and Rodeh [Itai and Rodeh 1981]. In contrast to the Itai-Rodeh algorithm, our algorithms are finite-state. So they can be analyzed using explicit state space exploration; we used the probabilistic model checker PRISM to verify, for rings up to size four, that eventually a unique leader is elected with probability one. Furthermore, we give a manual correctness proof for each algorithm.

Key Words: distributed computing, leader election, anonymous networks, probabilistic algorithms, formal verification, model checking
Category: C.2.4, D.2.4, F.3.1

1 Introduction

Leader election is the problem of electing a unique leader in a network, in the sense that the leader (process) knows that it has been elected and the other processes know that they have not been elected. Leader election algorithms require that all processes have the same local algorithm and that each computation terminates, with one process elected as the leader. This is a fundamental problem in distributed computing and has numerous applications. For example, it is an important tool for breaking symmetry in a distributed system. By choosing a process as the leader it is possible to execute centralized protocols in a decentralized environment. Leader election can also be used to recover from token loss for token-based protocols, by making the leader responsible for generating a new token when the current one is lost.

There exists a broad range of leader election algorithms; see e.g. the summary in the text books [Tel 1994, Lynch 1996]. These algorithms have different message complexity in the worst and/or average case. Furthermore, they vary in communication mechanism (*asynchronous vs. synchronous*), process names (*unique identities vs. anonymous*), and network topology (e.g. *ring, tree, complete graph*).

A first leader election algorithm for unidirectional rings was given by Le Lann [Le Lann 1977]. It requires that each process has a unique identity, with a total ordering on identities; the process with the largest identity becomes the leader. The basic idea of Le Lann's algorithm is that each process sends a message around the ring bearing its identity. Thus it requires a total of n^2 messages, where n is the number of processes in the ring. Chang and Roberts [Chang and Roberts 1979] improved Le Lann's algorithm by letting only the message with the largest identity complete the round trip; their algorithm still requires in the order of n^2 messages in the worst case, but only $n \log n$ on average. Franklin [Franklin 1982] developed a leader election algorithm for bidirectional rings with a worst-case message complexity of $\mathcal{O}(n \log n)$. Peterson [Peterson 1982] and Dolev, Klawe, and Rodeh [Dolev et al. 1982] independently adapted Franklin's algorithm so that it also works for unidirectional rings. All the above algorithms work both for asynchronous and for synchronous communication, and do not require a priori knowledge about the number of processes.

Sometimes the processes in a network cannot be distinguished by means of unique identities. First, as the number of processes in a network increases, it may become difficult to keep the identities of all processes distinct; or a network may accidentally assign the same identity to different processes. Second, identities cannot always be sent around the network, for instance for reasons of efficiency. An example of the latter is FireWire, the IEEE 1394 high performance serial bus (see Section 6 for a more detailed description). A leader election algorithm that works in the absence of unique process identities is also desirable from the standpoint of fault tolerance. In an *anonymous network*, processes do not carry an identity. Angluin [Angluin 1980] showed that there does not exist a terminating algorithm for electing a leader in an asynchronous anonymous network.

In [Itai and Rodeh 1981, Itai and Rodeh 1990] a probabilistic leader election algorithm for anonymous unidirectional rings is proposed, based on the Chang-Roberts algorithm. Each process selects a random identity from a finite domain, and processes with the largest identity start a new election round if they detect a name clash. It is assumed that the size of the ring is known to all processes,[1] algorithm exists to elect a leader in so that each process can recognize its own message (by means of a hop counter that is part of the message). The Itai-Rodeh algorithm terminates with probability one (it exhibits infinite traces, but the probability that such an infinite trace is executed is zero), and all its terminal states are correct, meaning that exactly one leader is elected; its average-case message complexity is $\Theta(n \log n)$.

[1] Given an anonymous ring of which the size is unknown to the processes, no (weakly terminating) algorithm exists for which in all terminal states the correct ring size has been computed, see e.g. [Tel 1994].

The Itai-Rodeh algorithm makes no assumptions about channel behavior, except fair scheduling. 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 (see Fig. 1 in Section 2.2). In order to avoid this problem, the algorithm proceeds in successive rounds, and each process and message is supplied with a round number. 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.

In this paper, we make the assumption that channels are FIFO. We show that in this case round numbers can be omitted from the Itai-Rodeh algorithm. We present two adaptations of the Itai-Rodeh algorithm, that are correct in the presence of FIFO channels. In the first algorithm, a process may only choose a new identity when its message has completed the round trip, as is the case in the Itai-Rodeh algorithm. In the second algorithm, a process selects a new identity as soon as it detects that another process in the ring carries the same identity (even though this identity may not be the largest one in the ring). Since both algorithms do not use round numbers, they are finite-state. This means that we can apply model checking [Clarke et al. 2000] to automatically verify properties of an algorithm, specified in some temporal logic. These properties can be checked against the explicit (finite) state space of the algorithm, for specific ring sizes. We used PRISM [Kwiatkowska et al. 2002], a probabilistic model checker that can be used to model and analyze systems containing probabilistic aspects. We specified both algorithms in the PRISM language, and for rings up to size four we verified the property: "with probability one, eventually exactly one leader is elected". Furthermore, we present a manual correctness proof for both algorithms, for arbitrary ring size.

PRISM offers the possibility to calculate the probability that our algorithms have terminated after some number of messages. These statistics show that the first algorithm on average requires more messages to terminate than the second algorithm.

1.1 Outline of the paper

Section 2 contains the original Itai-Rodeh algorithm. In Sections 3 and 4, we present two probabilistic leader election algorithms for anonymous rings with FIFO channels. We explain our verification results with PRISM, and give a manual correctness proof for each algorithm. Section 5 reveals some experimental results using PRISM on the number of messages needed to terminate. Related work is summarized in Section 6. We conclude this paper and discuss some future work in Section 7.

2 Itai-Rodeh Leader Election

We consider an *asynchronous*, *anonymous*, *unidirectional* ring consisting of $n \geq 2$ processes $p_0, \ldots, p_{n-1}$. Processes communicate asynchronously by sending and receiving messages over channels, which are assumed to be reliable, and have capacity n. Channels are unidirectional: a message sent by p_i is added to the message queue of $p_{(i+1) \bmod n}$. It is assumed that receiving a message, processing it, and possibly sending a subsequent message take zero time (i.e., are instantaneous). The message queues are guided by a *fair scheduler*, meaning that in each infinite execution sequence, every sent message eventually arrives at its destination. Processes are anonymous, so they do not have unique identities. The challenge is to present a uniform local algorithm for each process, such that one leader is elected among the processes.

2.1 The Itai-Rodeh algorithm

In [Itai and Rodeh 1981, Itai and Rodeh 1990] it is studied how to break the symmetry in anonymous networks using probabilistic algorithms. They presented a probabilistic algorithm to elect a leader in the above network model, under the assumption that processes know that the size of the ring is n. It terminates with probability one, and all its terminal states are correct, meaning that exactly one leader is elected. The Itai-Rodeh algorithm is based on the Chang-Roberts algorithm [Chang and Roberts 1979], where processes are assumed to have unique identities, and each process sends out a message carrying its identity. Only the message with the largest identity completes the round trip and returns to its originator, which becomes the leader.

In the Itai-Rodeh algorithm, each process selects a *random identity* from a finite set. So different processes may carry the same identity. Again each process sends out a message carrying its identity. Messages are supplied with a *hop counter*, so that a process can recognize its own message (by checking whether the hop counter equals the ring size n). Moreover, a process with the largest identity present in the ring must be able to detect whether there are other processes in the ring with the same identity. Therefore each message is supplied with a bit, which is dirtied when it passes a process that is not its originator but shares the same identity. When a process receives its own message, either it becomes the leader (if the bit is clean), or it selects a new identity and starts the next election round (if the bit is dirty). In this next election round, only processes that shared the largest identity in the ring are *active*. All other processes have been made *passive* by the receipt of a message with an identity larger than their own. The active processes maintain a *round number*, which initially starts at zero and is augmented at each new election round. Thus messages from earlier election rounds can be recognized and ignored.

We proceed to present a detailed description of the Itai-Rodeh algorithm. Each process p_i maintains three parameters:

- $id_i \in \{1, \ldots, k\}$, for some $k \geq 2$, is its identity;
- $state_i$ ranges over $\{active, passive, leader\}$;
- $round_i \in \mathbb{N}^+$ represents the number of the current election round.

Only active processes may become the leader; passive processes simply pass on messages. At the start of a new election round, each active process sends a message of the form $(id, round, hop, bit)$, where:

- the values of *id* and *round* are taken from the process that sends the message;
- *hop* is a counter that initially has the value one, and which is increased by one every time it is passed on by a process;
- *bit* is a bit that initially is *true*, and which is set to *false* when it visits a process that has the same identity but that is not its originator.

We will refer to a message traveling through the ring with the letter m. The parameters *id* and *round* of a message stay unchanged during a round trip, while its other two parameters may change.

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 one of 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.

We say that an execution sequence of the Itai-Rodeh algorithm has *terminated* if each process is either passive or elected as the leader, and there are no remaining messages in the channels.

Theorem 1. *[Itai and Rodeh 1981] The Itai-Rodeh algorithm terminates with probability one, and upon termination a unique leader has been elected.*

2.2 Round numbers are needed

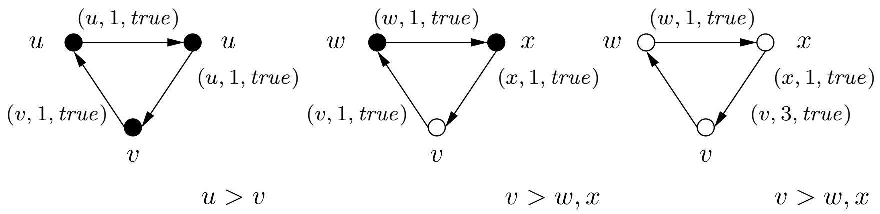

Figure 1: Round numbers are essential if channels are not FIFO

Fig. 1 presents a scenario to show that if round numbers were omitted, the Itai-Rodeh algorithm could produce an execution sequence in which all processes become passive, so that no leader is elected. This example uses the fact that channels are not FIFO. Let $k \geq 3$. Fig. 1 depicts a ring of size three; black processes are active and white processes are passive. Initially, all processes are active, and the two processes above select the same identity u, while the one below selects an identity $v < u$. (See the left side of Fig. 1.) The three processes send a message with their identity, and at the receipt of a message with identity u, process v becomes passive. Since channels are not FIFO, the message $(v, 1, \mathit{true})$ can be overtaken by the other two messages with identity u. The latter two messages return to their originators with a dirty bit. So the processes with identity u detect a name clash, select new identities $w < v$ and $x < v$, and send messages carrying these identities. (See the middle part of Fig. 1.) Finally, the message with identity v makes the processes with identities w and x passive. The three messages in the ring are passed on forever by the three passive processes. (See the right side of Fig. 1.)

3 Leader Election without Round Numbers

We observe that if channels are FIFO, round numbers are redundant. Thus we obtain a simplification of the Itai-Rodeh algorithm. Algorithm $\mathcal{A}$ is obtained by considering only those cases in the Itai-Rodeh algorithm where the active process p_i and the incoming message have the same round number. Correctness of Algorithm $\mathcal{A}$ follows from the proposition below.

Algorithm $\mathcal{A}$.

- Initially, all processes are active, and each process p_i randomly selects its identity $id_i \in \{1, \ldots, k\}$ and sends the message $(id_i, 1, \mathit{true})$.
- Upon receipt of a message (id, hop, bit), a passive process p_i ($state_i = \mathit{passive}$) passes on the message $(id, hop + 1, bit)$; an active process p_i ($state_i = \mathit{active}$) behaves according to one of the following steps:
  - if $hop = n$ and $bit = \mathit{true}$, then p_i becomes the leader ($state'_i = \mathit{leader}$);
  - if $hop = n$ and $bit = \mathit{false}$, then p_i selects a new random identity $id'_i \in \{1, \ldots, k\}$ and sends the message $(id'_i, 1, \mathit{true})$;
  - if $id = id_i$ and $hop < n$, then p_i passes on the message $(id, hop + 1, \mathit{false})$;
  - if $id > id_i$, then p_i becomes passive ($state'_i = \mathit{passive}$) and passes on the message $(id, hop + 1, bit)$;
  - if $id < id_i$, then p_i purges the incoming message.

Proposition 2. *Consider the Itai-Rodeh algorithm where all channels are FIFO. When an active process receives a message, then the round number of the process and of the message are always the same.*

Proof. Let message m, which originates from process p_j, arrive at active process p_i in the form $(id_j, round_j, hop, bit)$. Suppose that up to this moment, messages never arrived at active processes with a different round number. We prove that the round number $round_i$ of p_i is equal to $round_j$. We derive the desired equality in two steps.

- $round_i \leq round_j$.

  Let $round_i > 1$, for else we are done. Then a message m' with round number $round_i{-}1$ originated at p_i and completed the round trip, where all the active processes that it visited had round number $round_i{-}1$. FIFO behavior guarantees that after m' returned to p_i, no other message with round number $\leq round_i{-}1$ can arrive at p_i. So $round_i \leq round_j$.

- $round_i \geq round_j$.

  Let $round_j > 1$, for else we are done. Then a message m'' with round number $round_j{-}1$ originated at p_j and completed the round trip, where all the active processes that it visited (so in particular p_i) had round number $round_j-1$. Since m'' completed the round trip and passed p_i while this process remained active, it follows that both p_i and p_j had the maximal identity in round $round_j-1$. And by the induction hypothesis, the message m''' that originated at p_i with round number $round_j-1$ did not meet any active process with a round number $\geq round_j$. So m''' was not purged by any active process. FIFO behavior guarantees that m''' arrived at p_j before m'', so that m''' passed p_j before m was created at p_j. FIFO behavior also guarantees that m''' arrived at p_i before m. So $round_i \geq round_j$.

Hence, $round_i = round_j$. □

Theorem 3. *Let channels be FIFO. Then Algorithm $\mathcal{A}$ terminates with probability one, and upon termination exactly one leader is elected.*

Proof. By Proposition 2 and Theorem 1. □

3.1 Automated verification with PRISM

Owing to the elimination of round numbers, Algorithm $\mathcal{A}$ is finite-state, contrary to the Itai-Rodeh algorithm. Hence we can apply explicit state space generation and model checking to establish the correctness of Algorithm $\mathcal{A}$ for fixed ring sizes. This analysis of Algorithm $\mathcal{A}$ was actually performed before constructing the manual correctness proof of Algorithm $\mathcal{A}$ from the previous section, as a means to confirm our intuition that Algorithm $\mathcal{A}$ works correctly in case of FIFO channels. Moreover, this model checking exercise has some additional value compared to Theorem 3. Namely, since the manual proofs of Theorem 1, Proposition 2 and Theorem 3 were not formalized and checked with a theorem prover, there is no absolute guarantee that they are free of flaws.

3.1.1 A short introduction to PRISM

PRISM [Kwiatkowska et al. 2002] is a probabilistic model checker, being developed at the University of Birmingham. It allows one to model and analyze systems and algorithms containing probabilistic aspects. PRISM supports three kinds of probabilistic models: discrete-time Markov chains (DTMCs), Markov decision processes (MDPs) and continuous-time Markov chains (CTMCs). Analysis is performed through model checking such systems against specifications written in the probabilistic temporal logic PCTL [Hansson and Jonsson 1994, Baier and Kwiatkowska 1998] if the model is a DTMC or an MDP, or CSL [Baier et al. 2000] in the case of a CTMC.

In order to model check probabilistic properties of Algorithm $\mathcal{A}$, we first encoded the algorithm as an MDP model using the PRISM language, which is a simple, state-based language, based on the Reactive Modules formalism of Alur and Henzinger [Alur and Henzinger 1999]. A system is composed of a number of modules that contain local variables, and that can interact with each other. The behavior of an MDP is described by a set of commands of the form:

$$[a]\ g \rightarrow \lambda_1 : u_1 + \ldots + \lambda_\ell : u_\ell$$

a is an action label in the style of process algebras, which introduces synchronization into the model. It can only be performed simultaneously by all modules that have an occurrence of action label a in their specification. If a transition does not have to synchronize with other transitions, then no action label needs to be provided for this transition. The symbol g is a predicate over all the variables in the system. Each u_i describes a transition which the module can make if g is true. A transition updates the value of the variables by giving their new *primed* value with respect to their *unprimed* value. The λ_i are used to assign probabilistic information to the transition. It is required that $\lambda_1 + \cdots + \lambda_\ell = 1$. This probabilistic information can be omitted if $\ell = 1$ (and so $\lambda_1 = 1$). PRISM considers states without outgoing transitions as error states; terminating states can be modeled by adding a self-loop. PRISM models which are MDPs can also exhibit *local non-determinism*, which allows the moduels to make non-deterministic choices themselves. For example, the probabilistic choice in the previous command can be made non-deterministic as follows:

$[a]\ g \rightarrow u_1;$
...
$[a]\ g \rightarrow u_\ell;$

A more detailed description of PRISM can be found in [PRISM].

3.1.2 Verifying Algorithm $\mathcal{A}$ with PRISM

We used PRISM to verify that Algorithm $\mathcal{A}$ satisfies the probabilistic property "with probability 1, eventually exactly one leader is elected". We modeled each FIFO channel and each process as a separate module in PRISM. The following code in the PRISM language gives the specification for a channel of size two. The channel *channel1* receives a message (mes1_id,mes1_counter,mes1_bit) from process p_1 (synchronized on action label rec_from_p1) and sends it to process p_2 (synchronized on action label send_to_p2). Each position $\mathsf{i} \in \{1, 2\}$ in the channel is represented by a triple of natural numbers: one for the process identity contained in a message (b_1_2_i1), one for the hop counter (b_1_2_i2), and one for the bit (b_1_2_i3). If the natural numbers for a position in a channel are greater than zero, it means this position is occupied by a message. Otherwise, the position is empty.

We present the channel between processes p_1 and p_2. Both the number of processes and the size of the identity set are two (N=2; K=2).

```
module channel1
    b_1_2_11: [0..K]; b_1_2_12:[0..N]; b_1_2_13:[0..1];
    b_1_2_21: [0..K]; b_1_2_22:[0..N]; b_1_2_13:[0..1];
    [rec_from_p1] b_1_2_11=0
            → (b_1_2_11'=mes1_id) & (b_1_2_12'=mes1_counter) &
              (b_1_2_13'=mes1_bit);
    [rec_from_p1] (b_1_2_11>0) & (b_1_2_21=0)
            → (b_1_2_21'=mes1_id) & (b_1_2_22'=mes1_counter) &
              (b_1_2_23'=mes1_bit);
    [send_to_p2] b_1_2_11>0
            → (b_1_2_11'=b_1_2_21) & (b_1_2_12'=b_1_2_22) &
              (b_1_2_13'=b_1_2_23) & (b_1_2_21'=0) &
              (b_1_2_22'=0) & (b_1_2_23'=0);
endmodule
```

mes1_id, mes1_counter and mes1_bit are *shared* variables. They are used in the module process1 below for receiving and sending messages. Only in that module values can be assigned to these variables. mes1_id carries the identity of a message, mes1_counter its hop counter, and mes1_bit the clean (1) or dirty (0) bit. If no message is present, all three variables have the value zero. (So mes1_bit=0 can have two meanings: either there is no message, or the bit is dirty.)

Each process p_i is specified by means of a variable processi_id:[0..K] for its identity (where 0 means that the process is passive or selecting a new identity), a variable si:[0..5] for its local state (this is explained below), and a variable leaderi:[0..1] (where in state 0 means that the process is passive, and 1 that it is the leader). The following PRISM code is the specification for process p_1.

```
module process1
    process1_id:[0..K]; s1:[0..5]; leader1:[0..1];
    mes1_id:[0..K]; mes1_counter:[0..N]; mes1_bit:[0..1];
```

When a process is in state 0, it is active and can randomly (modeled by the probability rate R=1/K) select its identity, build a new message with this identity, and set its state to 1.

```
    [ ] s1=0
        → R: (s1'=1) & (process1_id'=1) & (mes1_id'=1) &
          (mes1_counter'=1) & (mes1_bit'=1)
        + R: (s1'=1) & (process1_id'=2) & (mes1_id'=2) &
          (mes1_counter'=1) & (mes1_bit'=1);
```

When s1=1, the process sends the new message into channel 1 (modeled by a synchronization with module channel1 on action rec_from_p1), and moves to state 2.

```
[rec_from_p1] s1=1
      → (s1'=2) & (mes1_id'=0) & (mes1_counter'=0) &
        (mes1_bit'=0);
```

In state 2 the process can receive a message from channel 2 (modeled by a synchronization with module channel2 on action send_to_p1), and go to state 3. Note that b_2_1_11, b_2_1_12 and b_2_1_31 are shared variables, representing the first position in the module channel2.

```
[send_to_p1] s1=2
      → (s1'=3) & (mes1_id'=b_2_1_11) &
        (mes1_counter'=b_2_1_12) & (mes1_bit'=b_2_1_13);
```

When a process is in state 3, it has received a message and takes a decision. If the process got its own message back (mes1_counter=N) and the bit of the message is clean (mes1_bit=1), the process is elected as the leader (leader1'=1), and moves to state 4.

```
[ ] (s1=3) & (mes1_counter=N) & (mes1_bit=1)
      → (s1'=4) & (process1_id'=0) & (mes1_id'=0) &
        (mes1_counter'=0) & (mes1_bit'=0) & (leader1'=1);
```

If mes1_counter=N and mes1_bit=0, the process changes its state to 0 and will select a new random identity.

```
[ ] (s1=3) & (mes1_counter=N) & (mes1_bit=0)
      → (s1'=0) & (process1_id'=0) & (mes1_id'=0) &
        (mes1_counter'=0) & (mes1_bit'=0);
```

If mes1_id=process1_id and mes1_counter<N, the process has received a message with the same identity, but the message does not originate from itself. It increases the hop counter in the message by one, makes the bit dirty, and moves to state 5 to pass on the message.

```
[ ] (s1=3) & (mes1_id=process1_id) & (mes1_counter<N)
      → (s1'=5) & (mes1_counter'=mes1_counter+1) &
        (mes1_bit'=0);
```

If mes1_id<process1_id, the process purges the message, and moves back to state 2 to receive another message.

```
[ ] (s1=3) & (mes1_id<process1_id)
      → (s1'=2) & (mes1_id'=0) & (mes1_counter'=0) &
        (mes1_bit'=0);
```

If mes1_id>process1_id, the process increases the hop counter in the message by one, and goes to state 4 where it becomes passive (i.e., the value of leader1 remains zero).

```
[ ] (s1=3) & (mes1_id>process1_id)
    → (s1'=4) & (process1_id'=0) &
      (mes1_counter'=mes1_counter+1);
```

In state 5, a process passes on a message, and moves to state 2.

```
[rec_from_p1] (s1=5)
    → (s1'=2) & (mes1_id'=0) & (mes1_counter'=0) &
      (mes1_bit'=0);
```

In state 4, a passive process (leader1=0) can only pass on messages with their hop counter increased by one.

```
[send_to_p1] (s1=4) & (leader1=0) & (mes1_id=0)
    → (mes1_id'=b_2_1_11) & (mes1_counter'=b_2_1_12+1) &
      (mes1_bit'=b_2_1_13);
[rec_from_p1] (s1=4) & (leader1=0) & (mes1_id>0)
    → (mes1_id'=0) & (mes1_counter'=0) & (mes1_bit'=0);
```

We added the conjunct leader1=0 to the predicate in order to emphasize that the leader does not have to deal with incoming messages. Namely, when a process is elected as the leader there are no remaining messages, owing to the fact that channels are FIFO.

A self-loop with synchronization on an action label done is added to processes in state 4, to avoid deadlock states.

```
    [done] (s1=4) → (s1'=s1);
endmodule
```

Other channels and processes can be constructed by carefully *module renaming* modules channel1 and process1. The initial value of each variable is the minimal value in its range.

Below we specify the property "with probability 1, eventually exactly one leader is elected" for a ring with two processes as a PCTL formula:

Property: P>=1 [true U (s1=4 & s2=4 & leader1+leader2=1 &
b_1_2_11+b_2_1_11=0)]

It states that the probability that ultimately both p_1 and p_2 get into state 4 (s1=4 & s2=4), with exactly one process elected as the leader (leader1+leader2=1), is

	Processes	Identities	Channel size	FIFO	States	Transitions
Ex.1	2	2	2	yes	127	216
Ex.2	3	3	3	yes	5,467	12,360
Ex.3	4	3	4	yes	99,329	283,872

Table 1: Model checking result for Algorithm $\mathcal{A}$ with FIFO channels

at least one. In addition, we check that the algorithm terminates with no message in the ring (b_1_2_11+b_2_1_11=0).

Note that, for MDPs, since probabilities can only computed once the nondeterministic choices have been resolved. Hence, there is actually a *minimum* and a *maximum* probability of a formula being satisfied, quantifying over all possible resolutions. Therefore, for MDPs PRISM allows two possible types of formula: $P_{max>=1}[\ldots]$ and $P_{min>=1}[\ldots]$ for the above property, which return the maximum and minimum probabilities, respectively. Our analysis of both Algorithms $\mathcal{A}$ and $\mathcal{B}$ in PRISM showed that nondeterminism does not make difference for the computation of the maximum and minimum probabilities of the PCTL formulas we want to check, namely for all the formulas we have checked in this paper, their minimum probability is equivalent to their maximum probability. Thus, we do not make distinction between the maximum and minimum probabilities in our presentation of the analysis results in PRISM (see Sections 4.1 and 5).

To model check this property, the algorithmic description (in the module-based language) was parsed and converted into an MTBDD [Fujita et al. 1997]. In PRISM, reachability is performed to identify non-reachable states and the MTBDD is filtered accordingly. Table 1 shows statistics for each model we have built. The first part gives the parameters for each model: the ring size n, the size of the identity set, and the size of the channel. It is not hard to see that at any time there are at most n messages in the ring, so channel size n suffices; and having n different possible identities means that in each "round", all active processes can select a different identity. The second part gives the number of states and transitions in the MTBDD representing the model.

Property was successfully checked on all the ring networks in Table 1 (we used the model checker PRISM 2.0 with its default options). Note that for $n = 4$, we could only check the property for an identity set of size three. For $n = 4$ and an identity set of size four, and in general for $n \geq 5$, PRISM fails to build a model due to the lack of memory.

4 Leader Election without Bits

Algorithm $\mathcal{B}$.

- Initially, all processes are active, and each process p_i randomly selects its identity $id_i \in \{1, \ldots, k\}$ and sends the message $(id_i, 1)$.
- Upon receipt of a message (id, hop), a passive process p_i $(state_i = passive)$ passes on the message $(id, hop + 1)$; an active process p_i $(state_i = active)$ behaves according to one of the following steps:
  - if $hop = n$, then p_i becomes the leader $(state'_i = leader)$;
  - if $id = id_i$ and $hop < n$, then p_i selects a new random identity $id'_i \in \{1, \ldots, k\}$ and sends the message $(id'_i, 1)$;
  - if $id > id_i$, then p_i becomes passive $(state'_i = passive)$ and passes on the message $(id, hop + 1)$;
  - if $id < id_i$, then p_i purges the incoming message.

In this section, we present another leader election algorithm, which is a variation of Algorithm $\mathcal{A}$. Again channels are assumed to be FIFO. We observe that when an active process p_i detects a name clash, meaning that it receives a message with its own identity and hop counter smaller than n, it is not necessary for p_i to wait for its own message to return. Instead p_i can immediately select a new random identity and send a new message. Algorithm $\mathcal{B}$ is obtained by adapting Algorithm $\mathcal{A}$ according to this observation. In particular all occurrences of bits are omitted.

We first discuss the automatic verification of Algorithm $\mathcal{B}$ with PRISM in Section 4.1. Then we give a manual correctness proof for Algorithm $\mathcal{B}$, for arbitrary ring size, in Section 4.2.

4.1 Automated verification with PRISM

Channels are modeled in the same way as in Section 3. We present each process p_i with a variable `process i_id:[0..K]` for its identity, a variable `s i:[0..4]` for its local state, and a variable `leader i:[0..1]`. We present only part of the PRISM specification for process p_1. The parts when a process is in state `0`, `1`, `2` or `4` are omitted, as this behavior is very similar to Algorithm $\mathcal{A}$ (see Section 3.1). State `5` is redundant here, because a process selects a new identity as soon as it detects a name clash.

```
module process1
    process1_id:[0..K]; s1:[0..4]; leader1:[0..1]; mes1_id:[0..K];
```

```
mes1_counter:[0..N];
```

When a process in state 3, it has received a message from the channel and takes a decision. If mes1_counter=N, the process is elected as the leader (leader1'=1), and moves to state 4.

```
[ ] (s1=3) & (mes1_counter=N)
      → (s1'=4) & (process1_id'=0) & (mes1_id'=0) &
        (mes1_counter'=0) & (leader1'=1);
```

If mes1_id=process1_id and mes1_counter<N, the process goes back to state 0 and will select a new identity.

```
[ ] (s1=3) & (mes1_id=process1_id) & (mes1_counter<N)
      → (s1'=0) & (mes1_id'=0) & (mes1_counter'=0) &
        (process1_id'=0);
```

If mes1_id<process1_id, the process purges the message, and moves back to state 2 to receive another message.

```
[ ] (s1=3) & (mes1_id<process1_id)
      → (s1'=2) & (mes1_id'=0) & (mes1_counter'=0);
```

If mes1_id>process1_id, the process becomes passive, increases the hop counter of the message by one, and goes to state 4.

```
    [ ] (s1=3) & (mes1_id>process1_id)
          → (s1'=4) & (process1_id'=0) &
            (mes1_counter'=mes1_counter+1);
    ...
endmodule
```

Other channels and processes can be constructed by module renaming.

Property was successfully model checked with respect to Algorithm $\mathcal{B}$, in a setting with FIFO channels, for rings up to size five. For any larger ring size, and in case of ring size five and an identity domain containing three elements, PRISM fails to produce an MTBDD. Table 2 summarizes the verification results for Algorithm $\mathcal{B}$ with PRISM.

4.2 The correctness proof

In this section we give a correctness proof for Algorithm $\mathcal{B}$, in case of FIFO channels, with respect to ring networks of arbitrary size. Intuitively, the processes and messages *between* a process p and a message m are the ones that are encountered when traveling in the ring from p to m. This notion is inductively defines as follows.

	Processes	Identities	Channel size	FIFO	States	Transitions
Ex.1	2	2	2	yes	97	168
Ex.2	3	3	3	yes	6,019	14,115
Ex.3	4	4	4	yes	537,467	1,615,408
Ex.4	5	2	5	yes	752,047	2,626,405

Table 2: Model checking result for Algorithm $\mathcal{B}$ with FIFO channels

Definition 4. Consider a state of Algorithm $\mathcal{B}$. If a message m is in the channel pq from process p to process q, then the messages between p and m are the ones that were sent by p after it sent m. If m is not in the channel pq, then (1) the messages in the channel pq, (2) the process q, and (3) the processes and messages between q and m are all between p and m.

Lemma 5. *Consider a reachable state of Algorithm $\mathcal{B}$. Let active process p have identity id_p and message m have identity id_m. If $id_p \neq id_m$, then there is an active process or message between p and m with an identity $\geq \min\{id_p, id_m\}$.*

Proof. We apply induction on the minimal number of transitions needed to reach this state from an initial state.
Basis: Prior to the first arrival of a message, every process is active and has generated a message with its own identity; thus the lemma trivially holds.
Induction step: When a message arrives at a passive process, it is simply forwarded. Assume a message m with parameters (id, hop) arrives at an active process p_i with identity id_i. If $hop = n$, then p_i is elected as the leader. Since channels are FIFO, in this case the round trip of the final message of p_i guarantees that there are no remaining messages; thus the lemma trivially holds. Now suppose that $hop < n$. We consider three cases. In each case we only consider each pair of an active process and a message that could violate the condition of the lemma due to the arrival of m at p_i.

– $id_i > id$. Then m is purged by p_i.

  Let p_j be an active process with identity id_j and m' a message with identity id', such that p_i and m are between p_j and m', and $id \geq \min\{id_j, id'\}$. The active process p_i between p_j and m' has identity $id_i > \min\{id_j, id'\}$.

– $id_i < id$. Then p_i becomes passive and sends the message $(id, hop + 1)$.

  Let p_j be an active process with identity id_j and m' a message with identity id', such that p_i and m are between p_j and m', and $id_i \geq \min\{id_j, id'\}$. The message $(id, hop + 1)$ between p_j and m' has identity $id > \min\{id_j, id'\}$.

– $id_i = id$. Then p_i selects a new identity id'_i and sends the message $(id'_i, 1)$.

We consider three cases, covering each pair of an active process and a message with different identities that is either newly created (the first two cases) or that could violate the condition of the lemma due to the new identity of p_i (the third case).

- For any message m' with identity $id' \neq id'_i$, $(id'_i, 1)$ is a message between p_i and m' with identity $id'_i \geq min\{id'_i, id'\}$.
- For any active process p_j with identity $id_j \neq id'_i$, p_i is an active process between p_j and $(id'_i, 1)$ with identity $id'_i \geq min\{id_j, id'_i\}$.
- Let p_j be an active process with identity id_j and m' a message with identity $id' \neq id_j$, such that p_i and m are between p_j and m', and $id_i \geq \min\{id_j, id'\}$. Since $id' \neq id_j$, either $id_j \neq id_i$ or $id_i \neq id'$. So by induction there is an active process or message either between p_j and m with an identity $\geq \min\{id_j, id_i\}$, or between p_i and m' with an identity $\geq \min\{id_i, id'\}$. Since $id_i \geq \min\{id_j, id'\}$, in either case there is an active process or message between p_j and m' with an identity $\geq \min\{id_j, id'\}$. □

Definition 6. Consider a state of Algorithm $\mathcal{B}$. An active process p is *related to* a message m if they have the same identity id, and all active processes and messages between p and m have an identity smaller than id.

Lemma 7. *Consider a reachable state of Algorithm $\mathcal{B}$. Let active process p be related to message m. Let ξ be the maximum of all identities of active processes and messages between p and m ($\xi = 0$ if there are none).*

1. *Between p and m, there is an equal number of active processes and of messages with identity ξ; and*
2. *if p is not the originator of m, then there is an active process or message between p and m.*

Proof. We apply induction on the minimal number of transitions needed to reach this state from an initial state.
Basis: Prior to the first arrival of a message, every process is active and has generated a message with its own identity; thus the lemma trivially holds.
Induction step: When a message arrives at a passive process, it is simply forwarded. Assume a message m with parameters (id, hop) arrives at an active process p_i with identity id_i. If $hop = n$, then p_i is elected as the leader. Since channels are FIFO, in this case the round trip of the final message of p_i guarantees that there are no remaining messages; thus the lemma trivially holds. Now suppose that $hop < n$. We consider three cases. In each of these cases we only

consider related pairs that were either created or affected by the arrival of m at p_i.

– $id_i > id$. Then m is purged by p_i.

  Let p_i be between an active process p_j and a message m'. Clearly, id is not the maximal identity of active processes and messages between p_j and m'. So if p_j and m' are related after the purging of m, they were also related before this moment. Hence, by induction, the pair p_j and m' satisfies condition 1 of the lemma. Furthermore, p_i is an active process between p_j and m', so the pair also satisfies condition 2.

– $id_i < id$. Then p becomes passive and sends the message $(id, hop + 1)$.

  If an active process p' is related to $(id, hop+1)$, then clearly it was also related to m. So by induction the pair p' and $(id, hop + 1)$ satisfies conditions 1 and 2.

  Let p_i and $(id, hop + 1)$ be between an active process p_j and a message m'. Clearly, id_i is not the maximal identity of active processes and messages between p_j and m'. So if p_j and m' are related after p_i has become passive, they were also related before this moment. Hence, by induction, the pair p_j and m' satisfies condition 1 of the lemma. Furthermore, $(id, hop + 1)$ is a message between p_j and m', so the pair also satisfies condition 2.

– $id_i = id$. Then p_i selects a new identity id'_i and sends the message $(id'_i, 1)$.

  Note that p_i is the only active process related to $(id'_i, 1)$, and vice versa. Clearly, conditions 1 and 2 of the lemma are satisfied by this pair.

  Let an active process p_j with identity id_j be related to a message m', such that p_i and $(id'_i, 1)$ are between p_j and m'. Since p_i is between p_j and m', condition 2 is satisfied by this pair. We proceed to prove condition 1 for this pair. We consider three cases.

  • $id_i > id_j$.

    Then by Lemma 5 there is an active process or message between p_i and m' with identity $\geq id_j$. This active process or message is also between p_j and m', which contradicts the fact that p_j is related to m'.

  • $id_i < id_j$.

    Then p_j and m' were already related before m reached p_i, so by induction this pair satisfied condition 1 before m reached p_i. Let ξ denote the maximum of all identities of active processes (and of messages) between p_j and m' before m reached p_i; and let # denote the number of active processes (and of messages) between p_j and m' with identity ξ before m reached p_i. Moreover, let ξ'_π and ξ'_μ denote the maximum of all identities of active processes and messages, respectively, between p_j and m' after m reached p_i; and let $\#'_\pi$ and $\#'_\mu$ denote the number of active processes and messages, respectively, between p_j and m' with identity ξ'_π and ξ'_μ, respectively, after m reached p_i. Clearly $id_i \leq \xi$. We consider five cases.

If $id'_i > \xi$, then $\xi'_\pi = id'_i = \xi'_\mu$ and $\#'_\pi = 1 = \#'_\mu$.

If $id'_i = \xi$ and $id_i = \xi$, then $\xi'_\pi = \xi = \xi'_\mu$ and $\#'_\pi = \# = \#'_\mu$.

If $id'_i = \xi$ and $id_i < \xi$, then $\xi'_\pi = \xi = \xi'_\mu$ and $\#'_\pi = \# + 1 = \#'_\mu$.

If $id'_i < \xi$ and $id_i = \xi$, then $\xi'_\pi = \xi = \xi'_\mu$ and $\#'_\pi = \# - 1 = \#'_\mu$. Namely, since $id_i < id_j$, by Lemma 5 there must be an active process or message between p_i and m' with identity $\geq id_i$. Since $id_i = \xi$, this identity must be equal to id_i.

If $id'_i < \xi$ and $id_i < \xi$, then $\xi'_\pi = \xi = \xi'_\mu$ and $\#'_\pi = \# = \#'_\mu$.

- $id_i = id_j$.

  Then before m reached p_i, p_j was related to m and p_i was related to m'. So by induction, before m reached p_i, these pairs satisfied condition 1. Let ξ_1 and ξ_2 denote the maximum of all identities of active processes (and of messages) between p_j and m and between p_i and m', respectively, before m reached p_i; and let $\#_1$ and $\#_2$ denote the number of active processes (and of messages) between p_j and m and between p_i and m', respectively, before m reached p_i. Moreover, let ξ'_π, ξ'_μ, $\#'_\pi$ and $\#'_\mu$ have the same meaning as in the previous case. We consider seven cases.

  If $id'_i > max\{\xi_1, \xi_2\}$, then $\xi'_\pi = id'_i = \xi'_\mu$ and $\#'_\pi = 1 = \#'_\mu$.

  If $\xi_1 > max\{id'_i, \xi_2\}$, then $\xi'_\pi = \xi_1 = \xi'_\mu$ and $\#'_\pi = \#_1 = \#'_\mu$.

  If $\xi_2 > max\{id'_i, \xi_1\}$, then $\xi'_\pi = \xi_2 = \xi'_\mu$ and $\#'_\pi = \#_2 = \#'_\mu$.

  If $id'_i = \xi_1 > \xi_2$, then $\xi'_\pi = id'_i = \xi'_\mu$ and $\#'_\pi = \#_1 + 1 = \#'_\mu$.

  If $id'_i = \xi_2 > \xi_1$, then $\xi'_\pi = id'_i = \xi'_\mu$ and $\#'_\pi = \#_2 + 1 = \#'_\mu$.

  If $\xi_1 = \xi_2 > id'_i$, then $\xi'_\pi = \xi_1 = \xi'_\mu$ and $\#'_\pi = \#_1 + \#_2 = \#'_\mu$.

  If $id'_i = \xi_1 = \xi_2$, then $\xi'_\pi = id'_i = \xi'_\mu$ and $\#'_\pi = \#_1 + \#_2 + 1 = \#'_\mu$. □

We say that an active process or message is *maximal* if its identity is maximal among the active processes or messages in the ring, respectively. In the following proposition we write ξ_π and ξ_μ for the identity of maximal active processes and messages, respectively. The number of active processes and messages with the same identity id is denoted by $\#^{id}_\pi$ and $\#^{id}_\mu$, respectively. We write $\#_\pi$ and $\#_\mu$ for the number of maximal active processes and messages, respectively.

Proposition 8. *For Algorithm $\mathcal{B}$, until a leader is elected, there exist active processes and messages in the ring, and $\xi_\pi = \xi_\mu$ and $\#_\pi = \#_\mu$.*

Proof. Consider a reachable state of Algorithm $\mathcal{B}$ in which no leader has yet been elected. We apply induction on the minimal number of transitions needed to reach this state from an initial state.
Basis: Prior to the first arrival of a message, every process is active and has generated a message with its own identity; thus the proposition trivially holds.
Induction step: By induction, $\xi_\pi = \xi_\mu$ and $\#_\pi = \#_\mu$; we write ξ for ξ_π and ξ_μ, and $\#$ for $\#_\pi$ and $\#_\mu$. When a message arrives at a passive process, it is simply forwarded. Assume a message m with parameters (id, hop) arrives at an active process p_i with identity id_i. If $hop = n$, then p_i is elected as the leader. Now suppose that $hop < n$. We consider four cases.

- $id_i > id$. Since $\xi_\pi = \xi_\mu$, m is not a maximal message. It is purged by p_i. The values of ξ_π and ξ_μ remain unchanged.
- $id_i < id$. Since $\xi_\pi = \xi_\mu$, p_i is not a maximal process. It becomes passive. The values of ξ_π and ξ_μ remain unchanged.
- $id_i = id < \xi$. Then p_i selects a new identity id'_i, and sends the message $(id'_i, 1)$. If $id'_i > \xi$, then $\xi'_\pi = id'_i = \xi'_\mu$ and $\#'_\pi = 1 = \#'_\mu$. If $id'_i = \xi$, then $\xi'_\pi = \xi = \xi'_\mu$ and $\#'_\pi = (\# + 1) = \#'_\mu$. If $id'_i < \xi$, then $\xi'_\pi = \xi = \xi'_\mu$ and $\#'_\pi = \# = \#'_\mu$.
- $id_i = id = \xi$. Then p_i selects a new identity id'_i, and sends the message $(id'_i, 1)$. We distinguish two cases.
  - $\# > 1$. If $id'_i > \xi$, then $\xi'_\pi = id'_i = \xi'_\mu$ and $\#'_\pi = 1 = \#'_\mu$. If $id'_i = \xi$, then $\xi'_\pi = \xi = \xi'_\mu$ and $\#'_\pi = \# = \#'_\mu$. If $id'_i < \xi$, then $\xi'_\pi = \xi = \xi'_\mu$ and $\#'_\pi = (\# - 1) = \#'_\mu$.
  - $\# = 1$. Then clearly p_i is related to m, and all other active processes and messages are between them. Since $hop < n$, p_i is not the originator of m, so by Lemma 7.2 there is some active process or message between them. Let $\xi_0 > 0$ be the maximum of all identities of active processes $\neq p_i$ and messages $\neq m$. By Lemma 7.1, $\#^{\xi_0}_\pi = \#^{\xi_0}_\mu$. If $id'_i > \xi_0$, then $\xi'_\pi = id'_i = \xi'_\mu$ and $\#'_\pi = 1 = \#'_\mu$. If $id'_i = \xi_0$, then $\xi'_\pi = \xi_0 = \xi'_\mu$ and $\#'_\pi = (\#^{\xi_0}_\pi + 1) = \#'_\mu$. If $id'_i < \xi_0$, then $\xi'_\pi = \xi_0 = \xi'_\mu$ and $\#'_\pi = \#^{\xi_0}_\pi = \#'_\mu$. $\square$

Theorem 9. *Let channels be FIFO. Then Algorithm $\mathcal{B}$ terminates with probability one, and upon termination exactly one leader is elected.*

Proof. By Proposition 8, some processes remain active until a leader is elected. A process can be elected as the leader only if it receives a message with a hop counter equal to n, which means the message has passed through all other processes and made them passive. Hence, we have uniqueness of the leader.

It remains to show that the algorithm terminates with probability one. When there are $\ell \geq 2$ active processes in the ring, these processes all remain active if and only if they all the time choose the same identity. Otherwise, at least one active process will become passive. The probability that all active processes select the same identity in one "round" is $(\frac{1}{k})^{\ell-1}$. So the probability for all ℓ active processes to choose the same identity m times in a row is $(\frac{1}{k})^{m(\ell-1)}$. Since $k \geq 2$, the probability that the number of active processes eventually decreases is one.

Clearly, when there is only one active process in the ring, it will be elected as the leader. After the round trip of its final message there are no remaining messages, because channels are FIFO. $\square$

5 Performance Analysis

A probabilistic analysis in [Itai and Rodeh 1981] reveals that if $k = n$, the expected number of rounds required for the Itai-Rodeh algorithm to elect a leader in a ring with size n is bounded by $e \cdot \frac{n}{n-1}$. The expected number of messages for each round is in the order of $n \log n$. Hence, the average message complexity of the Itai-Rodeh algorithm is $\Theta(n \log n)$. Likewise, Algorithms $\mathcal{A}$ and $\mathcal{B}$ have an average message complexity of $\Theta(n \log n)$.

The probabilistic temporal logic PCTL can be used to express *soft deadlines*, such as "the probability of electing a leader within t discrete time steps is at most 0.5".[2] A PCTL formula to calculate the probability of electing a leader within t discrete time steps for a ring with two processes is

```
P=? [ true U<=t (s1=4 & s2=4 & leader1+leader2=1)]
```

We used PRISM to calculate the probability that Algorithms $\mathcal{A}$ and $\mathcal{B}$ terminate within a given number of transitions, for rings of size two and three. The experimental results presented in Fig. 2 and Fig. 3 indicate that Algorithm $\mathcal{B}$ seems to have a better performance than Algorithm $\mathcal{A}$. Note that when t moves to infinity, both algorithms elect a leader with probability one.

6 Formal Verifications of Leader Election Algorithms

On the web page of PRISM [PRISM], the Itai-Rodeh algorithm for asynchronous rings was adapted for synchronous rings. In PRISM, processes synchronize on action labels, so a synchronous ring can simply be modeled by excluding channels from the specification. Processes are synchronized in the same round, thus round numbers are not needed (similar to our Algorithm $\mathcal{A}$). The state space therefore becomes finite, and PRISM could be used to verify the property "with probability

[2] Each discrete time step corresponds to one transition in the algorithm.

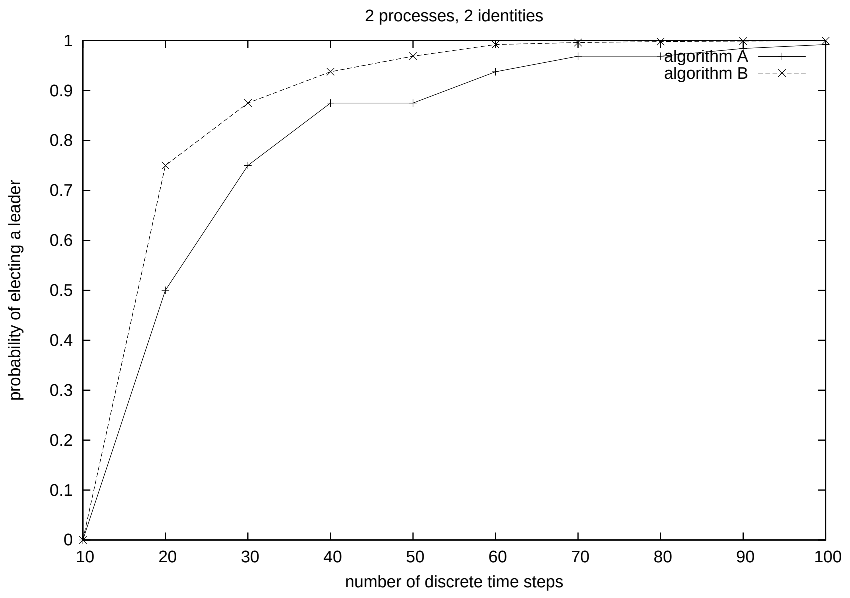

Figure 2: The probability of electing a leader with deadlines.

one, eventually a unique leader is elected", for rings up to size eight. Also the probability of electing a leader in one round was calculated.

Garavel and Mounier [Garavel and Mounier 1997] described both Le Lann's algorithm and the Chang-Roberts algorithm using the process algebraic language LOTOS. They studied these two algorithms in the presence of unreliable communication network and/or unreliable processes and suggested some improvements. Their verification was performed using the model checker CADP. Fredlund et al. [Fredlund et al. 1997] gave a manual correctness proof of the Dolev-Klawe-Rodeh algorithm in the process algebraic language μCRL, for arbitrary ring size. Brunekreef et al. [Brunekreef et al. 1996] designed a number of leader election algorithms for a broadcast network, where processes may participate and crash spontaneously. They used linear-time temporal logic to manually prove that the algorithms satisfy their requirements.

The IEEE 1394 high performance serial bus (called "FireWire") is used to transport video and audio signals within a network of multimedia devices. In the tree identify phase of IEEE 1394, which takes place after a bus reset in the network, a leader is elected. For the sake of performance, identities of nodes cannot be sent around the network, so that it is basically an anonymous network. The leader election algorithm in the IEEE 1394 standard works for acyclic, connected networks. If a cycle is present, it produces a timeout. The algorithm

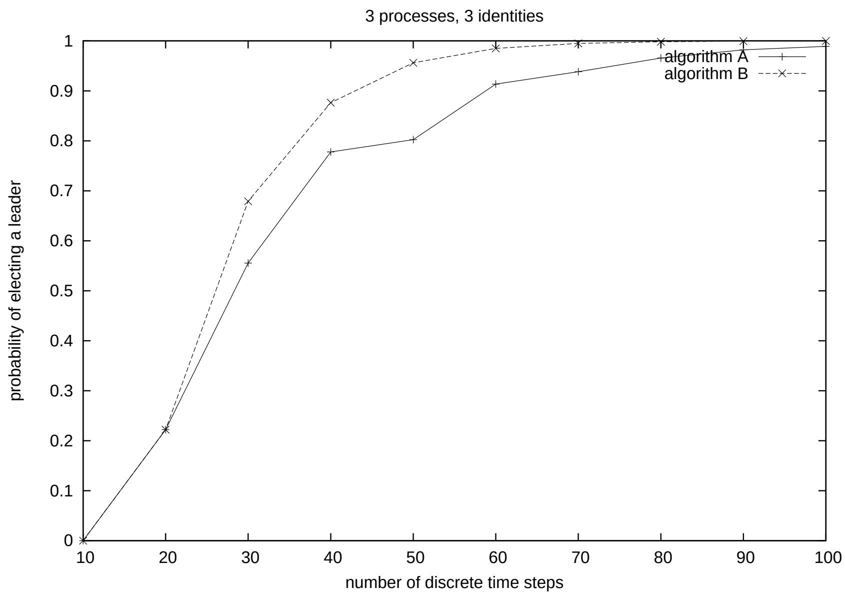

Figure 3: The probability of electing a leader with deadlines.

has been specified and verified with a number of different formal techniques. We give an overview of these case studies.

Shankland and van der Zwaag [Shankland and van der Zwaag 1998] manually verified the leader election algorithm in μCRL, at three different levels of detail. Shankland and Verdejo [Shankland and Verdejo 2001] used E-LOTOS to manually verify the algorithm. Abrial et al. [Abrial et al. 2003] used an event-driven approach with the B Method to develop mathematical models of the algorithm; the internal consistency of each model as well as its correctness with regard to its previous abstraction were proved mechanically. Verdejo et al. [Verdejo et al. 2003] described the algorithm at different abstract levels, using the language Maude based on rewriting logic; they verified the algorithm by an exhaustive exploration of the state space that always exactly one leader is chosen. Moreover, they gave a manual correctness proof for general acyclic networks. Devillers et al. [Devillers et al. 2000] verified the algorithm using an I/O automata model; the main part of their proof has been checked with the theorem prover PVS. Romijn [Romijn 2001] extended their I/O automata model with timing parameters from the IEEE 1394 standard, and manually proved that under certain timing restrictions the algorithm behaves correctly. Calder and Miller [Calder and Miller 2003] verified some properties of the algorithm using the model checker Spin, for networks with up to six nodes. Schuppan and Biere

[Schuppan and Biere 2003] used the model checker SMV to check the correctness of the algorithm for networks with up to ten nodes.

7 Conclusion and Future Work

In this paper, we presented two probabilistic leader election algorithms for anonymous unidirectional rings with FIFO channels. In [Fokkink and Pang 2004], we also showed that if processes can select identities from a set of only two elements, then both of our algorithms work correctly for non-FIFO channels. We gave a manual correctness proof for each algorithm. Future work is to formalize and check these proofs by means of a theorem prover such as PVS.

Both algorithms were specified and successfully model checked with PRISM. They satisfy the property "with probability 1, eventually exactly one leader is elected". The complete specifications in PRISM can be found at `seshome.informatik.uni-oldenburg.de/~jun/leader/`. The generation of state spaces and the verifications were performed on a 1.4 GHz AMD Althlon^{TM} Processor with 512 Mb memory. The PRISM automatic verification of our algorithms was reported in [Fokkink and Pang 2005]. Although the PRISM verification of the two algorithms is limited to ring size up to four, it allowed us to perform some analysis during the design phase, which gave us some insight into the algorithms. In particular, we found that the experiments feature in PRISM is quite useful. It provides a way of automating multiple instances of model checking. By making such experiments, we could show that the first algorithm is on average faster than the other (see Section 5). The work with PRISM has given additional value to this work and has proved beneficial.

In [Itai and Rodeh 1981] it is stated:

> "We could have used any of the improved algorithms [Dolev et al. 1982], [Burns 1980], [Hirschberg and Sinclair 1980], [Peterson 1982]."

Following this direction, we developed two more probabilistic leader election algorithms, based on the Dolev-Klawe-Rodeh algorithm [Dolev et al. 1982]. Both of them are finite-state, and we model checked them successfully in μCRL [Blom et al. 2001] up to ring size six. The adaptations of the Dolev-Klawe-Rodeh algorithm are very similar to our adaptations (Algorithms $\mathcal{A}$ and $\mathcal{B}$) of the Chang-Roberts algorithm; i.e., processes again select random identities, and name clashes are resolved in exactly the same way. Therefore our adaptations of the Dolev-Klawe-Rodeh algorithm are not presented here. The interested reader can find the specifications of all our algorithms at `seshome.informatik.uni-oldenburg.de/~jun/leader/`. These specifications are in the language μCRL, which was used for an initial non-probabilistic model checking exercise.

Acknowledgments

We thank Gethin Norman for helping us with PRISM, and Gerard Tel for his comments on an earlier version of this paper.

References

[Abrial et al. 2003] J.-R. Abrial, D. Cansell, and D. Mery. A mechanically proved and incremental development of IEEE 1394 FireWire tree identify protocol. *Formal Aspects of Computing*, 14(3): 215-227, 2003.

[Alur and Henzinger 1999] R. Alur and T.A. Henzinger. Reactive modules. *Formal Methods in System Design*, 15(1): 7-48, 1999.

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

[Baier et al. 2000] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model checking continuous-time Markov chains by transient analysis. In *Proc. 12th Conference on Computer Aided Verification*, LNCS 1855, pp. 358–372. Springer, 2000.

[Baier and Kwiatkowska 1998] C. Baier and M. Kwiatkowska. Model checking for a probabilistic branching time logic with fairness. *Distributed Computing*, 11(3): 125-155, 1998.

[Blom et al. 2001] S.C.C. Blom, W.J. Fokkink, J.F. Groote, I.A. van Langevelde, B. Lisser, and J.C. van de Pol. μCRL: A toolset for analysing algebraic specifications. In *Proc. 13th Conference on Computer Aided Verification*, LNCS 2102, pp. 250–254. Springer, 2001.

[Brunekreef et al. 1996] J.J. Brunekreef, J.-P. Katoen, R.L.C. Koymans, and S. Mauw. Algebraic specification of dynamic leader election protocols in broadcast networks. *Distributed Computing*, 9(4): 157-171, 1996.

[Burns 1980] J.E. Burns. A formal model for message passing systems. Technical Report TR-91, Indiana University, 1980.

[Calder and Miller 2003] M. Calder and A. Miller. Using Spin to analyse the tree identity phase of the IEEE 1394 high-performance serial bus (FireWire) protocol. *Formal Aspects of Computing*, 14(3): 247-266, 2003.

[Chang and Roberts 1979] E.J.H. Chang and R. Roberts. An improved algorithm for decentralized extrema-finding in circular configurations of processes. *Communication of the ACM*, 22(5): 281-283, 1979.

[Clarke et al. 2000] E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. MIT Press, 2000.

[Devillers et al. 2000] M.C.A. Devillers, W.O.D. Griffioen, J.M.T. Romijn, and F.W. Vaandrager. Verification of a leader election protocol - Formal methods applied to IEEE 1394. *Formal Methods in System Design*, 16(3): 307–320, 2000.

[Dolev et al. 1982] D. Dolev, M. Klawe, and M. Rodeh. An $\mathcal{O}(n \log n)$ unidirectional distributed algorithm for extrema finding in a circle. *Journal of Algorithms*, 3: 245-260, 1982.

[Fokkink and Pang 2004] W.J. Fokkink and J. Pang. Simplifying Itai-Rodeh leader election for anonymous rings. Technical Report SEN-R0405, CWI, Amsterdam, 2004.

[Fokkink and Pang 2005] W.J. Fokkink and J. Pang. Simplifying Itai-Rodeh leader election for anonymous rings. In *Proc. 4th Workshop on Automated Verification of Critical Systems*, Electronic Notes in Theoretical Computer Science, 128(6): 53-68, Elsevier, 2005.

[Franklin 1982] R. Franklin. On an improved algorithm for decentralized extrema finding in circular configurations of processors. *Communication of the ACM*, 25(5): 336-337, 1982.

[Fredlund et al. 1997] L.A. Fredlund, J.F. Groote, and H.P. Korver. Formal verification of a leader election protocol in process algebra. *Theoretical Computer Science*, 177: 459-486, 1997.

[Fujita et al. 1997] M. Fujita, P.C. McGeer, and J.C.-Y. Yang. Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. *Formal Methods in System Design*, 10(2/3): 149-169, 1997.

[Garavel and Mounier 1997] H. Garavel and L. Mounier. Specification and verification of various distributed leader election algorithms for unidirectional ring networks. *Science of Computer Programming*, 29(1/2): 171-197, 1997.

[Hansson and Jonsson 1994] H. Hansson and B. Jonsson. A Logic for Reasoning about Time and Reliability. *Formal Aspects of Computing* 6(5): 512-535, 1994.

[Hirschberg and Sinclair 1980] D.S. Hirschberg and J.B. Sinclair. Decentralized extrema-finding in circular configurations of processes. *Communication of the ACM*, 23(11): 627-628, 1980.

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

[Itai and Rodeh 1990] A. Itai and M. Rodeh. Symmetry breaking in distributed networks. *Information and Computation*, 88(1): 60-87, 1990.

[Kwiatkowska et al. 2002] M.Z. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic symbolic model checker. In *Proc. 12th Conference on Computer Performance Evaluation, Modelling Techniques and Tools*, LNCS 2324, pp. 200-204. Springer, 2002.

[Le Lann 1977] G. Le Lann. Distributed systems: Towards a formal approach. *Information Processing 77, Proc. of the IFIP Congress*, pp. 155-160, 1977.

[Lynch 1996] N.A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, 1996.

[Peterson 1982] G.L. Peterson. An $\mathcal{O}(n \log n)$ unidirectional algorithm for the circular extrema problem. *IEEE Transactions on Programming Languages and Systems*, 4: 758-762, 1982.

[PRISM] PRISM web page. `http://www.cs.bham.ac.uk/~dxp/prism`.

[Romijn 2001] J.M.T. Romijn. A timed verification of the IEEE 1394 leader election protocol. *Formal Methods in System Design*, 19(2): 165–194, 2001.

[Schuppan and Biere 2003] V. Schuppan and A. Biere. Verifying the IEEE 1394 FireWire tree identity protocol with SMV. *Formal Aspects of Computing*, 14(3): 267-280, 2003.

[Shankland and Verdejo 2001] C. Shankland and A. Verdejo. A case study in abstraction using E-LOTOS and the FireWire. *Computer Networks*, 37(3/4): 481-502, 2001.

[Shankland and van der Zwaag 1998] C. Shankland and M.B. van der Zwaag. The tree identify protocol of IEEE 1394 in μCRL. *Formal Aspects of Computing*, 10: 509-531, 1998.

[Tel 1994] G. Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, 1994.

[Verdejo et al. 2003] A. Verdejo, I. Pita, and N. Marti-Oliet. Specification and verification of the tree identify protocol of IEEE 1394 in rewriting logic. *Formal Aspects of Computing*, 14(3): 228-246, 2003.