A TECHNIQUE FOR PROVING LIVENESS OF COMMUNICATING FINITE STATE MACHINES WITH EXAMPLES

M. G. Gouda and C. K. Chang
Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712

ABSTRACT

Consider a network of communicating finite state machines that exchange messages over unbounded, FIFO channels. Each machine in this network has a finite number of states (called nodes), and state transitions (called edges), and can be defined by a labelled directed graph. A node in one of the machines is said to be "live" iff it is reached by its machine infinitely often during the course of communication, provided that the machines behave in some "fair" fashion. We discuss a technique to verify that a given node is live in such a network. This technique can be automated, and is effective even if the network under consideration is unbounded (i.e. has an infinite number of reachable states). We use our technique to establish the liveness of three distributed solutions to the mutual exclusion problem.

1. Introduction

Consider a network of some finite state machines that communicate exclusively by exchanging messages via connecting channels. There are two one-directional, unbounded, FIFO channels between any two machines in the network. Each machine has a finite number of states and state transitions, and each state transition is accompanied by either sending a message to one of the machine's output channels, or receiving a message from one of the machine's input channels.

Networks of communicating finite state machines are useful in modeling [3], analysis [1,2,12], and synthesis [3,7,13,23] of communication protocols, and distributed systems. The analysis problem for these networks can be stated as follows: "Given an arbitrary network of communicating finite state machines, prove that the communications within the network will satisfy some desirable properties." Most of the work to solve this problem has concentrated so far on properties such as boundedness [22], freedom of deadlocks [20,21] and unspecified receptions [11]. These are all safety properties [18]; i.e. they merely guarantee that nothing bad will happen during the course of communication. In order to guarantee that something good will happen (infinitely often), we need to establish some liveness properties for the given network. In this paper, we identify some liveness properties for networks of communicating finite state machines, and present techniques to prove these properties for such networks.

The pioneering works of Pnueli [19], Owicki and Lamport [18], Misra and Chandy [16,17], and Hailpern and Owicki [14] have established the foundations for defining and proving general liveness properties of concurrent programs and distributed systems. Pnueli [19] has introduced a version of temporal logic as a tool to specify and verify properties of concurrent programs. Owicki and Lamport [18] have used Pnueli's temporal logic to define "proof lattices", that are both rigorous and easy to understand, to verify liveness properties of concurrent programs. Later, Hailpern and Owicki [14] have used the same temporal logic in a modular verification methodology for distributed systems, where the intended system's assertions are verified using the assertions of the different processes in the system.

Misra and Chandy [16,17] have introduced a novel and modular verification methodology for distributed systems, that is not based on temporal logic. In their methodology, safety and liveness properties of any process in the system are defined by three assertions. Applying a "theorem of hierarchy", the process' assertions can be used to verify the safety and liveness properties of the whole system (which are also defined by three assertions).

The above approaches outline general frameworks where many types of liveness properties can be defined and verified for general systems. In this paper, we restrict our attention to defining and proving liveness for networks of communicating finite state machines. This is a significant restriction. For instance, under this restriction, there is no need for a general purpose temporal logic to define the required liveness properties of these networks. In fact, we feel that only one property needs to be defined and verified for any such network, namely that some state(s) in some machine(s) in the network will be reached infinitely often during the course of communication under some fairness assumption. These restrictions and simplifications have led to sufficient conditions that can be checked algorithmically for any given network, and if established for a network, can ensure the liveness of that network.

The paper is organized as follows: Networks of communicating finite state machines are defined in Section 2. The notions of "fairness" and "liveness" are defined in terms of "fair communication sequences" in Section 3. In Section 4, we discuss a sufficient condition that can be used to prove node liveness. In Section 5, we apply our technique to establish the liveness for a variety of solutions to the mutual exclusion problem. Concluding remarks are in Section 6.

All the results in this paper are applicable to networks with any number of communicating machines. But for the sake of clarity, we carry most of the discussion on networks with only two machines. Later in Section 5, we discuss examples with more than two machines to illustrate how to apply our results in this case.

2. Networks of Communicating Finite State Machines

A *communicating finite state machine* M is a directed labelled graph with two types of edges, namely *sending* and *receiving edges*. A sending (or receiving) edge is labelled -g (or +g, respectively) for some message g in a finite set G of messages. One of the nodes in M is identified as its *initial node*, and each node in M is reachable by a directed path from the initial node. For convenience, each node in M has at least one outgoing edge, and the outgoing edges of the same node must have distinct labels. A node in M whose outgoing edges are all sending (or all receiving) edges is called a sending (or *receiving*, respectively) *node*; otherwise it is called a *mixed node*.

Let M and N be two communicating finite state machines with the same set G of messages; the pair (M,N) is called a *network* of M and N. A *state* of network (M,N) is a four-tuple [v,w,x,y], where v and w are two nodes in M and N respectively, and x and y are two strings over the messages in G. Informally, a state [v,w,x,y] means that the executions of M and N have reached nodes v and w respectively, while the input channels of M and N have the message strings x and y respectively.

The *initial state* of network (M,N) is $[v_0,w_0,E,E]$ where v_0 and w_0 are the initial nodes in M and N respectively, and E is the empty string.

Let s=[v,w,x,y] be a state of network (M,N); and let e be an outgoing edge of node v or w. A state s' is said to *follow s over e* iff one of the following four conditions is satisfied:

i. e is a sending edge, labelled -g, from v to v' in M, and s'=[v',w,x,y.g], where "." is the concatenation operator.

ii. e is a sending edge, labelled -g, form w to w' in N, and s'=[v,w',x.g,y].

iii. e is a receiving edge, labelled +g, from v to v' in M, and s'=[v',w,x',y], where x=g.x'.

iv. e is a receiving edge, labelled +g, from w to w' in N, and s'=[v,w',x,y'], where y=g.y'.

Let s and s' be two states of network (M,N), s' follows s iff there is a directed edge e in M or N such that s' follows s over e.

Let s and s' be two states of (M,N), s' is *reachable from* s iff s=s' or there exist states $s_1,...,s_r$ such that $s=s_1$, $s'=s_r$ and s_{i+1} follows s_i for i=1,...,r-1.

A state s of network (M,N) is said to be *reachable* iff it is reachable from the initial state of (M,N).

The communication of a network (M,N) is said to be *bounded* iff there exists a nonnegative integer K such that for any reachable state [v,w,x,y] of (M,N), $|x| \leq K$ and $|y| \leq K$ where |x| is the number of messages in string x. If there is no such K, then the communication is *unbounded*.

In this paper, we characterize the notion of liveness for networks of communicating finite state machines. In particular, we argue that the liveness of such a network should guarantee that "something good" will "occur infinitely often" during the course of communication. This is better explained by an example.

Example 1. Chandy and Misra's Solution to the Mutual Exclusion Problem :

Consider two user processes that share a common resource and synchronize their activities by exchanging messages via two one-directional, unbounded, FIFO channels. Occasionally, each user enters its critical section to access the common resource. The problem is to devise a synchronization mechanism such that at any instant, at most one user is in its critical section. Many solutions to this problem appear in the literature. In this example, we follow the solution proposed by Chandy and Misra in [4]. (Other solutions to this

problem are discussed in Section 5.)

The solution of Chandy and Misra [4] is based on the following rules:

i. Initially, exactly one user has the right to its critical section.

ii. If a user has the right to its critical section, then it can enter its critical section any number of times provided that it frequently examines its input channel from the other user to check whether it has a request message. If it finds a request message, it sends an acknowledgement message to the other user, thus giving up the right to the critical section to the other user. If it finds no request, it keeps the right to its critical section.

iii. If a user has no right to its critical section, it can send a request message to the other user asking for it. Then, when it receives an acknowledgement, it has the right to its critical section.

The network (U_0, U_1) in Figure 1a models the above system, where the two communicating finite state machines U_0 and U_1 model the two user processes. Machine U_i (i=0,1) has the right to its critical section iff it is at its node 3. This implies that at the initial state of the network, machine U_0 has no right to its critical section, and U_1 has the right to its critical section.

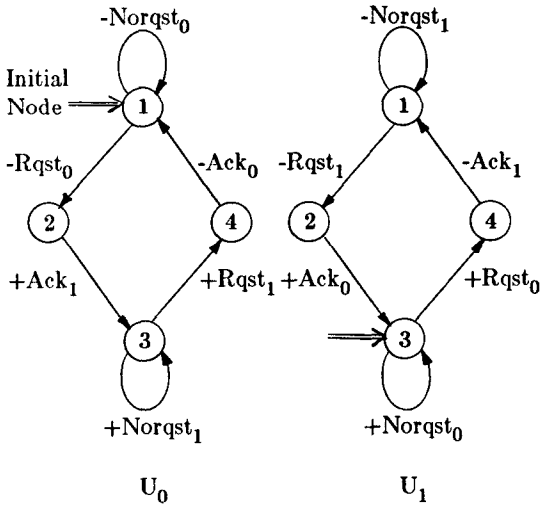


Figure 1a. Network (U_0,U_1) for a solution to the mutual exclusion problem.

The exchanged messages between U_0 and U_1 have the following meaning:

$Rqst_i$ denotes a request message sent by U_i.

Ack_i denotes an acknowledgement sent by U_i.

$Norqst_i$ denotes a "virtual message" sent by U_i to indicate that no $Rqst_i$ message is sent recently by U_i.

The liveness requirement of this network can be stated by requiring that both nodes 3 in machines U_0 and U_1 be "reached infinitely often" during the course of communication. Unfortunately, this is not true in general. For instance, if U_0 persists on ignoring the request message $Rqst_1$ in its input channel, then U_1 will not be able to reach node 3 infinitely often.

Therefore, to ensure that both nodes 3 in U_0 and U_1 will be reached infinitely often, it is important to assume that both U_0 and U_1 will behave in some fair fashion. The notion of fairness for networks of communicating finite state machines is discussed in the next section.

□

3. Liveness Properties

A *communication sequence* of a network (M,N) is a, possibly infinite, sequence $<s_0,s_1,...>$ of reachable states of (M,N) such that s_0 is the initial state of (M,N), and s_{i+1} follows s_i, i=0,1,... . If a communication sequence has a state that cannot be followed by any other state, then the sequence is finite, otherwise it is infinite.

A network (M,N) is called *safe* iff the following two conditions hold:

i. Each communication sequence of (M,N) is infinite.

ii. If any communication sequence $<s_0,s_1,...>$ of (M,N) has a state s_i=[v,w,x,y] such that v(w) is a receiving node and x(y) = E (the empty string), then this sequence must also have a subsequent state s_j=[v',w',x',y'] where x'(y') = g, and g is one of the expected messages at node v(w).

The second condition implies that if a machine reaches a receiving node where it must receive some message to progress further then this message will appear in its input channel in a finite time, and so the machine can progress. Therefore, a network is safe iff each machine can "progress infinitely often". A network can be proven safe using the technique of closed covers [9], as discussed later in Section 4.

Let (M,N) be a safe network. A node v in machine M is said to *occur infinitely often* in a communication sequence $<s_0,s_1,...>$ of (M,N) iff for any integer i there exists an integer j such that j>i and s_j is of the form [v,w,x,y], for some w, x, and y. Similarly, we can define that a node in machine N occurs infinitely often in a communication sequence of (M,N).

Let (M,N) be a safe network. An edge e in machine M is said to *occur infinitely often* in a communication sequence $<s_0,s_1,...>$ of (M,N) iff for any integer i there exists an integer j such that $j>i$ and s_{j+1} follows s_j over e. Similarly, we can define that an edge in N occurs infinitely often in a communication sequence of (M,N).

Based on the above definitions, we can state that something "good" will occur infinitely often in a safe network (M,N) by stating that a node in M or in N (identified as being "good" or "useful" by the network designer) is guaranteed to occur infinitely often in every communication sequence of (M,N). In fact, the node needs not to occur infinitely often in every sequence. In particular, based on the assumption that both machines will progress fairly, the node needs not to occur in any sequence where one machine progresses in unfair fashion. Next we define the fair sequences.

A communication sequence q of a safe network (M,N) is called *fair* iff the following two conditions are satisfied for any node u, in M or N, that occurs infinitely often in q:

i. Each outgoing sending edge of u must occur infinitely often in q.

ii. If there is an infinite number of states of the form $[u,w_i,x_i,y_i]$ (or $[v_i,u,x_i,y_i]$) in q where message g is the head message in x_i (or y_i), and if u has an outgoing receiving edge e, labelled +g, then edge e must occur infinitely often in q.

Informally, a fair sequence is one where each machine is forced to execute infinitely often each edge that it can execute infinitely often. The following theorem states that every safe network must have fair sequences.

Theorem 1 : Let (M,N) be a safe network. (M,N) must have at least one fair sequence.

Proof : Since (M,N) is safe, (M,N) must have a communication sequence $q=s_0,s_1,s_2,...$ that can be constructed as follows:

a. First, the initial state s_0 of (M,N) is added to q.

b. After adding a state s=[v,w,x,y] to q, one of the outgoing edges, say e, of v or w should be selected for execution; then the state s' that follows s over e should be added to q. If such an s' exists, then e is said to be executed; otherwise, e cannot be executed at this time.

c. To decide which of the outgoing edges should be selected for execution, the following priority scheme is adopted. Let k_e be the number of times an edge e in M or N is executed so far along q, on reaching a state [v,w,x,y] along q, the outgoing edge e of v or w with the smallest k_e is selected for execution. If e cannot be executed at this time, then the outgoing edge e' of v or w with the next smallest $k_{e'}$ is selected, and so on. This scheme guarantees that if an edge can be executed infinite times along q, then it will be executed infinite times along q, i.e. q is fair.

□

Based on the above definition of fair communication sequences, we can now present the definition of node liveness.

Let (M,N) be a safe network, and let u be a node in machine M or N. Node u is said to be *live* in (M,N) iff u occurs infinitely often in every fair sequence of (M,N).

Other types of node liveness are discussed in [10], and in [5], we show that the problem of "whether a node u is live in a safe network (M,N)" is undecidable in general, we also characterize some special classes of networks for which the problem becomes decidable. In the current paper, we are interested in developing techniques to *prove* a positive answer for many instances of the problem. In other words, we are interested in sufficient conditions which can be checked easily and which, if satisfied by any instance of the problem, guarantee that indeed node u is live in (M,N).

4. Proving Liveness Using Closed Covers

The technique of closed covers is presented in [9] to prove that a network is safe. One advantage of this technique is that it can be used with networks whose communications are unbounded. (No other technique seems to be successful with such networks.) In this section, we extend this technique to prove node liveness. But first, a brief presentation of closed covers is in order.

A *closed cover* C for a network (M,N) is a set of states of (M,N) that satisfies the following four conditions:

i. The initial state of (M,N) is in C.

ii. Each directed cycle in the directed graph of M or N must have at least one node referenced in some state in C.

iii. The *acyclic version* AM of M with respect to C can be constructed from M by partitioning each node v, which is referenced in some state in C, into two nodes: One node, called the input version of v, has all the output edges of v and no input edges; the other node, called the output version of v, has all the input edges of v and no output edges. Similarly, the acyclic version AN of N with respect to C can be defined. The third condition can now be defined in terms of these acyclic versions. If the network (AM,AN)

starts at a state s_1 in C and if it reaches a state s_2 after which no other state is reachable, then state s_2 must also be in C.

iv. The following condition should be satisfied for any state [v,w,x,y] in C, and for any two paths p and q, in the acyclic versions AM and AN, that start with the input versions of v and w respectively, and terminate at the output versions of some nodes: Let s_i (r_i) be the sequence of sent (received) messages along path i, where i=p,q; then

either [$(x.s_q \prec r_p)$ and $(y.s_p \prec r_q)$],
or [not$(x.s_q \prec r_p)$ and not$(y.s_p \prec r_q)$],
where
"." denotes the string concatenation operator, and
"$\prec$" denotes "is a proper prefix of".

A proof for the following theorem is in [9].

Theorem 2 : If a network has a closed cover, then it is safe.

□

Example 1 (Continues) : We show that the set C = {[1,3,E,E],[3,1,E,E]} is a closed cover for the network (U_0,U_1) in Figure 1a:

i. First, the initial state [1,3,E,E] of (U_0,U_1) is in C.

ii. Since nodes 1 and 3 in U_0 and U_1 are referenced in C, every directed cycle in U_0 or U_1 has one node referenced in C.

iii. The acyclic versions AU_0 and AU_1 of U_0 and U_1 (respectively) with respect to C are shown in Figure 1b. If the network (AU_0,AU_1) starts at state [1,3,E,E], it must end its communication at [1,3,E,E] or at [3,1,E,E]; both are in C. Similarly, if (AU_0,AU_1) starts at state [3,1,E,E], it must end at either [1,3,E,E] or [3,1,E,E].

iv. Each state [v,w,x,y] in C is such that x=y=E. Also, any path in AU_0 or AU_1, that starts with the input version of some node and terminates at the output version of some node, has exactly the same number of sending and receiving edges. Therefore, for any two such paths p and q in AU_0 and AU_1 respectively, we have

not$(s_q \prec r_p$ and not$(s_p \prec r_q)$
where
"$\prec$" denotes "is a proper prefix of", and
$s_i(r_i)$ is the sequence of messages sent (received) along path i, for i=p,q.

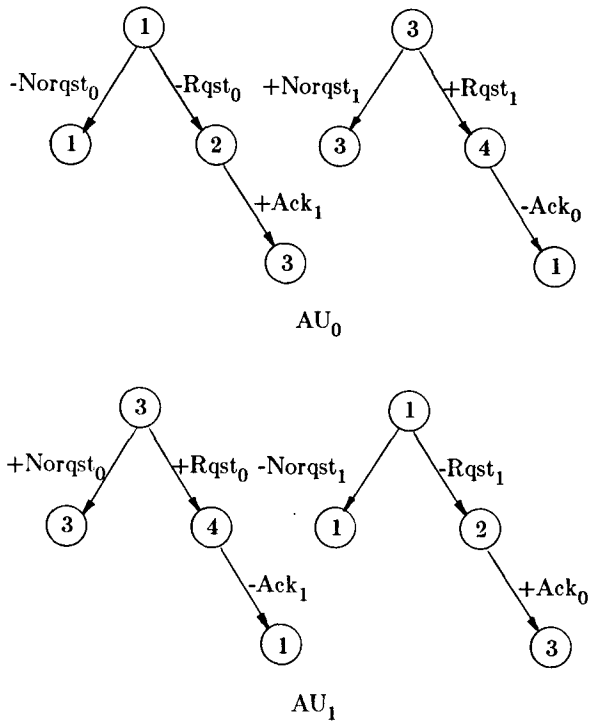


Figure 1b. Acyclic versions AU_0 and AU_1 of U_0 and U_1 in Figure 1a with respect to {[1,3,E,E], [3,1,E,E]}.

This completes the proof that C is a closed cover of (U_0,U_1), and so (U_0,U_1) is safe (by Theorem 2). Notice that the communication of this network is unbounded; hence, there is no other known systematic technique to prove it safe.

□

A closed cover C for a network (M,N) can be represented by a directed labelled graph G, called the *closed cover graph* of C, as follows:

i. Each state s in C is represented as a vertex, also labelled s, in G. (Notice that the "nodes" of G are called "vertices" to distinguish them from the "nodes" of machines M and N. For the same reason, the directed edges in G are called "arcs".)

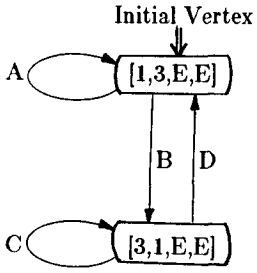
ii. Let AM and AN be respectively the acyclic versions of M and N with respect to C. If the network (AM,AN) can reach from state s in C to state s' in C over a finite sequence $<e_0,e_1,...,e_r>$ of directed edges in M or N. (In other words, there exists a finite subsequence $<s_0,s_1,...,s_{r+1}>$ of states of (AM,AN) such that $s=s_0$, $s'=s_{r+1}$, and s_{i+1} follows s_i over e_i, i=0,1,...,r.}.) Then there is a directed arc from the vertex labelled s to the vertex labelled s' in G; this arc is

labelled with the set of edges $\{e_0,e_1,...,e_r\}$.)

iii. No multiple arcs with identical labels are allowed in G.

Example 1 (Continues) : Figure 1c shows the closed cover graph G of the closed cover C={[1,3,E,E], [3,1,E,E]} for network (U_0,U_1) in Figure 1a. Notice that each directed edge e in U_0 or U_1 is defined in G by a tuple (i, j, k), where i is the source node of e, j is the label of e, and k is the destination node of e. Each arc in G is labelled $\{e_0,e_1,...,e_r\}$.

□



List of labels in the closed cover graph

$A=\{(1,\text{-Norqst}_0,1)_{U0},(3,+\text{Norqst}_0,3)_{U1}\}$
$B=\{(1,\text{-Rqst}_0,2)_{U0},(3,+\text{Rqst}_0,4)_{U1},$
$(4,\text{-Ack}_1,1)_{U1},(2,+\text{Ack}_1,3)_{U0}\}$
$C=\{(1,\text{-Norqst}_1,1)_{U1},(3,+\text{Norqst}_1,3)_{U0}\}$
$D=\{(1,\text{-Rqst}_1,2)_{U1},(3,+\text{Rqst}_1,4)_{U0},$
$(4,\text{-Ack}_0,1)_{U0},(2,+\text{Ack}_0,3)_{U1}\}$

Figure 1c. A closed cover graph G for the closed cover {[1,3,E,E], [3,1,E,E]}.

Let C be a closed cover for a network (M,N), and let G be the closed cover graph of C. The vertex in G labelled with the initial state of (M,N) is called the *initial vertex* of G. A vertex (arc or directed cycle) in G is called *reachable* iff there is a directed path from the initial vertex of G to this vertex (arc or directed cycle). As an example, the vertex [1,3,E,E] in G of Figure 1c is its initial vertex; also each vertex, arc, and directed cycle in this G is reachable.

Let C be a closed cover for a network (M,N), and let G be the closed cover graph of C, also let u be a node in M or N, and e be a directed edge in M or N. Node u is said to *occur* in an arc of G iff the finite set that labels the arc contains an ingoing or outgoing edge of node u. Edge e is said to *occur* in an arc of G iff the finite set that labels the arc contains e. Node u or edge e is said to *occur* in a directed path (or cycle) in G iff it occurs in at least one arc in the path (or cycle). Node u or edge e is said to *occur infinitely often* in an infinite path in G iff it occurs in an infinite number of arcs in the path.

In order to state a sufficient condition for liveness, we need first to define basic and composite cycles in closed cover graphs and to introduce the concept of a message being sent in a composite cycle. This is done next.

Let C be a closed cover for a network (M,N), and let G be the closed cover graph of C. A reachable directed cycle L in G is called *basic* iff each vertex in G occurs at most once in L. Cycle L is called *composite* iff it consists of one or more distinct basic cycles. For example, referring to the closed cover graph in Figure 1c, the self-loop at vertex [1,3,E,E] is basic. Also, the directed cycle that consists of:

1. the arc from vertex [1,3,E,E] to vertex [3,1,E,E],
2. the self-loop at vertex [3,1,E,E], and
3. the arc from vertex [3,1,E,E] to vertex [1,3,E,E]

is composite, since it consists of two basic cycles.

Let C be a closed cover for a network (M,N), and let G be the closed cover graph of C, and L be a composite cycle in G. A message g is said to be *sent by M(N) in L* iff one of the following two conditions holds,

i. There exists a sending edge labelled -g, in M(N), that occurs in L.

ii. One of the vertices in L is labelled with a state [v,w,x,y] where g is in y(x).

Based on these concepts, we can now state Theorem 3.

Theorem 3 : Let (M,N) be a safe network, and let C be a closed cover for (M,N) and G be the closed cover graph of C. Also let u be a node in M or N. u is live in (M,N), if for each reachable directed composite cycle L in G, one of the following two conditions holds:

i. u occurs in L.

ii. There is a node in M or N, that occurs in L, but one of its outgoing sending edges does not occur in L.

Proof : Assume that each reachable directed composite cycle in G satisfies either i or ii above, we show that u must occur infinitely often in every fair sequence of (M,N). Let q be any fair sequence of (M,N). Since q is fair, it corresponds to two infinite directed paths P and Q in machines M and N respectively such that every node occurrence in P or Q must also be in q, and vice versa. The two paths P and Q correspond to one infinite directed path p in G, such that p starts with the initial vertex in G, and every node occurrence in P or Q

must also be in p, and vice versa. It follows that any node in M or N occurs infinitely often in path p iff it occurs infinitely often in sequence q.

Since p is an infinite path in a finite graph G, a finite number of basic cycles in G must occur infinitely often in p, let these basic cycles be L_1, L_2, ..., L_m. Let L_{max} be any composite cycle that consists of L_1, L_2, ..., and L_m; it is straightforward to show that a node (or an edge) in M or N occurs in L_{max} iff it occurs infinitely often in path p (and in sequence q).

There are two cases to consider:

a. u occurs in L_{max}, in which case u occurs infinitely often in sequence q.

b. u does not occur in L_{max} and L_{max} satisfies condition ii. There is a node, say v, in M or N which occurs in L_{max}, but one of its outgoing sending edges does not occur in L_{max}. Therefore, v occurs infinitely often in q, but one of its outgoing sending edges does not occur infinitely often in q. This contradicts the assumption that q is a fair sequence.

□

Example 1 (Continues) : Let us apply Theorem 3 to establish the liveness of nodes 3 in machines U_0 and U_1 in the network of Figure 1a. Referring to the closed cover graph G of this network in Figure 1c, we observe the following:

i. To show that node 3 in U_0 is live in (U_0,U_1) by Theorem 3, we follow the next steps:

   a. Remove from G all the arcs where node 3 in U_0 occurs.

   b. This leaves only the self-loop at vertex [1,3,E,E]; it is a basic cycle that satisfies condition ii in Theorem 3, since node 1 in U_0 occurs in this cycle but its outgoing sending edge labelled -$Rqst_0$ does not occur in this cycle.

ii. To show that node 3 in U_1 is live in (U_0,U_1) by Theorem 3, we follow the same steps:

   a Remove from G all arcs where node 3 in U_1 occurs.

   b. This leaves only the self-loop at vertex [3,1,E,E]; it is a basic cycle that satisfies condition ii in Theorem 3.

□

The closed cover condition in Theorem 3 is applicable even if the considered closed cover C for network (M,N) is infinite, i.e. C contains an infinite number of states of (M,N). However, in this case the closed cover must have a ***finite representation*** so that its closed cover graph remains finite as required by the proof of Theorem 3. One finite representation of an infinite closed cover is as follows:

$C=\{[v_1,w_1,X_1,Y_1], ..., [v_r,w_r,X_r,Y_r]\}$ where

v_i (i=1,...,r)	is a node in machine M,
w_i (i=1,...,r)	is a node in machine N,
X_i (i=1,...,r)	is a (possibly infinite) set of message strings, and
Y_i (i=1,...,r)	is a (possibly infinite) set of message strings.

Each four-tuple $[v_i,w_i,X_i,Y_i]$, called a state schema of (M,N), represents a (possibly infinite) set of states of (M,N), each state is of the form $[v_i,w_i,x_i,y_i]$ where x_i is a string in set X_i and y_i is a string in set Y_i. In this case, each vertex in the corresponding closed cover graph G represents one state schema in C. Therefore, G is finite (i.e. has r vertices) and Theorem 3 is still applicable. (An example is discussed in [10].)

5. Examples

In this section, we use our technique to prove the liveness of two more solutions for the mutual exclusion problem mentioned in Example 1. Both solutions allow the user processes to communicate via shared memory. There are three objectives of this exercise: The first objective is to show how to use communicating finite state machines to model and verify systems where communication is via shared memory. Our second objective is to illustrate that our technique is applicable to networks with more than two machines. The third objective is to apply our technique to prove a system that has been proven earlier by the more general technique of temporal logic [18], thus give the reader an opportunity to compare both techniques.

Example 2. Dijkstra and Knuth's Solution to the Mutual Exclusion Problem:

The solution of Dijkstra [8] and Knuth [15] is based on the following rules:

i. A common memory is shared by the two users in the system. The shared memory contains a "register" whose value at any instant indicates the one user who has the right to its critical section at this instant.

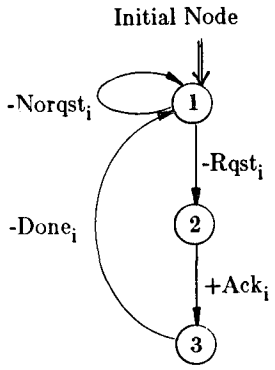
ii. Whenever a user U_i (i=0,1) wants to enter its critical section, it checks the shared register. If the register indicates that U_i has the right to its critical section, U_i can enter right away. Otherwise, U_i waits until the other user is not trying to enter the critical section, then changes the value of the register to indicate U_i and enters its critical section.

iii. Whenever a user exits from its critical sec-

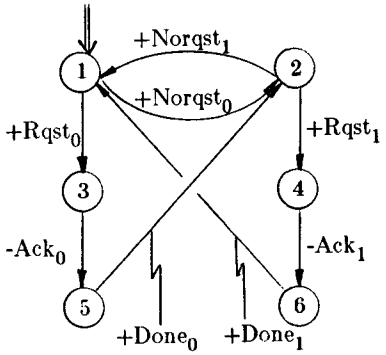
tion, it changes the value of the register to indicate the other user. (This last rule is due to Knuth [15].)

This system can be modeled as a network (U_0,U_1,M) of three communicating finite state machines (Figure 2a). Machines U_0 and U_1 model the two user processes, and machine M models the process that hosts the shared register and contains the information indicating which one of the users currently has the right to its critical section (figure 2b).

Figure 2a. Network (U_0,U_1,M)



U_i (i=0,1)



M

Figure 2b. Machines U_i (i=0,1) and M.

The exchanged messages in this network have the following meaning:

- $Rqst_i$ is a message sent by U_i to M to indicate that U_i wants to enter its critical section.
- $Norqst_i$ is a virtual message sent by U_i to M to indicate that no $Rqst_i$ message is sent recently by U_i.
- Ack_i is a message sent by M to U_i to indicate that U_i can enter its critical section.
- $Done_i$ is a message sent by U_i to M to indicate that U_i is done with its critical section.

To prove the liveness of this solution, it is sufficient to prove that both nodes 3 of machines U_0 and U_1 are live in the network (U_0,U_1,M). The proof is as follows:

A state of this network is of the form: $[v_0,v_1,w,x_0,x_1,y_0,y_1]$, where

- v_i is a node in machine U_i, i=0,1,
- w is a node in machine M,
- x_i is the content of the channel from M to U_i, i=0,1, and
- y_i is the content of the channel from U_i to M, i=0,1.

It is straightforward to check that the following set is a closed cover for this network: C={[1,1,1,E,E,E,E]}. The closed cover graph G of C is shown in Figure 2c, where the arc labels are as follows: (Notice that each arc label is a set of machine edges; each edge e is defined as $(i,j,k)_M$, where i(k) is the source (destination) node of e, j is the label of e, and M is the machine that has e.)

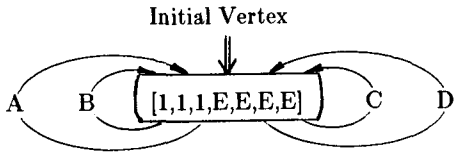


Figure 2c. A closed cover graph G for the closed cover {[1,1,1,E,E,E,E]}.

A= $\{(1,\text{-Norqst}_0,1)_{U0},(1,\text{+Norqst}_0,2)_M,$
$(1,\text{-Norqst}_1,1)_{U1},(2,\text{+Norqst}_1,1)_M\}$

B= $\{(1,\text{-Norqst}_0,1)_{U0},(1,\text{+Norqst}_0,2)_M,$
$(1,\text{-Rqst}_1,2)_{U1},(2,\text{+Rqst}_1,4)_M,$
$(4,\text{-Ack}_1,6)_M,(2,\text{+Ack}_1,3)_{U1},$
$(3,\text{-Done}_1,1)_{U1},(6,\text{+Done}_1,1)_M\}$

C= $\{(1,\text{-Rqsr}_0,2)_{U0},(1,\text{+Rqst}_0,3)_M,$
$(1,\text{-Norqst}_1,1)_{U1},(3,\text{-Ack}_0,5)_M,$
$(2,\text{+Ack}_0,3)_{U0},(3,\text{-Done}_0,1)_{U0},$

$(5,+Done_0,2)_M,(2,+Norqst_1,1)_M\}$

$D= \{(1,-Rqst_0,2)_{U0},(1,+Rqst_0,3)_M,$
$(1,-Rqst_1,2)_{U1},(3,-Ack_0,5)_M,$
$(2,+Ack_0,3)_{U0},(3,-Done_0,1)_{U0},$
$(5,+Done_0,2)_M,(2,+Rqst_1,4)_M,$
$(4,-Ack_1,6)_M,(2,+Ack_1,3)_{U1},$
$(3,-Done_1,1)_{U1},(6,+Done_1,1)_M\}$

From this closed cover graph, it is straightforward to use Theorem 3 to prove that both nodes 3 in U_0 and U_1 are live.

The network described above actually models Knuth's solution [15]. To model Dijkstra's solution [8], the two nodes 1 and 2 in machine M should be merged into one node, and all the edges labelled $+(-)Norqst_i$ should be removed from U_0, U_1, and M. We could not construct a closed cover for the resulting network; therefore, we cannot use Theorem 3 to establish its liveness. Fortunately, this network is bounded (i.e. has a finite number of reachable states), and so its liveness can be established by examining its finite reachability graph.

The liveness of our model of Dijkstra's solution can be disturbing at first glance, since his solution is known to be "non-live". However, one should remember that the established liveness is based on the assumption that M behaves fairly, and so whenever a user U_i sends a $Rqst_i$ message to M, M must receive and approve this request in a finite time. Indeed, Dijkstra's solution is live under this fairness assumption.

Example 3. Owicki and Lamport's Solution to the Mutual Exclusion Problem:

The solution of Owicki and Lamport [18] is based on the following rules. (These rules favor user U_0 whenever both U_0 and U_1 compete for their critical sections.)

i. Each user U_i owns a binary flag F_i that it can update, and the other user can only read. Whenever the value of F_i is *true*, it indicates (to the other user) that U_i wants to enter its critical section. Initially, the value of both flags F_0 and F_1 is *false*.

ii. Whenever U_0 wants to enter its critical section, it flips its flag F_0 to *true*, and reads the other flag F_1. If F_1 = *false*, U_0 enters its critical section, and at the end flips its flag F_0 back to *false*. If F_1 = *true*, then U_0 waits and reads F_1 later, and so on. This continues, until U_0 sees that F_1 = *false*; it then enters its critical section, and at the end flips its flag F_0 to *false*.

iii. Whenever U_i wants to enter its critical section, it flips its flag F_1 to *true* and reads F_0. If F_0 = *false*, then U_1 enters its critical section, and at the end flips its flag F_1 to *false*. If F_0 = *true*, then U_1 flips its flag F_1 back to *false* and waits for a finite time before it tries to enter its critical section again.

This system can be modeled as a network (U_0,U_1,P_0,P_1) of four communicating finite state machines (Figure 3a), where machines U_0 and U_1 (Figure 3b) model the two user processes, and machines P_0 and P_1 (Figure 3c) model processes that host the two flags F_0 and F_1, respectively. The exchanged messages in this network have the following meaning:

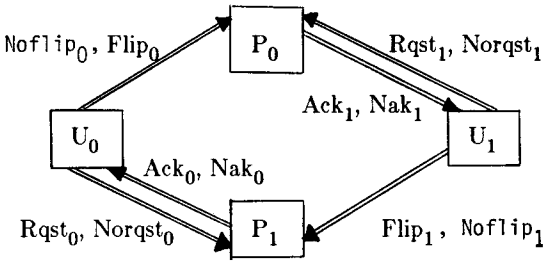


Figure 3a. Network (U_0, U_1, P_0, P_1).

$Flip_i$ is a message sent by U_i to P_i to change the value of the flag F_i either from *false* to to *true* or from *true* to *false*.

$Rqst_i$ is a message sent by U_i to $P_{(i+1 \bmod 2)}$ to inquire about the value of the flag $F_{(i+1 \bmod 2)}$

$Noflip_i$ is a virtual message sent by U_i to P_i to indicate that no $Flip_i$ message is sent recently by U_i.

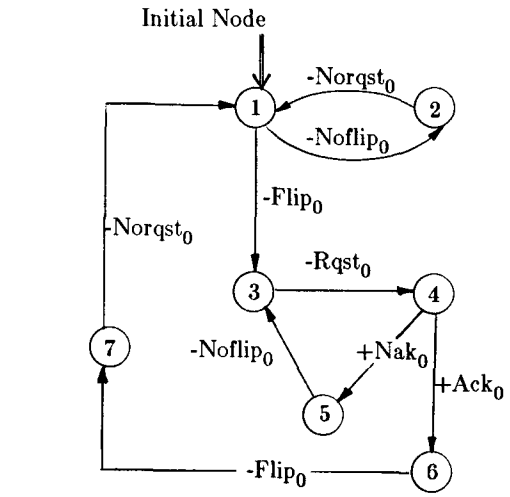
$Norqst_i$ is a virtual message sent by U_i to $P_{(i+1 \bmod 2)}$ to indicate that no $Rqst_i$ message is sent recently by U_i.

Ack_i is a message sent by $P_{(i+1 \bmod 2)}$ to U_i to indicate that the current value of the flag in $P_{(i+1 \bmod 2)}$ is *false*.

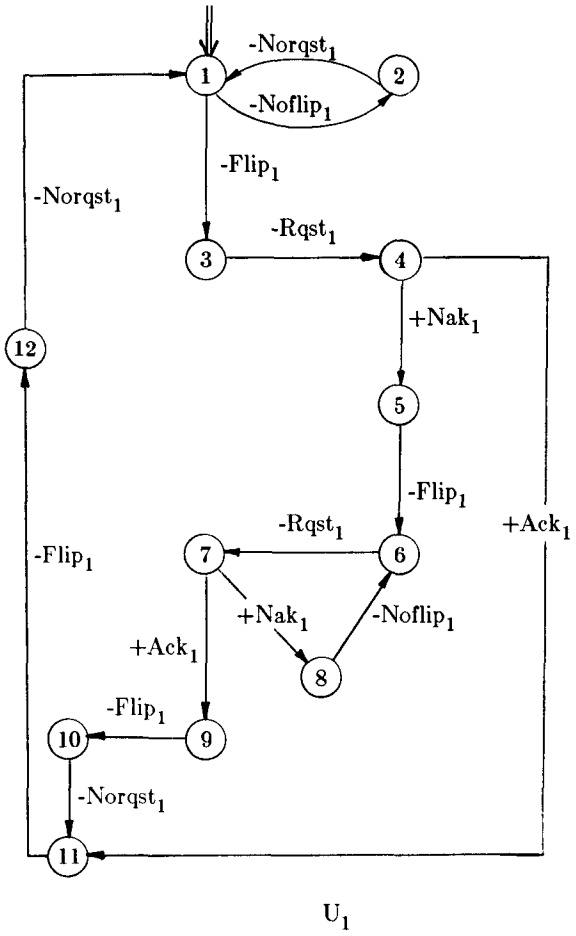
Nak_i is a message sent by $P_{(i+1 \bmod 2)}$ to U_i to indicate that the current value of the flag in $P_{(i+1 \bmod 2)}$ is *true*.

To prove the liveness of this solution to the mutual exclusion problem, it is sufficient to prove that node 6 in machine U_0 and node 11 in U_1 are both live in the network (U_0,U_1,P_0,P_1). The proof is next.

A state of this network is of the form: $[v_0,v_1,w_0,w_1,x_0,x_1,y_{00},y_{01},y_{10},y_{11}]$, where



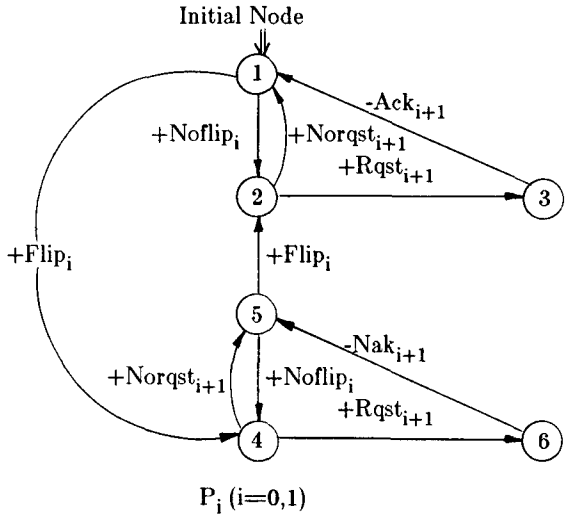
U_0



U_1

Figure 3b. Machines U_0 and U_1.

v_i is a node in machine U_i, i=0,1,
w_i is a node in machine P_i, i=0,1,
x_i is the content of the channel from $P_{(i+1 \bmod 2)}$ to U_i, i=0,1,
y_{ij} is the content of the channel from U_i to P_j, i=0,1 and j=0,1.



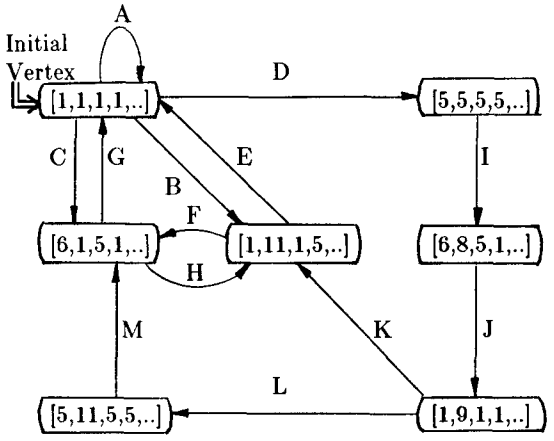
P_i (i=0,1)

Figure 3c. Machines P0 and P1.
(+ is mod 2)

It is straightforward to check that the following set is a closed cover for this network:

C= {[1,1,1,1,E,E,E,E,E,E],
[1,11,1,5,E,E,E,E,E,E],
[6,1,5,1,E,E,E,E,E,E],
[5,5,5,5,E,E,E,E,E,E],
[6,8,5,1,E,E,E,E,E,E],
[1,9,1,1,E,E,E,E,E,E],
[5,11,5,5,E,E,E,E,E,E] }

The closed cover graph G of C is shown in Figure 3d, where the arc labels are as follows:



[v,w,x,y,..] stands for [v,w,x,y,E,E,E,E,E,E]
where E is the empty string

Figure 3d. A closed cover graph G.

A= $\{(1,-\text{Noflip}_0,2)_{U0},(1,+\text{Noflip}_0,2)_{P0},$
$(1,-\text{Noflip}_1,2)_{U1},(1,+\text{Noflip}_1,2)_{P1},$
$(2,-\text{Norqst}_0,1)_{U0},(2,+\text{Norqst}_0,1)_{P1},$
$(2,-\text{Norqst}_1,1)_{U1},(2,+\text{Norqst}_1,1)_{P0}\}$

B= $\{(1,-\text{Noflip}_0,2)_{U0},(1,+\text{Noflip}_0,2)_{P0},$
$(1,-\text{Flip}_1,3)_{U1},(1,+\text{Flip}_1,4)_{P1},$
$(2,-\text{Norqst}_0,1)_{U0},(4,+\text{Norqst}_0,5)_{P1},$
$(3,-\text{Rqst}_1,4)_{U1},(2,+\text{Rqst}_1,3)_{P0},$
$(3,-\text{Ack}_1,1)_{P0},(4,+\text{Ack}_1,11)_{U1}\}$

C= $\{(1,-\text{Flip}_0,3)_{U0},(1,+\text{Flip}_0,4)_{P0},$
$(1,-\text{Noflip}_1,2)_{U1},(1,+\text{Noflip}_1,2)_{P1},$
$(3,-\text{Rqst}_0,4)_{U0},(2,+\text{Rqst}_0,3)_{P1},$
$(2,-\text{Norqst}_1,1)_{U1},(4,+\text{Norqst}_1,5)_{P0},$
$(3,-\text{Ack}_0,1)_{P1},(4,+\text{Ack}_0,6)_{U0}\}$

D= $\{(1,-\text{Flip}_0,3)_{U0},(1,+\text{Flip}_0,4)_{P0},$
$(1,-\text{Flip}_1,3)_{U1},(1,+\text{Flip}_1,4)_{P1},$
$(3,-\text{Rqst}_0,4)_{U0},(4,+\text{Rqst}_0,6)_{P1},$
$(3,-\text{Rqst}_1,4)_{U1},(4,+\text{Rqst}_1,6)_{P0},$
$(6,-\text{Nak}_0,5)_{P1},(4,+\text{Nak}_0,5)_{U0},$
$(6,-\text{Nak}_1,5)_{P0},(4,+\text{Nak}_1,5)_{U1}\}$

E= $\{(1,-\text{Noflip}_0,2)_{U0},(1,+\text{Noflip}_0,2)_{P0},$
$(11,-\text{Flip}_1,12)_{U1},(5,+\text{Flip}_1,2)_{P1},$
$(2,-\text{Norqst}_0,1)_{U0},(2,+\text{Norqst}_0,1)_{P1},$
$(12,-\text{Norqst}_1,1)_{U1},(2,+\text{Norqst}_1,1)_{P0}\}$

F= $\{(1,-\text{Flip}_0,3)_{U0},(1,+\text{Flip}_0,4)_{P0},$
$(11,-\text{Flip}_1,12)_{U1},(5,+\text{Flip}_1,2)_{P1},$
$(3,-\text{Rqst}_0,4)_{U0},(2,+\text{Rqst}_0,3)_{P1},$
$(12,-\text{Norqst}_1,1)_{U1},(4,+\text{Norqst}_1,5)_{P0},$
$(3,-\text{Ack}_0,1)_{P1},(4,+\text{Ack}_0,6)_{U0}\}$

G= $\{(6,-\text{Flip}_0,7)_{U0},(5,+\text{Flip}_0,2)_{P0},$
$(1,-\text{Noflip}_1,2)_{U1},(1,+\text{Noflip}_1,2)_{P1},$
$(7,-\text{Norqst}_0,1)_{U0},(2,+\text{Norqst}_0,1)_{P1},$
$(2,-\text{Norqst}_1,1)_{U1},(2,+\text{Norqst}_1,1)_{P0}\}$

H= $\{(6,-\text{Flip}_0,7)_{U0},(5,+\text{Flip}_0,2)_{P0},$
$(1,-\text{Flip}_1,3)_{U1},(1,+\text{Flip}_1,4)_{P1},$
$(7,-\text{Norqst}_0,1)_{U0},(4,+\text{Norqst}_0,5)_{P1},$
$(3,-\text{Rqst}_1,4)_{U1},(2,+\text{Rqst}_1,3)_{P0},$
$(3,-\text{Ack}_1,1)_{P0},(4,+\text{Ack}_1,11)_{U1}\}$

I= $\{(5,-\text{Noflip}_0,3)_{U0},(5,+\text{Noflip}_0,4)_{P0},$
$(5,-\text{Flip}_1,6)_{U1},(5,+\text{Flip}_1,2)_{P1},$
$(3,-\text{Rqst}_0,4)_{U0},(2,+\text{Rqst}_0,3)_{P1},$
$(6,-\text{Rqst}_1,7)_{U1},(4,+\text{Rqst}_1,6)_{P0},$
$(6,-\text{Nak}_1,5)_{P0},(7,+\text{Nak}_1,8)_{U1},$
$(3,-\text{Ack}_0,1)_{P1},(4,+\text{Ack}_0,6)_{U0}\}$

J= $\{(6,-\text{Flip}_0,7)_{U0},(5,+\text{Flip}_0,2)_{P0},$
$(8,-\text{Noflip}_1,6)_{U1},(1,+\text{Noflip}_1,2)_{P1},$
$(7,-\text{Norqst}_0,1)_{U0},(2,+\text{Norqst}_0,1)_{P1},$
$(6,-\text{Rqst}_1,7)_{U1},(2,+\text{Rqst}_1,3)_{P0},$
$(3,-\text{Ack}_1,1)_{P0},(7,+\text{Ack}_1,9)_{U1}\}$

K= $\{(1,-\text{Noflip}_0,2)_{U0},(1,+\text{Noflip}_0,2)_{P0},$
$(9,-\text{Flip}_1,10)_{U1},(1,+\text{Flip}_1,4)_{P1},$
$(2,-\text{Norqst}_0,1)_{U0},(4,+\text{Norqst}_0,5)_{P1},$
$(10,-\text{Norqst}_1,11)_{U1},(2,+\text{Norqst}_1,1)_{P0}\}$

L= $\{(1,-\text{Flip}_0,3)_{U0},(1,+\text{Flip}_0,4)_{P0},$
$(9,-\text{Flip}_1,10)_{U1},(1,+\text{Flip}_1,4)_{P1},$
$(3,-\text{Rqst}_0,4)_{U0},(4,+\text{Rqst}_0,6)_{P1},$
$(10,-\text{Norqst}_1,11)_{U1},(4,+\text{Norqst}_1,5)_{P0},$
$(6,-\text{Nak}_0,5)_{P1},(4,+\text{Nak}_0,5)_{U0}\}$

M= $\{(5,-\text{Noflip}_0,3)_{U0},(5,+\text{Noflip}_0,4)_{P0},$
$(11,-\text{Flip}_1,12)_{U1},(5,+\text{Flip}_1,2)_{P1},$
$(3,-\text{Rqst}_0,4)_{U0},(2,+\text{Rqst}_0,3)_{P1},$
$(12,-\text{Norqst}_1,1)_{U1},(4,+\text{Norqst}_1,5)_{P0},$
$(3,-\text{Ack}_0,1)_{P1},(4,+\text{Ack}_0,6)_{U0}\}$.

From this closed cover graph, it is straightforward to use Theorem 3 to prove that both node 6 in U_0 and node 11 in U_1 are live.

In [18], Owicki and Lamport prove the liveness of U_0 using temporal reasoning. They also prove the liveness of U_1 under the assumption that U_0 remains forever in its noncritical section. This assumption is needed because in their solution, U_1 can be blocked from entering its critical section by a "greedy" U_0 who enters its critical section too often. By contrast, in our model, whenever a user U_0 (U_1) exits from its critical section, its flag machine P_0 (P_1) will wait to receive either a Rqst or Norqst message from the other user before waiting to receive from its user again. Moreover, a user that fails to enter its critical section in the first trial will keep on sending Rqst messages to the other flag machine until it succeeds. This guarantees that a user who wants to enter its critical section cannot be blocked forever, thus both U_0 and U_1 are live in our model as our proof indicates.

It is possible to change our model slightly to reach the same results as those of Owicki and Lamport's. Simply add a directed edge, labelled -Norqst_1, from node 6 to node 1 in machine U_1. In this case, node 6 in U_0 is still live while node 11 in U_1 is no longer live. To show that node 11 in U_1 is not live, one needs only to define a fair sequence in which node 11 in U_1 does not occur infinitely often; we leave the details to the reader.

6. Concluding Remarks

The node liveness discussed in this paper is based on the notion of fair sequences. In [10], we identify two more types of fair sequences (named "weakly fair" and "strongly fair" sequences), and based on them we define two other degrees of node liveness (named "strongly live" and "weakly live", respectively). Also in [10], we extend the technique discussed in the current paper to

prove all three degrees of node liveness. The extended technique is used to prove the liveness of some real communication protocols, and some other solutions (including probabilistic ones) to the mutual exclusion problem [6].

REFERENCES

1. Brand, D. and P. Zafiropulo, "On communicating finite-state machines," *Journal ACM*, Vol. 30, No. 2, April 1983, pp. 323-342.

2. Bochmann, G. V., "Finite state description of communication protocols," *Computer Networks*, Vol. 2, 1978, pp. 361-371.

3. Bochmann, G. V. and C. Sunshine, "Formal methods in communication protocol design," *IEEE Trans. on Commun.*, April 1980, pp.624-631.

4. Chandy, K. M. and J. Misra, "The drinking philosophers problem," *TR LCS-8402*, Dept. of Computer Sciences, Univ. of Texas at Austin, Feb. 1984.

5. Chang, C. K., M. G. Gouda, and L. E. Rosier, "Deciding liveness for special classes of communicating finite state machines," In preparation.

6. Chang, C. K, "Proving liveness properties for communicating machines," Ph.D. Thesis, Univ. of Texas at Austin, In preparation.

7. Chow, C. H., M. G. Gouda, and S. S. Lam, "An exercise in constructing multi-phase communication protocols," In *Proc. of SIGCOMM'84 Symposium*, June 1984.

8. Dijkstra, E. W., "Solution of a problem in concurrent programming control," *Comm. ACM*, Vol. 8, No. 9, Sept. 1965, pp. 569.

9. Gouda, M. G., "Closed covers: to verify progress for communicating finite state machines," *TR-191*, Dept. of Computer Sciences, Univ. of Texas at Austin, Jan. 1982. Revised Jan. 1983. To appear in *IEEE Trans. on Software Engineering*.

10. Gouda, M. G. and C. K. Chang, "Proving liveness for networks of communicating finite state machines," *TR-84-4*, Dept. of Computer Sciences, Univ. of Texas at Austin, Jan. 1984.

11. Gouda, M. G., E. G. Manning, and Y. T. Yu, "On the progress of communication between two finite state machines," *TR-200*, Dept. of Computer Sciences, Univ. of Texas at Austin, May 1982. Revised Oct. 1983.

12. Gouda, M. G. and Y. T. Yu, "Protocol validation by maximal progress state exploration," *IEEE Trans. on Commun.*, Vol. COM-32. No. 1, Jan. 1984, pp. 94-97.

13. Gouda, M. G. and Y. T. Yu, "Synthesis of communicating finite state machines with guaranteed progress", *TR-179*, Dept. of Computer Sciences, Univ. of Texas at Austin, June 1981. Revised Jan. 1983. Revised Oct. 1983. To appear in *IEEE Trans. on Commun.*, July 1984.

14. Hailpern, B. T. and S. S. Owicki, "Modular verification of computer communication protocols," *IEEE Trans. on Commun.*, Vol. COM-31, No. 1, Jan. 1983, pp. 56-68.

15. Knuth, D. E., "Additional comments on a problem in concurrent programming control," *Comm. ACM*, Vol. 9, No. 5, May 1966, pp. 321-322.

16. Misra, J. and K. M. Chandy, "Proof of networks of processes," *IEEE Tran. on Software Eng.*, Vol. SE-7, No. 4, July 1981.

17. Misra, J., K. M. Chandy and T. Smith, "Proving safety and liveness of communicating processes with examples," in *Proc ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, Aug. 1982, pp. 18-20.

18. Owicki, S. and L. Lamport, "Proving liveness properties of concurrent programs," *ACM Trans. on Programming Languages and Syst.*, Vol. 4, No. 3, July 1982, pp. 455-495.

19. Pnueli, A., "The temporal semantics of concurrent programs," *Theoretical Computer Science*, Vol. 13, 1981, pp. 45-60.

20. Rosier, L. E. and M. G. Gouda, "Deciding progress for a class of communicating finite state machines," *TR-83-22*, Dept. of Computer Sciences, Univ. of Texas at Austin, Oct. 1983. Submitted for publication.

21. Yu, Y. T. and M. G. Gouda, "Deadlock detection for a class of communicating finite state machines," *IEEE Trans. on Commun.* , Dec. 1982, pp. 2514-2519.

22. Yu, Y. T. and M. G. Gouda, "Unboundedness detection for a class of communicating finite-state machines," *Information Processing Letters*, Vol. 17, Dec. 1983, pp. 235-240.

23. Zafiropulo, P., C. H. West, H. Rudin, D. Brand, and D. Cowan, "Towards analyzing and synthesizing protocols," *IEEE Trans. on Commun.*, Vol. COM-28, No. 4, April 1980, pp. 651-661.