A Multiprocess Network Logic with Temporal and Spatial Modalities

JOHN REIF*

Aiken Computation Laboratory, Harvard University, Cambridge, Massachusetts 02138

AND

A. P. SISTLA

Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, Massachusetts 01003

A modal logic which can be used to formally reason about synchronous fixed connection multiprocess networks such as of VLSI is introduced. The logic has both *temporal* and *spatial* modal operators. The various temporal modal operators can be used to relate the properties of the current state of a given process with properties of succeeding states of the same process. The spatial modal operators are useful to relate the properties of the current state of a given process with properties of the current state of neighboring processes. Many interesting properties of multiprocessor networks can be elegantly expressed in our logic. Examples of the diverse applications of the logic to packet routing, firing squad problems, systolic algorithms, and distributed system are given. Also some results in the decidability and complexity issues of this logic are presented.

1. INTRODUCTION

One of the fundamental models of parallel computation is a collection of synchronous processors with fixed interconnections. The iterative linearly connected, mesh connected, and multidimensional arrays of [8] and [3], the shuffle exchange networks of [15], the ultracomputer of [12], and the cube connected cycle networks of [10] are some such examples.

Parallel algorithms for such networks are difficult to formally describe and prove correct. For example, the systolic algorithms of [16] are not formally proved correct. in that paper, instead informal "picture proofs" are presented.

An informal description of a program or algorithm for a fixed connection network

* This work was supported by the National Science Foundation Grants NSF MCS82-00269 and NSF MCS79-08365 and the Office of Naval Research Contract N00014-80-0674.

Preprint of paper appearing in ***Journal of Computer and System Sciences*****, Vol. 30, No. 1, February 1985, pp. 41-53.**

would likely make reference to the spatial relationships between neighboring processes and properties holding for all processes, as well as the transformations over time. Indeed, natural English allows expression of spatial modal operators such as "everywhere," "somewhere," "across such and such connection," as well as temporal modal operators such as "until," "eventually," "hereafter," and "next time." However, natural English cannot suffice for formal semantics. This paper proposes a formal logic allowing use of these modal operators in the context of a fixed connection network.

Previous program logics contained only temporal modal operators [9], [7] or modal operators for the effect of program statements [4]. Especially, temporal logic has been used to reason about parallel programs; however, it is impractical to use this logic to reason about large numbers of processes operating synchronously and communicating through fixed connections. Our use of spatial as well as temporal modal operators is a new idea. (Note: Our spatial modal operators differ in an essential way from the modal operators of dynamic logic; see Section 2.3.) This combination of temporal and spatial modal operators allows us to formally reason about computations on networks with complex connections. Indeed, one can view our logic as multidimensional temporal logic.

The contribution of this paper is more than simply the definition of logic. We also describe different applications and present certain decidability and complexity results.

Section 2 defines the logic. Section 3 describes applications of our logic to routing on shuffle exchange network, to the firing squad problem on a linear array, and to systolic computation on arrays. We also show how some interesting properties in distributed systems can be expressed in our logic. Section 4 investigates the problem of testing validity of formulae in our logic. We prove that the set of valid formulae is Π_1^1-complete. In practice we are many times interested in deciding validity over models on a given network. We show that this problem is PSPACE-complete. Section 5 gives the coclusions.

2. Definitions

2.1. *Syntax*

The symbols in our logic are atomic propositions drawn from a set $\mathscr{F}_0$, the propositional connectives $\neg$, $\wedge$, the temporal modalities *hereafter*, *eventually*, *nexttime*, *until*, spatial modalities *somewhere*, *everywhere* and symbols called *links* drawn from a set L.

The set of formulae $\mathscr{F}$ is the minimal set containing $\mathscr{F}_0$ and such that if $f_1, f_2 \in \mathscr{F}$ then the following strings enclosed in parentheses are also in $\mathscr{F}$: $f_1 \wedge f_2$, $\neg f_1$, *eventually* f_1, *hereafter* f_1, f_1 *until* f_2, *nexttime* f_1, *somewhere* f_1, *everywhere* f_1, lf_1 for each link $l \in L$. We also use the symbols $\vee$, $\supset$ so that $(f_1 \vee f_2)$ is an abbreviation for $\neg(\neg f_1 \wedge \neg f_2)$ and $(f_1 \supset f_2)$ is an abbreviation for $(\neg f_1 \vee f_2)$. We avoid parentheses whenever the implied parsing of the formula is understood from the context.

2.2. *Networks*

A network is a pair $G=(P,E)$, where P is a countable set of elements called processes and $E: L \times P \to P$ is a partial mapping. Intuitively, for each process $p \in P$ and link $l \in L$, $E(l,p)$ if defined, is the process connected to p by link l.

2.3. *Semantics*

A model $\mathscr{M}$ (with network G) is a 4-tuple (G, S, Ψ, Δ), where

(i) $G=(P,E)$ is a network,

(ii) S is a set of states,

(iii) $\Psi: S \to 2^{\mathscr{F}_0}$ associates with each state the set of atomic propositions true in that state,

(iv) $\Delta: P \to S^{\omega}$ associates a ω-sequence of states with each process.

We require that for $p \neq q$ there is no state appearing both in $\Delta(p)$ and $\Delta(q)$. We denote the ith state in $\Delta(p)$ by $s_{p,i}$. We extend E to the domain L^* such that $E(\varepsilon, p)=p$ and $E(l_1 \cdot l_2, P)=E(l_1, E(l_2,p))$, where ε is the empty string. An interpretation is a triple $(\mathscr{M}, p, i)$, where $\mathscr{M}$ is a model, p is a process and i is a nonnegative integer. We define the $\models$ relation inductively. This relation denotes the truth of a formula in an interpretation.

$\mathscr{M}, p, i \models F$, where F is an atomic proposition iff $F \in \Psi(s_{p,i})$;

$\mathscr{M}, p, i \models f_1 \wedge f_2$ iff $\mathscr{M}, p, i \models f_1$ and $\mathscr{M}, p, i \models f_2$;

$\mathscr{M}, p, i \models \neg f_1$ iff $\mathscr{M}, p, i \not\models f_1$;

$\mathscr{M}, p, i \models \textit{nexttime}\ f_1$ iff $\mathscr{M}, p, i+1 \models f_1$;

$\mathscr{M}, p, i \models \textit{eventually}\ f_1$ iff $\exists k \geqslant i$ such that $\mathscr{M}, p, k \models f_1$;

$\mathscr{M}, p, i \models \textit{hereafter}\ f_1$ iff $\forall k \geqslant i$, $\mathscr{M}, p, k \models f_1$;

$\mathscr{M}, p, i \models f_1\ \textit{until}\ f_2$ iff $\exists k \geqslant i$ such that $\mathscr{M}, p, k \models f_2$ and $\forall j, i \leqslant j < k\ \mathscr{M}, p, j \models f_1$;

$\mathscr{M}, p, i \models l f_1$ iff $E(l,p)$ is defined and $\mathscr{M}, q, i \models f_1$, where $q=E(l,p)$;

$\mathscr{M}, p, i \models \textit{somewhere}\ f_1$ iff $\exists \alpha \in L^*$ such that $E(\alpha,p)$ is defined and $\mathscr{M}, q, i, \models f_1$, where $q=E(\alpha,p)$;

$\mathscr{M}, p, i \models \textit{everywhere}\ f_1$ iff $\forall \alpha \in L^*$ $(E(\alpha,p)$ is defined $\Rightarrow \mathscr{M}, q, i \models f_1$ where $q=E(\alpha,p))$.

Note the following identities:

$$\textit{hereafter}\ f_1 \equiv \neg\ \textit{eventually}\ (\neg f_1),$$
$$\textit{eventually}\ f_1 \equiv \text{True}\ \textit{until}\ f_1,$$
$$\textit{everywhere}\ f_1 \equiv \neg\ \textit{somewhere}\ (\neg f_1).$$

We can also define a model in a different way which is more in the style of PDL structures. We give a brief description of this definition below.

A model $\mathscr{M}$ is a triple (S, Ψ, Δ), where

(i) S is a set of states,

(ii) $\Psi: S \rightarrow 2^{\mathscr{F}_0}$,

(iii) Δ: $(L \cup \{\text{nexttime}\}) \rightarrow (S \rightarrow S)$ is a mapping such that $\Delta(\text{nexttime})$ is a total function, $\Delta(l)$ is a partial function and $\Delta(\text{nexttime}) \cdot \Delta(l) = \Delta(l) \cdot \Delta(\text{nexttime})$ for each $l \in L$.

We extend Δ as a partial mapping to the domain $(L \cup \{\text{nexttime}\})^*$ so that $\Delta(l_1 \cdot l_2) = \Delta(l_1) \cdot \Delta(l_2)$. In addition to the above conditions we require a model to satisfy the following condition:

(iv) For $s_1 \neq s_2$ if Δ (nexttime) $(s_1) = \Delta$ (nexttime)(s_2) then for some $i \geqslant 1$, $\Delta(\text{nexttime}^i)(s_1) = s_2$ or Δ $(\text{nextime}^i)(s_2) = s_1$.

We define two states s, s' to be nexttime equivalent if for some $i \geqslant 0$, $\Delta(\text{nexttime}^i)(s') = s$ or $\Delta(\text{nexttime}^i)(s) = s'$. It is easily seen that the relation "nexttime equivalent" is an equivalence relation. We let $\Pi(s)$ denote the nexttime equivalence class of s. Intuitively, $\Pi(s)$ denotes the set of states of a process, and the sequence of states given by Δ $(\text{nexttime}^i)(s)$ for $i \geqslant 0$, represents the computation of the process starting in state s. Whenever there is no confusion we let $\Pi(s)$ also denote the corresponding process. If $\Delta(l)(s) = s'$ then we say that process $\Pi(s)$ is connected to $\Pi(s')$ by the link l. We also place the following additional requirement on a model.

(v) Every $\Pi(s)$ has a starting point, that is for each $\Pi(s)$ there exists a $s' \in \Pi(s)$ such that for all $t \in \Pi(s)$ $t = \Delta$ $(\text{nexttime}^i)(s')$ for some $i \geqslant 0$.

Using these definitions it is easily seen how we can go from these type of models to the previously defined models and vice versa. Hereafter we only consider our first definition of a model.

We say that a formula f is *satisfiable* if there exists a model $\mathscr{M}$ such that $\mathscr{M}, p, i \models f$ for an $i \geqslant 0$ and a process p in the network of $\mathscr{M}$. A formula f is said to be *valid* if f is true in all interpretations.

2.4. *Extensions to a First Order Logic*

The first order version of this logic consists of the additional symbols like local variables, global variables, constant symbols, function and relation symbols, and the universal quantifier $\forall$. A term is defined as in the case of first order predicate calculus. An atomic formula is an atomic proposition or of the form $Rt_1 t_2 \cdots t_k$, where R is k-ary relation symbol (R can be equality in which case $k = 2$) and t_1, $t_2 \cdots t_k$ are terms. The additional requirement for the set of formulae is that if f is a formula and x is a global variable so is $\forall x(f)$. A model $\mathscr{M}$ is a 4-tuple (Σ, G, S, Δ), where $\Sigma = (D, \alpha, \beta)$ in which D is a countable domain in which the variables take values, α interprets the relation and function symbols, β is a mapping associating with each global variable and constant symbol a value from the domain; S is the set of states where each state is a mapping that associates a truth value to each atomic

proposition and a value from D with each local variable; G, Δ are the same as in the propositional case.

An interpretation is a triple $(\mathcal{M}, p, i)$, where $\mathcal{M}$ is a model, p is a process in the network of G and $i \geqslant 0$. Truth of a formula in an interpretation is defined analogous to the corresponding definitions for propositional version of the logic with appropriate modifications, with the following additional definition: $(\mathcal{M}, p, i) \models \forall x f$ iff for each $c \in D(\mathcal{M}^c, p, i) \models f$ where $\mathcal{M}^c$ is exactly same as $\mathcal{M}$ except that the global variable x is given the value c in $\mathcal{M}^c$. Satisfiability and validity of formulae are defined as usual.

3. Applications

This section gives some examples of the use of our logic to various multiprocess network applications.

3.1. *Routing on a Shuffle–Exchange Network*

A shuffle–exchange network G is a pair (P, E), where $P = \{0, 1\}^n$ and

$$E\colon \{exchange,\ shuffle\} \times P \to P$$

is defined as follows:

$$E(exchange, (a_{n-1}, a_{n-2}, \ldots, a_0)) = (a_{n-1}, a_{n-2}, \ldots, \bar{a}_0),$$
$$E(shuffle, (a_{n-1}, a_{n-2}, \ldots, a_0)) = (a_0, a_{n-1}, \ldots, a_1),$$

for all $a_{n-1}, a_{n-2}, \ldots, a_0 \in \{0, 1\}$. Intuitively, the exchange edge connects processes p_1 and p_2 if all the bits of p_1 and p_2 are the same excepting the least significant bits which are distinct. The shuffle edge connects two processes p_1 and p_2, if p_2 is obtained by one cyclic shift of bits in p_1.

The routing problem in this network is to route a packet present at some process to a given destination traversing only along the shuffle and exchange edges.

We capture the name of a process by the atomic propositions $A_{n-1}, A_{n-2}, \ldots, A_0$. The formula f_0 asserts that the name of a process is invariant over time;

$$f_0 = \bigwedge_{0 \leqslant i < n} (\textit{hereafter } A_i \vee \textit{hereafter } \neg A_i),$$

f_1, f_2 assert that exchange and shuffle edges are properly connected.

$$f_1 = \bigwedge_{1 \leqslant i < n} (A_i \leftrightarrow \textit{exchange } A_i) \wedge A_0 \leftrightarrow \textit{exchange } \neg A_0,$$

$$f_2 = \bigwedge_{0 \leqslant i < n} (A_i \leftrightarrow \text{shuffle}\, A_{(i-1) \bmod n}).$$

The presence of the packet at any process will be indicated by the atomic proposition X, and the destination by the atomic propositions $D_{n-1}, D_{n-2}, \ldots, D_0$. We

assume that the name of the destination travels with the message. Let g_0 assert that X is true in at most one place. It is not difficult to see that this can be expressed easily.

$$g_1 = X \supset \left[\bigwedge_{0 \leqslant i < n} (D_i \supset \textit{hereafter everywhere } (X \supset D_i)) \right.$$
$$\left. \wedge\, (\neg D_i \supset \textit{hereafter everywhere } (X \supset \neg D_i)) \right]$$

asserts that the name of the destination process travels with the packet.

$$g_2 = X \supset \textit{nexttime } (X \vee (\textit{shuffle } X) \vee \textit{exchange } X)$$

asserts that the packet travels along shuffle or exchange edges only.

The main correctness property is g_3 which asserts that the packet reaches its destination eventually

$$g_3 = \textit{eventually somewhere } \left(X \wedge \bigwedge_{0 \leqslant i \leqslant n-1} (A_i \leftrightarrow D_i) \right).$$

Let r be a formula which describes the actual routing algorithm. Then (*hereafter everywhere* $(r \wedge f_0 \wedge f_1 \wedge f_2 \wedge g_0 \wedge g_1 \wedge g_2)) \supset g_3$ is a valid formula iff the algorithm correctly routes packets.

Next we describe a specific routing algorithm for the shuffle exchange network and derive the corresponding formula r for its semantics. The packet will be routed in n stages, where for $i = 0, \ldots, n-1$, if at the start of the ith stage the packet is located at a process whose lowest order address bit is not the value of D_i, then the packet traverses an *exchange* link. In either case, the packet next traverses a *shuffle* link and reaches the $i+1$ stage.

To define a formula r for this routing algorithm, it is useful to introduce propositional variables $S_0, \ldots, S_{n-1}$ and require that only unique S_i be true at any process and that S be invariant on traversing an *exchange* link but that $S_{(i+1) \bmod n}$ be true on traversing a *shuffle* link. Thus we let

$$r_0 = \bigvee_{0 \leqslant i < n} \left(S_i \wedge \left(\bigwedge_{\substack{0 \leqslant j < n \\ i \neq j}} \neg S_j \right) \wedge (\textit{nexttime exchange } S_i) \right.$$
$$\left. \wedge\, (\textit{nexttime shuffle } S_{(i+1) \bmod n}) \right).$$

The formula for semantics of this routing algorithm is therefore

$$r_1 = r_0 \wedge \left(\left[X \wedge \bigvee_{0 \leqslant i < n} A_i \leftrightarrow \neg D_i \right] \right.$$
$$\supset \bigwedge_{0 \leqslant i < n} \left[S_i \wedge ((A_0 \leftrightarrow D_i) \supset \textit{nexttime shuffle } (X)) \right.$$
$$\left. \left. \wedge\, ((A_0 \leftrightarrow \neg D_i) \supset \textit{nexttime exchange } (X)) \right] \right).$$

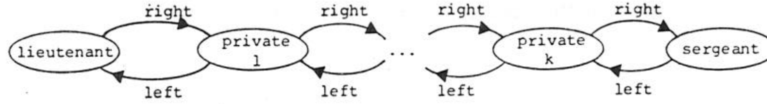

FIGURE 1

3.2. *The Firing Squad Problem for a Linear Array*

We briefly describe the problem and show how its correctness can be specified by our logic. A solution to the firing squad problem consists of a linear array of deterministic finite state processes as shown in Fig. 1. The next move of each process is a function of its present state and the states of its neighbors. All the privates are identical processes. The problem is to obtain the program for the lieutenant, the sergeant, and the privates so that whenever the lieutenant is in a designated initial state, then eventually all the processes simultaneously enter a special state called the firing state, and none of them enters this state before this time. The solution should work for linear arrays of *all sizes*.

We assume that all processes have the state set $Q = \{0, 1, 2, \ldots, m\}$, and the state 0 is the initial state of each process. State 1 is the specific state into which the lieutenant enters to start the operation, state m is the firing state. All the privates are identical. We use atomic propositions $P_0, P_1, \ldots, P_m$ to indicate the state of a process (P_i is true at a place iff the corresponding process is in state i at that instance). Now we assert the operation of the system as follows.

(i) I asserts that each process is in at most one state at any instant of time,

$$I = \textit{everywhere hereafter} \left[\bigwedge_{\substack{0 \leqslant i,j \leqslant k \\ i \neq j}} (P_i \supset \neg P_j) \right].$$

(ii) f_0 asserts that the moves of the lieutenant is according to its next move partial function $\delta_0: Q^2 \to Q$, and in the beginning the lieutenant is in state 0 or 1.

$$f_0 = \textit{everywhere} \left[\neg\, \textit{left}(\text{true}) \supset \left((P_0 \vee P_1) \right. \right.$$

$$\left. \left. \wedge\ \textit{hereafter} \bigwedge_{i,j} ((P_i \wedge \textit{right}\ P_j) \supset \textit{nexttime}\ P_{\delta_0(i,j)}) \right) \right].$$

Note that $\neg$ *left* (true) is true only on the lieutenant, the leftmost processor.

(iii) Similarly, let f_1, f_2 be the formulae that define the moves of all privates and the sergent, respectively. The positions of privates is identified by the truth of the formula

$$(\textit{left}(\text{true}) \wedge \textit{right}(\text{true})).$$

Note that the position of the sergeant is identified by the formula

$$\neg\text{right}(\text{True}).$$

(iv) Let g_0 be the formula that asserts that if any process (other than the lieutenant) and all its neighbors are in state 0 then it remains in state 0 in the next step. It is easily seen that this can also be asserted.

Now we assert that if all the above conditions are met and at any time the lieutenant enters the state 1 then all processes will eventually enter the firing state simultaneously at some future instance, and none of them will be in the firing state before that instance. This is captured by the formula:

$$g = (I \wedge f_0 \wedge f_1 \wedge f_2 \wedge g_0) \supset \textit{hereafter}\ [\textit{somewhere}(\neg \textit{left}(\text{true}) \wedge P_1) \supset ((\neg \textit{somewhere}\ P_m)\ \textit{until}\ (\textit{everywhere}\ P_m))];$$

g is valid on all models with linear arrays as networks iff the given solution to the firing squad problem is correct. A similar construction can be given for the firing squad problem over any given network.

3.3. *Systolic Arithmetic Computations*

The systolic algorithms of [16] are not formally proved correct in that paper; instead informal "picture proofs" are presented. Our logic is thus particularly useful here when extended to first order formulae.

We consider an interesting example of a network for matrix-vector multiplication due to [16]. The matrix is an infinite band matrix of bandwidth $(n+1)$. The network architecture is shown in Fig. 2.

The main processors are $P_0, P_1, \ldots, P_n$. The processors $P'_0, P'_1, \ldots, P'_n$ are the input processors, each of them contains a variable Z. The values of Z in P'_i change with time and they represent the values of the ith diagonal of the matrix. Each processor P_i has two variables X, Y. The values of the variables X in P_0 over time represent the input vector. The values of X move right with each time instance.

Thus

$$q_1 = \textit{left}(\text{true}) \supset \forall \alpha(\textit{left}(X = \alpha) \leftrightarrow \textit{nexttime}(X = \alpha))$$

asserts that the value of X at the nexttime instance in a process P_i $(i > 0)$, is the present value of X in the process left to P_i.

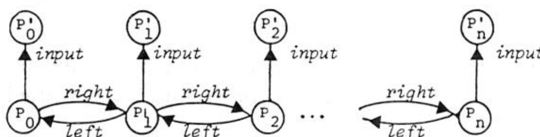

FIGURE 2

At each step P_i $(i < n)$ computes its value of Y to be the sum of the previous value of Y in process P_{i+1}, plus the product of X in P_i times Z in P_i'. This is captured by

$$q_2 = \mathit{right}(\text{true}) \supset \forall\alpha\forall\beta(\mathit{right}(Y=\alpha) \wedge \mathit{nexttime\ input}(Z=\beta) \supset \mathit{nexttime}(Y = \alpha + X \cdot \beta)).$$

At each step P_n computes its value of Y to be the product of the value of X in P_n and the value of Z in P_n'. This can also be easily asserted by formula

$$g_3 = \mathit{right}(\text{false}) \wedge \mathit{input}(\text{true}) \supset \forall\alpha\forall\beta(X=\alpha \wedge \mathit{input}(Z=\beta)) \supset \mathit{nexttime}(Y=\alpha\cdot\beta))$$

(note that $\mathit{right}(\text{false}) \wedge \mathit{input}(\text{true})$ holds only for process P_n).

The steady state correctness property at P_n can thus be expressed as *hereafter everywhere* $(g_1 \wedge g_2 \wedge g_3) \supset$ *hereafter* h, where

$$h = \mathit{left}(\text{false}) \wedge \mathit{input}(\text{true}) \supset \forall\alpha_0 \cdots \alpha_n \forall\beta_0 \cdots \beta_n \left[\left(\bigwedge_{i=0}^{n} \text{nexttime}^{2i}(X=\alpha_i) \wedge \mathit{right}^{n-i}\mathit{input} \times \text{nextime}^{n+i}(Z=\beta_i)\right) \supset \text{nexttime}^{2n}\, Y = \sum_{i=0}^{n} \alpha_i \cdot \beta_i\right].$$

3.4. *Expressing Some Properties in Distributed Systems*

Many distributed systems have the provision for broadcasting messages, i.e., a process can send a message to all processors to which it is connected. This property is expressed by the following formula:

hereafter everywhere (message-sent $\supset$ *everywhere eventually* message-received),

where message-sent and message-received are atomic propositions indicating that a message is sent or received in a state of a process. Indeed, using first order version of the logic we can assert that the message received in each process is exactly the message sent by the sender process.

4. Decidability and Complexity Issues

In this section we consider issues of decidability and complexity of different versions of our logic.

THEOREM 1. *The set of satisfiable formulae of multiprocess network logic is Σ_1^1-complete and the set of valid formulae is Π_1^1-complete.*

Proof. First we show that the set of satisfiable formulae is Σ_1^1-complete. From this it can easily be seen that the set of valid formulae is Π_1^1-complete.

We consider a deterministic Turing maching M on infinite strings. M has one read only infinite input tape and an infinite work tape. The input head of M always moves right by one cell after each step. M is said to accept an input if during its computation it goes into any of a set of final states infinitely often. The set of encodings of all Turing machines that accept at least one input is known to be Σ_1^1-complete. This fact is also used in [13]. We give a many-to-one reduction from this set to the set of satisfiable formulae.

A partial *id* of M consists of a sequence of composite symbols denoting the contents of the work tape, the head position on the work tape and the state of the finite state control. In our formula we use only one link symbol *right*. We assume that each process denotes one cell in the partial *id* of M, and we also assume we have one atomic propositions corresponding to each composite symbol. Thus if an atomic proposition is true in a state of a process then it denotes that the corresponding cell of the *id* has the corresponding composite symbol. Now we can assert that at any instance the sequence of states connected by the link *right* denotes a valid partial *id* of M. We can also assert that the *id* at any instance is obtained from the previous *id* by one move of M for some input symbol value. We can also assert that the beginning *id* is the initial *id*. Finally, we express the property that final *ids* appear infinitely often as follows: Let F be a propositional formula over the atomic propositions asserting that the state in the composite symbol is a final state. The formula *hereafter eventually* (*somewhere* F) asserts that F holds at some place in infinitely many *ids*. Let f be the conjunction of the previous formulae. Then it is easily seen that f is satisfiable iff M accepts at least one input.

We can show that the set of satisfiable formulae is in Σ_1^1 as follows: We use integers to denote the processes. Now the following predicate form asserts that a formula f is satisfiable $\exists g\tilde{R}(g,f)$, where g is a function on integers and defines the encoding of a model $\mathscr{M}=(G, S, \Psi, \Delta)$ (this can be done since G, Ψ, Δ can be encoded by functions from integers to integers), $\tilde{R}(g, f)$ is a recursive predicate prefixed with some first order quantifiers. Essentially $\tilde{R}(g,f)$ asserts that the formula f of our logic is satisfiable in an interpretation of the model encoded by g, of course it also asserts that g is a proper encoding of a model. It is straightforward to see that such a $\tilde{R}(g,f)$ exists. Hence the set of satisfiable formulae is Σ_1^1-complete. ∎

Let $\mathscr{M}=(G, S, \Psi, \Delta)$ be a model with a finite network $G=(P, E)$. Let f be a formula in our logic. For each $i \geqslant 0$, we define a function ϕ_i: $P \rightarrow 2^{\mathrm{SF}(f)}$, where $\mathrm{SF}(f)$ is the set of subformulae of f, such that $\phi_i(p)=\{g \in \mathrm{SF}(f) \mid (\mathscr{M}, p, i) \models g\}$. $\mathscr{M}$ is said to be ultimately periodic if there exist m, l such that $\forall i \geqslant l$, $\forall p \in P$, $\Psi(s_{p,i})=\Psi(s_{p,i+m})$.

LEMMA 1. *A formula f is satisfiable in a model with a finite network G iff f is satisfiable over an ultimately periodic model with network G.*

Proof. The proof of the lemma is exactly similar to the ultimately periodic model theorem for PTL given in [14] and is omitted here. ∎

THEOREM 2. *The set of formulae that are satisfiable in a model with a finite network is Σ_1^0-complete, and the set of valid formulae in models with finite networks is Π_1^0-complete.*

Proof. Using similar methods as in the proof of Theorem 1, we can easily give a reduction from the set of encodings of all Turing machines over *finite strings* that accept at least one input to the set of formulae satisfiable in a model with a finite network. Next, we give a Turing machine M which accepts the set of formulae satisfiable in a model with a finite network. M guesses a finite network and an ultimately periodic model with this network. It next verifies that f is satisfiable in this model. M halts only on the input formulae that are satisfiable in a model with a finite network. It is easy to see how one can obtain M. From this it follows that the set of formulae satisfiable in a model over a finite network is Σ_1^0-complete. It also follows that the set of valid formulae in all models over finite networks in Π_1^0-complete. ∎

THEOREM 3. *The set of all pairs of the form (G, f), where G is a finite network and f is a formula that is satisfiable in a model with network G, is* PSPACE-*complete.*

Proof. The PSPACE-hardness of the problem follows from the PSPACE-hardness of satisfiability for PTL given in [14]. We give a polynomial space bounded nondeterministic Turing machine M which when given a pair (G, f) checks if f is satisfiable in a model with network G. From Lemma 1, it is enough if M checks for the existence of an ultimately periodic model. M works as follows: M first guesses ϕ_0. It verifies for each $p \in P$, $\phi_0(p)$ is propositionally consistent. It also verifies that for each subformula h of f, for each $p \in P$,

(i) $h = l(g) \in \phi_0(p)$ iff $g \in \phi_0(p')$, where $p' = E(l, p)$,

(ii) $h = (\textit{somewhere } g) \in \phi_0(p)$ iff there is a process p' reachable from p using the links and $g \in \phi_0(p')$,

(iii) $h = (\textit{everywhere } g) \in \phi_0(p)$ iff for every process p' reachable from p in G, $g \in \phi_0(p')$.

Now M continues to guess $\phi_1, \phi_2, \ldots$ Each time, it keeps only ϕ_i, ϕ_{i+1} and checks that each ϕ_i also satisfies the previous rules. In addition it also verifies that ϕ_i, ϕ_{i+1} are temporally consistent, i.e., for a formula $h \in SF(f)$,

(i) $h = (\textit{nexttime } g) \in \phi_i(p)$ iff $g \in \phi_{i+1}(p)$,

(ii) if $h = (\textit{hereafter } g) \in \phi_i(p)$, then $g \in \phi_i(p)$ and $h \in \phi_{i+1}(p)$,

(iii) if $h = (g_1 \textit{ until } g_2) \in \phi_i(p)$ then $g_2 \in \phi_i(p)$ or ($g_1 \in \phi_i(p)$ and $h \in \phi_{i+1}(p)$),

(iv) if $h = (\textit{eventually } g) \in \phi_i(p)$, then $g \in \phi_i(p)$ or $h \in \phi_{i+1}(p)$.

At some value of i say l, M guesses that the periodic part starts and saves ϕ_l. It continues guessing ϕ_i, and at some point it guesses that the periodic part ends and takes $\phi_l = \phi_{i+1}$. M makes sure that certain formulae in $\phi_l(p)$ for each $p \in P$, are fullfilled, i.e., if (g_1 *until* g_2) $\in \phi_l(p)$ or (*eventually* g_2) $\in \phi_l(p)$, then it makes sure that $g_2 \in \phi_i(p)$ for some value of i in the period. By induction on f, it can easily be shown that M accepts a pair (G, f) iff f is satisfiable in a model with network G. Clearly M is polynomial space bounded. Now using Savitch's [11] theorem it is seen that there is a deterministic polynomial space bounded Turing maching that simulates M. ∎

5. Conclusions

We have proposed a logic to reason about computations of multiprocessor networks. We feel that our logic will be useful to specify the semantics and prove correctness of multiprocess networks. No such formal system for multiprocessor networks had been proposed previously. We have examined the application of our logic to some diverse multiprocess network problems, and presented some results in decidability and complexity of our logic.

It will be interesting to apply this formal system in verifying some of the examples given in the paper. It will also be interesting to examine the decidability of restricted versions of our logic.

Acknowledgments

The authors are very grateful for the anonymous referee for his valuable comments and suggestions.

References

1. M. Ben-Ari, Z. Manna, and A. Pnueli, The temporal logic of branching time, *in* "Eighth ACM Symposium on Principles of Programming Languages," Williamsburg, Va.,1981.
2. E. M. Clarke and A. Emerson, "Design and Synthesis of Programming Skeletons Using Branching Time Temporal Logic, IBM Conference of Logics of Programs, May 1981.
3. S. N. Cole, "Real time computations by n-dimensional iterative arrays of finite state machines," *IEEE Trans. Comput.* **18** (1969), 349–365.
4. M. Fischer and R. Ladner, Propositional dynamic logic of regular programs, *J. Comput. System Sci.* **18** (1979).
5. D. Gabbay, A. Pnueli, S. Shealah, and J. Stavi, Temporal analysis of fairness, *in* "Seventh ACM Symposium on Principles of Programming Languages," Las Vegas, NV.
6. J. Y. Halpern and J. H. Reif, The propositional dynamic logic of deterministic, well-structured programs, *in* Twenty-Second Symposium on Foundations of Computer Science," Nashville, Tenn. 1981.
7. Z. Manna and A. Pnueli, Verification of concurrent programs, *The Correctness Problem in Computer Science*," Academic Press, London/New York, 1981.

8. S. R. KOSARAJU, "Computations on Iterative Automata," Ph. D. Thesis, University of Pennsylvania, Pa., 1969.
9. A. PNUELI, The temporal logic of programs, *in* "Proceedings of 18th Symposium on Foundations of Computer Science," Providence, RI, November 1977.
10. F. P. PREPARATA AND J. VUILLEMIN, "The Cube Connected Cycles: A Versatile Network for Parallel Computation," pp. 140–147, FOCS 1979.
11. W. J. SAVITCH, Relationships between nondeterministic and deterministic tape complexities, *J. Comput. System Sci.* **4** 177–192.
12. J. T. SCHWARTZ, "Ultracomputers," *ACM Trans. Program. Languages Systems* **12** (4) (1980) 484–521.
13. A. P. SISTLA, E. M. CLARKE, N. FRANCEZ, AND Y. GUREVICH, Are message bufers characterizable in linear temporal logic? *in* "Proceedings of the Symposium on Principles of Distributed Computing," Ottawa, Canada, August 1982.
14. A. P. SISTLA AND E. M. CLARKE, The complexity of propositional linear temporal logics, *in* "ACM Symposium on Theory of Computing," pp. 159–167, 1982.
15. H. S. STONE, Parallel processing with the perfect shuffle, *IEEE Trans. Comput* **C-20** (2) (1971), 153–161.
16. H. T. KUNG AND C. E. LEISERSON, "Symposium on Sparse Matrix Computations and Their Applications," Knoxville, Tennessee, November 1978.

Printed by the St. Catherine Press Ltd., Tempelhof 41, Bruges, Belgium