Leader election in asynchronous undirected ring
based on
Symmetry Breaking in Distributed Networks by Itai, Rodeh

Filip Jasionowicz

December 2023

Abstract

The goal of this project is to implement a leader election algorithm proposed by Itai and Rodeh, modify it to work with an undirected ring and provide a sketch of the complexity and correctness analysis.

1 The model

The original algorithm proposed in [2] assumes the following model:

1. There is no global clock and processors are not synchronized in any way
2. Processors are indistinguishable from each other and run the same code
3. Each processor knows the total number n of processors in a ring
4. All channels are FIFO channels
5. Processors are connected in a ring (cycle) where each of them has an output channel to its successor and an input channel from its predecessor

Although 4. is not explicitly stated in an original paper, it is necessary for the proposed algorithm to work [1].

In this project we will modify the model by adding one additional input and output channel to each of the processors, so now they can send and receive messages to/from both of their neighbours on a ring. Note that the local orientation of the channels might not be the same for all processors - a message going locally "right" and then locally "left" might not end in the same processor.

2 Itai Rodeh algorithm for asynchronous directed ring

Let's start by recalling the idea of original Itai Rodeh algorithm for the asynchronous directed ring. All processors are initially active. In each turn (although the model is asynchronous, the channels are FIFO, so the turns are preserved) each active processor receive the message from its predecessor. If their bit is 0 and received bit is 1, they become passive. Otherwise they generate random bit and send them to their successor. Passive processors just pass the messages further. If a processor performed c steps and is still active, it becomes a candidate for the leader. All leader candidates send a message containing a counter. If the message returns with the counter equal to size of the ring, then the sender knows it's an only leader. Otherwise, all leader candidates receive message with smaller counter. In this situation the procedure is repeated.

Algorithm 1 directed Itai Rodeh

```
active ← true
bit ← random(0/1)                                   ▷ Chose 0 or 1 with probability $\frac{1}{2}$
bit → SEND
repeat
    msg ← RECEIVE
    if msg is bit then
        HandleBit(msg)                              ▷ Regular bit message
    else
        HandleCounter(msg)                          ▷ Leader verification message
until unique leader not found

procedure HandleBit(rbit)
    if active then
        if run for c turns then
            counter ← 1
            counter → SEND                          ▷ Send the verification message
        if rbit = 1 and bit = 0 then
            active ← false
        bit ← random(0/1)
        bit → SEND
    else
        rbit → SEND                                 ▷ Pass the message

procedure HandleCounter(counter)
    if active then
        if counter = n then
            return WE ARE THE LEADER                ▷ Leader found
        else
            go to 2                                 ▷ Multiple leader candidates, we try again
    else
        rbit → SEND                                 ▷ Pass the message
```

3 Sketch of analysis

3.1 Partial correctness

First let's look at the partial correctness. Assume the algorithm has stopped. If more than one leader has been found in the verification phase, then our algorithm would still be running, so the only possible answers it might have given are 0 or 1 leaders. To obtain 0 leaders however, all processors would have to become passive which is not possible - processor can become passive only if it has an active predecessor with bit set to 1, and this predecessor couldn't be killed in that round, since processor with bit 1 cannot be killed. This proves that if the algorithm finishes, it elects a single leader.

3.2 Termination

Proposed algorithm is obviously a Las Vegas randomized algorithm (we proved the partial correctness, and the run time depends on the randomly generated bits). Our objective is to show that for some c the expected running time is finite and reasonable. In [2] the authors propose the parameter c to be $5 \log n$. Intuition underlying this choice is actually pretty straightforward. In each round each active processor has a $\frac{1}{4}$ chance of becoming inactive, since it has to draw 0 and its predecessor 1, thus expected number of active processors reduces by a quarter in each round, giving that the expected number of rounds after only one processor remains active is around $\log_{\frac{4}{3}} n \approx 2.4 \log n$. By doubling this number and setting number of rounds to $c = 5 \log n$ we ensure that the probability of more than one processor being active at the end of the algorithm is relatively small. In each turn each active processor sends one message to its successor and each passive processor forwards only one message, so in each turn n messages are passed, giving the total complexity of $O(cn) = O(n \log n)$.

4 Itai Rodeh algorithm for asynchronous undirected ring

Let's modify the Algorithm 1 to work on undirected ring. The general idea is that active processors send their random bit to both our neighbours, and become passive if their bit is 0 and either of neighbours is 1. The verification message also needs to be send and received in both directions. In the following algorithm changes compared to directed version are marked red.

Algorithm 2 undirected Itai Rodeh

```
$active \leftarrow true$
$bit \leftarrow random(0/1)$                          ▷ Chose 0 or 1 with probability $\frac{1}{2}$
$bit \rightarrow SEND\ BOTH$                          ▷ Sent to both channels
repeat
    $msg \leftarrow RECEIVE\ ANY$                     ▷ Receive message from either channel
    if $msg$ is bit then
        HandleBit($msg$)                              ▷ Regular bit message
    else
        HandleCounter($msg$)                          ▷ Leader verification message
until unique leader not found

procedure HandleBit($rbit$)
    if $active$ then
        $rbit_2 \leftarrow RECEIVE\ OTHER$            ▷ Receive from other
                                                        channel than $rbit$
        if run for $c$ turns then
            $counter \leftarrow 1$
            $counter \rightarrow SEND\ BOTH$          ▷ Send the verification message
        if ($rbit = 1$ or $rbit_2 = 1$ ) and $bit = 0$ then
            $active \leftarrow false$
        $bit \leftarrow random(0/1)$
        $bit \rightarrow SEND\ BOTH$                  ▷ Sent to both channels
    else
        $rbit \rightarrow SEND\ OTHER$                ▷ Pass the message not to its sender

procedure HandleCounter($counter$)
    if $active$ then
        $counter_2 \leftarrow RECEIVE\ OTHER$         ▷ Receive from other
                                                        channel than $counter$
        if $counter = n$ and $counter_2 = n$ then
            return WE ARE THE LEADER                  ▷ Leader found
        else
            go to 2                                   ▷ Multiple leader candidates, we try again
    else
        $rbit \rightarrow SEND\ OTHER$                ▷ Pass the message not to its sender
```

Note that partial correctness argument holds. The probability of active processor becoming passive in one turn is actually higher in Algorithm 2, being $\frac{3}{8}$. This implicates that after $c = 5 \log n$ turns the probability of more than one processor being active is even smaller than in Algorithm 1. At the same time number of send messages is exactly doubled, giving the complexity of $O(2nc) = O(n \log n)$.

References

[1] Wan Fokkink and Jun Pang. Simplifying itai-rodeh leader election for anonymous rings. *Electronic Notes in Theoretical Computer Science*, 128(6):53–68, 2005. Proceedings of the Fouth International Workshop on Automated Verification of Critical Systems (AVoCS 2004).

[2] Alon Itai and Michael Rodeh. Symmetry breaking in distributed networks. *Information and Computation*, 88(1):60–87, 1990.