prises in Turkey. He has also worked as a UNIDO expert, conducting various professional seminars. His current research interests are in database systems, distributed systems, database computers, CAD/CAM, and AI and databases. He has an upcoming book, *Project Planning and Control Techniques*, to be published, in Turkish, by the Information Processing Society of Turkey.

Dr. Ozsu is a member of the IEEE Computer Society, the Association for Computing Machinery, and Sigma Xi.

Proving Liveness and Termination of Systolic Arrays Using Communicating Finite State Machines

MOHAMED G. GOUDA, MEMBER, IEEE, AND HUI-SENG LEE

Abstract—**We model a systolic array as a network of, mostly identical, communicating finite state machines that exchange messages over one-to-one, unbounded, FIFO channels. Each machine has a cyclic behavior; in each cycle, a machine first receives one message from each of its input channels, then sends one message to each of its output channels. If in a cycle a machine does not have any data message to send to one of its output channels, it sends a null message instead; thus, machines exchange two types of messages, data and null. We characterize the liveness and termination properties for such networks, and discuss two algorithms that can be used to decide these properties for any given network. We apply these algorithms to establish the liveness and termination properties of four systolic array examples. These examples include a linear matrix–vector multiplier, a linear priority queue, and a search tree.**

Index Terms—**Communicating finite state machines, communication progress, liveness, systolic array, termination, verification, VLSI.**

I. Introduction

KUNG and Leiserson [8] have introduced a new model of computation called "systolic arrays." A systolic array is a network of identical, simple processors; each processor performs a simple function and communicates with its neighboring processors by exchanging messages over connecting channels, or wires. Systolic arrays are useful for two reasons. First, many "large" time-consuming problems (e.g., the multiplication of two large matrices) can be solved extremely fast over large systolic arrays. Second, due to recent advances in VLSI technology, large systolic arrays can be realized and implemented with reasonable cost and effort.

Usually, after designing a systolic array to solve some problem, one is tempted to prove the correctness of the designed array. The proof consists of showing that the array satisfies three types of properties: safety, liveness, and termination. Proving safety properties of systolic arrays is discussed by Ossefort [13], [14] based on a verification methodology developed earlier by Misra and Chandy [11], [12]. In this paper, we discuss techniques to prove liveness and termination properties of systolic arrays.

Manuscript received September 30, 1984; revised May 14, 1985.

The authors are with the Department of Computer Sciences, University of Texas at Austin, Austin, TX 78712.

We model a systolic array as a network of communicating finite state machines that exchange messages over error-free, one-to-one, unbounded, FIFO channels. Each machine has a cyclic behavior: in each cycle, a machine receives one message via each of its input channels and sends one message via each of its output channels. If in some cycle a machine does not have a data message to send via an output channel, it sends a null message instead. Thus machines exchange two types of messages, namely data and null. Because of this cyclic behavior and the fact that machines exchange only two types of messages, the communicating finite state machines in this paper constitute a proper subset of the general class of communicating finite state machines, as defined in [2] and [5].

Networks of communicating finite state machines have been proposed earlier to model communication protocols in computer networks and distributed systems; see for example Bochmann [1], and Zafiropulo *et al.* [17]. However, networks that model systolic arrays differ from those that model communication protocols as follows. The machines in a network that models a systolic array are mostly identical, and the structure of each machine is relatively simple. The converse is usually true for networks that model protocols. For this reason, it is more convenient to develop new "special-purpose" but simpler methodologies to verify liveness and termination for networks that model systolic arrays rather than use "general-purpose" verification methodologies such as those proposed by Owicki and Lamport [10], and Manna and Pnueli [16].

The verification methodology proposed in this paper is based on the concept of a channel history which is a (finite or infinite) sequence of (null or data) messages. The methodology consists of two algorithms. The first algorithm takes 1) a network of communicating finite state machines that models a systolic array, and 2) a finite set of infinite histories, with each history being assigned to one channel in the network. The algorithm then decides whether the set of assigned histories is consistent with the given network. If the answer is positive, then since each of the assigned histories is infinite, each machine is guaranteed to progress infinitely often and the network is live. The second algorithm takes 1) a network of communicating finite state machines, 2) a consistent set of infinite histories,

with each history being assigned to one channel in the network, and 3) a positive integer k. The algorithm then decides whether the network's computation terminates after k steps (i.e., after each machine executes exactly k cycles) under the given channel histories.

Following the introduction, the paper is organized as follows. Networks of communicating finite state machines that model systolic arrays are defined in Section II, followed by the concept of channel histories in Section III. Then two algorithms that can be used to decide liveness and termination of networks are discussed in Sections IV and V, respectively. Three systolic array examples whose liveness and termination properties can be established using our algorithms are discussed in Sections VI, VII, and VIII. They are a linear priority queue [9], a matrix–vector multiplier [8], and a search tree [9]. Section IX contains some concluding remarks.

II. Modeling Systolic Arrays Using Networks of Communicating Finite State Machines

Kung [7] identifies three attributes that define a systolic array: 1) its communication geometry, 2) its data movement, and 3) the function of each processor in the array. In this paper, we use three "structures" to define the first two of these attributes. Specifically, we use

1) a *network topology* to define the communication geometry,

2) a set of *communicating finite state machines*, each of which defines the data movement caused by one processor in the array, and

3) a *network assignment* that assigns one communicating finite state machine to each position in the network.

Notice that this model does not specify the function of each processor in the array since it is irrelevant to the liveness and termination arguments that are the focus of this paper. Next, we define each of these structures in more detail.

A *network topology* T is a labeled directed graph, where vertices are called *positions* and arcs are called *channels*. A channel in T that has both source and sink positions is called an *intermediate channel* of T. A channel in T that has no source (sink) position is called an *external input* (*output*) *channel* of T. If a position p is the sink (source) of a channel c in T, then c is called an *input* (*output*) *channel* of p.

Example 1: Consider the array multiplier in [7] that can be used to multiply a matrix A_{n*r} by a vector X_{r*1}. It consists of a linear array of r identical processors. The ith processor stores in its local memory the ith element x_i of X and is supplied, one by one, with the elements of the ith column in A. The first processor computes the n elements $e_i = a_{i,1}x_1$, $i = 1, \cdots, n$, and forwards them one by one to the second processor. The second processor computes the n elements $f_i = e_i + a_{i,2}x_2$, $i = 1, \cdots, n$, and forwards them one by one to the third processor, and so on. The final (rth) processor computes the n elements that constitute the resulting vector of multiplying A by X and forwards them to the "host computer."

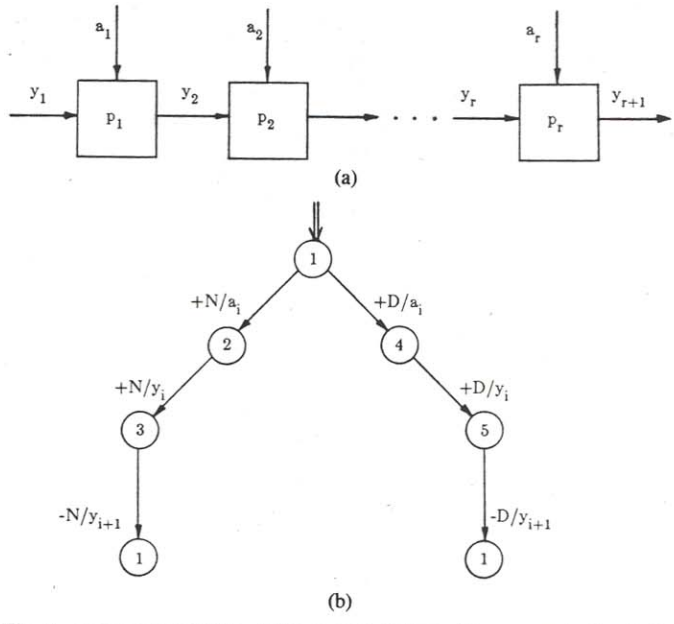

Fig. 1. An array multiplier: (a) network topology, (b) communicating finite state machines M_i, $i = 1, \cdots, r$.

A network topology for this array is shown in Fig. 1(a). It has r positions $p_1, \cdots, p_r$, and $2r + 1$ channels $a_1, \cdots, a_r, y_1, \cdots, y_{r+1}$. Channels $a_1, \cdots, a_r, y_1$ are external input channels, and y_{r+1} is an external output channel. Channel y_1 is provided only so that the processor at position p_1 can be made identical to the other processors; in fact, zero elements are the only useful data to be sent along channel y_1. □

As this example illustrates, a network topology merely defines the "positions" of different processors and their connecting channels in the systolic array. It does not define the order in which each processor sends and/or receives messages via its channels. For that, we need to define a communicating finite state machine for each processor in the array. Communicating finite state machines are discussed next.

A *communicating finite state machine* M is a labeled directed graph with two types of edges called receiving and sending edges. A *receiving* (*sending*) *edge* is labeled $+g/c$ $(-g/d)$, where

- g is a *message* from the set $\{N,D\}$ of messages; N is called a *null message*, and D is called a *data message*;
- c is a *channel* from the set I_M of *input channels* of M; and
- d is a *channel* from the set O_M of *output channels* of M.

(I_M and O_M are disjoint.)

There are two types of nodes in M, called receiving and sending nodes. A *receiving node* has one or two outgoing receiving edges with distinct labels. A *sending node* has one outgoing sending edge. These restrictions ensure that the behavior of M is *deterministic* as explained in Section IX.

One of the nodes in M is identified as its *initial node*.

All the nodes and edges of M constitute a set of directed *cycles*; each cycle passes through the initial node and satisfies the following three conditions.

Fairness: For each input (output) channel c of M, each cycle in M has exactly one receiving (sending) edge labeled $+g/c$ ($-g/c$), for some message g in $\{N,D\}$.

Uniformity: The order in which the channels of M are referenced along each cycle in M is the same.

Receiving Before Sending: Starting from the initial node, receiving edges precede sending edges along each cycle.

Let M be a communicating finite state machine with input channels $c_1, \cdots, c_m$, and output channels $d_1, \cdots, d_n$. From the fairness condition, each cycle in M must reference each channel exactly once, hence each cycle is of length $m + n$. From uniformity condition, each cycle in M must reference the channels of M in the same order. This order can be defined by an ordered tuple, called Order(M), of the $m + n$ channels of M. From the receiving-before-sending condition, all input channels of M must appear before all output channels of M in Order(M). For example

$$\text{Order}(M) = [c_1, \cdots, c_m, d_1, \cdots, d_n].$$

Assume that Order(M) $= [c_1, \cdots, c_m, d_1, \cdots, d_n]$. Define Msgs($M$) to be the set of tuples

$$\{[g_1, \cdots, g_m, g_{m+1}, \cdots, g_{m+n}] \mid \text{there is a cycle in } M \text{ whose edges are labeled, in order, with } +g_1/c_1, \cdots, +g_m/c_m, -g_{m+1}/d_1, \cdots, -g_{m+n}/d_n\}.$$

In other words, each tuple in Msgs(M) defines the sequence of messages received or sent along one cycle in M.

Example 1 (Continued): Each processor in the array multiplier discussed earlier executes the same routine over and over. In each execution of the routine, the processor at position p_i, $i = 1, \cdots, r$, executes the following:

1) It receives one message via each of its input channels a_i and y_{i+1}.
2) **If** the received messages are both data

then the processor computes the result and sends one data message via its output channel y_{i+1}

else {* the received messages are both null *}
it sends one null message via y_{i+1}.

The receiving and sending activities of the processor at position p_i is defined by the communicating finite state machine M_i in Fig. 1(b). M_i consists of two directed cycles that pass through the initial node. Notice that starting from the initial node, two receiving edges precede a single sending edge along each cycle. Notice also that both cycles are equal length (three), and that they reference the channels of M_i in the same order. From Fig. 1(b), we get

$$\text{Order}(M_i) = [a_i, y_i, y_{i+1}] \text{ and } \text{Msgs}(M_i) = \{[N, N, N], [D, D, D]\}.$$ □

A *network* W is a triple $< T, S, A >$, where

T is a network topology as defined earlier;
S is a finite set of communicating finite state machines as defined earlier; and
A is a total function, called the *network assignment*, from the set of positions in T to S such that for every position p in T, both p and its assigned machine $A(p)$ have identical input channels and identical output channels.

Example 1 (Continued): Consider the triple $W = < T, S, A >$, where

T is the network topology in Fig. 1(a);
S is the set of communicating finite state machines $\{M_1, \cdots, M_r\}$, where M_i is defined in Fig. 1(b); and
A is the function that assigns to each position p_i in T, the machine M_i in S, $i = 1, \cdots, r$.

W is a network that models the array multiplier discussed earlier. □

So far, we have only defined the "syntax" of networks; next we define their "semantics." For the remainder of this section, let $W = < T, S, A >$ be a network, whose topology T has r positions $p_1, \cdots, p_r$, and s channels $c_1, \cdots, c_s$, and let M_i be the communicating finite state machine, in S, that is assigned, by A, to position p_i, $i = 1, \cdots, r$.

A *state* q of W is a tuple $q = (v_1, \cdots, v_r; x_1, \cdots, x_s)$, where v_i ($i = 1, \cdots, r$) is a node in machine M_i, and x_j ($j = 1, \cdots, s$) is a finite string over the messages in $\{N, D\}$, called the *contents of channel* c_j in W.

Informally, a state q of W implies that each machine M_i in W has reached node v_i in its execution, while each channel c_j in W contains the sequence of messages x_j.

The *initial state* of network W is a state $(v_1, \cdots, v_r; x_1, \cdots, x_s)$, where each v_i, $i = 1, \cdots, r$, is the initial node in its machine M_i, and each x_j, $j = 1, \cdots, s$, is a string with a single message from the set $\{N, D\}$.

Let $q = (v_1, \cdots, v_r; x_1, \cdots, x_s)$ be a state of W and let e be an edge from node v_i to node v_i' in M_i. A state q' of W is said to *follow q over e* iff one of the following two conditions hold:

1) e is a sending edge labeled $-g/c_j$, $q' = (v_1, \cdots, v_{i-1}, v_i', v_{i+1}, \cdots, v_r; x_1, \cdots, x_{j-1}, x_j', x_{j+1}, \cdots, x_s)$, and $x_j' = x_j; g$, where ";" is the string concatenation operator.

2) e is a receiving edge labeled $+g/c_j$, $q' = (v_1, \cdots, v_{i-1}, v_i', v_{i+1}, \cdots, v_r; x_1, \cdots, x_{j-1}, x_j', x_{j+1}, \cdots, x_s)$, and $x_j = g; x_j'$.

Let q and q' be two states of W. q' is said to *follow* q iff one of the following two conditions is satisfied.

1) There is an edge e such that q' follows q over e.

2) q' is constructed from q by concatenating or appending one message from $\{N, D\}$ to the tail of the contents of one external input channel.

Let q and q' be two states of W. q' is *reachable from* q

iff there exist states $q_1, \cdots, q_n$, where $q = q_1$, $q' = q_n$, and for $i = 1, \cdots, n-1$, q_{i+1} follows q_i.

A state q of W is *reachable* iff it is reachable from the initial state of W.

Later in this paper, we characterize the notions of "liveness" and "proper termination" for networks, and discuss techniques to prove that a given network is live and will terminate properly. But first, we need to define the concept of channel histories.

III. Channel Histories

A *history* is a (finite or infinite) sequence of messages from the set $\{N, D\}$.

We adopt four forms to write a history H, depending on whether H is finite or not, and whether it is repetitious or not:

Form 1 (H is finite and repetitious): In this case, $H = x[k]$, where x is a finite sequence of messages from $\{N, D\}$, and k is a positive integer called the *sequence run* of H. This notation means that $H = x; x; \cdots; x$ repeated k times. If $k = 1$, then it can be dropped yielding $H = x$.

Form 2 (H is finite but not repetitious): In this case, $H = H_1; H_2; \cdots; H_r$, where H_i $(i = 1, \cdots, r)$ is a finite and repetitious history.

Form 3 (H is infinite and repetitious): In this case, $H = x[\infty]$, where x is a finite sequence of messages from $\{N, D\}$. (This means that $H = x; x; \cdots$ repeated infinite times.)

Form 4 (H is infinite but not repetitious): In this case, $H = H_1; H_2; \cdots, H_{r-1}; H_r$, where H_i $(i = 1, \cdots, r-1)$ is a finite and repetitious history, and H_r is an infinite and repetitious history.

Two infinite histories

$$H = x_1[k_1]\ ; x_2[k_2]\ ; \cdots ; x_{r-1}\ [k_{r-1}]\ ; x_r[\infty],$$

and

$$H' = y_1[k_1]\ ; y_2[k_2]\ ; \cdots ; y_{r-1}[k_{r-1}]\ ; y_r[\infty]$$

are said to be in *compatible forms* iff for $i = 1, \cdots, r$, $|x_i| = |y_i|$, where $|x|$ is the number of messages in sequence x.

Let H be a history that has at least i messages; we adopt the following notation:

$H^{(i)}$ denotes the ith message in H, and

H^i denotes the history constructed by removing the first (left-most) i messages from H.

Let M be a communicating finite state machine with channels $d_1, \cdots, d_t$. Assign to each channel d in M, a history, denoted by $H(d)$. The set of assigned histories $\{H(d_1), \cdots, H(d_t)\}$ is said to be *consistent* with M iff for $i = 1, 2, \cdots$, there exists a cycle C in M such that the following two conditions hold:

1) For every input channel d of M, there exists an edge in C labeled $+H(d)^{(i)}/d$.

2) For every output channel d of M, there exists an edge in C labeled $-H(d)^{(i+1)}/d$.

Let $W = < T, S, A >$ be a network with positions $p_1, \cdots, p_r$, and channels $c_1, \cdots, c_s$ in T. Assign to each channel c in T, a history $H(c)$. The set of assigned histories $\{H(c_1), \cdots, H(c_s)\}$ is called *consistent* with W iff for every position p whose input and output channels are $d_1, \cdots, d_t$ in T, the set of histories $\{H(d_1), \cdots, H(d_t)\}$ is consistent with the communicating finite state machine $A(p)$.

Example 1 (Continued): Consider the network $W = < T, S, A >$ of the array multiplier defined earlier. Recall that W has the channels $a_1, \cdots, a_r, y_1, \cdots, y_{r+1}$. Assign to the channels of W, the following histories:

For $i = 1, \cdots, r$, $H(a_i) = N[i]; D[n]; N[\infty]$.

For $i = 1, \cdots, r+1$, $H(y_i) = N[i]; D[n]; N[\infty]$.

It is required to prove that this set of assigned histories is consistent with W. In the next section, we present an algorithm to decide whether a given set of assigned histories is consistent with a given network, then apply this algorithm to the current example to show that these assigned histories are indeed consistent with W.

□

IV. Proving Liveness

Informally, a network $W = < T, S, A >$ is live iff the machine $A(p)$ assigned to each position p in T is guaranteed to progress infinitely often, i.e., is guaranteed to receive and send (data or null) messages infinitely often. As mentioned earlier, each machine consists of some cycles, and along each cycle, the machine must access each of its channels exactly once. Therefore, establishing indefinite progress of a machine is equivalent to establishing that each of its (input or output) channels has an infinite history. This observation motivates the following definition.

Let $W = < T, S, A >$ be a network with channels $c_1, \cdots, c_s$, and let $F = \{H(c_1), \cdots, H(c_s)\}$ be a set of infinite histories assigned to the channels of W. W is said to be *live under* F iff F is consistent with W.

Given a network $W = < T, S, A >$ with channels $c_1, \cdots, c_s$, and given a set $F = \{H(c_1), \cdots, H(c_s)\}$ of infinite histories which are assigned to the channels of W, it is required to decide whether W is live under F. This decision can be made using Algorithm 1, discussed below, to decide whether the set of histories assigned to the input and output channels of each position p in T is consistent with the communicating finite state machine $A(p)$.

Algorithm 1 takes a communicating finite state machine M, and a set of infinite histories assigned to the channels of M; it then decides whether the given set is consistent with M. The basic idea of the algorithm is simple enough. First align the given histories to one another. Then for each i, construct a tuple that consists of the ith message from each history. Thus, the ith tuple consists of the ith messages supposedly received or sent by M along its various channels. If so, these messages should have been received or sent as M executes exactly one cycle, and the tuple must exist in Msgs(M). Therefore the last step of the algorithm is to check whether each of these tuples is in Msgs(M). It is straightforward to show that the time com-

plexity of the algorithm is polynomial in the size of its inputs.

Algorithm 1

Input: A communicating finite state machine M with input channels $c_1, \cdots, c_m$ and output channels $d_1, \cdots, d_n$ and a set of infinite histories $\{H(c_1), \cdots, H(c_m), H(d_1), \cdots, H(d_n)'\}$ assigned to the channels of M, respectively.

Output: A decision of whether the set of assigned histories is consistent with M.

Steps:

1) Define Order(M) and Msgs(M). Without loss of generality, let Order(M) $= [c_1, \cdots, c_m, d_1, \cdots, d_n]$.

2) Remove the leftmost message from the histories assigned to the output channels of M; i.e., construct the histories $H(d_1)^1, \cdots, H(d_n)^1$ from $H(d_1), \cdots, H(d_n)$, respectively.

3) Rewrite the histories $H(c_1), \cdots, H(c_m), H(d_1)^1, \cdots, H(d_n)^1$ in compatible forms as follows:

$$H(c_1) = u_1[k_1]; \cdots; u_r[k_r],$$
$$\cdots$$
$$H(c_m) = v_1[k_1]; \cdots; v_r[k_r],$$
$$H(d_1)^1 = x_1[k_1]; \cdots; x_r[k_r],$$
$$\cdots$$
$$H(d_n)^1 = y_1[k_1]; \cdots; y_r[k_r].$$

In other words, for $i = 1, \cdots, r$, $|u_i| = \cdots = |v_i| = |x_i| = \cdots = |y_i|$.

4) Reduce each of these infinite histories into a finite history by reducing each sequence run in each history to one. Let $R(c_i)$ be the reduced histories of $H(c_i)$, $i = 1, \cdots, m$, and $R(d_i)$ be the reduced history of $H(d_i)^1$, $i = 1, \cdots, n$. These reduced histories are as follows:

$$R(c_1) = u_1; \cdots; u_r,$$
$$\cdots$$
$$R(c_m) = v_1; \cdots; v_r,$$
$$R(d_1) = x_1; \cdots; x_r,$$
$$\cdots$$
$$R(d_n) = y_1; \cdots; y_r.$$

5) Let k be the length of each reduced history obtained from Step 4). The set of assigned histories $\{H(c_1), \cdots, H(c_m), H(d_1), \cdots, H(d_n)\}$ is consistent with M iff for every j, $j = 1, \cdots, k$, $[R(c_1)^{(j)}, \cdots, R(c_m)^{(j)}, R(d_1)^{(j)}, \cdots, R(d_n)^{(j)}]$ is in Msgs(M).

□

Example 1 (Continued): Let $W = < T, S, A >$ be the network of array multiplier, where T and M_i in S are defined in Fig. 1(a) and (b), respectively. We now show that the following set of histories assigned to the channels in T is consistent with W.

For $i = 1, \cdots, r$, $H(a_i) = N[i]\ ;D[n]\ ;N[\infty]$.

For $i = 1, \cdots, r+1$, $H(y_i) = N[i]\ ;D[n]\ ;N[\infty]$.

The communicating finite state machine M_i is assigned to position p_i in T, $i = 1, \cdots, r$. Its channels are assigned the following histories:

$$H(a_i) = N[i]\ ;D[n]\ ;N[\infty],$$
$$H(y_i) = N[i]\ ;D[n]\ ;N[\infty],$$

and

$$H(y_{i+1}) = N[i+1]\ ;D[n]\ ;N[\infty].$$

Following Algorithm 1, we get Order(M_i) $= [a_i, y_i, y_{i+1}]$ and Msgs(M_i) $= \{[N, N, N], [D, D, D]\}$ from Step 1). From Step 2), we have $H(y_{i+1})^1 = N[i-1];D[n];N[\infty]$. We can skip Step 3) since $H(a_i)$, $H(y_i)$, and $H(y_{i+1})^1$ are already in compatible forms. In Step 4), we get the following reduced histories:

$$R(a_i) = NDN,$$
$$R(y_i) = NDN,$$

and

$$R(y_{i+1}) = NDN.$$

It is straightforward to see the following (Step 5):

$[R(a_i)^{(1)}, R(y_i)^{(1)}, R(y_{i+1})^{(1)}] = [N, N, N]$ is in Msgs(M_i).

$[R(a_i)^{(2)}, R(y_i)^{(2)}, R(y_{i+1})^{(2)}] = [D, D, D]$ is in Msgs(M_i).

$[R(a_i)^{(3)}, R(y_i)^{(3)}, R(y_{i+1})^{(3)}] = [N, N, N]$ is in Msgs(M_i).

Therefore, $\{H(a_i), H(y_i)$, and $H(y_{i+1})\}$ is consistent with M_i. Since this is true for every machine M_i, $i = 1, \cdots, r$, so the set of assigned histories is consistent with W.

□

V. Proving Termination

An infinite history $H = x_1[k_1]; \cdots; x_{r-1}[k_{r-1}]; x_r[\infty]$ is said to *return to its initial pattern* iff $x_1 = x_r$.

Let $H = x_1[k_1]; \cdots; x_{r-1}[k_{r-1}];x_r[\infty]$ be an infinite history that returns to its initial pattern, and let k be a positive integer. H is said to *return to its initial pattern in k steps* iff there exists a natural number s ($s = 0, 1, \cdots$), such that $k = |x_1| * k_1 + \cdots + |x_{r-1}| * k_{r-1} + |x_r| * s$, where $|x|$ is the number of messages in string x, and "*" is the usual integer multiplication operator.

Let $W = < T, S, A >$ be a network with channels $c_1, \cdots, c_s$, and let $F = \{H(c_1), \cdots, H(c_s)\}$ be a consistent set of infinite histories for the channels of W. Also let k be a positive integer. W is said to *terminate properly under F in k steps* iff each history in F returns to its initial pattern in k steps.

The following algorithm takes a network W, a consistent set F of infinite histories assigned to the channels of W, and a positive integer k, and decides whether W terminates properly after k steps under F. The basic idea of the al-

gorithm is to check whether each channel history in F returns to its initial pattern in k steps. The time complexity of the algorithm is polynomial in the size of its input.

Algorithm 2

Input: A network W, a consistent set F of infinite histories for the channels in W, and a positive integer k. {* Use Algorithm 1 to decide whether F is indeed consistent with W *}

Output: A decision of whether network W terminates properly after k steps under F.

Steps:

1) **for** each history $H = x_1[k_1]; \cdots ; x_{r-1}[k_{r-1}];x_r[\infty]$ in F **do**

   if $x_1 \neq x_r$ {* the history does not return to its initial pattern *}

   then stop: W does not terminate properly under F.

2) **for** each history $H = x_1[k_1]; \cdots ; x_{r-1}[k_{r-1}];x_r[\infty]$ in F **do**

   a) compute the function $f_H(s)$ as follows: $f_H(s) = |x_1| * k_1 + \cdots + |x_{r-1}| * k_{r-1} + |x_r| * s$, where $|x|$ is the number of messages in sequence x, "*" is the integer multiplication operator, and s is a natural number 0, 1, $\cdots$;

   b) **if** there is no value of s such that $k = f_H(s)$ {* the history does not return to its initial pattern in k steps *}

   then stop W does not terminate properly under F in k steps.

3) **stop** W terminates properly under F in k steps.

□

Example 1 (Continued): Let W be the network of the array multiplier discussed earlier, and let F be the set of histories assigned earlier to the channels $a_1, \cdots, a_r, y_1, \cdots, y_{r+1}$ of W. We use Algorithm 2 to show that W will terminate properly under F after $k = n + r + 1$ steps. Recall from the preceding section that F is consistent with W.

Each history in F is of the form $x_1[k_1];x_2[k_2];x_3[\infty]$, where $x_1 = x_3 = N$. Therefore, each history in F returns to its initial pattern as required by Step 1) in Algorithm 2.

According to Step 2), we need to compute the function $f_H(s)$ for each history in F, and show that there is a value of s that makes $k = f_H(s)$. For each history $H(a_i) = N[i]; D[n];N[\infty]$, $i = 1, \cdots, r$, in F, we have $f_H(s) = i + n + s$. Therefore, the value of $s = r + 1 - i$ ensures that $k = f_H(s)$. For each history $H(y_i) = N[i]; D[n]; N[\infty]$, $i = 1, \cdots, r + 1$, in F, we have $f_H(s) = i + n + s$. Therefore, the value of $s = r + 1 - i$ ensures that $k = f_H(s)$. This completes the proof that W terminates properly under F in $n + r + 1$ steps.

□

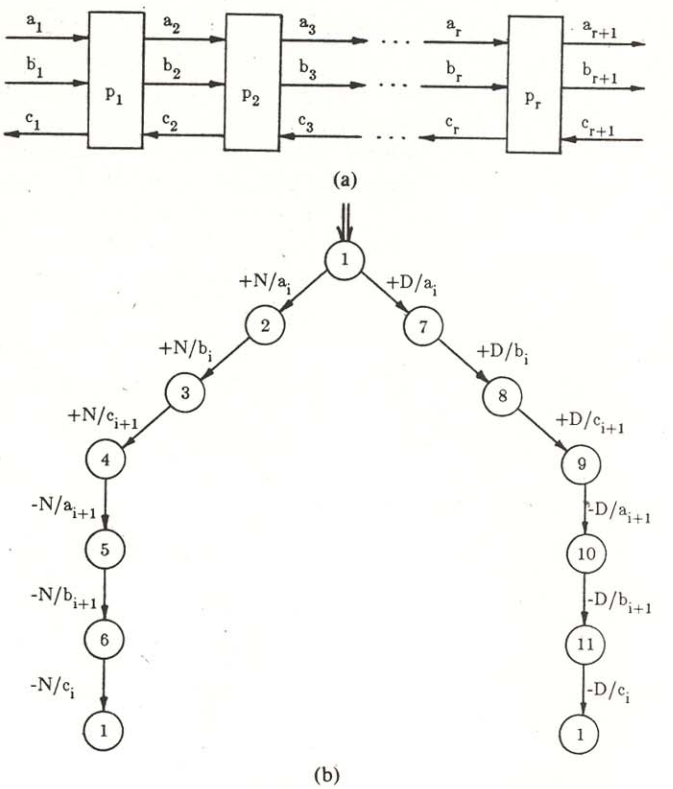

Fig. 2. A linear priority queue: (a) network topology, (b) M_i, $i = 1, \cdots, r$.

VI. A Linear Priority Queue Example

A. Network

Consider the linear priority queue defined in [9]. Using our notation, this queue can be defined by the network $W = < T, S, A >$, where

- T is the network topology in Fig. 2(a),
- S is a set of communicating finite state machines $\{M_1, \cdots, M_r\}$, where M_i is defined in Fig. 2(b), and
- A is the network assignment that assigns to each position p_i in T, the communicating finite state machine M_i in S, $i = 1, \cdots, r$.

Later, we prove the liveness and termination of W using the two algorithms discussed earlier, but first we give an informal explanation on how W acts as a priority queue. (This explanation is not needed for the proof; it is provided only for the reader's convenience.)

As shown in Fig. 2(a), W has r positions, $p_1, \cdots, p_r$, and $3(r + 1)$ channels, $a_1, \cdots, a_{r+1}, b_1, \cdots, b_{r+1}, c_1, \cdots, c_{r+1}$. Channels $a_1, b_1, c_1, a_{r+1}, b_{r+1}$, and c_{r+1} are external channels and so can communicate messages to and from the external environment (i.e., the host machine).

To insert a value "a" into the queue, the two values "a" and "$-\infty$" should be placed into the two external channels a_1 and b_1, respectively. Moreover, each of these values should be followed by a null message in its channel. (Recall that our model does not distinguish between the different values of data messages; i.e., each of the values "a" and "$-\infty$" is modeled by a data message D.)

To extract the current minimum value from the queue, a value "$+\infty$" followed by a null message should be placed into each of the two channels a_1 and b_1. (As before, each "$+\infty$" value is modeled by a data message D.)

To stabilize the contents of the queue, the two values "$+\infty$" and "$-\infty$," should be placed into channels a_1 and b_1, respectively. Each of these values should be followed by a null message in its channel.

Initially, every input channel of an even-numbered position (p_i, $i = 2, 4, 6, \cdots$) has one null message N, and every input channel of an odd-numbered position (p_i, $i = 1, 3, 5, \cdots$) has one data message D. (The data messages in a_1 and b_1 model the initial input to the queue; the rest of data messages model "$+\infty$" values.)

At each cycle, the processor assigned to position p_i, $i = 1, \cdots, r$, executes the following routine:

1) It receives one message from each of its input channels a_i, b_i, and c_{i+1}.
2) **If** all the received messages are null
   then the processor sends one null message via each of its output channels a_{i+1}, b_{i+1}, and c_i.
   else {* all the received messages are data *}
   the processor sorts the received values then sends the smallest value via channel c_i and the other two values via channels a_{i+1} and b_{i+1}.

B. Channel Histories

We assign the channels in W the following histories:
For $i = 1, 3, 5, \cdots, 2\lfloor r/2 \rfloor + 1$,

$$H(a_i) = DN[\infty],$$

$$H(b_i) = DN[\infty],$$

and

$$H(c_i) = ND[\infty].$$

For $i = 2, 4, 6, \cdots, 2\lceil r/2 \rceil$,

$$H(a_i) = ND[\infty],$$

$$H(b_i) = ND[\infty],$$

and

$$H(c_i) = DN[\infty].$$

C. Proving Liveness

We use Algorithm 1 to show that the set of histories assigned to the channels of M_i is consistent with M_i, $i = 1, \cdots, r$. We carry the proof for the odd-position machines M_i, $i = 1, 3, 5, \cdots$. (The proof for the other machines is similar.)

Each M_i, $i = 1, 3, 5, \cdots$, in S has three input channels a_i, b_i, and c_{i+1}, and three output channels a_{i+1}, b_{i+1}, and c_i, with the following assigned histories:

$$H(a_i) = H(b_i) = H(c_{i+1}) = DN[\infty],$$

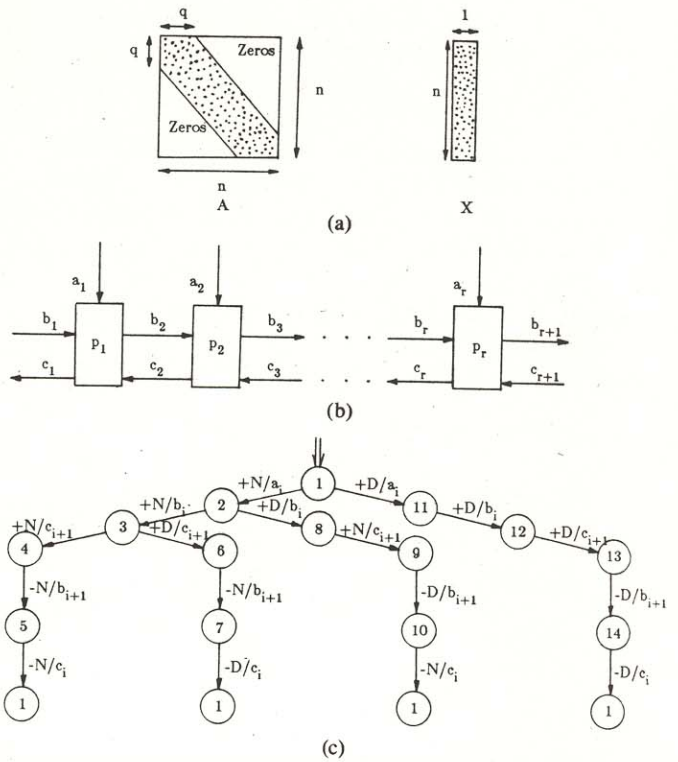

Fig. 3. A matrix-vector multiplier: (a) a banded matrix $A_{n\times n}$ of width $2q - 1$, and a vector $X_{n\times 1}$, (b) network topology, (c) M_i, $i = 1, \cdots, r$.

and

$$H(a_{i+1}) = H(b_{i+1}) = H(c_i) = ND[\infty].$$

Following Algorithm 1, the reduced histories of these channels are as follows: $R(a_i) = R(b_i) = R(c_{i+1}) = R(a_{i+1}) = R(b_{i+1}) = R(c_i) = DN$.

Since

$$\text{Order}(M_i) = [a_i, b_i, c_{i+1}, a_{i+1}, b_{i+1}, c_i],$$

and

$$\text{Msgs}(M_i) = \{[N,N,N,N,N,N], [D,D,D,D,D,D]\},$$

then each of the two tuples

$$[R(a_i)^{(1)}, R(b_i)^{(1)}, R(c_{i+1})^{(1)}, R(a_{i+1})^{(1)}, R(b_{i+1})^{(1)}, R(c_i)^{(1)}]$$

and

$$[R(a_i)^{(2)}, R(b_i)^{(2)}, R(c_{i+1})^{(2)}, R(a_{i+1})^{(2)}, R(b_{i+1})^{(2)}, R(c_i)^{(2)}]$$

is in $\text{Msgs}(M_i)$. Therefore, the assigned histories to the channels of M_i are consistent with M_i. This implies the liveness of network W under the assigned histories.

D. Proving Termination

Using Algorithm 2, it is straightforward to show that W terminates properly after an even number $\{2, 4, 6, \cdots\}$ of steps under the assigned histories.

VII. A Matrix–Vector Multiplier Example

A. Network

Consider the matrix–vector multiplier described in [8]. It can be used to multiply a band matrix A_{n*n} of width $r = 2q - 1$, with a vector X_{n*1}; see Fig. 3(a). A network

for this multiplier is $W = <T, S, A>$, where

- T is the network topology in Fig. 3(b),
- S is a set of communicating finite state machines $\{M_1, \cdots, M_r\}$, where M_i, $i = 1, \cdots, r$, is the communicating finite state machine defined in Fig. 3(c), and
- A is the network assignment that assigns to each position p_i in T, the communicating finite state machine M_i in S, $i = 1, \cdots, r$.

The following is an informal explanation on how W performs a matrix–vector multiplication.

Initially, each channel in the array has a null message. The elements of vector X and matrix A are entered into the array via its external input channels as follows.

1) The n elements of vector X are entered via channel b_1 as n data messages, with a null message being inserted between any two of them.

2) Zero values are entered via channel c_{r+1} as data messages, with a null message being inserted between any two of them.

3) The elements of matrix A are entered via channels $a_1, \cdots, a_r$. Elements of the main diagonal are entered via channel a_q. Elements of the subdiagonal with distance i to the right (left) of the main diagonal are entered via channel $a_{q-i}(a_{q+i})$. In each case, the (sub)diagonal elements are entered as data messages with a null message being inserted between any two of them.

The processor assigned to position p_i, $i = 1, \cdots, r$, executes, over and over, the following routine.

1) It receives one message from each of its input channels a_i, b_i, and c_{i+1}.
2) **If** all the received messages are data,

   then it multiplies the values received from channels a_i and b_i and adds the product to the value received from channel c_{i+1}; it then sends the result via channel c_i and forwards the value received from channel b_i via channel b_{i+1}

   else {* only the message received from channel b_i (c_{i+1}) is data, while the other two received message are null *}
   it forwards the messages received from channels b_i and c_{i+1} via channels b_{i+1} and c_i, respectively.

Elements of the vector that results from multiplying A with X are produced by the array as data messages at the external output channel c_1.

B. Channel Histories

We assign the channels in W the following histories:

For $i = 1, \cdots, q$,

$$H(a_i) = N[2q - i] \quad ;DN[i + n - q] \quad ;N[\infty],$$

$$H(b_i) = N[i] \quad ;DN[n] \quad ;N[\infty],$$

and

$$H(c_i) = N[2q - i + 1] \quad ;DN[n] \quad ;N[\infty].$$

For $i = q + 1, \cdots, r$,

$$H(a_i) = N[i] \quad ;DN[n + q - i] \quad ;N[\infty],$$

$$H(b_i) = N[i] \quad ;DN[n] \quad ;N[\infty],$$

and

$$H(c_i) = N[2q - i + 1] \quad ;DN[n] \quad ;N[\infty].$$

For $i = r + 1$,

$$H(b_i) = N[i] \quad ;DN[n] \quad ;N[\infty],$$

and

$$H(c_i) = N[2q - i + 1] \quad ;DN[n] \quad ;N[\infty].$$

C. Proving Liveness

We use Algorithm 1 to show that the histories assigned to the channels of M_i are consistent with M_i, $i = 1, \cdots, r$. We carry the proof for the machines in the first half of the array, i.e., M_i, $i = 1, \cdots, q$. (The proof for the other machines is similar.)

Each M_i ($i = 1, \cdots, q$) in S has three input channels, a_i, b_i, and c_{i+1}, and two output channels, b_{i+1} and c_i, with the following assigned histories:

$$H(a_i) = N[2q-i] \quad ;DN[i+n-q] \quad ;N[\infty],$$

$$H(b_i) = N[i] \quad ;DN[n] \quad ;N[\infty],$$

$$H(c_{i+1}) = N[2q-i] \quad ;DN[n] \quad ;N[\infty],$$

$$H(b_{i+1}) = N[i+1] \quad ;DN[n] \quad ;N[\infty],$$

and

$$H(c_i) = N[2q-i+1] \quad ;DN[n] \quad ;N[\infty].$$

By removing the leftmost message from the histories assigned to the output channels, we get

$$H(b_{i+1})^1 = H(b_i) \quad \text{and} \quad H(c_i)^1 = H(c_{i+1}).$$

The histories can be written in compatible forms as follows:

$$H(a_i) = N[i] \;;NN[q-i] \;;DN[i+n-q] \;;NN[q-i] \;;N[\infty],$$

$$H(b_i) = N[i] \;;DN[q-i] \;;DN[i+n-q] \;;NN[q-i] \;;N[\infty],$$

$$H(c_{i+1}) = N[i] \;;NN[q-i] \;;DN[i+n-q] \;;DN[q-i] \;;N[\infty],$$

$$H(b_{i+1})^1 = N[i] \;;DN[q-i] \;;DN[i+n-q] \;;NN[q-i] \;;N[\infty],$$

and

$$H(c_i)^1 = N[i] \;;NN[q-i] \;;DN[i+n-q] \;;DN[q-i] \;;N[\infty].$$

From Algorithm 1, the reduced histories of the above histories can be written as follows:

$$R(a_i) = NNNDNNNN,$$

$$R(b_i) = NDNDNNNN,$$

$$R(c_{i+1}) = NNNDNDNN,$$

$$R(b_{i+1}) = NDNDNNNN,$$

and

$$R(c_i) = NNNDNDNN.$$

It is clear that for $j = 1, \cdots, 8$, $[R(a_i)^{(j)}, R(b_i)^{(j)}, R(c_{i+1})^{(j)}, R(b_{i+1})^{(j)}, R(c_i)^{(j)}]$ is in $\text{Msgs}(M_i)$, $i = 1, \cdots, q$. Hence, the histories assigned to the channels of M_i are consistent with M_i. This implies the liveness of W under the assigned histories.

D. Proving Termination

Using Algorithm 2, it is straightforward to show that W terminates properly in $2q + 2n$ steps under the assigned histories.

VIII. A Search Tree Example

A. Network

Consider the search tree discussed in [9]; it can be defined by the network $W = <T, S, A>$, where

- T is the network topology defined in Fig. 4(a) with positions $p_{i,j}$, $i = 1, \cdots, r$ and $j = 1, \cdots, 2^{i-1}$,
- S is a set of communicating finite state machines $\{M_{i,j} | i = 1, \cdots, r, \text{ and } j = 1, \cdots, 2^{i-1}\}$, where $M_{i,j}$ is defined in Fig. 4(b) and (c) and
- A assigns to each position $p_{i,j}$ in T, the communicating finite state machine $M_{i,j}$ in S, $i = 1, \cdots, r$, and $j = 1, \cdots, 2^{i-1}$.

First, we give an informal explanation on how W operates as a search tree.

As shown in Fig. 4(a), the network topology of W consists of a balanced binary tree whose leaves are connected together into a linear queue. Each processor assigned to a leaf position stores one value. Initially, each leaf processor stores a "$+\infty$" value, and each $a_{i,j}$ or $b_{i,j}$ channel has a null message, and each c_i or d_i channel has a data message whose value is "$+\infty$."

To insert a value "a" into the tree, a data message whose value "insert (a)" is entered into the external input channel $a_{1,1}$. This message is broadcast down the tree to each leaf processor. The appropriate leaf processor stores this value "a" instead of its previous value, and sends back an acknowledging data message whose value is "$+\infty$." Each of the other leaf processors does not store "a," but sends back an acknowledging data message whose value is "$+\infty$." The acknowledging data messages climb the tree and are merged together, until they are delivered as one acknowledging message with value "$+\infty$" via the external output channel $b_{1,1}$. The reception of this message from channel $b_{1,1}$ indicates the completion of the "insert(a)" operation.

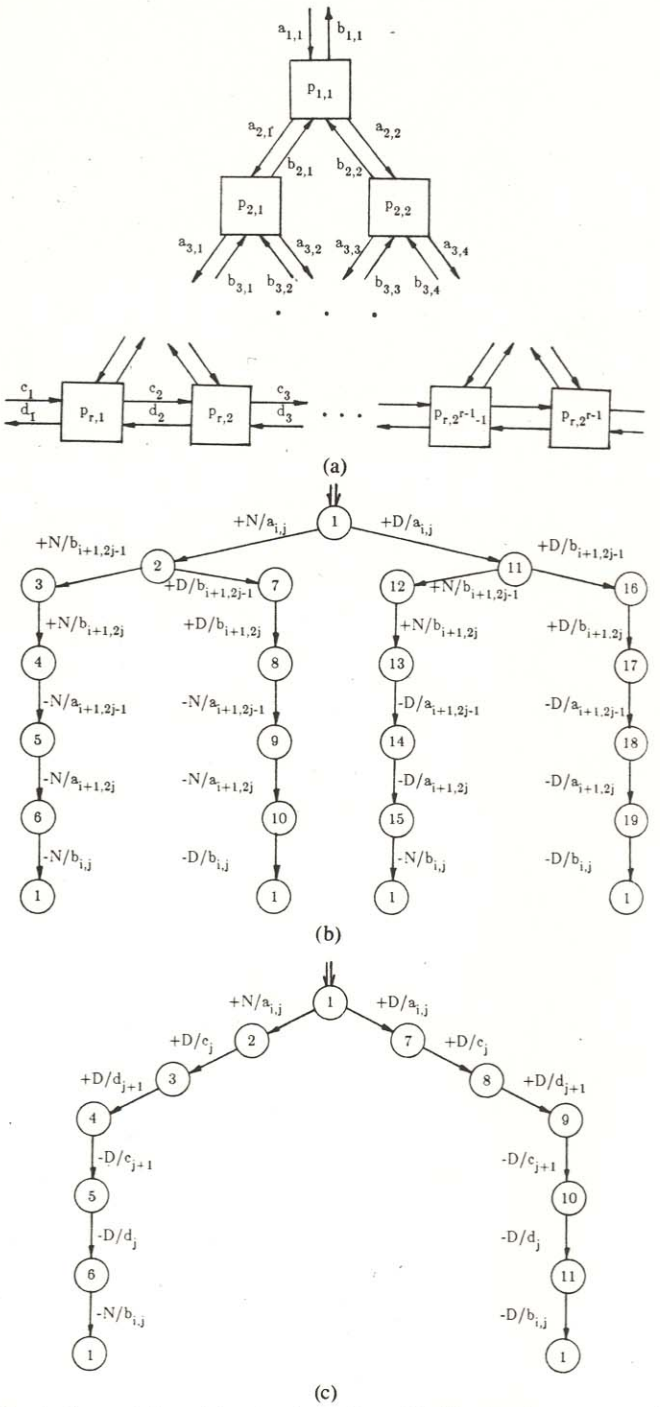

Fig. 4. A search tree: (a) network topology, (b) $M_{i,j}$, $i = 1, \cdots, r-1$, and $j = 1, \cdots, 2^{i-1}$, (c) $M_{i,j}$, $i = r$, and $j = 1, \cdots, 2^{i-1}$.

To extract the smallest value, in the tree, that is greater than or equal to a given value "b," a data message whose value is "extract(b)" is entered into the external input channel $a_{1,1}$. This message is broadcast down the tree to each leaf processor. The appropriate processor sends back its current value as a data message. Each of the other leaf processors sends back an acknowledging data message whose value is "$+\infty$." The tree filters out the value sent by the appropriate leaf processor and delivers it to the external output channel $b_{1,1}$.

The routine executed by the processor assigned to a

nonleaf position $p_{i,j}$ ($i = 1, \cdots, r-1$, and $j = 1, \cdots, 2^{i-1}$) is as follows.

1) It receives one message from each of its input channels, $a_{i,j}$, $b_{i+1,2j-1}$, and $b_{i+1,2j}$.

2) It then sends copies of the message that it received from channel $a_{i,j}$ via channels $a_{i+1;2j-1}$ and $a_{i+1,2j}$.

3) **If** the two messages received from channels $b_{i+1,2j-1}$ and $b_{i+1,2j}$ are null messages

then it sends a message via channel $b_{i,j}$

else {* the messages received from channels $b_{i+1,2j-1}$ and $b_{i+1,2j}$ are data messages *} it sends the smaller of the two received values via channel $b_{i,j}$.

The routine executed by the processor assigned to a leaf position $p_{i,j}$ ($i = r$, and $j = 1, \cdots, 2^{i-1}$) is as follows.

1) It receives one message from each of its input channels, $a_{i,j}$, c_j, and d_{j+1}.

2) **case** message received from channel $a_{i,j}$ **of**

a) null message:

the processor sends a null message via channel $_{i,j}$

b) insert(a):

if $v_{i-1} < a$ and $v_i > a$, where v_{i-1} is the value receive from channel c_j, and v_i is the value stored the processor

then the processor stores "a" instead of "v_i," and sends an ∞ value via channel $b_{i,j}$ as an acknowledgement

else if $v_{i-1} > a$, where v_{i-1} is the value received from channel c_j

then the processor stores the value "v_{i-1}" instead of its previous value, and sends an ∞ value via channel $b_{i,j}$ as an acknowledgment

else {* $v_i < a$, where v_i is the value stored in the processor *} the processor keeps the value "v_i" as it is, and sends an ∞ value as an acknowledgment.

c) extract(b):

if $v_{i-1} < a$ and $v_i > a$, where v_{i-1} is the value received from channel c_j, and v_i is the value stored in the processor

then the processor stores the value "v_{i+1}" instead of its previous value "v_i," where "v_{i+1}" is the value received from channel d_{j+1}, and sends "v_i" via channel $b_{i,j}$ as an acknowledgment

else if $v_{i-1} \geq b$, where v_{i-1} is the value received from channel c_j

then the processor stores the value "v_{i+1}" instead of "v_i," where "v_{i+1}" is the value received from channel d_{j+1} and "v_i" is the value stored in the processor, and sends an ∞ value via channel $b_{i,j}$ as an acknowledgment

else {* $v_i < b$, where v_i is the value stored in the processor *} the processor keeps the value "v_i" as it is, and sends an ∞ as an acknowledgment.

3) In each of the above cases, the processor sends the value stored in it via channels c_{j+1} and d_j.

B. Channel Histories

We assign the channels in W the following histories:

For $i = 1, \cdots, r$, and $j = 1, \cdots, 2^{i-1}$,

$$H(a_{i,j}) = N[i] \quad ;D[k] \quad ;N[\infty],$$

$$H(b_{i+1,2j-1}) = N[2r-i] \quad ;D[k] \quad ;N[\infty],$$

and

$$H(b_{i+1,2j}) = N[2r-i] \quad ;D[k] \quad ;N[\infty],$$

where $k = 1, 2, 3, \cdots$.

For $i = 1, \cdots, 2^{r-1} + 1$,

$$H(c_i) = D[\infty],$$

and

$$H(d_i) = D[\infty].$$

C. Proving Liveness

We use Algorithm 1 to show that the set of histories assigned to the channels of $M_{i,j}$ is consistent with $M_{i,j}$, $i = 1, \cdots, r$, and $j = 1, \cdots, 2^{i-1}$. We carry the proof for the nonleaf machines, i.e., $M_{i,j}$, $i = 1, \cdots, r-1$, and $j = 1, \cdots, 2^{i-1}$. (The proof for the leaf machines is similar.)

Each $M_{i,j}$, ($i = 1, \cdots, r-1$, and $j = 1, \cdots, 2^{i-1}$) in S has three input channels $a_{i,j}$, $b_{i+1,2j-1}$, and $b_{i+1,2j}$, and three output channels $a_{i+1,2j-1}$, $a_{i+1,2j}$, and $b_{i,j}$ with the following assigned histories:

$$H(a_{i,j}) = N[i] \quad ;D[k] \quad ;N[\infty],$$

$$H(b_{i+1,2j-1}) = N[2r-i] \quad ;D[k] \quad ;N[\infty],$$

$$H(b_{i+1,2j}) = N[2r-i] \quad ;D[k] \quad ;N[\infty],$$

$$H(a_{i+1,2j-1}) = N[i+1] \quad ;D[k] \quad ;N[\infty],$$

$$H(a_{i+1,2j}) = N[i+1] \quad ;D[k] \quad ;N[\infty],$$

and

$$H(b_{i,j}) = N[2r-i+1] \quad ;D[k] \quad ;N[\infty],$$

By removing the leftmost messages from the histories assigned to the output channels, then writing the histories in compatible forms, we get the following histories. (There are three cases depending on whether k is less than, equal, or greater than $2r - 2i$.)

Case 1 ($k < 2r - 2i$):

$$H(a_{i,j}) = N[i] \;;D[k] \;;N[2r-2i-k] \;;N[k] \;;N[\infty],$$

$$H(b_{i+1,2j-1}) = N[i] \;;N[k] \;;N[2r-2i-k] \;;D[k] \;;N[\infty],$$

$$H(b_{i+1,2j}) = N[i] \;;N[k] \;;N[2r-2i-k] \;;D[k] \;;N[\infty],$$

$$H(a_{i+1,2j-1})^1 = N[i]\ ;D[k]\ ;N[2r-2i-k]\ ;N[k]\ ;N[\infty],$$

$$H(a_{i+1,2j})^1 = N[i]\ ;D[k]\ ;N[2r-2i-k]\ ;N[k]\ ;N[\infty],$$

and

$$H(b_{i,j})^1 = N[i]\ ;N[k]\ ;N[2r-2i-k]\ ;D[k]\ ;N[\infty].$$

Case 2 ($k = 2r - 2i$):

$$H(a_{i,j}) = N[i]\ ;D[k]\ ;N[k]\ ;N[\infty],$$

$$H(b_{i+1,2j-1}) = N[i]\ ;N[k]\ ;D[k]\ ;N[\infty],$$

$$H(b_{i+1,2j}) = N[i]\ ;N[k]\ ;D[k]\ ;N[\infty],$$

$$H(a_{i+1,2j-1})^1 = N[i]\ ;D[k]\ ;N[k]\ ;N[\infty],$$

$$H(a_{i+1,2j})^1 = N[i]\ ;D[k]\ ;N[k]\ ;N[\infty],$$

and

$$H(b_{i,j})^1 = N[i]\ ;N[k]\ ;D[k]\ ;N[\infty].$$

Case 3 ($k > 2r - 2i$):

$$H(a_{i,j}) = N[i]\ ;D[2r-2i]\ ;D[k+2i-2r]\ ;N[2r-2i]\ ;N[\infty],$$

$$H(b_{i+1,2j-1}) = N[i]\ ;N[2r-2i]\ ;D[k+2i-2r]\ ;D[2r-2i]\ ;N[\infty],$$

$$H(b_{i+1,2j}) = N[i]\ ;N[2r-2i]\ ;D[k+2i-2r]\ ;D[2r-2i]\ ;N[\infty],$$

$$H(a_{i+1,2j-1})^1 = N[i]\ ;D[2r-2i]\ ;D[k+2i-2r]\ ;N[2r-2i]\ ;N[\infty],$$

$$H(a_{i+1,2j})^1 = N[i]\ ;D[2r-2i]\ ;D[k+2i-2r]\ ;N[2r-2i]\ ;N[\infty],$$

and

$$H(b_{i,j})^1 = N[i]\ ;N[2r-2i]\ ;D[k+2i-2r]\ ;D[2r-2i]\ ;N[\infty].$$

We carry the rest of the proof for the compatible histories in Case 1. (The proofs for Cases 2 and 3 are similar.)

The reduced histories of the above histories can be written as follows:

$$R(a_{i,j}) = NDNNN,$$

$$R(b_{i+1,2j-1}) = NNNDN,$$

$$R(b_{i+1,2j}) = NNNDN,$$

$$R(a_{i+1,2j-1}) = NDNNN,$$

$$R(a_{i+1,2j}) = NDNNN,$$

and

$$R(b_{i,j}) = NNNDN.$$

It is clear that for $k = 1, \cdots, 5$, the tuple $[R(a_{i,j})^{(k)}, R(b_{i+1,2j-1})^{(k)}, R(b_{i+1,2j})^{(k)}, R(a_{i+1,2j-1})^{(k)}, R(a_{i+1,2j})^{(k)}, R(b_{i,j})^{(k)}]$ is in $\mathrm{Msgs}(M_{i,j})$, $i = 1, \cdots, r-1$, and $j = 1, \cdots, 2^{i-1}$. Hence, the histories assigned to the channels of $M_{i,j}$ are consistent with $M_{i,j}$. This implies the liveness of W under the assigned histories.

D. Proving Termination

Using Algorithm 2, it is straightforward to show that W terminates properly in $k + 2r - 1$ steps under the assigned histories.

IX. Concluding Remarks

We have presented two algorithms to decide liveness and termination properties for networks of communicating finite state machines that model systolic arrays. We have demonstrated the usefulness of these algorithms by using them to establish the liveness and termination properties for some systolic array examples taken from the literature. Besides the four examples discussed in this paper, we have used the algorithms to verify two more examples, namely a hexagonal array for matrix multiplication [8], and a "triangle-like" array to perform dynamic programming [6]. The details of these two examples were similar to those already discussed in the present paper.

The communicating finite state machines discussed in this paper exchange only two types of messages, namely data D and null N. Although these machines seem adequate for modeling systolic arrays, our results, especially Algorithms 1 and 2, can be extended in a straightforward fashion to include machines that exchange many types of messages (but still satisfy the conditions of cyclic and deterministic behavior, fairness, uniformity, and receiving before sending in Section II).

It is interesting to compare our technique to prove liveness for networks that model systolic arrays to that proposed earlier by Gouda and Chang [3], [4] to prove liveness for general networks of communicating finite state machines. Each of the two techniques is at the same time more and less powerful than the other.

1) The technique in this paper is more powerful since it is applicable to networks where the number of machines is a parameter rather than a fixed value. By contrast, the technique of Gouda and Chang applies only to networks with fixed number of machines.

2) The technique of Gouda and Chang is more powerful since it is based on a model of communicating finite state machines that is more general than the one discussed in this paper. In particular, the model in this paper is *deterministic*, i.e., since each sending node has exactly one outgoing edge, the history of each output channel of a machine is determined uniquely from 1) the machine's definition, and 2) the history of each input channel of the machine. By contrast, the general model is nondeterministic, and so its liveness properties cannot in general, be established using the algorithm in our paper.

There are two side benefits to the discussion in this paper.

1) The characterization of communicating finite state

machines that model systolic arrays in Section II represents an important step towards the formal characterization of systolic arrays in general. (So far, these arrays have been characterized only informally [8].) In particular, we notice that properties such as the deterministic and cyclic behavior of each machine, and its fair and uniform access to its channels are important characteristics that distinguish systolic arrays from other distributed systems.

2) Channel histories emerge from this paper as a useful specification technique that complements the definition of systolic arrays. For example, once a set F of histories is proved, by Algorithm 1, to be consistent with a network W, then the histories of external channels in F complement the definition of W by specifying the required frequency of pumping data messages into the array.

Finally, we observe that our technique to prove liveness and termination of systolic arrays is based on similar concepts to those used by Ossefort [14] to prove safety properties of the same arrays. In particular, both techniques are based on the concept of "channel histories" or "communication traces." An interesting problem is to investigate whether these two techniques can be merged into one technique to prove all required properties of systolic arrays.

Acknowledgment

We are thankful to M. Ossefort, M. Chandy, and J. Misra for directing our attention to the verification problem of systolic arrays, and to C. K. Chang for many stimulating discussions during the course of this work, and to C. Lengauer and the referees for many comments on an earlier draft of this paper. After the first author discussed this paper with S. Owicki, she pointed out to him how to extend the verification methodology to cover safety properties as well [16].

References

[1] G. V. Bochmann, "Finite state description of communication protocols," *Comput. Networks*, pp. 361–371, vol. 2, 1978.

[2] M. G. Gouda, "Closed covers: To verify progress for communicating finite state machines," *IEEE Trans. Software Eng.*, vol. SE-10, pp. 846–855, Nov. 1984.

[3] M. G. Gouda and C. K. Chang, "A technique for proving liveness of communicating finite state machines with examples," in *Proc. 3rd ACM Symp. Principles of Distributed Comput.*, Aug. 1984.

[4] ——, "Proving liveness for networks of communicating finite state machines," Dept. of Comput. Sci., Univ. Texas at Austin, *Tech. Rep. TR-84-4*, Jan. 1984; submitted for journal publication.

[5] M. G. Gouda and Y. T. Yu, "Synthesis of communicating finite state machines with guaranteed progress," *IEEE Trans. Commun.*, vol. COM-32, pp. 779–788, July 1984.

[6] L. H. Guibas, H. T. Kung, and C. D. Thompson, "Direct VLSI implementation of combinatorial algorithms," in *Proc. Cal. Tech. Conf. VLSI*, Jan. 1979, pp. 509–525.

[7] H. T. Kung, "Let's design algorithms for VLSI systems," in *Proc. Cal. Tech. Conf. VLSI*, Jan. 1979, pp. 65–90.

[8] H. T. Kung and C. E. Leiserson, "Algorithms for VLSI processor arrays," in *Proc. Symp. Sparse Matrix Computations*, Knoxville, TN, Nov. 1978.

[9] C. E. Leiserson, "Systolic priority queues," in *Proc. Cal. Tech. Conf. VLSI*, Jan. 1979, pp. 199–214.

[10] Z. Manna and A. Pnueli, "Adequate proof principles for invariance and liveness properties of concurrent programs," *Sci. Comput. Program.*, vol. 4, no. 3, 1984, pp. 257–289.

[11] J. Misra and K. M. Chandy, "Proof of networks of processes," *IEEE Trans Software Eng.*, vol. SE-7, July 1981.

[12] J. Misra, K. M. Chandy, and T. Smith, "Proving safety and liveness of communicating processes with examples," in *Proc. 1st ACM Symp. Principles of Distributed Comput.*, Aug. 1982, pp. 18–20.

[13] M. J. Ossefort, "A unified approach to formal verification of network safety properties," Ph.D. dissertation, Dept. Comput. Sci., Univ. Texas at Austin, Aug. 1982.

[14] ——, "Correctness proofs of communicating processes: Three illustrative examples from the literature," *ACM Trans. Program. Lang. Syst.*, vol. 5, no. 4, pp. 620–640, Oct. 1983.

[15] S. Owicki and L. Lamport, "Proving liveness properties of concurrent programs," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, pp. 455–495, 1982.

[16] S. Owicki, private communication, Nov. 1984.

[17] P. Zafiropulo, C. H. West, H. Rudin, D. Brand, and D. Cowan, "Towards analyzing and synthesizing protocols," *IEEE Trans. Commun.*, vol. COM-28, pp. 651–661, Apr. 1980.

Mohamed G. Gouda (S'76–M'77) received the B.Sc. degrees in engineering and mathematics from Cairo University, Cairo, Egypt, in 1968 and 1971, respectively, the M.A. degree in mathematics from York University, Toronto, Ont., Canada, in 1972, and the M.Math and Ph.D. degrees in computer science from the University of Waterloo, Waterloo, Ont., in 1973 and 1977, respectively.

From 1977 to 1980 he worked for the Honeywell Systems and Research Center and the Honeywell Corporate Technology Center, Minneapolis, MN. Since 1980 he has been an Assistant Professor in the Department of Computer Sciences at the University of Texas at Austin. His research interests include formal verification and synthesis of distributed systems and communication protocols.

Hui-Seng Lee was born in Kelang, Malaysia, on March 21, 1960. After completing his work at Kelang La Salle, Malaysia, in 1979, he entered Del Mar Junior College, Corpus Christi, TX. After transferring to the University of Texas at Austin, he received the B.A. and M.Sc. degrees in computer sciences in 1982 and 1984, respectively.