Available online at www.sciencedirect.com
SCIENCE DIRECT®
ELSEVIER
Electronic Notes in Theoretical Computer Science 128 (2005) 53–68
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Simplifying Itai-Rodeh Leader Election for Anonymous Rings

Wan Fokkink[1]

Department of Software Engineering, CWI, Amsterdam, The Netherlands
Department of Computer Science, Vrije Universiteit Amsterdam, The Netherlands

Jun Pang[2]

Department of Software Engineering, CWI, Amsterdam, The Netherlands

Abstract

We present two probabilistic leader election algorithms for anonymous unidirectional rings with FIFO channels, based on an algorithm from Itai and Rodeh [14]. 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.

Keywords: Distributed computing, leader election, anonymous networks, probabilistic algorithms, model checking.

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 leader. This is a fundamental

[1] Email: wan@cwi.nl; wanf@cs.vu.nl
[2] Email: pangjun@cwi.nl

doi:10.1016/j.entcs.2005.04.004

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. These algorithms have different message complexity in 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 [17]. 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 [7] 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 [10] developed a leader election algorithm for bidirectional rings with a worst-case message complexity of $\mathcal{O}(n \log n)$. Peterson [18] and Dolev, Klawe, and Rodeh [8] 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. 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 [2] showed that there does not exist a terminating algorithm for electing a leader in an asynchronous anonymous network. According to this result, a *Las Vegas* algorithm (meaning that the probability that the algorithm terminates is greater than zero, and all terminal configurations are correct) is the best possible option.

Itai and Rodeh [14,15] proposed a probabilistic leader election algorithm for anonymous unidirectional rings, 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, 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 is a Las Vegas algorithm that terminates with probability one; it takes $n \log n$ messages on average.

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 claim 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 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 [16], 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".

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.

On the web page of PRISM (`http://www.cs.bham.ac.uk/~dxp/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 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. More related work on formal verification of leader election algorithms can be found in the full version of this paper [9].

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. Section 5 reveals some experimental results using PRISM on the number of messages needed to terminate. We conclude this paper and discuss some future work in Section 6.

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. Channels are unidirectional: a message sent by p_i is added to the message queue of $p_{(i+1) \bmod n}$. 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

Itai and Rodeh [14,15] 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 is a Las Vegas algorithm that terminates with probability one. The Itai-Rodeh algorithm is based on the Chang-Roberts algorithm [7], 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 say that an execution sequence of the Itai-Rodeh algorithm has *terminated* if each process is either passive or elected as leader, and there are no remaining messages in the channels.

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 $(round, id) > (round_i, id_i)$,[a] 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.

[a] We compare $(round, id)$ and $(round_i, id_i)$ lexicographically.

2.2 Round numbers are needed

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 $(v, 1, true)$ 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.)

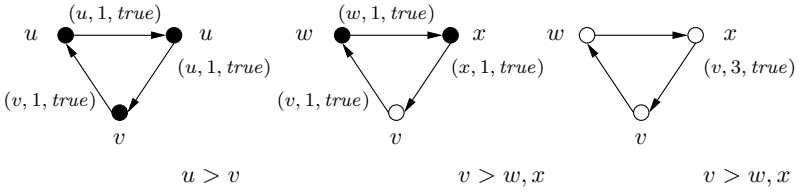

Fig. 1. Round numbers are essential if channels are not FIFO

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.

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 $(id_i, 1, true)$.
- Upon receipt of a message (id, 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 $id_i' \in \{1, \ldots, k\}$ and sends the message $(id_i', 1, true)$;
  - if $id = id_i$ and $hop < n$, then p_i passes on the message $(id, hop + 1, false)$;
  - if $id > id_i$, then p_i becomes passive ($state_i' = passive$) and passes on the message $(id, hop + 1, bit)$;
  - if $id < id_i$, then p_i purges the message.

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.

PRISM [16] is a probabilistic model checker. 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 [12,4] if the model is a DTMC or an MDP, or CSL [3] in the case of a CTMC.

In order to model check probabilistic properties of Algorithm $\mathcal{A}$, we first encoded the algorithm as a DTMC model using the PRISM language, which is a simple, state-based language, based on the Reactive Modules formalism of Alur and Henzinger [1]. A system is composed of a number of modules that contain local variables, and that can interact with each other. The behavior of a DTMC 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.

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 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. The identity of a message is mes1_id, 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 has 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 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);

	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 results for Algorithm $\mathcal{A}$ with FIFO channels

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 are in state 4 (s1=4 & s2=4), with exactly one process elected as the leader (leader1+leader2=1), is 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).

To model check this property, the algorithmic description (in the module-based language) was parsed and converted into an MTBDD [11]. 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

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.

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, increasing the counter hop by one; an active process p_i ($state_i = active$) behaves according to one of the following steps:
  - if $hop = n$, 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 message.

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). 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 is 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.

	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	3	4	yes	176,068	521,452
Ex.4	4	4	4	yes	537,467	1,615,408
Ex.5	5	2	5	yes	752,047	2,626,405

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

```
[ ] (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.

5 Performance Analysis

A probabilistic analysis in [14] 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 $\mathcal{O}(n \log n)$. Hence, the average message complexity of the Itai-Rodeh algorithm is $\mathcal{O}(n \log n)$. Likewise, Algorithms $\mathcal{A}$ and $\mathcal{B}$ have an average message complexity of $\mathcal{O}(n \log n)$.

The probabilistic temporal logic PCTL [12,4] can be used to express *soft*

	Processes	Identities	Channel size	Steps ($\mathcal{A}$)	Steps ($\mathcal{B}$)
Ex.1	2	2	2	25.0	19.0
Ex.2	3	3	3	33.6	29.3
Ex.3	4	3	4	52.5	46.0

Table 3
The expected number of steps before a unique leader is elected for each algorithm.

deadlines, such as "the probability of electing a leader within t discrete time steps is at most 0.5". [3] 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.

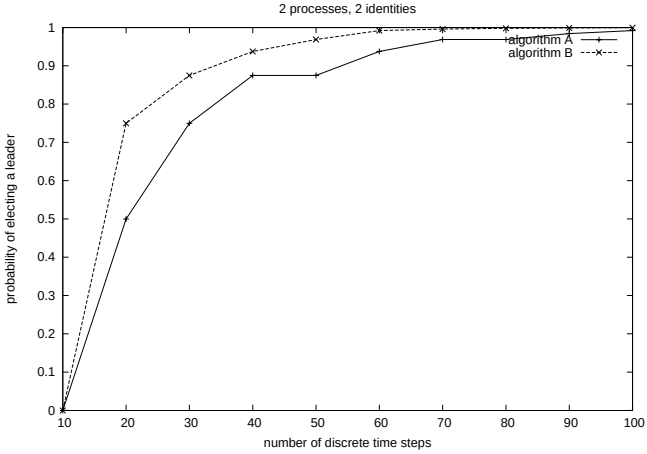

Fig. 2. The probability of electing a leader with deadlines.

We also used a development version of PRISM to compute the expected number of steps before a unique leader is elected for each algorithm with fixed configurations. (This new feature will be included in the next release of PRISM.) Some experimental results (see Table 3) show that Algorithm $\mathcal{B}$ is on average faster than Algorithm $\mathcal{A}$.

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

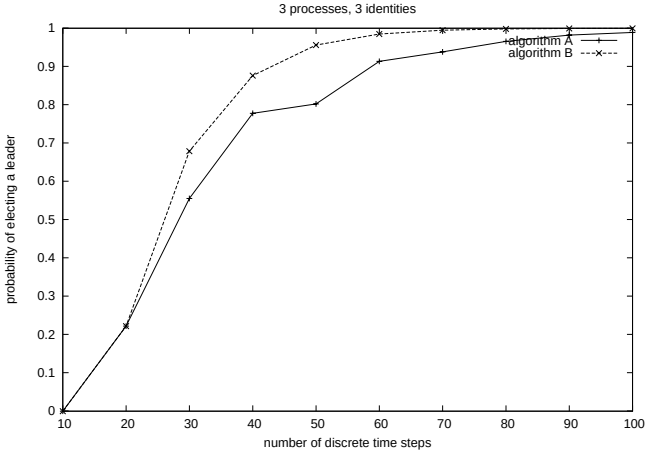

Fig. 3. The probability of electing a leader with deadlines.

6 Conclusion and Future Work

In this paper, we presented two probabilistic leader election algorithms for anonymous unidirectional rings with FIFO channels. 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 `www.cwi.nl/~pangjun/leader`. The generation of state spaces and the verifications were performed on a 1.4 GHz AMD Althlon^{TM} Processor with 512 Mb memory. Furthermore, we gave a manual correctness proof for each algorithm (see [9]). Future work is to formalize and check these proofs by means of a theorem prover such as PVS.

Itai and Rodeh [14] stated:

"We could have used any of the improved algorithms [6], [8], [13], [18]."

Following this direction, we developed two more probabilistic leader election algorithms, based on the Dolev-Klawe-Rodeh algorithm [8,10]. Both of them are finite-state, and we model checked them successfully in μCRL [5] 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 `www.cwi.nl/~pangjun/leader`. These specifications are in the language μCRL, which was used for an initial non-probabilistic model checking exercise.

Acknowledgement

This research is supported by the Dutch Technology Foundation STW under the project CES5008. We thank Gethin Norman for helping us with PRISM.

References

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

[2] D. Angluin. Local and global properties in networks of processors (extended abstract). In *Proc. STOC'80*, pp. 82-93, ACM, 1980.

[3] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model checking continuous-time Markov chains by transient analysis. In *Proc. CAV'00*, LNCS 1855, pp. 358–372. Springer, 2000.

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

[5] 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. CAV'01*, LNCS 2102, pp. 250–254. Springer, 2001.

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

[7] 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.

[8] 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.

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

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

[11] 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.

[12] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing* 6(5): 512-535, 1994.

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

[14] A. Itai and M. Rodeh. Symmetry breaking in distributive networks. In *Proc. FOCS'81*, pp. 150–158. IEEE Computer Society, 1981.

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

[16] M. Z. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic symbolic model checker. In *Proc. TOOLS'02*, LNCS 2324, pp. 200-204. Springer, 2002.

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

[18] 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.