A FORMAL MODEL FOR MESSAGE PASSING SYSTEMS

James E. Burns

Computer Science Department
Indiana University
Bloomington, IN 47405

TECHNICAL REPORT NO. 91

A FORMAL MODEL FOR MESSAGE PASSING SYSTEMS

JAMES E. BURNS

REVISED: SEPTEMBER 1980

A FORMAL MODEL FOR MESSAGE PASSING SYSTEMS

James E. Burns
Indiana University

Abstract: A formal model for systems of asynchronous processes which communicate only by passing messages is developed. Although each process is deterministically specified, two forms of indeterminism are used in their interactions: 1) the choice of which process takes the next step, and 2) the amount of delay between the sending of a message and its receipt. An example is given illustrating the use of the model.

Keywords: asynchronous processing, distributed systems, message passing.

1. Introduction

In recent research [1], a formal model for systems of asynchronous processes which communicate only by passing messages is developed. This model is designed specifically for presenting the results in that paper. A more general version of the model is documented here. This model is very similar in style to the models of Burns, Fischer, Jackson, Lynch and Peterson [2] and Lynch and Fischer [3].

In distributed systems, processes are inherently asynchronous since there is no universally available clock for synchronizing their actions. The uncertainties of long distance communication imply indeterminate behavior in the sending and receiving of messages. Both forms of indeterminism are allowed in the model given in this paper. (Note: the term "*in*determinism" is used rather than "*non*determinism" since all possible exectutions must be correct before an algorithm is considered correct.)

Processes are allowed to communicate only by sending messages out of a finite number of output ports. Output ports are connected (by a fixed communications network defined separately from the processes) to the input ports of other processes. Thus, a process need not know the identity of the processes with which it is communicating. (Of course, the given facility may be used to construct a virtual communications systems where processes communicate with one another by name, but all of this logic must be part of the programs of the processes.)

Indeterminate delays between the sending and receiving of a message are explictly allowed. Messages sent between a given pair of ports may even arrive out of order. The only requirement is that if a process continues to try to receive messages from a certain input port, then any message sent to this port will be received within a finite number of steps. Communication has no implied synchronization. A process always continues to execute after sending a message. If a process attempts to receive a message when none is ready, the process may continue with other tasks or wait for a message by explicit looping. The given mechanism can be used to implement a virtual system in which processes exchange messages in pairs, but this must be done explicitly in the programs of the individual processes.

In the following section a general model for describing message passing systems will be given. The next section formally defines a communications system in which processes are arranged in a ring. A synchronization problem for this type of communications system is also defined, and a solution of the problem is given. The fourth section presents a lower bound proof (which also appears in Burns [1]) which illustrates the usefulness of the model. The final section presents conclusions.

2. The Formal Model

The following model is very similar to the models in Burns, et al. [2], and Lynch and Fischer [3]. The main differences are emphasis on message passing rather than communication through shared variables and explicit handling of indeterminate delay in message passing.

A process is envisioned as a deterministically programmed processor, residing at a node of a distributed network. A finite set of ports connects each process to the network. Every step, or transition, of a process can be identified as a send, receive or local transition. One of the ports is associated with each send or receive.

Formally, a process is an 8-tuple p = (States_p, Input_p, Output_p, M_p, X_p, T_p, Msg_p, Port_p), where States_p is a non-empty set of process states, Input_p is a finite set of input ports, Output_p is a finite set of output ports, M_p is a set of messages, X_p is a distinguished element of States_p called the initial state of p, T_p is the process state transition function of p, Msg_p is the message output function of p, and Port_p is the port designation function of p. States_p is partitioned into three sets: Send_p, $\mathrm{Receive}_p$ and Local_p. The transition function, T_p, is total on two distinct domains, T_p: $\mathrm{Send}_p \cup \mathrm{Local}_p \rightarrow \mathrm{States}_p$ and T_p: $(\mathrm{Receive}_p \cup \mathrm{NONE}) \times M_p \rightarrow \mathrm{States}_p$. (NONE is a unique value, distinct from any message.) The message output function, Msg_p: $\mathrm{Send}_p \rightarrow M_p$ is also total. Finally, the total function Port_p: $\mathrm{Send}_p \cup \mathrm{Receive}_p \rightarrow \mathrm{Input}_p \cup \mathrm{Output}_p$ is such that if $x \in \mathrm{Send}_p$, then $\mathrm{Port}_p(x) \in \mathrm{Output}_p$, and if $x \in$

$\mathrm{Receive}_p$, then $\mathrm{Port}_p(x) \in \mathrm{Input}_p$.

The basic unit of communication in the model is a "message". To explicitly allow indeterminate delays between the sending a receiving of a message, a "delay factor" is associated with each message sent. The following definitions allow manipulation of messages which are being transmitted.

For any process p, a message pair of p is an element of $M_p \times N$, where N is the set of non-negative integers. The first element of a message pair is the message, and the second is the delay factor. An input queue of p is a finite list of message pairs of p. Let $mq = (m_1,n_1),(m_2,n_2),\ldots,(m_k,n_k)$ be an input queue of p. Message m_i of mq is said to be ready if $n_i = 0$. For any message pair (m,n), define $\mathrm{add}(mq,(m,n)) = (m_1,n_1),(m_2,n_2),\ldots,(m_k,n_k),(m,n)$. Define first(mq) to be m_i if there exists a least i, $1 \leq i \leq k$, such that $n_i=0$ and to be "NONE" otherwise. Define remove(mq) to be mq if first(mq) = NONE and otherwise to be $(m_1,n_1),\ldots,(m_{i-1},n_{i-1}),(m_{i+1},n_{i+1}),\ldots,(m_k,n_k)$ where i is the least positive integer such that $n_i=0$. Finally, define $\mathrm{decr}(mq) = (m_1,n_1'),(m_2,n_2'),\ldots,(m_k,n_k')$ where $n_i' = 0$ if $n_i=0$ and n_i-1 otherwise, for $i=1,2,\ldots,k$.

Let P be a set of processes. Define $\mathrm{Input}_P = \bigcup_{p \in P} \mathrm{Input}_p$, $\mathrm{Output}_P = \bigcup_{p \in P} \mathrm{Output}_p$ and $M_P = \bigcup_{p \in P} M_p$. P is said to be compatible if for every $p,r \in P$ such that $p \neq r$, $M_p = M_r$ and the intersections of Input_p with Input_r and Output_p with Output_r are empty. An instantaneous description (id), q, of a compatible set of processes, P, specifies the state, p(q), of p for each $p \in P$, and the input queue, i(q), at input port i for each $i \in \mathrm{Input}_P$. If q and q' are both id's of P and $p(q)=p(q')$ for each $p \in P$ and $i(q)=i(q')$ for each $i \in \mathrm{Input}_P$, then

$q = q'$. Let Q_P be the set of all id's of P. Define the id initid(P) $\in Q_P$ to be the unique id such that $p(\text{initid}(P)) = X_p$ for all $p \in P$ and such that i(initid(P)) is empty for each $i \in \text{Input}_P$; initid(P) is called the *initial id* of P.

A *message system* is a pair, $S = (P,C)$, where P is a compatible set of processes, and C is a subset of $\text{Output}_P \times \text{Input}_P$. C is called the *communication relation* of S. The *transition function of S*, T_S: $Q_P \times P \times N \to Q_P$, is defined as follows. Assume that q,q' are in Q_P, $p \in P$, $j \in N$ and $T_S(q,p,j) = q'$. Then, for every $r \in P$, $r \neq p$ implies $r(q') = r(q)$. (That is, no process can affect the state of another process.) In addition, the appropriate one of the following three cases must hold.

Case 1: $p(q) \in \text{Send}_p$. Then $q \to q'$ is called a send transition of p.

Let $I = \{i \in \text{Input}_P : (\text{Port}_p(p(q)),i) \in C\}$.

1) For every $i \in \text{Input}_P - I$, $i(q') = i(q)$.
2) $p(q') = T_p(p(q))$.
3) For every $q'' \in Q_P$ such that $p(q'') = p(q)$ and every $k \in N$, $p(T_S(q'',p,k)) = p(q')$.
4) For every $i \in I$, $i(q') = \text{add}(i(q),(\text{Msg}_p(p(q)),j))$.

Case 2: $p(q) \in \text{Receive}_p$. Then $q \to q'$ is called a receive transition of p.

1) For every $i \in \text{Input}_P - \text{Port}_p(p(q))$, $i(q') = i(q)$.
2) $p(q') = T_p(p(q),\text{first}(i(q)))$, where $i=\text{Port}_p(p(q))$.
3) For every q'' in Q_P such that $p(q'') = p(q)$ and $\text{first}(i(q'')) = \text{first}(i(q))$ for $i=\text{Port}_p(p(q))$, and for every $k \in N$, $p(T_S(q'',p,k)) = p(q')$.
4) $i(q') = \text{decr}(\text{remove}(i(q)))$, where $i=\text{Port}_p(p(q))$.

Case 3: $p(q) \in Local_p$. Then $q \to q'$ is called a local transition of p.

1) For every $i \in Input_p$, $i(q') = i(q)$.
2) $p(q') = T_p(p(q))$.
3) For every $q'' \in Q_p$ such that $p(q'') = p(q)$ and every $k \in N$, $p(T_S(q'',p,k)) = p(q')$.

By 1), a send does not affect the input queues that are not connected to its output port, a receive does not affect any input queue but the one it is receiving from, and a local transition does not affect any input queue. 2) requires that T_S conform to the transition functions of the individual processes. 3) states that the action taken depends only on the current state of the process and (for receive only) on the first ready message, if any. Finally, 4) says that a send causes its message to be appended (with delay factor j) to the input ports connected to its output ports, and a receive removes a message (if one is ready) from its input queue and decrements the waiting time of all of the remaining, unready messages.

A message system thus allows messages to be sent with arbitrary, but finite, delay factors. (If a message is sent with delay factor j, then at least j receive transitions must be made on the input port before the message can be read. Note that the FIFO handling of ready messages guarantees that no message will be delayed forever, assuming that the receiving process continues to execute receive transitions on the appropriate input port.) Messages from the same process may arrive in a different order than they were sent. (The messages can be forced to be delivered in order by an appropriate restriction on the choice of delay factors. For example, all the delay factors could be forced to be zero.) Also note that a communication relation can be chosen to allow broadcasts or one-to-one process communication.

Let S=(P,C) be a message system. A schedule, h, of S is a sequence whose elements are chosen from the set P x **N**. Schedule h is admissible if each $p \in P$ occurs infinitely often as a first component in h. Admissible schedules are those in which no process in the system ever stops. (For some problems it might be desirable to allow processes to halt under certain conditions as in Burns, et al. [2] and Fischer, et al. [4].)

Schedules allow us to select a single path out of the computation tree. The two components of an element of a schedule select among the possibilities for each type of indeterministic choice available. To prove a lower bound conjecture, typically the conjecture is assumed to be false, and a schedule is derived which shows a contradiction.

If q is an id of S and $h=(p_1,n_1),(p_2,n_2),\ldots$ is a schedule of S, then the computation from q by h is the sequence of id's $q_1,q_2,\ldots$ such that $q=q_1$ and for $j\geq 1$, $T_S(q_j,p_j,n_j) = q_{j+1}$. Id q' is reachable from q by h if q' occurs in the computation from q by h. If h is finite, then final(S,q,h) is the last id in the computation from q by h. Let $h=(p_1,n_1),(p_2,n_2),\ldots$ be a schedule of S and $q_1,q_2,\ldots$ be the computation from q_1 by h. Then the number of message transmissions from q_1 by h is $\mathrm{msgs}(S,q_1,h) = |\{\ j\geq 1 :\ p_j(q_j) \in \mathrm{send}(p_j)\ \}|$. An id q is called quiescent if for all schedules h, msgs(S,q,h) = 0. The worst case number of messages for a system S is given by MSGS(S) = max {msgs(S,initid(S),h) : h is a schedule of S}.

3. The Election Problem

In order to illustrate the model, a formal definiton of a distributed communication problem is presented. A set of processes, P, is two-way if for every p in P, $\text{Input}_p = \{\text{cli}_p, \text{ccli}_p\}$ and $\text{Output}_p = \{\text{clo}_p, \text{cclo}_p\}$. (Cl is for clockwise, and ccl is for counterclockwise.) A message system, R=(P,C), is a ring if P is a compatible, two-way set of processes and if there exists a function, left: P -> P, such that for any p in P, P = {p, left(p), left(left(p)), ... } and such that $C = \{ (\text{clo}_{\text{left}(p)}, \text{cli}_p) : p \in P\} \cup \{ (\text{cclo}_p, \text{ccli}_{\text{left}(p)}) : p \in P\}$. (See Figure 1.) The size of ring R is equal to |P|.

Representation of a ring R=(P,C) of size N with $p \in P$.

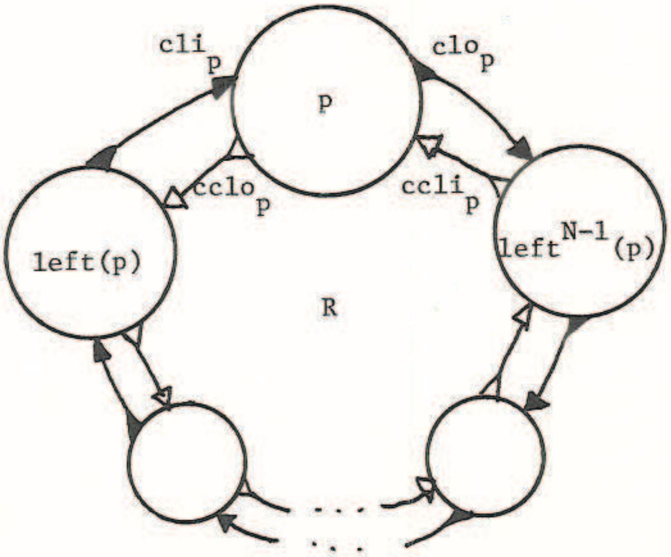

Figure 1.

Let R=(P,C) be a ring such that for each p $\in$ P, elected_p is a subset of States_p. Then R is said to solve the election problem if for every admissible schedule, h, of R, there is a q $\in$ Q_P reachable from q_0=initid(P) by h such that p(q) $\in$ elected_p , and for all r $\in$ P, r$\neq$p implies that for all q' reachable from q_0 by h, r(q') $\notin$ elected_r.

Let P be an infinite, two-way set of compatible processes, such that for each p $\in$ P, elected_p is a subset of States_p. Then P is said to solve the general election problem if for every finite non-empty subset, P', of P and every ring, R=(P',C), R solves the election problem.

The general election problem was first posed by LeLann [5], who provided a solution which required N^2 messages for a ring of size N. Chang and Roberts [6] gave a solution which uses only N log N messages on the average, but still has a worst case performance of N^2 messages. Hirschberg and Sinclair [7] found a solution which uses O(N log N) messages in the worst case. The algorithm given below is a slight improvement of Hirschberg and Sinclair's. Each process is identically programmed except for a constant, which is initialized to a distinct integer value for each process, and for its input and output ports, which are uniquely named. The process named process_I has input ports cli_I and ccli_I and output ports clo_I and cclo_I.

The algorithm for each process is expressed in a Pascal-like notation. The instructions send and receive have been added. The instruction send(dir,msg) in process_I causes a message with value msg to be sent out of port clo_I for dir=clockwise and out of port cclo_I if dir=counterclockwise. The function ready(dir) tests for the presence of a message which is ready to be input from port cli_I or $ccli_I$ if dir=clockwise or counterclockwise, respectively. Finally, receive(dir,msg) causes a ready message to be received and placed in variable msg from input port cli_I if dir=clockwise and $ccli_I$ if dir=counterclockwise.

The reader should be able to see how to transform the following algorithm into the state transition system of the formal model. Note that the "receive" states of the formal model actually correspond to statements of the form "if ready(dir) then receive(dir,msg)" in the algorithm. This causes a message to be received if one is ready and continues otherwise. Finally, the "elected" states of each process are exactly those in which Boolean variable "elected" is true.

When connected to form a ring, any subset of the processes defined below will execute as follows (from the initial id). Each process tries to become elected by sending probes around the ring. (A probe contains the priority of the originating process and the remaining distance that it is to be sent.) A probe goes a fixed distance around the ring and is acknowledged if it does not encounter a priority with a higher value. (An acknowledgment has a distance field of zero.) Each probe is sent twice as far as the last in the opposite direction. The probes of the process with the highest priority value will always be acknowledged. One of its probes will therefore eventually go far enough so that it will return to the originating process, which causes the highest priority process to become elected. (Election occurs when a probe is received by its originating process.) Also, no other process can become elected since the highest priority process will not pass on their messages. The solution is thus easily seen to be correct in that it chooses exactly one process in finite time.

```
program process_I;                (* I is an integer which is *)
   const pri := I;                (* unique for each process *)
   type direction = (clockwise, counterclockwise);
        message = record
                    mpri  : integer;
                    mdist : integer
                  end;
   var maxpri, dist : integer;
       elected, ack : Boolean;
       msg, mmsg    : message;
       dir, mdir    : direction;
   function rev( dir : direction): direction;
      begin
         if dir = clockwise then rev := counterclockwise
         else  rev := clockwise
         end;

   begin
      elected  := false;
      maxpri   := pri;
      msg.mpri  := pri;
      msg.mdist := 1;
      dir      := clockwise;
      while true do begin
         send(dir, msg);                            (* send probe *)
         ack := false;
         while not ack do
            for mdir := clockwise to counterclockwise do
               if ready(mdir) then begin
                  receive(mdir,mmsg);
                  if mmsg.mpri >= maxpri then with mmsg do
                     if mdist=0 then                (* received ack *)
                        if mpri=pri then ack := true
                        else send(mdir,mmsg)        (* pass on ack *)
                     else begin                     (* received probe *)
                        maxpri:= mpri;
                        mdist := mdist -1;
                        if mpri = pri then elected := true
                        else if mdist > 0 then
                           send(mdir,mmsg)          (* pass on probe *)
                           else send(rev(mdir),mmsg)  (* acknowledge *)
                        end
         msg.mdist := 2*msg.mdist;   (* send probe twice as far *)
         dir      := rev(dir);       (* in the opposite direction *)
         end;
   end.
```

Note that this solution is designed specifically to solve the general election problem as formally specified. Details necessary in a more realistic solution are omitted. For example, additional types of messages are needed to terminate the election so that the processes may be freed to do other processing. The reader should have no trouble with the appropriate modifications.

The algorithm given here is very similar to Hirschberg and Sinclair's. The main difference is that in Hirschberg and Sinclair [7], messages are sent in both directions at the same time for a given distance, while the new algorithm saves messages by alternating the direction in which messages are sent. This modification provides an algorithm which has better worst case performance than Hirschberg and Sinclair for any given arrangement of priorities about the ring.

Theorem 1: There exists a solution, P, to the general election problem such that for any subset, P', of P and any ring, R=(P',C), formed from P', $MSGS(R) \leq 4N + 6N \log N$, where $N = |P'|$.

The proof of this theorem and that of the following corollary appear in [1].

Corollary 2: There exists a solution, P, to the general election problem such that for any subset, P', of P with $|P'| = N$, a power of two, and any ring, R=(P',C), formed from P', $MSGS(R) \leq N + 3N \log N$.

4. A Lower Bound

This section shows how the model can be used in the proof of a lower bound corresponding to the algorithm given in the previous section. The proof requires induction on the length of "lines" (rings which have been "cut"), which are defined below.

Representation of Line L=(P,C)

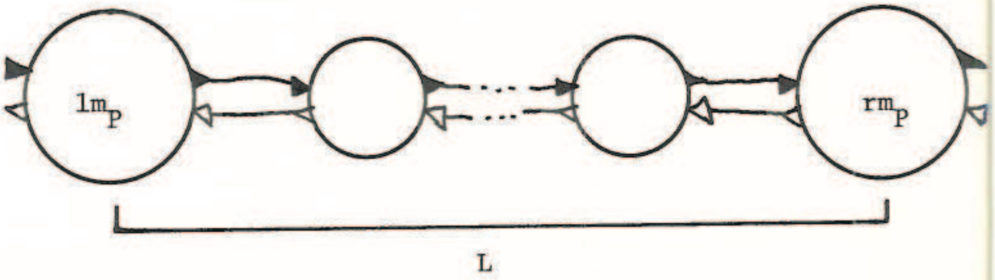

Figure 2.

Let P be a compatible set of two-way processes, with distinguished elements lm_P and rm_P (lm and rm are used for "leftmost" and "rightmost", respectively). Let left be a function, left: $P-\{lm_P\}$ -> $P-\{rm_P\}$ such that $P = \{rm_P, left(rm_P), left^2(rm_P), \ldots , left^{|P|-1}(rm_P)\}$, where $left^i(rm_P) = left(left^{i-1}(rm_P))$ for $i>1$, and where $left^1$ = left. Then the message system L=(P,C) is a *line* if C = { $(clo_{left(p)},cli_p)$: $p \in P-\{lm_P\}$ } U { $(cclo_p,ccli_{left(p)})$: $p \in P-\{lm_P\}$ }. (See Figure 2.) The *length* of line L=(P,C) is |P|. A set of lines, L, is *compatible* if for every pair of lines, L1=(P1,C1) and L2=(P2,C2), in L, P1 U P2 is a compatible set of processes. If L1=(P1,C1) and L2=(P2,C2) are lines, then join(L1,L2) is the line (P,C) where P = P1 U P2, $lm_P = lm_{P1}$, $rm_P = rm_{P2}$ and where C = C1 U C2

U $\{(clo_{rm_{P1}}, cli_{lm_{P2}})\}$ U $\{(cclo_{lm_{P2}}, ccli_{rm_{P1}})\}$. (See Figure 3.) If L=(P,C) is a line, then ring(L) is the message system system (P,C'), where C' = C U $\{(clo_{rm_P}, cli_{lm_P})\}$ U $\{(cclo_{lm_P}, ccli_{rm_P})\}$. A line, L=(P',C), is a line of P if P' is a subset of P.

Representation of join(L1,L2)

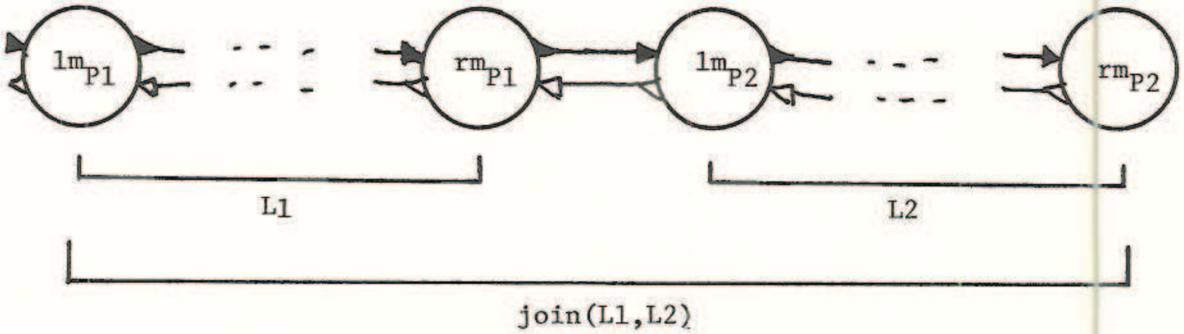

Figure 3.

Lemma 3: Let P be a solution to the general election problem. For every $i \geq 0$ there is an infinite compatible set, L_i, of lines of P such that for every L $\in$ L_i, the length of L is $N=2^i$ and MSGS(L) $\geq$ 1 + (1/4)N log N.

Proof: By induction on i.

BASIS. Suppose there are three processes p,p',p" in P which will not send a message before receiving one. Consider three rings, Rx=({p,p'},Cx), Ry=({p',p"},Cy) and Rz=({p",p},Cz). By assumption, no messages are sent in any of these rings, so each process proceeds independently. Since P solves the general election problem, one of the processes in Rx must reach an elected state after a finite number of steps. Without loss of generality, assume that p becomes elected in Rx, which implies that p' cannot be elected. But then p must also be elected in Rz, so p" cannot. Now p' and p" cannot become elected on their own, so Ry will never elect a process, a contradiction. Therefore, by removing at most two processes from P, an infinite set of processes, P', is obtained, each of which will send a message after a finite number of its own steps. Then the lemma holds for i=0 with $L_0 = \{ (\{p\},\emptyset) : p \in P' \}$.

INDUCTIVE STEP. Assume that the lemma is true for i-1. Let $N = 2^i$. Let L, L' and L" be any three lines in L_{i-1}.

Claim: At least one of the six lines: L_a = join(L,L'), L_b = join(L',L"), L_c = join(L",L), L_d = join(L,L"), L_e = join(L",L') and L_f = join(L',L), can be made to send 1 + (1/4)N log N messages, from their respective initial id's. (See Figure 4.) Since sets of three lines can repeatedly be chosen from L_{i-1} without repetition, it is clear that the claim implies the inductive step.

By the inductive assumption, there must be a finite schedule, h_1, of L for which msgs(L,initid(L),h_1) $\geq$ 1 + (1/4)N/2 log (N/2). Also, it may be assumed that final(L,initid(L),h_1) is quiescent, since, if this cannot be done, the claim holds triviallly for L_a. Let h_1' and h_1'' be similar schedules for L' and L", respectively. The delay factor for messages sent out of clo_{rm_L} and $cclo_{lm_L}$ can have no affect on the state changes of processes in L. Let h be identical to h_1 except that the delay factor for each send transition using port clo_{rm_L} or $cclo_{lm_L}$ is set to $|h_1| + |h_1'| + |h_1''|$. Define h' and h" similarly for L' and L", respectively.

Representation of lines $L_a, L_b, \ldots, L_f$ and rings R1,R2

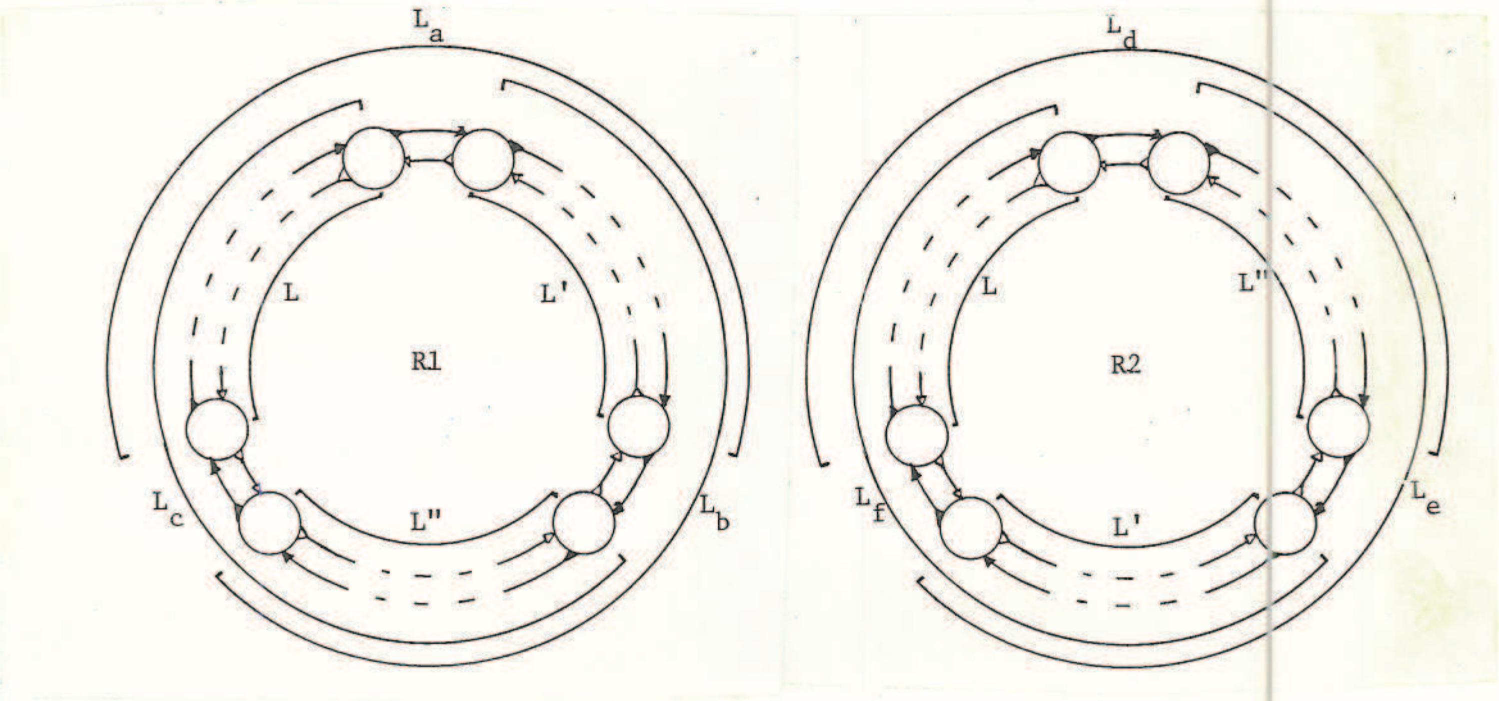

Figure 4.

Suppose the claim is false. Let q_a = final(L_a,initid(L_a),hh'). A set P_a and a schedule h_a of L_a which contains only elements of P_a in the first component and such that final(L_a,q_a,h_a) is quiescent may be generated by the procedure defined below. By assumption, final(L,initid(L),h) and final(L',initid(L'),h') are quiescent, so the only processes which may send messages from q_a are those which have received messages resulting from the join. (The choice of delay factors prevents any interaction during hh'.)

P_a and h_a are generated by the following iteration. Initially, let $P_a = \{rm_L, lm_{L'}\}$ and let h_a = the null sequence. Repeat the following four steps while final(L_a,q_a,h_a) is not quiescent.

1) Let $h_2 = (p_1,0),(p_2,0),\ldots,(p_k,0)$, where $P_a=\{p_1,\ldots,p_k\}$.
2) Let h_3 be the shortest non-null prefix of $h_2h_2h_2\ldots$ for which the last transition in computation from q_a by h_ah_3 is a send.
3) Add the process which will receive the message generated in step 2) to P_a.
4) Append h_3 to h_a.

(Note that the above procedure is not constructive since there is no procedure for deciding if a given id is quiescent.) If the claim is false, the above procedure always terminates with less than N/4 sends because $\mathrm{msgs}(L_a,\mathrm{initid}(L_a),hh') = \mathrm{msgs}(L,\mathrm{initid}(L),h) + \mathrm{msgs}(L',\mathrm{initid}(L'),h') \geq 2*(1 + (1/4)N/2 \log (N/2)) = 1 + (1/4)N \log N + (1 - N/4)$. Generate h_b,h_c,h_d,h_e,h_f and P_b,P_c,P_d,P_e,P_f in a similar way for L_b,L_c,L_d,L_e and L_f, respectively. Note that since messages may not travel further than half a line length from the point of the join, the sets P_a, P_b, P_c and P_d, P_e, P_f are mutually disjoint.

Now consider $R1 = \mathrm{ring}(\mathrm{join}(L_a,L''))$ and $R2 = \mathrm{ring}(\mathrm{join}(L_d,L'))$. (See Figure 4.) It should be clear that $q_1 = \mathrm{final}(R1,\mathrm{initid}(R1), hh'h''h_ah_bh_c)$ and $q_2 = \mathrm{final}(R2,\mathrm{initid}(R2),hh'h''h_dh_eh_f)$ are quiescent.

Thus, for each ring, there are processes, p_1 and p_2, which will be elected on their own (with no further messages sent) from q_1 and q_2, respectively. (That is, there is a finite (possibly empty) schedule h_4 containing only p_1 in its first components such that $p_1(\mathrm{final}(R1,q_1,h_4)) \in \mathrm{elected}(p_1)$. A similar schedule exists for p_2.) Assume, without loss of generality, that $p_1 \in L_a$ and $p_1 \notin P_b \cup P_c$. There are three possibilities for p_2.

Case 1: $p_2 \in L_d$ and $\notin P_e \cup P_f$. Then no process can become elected in Rb = ring(L_b) from q'=final(Rb,initid(Rb),h'h'$h_b h_e$). This contradicts the fact that P solves the election problem. This case will be explained more carefully; the others are similar. Since P_b and P_e are disjoint, q' is quiescent. Suppose some process, p_3, is elected from q' in R_b, and assume without loss of generality that p_3 is within N/2 positions of the middle of L_b. But then p_3 must be distinct from p_1, and so both p_1 and p_3 can be elected in Rl at the same time, a contradiction.

Case 2: $p_2 \in L_e$ and $\notin P_d \cup P_f$. Then no process can become elected in Rc = ring(L_c) from final(Rc,initid(Rc),hh"$h_c h_d$), another contradiction.

Case 3: $p_2 \in L_f$ and $\notin P_d \cup P_e$. Then two process may become elected simultaneously in Ra = ring(L_a) from final(Ra,initid(Ra), hh'$h_a h_f$), again, a contradiction.

Since the hypothesis leads to contradiction, it must be false, hence the claim is proved and the lemma holds. □

Theorem 4: If P is a solution to the general election problem, then for all $N \geq 1$, there exists a subset P' of P with |P'| = N, and a ring R=(P',C) with a schedule h such that at least (1/8)N log (N/2) messages are sent in R from initid(R) by h.

Proof: Let i = floor(log N). By the lemma, a line of P can be found with length 2^i which can be made to send more than $(1/4)*2^i*(\log 2^i)$ $= (1/8)*2^{i+1} *(\log 2^{i+1}/2) \geq (1/8)*N*(\log (N/2))$. Such a line can easily be incorporated in a ring of P of length N, so the theorem is true. □

Corollary 5: If N is a power of two and P is a solution to the general election problem, then there is a ring R=(P',C) with P' a subset of P and |P'| = N and a schedule h of R such that at least (1/4)N log N messages are sent in R by h from initid(R).

Proof: This follows directly from the lemma. □

5. Conclusion

A formal model has been presented for precisely specifying problems for distributed systems which communicate only by message passing. Tight bounds were shown for the number of messages used to solve a particular problem, the general election problem, in a ring network. The techniques used proving the lower bound may find a wider application in other problems of this kind.

The model is essentially a shared variable model of asynchronous computation in which access to the shared variables (input queues) is restricted. Any process which is connected to an input queue by the communication network may append a message to the queue, but only the receiving process may retrieve information from the queue. This retrieval is highly constrained in that only one message may be read at a time, and the message must be ready. This paper has shown how a shared variable model may be used to model a reasonable system of processes communicating by message passing.

Further work remains to be done for other types of communications problems and for other forms of networks. A point of particular interest is dynamic systems. Dynamic systems allow processes to freely enter and leave the system, either with or without explicit notification. The model in this paper can be extended to handle dynamic systems by introducing new types of transitions which allow processes to be added or deleted and allow the communication relation to be changed.

Acknowledgments: The investigation of the lower bound problem was begun as a result of a discussion with Dan Hirschberg. Robert Filman, Edward Robertson and Dan Friedman were kind enough to read and criticize the work. Thanks also goes to Nancy Lynch for pointing out an error in the proof of the lower bound.

REFERENCES

[1] Burns, J. Bounds on message passing in asynchronous networks. In preparation.

[2] Burns, J., Fischer, M., Jackson, P., Lynch, N., and Peterson, G. Shared data requirements for implementation of mutual exclusion using a test-and-set primitive. Proceedings of the 1978 Int'l. Conf. on Parallel Processing, Aug. 1978, pp. 79-87.

[3] Lynch, N., and Fischer, M. On describing the behavior and implementation of distributed systems. *Lecture Notes in Computer Science 70, Semantics of Concurrent Computation*, Goos, G., and Hartmanis, J., (Eds.), Springer-Verlag (1979) 147-171.

[4] Fischer, M., Lynch, N., Burns, J., and Borodin, A. Resource allocation with immunity to limited process failure. Proc. 20th Annual Symp. on Foundations of Computer Science, Oct. 1979, pp. 234-254.

[5] LeLann, G. Distributed systems - towards a formal approach, *Information Processing 77* (IFIP), North-Holland Pub. Co., Amsterdam (1977), 155-160.

[6] Chang, E., and Roberts, R. An improved algorithm for decentralized extrema-finding in circular configurations of processes. *CACM 22*, 5 (May 1977), 281-283.

[7] Hirschberg, D.S., and Sinclair, J.B. An efficient algorithm for decentralized extrema-finding in circular configurations of processors. Manuscript, Rice University, 1979, 5 pp.