THE LORD OF THE RING
or
PROBABILISTIC METHODS
FOR BREAKING SYMMETRY IN DISTRIBUTIVE NETWORKS

by
Alon Itai and Michael Rodeh

Technical Report #253

November 1982

1. IBM Israel Scientific Center, Technion City, Haifa, Israel. Part of this work was conducted while visiting IBM Research, San Jose Laboratory, San Jose, CA 95193.
THE LORD OF THE RING

or

PROBABILISTIC METHODS

FOR BREAKING SYMMETRY IN DISTRIBUTIVE NETWORKS

Alon Itai
Michael Rodeh

ABSTRACT

Given a ring of \( n \) processes it is required to design the processes such that they will be able to choose a leader (a uniquely designated process) by sending messages along the ring. If the processes are indistinguishable then a deterministic algorithm to solve the problem does not exist. To overcome this difficulty, probabilistic algorithms are proposed. The algorithms may run forever but they terminate within finite time on the average. For the synchronous case several algorithms are presented: The simplest requires on the average that no more than \( 2.442n \) bits are transmitted and \( O(n) \) time elapses until the algorithm terminates, while higher order algorithms trade in time for reducing the communication complexity. If the processes work asynchronously then on the average \( O(n \log^2 n) \) bits are transmitted. In the above cases the size of the ring was assumed to be known to all the processes. If the size is not known then finding it may be done only with

---

(1) IBM Israel Scientific Center, Technion City, Haifa, Israel. Part of this work was conducted while visiting IBM Research, San Jose Laboratory, San Jose, CA. 95193
probability: any algorithm may yield incorrect results (with nonzero probability) for some values of \( n \). Another difficulty is that, if we insist on correctness, the processes may not explicitly terminate. Rather, the entire ring reaches an inactive state, in which no process initiates communication.
1. INTRODUCTION

Given a network of \( n \) processes it is required to design the processes to solve some network problems, such as constructing a spanning tree. This task is easier if the processes are distinguishable (i.e., by having unique names). However, if all processes with the same number of neighbors are identical then the problem becomes symmetric and computing global network functions becomes harder.

To demonstrate our ideas, we consider a ring, a cycle of \( n \) processes, on which we shall compute two global functions:

(i) Choosing a leader - a uniquely designated process, such that each process knows whether it is the leader;

(ii) Finding \( n \) - the size of the ring.

Angluin [A] has investigated the problem of choosing a leader using Milne and Milner's [MM] model for distributed systems. She has shown that there exists no process, the allocation of which to all the nodes of a ring of arbitrary size, yields a network which is able to choose a unique leader within finite time. The fundamental phenomenon is that symmetry cannot be broken without allowing either an infinite computation or an erroneous result. Global information (such as knowing the size of the ring) does not always help.

Since either termination or correctness must be compromised, we shall construct probabilistic algorithms [R1] - algorithms which usually terminate with a correct result. Thus, we assume that the processes, even though identical, each have independent random number generators. (For a use of probabilistic algorithms for symmetry breaking in a shared memory environment, see [R2].) Our theme is that to break symmetry one can develop a probabilistic routine to suggest and improve solutions together.
with a routine to test the correctness of the proposed solution. Obviously, we are interested in efficient algorithms, and we shall try to minimize the communication complexity - the number of bits transmitted.

The problem of choosing a leader is an example of a case where correctness is achievable. This is proven in Section 3: several algorithms for choosing a leader are proposed for a synchronous model in which the size of the ring is known. Then it is shown how to apply these ideas the asynchronous case.

The problem of finding $n$, the size of the ring, is discussed in Section 4. This problem is directly connected with that of termination - the cessation of all communications. If there exists a process which is able to sense termination, then the termination is distributed. If each process only knows that it will not initiate any communication (unless receiving some message) then we have the weaker notion of global termination. Global termination can be detected only by examining all the processes. Only if we know that $N < n < N$ for some $N$, there exists an algorithm to find $n$ which terminates distributed. Otherwise, we must resort to the weaker notion of global termination, for which an algorithm is presented. It terminates within polynomial time and yields a correct result with high probability.

It should be mentioned that the notion of distributive termination was discussed in ([DS], [F], [FR1], [FR2], [FRS]), however in these papers asymmetry was assumed.

We now turn to describe the model of computation and the notions of correctness and termination.
2. THE MODEL OF COMPUTATION

In this section the notion of a network of asynchronous probabilistic nondeterministic processes is defined and correctness criteria are discussed. The main difficulty is to distinguish between randomization and the nondeterminism introduced both at the process level and the level of the communication scheduling.

Let $Z_m = \{1, \ldots, m\}$ and $Z = \{1, 2, \ldots\}$. Also let $A$ be a finite communication alphabet and $d \in Z$. A $d$-process is a pair $(Q_d, M_d)$ where $Q_d$ is a nonempty (possibly infinite) set of states and $M_d$ is a mapping from $Q_d \times Z_d \times Z_2$ to finite subsets of $Q_d \times A \times A$. Each process has $d$ ports through which it communicates. The meaning is that if $(q', a, b) \in M_d(q, p, x)$ then:

(i) The process is in state $q$ and is willing to exchange $a$ for $b$ through port $p$, one of the $d$ ports of the process.

(ii) $q'$ is the next state of the process.

(iii) $x \in Z_2$ is the "random" element which influences the transition. The method of obtaining these values will be discussed below. An obvious extension is to substitute $Z_1$ for $Z_2$.

Notice the inherent nondeterminism introduced by using set-valued transition functions. No probabilistic assumptions concerning this type of nondeterminism are made.

A network of processes is a pair $(G, P)$ where:

(i) $G$ is a finite connected undirected graph with neither self-loops nor parallel edges. The edges incident with each node are uniquely numbered. Each process associates a port with each one of the edges incident with the node.
(ii) $P$ is a sequence $P_1, P_2, \ldots$ where $P_d$ is a $d$-process.

(iii) $P_d$ is assigned to all the nodes of degree $d$.

An instantaneous description of a network is a map $I$ which assigns to each node $v$ an element of $Q_d(v) \times \mathbb{Z}$. $I(v) = (q(v), s(v))$ means that the process $P(v)$ is in state $q(v) \in Q_d(v)$ and has completed $s(v)$ computation steps.

Let $\rho$ be a mapping which assigns to every node $v$ of $G$ an infinite sequence of elements of $Z_d$. Let $I^1$ and $I^2$ be instantaneous descriptions, then $I^1$ yields $I^2$ via $\rho$ if there exists an edge $(v_i, v_{i+1})$ and $a_1, a_2 \in A$ such that for $i=1,2$:

(i) For every node other than $v_1$ and $v_2$, $I^1(u) = I^2(u)$;

(ii) $s^2(u) = s^1(u) + 1$

(iii) $(q^2(u), a_1, a_{2-1}) \in M_d(v)(q^1(u), p_i, r_i)$

where

(i) $I^1(u) = (q^1, s^1)$; $I^2(u) = (q^2, s^2)$;

(ii) $d(v_i)$ is the degree of $v_i$;

(iii) the edge connecting $v_i$ and $v_j$ is the $p_i$-th port of $v_i$;

(iv) $r_i$ is the $s^1(v_i)$-th element of $\rho(v_i)$.

Initially, the node $v_i$ chooses at random the element $r_i$. (The step numbers were introduced to enable the processes to choose new values of the "random sequences" $\rho(v_i)$. All the step numbers are initially zero.) After choosing $r_i$ the process is willing to exchange $a_1$ for $a_{2-1}$ with its neighbor $v_{a-1}$. Consequently, $v_i$ changes its state to $q^2(v_i)$ and its step number is increased by one.

An algorithm is a pair $(N, I)$ where $N$ is a network and $I$ is an initial instantaneous description (all step numbers are zero). We will be interested
only in uniform initial instantaneous descriptions - those which assign the same state to all nodes with the same degree. The corresponding algorithms will also be called uniform.

Let \((N,I)\) be an algorithm. A \(\rho\)-computation is a sequence of instantaneous descriptions \(I = \{I^1, I^2, \ldots\}\) such that \(I^i\) yields \(I^{i+1}\) via \(\rho\). If the sequence is finite then the last instantaneous description does not yield any other instantaneous description via \(\rho\).

An algorithm is \(\rho\)-terminating if every \(\rho\)-computation is finite. An algorithm is \(\rho\)-partially correct (with respect to a post-condition \(\psi\)) if \(\psi\) holds for the last instantaneous description of every finite \(\rho\)-computation. The post-condition \(\psi\) is not mentioned when it is understood from the context.

An algorithm terminates (is partially correct) if it is \(\rho\)-terminating. (\(\rho\)-partially correct) for all \(\rho\). An algorithm is correct if it is both terminating and partially correct.

The set of all \(\rho\)'s (with uniform probability measure) constitutes our probability space. Let \(0 < \rho < 1\), then an algorithm terminates (is partially correct) with probability \(\rho\) if the probability is at least \(\rho\) that the algorithm \(\rho\)-terminates (is \(\rho\)-partially correct).

For every sequence \(\rho\) let \(C(\rho)\), the communication complexity, be the length of the longest \(\rho\)-computation. The average communication complexity is the average of \(C(\rho)\) over all sequences \(\rho\).

Termination is a global condition, it occurs when no communication is possible. However, in general, inspecting each node separately does not suffice to conclude that the node will not participate in any communication sometime in the future. Therefore, we define a stronger notion - that of distributive termination: A state \(q_d \in Q_d\) is terminal if for all \(i \in Z_d\) and \(x \in Z_2\), \(M_d(q_d, i, x) = \phi\). An instantaneous description is distributively terminal if all
its nodes are in terminal states. An algorithm terminates distributively if every finite \( \rho \)-computation ends at a distributively terminal instantaneous description. Notice that an algorithm which terminates distributively may sometimes diverge. In Section 4.1 it is proved that there is no distributively terminating algorithm for finding the size of the ring. While in Section 4.3 a globally terminating algorithm is given, thus demonstrating that distributive termination is indeed stronger than global termination.

Another model is that of synchronous processes. We shall avoid detailed definitions, only state that the computation proceeds in time quanta. At each quantum all adjacent processes can exchange messages concurrently. Thus, a process can send and receive messages simultaneously.
3. CHOOSING A LEADER WHEN $n$ IS KNOWN AND THE PROCESSES ARE
INDISTINGUISHABLE

3.1 Preview

Angluin [A] has shown that there does not exist a correct uniform algo­rithm to find a leader of the ring. Her argument considers a ring of four processes where the antipodal processes are always in the same state. When a process decides that it is the leader, then its image comes to the same conclusion, thus either two leaders are chosen or the algorithm does not terminate.

If $n$ is known then there is no need to consider an algorithm which may yield incorrect results since it can be transformed to a partially correct algorithm that terminates in all cases for which the original algorithm does. An algorithm might terminate incorrectly either by

(i) Selecting more than one leader, or

(ii) Choosing no leader.

To overcome the first difficulty, whenever a leader is chosen, it sends a verification message along the ring. If additional candidates for being a leader are found then the algorithm is reinvoked. To distinguish between the various invocations, phase numbers may be added.

To overcome the second difficulty, the algorithm is reinvoked whenever deadlock is detected. (Again, phase numbers may be used.) We now describe a general method to distributively detect deadlocks.

The deadlock detection mechanism sends its own messages, in addition to the algorithm's original messages. The processes must always be willing to accept and pass these messages, so that the deadlock detection mechanism itself does not get deadlocked.
Each process, immediately after being involved in a regular, communication, initiates a deadlock detection message, which traverses the ring until finding a pair willing to exchange an original message. If a deadlock detection message completes an entire cycle, the process that sent this message deduces that the algorithm is deadlocked.

Notice that this deadlock detection scheme is applicable to any ring, provided that each process can identify its own messages.

When deadlock is detected, some corrective measure can be taken, such as releasing resources, or as in our case (choosing a leader) reinvoking the entire algorithm.

The two transformations described above replace incorrect termination by reinvocation of the algorithm. However, if the original algorithm terminates with probability 1 and is partially correct with probability \( p > 0 \) then the modified algorithm is partially correct and its probability of termination is still 1. To see this property of the modified algorithm, notice that the probability that the original algorithm be invoked exactly \( k \) times is 
\[
p(1 - p)^{k-1}
\]
and thus the average number of invocations is \( 1/p \) which proves that the modified algorithm has probability 0 to diverge.

### 3.2. The synchronous case - a first order algorithm to choose a leader.

The algorithm proceeds in phases. Each phase consists of \( n \) time units. In the \( i \)th phase there are \( n_i \) active processes, each of which randomly chooses an integer between 1 and \( n_i \). To start, \( n_1 = 1 \). Let \( k_i \) denote the number of processes which chose the integer 1. Each of these \( k_i \) processes sends a pebble around the ring. After \( n \) time units, the \( k_i \) pebbles return to their originators. By counting the number of pebbles which passed through each process, each of the processes calculates \( k_i \). If \( k_i = 1 \) then the process
that chose the integer one declares itself a leader (all the others know that a leader has been chosen). If $k_i = 0$ then this phase has been useless and all active processes continue to the next phase. Otherwise $1 < k_i \leq n_i$ and the $k_i$ processes that chose the number one are the active processes of the next phase. Thus,

$$n_{i+1} = \begin{cases} n_i & k_i = 0; \\ k_i & k_i > 0 \end{cases}$$

Assume that each one of $m$ active processes chooses an integer in the range $[1,m]$ with probability $1/m$. Let $p(m,j)$ denote the probability that $j$ out of $m$ active processes choose the number one. Then,

$$p(m,j) = \binom{m}{j} \left( \frac{1}{m} \right)^j \left( 1 - \frac{1}{m} \right)^{m-j} = \left[ 1 - \frac{1}{m} \right]^{m} \left( \frac{m}{j} \right) / (m-1)^j.$$

Let $l(m)$ denote the expected number of phases required to reduce the number of active processes from $m$ to one. Then $l(m)$ satisfies the relation

$$l(m) = 1 + p(m,0)l(m) + \sum_{k=1}^{m-1} p(m,k)l(k).$$

By definition, $l(1) = 0$. Thus,

$$l(m)(1-p(m,0)-p(m,m)) \geq 1 + \sum_{k=2}^{m-1} p(m,k)l(k). \quad (3.1)$$

Let $d(m) = 1-p(m,0)-p(m,m)$. Then, $l(m) = (1 + \sum_{k=2}^{m-1} p(m,k)l(k))/d(m)$.

From its definition, $l(m)$ increases. We now show that it is bounded. By (3.1)

$$l(m)(1-p(m,0)-p(m,m)) \leq 1 + l(m)(1-p(m,0)-p(m,m)-p(m,1)).$$

Thus,

$$l(m) \leq 1/p(m,1) = \left[ \left( 1 - \frac{1}{m} \right)^m \frac{m}{m-1} \right]^{-1} < \hat{c}.$$  

We see that $l(m)$ is increasing and bounded and therefore it approaches a limit, $l(\infty) \leq \hat{c}$.  

To calculate $l(\infty)$ we shall first consider the probabilities $p(m,k)$.

**Lemma 3.1:** For fixed $k \geq 3$, $p(m,k)$ increases with $m$.

**Proof:** By induction on $k$.

**Base:** $k = 3$. We have,

\[
p(m,3) = \frac{m}{3} \left(1 - \frac{1}{m}\right)^m \chi(m-1)^3
\]

\[
= \frac{1}{6} \left(1 - \frac{1}{m}\right)^m \frac{m(m-1)(m-2)}{(m-1)^3}
\]

\[
= \frac{1}{6} \left(1 - \frac{1}{m}\right)^m \left[1 - \frac{1}{(m-1)^2}\right]
\]

which clearly increases with $m$.

**Induction step:**

\[
p(m,k+1) = \frac{m}{k+1} \left(1 - \frac{1}{m}\right)^m \chi(m-1)^{k+1}
\]

\[
= \frac{1}{k+1} p(m,k) \left[1 - \frac{1}{m}\right]^{k+1}
\]

By the induction hypothesis $p(m,k)$ increases and so does the third multiplicand.

Since $p(m,k) \leq 1$ then for $k \leq 3$, $p(\infty,k) = \lim_{m \to \infty} p(m,k)$ exists. Moreover, $p(\infty,k)$ is equal to $e^{k-1}/k!$. It is also easy to see that $p(\infty,2) = \lim_{m \to \infty} p(m,2) = e^{-1}/2$. The sequence $d(m)$ also converges, and $d(\infty) = \lim_{m \to \infty} d(m) = 1 - e^{-1}$.

Now, define $b(m,j) = \sum_{k=3}^{j} l(k)p(m,k)$. From Lemma 3.1, for fixed $j$, $b(m,j)$ increases and is bounded by $l(\infty)$. Thus, $\lim_{m \to \infty} b(m,j)$ exists and is equal to $b(\infty,j) = \sum_{k=3}^{j} l(k)p(\infty,k)$.

Let $L(m,j)$ be a truncated approximation to $l(m)$. I.e.,
\[
L(m,j) = \left(1 + \sum_{k=2}^{j} p(m,k)l(k)\right)/d(m)
\]
\[
= \left(1 + p(m,2)l(2) + b(m,j)\right)/d(m).
\]

Thus,
\[
L(\infty,j) = \lim_{m \to \infty} L(m,j) = \left(1 + p(\infty,2)l(2) + b(\infty,j)\right)/d(\infty).
\]

\(L(\infty,j)\) is easily calculated, and is a very good approximation to \(l(\infty)\):

**Lemma 3.2:** \(0 \leq l(\infty) - L(\infty,j) < e / (j!(e-1))\).

**Proof:**
\[
0 \leq l(\infty) - L(\infty,j) = \left(\sum_{k=j+1}^{\infty} p(\infty,k)l(k)\right)/d(\infty)
\]
\[
< \frac{1}{d(\infty)} \sum_{k=j+1}^{\infty} \frac{1}{k!} < \frac{1}{j!d(\infty)}.
\]

Thus, \(l(\infty)\) can be calculated up to 10 digits using only the 13 values of \(l(2), \ldots, l(14)\). In fact \(l(\infty) \approx 2.441415\). To summarize we have:

**Theorem 3.1:** A leader of a ring of \(n\) synchronous processes which know \(n\), may be chosen in approximately \(2.441415n\) time on the average.

**Theorem 3.2:** Under the conditions of Theorem 3.1, The communication complexity is at most \(2.441415n\) bits.

**Proof:** Since the probability to choose 1 is \(1/m\),
\[
\sum_{k=0}^{m} kp(m,k) = m(1/m) = 1.
\]
Thus the expected communication complexity per phase is \(n\). Since there are at most \(2.441415\) phases on the average, the theorem follows.

### 3.3 The synchronous case - higher order algorithms to choose a leader.

In the above algorithm we only count the number of pebbles. Lempel [Lem] suggested to
inspect the circular Boolean pattern created by the pebbles, and to choose as leader the process with lexicographically largest pattern. If there exist several such processes, rerun the algorithm. Notice that all the processes are active, until a leader is chosen.

Let \( c \) be a constant, and assume that the processes decide to transmit a pebble with probability \( 1 / c \). Then the following results are obtained:

**Theorem 3.3:** In the improved algorithm the expected number of phases converges to \( (1 - e^{-c})^{-1} \) and the expected communication cost per process converges to \( c(1 - e^{-c})^{-1} \).

To prove this theorem we need the following lemma:

**Lemma 3.3:** The probability that a phase is useless converges to \( e^{-c} \) as \( n \) increases.

**Proof:** Two disjoint events contribute to the probability that a phase is useless:

(i) No pebble is sent. The probability of this is \( (1 - c/n)^n \).

(ii) Some pebbles are sent but the resultant word is periodic.

The probability of the first event is \( (1 - c/n)^n \) which converges to \( e^{-c} \).

In order to analyze the probability of the second event, we denote the probability to have \( j \) pebbles by \( P_c(n,j) \) and the probability that a word with \( j \) ones is periodic by \( q(n,j) \), the contribution of this event is

\[
\sum_{j=2}^{n} P_c(n,j)q(n,j). \tag{3.2}
\]

where \( P_c(n,j) = \binom{n}{j} \left( \frac{c}{n} \right)^j \left( 1 - \frac{c}{n} \right)^{n-j} \)

For \( j \geq \sqrt{n} / 2 \), \( P_c(n,j)/P_c(n,j + 1) > 2 \) and therefore (3.2) is bounded by
To overcome these difficulties, \( p_c(n, \sqrt{n}/2) \) tends to zero as \( n \) increases therefore we must prove that the summation also tends to zero. To this end, let us estimate \( q(n,j) \) for \( j \leq \sqrt{n}/2 \). Consider a word of length \( n \) with \( j \) ones in it. If it consists of \( d \) repetitions of some smaller word, then the number of such words is at most

\[
\frac{n[n/d]}{d^j/j/d} \quad (3.4)
\]

where \( n/d \) is the number of possible circular shifts of the word and the binomial coefficient is the number of words with length \( n/d \) and \( j/d \) bits equal to 1. Thus, the number of circular words with \( j \) ones is bounded by:

\[
\sum_{d>1; j=d|n} \frac{n[n/d]}{d^j/j/d} \quad (3.5)
\]

Since \( j \leq \sqrt{n}/2 \), (3.5) is bounded by \( \frac{n[n/2+1]}{2^{j/2+1}} \). Therefore,

\[
q(n,j) \leq \frac{n[n/2+1]}{2^{j/2+1}} \quad (3.6)
\]

A direct check of (3.4) shows that \( q(n,2) \) tends to zero as \( n \) increases, while

\[
\sum_{j=2}^{\sqrt{n}/2} q(n,j)
\]

tends to zero too, by (3.6). Therefore, (3.3) converges to zero and the lemma is proved.

As for the space complexity, each process needs \( O(n) \) space to compute the lexicographically maximal word. However, the number of pebbles generated in a phase is relatively small. Thus, one can keep track only of the zero-runs, and thereby reduce the space complexity. By allowing at most \( d \) pebbles (and otherwise declaring a phase useless) the time and communication complexity are increased only slightly, while the space complexity is reduced to \( O(d \log n) \).
The above results exhibit the tradeoff between time and communication: In order to reduce the communication, more time must be spent. For $c = 1$ the expected number of phases is approximately 1.582 and this is also the communication cost per process.

3.4. The asynchronous case

In an asynchronous setup two difficulties arise:

(i) No process may decide on its own to become inactive without being sure that some other processes remain active (otherwise the entire ring may run into Nirvana - all processes become inactive without choosing a leader).

(ii) A process is unable to count time steps in order to deduce that a message it initiated has returned.

However, a combination of the former algorithm and that of Chang and Roberts [CR] is developed. During the execution sequence of the algorithm some processes are designated active. Only an active process can become the leader. Initially all processes are active.

Each process $v$ has the following variables:

(i) $active_v$ - a Boolean variable indicating whether $v$ is active.

(ii) $ph_v$ - the phase number (initially zero).

(iii) $id_v$ - an integer identification number between 1 and $n$. At the beginning of each phase, each process chooses $id_v$ at random. Notice that there may be more than one process with the same identification number.

(iv) $buf_v$ - a buffer containing at most one message.
Active processes originate messages to be sent around the ring. Both active and inactive processes pass messages originated by other processes. A message is a quadruple

\[ m = (p_m, i_m, c_m, u_m) \]

where

(i) \( p_m \) is the phase number of the originator of the message.

(ii) \( i_m \) is its identification number.

(iii) \( c_m \) is the distance the message travelled (an integer between 1 and \( n \)). Each time a message is passed (leftwards) \( c_m \) is increased by one. Thus, each process can identify its own messages.

(iv) \( u_m \) is a Boolean variable, indicating whether any active processes, other than the originator, have been encountered during the traversal of the ring.

Let \( \text{nextphase}_v \) be a procedure which increases the phase number \( p_v \), chooses at random a new value for \( i_v \) and sets \( b_v \) to \((p_v, i_v, 1, \text{true})\). At any time a process can do any one of the following two activities (the order in which these activities are performed depends on the scheduler, over which we have no control).

A1. If a message is pending \((b_v \neq \varnothing)\) then the message can be sent leftwards and \( \varnothing \) is assigned to \( b_v \).

A2. At any time a process \( v \) can receive the message \((p_m, i_m, c_m, u_m)\) whereupon Program no. 1 is executed.

We envision a message originating at a node (its \textit{originator}) and moving leftwards, until returning or until it is \textit{purged} – not transmitted any further. The first two components of messages are compared lexicographically, a message is \textit{maximal} if it is lexicographically maximal.
begin if \( \text{count}_m = n \) then
  if \( \text{active}_v \) then
    if \( \text{unique}_m \) then \( v \) is the leader
    else nextphase
  else skip
else if \( (\text{ph}_m, \text{id}_m) >_L (\text{ph}_v, \text{id}_v) \) then begin
  \( (\text{ph}_v, \text{id}_v) \leftarrow (\text{ph}_m, \text{id}_m) \);
  \( \text{active}_v \leftarrow \text{false} \);
  \( \text{buf}_v \leftarrow (\text{ph}_m, \text{id}_m, \text{count}_m + 1, \text{unique}_m) \) end
else if \( (\text{ph}_m, \text{id}_m) = (\text{ph}_v, \text{id}_v) \) then
  \( \text{buf}_v \leftarrow (\text{ph}_m, \text{id}_m, \text{count}_m + 1, \text{false}) \)
else skip

end

Program no. 1

Lemma 3.4: After the first message has been created and until a leader is chosen, there exist messages and the originators of all maximal messages are active.

Proof: By induction on the execution sequence of the scheduler.

Basis: Prior to the first communication no purging have taken place and all processes are active, thus the lemma is trivially true.

Induction step: Assume that messages \( m_1, m_2, \ldots, m_k \) arrive at processes \( u_1, u_2, \ldots, u_k \). We assume that all the messages are maximal since other messages can neither purge a maximal message nor cause an originator of a maximal message to become inactive.

If some of the \( m_i \)'s return to their originators then either a leader is chosen, or new larger messages are created, the originators of which are active by the induction hypothesis.
If \( u_i \) is not the originator of \( m_i \) then \( m_i \) is not purged (although a maximal message stored in the buffer may be purged) and \( u_i \) may become inactive only if \( (p_{h_i}, id_{u_i}) \) is lexicographically smaller than \( (p_{h_{m_i}}, id_{m_i}) \), the phase and the identification number carried by the message \( m_i \). Therefore, \( u_i \) is not an originator of a maximal message.

The following lemma shows partial correctness.

**Lemma 3.5:** On termination a unique leader is chosen.

**Proof:** The previous lemma shows that not all the processes terminate without choosing a leader. Uniqueness follows from the fact that a node \( v \) proclaims itself a leader only if when the message returns, \( unique_v \) is true and this implies that all other nodes are inactive.

The remaining question is that of the communication complexity of the algorithm. Obviously the algorithm may diverge. Thus, we are interested in the average communication complexity. An adversary scheduler must be assumed. It is obvious that such a scheduler tries to minimize message purges. Thus, it tries to ensure that the buffers are empty when a lexicographically larger message arrives. Therefore, when an adversary scheduler is active, the network acts almost synchronously. We assume that a process can send and receive messages simultaneously, thereby eliminating message purges. This assumption leads to an upper bound on the average complexity.

**Lemma 3.6:** The average communication complexity of a single phase is \( O(n \log n) \).

**Proof:** Consider the first phase: Let \( p(i, d) \) be the probability that a message sent from a process \( v \) who chose the integer \( i \), travels distance \( d \) until it is purged.
\[ p(i,d) = \begin{cases} p_i^{n-1} & d=n \\ p_i^{d-1}(1-p_i) & d<n \end{cases} \]

where \( p_i = i/n \).

The expected distance \( \bar{D} \) travelled by the message originated at a node \( v \) depends on the value of \( id_v \): For \( id_v = n \), \( \bar{D}(n) = n \). Otherwise, using \( i = id_v \) we get:

\[
\bar{D}(i) = \sum_{d=1}^{n} dp(i,d)
\]

\[
= np_i^{n-1} + \sum_{d=1}^{n-1} dp_i^{d-1}(1-p_i)
\]

\[
= np_i^{n-1} + (1-p_i) \frac{d}{dp_i} \sum_{d=1}^{n-1} p_i^d
\]

\[
= np_i^{n-1} + (1-p_i) \frac{d}{dp_i} \frac{p_i^n - p_i}{1-p_i}
\]

\[
= np_i^{n-1} + (1-p_i) \frac{(1-np_i^{n-1})(1-p_i) + p_i - p_i^n}{(1-p_i)^2}
\]

Since there are \( n \) processes and there is probability \( 1/n \) for a process to choose the integer \( i \), \( N \), the number of communications during the first phase, is bounded by:

\[
N \leq \sum_{i=1}^{n} \bar{D}(i) \leq n + \sum_{i=1}^{n-1} \frac{1-p_i^n}{1-p_i} = n + \sum_{i=1}^{n-1} \frac{1-(i/n)^i}{1-i/n}
\]

\[
< n + \sum_{i=1}^{n-1} \frac{1}{1-i/n} = n + n \sum_{i=1}^{n-1} \frac{1}{n-i} = n + nH_{n-1} = \mathcal{O}(n \log n).
\]

In subsequent phases the number of active processes can only decrease, in which case the communication complexity is even lower.

To compute the average communication complexity we must compute the expected number of phases required to choose a unique leader.

Lemma 3.7: \( l_n(m) \), the expected number of phases required to choose a leader in a ring of length \( n \) with \( m \) asynchronous active processes is
bounded by $e^{-\frac{n}{n-1}}$.

**Proof:** Let $p_n(m,k)$ denote the probability that $k$ processes out of $m$ choose an identification number with a maximal value. Then, using the convention that $0^0=1$,

$$p_n(m,k) = \left(\frac{1}{n}\right)^m \binom{n}{k} \sum_{i=0}^{n-1} i^{m-k}.$$ 

In particular, consider $p_n(m,1)$ for $n > 1$

$$p_n(m,1) = \frac{m}{n^m} \sum_{i=0}^{n-1} i^{m-1} = \frac{m}{n^m} \int_0^1 x^{m-1} dx$$

$$= \left[ \frac{x^n}{n} \right]_0^{n-1} = \left(\frac{n-1}{n}\right)^m$$

$$= \left(1 - \frac{1}{n}\right)^m \geq e^{-\left(1 - \frac{1}{n}\right)}$$

(3.7)

Since $l_n(1) = 0$ for all $n$ and $l_n(j)$ increases with $j$ we get

$$l_n(m) \leq 1 + l_n(m) \sum_{k=2}^{m} p_n(m,k).$$

This implies

$$l_n(m) \leq 1/p_n(m,1).$$

Using (3.7) we get

$$l_n(m) \leq \frac{en}{(n-1)} \quad (n > 1)$$

As a result, the number of phases is bounded by $en/(n-1)$.

Corollary: The expected number of messages is bounded by

$$enN/(n-1) = O(n \log n).$$

Even though the average number of phases is small it is unbounded and therefore the message length and memory requirement of the processes are unbounded too. In the following we shall show how to circumvent this problem.
Let $p_h(t)$ denote the value of $p_h$ at time $t$, and $\maxph(t)$ and $\minph(t)$ the maximum and minimum values of $p_h(t)$. The following lemma shows that the processes are almost synchronized.

**Lemma 3.3:** For all computations, at instance $t$, $\maxph(t) - \minph(t) \leq 1$.

**Proof:** Assume to the contrary that at some instance $t_0$, $\maxph(t_0) - \minph(t_0) > 1$. Moreover, assume that $t_0$ is the earliest such instant. The initial value of all $p_h$ is 0 and thus $\maxph(0) = \minph(0)$ which implies that $t_0 > 0$. Therefore, there exists a message $(\maxph(t_0) - 1, id, n, false)$ which returned to its originator at time $t' \leq t_0$ and caused $v$ to increase its phase number to $\maxph(t_0)$. The fact that the message arrived at $v$ implies that it passed through all the processes and, therefore, for every process $u$, $p_u(t') \geq \maxph(t_0) - 1$. Thus, $\minph(t_0) \geq \minph(t') \geq \maxph(t_0) - 1$, a contradiction.

**Corollary:** The size of each message can be limited to $O(\log n)$ bits, and the average communication complexity to $O(n \log^2 n)$ bits.

**Proof:** Change the algorithm so that the phase number is calculated modulo 3, and comparisons between phase numbers follow the cyclic order $(0 < 1 < 2 < 0)$. 

At any instance there are at most $2n$ messages. Therefore, we may limit the buffer size of each process to hold three messages. Processes can receive messages only while their buffer is not full. The total buffer size is sufficiently large to prevent deadlock - no messages being sent because all the buffers of the destinations are full. This implementation may reduce the degree of parallelism, but it also reduces the memory requirements to $O(\log n)$ bits per process.
4. FINDING THE SIZE OF THE RING.

4.1 Negative Results.

Choosing a leader when the size is unknown and all the processes are indistinguishable is at least as hard as finding the size of the ring. Thus, let us concentrate on the easier problem - that of finding \( n \), in a setup of indistinguishable processes. In the rest of this section all algorithms will be uniform algorithms which terminate with positive probability and consist of indistinguishable processes which do not know the size of the ring.

Theorem 4.1: There exists no partially correct algorithm to decide whether the size of the ring is \( N \) or \( 2N \).

Proof: Suppose to the contrary that such an algorithm existed for the ring of processes \( R^1 = (v_0, \ldots, v_{N-1}) \). Consider the ring \( R^2 = (v_0^1, \ldots, v_{N-1}^1, v_0^2, \ldots, v_{N-1}^2) \) of size \( 2N \) with the same process allocation. For any computation that deduces that the size of \( R^1 \) is \( N \), there exists a computation on \( R^2 \) which concludes (erroneously) that the size of \( R^2 \) is \( N \). The computation on \( R^2 \) simulates that of \( R^1 \) - both \( v_i^1 \) and \( v_i^2 \) conduct the same communication as \( v_i \). However when \( v_0 \) communicates with \( v_{N-1} \) then \( v_0^1 \) communicates with \( v_{N-1}^2 \) and \( v_0^2 \) communicates with \( v_{N-1}^1 \).

Formally, if \( I^1 = (I_1, \ldots, I_i) \) is a \( \rho^1 \)-computation on \( R^1 \), then define

\[
\rho^2(v_i^1) = \rho^2(v_i^2) = \rho^1(v_i).
\]

Also \( I^2 = (I_1^2, \ldots, I_i^2) \) where

\[
I_{k-1}^2(v_i^1) = I_k^2(v_i^1) = I_{k+1}^2(v_i^2) = I_k^1(v_i).
\]

A communication involving \( v_i \) and \( v_{(i+1) \text{mod} N} \) is replaced by two communications as explained above.

It is easy to see that \( I^2 \) is a \( \rho^2 \)-computation which erroneously yields \( N \) as the size of the ring.
Corollary: There exists no partially correct algorithm to calculate the size of the ring.

The following theorem shows that if we insist on distributive termination the probability of error cannot be bounded away from 1.

Theorem 4.2 For all $p < 1$, the probability of error for any distributively terminating algorithm to find $n$ is greater than $p$.

Proof: Suppose to the contrary that such an algorithm existed. Consider first a ring $(v_0, \ldots, v_{n-1})$ and a $\rho$-computation of length $t$ which distributively terminates and which successfully computes $n$. The behavior of the $n$ processes depends on the prefixes of length $t$ of $\rho(v_i)$ and on some non-deterministic choices made by the processes. Now consider a ring $(v_0', \ldots, v_{m-1}')$ of size $m > n$ with the same process assigned to its nodes. Let $\text{pref}_j(\rho(v_i))$ be the prefix of length $j$ of the vector $\rho(v_i)$. The probability tends to 1 as $m$ increases that there exists a segment $(v'_{k+1}, \ldots, v'_{k+2t-1})$ of the ring such that

$$\text{pref}_j(\rho(v'(k+f) \text{mod} m)) = \text{pref}_j(\rho(v_j \text{mod } n)) \quad (1 < j < 2t).$$

However, for such a segment, the process $v'_{k+f}$ may decide that the size of the ring is $n$. At that moment $v'_{k+t}$ must halt (with an incorrect result) since the algorithm is assumed to terminate distributively. There is no way to reactivate $v'_{k+t}$ and thus, even if some other process discovers that $n$ is incorrect, it will not be able to tell this to $v'_{k+t}$. Therefore, with probability arbitrarily close to 1 the algorithm either yields an incorrect result or loops forever.

4.2 Choosing a leader when $N < n < 2N$.

In this section we give partially correct algorithms to choose a leader when $n$ is known to lie in the region $[N, 2N-1]$. Consider first the
synchronous model. The algorithm takes linear time and a linear number of communication bits. The basic idea is to kill active processes until a lone survivor remains. Since \( n \) is not known precisely, processes cannot recognize their own messages. However, a message travelling distance \( 2N-1 \) visits all the processes, and passes through its originator at most once. Thus the algorithm consists of phases of length \( 2N-1 \), and messages produced at the beginning of a phase are passed leftwards in each subsequent time unit. In each phase some of the processes are \( \text{active} \). The number of active processes decreases in each phase until finally a single active process remains - the leader. At the beginning of a phase, each of the active processes chooses an \( \text{id: id}_v \in \{0,1\} \). An active process \( v \) becomes inactive if in the current phase there exists a process \( u \) such that \( \text{id}_u < \text{id}_v \). In all other cases \( v \) remains active in the next phase. The last active process \( v \) is the leader and can detect that it is the last since its \( \text{id}_v = 1 \), and only one message with \( \text{id}=1 \) ends or passes through \( v \). Since no messages are sent after a leader is chosen, after \( 2N-1 \) time units this fact becomes known to the entire ring.

In order to reduce the communication complexity, the first phase is a preliminary phase, in which the probability that \( \text{id}_v = 1 \) is \( p = \frac{1}{2N} \) and only processes with \( \text{id}=1 \) send messages. If no messages are sent in a preliminary phase then \( \text{id}_v = 0 \) for all \( v \), whereupon an additional preliminary phase is conducted. The probability that in a preliminary phase s processes chose \( \text{id}=1 \) is

\[
\frac{\left(\frac{n}{s}\right)p^s(1-p)^{n-s}}{1-(1-p)^n}
\]

Suppose \( s \) processes survived the preliminary phases. Let \( (2N-1)T(s) \) be the expected value of the communication complexity of the remaining
phases. The processes do not know for sure the value of \( s \), therefore, for the remaining phases, the active processes choose \( \ell_k = 1 \) with probability \( \frac{1}{2} \).

\[ P_k(t) = 2 \frac{\binom{s}{t}}{2^t} \] is the probability that exactly \( t \) of the \( s \) active processes choose \( \ell_k = 1 \).

**Lemma 4.1** \( T(1) = 0 \)

\[ T(s) = 2s \quad s \geq 2. \]

**Proof:** By induction on \( s \).

**Base:** When \( s = 1 \) the algorithm has terminated, no further communication takes place, thus, \( T(1) = 0 \).

**Induction step:**

\[
T(s) = s + \sum_{i=2}^{s} P_i(t) T(i) + P_0 T(s)
\]

\[ = s + \sum_{i=2}^{s} P_i(t) T(i) + (P_0 + P_s) T(s) \]

\[ = s + 2^s \sum_{i=2}^{s} \binom{s}{i} T(i) + 21^s T(s) \]

\[ T(s) = \left[ s + 2^s \sum_{i=2}^{s} \binom{s}{i} 2^i \right] \left[ 1 - 2^{1-s} \right]^{-1} \]

\[ = \left[ s + 2^{1-s}(2^{s-1} - 2s) \right] \left[ 1 - 2^{1-s} \right]^{-1} \]

\[ = s(2 - 2^{1-s})/(1 - 2^{1-s}) = 2s \]

The communication complexity of the preliminary stage is \( s(2N-1) \).

Thus, the expected communication complexity of the entire algorithm is

\[
(2N-L) \sum_{s=1}^{n} \frac{\binom{s}{t} p^s (1-p)^{n-s}}{1 - (1-p)^n} (s + T(s))
\]

\[ \leq \frac{3}{1 - (1-p)^n} \sum_{s=0}^{n} s \binom{n}{s} p^s (1-p)^{n-s} = \frac{3pn}{1 - (1-p)^n} \]
Corollary: If \( p \leq 1/n \) then the communication complexity 
\[ \leq 3(2N-1)/(1-e^{-1}) \]. The time is also linear.

After chosen the leader can deduce \( n \) by sending a pebble around the 
ring. Another pebble is required to communicate this to all the processes.

Similar algorithms can be devised for the asynchronous model. The 
complexity will be larger by a factor of \( O(\log n) \).

4.3 A globally terminating algorithm to find \( n \) with low probability of error

In this section we present an algorithm with the following properties:

(i) It always non-distributively terminates. Its time and communication 
    complexities are polynomial.

(ii) The probability of error depends on the external parameter \( r \) and can 
    be made arbitrarily small independently of the size of the ring.

During the execution of the algorithm, each process \( v \) has a candidate 
\( k_v \) for the size of the ring: initially \( k_v = 2 \). The process may create, pass or 
cancel messages. The messages are used in tests which may either increase 
the confidence that \( k_v = n \) or show that \( k_v < n \). For each value of \( k_v \), \( rk_v \) 
successive tests are conducted. If any of the tests fail or any other indica­
tion implies that \( k_v < n \) then the process increases \( k_v \) and repeats the test 
\( rk_v \) times (for the new value of \( k_v \) ). The algorithm terminates when all the 
processes which tested the value of \( k_v \) finished all their tests successfully.
At this point no further communication is issued by any of the processes. It 
will be shown that the algorithm always terminates, on termination all 
processes have the same value of \( k_v \) and that with high probability this 
value is equal to \( n \).
4.3.1. Description of the algorithm

Prior to termination, some of the processes are active. Each active process $v$ tries to send a message carrying $k_v$ leftwards, to a distance $k_v$. If $k_v = n$ then the message returns to $v$. If $k_v < n$ then the message terminates at some node $u \neq v$. If $k_u \leq k_v$ and $u$ knew that the message is not its own then $u$ could deduce that $k_v < n$ and thus update $k_u$ to $k_v + 1$, thereby becoming active. The main difficulty is that if $n$ is not known, no process can identify its own messages with certainty. To help with the identification, each process $v$ randomly chooses 0 or 1 as its identity - $id_v$. This identity is incorporated in the message sent by $v$. Thus, the components of the message are $(k_v, id_v, count)$ where the count field is initially 1.

Assume that a message $(k_m, id_m, count)$ terminates at $v$ (thus, count = $k_m$). Several cases may arise:

L1. $k_m < k_v$. In this case the message is cancelled.

L2. $k_m > k_v$. The process sets $k_v = k_m + 1$ and originates the message $(k_v, id_v, 1)$.

L3. $k_m = k_v$, and the process has not sent a message or $id_m \neq id_v$. The process $v$ is not the originator of the message, and thus proceed as in L2.

L4. $k_m = k_v$ and $id_m = id_v$ and $v$ has originated a message $(k_m, id_m, 1)$. The process runs to the (erroneous) conclusion that its own message has returned. If this test has succeeded less than $\tau k_v$ times, an additional test is initiated.

A difficulty which may arise is the accumulation of messages at a node. This could occur if a process receives messages at a higher rate than the rate of transmission. To solve this problem, each process has a queue $q_v$ of
messages. It will be shown how to limit the size of the queue.

Initially, each process \( v \) randomly chooses an identity \( id_v \in \{0, 1\} \); and executes the procedure \( \text{origin} \text{ate}_v(2) \), which sets \( k_v \leftarrow 2 \), \( \text{times}_v \leftarrow 1 \) and prepares an appropriate message.

**procedure** \( \text{origin} \text{ate}_v(k; \text{integer}) \):

```
begin \( k_v \leftarrow k; \)
\( \text{times}_v \leftarrow 1; \)
delete all messages from \( q_v; \)
add \((k_v, id_v, 1)\) to \( q_v \) end
```

At any time after initialization, each process is ready to send and receive messages. If \( q_v \) is not empty then \( v \) tries to send the first message in \( q_v \) leftwards.

Upon receiving a message \((k_m, id_m, count_m)\), process \( v \) executes **Program no. 2**. The program uses the procedure \( \text{confirm}_v \).

**procedure** \( \text{confirm}_v(); \)

```
begin \( \text{times}_v \leftarrow \text{times}_v + 1; \)
add \((k_v, id_v, 1)\) to \( q_v \) end
```

### 4.3.2 Properties of the algorithm

**Lemma 4.2**: Throughout the algorithm

(i) \( count_m \leq k_m \).

(ii) \( k_v \leq n \).

**Proof**:

(i) \( count_m \) is increased only at lines \( L1 \) and \( L2 \), which in turn are called only when \( k_m > count_m \).
begin if \( k_m > k_v \) then

\[
\begin{align*}
\text{if } k_m \geq \text{count}_m \text{ then begin } \\
\text{\hspace{1cm} originates}_v(k_m); \\
\text{L1: } \text{add } (k_m, id_m, \text{count}_m + 1) \text{ to } q_v \text{ end} \\
\text{else } \text{originate}_v(k_m + 1) \\
\end{align*}
\]

else if \( k_m = k_v \) then

\[
\begin{align*}
\text{if } k_m \geq \text{count}_m \text{ then } \\
\text{L2: } \text{add } (k_m, id_m, \text{count}_m + 1) \text{ to } q_v \\
\text{else if } id_m = id_v \text{ then } \\
\text{\hspace{1cm} if } r < \text{times}_v \text{ then skip } \\
\text{else begin } \\
\text{\hspace{2cm} id}_v \leftarrow \text{random}[0,1]; \\
\text{\hspace{2cm} confirm}_v \text{ end} \\
\text{L3: } \text{else } \text{originate}_v(k_v + 1, 1, \text{true}) \text{ end}
\end{align*}
\]

Program no. 2

(ii) \( \max \{k_v\} \) is initially \( 2 \leq n \), and it may be increased only at line L3, in which case \( k_v = k_m = \text{count}_m \) but the message did not originate at \( v \) since \( id_m \neq id_v \). Thus, there is another node at distance \( k_v \) from \( v \). Since \( k_v \leq n \) we deduce that \( k_v < n \) and by increasing \( k_v \) by one (ii) still holds.

The value \( k_v \) is non-decreasing. Moreover, each process originates at most \( r \) messages with the same value of \( k_v \). Thus, the total number of messages is \( r n^2 \). Consequently, the algorithm is finite and we have the following:
Lemma 4.3: The communication complexity is $O(m^3)$ messages each of $O(\lg m)$ bits.

Let $f_v$ denote the final value of $k_v$. The following lemma shows that the algorithm is consistent.

Lemma 4.4: For all processes $u, w$, $f_u = f_w$.

Proof: Suppose to the contrary, that there exist processes $u$ and $w$ for which $f_w < f_u$. The procedure $\text{originate}_u$ increases $k_u$ to $f_u$. Thus, a message carrying $f_u$ existed. This message could not be cancelled as a result of the arrival of another message. The message could disappear at a node $v$ only if $v$ has sent a message with the same value of $f_u$ exactly $r$ times. Thus, some message carrying $f_u$ succeeds passing through every node, in particular through $w$, increasing $k_w$ to $f$, thus $k_w \geq f$.

Denote the processes $v_0, \ldots, v_{n-1}$ and let $f$ denote the common value of $f_{v_i}$. If $f < n$, then $v_0$ sends the $r$ messages $(f, id^1(v_0), 1), \ldots, (f, id^r(v_0), 1)$. These messages terminate at $v_f$, whereupon $k_{v_f} = f$, and $v_f$ also sent identical messages (otherwise $v_f$ would increase $k_{v_f}$ to $f + 1$). Consequently, $id^1(v_0) = id^1(v_f), \ldots, id^r(v_0) = id^r(v_f)$.

Let $F$ be the equivalence relation $v_i F v_j$ if exists $l$ such that $(j = i + lf \mod n)$. From the above discussion it is clear that if $v_i F v_j$, then $id^1(v_i) = id^1(v_j), \ldots, id^r(v_i) = id^r(v_j)$.

Let $C_0, \ldots, C_{n-1}$ be the equivalence classes of $F$ (see Figure 1), each containing $b = n/a$ processes. Let $g = \gcd(f, n)$ then

$$C_i = \{v_j : j = i + lf \mod n\} = \{v_j : f + i + lg, l = 0, \ldots, b-1\}.$$ Therefore, there are $a = g$ equivalence classes. There is probability $2^{-(b-1)}$ that $id^s(v) = id^s(u)$ for all $u, v$ of the same equivalence class, and probability $2^{-(b-1)r}$ that this holds for $s = 1, \ldots, r$. If $f < n$, then this must hold for all equivalence classes, thus the probability of error is $2^{-(b-1)a} = 2^{-(n-a)r}$. 

Lemma 4.5

(i) For $n$ prime the probability of error is $2^{-(n-1)r}$.

(ii) For any $n$ the probability of error is less than or equal to $2^{-nr/2}$.

Proof:

(i) Follows from the fact that if $n$ is prime then $\alpha = \gcd(f,n) = 1$.

(ii) Follows since $\alpha = \gcd(f,n) \leq n/2$.

4.3.3 Limiting the size of the queues

At any time, there are at least $n$ messages. This follows since initially there were $n$ messages, and new messages are generated only when old ones are cancelled.

This implies that the size of $q_0$ does not exceed $n$. If $n$ is large the space requirement, being $O(n \log n)$ becomes excessive. However, the size of each queue can be limited to two, provided a process can receive a message only when its queue is not full. Deadlock does not occur since not all queues can be full. The space requirement is, thereby, reduced to $O(\log n)$ bits per process.
5. Conclusions

Since its conceptions, distributed programming had to face the problem of symmetry (for example, two processes demanding the same resource). Here we have investigated the effects of symmetry on distributed and randomized algorithms. Because the processes cannot recognize their own messages, some problems which are solvable in an asymmetric network have no algorithmic solution for the symmetric case. Other problems are solvable by algorithms of greater complexity because each process acted independently and the same message was sent many times. A natural continuation is to look at other types of networks. Perhaps, asymmetric networks of identical processes can take advantage of the asymmetry, even if the topology of the network is not known.

It should be pointed out that our negative results are applicable to all network-independent algorithms. I.e., there exists no distributively terminating algorithm to choose a leader or to find n.

Other interesting problems are:

(i) reliably maintaining a leader in a symmetric network subject to process and communication failure;

(ii) developing lower bounds on the complexity of probabilistic algorithms for symmetry breaking both for rings and for general networks.
REFERENCES


[FRS] N. Francez, M. Rodeh and M. Sintozoff, *Distributed Termination with Interval Assertions*. Proceedings of the International Colloquium on the
Formalization of Programming Concepts, April 1981, Lecture Notes in Computer Science 107, (J. Diaz and I. Ramos eds.).


