"SOMETIME" IS SOMETIMES "NOT NEVER"

On the Temporal Logic of Programs

Leslie Lamport
Computer Science Laboratory
SRI International

1. INTRODUCTION

Pnueli [15] has recently introduced the idea of using temporal logic [18] as the logical basis for proving correctness properties of concurrent programs. This has permitted an elegant unifying formulation of previous proof methods. In this paper, we attempt to clarify the logical foundations of the application of temporal logic to concurrent programs. In doing so, we will also clarify the relation between concurrency and nondeterminism, and identify some problems for further research.

In this paper, we consider logics containing the temporal operators "henceforth" (or "always") and "eventually" (or "sometime"). We define the semantics of such a temporal logic in terms of an underlying model that abstracts the fundamental concepts common to almost all the models of computation which have been used. We are concerned mainly with the semantics of temporal logic, and will not discuss in any detail the actual rules for deducing theorems.

We will describe two different temporal logics for reasoning about a computational model. The same formulas appear in both logics, but they are interpreted differently. The two interpretations correspond to two different ways of viewing time: as a continually branching set of possibilities, or as a single linear sequence of actual events. The temporal concepts of "sometime" and "not never" ("not always not") are equivalent in the theory of linear time, but not in the theory of branching time -- hence, our title. We will argue that the logic of linear time is better for reasoning about concurrent programs, and the logic of branching time is better for reasoning about nondeterministic programs.

The work reported herein was funded by the National Science Foundation under Grant No. MCS-7816783.

The logic of linear time was used by Pnueli in [15], while the logic of branching time seems to be the one used by most computer scientists for reasoning about temporal concepts. We have found this to cause some confusion among our colleagues, so one of our goals has been to clarify the formal foundations of Pnueli's work.

The following section gives an intuitive discussion of temporal logic, and Section 3 formally defines the semantics of the two temporal logics. In Section 4, we prove that the two temporal logics are not equivalent, and discuss their differences. Section 5 discusses the problems of validity and completeness for the temporal logics. In Section 6, we show that there are some important properties of the computational model that cannot be expressed with the temporal operators "henceforth" and "eventually", and define more general operators.

2. AN INTRODUCTION TO TEMPORAL LOGIC

2.1. Assertions

The well-formed formulas of temporal logic are called assertions. The set of assertions is obtained in the obvious way from a set of atomic symbols -- called atomic predicates -- together with the usual logical operators $\wedge$, $\vee$, $\supset$ and $\sim$ (negation), and the unary temporal operators $\Box$ and $\rightarrow$. Thus, if P, Q, and R are atomic predicates, then $P \supset \Box(\sim Q \vee \rightarrow R)$ is an assertion. Assertions that do not contain either of the temporal operators $\Box$ or $\rightarrow$ are called predicates.[1] In this section, we give an intuitive description of how these temporal logic assertions are to be understood as statements about some system. Formal semantics are treated in the next section.

A predicate P represents a simple declarative statement about the state of the system; it is interpreted to mean "P is true now". An assertion represents a statement about the system which may refer to its state both now and in the future. The assertion $\Box A$ represents the statement that A is true now and will always be true in the future.

[1] We define the terms "predicate" and "assertion" to be consistent with their use in the field of program correctness, which differs from their use in logic.

The assertion $\rightarrow A$ represents the statement that A is true now or will become true sometime in the future. (After becoming true, it could then become false again.) We read $\Box$ as "henceforth" or "always", and $\rightarrow$ as "eventually" or "sometime".

The dual $\Diamond$ to the operator $\Box$ is defined by $\Diamond A \equiv \sim\Box\sim A$. Since $\Box\sim A$ states that A will never become true, we can read $\Diamond$ as "not never". If it is not the case that A is never true, can we conclude that A must eventually become true? In other words, is "sometime" the same as "not never"? The answer depends upon one's conception of the nature of time. In a nondeterministic system, the present does not determine a unique future, but rather a (perhaps infinite) set of possible futures. There are two radically different ways of viewing these possible futures: the theories of branching time and linear time.

In the branching time theory, all of the possible futures are equally real and must be considered. The assertion that a statement is "henceforth" (or "always") true means that it is true now and will remain true during all possible futures. A statement is "eventually" true if it is true now or else during every possible future it will be true at some time. A statement is "not always not" true if there is some possible future in which it becomes true. This does not mean that it becomes true during every possible future, as required for it to be "eventually" (or "sometime") true. Hence, "sometime" and "not never" are not equivalent in branching time.

In the theory of linear time, at each instant there is only one future that will actually occur. All assertions are interpreted as statements about that one real future. The assertion that a statement is "always" true means that it is true now and will remain true during the one real future. Similarly, it is "eventually" true if it is true now or will become true at some time during the real future. Since only the one real future is considered, any statement either is "never" true or else it is "eventually" true. Hence, "sometime" is equivalent to "not never" in the linear time theory.

2.2. Expressing Properties of Programs in Temporal Logic

We now indicate how temporal logic is used to express properties of a program -- particularly properties of a concurrent program. We have found that there are two fundamental types of properties one wants to prove of a program: safety properties, which assert that "something bad never happens"; and liveness properties, which assert that "something good must eventually happen". For a sequential program, partial correctness is an example of a safety property -- it asserts that the program never halts with the wrong answer; termination is a liveness property -- it asserts that the program must eventually halt.

To state a safety property, we need a predicate GOOD which represents the statement that the system is in an acceptable state. The property that "something bad never happens" is expressed by the assertion $\Box$ GOOD . We do not assume any special starting state, so the initial conditions must be explicitly specified. Letting INIT be the predicate which means that the system is in a proper initial state, our desired safety property is expressed by the following assertion:

$$\text{INIT} \supset \Box \text{ GOOD} . \qquad (2\text{-}1)$$

Safety properties can be expressed using only the concept of "always", but one needs the additional concept of "eventually" in order to express liveness properties. Manna and Waldinger [12] introduced the temporal operator $\rightarrow$, which they called "sometime", for deterministic sequential programs. In [9], we introduced the operator $\rightsquigarrow$ -- read "leads to" -- for concurrent programs. It can be defined in terms of $\rightarrow$ by

$$A \rightsquigarrow B \equiv A \supset \rightarrow B . \qquad (2\text{-}2)$$

A liveness property is of the form "something which should happen eventually does happen". To express it formally, we need a predicate REQUESTED which expresses the statement that the system is in a state in which "something should happen", and a predicate DONE which expresses the statement that the required event has happened. The liveness property is then expressed by the assertion REQUESTED $\rightsquigarrow$ DONE , where $\rightsquigarrow$ is defined by (2-2). As before, we want this property always to hold if the system is started in a proper initial state. The desired liveness property is then formally expressed by the assertion

$$\text{INIT} \supset \Box [\text{ REQUESTED} \rightsquigarrow \text{DONE }] , \qquad (2\text{-}3)$$

where INIT is as above.

Because we are not assuming any preferred initial states, properties about the system must be stated in the form INIT $\supset$ ... . It might seem more convenient to specify the starting states as part of the system instead of always writing the initial predicate. However, in proving correctness properties, one must often reason about the behavior of the program when started in other states. We have found it easier to write an explicit initial predicate in our assertions than to introduce preferred initial states and have to keep track of what initial state is being assumed when.

Safety and liveness properties seem adequate to describe the desired behavior of most complete programs. However, one sometimes needs to express more complicated concepts when proving these properties. One such concept is that of something happening "infinitely often". The assertion $\Box \rightarrow P$ represents the statement that property P is true infinitely often. Examples of statements requiring even more complex combinations of temporal operators can be found in the work of Francez and Pnueli [5].

3. THE SEMANTICS

3.1. Models

The assertions of temporal logic are defined as combinations of atomic predicates, logical connectives, and the temporal operators $\Box$ and $\rightarrow$. To define the semantics of temporal logic, we must formally define how these assertions are to be interpreted as statements about an underlying model. The type of model we use is quite general, and includes almost all formal models of program

execution that we know of. (The one exception is the model implicitly used in [11].) However, our class of models is restrictive enough so we can avoid many of the philosophical difficulties discussed in [18], which plague more general theories of temporal logic.

We define a state to be a truth-valued function on the set of atomic predicates. To see why such a function represents what we ordinarily think of as a state, recall that an atomic predicate is an uninterpreted symbol -- for example, the string of characters "a > 0". For a program having a variable a, a state x can be interpreted as one in which a has the value 1 if x("a > 0") = true, x("a > 1") = false, x("a > 2") = false, etc. Thus, the state of the program is defined by the truth or falsity of each atomic predicate.

Since any predicate is a logical combination of atomic predicates, there is a natural way to define a state x to be a truth-valued function on predicates. For example, for any atomic predicates P and Q, we define $x(\neg P \wedge Q)$ to equal $\neg x(P) \wedge x(Q)$. The generalization to arbitrary logical combinations should be obvious.

A model M is a pair (S, Σ), where S is a set of states and Σ is a set of sequences of states satisfying property E below. The set of states can be thought of as the set of all conceivable states of a program; i.e., all possible combinations of values of variables and "program counter" values. A sequence $s_0, s_1, s_2, \ldots$ in Σ represents an execution that starts in state s_0, performs the first program step to reach state s_1, performs the next program step to reach state s_2, etc. The execution terminates if and only if the sequence is finite. The set Σ represents all possible executions of the program, starting in any possible state.

The one assumption we make about a model, expressed formally by property E below, is that the future behavior depends only upon the current state, and not upon how that state was reached. Before formally stating this property, we introduce some notation. For any element s of Σ, we write $s = s_0, s_1, s_2, \ldots$ where the s_i are elements of S. If s is a finite sequence, so $s = s_0, \ldots, s_n$ for some n, then we define s_m to equal s_n for all $m > n$. Intuitively, s_i represents the state of the program at "time" i. The finite sequence $s_0, \ldots, s_n$ represents an execution in which the program halted at time n in state s_n. At all later times, it is still in state s_n.

If s is a sequence of length greater than one, then we define s^+ to equal $s_1, s_2, \ldots$ -- the sequence obtained by deleting the first element of s. If s is of length one, then s^+ is defined to equal s. The equality

$$(s^+)_i = s_{i+1} \qquad (3\text{-}1)$$

holds for all sequences s and nonnegative integers i.

We can now state our condition which the set Σ must satisfy as follows:

E. If $s \in \Sigma$ then $s^+ \in \Sigma$.

This condition means that after the program reaches state s_1, its subsequent behavior is not affected by how that state was reached.

We now define some more notation for later use. For any element x in S, we let Σ_x denote the set of all sequences in Σ which begin with x; so

$$\Sigma_x = \{ s \in \Sigma : s_0 = x \} . \qquad (3\text{-}2)$$

If s is the sequence $s_0, s_1, \ldots$; then we define s^{+n} to be the sequence $s_n, s_{n+1}, \ldots$. More precisely, for any sequence s in Σ, we define s^{+n} inductively by:

$$s^{+0} = s$$
$$s^{+n} = (s^{+(n-1)})^+, \quad \text{for } n > 0 . \qquad (3\text{-}3)$$

In almost all models of programs, one defines a "next state" relation next on pairs of states, where y next x means that starting in state x and executing one program step can put the program into state y. For a nondeterministic program, there may be several possible next states y. In some models of programs, the set Σ of possible executions is the set of all sequences of states $s_0, s_1, \ldots$ such that s_{i+1} next s_i. This set Σ satisfies property E. In concurrent programs, the next state relation is usually defined in terms of arbitrarily choosing an active process and executing one step of that process. However, some restriction is often placed on how that choice can be made in order to guarantee some sort of "fair scheduling" of process execution. The following are three possible scheduling requirements.

Weak Eventual Fairness
: A process cannot remain active forever without ever being chosen.

Strong Eventual Fairness
: A process cannot be active infinitely often without ever being chosen.

Short-Term Fairness
: There is an integer N -- which may be a function of the state -- such that a process cannot be active for N consecutive steps without being chosen. (If N is the number of processes, then this is round-robin scheduling.)

One can then define the set Σ to consist of all sequences of states $s_0, s_1, \ldots$ such that (i) s_{i+1} next s_i for all i, and (ii) the appropriate scheduling requirement holds. For each of the above three scheduling requirements, the resulting set Σ satisfies property E.

3.2. Interpretation of the Assertions

The semantics of a temporal logic system are specified by defining how a temporal logic assertion is to be interpreted as a statement about an underlying model. This is done in two ways: one for the logic of branching time, and one for the logic of linear time.

3.2.1. Branching Time

In the logic of branching time, an assertion A represents a statement about the current state. Hence, we interpret A as a truth-valued function on states. We let the branching time interpretation of an assertion A in the model M = (S, Σ) be the mapping

$$A_B^M : S \rightarrow \{\text{true, false}\}$$

defined inductively as follows.

- If A is an atomic predicate, then for any state x :

  $$A_B^M(x) \equiv x(A) .$$

  (Recall that a state is by definition a truth-valued function on atomic predicates.)

- If A is the logical combination of simpler assertions, then its interpretation is defined in the obvious way in terms of the interpretations of its components. For example, we have:

  $$(C \vee D)_B^M(x) \equiv C_B^M(x) \vee D_B^M(x) \qquad (3\text{-}4)$$

  for any x in S. (Note that the $\vee$ on the left side of (3-4) is an operator on temporal logic formulas; the one on the right side is the ordinary logical operation on truth values.)

- The interpretations of $\Box A$ and $\rightarrow A$ are defined as follows in terms of the interpretation of A , where x is any element of S. (Recall the definition of Σ_x given by (3-2).)

  $$(\Box A)_B^M(x) \equiv \forall\, s \in \Sigma_x : [\forall\, n \geq 0 : A_B^M(s_n)] \qquad (3\text{-}5)$$

  $$(\rightarrow A)_B^M(x) \equiv \forall\, s \in \Sigma_x : [\exists\, n \geq 0 : A_B^M(s_n)] \qquad (3\text{-}6)$$

Since all assertions are obtained from atomic predicates, logical connectives and the temporal operators, this defines A_B^M for any assertion A .

We say that the assertion A is M-valid in the logic of branching time, written $M \models_B A$, if $A_B^M(x)$ is true for all x in S . In other words, we have:

$$(M \models_B A) \equiv \forall\, x \in S : A_B^M(x) . \qquad (3\text{-}7)$$

Using the definition $\Diamond A \equiv \sim\Box\sim A$, one easily obtains the following from (3-5):

$$(\Diamond A)_B^M(x) \equiv \exists\, s \in \Sigma_x : [\exists\, n \geq 0 : A_B^M(s_n)] .$$

Comparing this with (3-6), we see that the interpretations of $\rightarrow A$ and $\Diamond A$ in general are not equal. The former involves a universal quantification over all possible futures, and the latter involves an existential quantification. For any assertion A , it is easy to find models in which one of the assertions $\Diamond A$ and $\rightarrow A$ is valid and the other is not. This formalizes our previous observation that "eventually" means eventually happening in every possible future, while "not never" means eventually happening in some possible future. Hence, "sometime" and "not never" are not equivalent in the branching time theory.

A deterministic system is one in which for every state x there is at most one possible next state y . In this case, Σ_x consists of a single element, so universal and existential quantification over it are the same. Hence, "eventually" and "not never" are equivalent for a deterministic system. In fact, the theories of branching time and linear time are equivalent for a deterministic system, since the only "possible" future is the single "real" one.

3.2.2. Linear Time

In the temporal logic of linear time, an assertion represents a statement about the actual current and future behavior of the program. Hence, we let the linear time interpretation of an assertion A in the model M = (S, Σ) be the mapping

$$A_L^M : \Sigma \rightarrow \{\text{true, false}\}$$

defined inductively as follows.

- If A is an atomic predicate, then for any state x :

  $$A_L^M(x) = x(A) . \qquad (3\text{-}8)$$

- If A is the logical combination of simpler assertions, then its interpretation is defined in the obvious way in terms of the interpretations of its components. For example,

  $$(C \vee D)_L^M(x) \equiv C_L^M(x) \vee D_L^M(x) \qquad (3\text{-}9)$$

  for any x in S.

- For any assertion A , the interpretations of $\Box A$ and $\rightarrow A$ are defined as follows, where x is any element of S. (Recall the definition of $s^{(+n)}$ given by (3-3).)

  $$(\Box A)_L^M(s) \equiv \forall\, n \geq 0 : A_L^M(s^{+n}) \qquad (3\text{-}10)$$

  $$(\rightarrow A)_L^M(s) \equiv \exists\, n \geq 0 : A_L^M(s^{+n}) \qquad (3\text{-}11)$$

We say that an assertion A is M-valid in the logic of linear time, written $M \models_L A$, if $A_L^M(s)$ is true for every sequence s in Σ . In other words, we have:

$$(M \models_L A) \equiv \forall\, s \in \Sigma : A_L^M(s) . \qquad (3\text{-}12)$$

Using the definition $\Diamond A \equiv \sim\Box\sim A$, it follows easily from (3-10) and (3-11) that $(\rightarrow A)_L^M \equiv (\Diamond A)_L^M$ for any assertion A . Hence, the linear time assertion $\rightarrow A \equiv \Diamond A$ is M-valid for every model M , so "sometime" is the same as "not never" in the theory of linear time.

4. EXPRESSIVENESS

We now consider the expressiveness of the branching and linear time temporal logics -- i.e., what statements about the underlying models can be expressed by assertions in these logics. Not all statements about an underlying model are expressible. For example, the statement that a

model satisfies the short-term fairness condition described in Section 3.1 is not expressible in either of our two temporal logics.

In this section, we compare the expressiveness of the temporal logic systems of branching time and linear time. We prove that neither is more expressive than the other -- each can express things that the other cannot. We also argue that the expressive power of the logic of branching time indicates that it is better for reasoning about nondeterministic programs, while the logic of linear time is better for reasoning about concurrent programs.

4.1. Definitions

To discuss formally the expressive power of our temporal logics, we must define what it means for an assertion of the branching time logic to have the same meaning as an assertion of the linear time logic. We can do this because we have defined the semantics of both temporal logics in terms of the same underlying models. We make the following obvious definition: an assertion A in a logic X is M-equivalent to an assertion B in a logic Y if either A and B are both M-valid (in their respective logics), or if neither one is M-valid. In other words, A and B are M-equivalent if $(M \models_X A) \equiv (M \models_Y B)$.

M-equivalence for a single model M is not an interesting concept, because any assertion is M-equivalent to one of the trivial assertions true or false. (We can define true to equal $P \vee \neg P$ and false to equal $P \wedge \neg P$, for some atomic predicate P .) In other words, every assertion is simply true or false for a particular model. The interesting concepts of equivalence are ones in which the two assertions are equivalent for some class of models.

The strongest form of equivalence is when two assertions are equivalent for all models. We therefore say that two assertions are strongly equivalent if they are M-equivalent for all models M . For example, any predicate P in branching time logic is strongly equivalent to the assertion $\Box P$ in linear time logic. To prove this, note that for a model $M = (S, \Sigma)$, P is M-valid in branching time logic if and only if $\forall x \in S: \forall s \in \Sigma_x : s(P)$ is true, $\Box P$ is M-valid in linear time logic if and only if $\forall s \in \Sigma \; \forall n \geq 0 : s^{+n}(P)$ is true, and property E implies that these two conditions are equivalent.

Strong equivalence implies that the two assertions have the same meaning regardless of the meaning of their component atomic predicates. It is natural to define a weaker form of equivalence -- one in which two assertions are the same for a particular meaning of their atomic predicates. For example, we might be interested only in models for which atomic predicates such as "a > 0" can be interpreted as statements about a program variable named "a", so that the truth of "a > 1" implies the truth of "a > 0". This means that we want to restrict ourselves to a particular set of states. For a set of states S , we define an assertion in one logic to be S-equivalent to an assertion in another logic if the two assertions are M-equivalent for every model M having S as its set of states. Strong equivalence obviously implies S-equivalence.

4.2. Inequivalence Results

We now show that the two temporal logics have different expressive powers, and that neither is strictly more expressive than the other -- each can express statements that the other cannot. We do this by showing that for any nontrivial set of states S , there is an assertion in each logic that is not S-equivalent to any assertion in the other logic. Note that this is a stronger result than proving the nonexistence of any strongly equivalent assertion.

We say that a predicate P is trivially true for a set of states S if $x(P)$ = true for all x in S . Trivially false is similarly defined. A predicate is said to be trivial for a set of states if it is trivially true or false.

Our inequivalence results are contained in the following two theorems.

Theorem 1: For any set of states S , if P is a nontrivial predicate for S , then the assertion $\Diamond P$ in the branching time logic is not S-equivalent to any assertion in the linear time logic.

Proof: We first define two models $M_1 = (S, \Sigma_1)$ and $M_2 = (S, \Sigma_2)$, with the given set of states. Since P is nontrivial, there are states p and q in S such that $p(P)$ = true and $q(P)$ = false . Let Σ_1 consist of all infinite sequences of the form $x, p, p, p, \ldots$ with x different from q , together with the sequence $q, q, q, \ldots$. Let Σ_2 be the set containing all the elements of Σ_1 together with the sequence $q, p, p, \ldots$.

From equations (3-5) and (3-7) and the definition of $\Diamond$, it is easy to see that for both of these models M_i , $\Diamond P$ is M_i-valid in branching time logic if and only if the following expression is true:

$$\forall x \in S: \exists s \in \Sigma_x: \exists n \geq 0: s_n(P) .$$

From this, we see that $\Diamond P$ is M_2 valid, but it is not M_1 valid. However, Σ_1 is a subset of Σ_2, and it is an immediate consequence of the definition of M-validity for the linear time logic (equation (3-12)) that any assertion which is M-valid for one model must be M-valid for a model having fewer execution sequences. Any linear time assertion that is M_2-valid must therefore be M_1-valid. Hence, there is no assertion of the linear time logic which is S-equivalent to the assertion $\Diamond P$ of the branching time logic. □

Theorem 2: For any set S of states, if P is a nontrivial predicate, then the assertion $\rightarrow \Box P$ of the linear time logic is not S-equivalent to any assertion of the branching time logic.

Proof: Let PP denote the set of all states p such that $p(P)$ equals true. Let $M_1 = (S, \Sigma_1)$ and $M_2 = (S, \Sigma_2)$, where Σ_1 consists of all sequences which end with an infinite sequence of

elements all in PP, and Σ_2 consists of all sequences containing an infinite number of elements of PP. Since P is not trivially true, it is easy to verify from (3-12) that the assertion $\rightarrow\Box P$ is M_1-valid but is not M_2-valid in the linear time logic.

Now let A be any assertion. We first show that when M is either of the models M_1 or M_2 :

$$(\Box A)^M_B(x) \equiv \forall\, y \in S\colon A^M_B(y) \qquad (4\text{-}1)$$

$$(\rightarrow A)^M_B(x) \equiv A^M_B(x) \vee \forall\, p \in PP\colon A^M_B(p)\ . \qquad (4\text{-}2)$$

By (3-5) of Section 3.2, to prove (4-1) we have to show that

$$\forall\, s \in \Sigma_x\colon [\forall\, n \geq 0\colon A^M_B(s_n)\ ] \equiv \forall\, y \in S\colon A^M_B(y)$$

This follows easily from the observation that in both models, for any states x and y , Σ_x contains a sequence of the form x, y , ...

To prove (4-2), it follows from (3-6) that we must show that for both models:

$$\forall\, s \in \Sigma_x\colon [\exists\, n \geq 0\colon A^M_B(s_n)] \equiv A^M_B(x) \vee [\forall\, p \in PP\colon A^M_B(p)]\ .$$

The right side implies the left side because in both models, every sequence in Σ_x contains x and some element of PP . The left side implies the right side because in both models, Σ_x contains the sequence x, p, p, p, ... for every p in PP. This completes the proof of (4-1) and (4-2) for both models.

Let x be any element of S. If Q is a predicate, then $Q^M_B(x)$ has the same truth value in both models. If A is any assertion such that $A^M_B(x)$ has the same truth value for both models, then: (i) since (4-1) holds in both models, $(\Box A)^M_B(x)$ must also have the same truth value for both models; and (ii) since (4-2) holds for both models, $(\rightarrow A)^M_B(x)$ must also have the same truth value for both models. Any assertion is built up from predicates using only ordinary logical operators and the temporal operators $\Box$ and $\rightarrow$, so a simple induction argument shows that for any assertion A , $A^M_B(x)$ has the same truth values for both models. Since this is true for any state x , it follows from the definition of M-validity for the branching time logic (3-7 of Section 3.2) that any assertion A of branching time logic is M_1-valid if and only if it is M_2-valid. However, the assertion $\rightarrow\Box P$ of linear time logic is M_1-valid and not M_2-valid. Hence, it is not S-equivalent to any assertion in branching time logic. □

4.3. Nondeterminism versus Concurrency

In almost all formal models of concurrent processing, a concurrent system is represented by a nondeterministic sequential one. The concurrent execution of two operations that takes place in the real system appears in the model as the nondeterminism of which one occurs first. This use of nondeterminism to model concurrency has caused some confusion, since the type of nondeterminism involved is conceptually quite different from the nondeterminism studied in automata theory and in the theory of nondeterministic algorithms.

4.3.1. Nondeterminism

In automata theory, a nondeterministic machine is thought of as one that simultaneously pursues all possibilities. The machine is considered to complete its computation successfully if one of these possibilities succeeds. This has led to the study of nondeterministic algorithms, implemented by concurrently executing all possibilities and stopping the entire computation if one succeeds. The theory of branching time is appropriate for reasoning about this type of nondeterminacy. If H is a predicate which represents the statement that the machine has reached a successful completion, then the assertion $\Diamond H$ in the logic of branching time states that some computation will succeed. Theorem 1 shows that this cannot be expressed in the temporal logic of linear time, so there is no choice about which type of temporal logic is appropriate here.

Pratt [16] has developed the elegant formalism of dynamic logic for discussing nondeterminism of this kind. To express his system in terms of our model, we must divide the set of halting states -- those with no possible next states -- into two classes: failed states, and successful states. Let H be the predicate that is true only for successful halted states. The dynamic logic assertion [a]P corresponds to the temporal logic assertion $\Box(H \supset P)$ for the model defined by the program a .

Harel and Pratt [7] extended the original dynamic logic to the system DL^+ in order to consider nondeterministic algorithms for which all the possible choices lead to terminating computations. From our point of view, we see that an extension was needed because this new type of termination cannot be expressed with only the temporal operator $\Box$, but requires the additional operator $\rightarrow$. All the formulas of DL^+ can be obtained by adding formulas of the form $[a]^+$"true" to the original dynamic logic of [16]. This formula can be expressed in the branching time logic defined by the program a as $\rightarrow H$. The meaning of termination for nondeterministic programs is discussed at length in [8].

Dynamic logic only allows one to reason about the states before and after program execution. In [17], Pratt extended dynamic logic to "process logic" which allows reasoning about the states entered during execution. Process logic is a form of branching time temporal logic.

4.3.2. Concurrency

Our view of concurrent programs is that the nondeterminism represents different possibilities, only one which actually occurs. This suggests that the linear time temporal logic should be more appropriate for reasoning about concurrent programs. Although "appropriateness" is not a provable property, we will give what we feel to be strong arguments that this is indeed the case.

Recall that the two basic properties one proves about concurrent programs -- safety and liveness -- are expressed by the assertions (2-1) and (2-3). It can be shown that each of those assertions in the linear time logic is strongly equivalent to the identical assertion in the branching time logic. Hence, both logics can express the required correctness properties. The superiority of linear time logic manifests itself only in the attempt to prove these properties.

The correctness of a concurrent program usually depends upon the fairness properties assumed for scheduling the execution of operations in different processes. One type of fairness that is often assumed is the strong eventual fairness condition described in Section 3.1. This condition can be expressed in the temporal logic of linear time by the assertion $(\rightarrow \Box \sim ACTIVE) \vee \rightarrow CHOSEN$, where ACTIVE and CHOSEN represent appropriate predicates. It follows from Theorem 2 that this cannot be expressed in the logic of branching time. I.e., this assertion is not S-equivalent to any branching time assertion for a nontrivial set of states S. (If it were, then it would have to be equivalent for the subset of states in which CHOSEN is trivially false, contradicting Theorem 2 for $P = \sim ACTIVE$.) This strongly suggests that the linear time logic is more appropriate than the branching time logic.

Another argument in favor of the linear time logic comes from our experience in proving liveness properties of concurrent programs. We find ourselves continually using the following type of reasoning to prove that P eventually becomes true.

> We show that if P is always false during the program execution, then the program will cause P to become true. Hence, P cannot remain false forever, so it must eventually become true.

This reasoning is based upon the hypothesis that either P is eventually true, or it is always false. In other words, it assumes an axiom of the form $\rightarrow P \vee \Box(\sim P)$. In the linear time theory, this assertion is M-valid for all models M. However, it is easy to construct a model for which the assertion is not M-valid in the branching time logic, so this reasoning cannot be used in branching time logic.

The logic of linear time corresponds to the way one tends to reason informally about concurrent program execution. We have therefore found it easy to use the linear time logic to formalize the proof techniques described in [9] and [14]. We do not know if it is always possible to prove the same properties of concurrent programs by reasoning within the branching time logic. However, our experience has convinced us that even if it is possible, the resulting proofs will not be as simple and natural as the ones using the linear time logic.

It might be argued that one should have a system powerful enough to subsume both the branching and linear time logics. Such systems can be constructed. However, that approach is based upon the misguided notion that the more expressive a system is, the better it is. We could get a very expressive system by simply reasoning about the underlying models. However, one uses temporal logic to hide the irrelevant details of the models. The ideal logic would be one in which we could express all the relevant properties of the models and none of the irrelevant ones. We have not considered temporal logics having an explicit "next instant" operator, such as the one studied in [6], because we feel that they are too expressive.

Since what is relevant depends upon the application, different logics should be better for different applications. We believe that the temporal logic of linear time (as generalized in Section 6 has precisely the expressiveness that one needs for reasoning about concurrent programs.

5. THE THEOREMS OF TEMPORAL LOGIC

5.1. Validity

An assertion is M-valid in a temporal logic if its interpretation is true for the model M. We are also interested in assertions whose interpretations are true for more than just a single model. We define an assertion A to be strongly valid if it is M-valid for all models M. We say that A is S-valid for a set S of states if it is M-valid for every model M having S as its set of states.

A strongly valid assertion is one that is true for any interpretation of its atomic predicates. For example, it is easy to check that the following assertion is strongly valid for the logics of both branching and linear time.

$$\Box(P \wedge Q) \equiv \Box P \wedge \Box Q$$

This assertion is a true statement about any model, regardless of how the atomic predicates P and Q are interpreted in that model. Strongly valid assertions are the tautologies of temporal logic.

Now consider the assertion

$$\Box \text{"a > 1"} \supset \Box \text{"a > 0"},$$

where "a > 0" and "a > 1" are atomic formulas. This assertion is not valid for all models, since there are models having states in which "a > 1" is true and "a > 0" is false. However, such models are of no interest if we are trying to reason about a program variable named "a". In this case, we are interested in S-validity, where S is the set of possible program states. The above assertion will be S-valid for such a set S of states in both the branching and linear time theories.

5.2. Deductive Systems

Thus far, we have discussed the validity of temporal logic assertions, but have said nothing about proving things. A temporal logic deductive system consists of a formal method for deriving theorems. We write $\vdash A$ to denote that the assertion A is a theorem of a deductive system.

A deductive system generally consists of a collection of axioms -- assertions that are assumed to be theorems -- and a collection of inference rules for deriving theorems from other theorems. For example, the following might be taken as a

temporal logic axiom (in either a branching or linear time logic):

$$\vdash \Box(A \wedge B) \equiv \Box A \wedge \Box B \ . \qquad (5\text{-}1)$$

In this formula, A and B are formal parameters that represent any assertion. It may be viewed either as an infinite set of axioms -- one for each choice of the assertions A and B ; or else as a single axiom -- in which case there must be a rule of inference that permits one to obtain new theorems by substituting arbitrary assertions for A and B in (5-1). The following is an example of an inference rule:

$$\text{If } \vdash A \text{ then } \vdash \Box A \ . \qquad (5\text{-}2)$$

Note that this is not the same as $\vdash A \supset \Box A$, which is not a valid theorem. Rule (5-2) is the "necessitation" rule of modal logic.

A deductive system is said to be valid if all its theorems are valid. It is said to be complete if every valid assertion is a theorem. There are three types of validity that are of interest: strong validity, S-validity and M-validity. They lead to three classes of theorems: theorems true for all models, theorems true for all models with a specific set of states, and theorems true for a model representing a particular program.

5.3. Tautologies

A strongly valid assertion is one that is trivially true, in the sense that its truth does not depend in any way upon the model under consideration. We therefore call such an assertion a tautology. A temporal logic deductive system should be able to prove such trivial theorems, so it should contain a subcollection of axioms and rules of inference for proving tautologies. For example, it might contain the axiom (5-1) and the rule of inference (5-2). We now consider the deductive system formed by this subcollection of axioms and inference rules for deriving tautologies.

A deductive system is called tautologically valid if all of its theorems are tautologies. To prove that a system is tautologically valid, one must prove that each axiom is a tautology, and that each inference rule can generate only tautologies. This is a straightforward task.

A deductive system is said to be tautologically complete if every tautology is a theorem. Finding a complete logical system is more difficult than finding a valid one. In [18], Rescher and Urquhart give axioms and inference rules that are sufficient to prove every tautology for the tense logics K_b and K_1 , which are closely related to our temporal logics of branching time and linear time, respectively. It should be possible to adapt their axioms and inference rules to obtain ones that are sufficient to ensure tautological completeness for temporal logic systems of branching and linear time. However, such an exercise is beyond the scope of this paper.[2]

5.4. S-Valid Formulas

An S-valid assertion is one that is true when its atomic predicates are given the meanings implied by the set S of states. To prove S-valid theorems, a deductive system must be able to prove tautologies, and it must also be able to derive theorems about predicates. The following theorem shows that this is sufficient, because any S-valid assertion can be derived from a tautology and an S-valid predicate. Questions of validity and completeness are reduced to the corresponding questions for tautologies and for predicates.

To see that the following theorem does what we claim, observe that by using the valid inference rule (5-2), substitution, and modus ponens (deducing $\vdash B$ from $\vdash A$ and $\vdash A \supset B$), we can deduce $\vdash A$ from $\vdash P$ and $\vdash \Box P \supset A$.

Theorem 3: For the logics of both branching and linear time: if S is a set of states and A is an S-valid assertion, then there exists an S-valid predicate P such that $\Box P \supset A$ is strongly valid.

Proof: Let $P_1, \ldots, P_n$ be the predicates appearing in the assertion A . Define a boolean-valued function F of n boolean arguments by letting $F(a_1, \ldots, a_n)$ equal true if and only if there is a state x in S such that $x(P_i) = a_i$ for all i . Such a function can always be expressed as a logical combination of its arguments. We can then define P to be $F(P_1, \ldots, P_n)$, where the latter expression is the predicate obtained in the obvious way from the expression of F as a logical combination of its arguments. Note that for any state x (not necessarily in S):

$$x(P) \equiv \exists\, x' \in S: \forall\, i: x(P_i) = x'(P_i). \qquad (5\text{-}3)$$

This clearly implies that P is S-valid.

To complete the proof, we must show that the assertion $\Box P \supset A$ is M-valid for all models M . We prove this for the linear time logic. The proof for the branching time logic is very similar and is omitted. It follows from the definitions of Section 3.2 that for any model M and any sequence of states s :

$$(\Box P \supset A)^M_L(s) \equiv$$

$$(\forall\, n \geq 0: s_n(P)) \supset A^M_L(s) \ . \qquad (5\text{-}4)$$

For any model $M = (S'', \Sigma)$, define a new model $M' = (S', \Sigma')$ by letting $S' = \{x \in S'' : x(P) = \text{"true"}\}$ and $\Sigma' = \{s \in \Sigma : s_n \in S' \text{ for all } n\}$. It follows easily from (5-4) and (3-12) that the assertion $\Box P \supset A$ is M-valid if and only if it is M'-valid. Moreover, this assertion is M'-valid if and only if the following is true:

$$\forall\, s \in \Sigma' : A^{M'}_L(s) \qquad (5\text{-}5)$$

Hence, we need only prove (5-5).

---

[2] Such a completeness result is claimed by Pnueli in [15]. However, he based his temporal logic of linear time on the tense logic K_b of branching time (plus the identification of $\rightarrow$ and $\diamond$), so it is difficult to evaluate his claim.

Let ϕ: S' $\rightarrow$ S be any mapping such that for every $x \in S'$: $\phi(x)(P_i) = x(P_i)$ for all i . It follows from (5-3) that such a mapping exists. We extend ϕ to be a mapping on sequences in the obvious way so that $\phi(s)_n = \phi(s_n)$. We next define a model $M'' = (S, \Sigma'')$, where S is our original state space, by letting $\Sigma'' = \{\phi(s) : s \in \Sigma'\}$. Since the P_i are the only atomic predicates in the assertion A , it is easy to verify that for all $s \in \Sigma'$:

$$A_L^{M'}(s) \equiv A_L^{M''}(\phi(s)) .$$

Since A is S-valid, it is M"-valid. Hence, this equality and (3-12) imply (5-5), completing our proof of the M-validity of $\Box P \supset A$ for any model M . □

5.5. M-Validity

To prove theorems that are valid only for an individual model, there must be some way to prove properties of that model. For a temporal logic of programs, this means a way of proving properties of a particular program. In practice, one begins with certain elementary theorems about a program, and then manipulates them to prove more complex theorems. For example, consider the following portion of a sequential program without gotos.

var1 := 1

label: ...

We should be able to deduce the following theorem about this program:

$\vdash$ (control at label) $\supset$ (var1 = 1) .

Given this type of elementary theorem, the axioms and deduction rules for deriving tautologies and state valid theorems may be used to deduce more complex theorems about the specific program.

A deductive system for proving properties of programs must therefore have a method for deducing these elementary properties. Such a method provides a formal definition of the semantics of a programming language. For concurrent programs, we believe that the only types of elementary properties that are needed are safety properties and liveness properties.

5.5.1. Safety Properties

The usual method of deducing safety properties rests upon the following induction principle, where next_π denotes the "next state" relation for a program π , and $M(\pi)$ is the model defined by this program.

For any predicate P :

If $\forall\, x,y \in S$:
$(x(P) \wedge y \ \text{next}_\pi \ x) \supset y(P)$

then $M(\pi) \models P \supset \Box P$.

Observe that the hypothesis is a statement about the model and the conclusion asserts the validity of a temporal logic assertion. Thus, it leads to an inference rule for deducing temporal logic theorems from theorems about the underlying model. This induction principle is the basis of all the inductive methods that have been proposed for proving safety properties of programs -- starting from Floyd's original inductive assertion method for proving partial correctness [4]. To prove the safety property (2-1) of Section 2.2, one proves three theorems: (i) $\vdash$ INIT $\supset$ P ; (ii) $\vdash$ P $\supset \Box$P ; and (iii) $\vdash$ P $\supset$ GOOD -- for some suitable predicate P . The first and last of these are theorems about predicates, and are usually easy to prove. The difficult part of constructing a proof is choosing the appropriate predicate P so that (ii) can be proved from the induction principle.

To apply this induction principle, we need a formal method of proving theorems about the model's next relation. This requires developing a formal semantics of the safety properties of the programming language. We have recently developed a method of doing this for concurrent programs, which is described in [10]. It is based upon a method for proving formulas of the form {P} π {Q} , where P and Q are predicates and π is a program statement. This formula is interpreted to mean that if execution is begun anywhere inside π in a state such that P is true, then P will stay true while control remains in π , and Q will be true if and when π terminates. We can then restate the induction principle as follows.

For any predicate P :

if {P} π {P} then $M(\pi) \models P \supset \Box P$.

5.5.2. Liveness Properties

General methods for deducing elementary liveness properties have not yet been developed. Liveness properties for programs written in the simple flowchart language used in [9] and [15] can be proved by introducing the following axiom for each flowchart box of each process:

$\vdash$ (control on arc leading into box) $\supset$

$\rightarrow$ (control on one of the arcs leading from box) .

These axioms describe a system satisfying the eventual fairness properties described in Section 3.1. (In these simple flowchart programs, a process is always active, so weak and strong eventual fairness are equivalent. Waiting is represented by a loop.)

More sophisticated axioms are needed for concurrent programs written in languages with explicit synchronization primitives. For example, consider programs using a semaphore s . There are several types of liveness assumptions we can make for semaphore operations. A common assumption is that if the value of the semaphore s becomes positive infinitely often, then every process waiting on a P(s) operation must eventually complete that operation. (This is a strong eventual fairness assumption.) It can be expressed in linear time logic by adding the following axiom for each occurrence of a semaphore operation P(s) .

$\vdash$ [(control at the P(s) operation)
$\wedge \ \Box\rightarrow (s > 0)$] $\supset$

→ (control after the P(s) operation).

Now consider a weaker form of semaphore which simply guarantees that if s becomes positive, then eventually some pending P(s) operation must be executed. (This is weaker because it allows an individual process to wait forever if other processes are repeatedly executing P(s) operations.) To express the liveness property of such a semaphore, one must introduce a single, complicated axiom that depends upon all the P(s) operations of the entire program.

Further work is needed in the formal specification of liveness properties for synchronization primitives. One can, of course, define these primitives in terms of flowchart programs, thus basing the semantics of the language on the semantics of flowchart programs. However, this does not solve the practical problem of proving the global liveness properties that are achieved by these primitives; it merely pushes the problem back one level.

5.5.3. Completeness

We now consider the question of completeness -- the ability of a deductive system to prove all valid assertions about individual programs. We cannot in general expect this type of completeness, since we can construct an assertion which states that the given program halts. Instead, one can try to construct a deductive system for which a result analogous to Theorem 3 of Section 5.4 holds: any assertion that is valid for a given program can be deduced from a tautology and a valid theorem about predicates. If the system were tautologically complete, any incompleteness would then be due to an incompleteness in the system for reasoning about predicates. This concept of "relative completeness" was introduced by Cook [2] for sequential programs.

Such completeness results have been obtained for particular types of assertions. For example, Owicki [13] and Flon and Suzuki [3] have shown that proving a valid safety property can be reduced to the problem of proving valid theorems about predicates. However, Owicki's proof requires the addition of "dummy" variables to the program, while Flon and Suzuki's result requires the use of nonrecursive predicates. Apt [1] has shown that this is unavoidable: if dummy variables are not allowed (other than for describing the state of program control), then nonrecursive predicates are required.

Certain relative completeness results for liveness properties have also been obtained for models with no fair scheduling requirements. Flon and Suzuki [3] showed that if the set of predicates is rich enough, then the problem of proving certain types of simple liveness properties can be reduced to the problem of proving valid predicates. They considered programs written in a flowchart language in which a waiting condition can be added to delay the execution of an operation. Pnueli [15] has proved a similar result for more general liveness properties of simple flowchart programs.

Although such relative completeness results are of interest, they do not answer what we feel to be the most important question: is any deductive power lost by using temporal logic instead of reasoning directly about the underlying model? Let TL be a deductive system for linear time temporal logic, and let ML be a deductive system for proving theorems about the model $M = (S, \Sigma)$. We say that TL is complete relative to ML if for every assertion A: if $\forall s \in \Sigma : A_L^M(s)$ is provable in ML, then A is provable in TL. (A similar definition can be made for a branching time temporal logic.) No deductive power is lost by using the temporal logic system TL instead of reasoning directly about the model with ML if and only if TL is complete relative to ML.

If we are allowed to introduce dummy variables for reasoning about programs, then for any ML we can construct a temporal logic deductive system TL which is complete relative to ML. This is done by adding a variable that records a complete "trace" of the program's execution. Any reasoning in ML about execution sequences can be mirrored in TL by reasoning about the value of this dummy variable. (This approach was used in [13].)

Introducing such a dummy variable obviously defeats the whole purpose of using temporal logic, since it brings us back to reasoning directly about the model. Given a system ML for reasoning about program execution sequences, we would like to find a temporal logic deductive system TL that does not use dummy variables and is complete relative to ML. We suspect that this is not always possible. For example, if a multiprocess program uses a short term fairness scheduling discipline, then the set of execution sequences Σ would satisfy an important scheduling property that cannot be expressed in our temporal logics, so reasoning about the model should enable one to prove properties that cannot be proved with a temporal logic deductive system. However, we conjecture that one can construct such a relatively complete temporal logic deductive system for a useful class of programs and an interesting class of systems ML. We regard the study of this type of relative completeness to be a useful area for further research.

6. "AS LONG AS"

Thus far, we have restricted the discussion to temporal logics that use only the temporal operators "always" ($\Box$) and "eventually" ($\rightarrow$). We now show that these operators cannot express certain important properties of concurrent programs, and briefly describe a more general operator. We consider only the linear time theory, since we are concerned with describing properties of concurrent programs. More general operators can also be defined for the branching time theory.

In the linear time theory, $\rightarrow$ is equivalent to $\Diamond$, which is equivalent to $\sim\Box\sim$; so we have to consider only the single temporal operator $\Box$. The assertion $\Box B$ represents the statement that B is "always" true in the single (real) future. We can generalize this to an assertion $A \Box B$ which asserts that B is true "as long as" the assertion A remains true. Formally, the meaning of $A \Box B$ is defined by extending the interpretation defined in Section 3.2.2 as follows.

$$(A \Box B)_L^M(s) \equiv$$

$$\forall n \geq 0: [\forall i \in \{0, \ldots, n\}: A_L^M(s_i)]$$

$$\supset B_L^M(s^{+n}) \qquad (6\text{-}1)$$

A temporal logic using assertions constructed with the dyadic operator $\Box$ will be a called a generalized temporal logic, and the logic we have been discussing up to now will be called ordinary temporal logic. It is easy to check the strong validity of the following equivalence, which shows that the generalized temporal logic is at least as expressive as the ordinary one:

$$\text{true } \Box\ B \equiv \Box B .$$

The following theorem shows that the generalized logic is actually more expressive than the ordinary one.

Theorem 4: For any set of states S, if P, Q and R are any predicates such that none of the three predicates $P \wedge Q \wedge R$, $\sim Q$, $\sim P \wedge Q \wedge \sim R$ is trivially false for S, then the assertion $P \supset (Q \Box R)$ is not S-equivalent to any ordinary temporal logic formula.

Proof: The idea is the same as for the proofs in Section 4.2: we construct two models $M = (S, \Sigma)$ and $M' = (S, \Sigma')$ which cannot be distinguished by any ordinary temporal logic formula, such that the assertion $P \supset (Q \Box R)$ is valid for one model and false for the other. Let a, b and c be states such that $a(P \wedge Q \wedge R)$, $b(\sim Q)$ and $c(\sim P \wedge Q \wedge \sim R)$ are true. Let Σ consist of the following three infinite sequences:

$s[1] = a, b, c, a, b, c, \ldots$

$s[2] = b, c, a, b, c, a, \ldots$

$s[3] = c, a, b, c, a, b, \ldots$;

and let Σ' consist of the following three infinite sequences:

$s'[1] = a, c, b, a, c, b, \ldots$

$s'[2] = b, a, c, b, a, c, \ldots$

$s'[3] = c, b, a, c, b, a, \ldots$.

It is easy to verify that $P \supset (Q \Box R)$ is M-valid but is not M'-valid.

We now prove that any assertion constructed using predicates, logical connectives, and the unary operator $\Box$ cannot be valid for one model and invalid for the other. To do this, we show that for any such assertion A, and each $i = 1, 2, 3$:

$$A_L^M(s[i])\} \equiv A_L^{M'}(s'[i]) . \qquad (6\text{-}2)$$

The proof is by induction. It is obviously true if A is a predicate, since the first element of the sequence $s[i]$ is the same as the first element of the sequence $s'[i]$. It is easy to see that if (6-2) holds for a collection of assertions A, then it holds for any logical combination of those assertions. To complete the proof, we need only show that if (6-2) holds for all i, then:

$$(\Box A)_L^M(s[i]) \equiv (\Box A)_L^{M'}(s'[i]) .$$

But this follows easily from (3-10). From (6-2) we conclude that any ordinary linear time assertion A that is M-valid must also be M'-valid, completing the proof. □

Assertions of the form $A \Box B$ can be used to express a more general class of safety properties. One such property is "first-come-first-served", which can be expressed as follows: "if process p requests service before process q does, then process q cannot be served before process p". This is not a liveness property, since it does not state that any process eventually does get served. It is expressed formally by the assertion

$$P.FIRST \supset (P.WAITING \Box Q.NOT.SERVED),$$

where the predicates are defined to have the following meanings.

P.FIRST:
p is waiting for service and q is neither waiting for service nor being served.

P.WAITING:
p is waiting for service.

Q.NOT.SERVED:
q is not being served.

The above theorem shows that this assertion expresses a property that cannot be expressed with only the unary operator $\Box$.

One can define a dual $\Diamond$ to the dyadic operator $\Box$ as follows:

$$A \Diamond B \equiv \sim(\sim A \Box \sim B) .$$

(This makes $\Box$ and $\Diamond$ duals in the same sense that $\wedge$ and $\vee$ are.) The assertion $A \Diamond B$ represents the statement that B eventually becomes true, and it becomes true before A does. This is equivalent to the following assertion: $\Diamond B \wedge (\sim B \Box \sim A)$. We have thus far found no need for the dyadic operator $\Diamond$.

7. CONCLUSION

Temporal logic provides a very convenient language for stating and proving properties of concurrent programs, and we believe that it will also provide an important logical foundation for the semantics of concurrent programs. The linear time temporal logic originally used by Pnueli in [15] is very simple, involving the addition of the single temporal operator $\Box$ to ordinary logic. It allows one to hide many irrelevant concepts that appear in the ordinary computational models used to describe concurrent programs. We believe that when generalized as in Section 6, it is adequate for expressing all the relevant properties of concurrent programs, but more experience is needed before we can be certain of this. We still do not know if a temporal logic deductive system can prove all of the relevant properties that could be proved directly from the models.

We have found that when thinking informally about such concepts as "henceforth" and "eventually", most computer scientists seem to adopt the branching time theory. This is one reason for our interest in branching time logic. Branching time logic is important for studying nondeterministic programs, and comparing the two types of temporal logic has helped us to understand better the relationship between concurrency and nondeterminism.

ACKNOWLEDGMENTS

We wish to thank Albert Meyer and David Harel for their critical comments on an earlier version of this paper. We also benefited from numerous discussions with Susan Owicki.

REFERENCES

1. Krzystof R. Apt. Recursive Assertions and Parallel Progams, 2 October 1979.

2. Stephen A. Cook. Soundness and Completeness of an Axiom System for Program Verification. SIAM J. Comput. 7, 1 (February 1978), 70-90.

3. L. Flon and N. Suzuki. Consistent and Complete Proof Rules for the Total Correctness of Parallel Programs. Proceedings of 19th Annual Symp. on Found. of Comp. Sci., IEEE, October, 1978.

4. R. W. Floyd. Assigning Meanings to Programs. Proc. Symposium on Applied Math., Vol. 19, Amer. Math. Soc., 1967, pp. 19-32.

5. N. Francez and A. Pnueli. A Proof Method for Cyclic Programs. Proceedings of the 1976 International Conference on Parallel Processing, IEEE, 1976, pp. 235-245.

6. D. Gabbay, A. Pnueli, S. Shelah and Y. Stavi. Completeness Results for the Future Fragment of Temporal Logic.

7. D. Harel and V. R. Pratt. Nondeterminism in Logics of Programs. Proceedings of a Symposium on Principles of Programming Languages, ACM-Sigplan, January, 1978.

8. David Harel. On the Total Correctness of Nondeterministic Programs. RC 7691, IBM T.J. Watson Research Center, 1979. To appear in Theoretical Computer Science.

9. L. Lamport. Proving the Correctness of Multiprocess Programs. IEEE Trans. on Software Engineering SE-3, 2 (March 1977), 125-143.

10. L. Lamport. The 'Hoare Logic' of Concurrent Programs. CSL-79, SRI International, October, 1978.

11. L. Lamport. A New Approach to Proving the Correctness of Multiprocess Programs. ACM Trans. on Programming Languages and Systems 1, 1 (July 1979), 84-97.

12. Z. Manna and R. Waldinger. Is 'Sometime' Sometimes Better than 'Always'?. Comm. ACM 21, 2 (February 1978), 159-172.

13. S. Owicki. Axiomatic Proof Techniques for Parallel Programs. Ph.D. Th., Cornell University, August 1975.

14. S. Owicki and D. Gries. An Axiomatic Proof Technique for Parallel Programs. Acta Informatica 6, 4 (1976), 319-340.

15. A. Pnueli. The Temporal Logic of Programs. 18th Annual Symposium on the Foundations of Computer Science, IEEE, November, 1977.

16. V. R. Pratt. Semantical Considerations on Floyd-Hoare Logic. 17th Symposium on Foundations of Computer Science, IEEE, October, 1976.

17. V.R. Pratt. Process Logic: Preliminary Report. Proc. 6th Ann. ACM Symp. on Principles of Programming Languages, ACM, January, 1979, pp. 93-100.

18. N. Rescher and A. Urquhart. Temporal Logic. Springer-Verlag, New York, 1971.