USING (N–1)-PROCESS ELECTION TO SOLVE N-PROCESS ELECTION

James H. Anderson

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188

TR-89-10 March 1989

Using $(N-1)$-Process Election to Solve N-Process Election *

James H. Anderson
Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188

March 1989

Abstract

We present a solution strategy for N-process election in which a leader is chosen based upon the results of a number of $(N-1)$-process elections. We show that the existence of such a solution depends on the constraints that are placed on the $(N-1)$-process elections.

Keywords: election primitives, impossibility proof, leader election, knowledge-based reasoning, program composition, random assignment

CR Categories: D.4.1, F.1.2, F.3.1

1 Introduction

We consider the problem of electing a leader from among a set of $N \geq 2$ processes; this problem was first studied by LeLann in [5]. Our definition of the problem, which we call *N-ary election*, is adopted from [1], and is similar to the distributed consensus problem defined by Fischer, Lynch, and Patterson in [4]. We model election by requiring each process to assign a value, either 0 or 1, to a private "decision variable" — a process is "elected" iff it assigns 1 to its decision variable.

We propose to solve the N-ary election problem, where $N > 2$, by using $(N-1)$-ary election as a "primitive." In our proposed solution, each process executes in two phases. In its first phase, a process participates in a number of $(N-1)$-ary elections. In its second phase, a process assigns a value to its N-ary decision variable based upon the values of its

*Work supported in part by Office of Naval Research Contract N00014-86-K-0763.

$(N-1)$-ary decision variables. We show in Section 4 that N-ary election cannot be solved in this manner if the processes can randomly "choose" any outcome for the $(N-1)$-ary elections. Our proof uses knowledge-based reasoning, and makes use of several results from [2]. In Section 5, we give a solution in which some outcomes are prevented from occurring.

2 The N-ary Election Problem

We begin with some preliminaries. A *concurrent program* consists of two or more processes that access a set of variables. We assume that each process consists of ordinary sequential statements such as Dijkstra's guarded commands [3]. A *state* of a program is an assignment of values to the variables of the program; one state of a program is designated as its *initial state*. The semantics of a program is defined by its computations. A *computation* is a sequence of states $s_0 s_1 \cdots s_k$ such that s_0 is the initial state of the program, and for each i, where $0 \leq i < k$, s_{i+1} is the result of executing some statement of the program at state s_i. The last state s_k is called a *final state* if $s_0 s_1 \cdots s_k$ is a proper prefix of no computation. A computation is *complete* if it ends with a final state.

In the N-ary election problem, we are required to construct a concurrent program of $N \geq 2$ processes. Each process has a variable, called its *decision variable*, that is accessed by no other process. Each decision variable ranges over the set $\{\perp, 0, 1\}$ and is initially $\perp$. Each process assigns a value, either 0 or 1, to its decision variable so that the following conditions are satisfied.

- *Integrity*: In each final state, one of the decision variables has the value 1 and the rest have the value 0.
- *Equity*: For each process, there exists a final state in which that process's decision variable has the value 1.

We say that a process *wins* (respectively, *loses*) the N-ary election if it assigns the value 1 (respectively, 0) to its decision variable. Observe that if a process loses the N-ary election, where $N > 2$, then based upon its decision variable alone, it cannot determine which process wins. Thus, the N-ary election problem is not merely a restatement of the distributed consensus problem.

3 Proposed Solution

We propose to solve the N-ary election problem, for $N > 2$, by a program in which each process executes in two phases. In its first phase, a process participates in a number of $(N-1)$-ary elections. For each process, an $(N-1)$-ary election is held in which that process does not participate; thus, the total number of $(N-1)$-ary elections is N. In its second phase, a process assigns a value to its N-ary decision variable based upon the values of its

$(N-1)$-ary decision variables.

Notation: The N processes are denoted $0, \ldots, N-1$. Unless otherwise stated, the variables i, j, and k have the range $\{0, \ldots, N-1\}$. We call the $(N-1)$-ary election in which process j does not participate *election* j. We let $d.i.j$, where $i \neq j$, denote the $(N-1)$-ary decision variable for process i in election j, and let $d.i.i$ denote the N-ary decision variable for process i. □

In our proposed solution, process i has the following structure.

```
/* Phase 1 */
j := 0;
do j < N →
    if i = j → skip
    [] i ≠ j → d.i.j := ELECT(i, j)
    fi;
    j := j + 1
od;

/* Phase 2 */
d.i.i := decide(d.i.0, ..., d.i.(i−1), d.i.(i+1), ..., d.i.(N−1))
```

$ELECT$ is a procedure that returns either 0 or 1; we assume that $ELECT$ does not modify the variable j or any component of the array d. *decide* is a function that ranges over the set $\{0, 1\}$. Corresponding to the two conditions of the $(N-1)$-ary election problem, we require the procedure $ELECT$ to be defined so that the following conditions are met.

- *Phase 1 Integrity*: In each final state, the following assertion holds.

$$(\forall j :: (\exists i : i \neq j : d.i.j = 1 \land (\forall k : k \neq i \land k \neq j : d.k.j = 0))$$

  We call an assignment of values to the $(N-1)$-ary decision variables *valid* iff it satisfies the above assertion.

- *Phase 1 Equity*: For each valid assignment of values to the $(N-1)$-ary decision variables, there exists a final state in which that assignment occurs.

In the next section, we prove that the N-ary election problem cannot be solved as proposed above. The proof is based on the fact that, according to the Phase 1 equity condition, any valid assignment of values to the $(N-1)$-ary decision variables can be computed. In Section 5, we propose weaker Phase 1 integrity and equity conditions, and show that under these new constraints the N-ary election problem can be solved.

4 Impossibility Proof

Definition: For each valid assignment of values to the $(N-1)$-ary decision variables, we define a vector X of N components, denoted $X.0, \ldots, X.(N-1)$, where for each j, $X.j = i$ iff $i \neq j$ and $d.i.j = 1$. That is, the component $X.j$ denotes the winning process for election j. We call such a vector an *outcome*. □

Observe that, by the Phase 1 equity condition, any vector Y of N components is an outcome if for each j, $0 \leq Y.j < N$ and $Y.j \neq j$.

Two computations "look the same" to some process i if each of the variables $d.i.0, \ldots, d.i.(i-1), d.i.(i+1), \ldots, d.i.(N-1)$ is assigned the same value in both computations. We formalize this notion by defining a relation $[i]$ on the set of outcomes.

Definition: For outcomes X and Y, $X[i]Y \equiv (\forall k : k \neq i : X.k = i \Leftrightarrow Y.k = i)$. □

Thus, two computations "look the same" to process i if the outcomes that correspond to the two computations are related by $[i]$. Note that $[i]$ is an equivalence relation. The following properties follow from the definition of $[i]$.

Property 1: If outcomes X and Y differ only in the i^{th} component, then $X[i]Y$. □

Property 2: If outcomes X and Y differ only in the j^{th} component, and $j \neq i$, $X.j \neq i$, and $Y.j \neq i$, then $X[i]Y$. □

By the integrity condition for N-ary election, exactly one process assigns 1 to its N-ary decision variable in each complete computation. Let $f : \{\text{all outcomes}\} \rightarrow \{0, \ldots, N-1\}$ be the function that identifies the "winning process" for each outcome.

According to the equity condition for N-ary election, the function f satisfies the following restriction.

$$(\forall i :: (\exists X :: f(X) = i)) \tag{1}$$

If two computations of Phase 1 "look the same" to some process, then that process either wins the N-ary election for both computations, or loses the N-ary election for both computations. In other words, for any outcomes X and Y and each i,

$$(f(X) = i) \wedge (X[i]Y) \ \Rightarrow \ f(Y) = i \tag{2}$$

The following expression is an immediate consequence of (2).

$$(f(X) = i) \wedge (X[j]Y) \wedge (i \neq j) \ \Rightarrow \ f(Y) \neq j \tag{3}$$

The next theorem shows that $(N-1)$-ary election cannot be used to solve N-ary election as proposed in the previous section.

Theorem: There is no function $f : \{\text{all outcomes}\} \rightarrow \{0, \ldots, N-1\}$ that satisfies (1) and (2).

Proof: Suppose that f satisfies (1) and (2). We derive a contradiction by showing that there exists an outcome X for which f is undefined, i.e., for each i, $f(X) \neq i$. We treat the two cases $N = 3$ and $N > 3$ separately.

Case 1: Suppose that $N = 3$. By (1), there exist outcomes A, B, and C such that $f(A) = 0$, $f(B) = 1$, and $f(C) = 2$. We use A, B, and C to define four other outcomes D, E, F, and G. We then show that f is undefined for outcome G. The four outcomes are defined as follows.

$$\begin{aligned} D.0, D.1, D.2 &:= B.0, A.1, A.2 \\ E.0, E.1, E.2 &:= B.0, C.1, B.2 \\ F.0, F.1, F.2 &:= C.0, C.1, A.2 \\ G.0, G.1, G.2 &:= B.0, C.1, A.2 \end{aligned}$$

Note that A and D may differ only in the 0^{th} component; thus by Property 1, $A[0]D$. Therefore, by (2) with $X, Y, i := A, D, 0$, we have $f(D) = 0$. Similarly, we can show that $B[1]E$ and $C[2]F$; thus, by (2), we have $f(E) = 1$ and $f(F) = 2$.

Observe that F and G may differ only in the 0^{th} component; thus, by Property 1, $F[0]G$. Therefore, because $f(F) = 2$, by (3) with $X, Y, i, j := F, G, 2, 0$, we have $f(G) \neq 0$. We can also show that $D[1]G$ and $E[2]G$; thus, because $f(D) = 0$ and $f(E) = 1$, by (3), we have $f(G) \neq 1$ and $f(G) \neq 2$.

Case 2: Suppose that $N > 3$. Let A be the outcome where for each i, $A.i = i \oplus 1$ ($\oplus$ denotes modulo-N addition). We show that for every i, $f(A) \neq i \oplus 1$. This implies that for every i, $f(A) \neq i$.

To prove that $f(A) \neq i \oplus 1$, we show that there exists an outcome E such that the following assertion holds.

$$E.i = i \oplus 1 \ \wedge \ (\forall j : j \neq i : E.j \neq i \oplus 1) \ \wedge \ f(E) = i$$

Note that in both A and E, only the i^{th} component equals $i \oplus 1$. Therefore, $E[i \oplus 1]A$. Thus, by (3) with $X, Y, i, j := E, A, i, i \oplus 1$, we have $f(A) \neq i \oplus 1$. We show that E exists by considering three other outcomes B, C, and D.

By (1), for each i there exists an outcome B such that $f(B) = i$. Suppose that $B.j = i \oplus 1$ for some j, where $j \neq i$. Because $N > 3$, there exists m, where $0 \leq m < N$, such that $m \neq j$, $m \neq i$, and $m \neq i \oplus 1$. Let C be the outcome defined as follows.

$$C.j = m \ \wedge \ (\forall k : k \neq j : C.k = B.k)$$

Note that B and C differ only in the j^{th} component and that $B.j \neq i$ and $C.j \neq i$. Thus, by Property 2, $B[i]C$. Therefore, by (2) with $X, Y, i := B, C, i$, we have $f(C) = i$.

By repeating this argument, we see that there exists an outcome D such that $f(D) = i$, and for each j, where $j \neq i$, $D.j \neq i \oplus 1$. Let E be the outcome defined as follows.

$$E.i = i \oplus 1 \ \wedge \ (\forall j : j \neq i : E.j = D.j)$$

Observe that D and E may differ only in the i^{th} component; thus, by Property 1, $D[i]E$. Therefore, by (1) with $X, Y, i := D, E, i$, we have $f(E) = i$. □

The fact that we had to split the proof of the theorem into two cases is somewhat disconcerting. However, this dichotomy is the result of a fundamental difference between 2-ary election and N-ary election, where $N > 2$. In a 2-ary election, the process that loses can conclude that the other process wins. In an N-ary election, where $N > 2$, a process that loses cannot determine, based on its decision variable alone, which of the other processes wins. As a result of this difference, it seems necessary to treat separately the two cases $N = 3$ (in which case 2-ary election is used in the solution) and $N > 3$ (in which case $(N-1)$-ary election, where $N-1 > 2$, is used in the solution).

5 Making it Solvable

We now show that N-ary election can be solved as proposed in Section 3 if we weaken the Phase 1 integrity and equity conditions as follows.

- *Phase 1 Integrity*: In each final state, the following assertion holds.

$$(\forall i, j : i \neq j : d.i.j = 1 \Rightarrow (\forall k : k \neq i \wedge k \neq j : d.k.j \neq 1))$$

- *Phase 1 Equity*: For each i and j, where $i \neq j$, there exists a final state in which $d.i.j = 1$.

In our solution, the processes share N variables $z.0, \ldots, z.(N-1)$; each $z.j$ ranges over the set $\{0, 1\}$ and is initially 1. We define the procedure $ELECT$ as follows.

procedure $ELECT(i, j)$ **returns** x
begin
  if $(\forall k : 0 \leq k < j \wedge k \neq i : d.i.k = 1) \rightarrow x, z.j := z.j, 0$
  $[\!]$ $(\exists k : 0 \leq k < j \wedge k \neq i : d.i.k \neq 1) \rightarrow x := 0$
  fi
end

Thus, if a process loses some $(N-1)$-ary election, then it is "forced" to lose all subsequent elections. The function *decide* is defined as follows.

$$decide(\cdots) = \begin{cases} 1 & \text{if } (\forall k : 0 \leq k < 3 \wedge k \neq i : d.i.k = 1) \\ 0 & \text{otherwise} \end{cases}$$

In order to verify the correctness of our solution, we find it convenient to re-write process i as follows.

```
/* Phase 1 */
j := 0;
do j < N →
  if i = j → skip
  [] i ≠ j ∧ (∀k : 0 ≤ k < j ∧ k ≠ i : d.i.k = 1) → d.i.j, z.j := z.j, 0
  [] i ≠ j ∧ (∃k : 0 ≤ k < j ∧ k ≠ i : d.i.k ≠ 1) → d.i.j := 0
  fi;
  j := j + 1
od;

/* Phase 2 */
if (∀k : 0 ≤ k < 3 ∧ k ≠ i : d.i.k = 1) → d.i.i := 1
[] (∃k : 0 ≤ k < 3 ∧ k ≠ i : d.i.k ≠ 1) → d.i.i := 0
fi
```

To prove that our solution is correct, we first show that it satisfies the new Phase 1 integrity and equity conditions defined above, and then show that it satisfies the integrity and equity conditions for the N-ary election problem. The Phase 1 integrity condition is satisfied since, as the reader can check, the following assertion is an invariant.

$$(\forall j :: z.j + (\Sigma i : i \neq j \wedge d.i.j \neq \perp : d.i.j) = 1)$$

To see that Phase 1 equity is satisfied, observe that if some process finishes executing before any other process starts executing, then it assigns the value 1 to each of its $(N-1)$-ary decision variables. By the definition of *decide*, this also establishes the equity condition for the N-ary election problem.

We can prove that the integrity condition for the N-ary election problem is satisfied by proving that in every complete computation of the program (i) at most one process wins the N-ary election, and (ii) at least one process wins the N-ary election. For brevity, we merely sketch the proofs for (i) and (ii), and leave the formal details to the reader.

Proof of (i): Given any two processes, there exists an election j, where $0 \leq j < 3$, in which both processes participate. Therefore, by the definition of *decide* and Phase 1 integrity, both processes cannot assign the value 1 to their N-ary decision variables.

Proof of (ii): Because the assertion $(\forall k : 0 \leq k < 1 \wedge k \neq 0 : d.0.k)$ is vacuously true, process 0 executes the assignment $d.0.1, z.1 := z.1, 0$ in every complete computation. Thus, in every complete computation, some process wins election 1. Observe that process 2 wins election 1 only if it wins election 0. Thus, by the definition of *decide*, if process 2 wins election 1, then it wins the N-ary election.

If, on the other hand, some process i, where $i \neq 1$ and $i \neq 2$, wins election 1, then it establishes the assertion $(\forall k : 0 \leq k < 2 \wedge k \neq i : d.i.k = 1)$; consequently, it executes the assignment $d.i.2, z.2 := z.2, 0$. This implies that some process wins election 2. Observe that a process wins election 2 only if it wins each lower numbered $(N-1)$-ary election in which it participates. Thus, by the definition of *decide*, the process that wins election 2 also wins the N-ary election.

6 Concluding Remarks

The original Phase 1 integrity and equity conditions given in Section 3 are satisfied iff every $(N-1)$-ary election outcome can be computed. We have shown that this requirement is too strong, making it impossible to solve the N-ary election problem as proposed. On the other hand, the weaker Phase 1 integrity and equity conditions given in Section 5 allow us to coordinate the $(N-1)$-ary elections, thereby preventing some outcomes from being computed. We have shown that in this case it is possible to solve the N-ary election problem as proposed. In the solution given, for example, no outcome in which process 2 loses election 0 and wins election 1 can be computed.

The impossibility proof of Section 4 can be generalized by allowing any number of $(N-1)$-ary elections in Phase 1. Although no major changes are required for the proof, the notation becomes a bit cumbersome. For example, in the general case each component of an outcome is a set of values instead of a single value.

Acknowledgements: I would like to thank Anish Arora, Ken Calvert, Mohamed Gouda, Jay Misra, Ambuj Singh, and the members of the UT Distributed Systems Discussion Group for their comments on this paper.

References

[1] J. Anderson and M. Gouda, The virtue of patience: concurrent programming with and without waiting, submitted for publication.

[2] K. Chandy and J. Misra, How processes learn, *Distributed Computing*, Vol. 1, No. 1, 1986, pp. 40-52.

[3] E. Dijkstra, *A Discipline of Programming*, Prentice-Hall, Englewood Cliffs, New Jersey, 1976.

[4] M. Fischer, N. Lynch, and M. Patterson, Impossibility of distributed consensus with one faulty process, *Journal of the ACM*, Vol. 32, No. 2, April 1985, pp. 374-382.

[5] G. LeLann, Distributed systems — Towards a formal approach, *Information Processing 77*, North-Holland Pub. Co., New York, 1977, pp. 155-160.