Leader Election in Rings with Bounded Multiplicity (Short Paper)

Karine Altisen[1], Ajoy K. Datta[2], Stéphane Devismes[1], Anaïs Durand[1], and Lawrence L. Larmore[2]

[1] Université Grenoble Alpes, Grenoble, France, Firstname.Lastname@imag.fr
[2] UNLV, Las Vegas, USA, Firstname.Lastname@unlv.edu

Abstract. We study leader election in unidirectional rings of homonym processes that have no *a priori* knowledge on the number of processes. We show that message-terminating leader election is impossible for any class of rings $\mathcal{K}_k$ with bounded multiplicity $k \geq 2$. However, we show that process-terminating leader election is possible in the sub-class $\mathcal{U}^* \cap \mathcal{K}_k$, where $\mathcal{U}^*$ is the class of rings which contain a process with a unique label.

1 Introduction

We consider *deterministic leader election in unidirectional rings of homonym processes*. The model of homonym processes [1, 3] has been introduced as a generalization of the classical fully identified model. Each process has an identifier, called here *label*, which may not be unique. Let $\mathcal{L}$ be the set of labels present in a system of n processes. Then, $|\mathcal{L}| = 1$ (resp., $|\mathcal{L}| = n$) corresponds to the fully anonymous (resp., fully identified) model.

Related Work. Homonyms have been mainly studied for solving the consensus problem in networks where processes are subjected to Byzantine failures [1]. However, Delporte *et al* [2] have recently considered the leader election problem in *bidirectional rings* of homonym processes. They have given a necessary and sufficient condition on the number of distinct labels needed to design a leader election algorithm. Precisely, they show that there exists a deterministic solution for *message-terminating* (*i.e.*, processes do not terminate but only a finite number of messages are exchanged) leader election on a bidirectional ring if and only if the number of labels is strictly greater than the greatest proper divisor of n. Assuming this condition, they give two algorithms. The first one is message-terminating and does not assume any further extra knowledge. The second one assumes the processes know n, is process-terminating (*i.e.*, every process eventually halts), and is asymptotically optimal in messages. In [3], Dobrev and Pelc investigate a generalization of the process-terminating leader election in both bidirectional and unidirectional rings of homonym processes. In their model, processes *a priori* know a lower bound m and an upper bound M on the (unknown) number of processes n. They propose algorithms that decide whether the election is possible and perform it, if so. They give synchronous algorithms for bidirectional and unidirectional rings working in time $O(M)$ using

$O(n \log n)$ messages. They also give an asynchronous algorithm for bidirectional rings that uses $O(nM)$ messages, and show that it is optimal; no time complexity is given.

Contribution. We explore the design of *process-terminating* leader election algorithms in unidirectional rings of homonym processes which, contrary to [2, 3], know neither the number of processes n, nor any bound on it. We study two different classes of unidirectional rings with homonym processes, denoted by $\mathcal{U}^*$ and $\mathcal{K}_k$. $\mathcal{U}^*$ is the class of all ring networks in which at least one label is unique. $\mathcal{K}_k$ is the class of all ring networks where no label occurs more than k times, so k is an *upper bound on the multiplicity* of the labels. We prove that there are no message-terminating leader elections for any class $\mathcal{K}_k$ with $k \geq 2$ despite processes know k, since $\mathcal{K}_k$ includes symmetric labeled rings. However, we give a process-terminating leader election algorithm for the sub-class $\mathcal{U}^* \cap \mathcal{K}_k$. Interestingly, there are labeled rings (*e.g.*, a ring of three processes with labels 1, 2, and 2) for which we can solve process-terminating leader election, whereas it cannot be solved in the model of [2, 3].

2 Preliminaries

Ring Networks. We assume unidirectional rings of $n \geq 2$ processes, $p_1, \ldots, p_n$, operating in asynchronous message-passing model, where links are FIFO and reliable. p_i can only receive messages from its *left* neighbor, p_{i-1}, and can only send messages to its *right* neighbor, p_{i+1}. Subscripts are modulo n.

We assume that each process p has a *label*, $p.id$; labels may not be distinct. For any label ℓ in the ring R, let $mlty[\,\ell\,] = |\{p : p.id = \ell\}|$, the *multiplicity* of ℓ in R. Comparison is the only operator permitted on labels.

Leader Election. An algorithm ALG solves the *message-terminating leader election* problem, noted MT-LE, in a ring network R if every execution of ALG on R satisfies the following conditions:

1. The execution is finite.
2. Each process p has a Boolean variable $p.isLeader$ s.t. when the execution terminates, $L.isLeader$ is TRUE for a unique process (*i.e.,* the leader).
3. Every process p has a variable $p.leader$ s.t. when the execution terminates, $p.leader = L.id$, where L satisfies $L.isLeader$.

An algorithm ALG solves the *process-terminating leader election* problem, noted PT-LE, in a ring network R if it solves MT-LE and satisfies the following additional conditions:

4. $p.isLeader$ is initially FALSE and never switched from TRUE to FALSE: each decision of being the leader is irrevocable. Consequently, there should be at most one leader in each configuration.

5. Every process $p \in R$ has a Boolean variable *p.done*, initially FALSE, such that *p.done* is eventually TRUE for all p, indicating that p knows that the leader has been elected. More precisely, once *p.done* becomes TRUE, it will never again become FALSE, $L.isLeader$ is equal to TRUE for a unique process L, and $p.leader$ is permanently set to $L.id$.
6. Every process p eventually *halts* (local termination decision) after *p.done* becomes TRUE.

Ring Network Classes. An algorithm ALG is MT-LE (resp., PT-LE) *for the class of ring network* $\mathcal{R}$ if ALG solves MT-LE (resp., PT-LE) for every network $R \in \mathcal{R}$. It is important to note that, for ALG to be MT-LE (resp., PT-LE) for a class $\mathcal{R}$, ALG cannot be given any specific information about the network (such as its cardinality) unless that information holds for all members of $\mathcal{R}$, since we require that ALG works for every $R \in \mathcal{R}$ without any change in its code.

We consider two main classes of ring networks. $\mathcal{U}^*$ is the class of all ring networks in which at least one label is unique. $\mathcal{K}_k$ is the class of all ring networks such that no label occurs more than k times, where $k \geq 1$.

3 Impossibility Result

A labeled ring network R is *symmetric* if it has a non-trivial rotational symmetry, *i.e.*, there is some integer $0 < d < n$ such that p_{i+d} and p_i have the same label for all i. In our model, it is straightforward to see that there is no solution to the leader election problem for a symmetric ring. Now, for any $k \geq 2$, $\mathcal{K}_k$ contains symmetric rings. Hence, follows.

Theorem 1. *For any $k \geq 2$, there is no algorithm that solves* MT-LE *for $\mathcal{K}_k$.*

4 Leader Election in $\mathcal{U}^* \cap \mathcal{K}_k$

For any $k \geq 2$, we give the algorithm U_k that solves PT-LE for the class $\mathcal{U}^* \cap \mathcal{K}_k$ (see Table 1). U_k always elects the process of minimum unique label to be the leader, namely the process L such that $L.id = \min\{x : mlty[x] = 1\}$. In U_k, each process p has the following variables.

1. *p.id*, constant of unspecified *label type*, the label of p.
2. *p.init*, Boolean, initially TRUE.
3. *p.active*, Boolean, which indicates that p is *active*. If $\neg p.active$, we say p is *passive*. Initially, all processes are active, and when U_k is done, the leader is the only active process. A passive process never becomes active.
4. *p.cnt*, an integer in the range $0 \ldots k+1$. Initially, $p.cnt = 0$. *p.cnt* will give to p a rough estimate of the frequency of its label in the ring.
5. *p.leader*, of label type. When U_k is done, $p.leader = L.id$.
6. *p.isLeader*, Boolean, initially FALSE, follows the problem specification. Eventually, $L.isLeader$ becomes TRUE and remains TRUE, while, for all $p \neq L$, *p.isLeader* remains FALSE for the entire execution.

7. *p.done*, Boolean, initially FALSE, follows the problem specification.

U_k uses only one kind of message. Each message is the forwarding of a *token* which is generated at the initialization of the algorithm, and is of the form $\langle x, c\rangle$, where x is the label of the originating process, and c is a *counter*, an integer in the range $0 \ldots k+1$, initially zero.

Table 1: Actions of Process p in Algorithm U_k

A1	$p.init$	$\to$	$\mathbf{send}\langle p.id, 0\rangle$ $p.init \leftarrow$ FALSE
A2	$\neg p.init \wedge \neg p.active \wedge \mathbf{rcv}\langle x, c\rangle \wedge x \neq p.id \wedge c \leq k$	$\to$	$\mathbf{send}\langle x, c\rangle$
A3	$\neg p.init \wedge p.active \wedge \mathbf{rcv}\langle x, c\rangle \wedge x \neq p.id \wedge (p.cnt = 0 \vee c > p.cnt)$	$\to$	$\mathbf{send}\langle x, c\rangle$
A4	$\neg p.init \wedge p.active \wedge \mathbf{rcv}\langle x, c\rangle \wedge x \neq p.id \wedge c < p.cnt$	$\to$	$\mathbf{send}\langle x, c\rangle$ $p.active \leftarrow$ FALSE
A5	$\neg p.init \wedge p.active \wedge \mathbf{rcv}\langle x, c\rangle \wedge x > p.id \wedge c = p.cnt \wedge c \geq 1$	$\to$	$\mathbf{send}\langle x, c\rangle$
A6	$\neg p.init \wedge p.active \wedge \mathbf{rcv}\langle x, c\rangle \wedge x < p.id \wedge c = p.cnt \wedge c \geq 1$	$\to$	$\mathbf{send}\langle x, c\rangle$ $p.active \leftarrow$ FALSE
A7	$\neg p.init \wedge \neg p.active \wedge \mathbf{rcv}\langle x, c\rangle \wedge x = p.id$	$\to$	(nothing)
A8	$\neg p.init \wedge p.active \wedge \mathbf{rcv}\langle x, c\rangle \wedge x = p.id \wedge c = p.cnt \wedge c \leq k-1$	$\to$	$\mathbf{send}\langle x, c+1\rangle$ $p.cnt \leftarrow c+1$
A9	$\neg p.init \wedge p.active \wedge \mathbf{rcv}\langle x, k\rangle \wedge x = p.id \wedge p.cnt = k$	$\to$	$\mathbf{send}\langle x, k+1\rangle$ $p.isLeader \leftarrow$ TRUE $p.leader \leftarrow p.id$ $p.done \leftarrow$ TRUE $p.cnt \leftarrow k+1$
A10	$\neg p.init \wedge \neg p.active \wedge \mathbf{rcv}\langle x, k+1\rangle$	$\to$	$\mathbf{send}\langle x, k+1\rangle$ $p.leader \leftarrow x$ $p.done \leftarrow$ TRUE (halt)
A11	$\neg p.init \wedge p.active \wedge \mathbf{rcv}\langle x, k+1\rangle \wedge x = p.id \wedge p.cnt = k+1$	$\to$	(halt)

Overview of U_k. The explanation below is illustrated by the example in Figure 1. The fundamental idea of U_k is that a process becomes passive, *i.e.,* is no more candidate for the election, if it receives a message that proves its label is not unique or is not the smallest unique label. Initially, every process initiates a token with its own label and counter zero (see (a)). No tokens are initiated afterwards. The token continually moves around the ring – every time it is forwarded, its counter and the local counter of the process are incremented if the forwarding process has the same label as the token (*e.g.*, Step (a)$\mapsto$(b)). Thus, if the message $\langle x, c\rangle$ is in a channel, that token was initiated by a process whose label is x, and has been forwarded c times by processes whose labels are also x. The token could also have been forwarded any number of times by processes with labels which are not x. Thus, the counter in a message is a rough estimate of the frequency of its label in the ring.

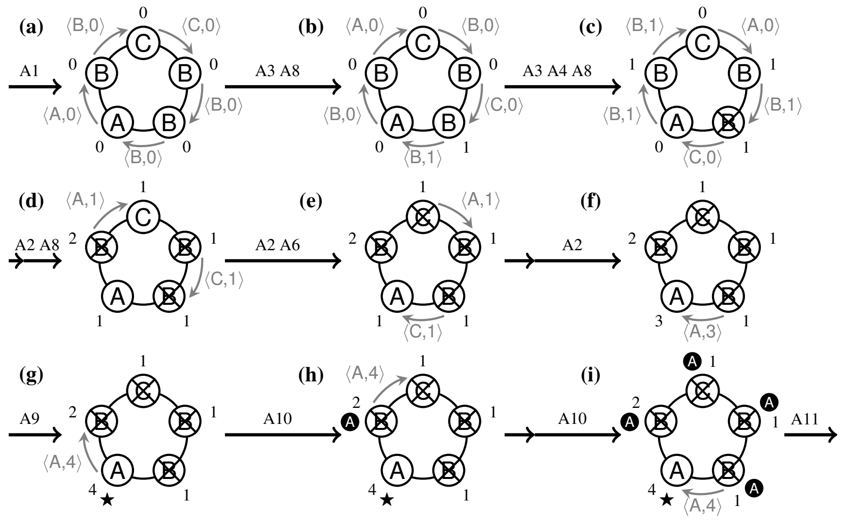

Fig. 1: Extracts from an example of execution of U_k where $k = 3$. The counter of a process is next to the corresponding node. Crossed out nodes are passive. $p.isLeader =$ TRUE if there is a star next to the node. The black bubble contains the elected label.

If a process receives a message whose counter is less than $p.cnt$, and $p.cnt \geq 1$, this proves its label is not unique since its counter grows faster than the one of another label. In this case, p executes Action A4 and becomes passive (*e.g.*, Step (b)↦(c)). Similarly, if a process p has a unique label but not the smallest one, it will become passive executing Action A6 when p receives a message with the same non-zero counter but a label lower than $p.id$ (*e.g.*, Step (d)↦(e)). In both cases, it happens at the latest when the process receives the message $\langle L.id, 1\rangle$, *i.e.,* before the second time L receives its own token.

So, after the token of L has made two traversals of the ring, it is the only surviving token (the others are consumed by Action A7) and every process but L is passive. The execution continues until the leader L has seen its own label return to it $k+1$ times, otherwise L cannot be sure that what it has seen is not part of a larger ring instead of several rounds of a small ring. Then, L designates itself as leader by Action A9 (see Step (f)↦(g)) and its token does a last traversal of the ring to inform the other processes of its election (*e.g.*, Step (g)↦(h)). The execution ends when L receives its token after $k+2$ traversals (see (i)).

References

1. Delporte-Gallet, C., Fauconnier, H., Guerraoui, R., Kermarrec, A., Ruppert, E., Tran-The, H.: Byzantine agreement with homonyms. Distributed Computing 26(5-6), 321–340 (2013)
2. Delporte-Gallet, C., Fauconnier, H., Tran-The, H.: Leader election in rings with homonyms. In: Networked Systems - 2nd International Conference, NETYS. pp. 9–24 (2014)
3. Dobrev, S., Pelc, A.: Leader election in rings with nonunique labels. Fundam. Inform. 59(4), 333–347 (2004)