Leader Election

We study Leader Election in rings.

Why rings?

- historical reasons
  - original motivation: regenerate lost token in token ring networks
- illustrates techniques and principles
- good for lower bounds and impossibility results

Outline

- Specification of Leader Election
- YAIR
- Leader election in asynchronous rings:
  - An $O(n^2)$ algorithm
  - An $O(n \log(n))$ algorithm
- The revenge of the lower bound!
- Leader election in synchronous rings
  - Breaking the $\lceil \log(n) \rceil$ barrier
Message passing: Model

- \( n \) processors \( p_0 \ldots p_{n-1} \)
- connected by bi-directional communication channels
- topology represented by undirected graph

Some links may be missing

Processes

- Each \( p_i \) is a state machine
- \( p_i \)'s state includes:
  - state set \( Q_i \)
  - distinguished initial states
  - could be infinite

State Transitions

A state transition:

- input: accessible state of \( p_i \) (doesn't depend on \( \text{outbuf}_i \))
- consumes all messages in \( \text{inbuf}_i \)
- outputs at most a message per channel

Terminology

Definition: A configuration is a vector \( C = (q_0, \ldots, q_{n-1}) \)
- each \( q_i \) is a state of \( p_i \)
- set of \( \text{outbuf}_i \) are messages in transit
- In an initial configuration each \( q_i \) is an initial state of \( p_i \)

Definition: An event is
- a computation event \( \text{comp}(i) \)
- a delivery event \( \text{del}(i,m) \)

Definition: An execution is an infinite sequence \( C_0, f_0, C_1, f_1, \ldots \) where
- \( C_0 \) is an initial configuration
- each \( C_i \) is a configuration
- each \( f_i \) is an event

Definition: A schedule for the above execution is the sequence of events \( f_0, f_1, \ldots \)
Safety and Liveness

Safety property: “nothing bad happens”
• holds in every finite execution prefix
  – Windows™ never crashes
  – if one general attacks, both do
  – a program never terminates with a wrong answer

Liveness property: “something good eventually happens”
• no partial execution is irremediable
  – Windows™ always reboots
  – both generals eventually attack
  – a program eventually terminates

Admissible executions satisfy safety and liveness properties.

A really cool theorem

Every property is a combination of a safety property and a liveness property

(Alpern and Schneider)

Asynchronous Message-Passing Systems

If \( f_k = \text{del}(i,j,m) \)
• \( m \) is in outbuf\([i]\), where \( i \) is \( p_i \)'s label for channel \( \{p_i, p_j\} \)
• in \( C_{k-1} \)
  – remove \( m \) from outbuf\([i]\)
  – add \( m \) to inbuf\([j]\), where \( j \) is \( p_j \)'s label for channel \( \{p_i, p_j\} \)

Admissible if:
• Every processor takes an infinite number of computation steps
• Every message sent is eventually delivered

Synchronous Message-Passing Systems

If \( f_k = \text{comp}(i) \)
• \( p_i \) changes state according to its transition function
• empties inbuf\([i]\) in \( C_{k-1} \)
• might add messages to outbuf\([i]\) in \( C_k \)

Admissible if:
• all asynchronous constraints, plus
• execution partitioned into disjoint rounds
• one delivery event for every message in every outbuf
• followed by one computation event for every processor

Remarks
• not realistic, but
• good for algorithm design
• good for lower bounds
Timed-Asynchronous

Complexity

- **TIME**
  - each processor’s state set includes terminated states
  - termination:
    - all processors in terminated states
    - no messages in transit

- **SPACE**
  - Count maximum total number of messages

Synchronous: count number of rounds until termination
Asynchronous: set unit of time as maximum message delay

The Problem

- Final states of processes partitioned in two classes:
  - elected
  - non-elected

- Once entered a state, always in that state

- In every admissible execution, exactly one process (the leader) enters an elected state. All remaining enter a non-elected state

Lots of variations...

- The ring can be unidirectional or bidirectional
- The number \( n \) of processors may be known or unknown
- Processors can be identical or can be somehow distinguished
- Communication may be synchronous or asynchronous
Uni- vs. Bidirectional

In unidirectional rings, messages can only be sent in a clockwise direction.

Can processors be distinguished?

If no, anonymous algorithms

• Processors have no UID
• Formally: identical automata
• Can distinguish between left and right.

If yes:

• Processors have unique IDs
• Chosen from some large totally ordered space of IDs (e.g. \(\mathbb{N}\))
• No constraint on which ID are used (e.g. integers may not be consecutive)
• IDs can be otherwise manipulated only by certain operations (e.g. comparison)
• Or by unrestricted operations

Is \(n\) known?

If no, uniform algorithms

• Algorithm cannot use information about ring size
Communication: 
Asynchronous vs. Synchronous

<table>
<thead>
<tr>
<th>Asynchronous</th>
<th>Synchronous</th>
</tr>
</thead>
<tbody>
<tr>
<td>• no upper bound on message delivery time</td>
<td>• communication in rounds</td>
</tr>
<tr>
<td>• no centralized clock</td>
<td>• In a round a process:</td>
</tr>
<tr>
<td>• no bound on relative speed of processes</td>
<td>– delivers all pending messages</td>
</tr>
</tbody>
</table>
| | – takes an execution step (which may involve sending one or more messages)

if no failures, every message sent is eventually delivered

An Impossibility Result

Theorem
There is no deterministic solution to the leader election problem for a synchronous, non-uniform, anonymous bidirectional ring.

Proof
Suppose that a solution exists for a system $A$ of $n > 1$ processes.
Each process of $A$ starts in the same state.

Lemma The states of all processors at the end of each round of the execution of $A$ are the same.

Proof By induction on number of rounds $k$:
• Base case: $k = 0$
  Easy, since processes start in same state.
• Inductive step: Lemma holds for $k = t-1$:
  – processors are identical up to round $t = 1$
  – send same messages to left and right neighbors
  – every processor receives identical messages on left and right channel
  – all processors apply same transition function to identical states in round $t$
  – all processors have identical states at the end of round $t$

Then, if one enters leader state, all do!

Observations

• What are the implication for asynchronous rings?

• What are the implication for uniform rings?