"Sometimes" and "Not Never" Revisited: On Branching versus Linear Time Temporal Logic

E. ALLEN EMERSON

University of Texas, Austin, Texas

AND

JOSEPH Y. HALPERN

IBM Research Laboratory, San Jose, California

Abstract. The differences between and appropriateness of branching versus linear time temporal logic for reasoning about concurrent programs are studied. These issues have been previously considered by Lamport. To facilitate a careful examination of these issues, a language, CTL*, in which a universal or existential path quantifier can prefix an arbitrary linear time assertion, is defined. The expressive power of a number of sublanguages is then compared. CTL* is also related to the logics MPL of Abrahamson and PL of Harel, Kozen, and Parikh. The paper concludes with a comparison of the utility of branching and linear time temporal logics.

Categories and Subject Descriptors: D.2.1 [**Software Engineering**]: Requirements/Specifications–*languages*; F.3.1. [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs–*assertions*; *logics of programs*; *specification techniques*

General Terms: Design, Theory

Additional Key Words and Phrases: Concurrent programs, parallelism, temporal logic

1. *Introduction*

Temporal logic [27, 28] provides a formalism for describing the occurrence of events in time that is suitable for reasoning about concurrent programs (cf. [24]). In defining temporal logic, there are two possible views regarding the underlying nature of time. One is that time is linear: At each moment there is only one possible future. The other is that time has a branching, treelike nature: At each moment, time may split into alternate courses representing different possible futures. De-

---

A preliminary version of this paper was presented at the 1983 ACM Symposium on Principles of Programming Languages, under the title "Sometimes" and "Not Never" Revisited: On Branching versus Linear Time.

E. A. Emerson was partially supported by a University of Texas URI Summer Research Award, a departmental grant from IBM, and National Science Foundation grant MCS 83-02878. Some of this work was performed while J. Y. Halpern was a visiting scientist jointly at MIT and Harvard, where he was partially supported by a grant from the National Sciences and Engineering Research Council of Canada and National Science Foundation grant MCS 80-10707.

Authors' addresses: E. A. Emerson, Computer Sciences Department, University of Texas, Austin, TX 78712; J. Y. Halpern, IBM Research Laboratory, San Jose, CA 95193

pending upon which view is chosen, we classify (cf. [29]) a system of temporal logic as either a linear time logic in which the semantics of the time structure is linear, or a system of branching time logic based on the semantics corresponding to a branching time structure. The modalities of a temporal logic system usually reflect the semantics regarding the nature of time. Thus, in a logic of linear time, temporal modalities are provided for describing events along a single time path (cf. [14]). In contrast, in a logic of branching time, the modalities reflect the branching nature of time by allowing quantification over possible futures (cf. [1, 7]).

Some controversy has arisen in the computer science community regarding the differences between and appropriateness of branching versus linear time temporal logic. In a landmark paper [19] intended to "clarify the logical foundations of the application of temporal logic to concurrent programs," Lamport addresses these issues. He defines a single language based on the temporal operators "always" and "sometimes". Two distinct interpretations for the language are given. In the first interpretation, formulas make assertions about paths, whereas in the second interpretation they make assertions about states. Lamport associates the former with linear time and the latter with branching time (although it should be noted that in both cases the underlying time structures are branching). He then compares the expressive power of linear time and branching time logic and finds them incomparable. On the basis of his comparison and other arguments, he concludes that, although branching time logic is suitable for reasoning about nondeterministic programs, linear time logic is preferable for reasoning about concurrent programs.

In this paper, we reexamine Lamport's arguments and reach somewhat different conclusions. We first reprove Lamport's incomparability results in a setting more appropriate to concurrency by considering **R**-*generable* structures (i.e., structures generated by a binary relation similar to those used in the logics of [13] and [2]; cf. [5]). But we then show that these incomparability results only apply to the two particular systems he defines. Since Lamport's arguments, all of which are based on this one comparison, do not apply in general, sweeping conclusions regarding branching versus linear time logic are not justified.

We argue that there are several different aspects to the problem of designing and reasoning about concurrent programs. Although the specific modalities needed in a logic depend on the precise nature of the purpose for which it is intended, we can make some general observations regarding the choice between a system of branching or linear time. We believe that linear time logics are generally adequate for verifying the correctness of preexisting concurrent programs. For verification purposes, we are typically interested in properties that hold for all computation paths. It is thus satisfactory to pick an arbitrary path and reason about it. However, there are applications where we need the ability to assert the existence of alternative computation paths as provided by a branching time logic. This arises from the nondeterminism—beyond that used to model concurrency—present in many concurrent programs. In order to give a complete specification of such a program, we must ensure that there are viable computation paths coresponding to the nondeterministic choices the program might make. (An example is given in Section 6.) Neither of Lamport's systems is entirely adequate for such applications.

In order to examine these issues more carefully, we define a language, CTL*, in which a universal or existential path quantifier can prefix an arbitrary linear time assertion. CTL* is an extension of the Computation Tree Logic (CTL), defined in [3] and studied in [9]. This language subsumes both of Lamport's interpretations and allows us to compare branching with linear time. Moreover, the syntax of

CTL* makes it clear which interpretation is intended. We then compare several sublanguages of CTL* in expressive power. These sublanguages correspond to ones that have been considered elsewhere in the literature (cf. [2, 3, 7–9, 14]. By making such a comparison, we can better understand which of these languages is most appropriate for reasoning about a given application.

The paper is organized as follows: In Section 2 we summarize Lamport's approach and discuss its limitations. In Section 3 we present the syntax and semantics of CTL*. We also define some natural sublanguages of CTL* and compare their expressive power in Section 4. In particular, we show (cf. Theorem 4.2) that a language substantially less expressive than CTL* still subsumes both of Lamport's interpretations. Section 5 then shows how CTL* can be embedded in MPL [1] and PL [16]. Finally, Section 6 concludes with a comparison of the utility of branching and linear time logics.

2. *A Critique of Lamport's Approach*

For the reader's convenience we summarize Lamport's approach here (we do take the liberty of slightly altering his notation):

2.1 STRUCTURES. A *structure* $\mathbf{M} = (\mathbf{S}, \mathbf{X}, \mathbf{L})$ where

S is a nonempty set of *states*,
X is a nonempty set of *paths* (where a path is a nonempty sequence of states), and
L is a *labeling* that assigns to each state a set of atomic propositions true in the state.

We use $s, t, s', t_1, \ldots$ to denote states in **S** and $x, y, x', y_1, \ldots$ to denote (finite or infinite) sequences of states (with repetitions allowed) over **S**. We think of a state as being the state of a system during the execution of a concurrent program. A path is the sequence of states describing one particular (entire) execution of the program. Thus, **X** represents a set of possible executions of the concurrent program.

Certain restrictions are often placed on the set of paths **X**. In order to describe them, we first need some definitions: A sequence x is of *length* k, written $|x| = k$, if it consists of $1 + k$ states.[1] We use *first*(x) to denote the first state, s_0, of x. If $|x| > 0$, we define $x^{\text{succ}} = (s_1, \ldots, s_k, \ldots)$; otherwise, $x^{\text{succ}} = x$. We define the *suffixes* of x, $x^0 = x$, $x^{m+1} = (x^m)^{\text{succ}}$. If $y \neq x$ is a suffix of x, then y is a *proper suffix* of x. The *prefixes* and *proper prefixes* of x are defined similarly.

Lamport, in particular, requires that **X** be *suffix closed*, that is, if $x \in \mathbf{X}$ then $x^{\text{succ}} \in \mathbf{X}$. (The motivation for this requirement will be discussed subsequently.)

2.2 SYNTAX. Lamport inductively defines the syntax of a class of *temporal formulas*:

(1) Any atomic proposition P is a temporal formula.
(2) If p, q are temporal formulas, then so are $p \wedge q$ ("conjunction") and $\neg p$ ("negation").
(3) If p is a temporal formula then so are $\Box p$ (meaning "always p") and $\rightsquigarrow p$ (meaning "sometimes p").

2.3 SEMANTICS. A temporal formula's meaning depends on whether it is interpreted as a formula of branching time or a formula of linear time. For the branching time interpretation, we write $\mathbf{M}, s \models_{\mathbf{B}} p$ to indicate that formula p is

[1] An infinite sequence $x = s_0, s_1, s_2, \ldots$ is of length ω because it consists of $1 + \omega = \omega$ states.

interpreted as true in structure $\mathbf{M}$ at state s. We define $\models_B$ inductively:

(1) $\mathbf{M}, s \models_B P$ iff $P \in \mathbf{L}(s)$.
(2) $\mathbf{M}, s \models_B p \wedge q$ iff $\mathbf{M}, s \models_B p$ and $\mathbf{M}, s \models_B q$;
$\mathbf{M}, s \models_B \neg p$ iff not $(\mathbf{M}, s \models_B p)$.
(3) $\mathbf{M}, s \models_B \Box p$ iff $\forall$path $x \in \mathbf{X}$ with first$(x) = s$, $\forall n \geq 0$, $\mathbf{M}$, first$(x^n) \models_B p$;
$\mathbf{M}, s \models_B \rightsquigarrow p$ iff $\forall$path $x \in \mathbf{X}$ with first$(x) = s$, $\exists n \geq 0$, $\mathbf{M}$, first$(x^n) \models_B p$.

Similarly, for the linear time interpretation we write $\mathbf{M}, x \models_L p$ to indicate that in structure $\mathbf{M}$ formula p is true of path x. Again, we define $\models_L$ inductively:

(1) $\mathbf{M}, x \models_L P$ iff $P \in \mathbf{L}(\text{first}(x))$.
(2) $\mathbf{M}, x \models_L p \wedge q$ iff $\mathbf{M}, x \models_L p$ and $\mathbf{M}, x \models_L q$;
$\mathbf{M}, x \models_L \neg p$ iff not $(\mathbf{M}, x \models_L p)$.
(3) $\mathbf{M}, x \models_L \Box p$ iff $\forall n \geq 0$, $\mathbf{M}, x^n \models_L p$;
$\mathbf{M}, x \models_L \rightsquigarrow p$ iff $\exists n \geq 0$, $\mathbf{M}, x^n \models_L p$.

For both interpretations, the modality $\Diamond p$ ("not never p") is introduced as an abbreviation for $\neg\Box\neg p$ and the other logical connectives are introduced as abbreviations in the usual way. It can be easily checked that $\rightsquigarrow p$ is equivalent to $\Diamond p$ in the linear time, but not the branching time interpretation. This justifies Lamport's comment that "sometimes" is sometimes (but not always) "not never."

Note that in the branching-time interpretation, a formula is true or false of a state whereas in the linear time interpretation, a formula is true or false of a path. Thus, we cannot directly compare the expressive power of linear time with branching time. In an attempt to overcome this difficulty, Lamport extends $\models_B$ and $\models_L$ to entire models:

Definition 1. Given structure $\mathbf{M} = (\mathbf{S}, \mathbf{X}, \mathbf{L})$ temporal formula p is $\mathbf{M}$-*valid under the branching time interpretation*, written $\mathbf{M} \models_B p$, provided that for every state $s \in \mathbf{S}$, $\mathbf{M}, s \models_B p$. Similarly, p is $\mathbf{M}$-*valid under the linear time interpretation*, written $\mathbf{M} \models_L p$, provided that for every path $x \in \mathbf{X}$, $\mathbf{M}, x \models_L p$.

Next, Lamport defines his notion of equivalence:

Definition 2. Formula p under interpretation I is *strongly equivalent* to formula q under interpretation J (with respect to a class of structures $\mathscr{C}$), written $p \equiv_s^{\mathscr{C}} q$, provided that for every structure $\mathbf{M} \in \mathscr{C}$, $\mathbf{M} \models_I p$ iff $\mathbf{M} \models_J q$. (When $\mathscr{C}$ is understood, we simply write $\equiv_s$.)

Using this formalism, Lamport argues that linear time and branching time have incomparable expressive power:

THEOREM 1 [19]. *$\Diamond P$ in branching time is not strongly equivalent with respect to suffix closed structures to any assertion of linear time.*

THEOREM 2 [19]. *$\rightsquigarrow\Box P$ in linear time is not strongly equivalent with respect to suffix closed structures to any assertion of branching time.*

We now provide our critique of Lamport's approach. Although we do have a few minor criticisms regarding some peculiar technical features and limitations of Lamport's formalism, we would like to emphasize, before we begin, that Lamport's formal results are technically correct—that is, they follow via sound mathematical arguments from his definitions. Our main criticisms instead center around

(1) Lamport's basic definitions and underlying assumptions, and
(2) the informal conclusions regarding the application of temporal logic to reasoning about concurrent programs that Lamport infers from his technical results.

First, to make the technical results comparing expressive power relevant to concurrent programming, the underlying assumptions about the semantics of a concurrent program as given by the structure should reflect essential properties of genuine concurrent programs. Lamport correctly observes that "the future behavior [of a concurrent program] depends only upon the current state, and not upon how that state was reached." With this motivation, Lamport requires that the set of paths **X** be *suffix closed*, that is, if $x \in \mathbf{X}$ then $x^{\text{succ}} \in \mathbf{X}$. As observed in [5], however, suffix closure is not sufficient to guarantee that a program's future behavior depends only on its current state. We also need to require that, at least, **X** be *fusion closed* (cf. [26]), meaning that if $x_1sy_1, x_2sy_2 \in \mathbf{X}$ then $x_1sy_2 \in \mathbf{X}$.[2] Moreover, there are some additional properties that the set **X** of all computations of a concurrent program can be expected to satisfy. We say that **X** is *limit closed* (cf. [1]) if whenever each of the infinite sequence of paths $x_1y_1, x_1x_2y_2, x_1x_2x_3y_3, \ldots$ is in **X**, then the infinite path $x_1x_2x_3 \cdots$ (which is the "limit" of the prefixes $(x_1, x_1x_2, x_1x_2x_3, \ldots)$) is also in **X**. We say that a set **X** of paths is **R**-*generable* iff there exists a total, binary relation **R** on **S** such that **X** consists precisely of the infinite sequences $(s_0, s_1, s_2, \ldots)$ of states from **X** for which $(s_i, s_{i+1}) \in \mathbf{R}$. **R**-generability corresponds to the common approach (cf. [2, 7, 9, 13, 21]) of describing a concurrent program via a transition relation (**R**) on states. It is also convenient for many applications to assume that every path in **X** is infinite, thus identifying a finite execution $(s_0, \ldots, s_k)$ with the infinite execution $(s_0, \ldots, s_k, s_k, s_k, \ldots)$. (Note: This is essentially what is done by our successor operation on paths.) As shown in [5], a set **X** of infinite paths is **R**-generable iff it is suffix closed, fusion closed, and limit closed. We say that a structure $\mathbf{M} = (\mathbf{S}, \mathbf{X}, \mathbf{L})$ is suffix closed (respectively, fusion closed, limit closed, **R**-generable) exactly if the set of paths **X** is. Finally, a structure **M** is *state complete* if, for all $s \in \mathbf{S}$, there is some path $x \in \mathbf{X}$ whose first state is s.

As the above remarks indicate, the expressiveness results should be developed with respect to either **R**-generable structures or structures that are both suffix closed and fusion closed. However, Lamport's proof of Theorem 1 only applies to structures that are suffix closed but not fusion closed, while his proof of Theorem 2 only applies to structures that are suffix closed but not limit closed (and hence not **R**-generable). Nonetheless, as we show in Section 4, Theorems 1 and 2 do extend to the more relevant types of semantics.

We also have a technical objection concerning the notion of strong equivalence. By only considering the truth of a formula in a (whole) model, rather than at a state or path, a great deal of information is lost. For example, in the branching time interpretation, although there is a model **M** with state s such that $\mathbf{M}, s \models_B \rightsquigarrow P \wedge \neg P$, there is no model **M** such that $\mathbf{M} \models_B \rightsquigarrow P \wedge \neg P$. Similar remarks apply for the linear time interpretation. Thus, we get

PROPOSITION 1. *In linear time or in branching time,* $\rightsquigarrow P \wedge \neg P \equiv_s$ **false**.

This suggests that $\equiv_s$ is too coarse an equivalence relation for properly comparing the expressive powers of logics, in that it classifies satisfiable formulas as equivalent to false.

We are further concerned that, since the same notation is used for both branching and linear time formulas, it is not clear from the syntax which interpretation is intended. This has the effect of obscuring an essential difference between the two

[2] Roughly speaking, when **X** is fusion closed, if xs is a prefix of some execution of the program and sy is a suffix of another execution, then xsy is also an execution.

interpretations, namely, that linear time formulas make assertions about paths and branching time formulas make assertions about states. It also causes difficulties when translating from English into the formalism.

Our chief disagreement is, of course, with Lamport's conclusion that linear time logic is superior to branching time logic for reasoning about concurrent programs. We do not think that sweeping conclusions regarding branching versus linear time logics in general are justified merely on the basis of the comparison of the two particular systems that Lamport considers. Indeed, Lamport gives two specific arguments to justify his conclusion:

(1) To establish certain liveness properties of a concurrent program, it is frequently necessary to appeal to some sort of fair scheduling constraint such as strong eventual fairness (which means that if a process is enabled for execution infinitely often, then eventually the process must actually be executed). This constraint can be expressed in linear time logic by the formula $(\rightsquigarrow \Box \neg \mathit{enabled}) \vee \rightsquigarrow \mathit{executed}$. However, by Theorem 2, it is not expressible in branching time logic.

(2) In proving a program correct, it is often helpful to reason using the principle that, along any path, either property P is eventually true or is always false. This amounts to assuming an axiom of the form $\rightsquigarrow P \vee \Box\neg P$, which is **M**-valid for all models **M** under the linear time interpretation, but not under the branching time interpretation.

The first observation is certainly true for the particular systems that Lamport has defined. However, by using a branching-time logic with a richer set of modalities (for example, by allowing the "infinitary" quantifiers used in [7]), these assertions can be easily expressed. Indeed, by adding enough modalities to a branching time logic, *any* assertion of Lamport's linear time can be expressed as described in Theorem 4. In regard to the second point, it is true that the given formula is valid (i.e., true in all models) under the linear time interpretation but not under the branching time interpretation. However, the formula is not a correct translation of the principle into the formalism under the branching time interpretation. (We believe that this is an instance of the confusion caused by the use of the same syntax for both interpretations.) Again, as shown in Section 3, it is possible to write a formula in a branching time system that accurately renders the principle.

3. *A Unified Approach*

In this section we exhibit a uniform formalism for comparing branching with linear time that avoids the technical difficulties of Lamport's and allows us to examine the issues more closely. To illustrate our approach, we describe a language, CTL*, that subsumes Lamport's branching and linear time systems, as well as UB [2] and CTL [3, 9]. CTL* is closely related to MPL [1]. (CTL* is also used in [4].) In CTL* we allow a path quantifier, either A ("for all paths") or E ("for some paths"), to prefix an assertion p composed of arbitrary combinations of the usual linear time operators G ("always"), F ("sometimes"), X ("nexttime"), U ("until"), as well as the infinitary state quantifiers of [7], $\overset{\infty}{F}$ ("infinitely often"), $\overset{\infty}{G}$ ("almost everywhere").

3.1 SYNTAX. We inductively define a class of state formulas (true or false of states) and path formulas (true or false of paths):

S1. Any atomic proposition P is a state formula.
S2. If p, q are state formulas, then so are $p \wedge q$, $\neg p$.

S3. If p is a path formula, then Ap, Ep are state formulas.
P1. Any state formula p is a path formula.
P2. If p, q are path formulas, then so are $p \wedge q$, $\neg p$.
P3a. If p is a state formula, then Fp is a path formula.
P3b. If p is a path formula, then Fp is a path formula.
P4a. If p is a state formula, then Xp is a path formula.
P4b. If p is a path formula, then Xp is a path formula.
P5a. If p, q are state formulas, then $(p\ U\ q)$ is a path formula.
P5b. If p, q are path formulas, then $(p\ U\ q)$ is a path formula.
P6a. If p is a state formula, then $\overset{\infty}{F}p$ is a path formula.
P6b. If p is a path formula, then $\overset{\infty}{F}p$ is a path formula.

Remark. The other truth-functional connectives are introduced as abbreviations in the usual way. We also let Gp abbreviate $\neg F \neg p$ and $\overset{\infty}{G}p$ abbreviate $\neg \overset{\infty}{F} \neg p$. In addition, we could take the view that Ap abbreviates $\neg E \neg p$, Fp abbreviates (**true** $U\ p$), and $\overset{\infty}{F}p$ abbreviates GFp. Thus, we could give a substantially more terse syntax and semantics for our language by defining all the other operators in terms of just the primitive operators E, X, U, $\neg$, and $\wedge$. We could also consider state formulas as a special case of path formulas whose truth value depends on the first state of the path and thus view all formulas as path formulas. This is essentially what is done in PL (cf. [16]) and also leads to a slightly easier formulation of the syntax and semantics. However, like Abrahamson [1], we consider the distinction between quantification over states and over paths an important one that should be maintained. Moreover, this approach makes it easier to give the syntax of each of the sublanguages that we consider.

The set of state formulas generated by all the above rules forms the language CTL*. Since we are also concerned with the power of the various modalities, we also want to consider various sublanguages of CTL*. The set of path formulas generated by rules S1, P1, 2, 3b gives the (linear time) language L(F). By adding rule 5b we get the language L(F, U) while adding both rules 4b, 5b gives us L(F, X, U). We get the (branching time) language B(F) by considering the state formulas generated by rules S1–3, P3a. Adding rules P4a, P5a gives B(F, X, U), adding rules P2, P4a, P5a gives B(F, X, U, $\wedge$, $\neg$), and adding rules P2, P4a, P5a, P6a gives B(F, X, U, $\overset{\infty}{F}$, $\wedge$, $\neg$). Thus, for example, $A[FP \wedge GQ]$ is a formula of B(F, X, U, $\wedge$, $\neg$) but not of B(F, X, U).

The languages defined here include a number of the linear time and branching time logics considered in the literature. As we shall see, L(F) and B(F) correspond precisely to Lamport's linear time interpretation and branching time interpretation, respectively. L(F, X, U) has been used in many applications (cf. [14, 22]); in [30] the same language is called L(F, G, X, U) (we have omitted the G here for reasons of brevity). B(F, X, U) is the logic CTL of [3], [8], and [9], while B(F, X, U, $\overset{\infty}{F}$, $\wedge$, $\neg$) is essentially the language studied in [7] for describing fairness properties.

We use $|p|$ to denote the *length* of formula p, that is, the number of symbols in p viewed as a string over the set of atomic propositions union the set of connectives ($\wedge$, $\neg$, A, E, F, $($, $)$, etc.).

3.2 SEMANTICS. We write $\mathbf{M}, s \models p$ ($\mathbf{M}, x \models p$) to mean that state formula p (path formula p) is true in structure $\mathbf{M}$ at state s (of path x, respectively).[3] When

[3] Note that $\mathbf{M}$ may be an arbitrary structure, and that we define the semantics of path formulas relative to *all* paths over the states of $\mathbf{M}$, not just the paths in $\mathbf{X}$; it is the path quantifiers, A or E, of the state formulas that restrict quantification to paths in $\mathbf{X}$.

M is understood, we write simply $s \models p$ ($x \models p$). We define $\models$ inductively:

S1. $s \models P$ iff $P \in \mathbf{L}(s)$ where P is an atomic proposition.
S2. $s \models p \wedge q$ iff $s \models p$ and $s \models q$ where p, q are state formulas.
$s \models \neg p$ iff not $(s \models p)$ where p is a state formula.
S3. $s \models Ap$ iff for every path $x \in \mathbf{X}$ with first$(x) = s$, $x \models p$ where p is a path formula.
$s \models Ep$ iff for some path $x \in \mathbf{X}$ with first$(x) = s$, $x \models p$ where p is a path formula.
P1. $x \models p$ iff first$(px) \models p$ where p is a state formula.
P2. $x \models p \wedge q$ iff $x \models p$ and $x \models q$ where p, q are path formulas.
$x \models \neg p$ iff not $(x \models p)$ where p is a path formula.
P3a. $x \models Fp$ iff for some $i \geq 0$, first$(x^i) \models p$ where p is a state formula.
P3b. $x \models Fp$ iff for some $i \geq 0$, $x^i \models p$ where p is a path formula.
P4a. $x \models Xp$ iff $|x| \geq 1$ and first$(x^1) \models p$ where p is a state formula.
P4b. $x \models Xp$ iff $|x| \geq 1$ and $x^1 \models p$ where p is a path formula.
P5a. $x \models (p\ U\ q)$ iff for some $i \geq 0$, first$(x^i) \models q$ and for all $j \geq 0$ [$j < i$ implies first$(x^j) \models p$].
P5b. $x \models (p\ U\ q)$ iff for some $i \geq 0$, $x^i \models q$ and for all $j \geq 0$ [$j < i$ implies $x^j \models p$].
P6a. $x \models \overset{\infty}{F} p$ iff for infinitely many distinct i, first$(x^i) \models p$ where p is a state formula.
P6b. $x \models \overset{\infty}{F} p$ iff for infinitely many distinct i, $x^i \models p$ where p is a path formula.

Remarks

(1) It is easy to check that all the equivalences mentioned in the remark in Section 3.1 hold.

(2) The notions of **M**-validity and strong equivalence (defined in Definitions 1 and 2, respectively) generalize to apply to arbitrary state and path formulas. □

As mentioned above, Lamport's linear time and branching time formalisms correspond to L(F) and B(F), respectively. Let p be a temporal formula as defined in Section 2.2. Let p^{L} be the path formula that results from replacing each $\Box$ in p by G, and each $\rightsquigarrow$ by F. Let p^{B} be the state formula that results from replacing each $\Box$ in p by AG, and each $\rightsquigarrow$ in by AF. Clearly, p^{L} is an L(F) formula and p^{B} is a B(F) formula. Moreover, each L(F) (respectively, B(F)) formula corresponds to a unique temporal formula via this translation. We then have

PROPOSITION 2

$$\mathbf{M}, s \models_B p \quad \textit{iff} \quad \mathbf{M}, s \models p^B;$$
$$\mathbf{M}, x \models_L p \quad \textit{iff} \quad \mathbf{M}, x \models p^L.$$

Note that under the linear time interpretation the formula discussed in the previous section, $\rightsquigarrow P \vee \Box\neg P$, corresponds to the L($F$) formula $FP \vee G\neg P$, which is clearly valid. Under the branching time interpretation, it corresponds to $AFP \vee AG\neg P$, which is not valid. However, the valid B(F, X, U, $\wedge$, $\neg$) formula $A(FP \vee G\neg P)$ (obtained by simply prefixing the L(F) formula with A) does capture the intended principle.

Clearly, a direct comparison of linear time (i.e., path) formulas with branching time (i.e., state) formulas is impossible. As we have seen, Lamport's approach of defining a formula as true or false of an entire structure gives us too coarse an equivalence relation. How then can we compare linear time with branching time? Since in program verification applications there is an implicit universal quantifi-

cation over all possible futures when a linear time assertion is used, this suggests that we associate with every path formula p the state formula Ap and ask whether this is expressible in a given branching time logic. Thus, we have the following definition:

Definition 3. Given any language L of path formulas, we define the language of associated state formulas $\mathrm{B}(L) = \{Ap : p \in L\}$. (Note that $\mathrm{B}(L)$ is not closed under semantic negation or disjunction (cf. [1]).)

On this basis we can compare any linear time logic L with branching time logic B by first converting L into the associated branching time logic $\mathrm{B}(L)$. This time, however, equivalence of the branching time formulas is measured by the "usual" notion:

Definition 4. Given state formulas p, q we say that p is *equivalent* to q, with respect to a class $\mathscr{C}$ of structures written $p \equiv^{\mathscr{C}} q$, provided that for every structure $\mathbf{M}$ in $\mathscr{C}$, for every state s of $\mathbf{M}$, $\mathbf{M}, s \models p$ iff $\mathbf{M}, s \models q$. When $\mathscr{C}$ is clear from context, we simply write $\equiv$.

It is easy to check that $\equiv$ is an equivalence relation on state formulas that refines $\equiv_s$ and avoids the problems of Proposition 1. In fact, we have the following results which clarify the relation between $\equiv$ and $\equiv_s$. Let $\mathscr{F}$ be the class of all structures that are both fusion closed and state complete.

PROPOSITION 3. *For any path formula p, $p \equiv_s Ap$.*

PROOF. Let $\mathbf{M} = (\mathbf{S}, \mathbf{X}, \mathbf{L})$ be an arbitrary structure. We show $\mathbf{M} \models p$ iff $\mathbf{M} \models Ap$. If $\mathbf{M} \models p$, then for all $x \in \mathbf{X}$, $\mathbf{M}, x \models p$. So for all $s \in \mathbf{S}$, $\mathbf{M}, s \models Ap$ and thus $\mathbf{M} \models Ap$. Conversely, if $\mathbf{M} \models Ap$, then for all $s \in \mathbf{S}$, $\mathbf{M}, s \models Ap$ and for all $x \in \mathbf{X}$ starting at s, $\mathbf{M}, x \models p$. Since each $x \in \mathbf{X}$ starts at some $s \in \mathbf{S}$, $\mathbf{M}, x \models p$ for all $x \in \mathbf{X}$. Thus, $\mathbf{M} \models p$. □

PROPOSITION 4. *For any state formulas p, q, $p \equiv_s^{\mathscr{F}} q$ iff $AGp \equiv^{\mathscr{F}} AGq$.*

PROOF

($\Rightarrow$:) Assume $p \equiv_s^{\mathscr{F}} q$. It will suffice to show that $\mathbf{M}, s \models AGp$ implies $\mathbf{M}, s \models AGq$ because, by a symmetric argument, we can then conclude $AGp \equiv^{\mathscr{F}} AGq$. So suppose $\mathbf{M}, s \models AGp$, where $\mathbf{M} = (\mathbf{S}, \mathbf{X}, \mathbf{L})$ is an arbitrary fusion closed structure and $s \in \mathbf{S}$. Define $\mathbf{M}' = (\mathbf{S}', \mathbf{X}', \mathbf{L}')$ where $\mathbf{S}' = \{s' \in \mathbf{S} : s' \text{ appears on some } x' \in \mathbf{X} \text{ with first}(x') = s\}$, $\mathbf{X}' = \{x' \in \mathbf{X} : \text{first}(x') \in \mathbf{S}'\}$, and $\mathbf{L}' = \mathbf{L} \mid \mathbf{S}'$. Since $\mathbf{X}$ is fusion closed $\{s'' \in \mathbf{S} : s'' \text{ appears on some } x' \in \mathbf{X}'\} = \mathbf{S}'$ and $\mathbf{M}'$ is thus a structure. Observe that for any state formula r, $\mathbf{M}, s \models AGr$ iff $\mathbf{M}', s \models AGr$ iff $\forall s' \in \mathbf{S}'$, $(\mathbf{M}', s' \models r)$. Taking $r = p$, we get $\forall s' \in \mathbf{S}'$, $\mathbf{M}', s' \models p$. Since $p \equiv_s^{\mathscr{F}} q$, $\forall s' \in \mathbf{S}'$, we have $\mathbf{M}', s' \models q$. Now take $r = q$, to see that $\mathbf{M}, s \models AGq$ as desired.

($\Leftarrow$:) Assume $AGp \equiv^{\mathscr{F}} AGq$, that is, $\mathbf{M}, s \models AGp$ iff $\mathbf{M}, s \models AGq$ for all $\mathbf{M}$ and s in $\mathbf{M}$. It will suffice to show that $\mathbf{M} \models p$ implies $\mathbf{M} \models q$, as a symmetric argument will yield $p \equiv_s^{\mathscr{F}} q$. Now suppose $\mathbf{M} \models p$, where $\mathbf{M} = (\mathbf{S}, \mathbf{X}, \mathbf{L})$. Then, $\forall s \in S$, we have $\mathbf{M}, s \models p$ whence, $\forall s \in \mathbf{S}$, we also have $\mathbf{M}, s \models AGp$. Since $AGp \equiv^{\mathscr{F}} AGq$, $\forall s \in \mathbf{S}$, $\mathbf{M}, s \models AGq$. Since $\mathbf{M}$ is state complete, $\forall s \in \mathbf{S}$, we have $\mathbf{M}, s \models q$. Thus $\mathbf{M} \models q$ as desired. □

Remark. Both fusion closure and state completeness are needed for the previous result. Considering the formulas $p = P \wedge EFEX\neg P$ and $q = \mathbf{false}$, we see that, while $p \equiv_s q$, we also have $AGp \not\equiv AGq$ if we allow structures that are not fusion

closed. Similarly, if we take $p = AGEF\mathbf{true} \wedge EF\mathbf{true}$ and $q = [AGEF\mathbf{true} \wedge EF\mathbf{true}] \vee AG\mathbf{false}$, we have $AGp \equiv AGq$, but also $p \not\equiv_s q$ if we allow structures that are not state complete.

COROLLARY 1. *For any path formula p and state formula q, $p \equiv_s^{\mathscr{F}} q$ iff $AGAp \equiv^{\mathscr{F}} AGq$.*

Finally, we compare the expressive power of two branching time languages as follows:

Definition 5. As measured with respect to a class of structures $\mathscr{C}$, we say that L_2 is *at least as expressive as* L_1, written $L_1 \leq^{\mathscr{C}} L_2$, provided that for every $p \in L_1$ there exists $q \in L_2$ such that $p \equiv^{\mathscr{C}} q$. We say that L_1 is *exactly as expressive as* L_2, written $L_1 \equiv^{\mathscr{C}} L_2$, provided $L_1 \leq^{\mathscr{C}} L_2$ and $L_2 \leq^{\mathscr{C}} L_1$. Finally, L_1 is *strictly less expressive than* L_2, written $L_1 <^{\mathscr{C}} L_2$, provided $L_1 \leq^{\mathscr{C}} L_2$ and $L_1 \not\equiv^{\mathscr{C}} L_2$. (When it is clear from the context, the superscript $\mathscr{C}$ is omitted.)

4. *Expressiveness Results*

We now compare linear time and branching time logics using the formalism of the previous section. In line with our criticism of Lamport's underlying semantic assumptions, our comparison is done with respect to the class of **R**-generable structures. (We leave it to the reader to check that all our proofs hold without change for the class of suffix closed and fusion closed structures as well.) In this framework, we reprove Lamport's incomparability results establishing that the branching time logic B(F) and the linear time logic B($L(F)$) are incomparable in expressive power. We also prove that B($L(F)$) $<$ B(F, X, U, $\tilde{F}$, $\wedge$, $\neg$), thus demonstrating that branching time logic can be more expressive than a linear time logic.[4] Moreover, this result is essentially tight. If we add X and U (actually either X or U alone suffices) to the linear time logic, the resulting language is incomparable in expressive power to B(F, X, U, $\tilde{F}$, $\wedge$, $\neg$). Indeed, we show that Figure 1 completely describes the relative expressive power of the languages we have considered, where any two languages not connected by a chain of $<$'s and $\equiv$'s are of incomparable expressive power. Such expressiveness results provide important information when choosing an appropriate logic for reasoning about a particular application.

We first show that no linear time language (i.e., one of the form B($L(-)$)) can be more expressive than even B(F). This reproves Lamport's result (Theorem 1) in the **R**-generable framework.

THEOREM 3. *The B(F) formula EFP is not equivalent to any B($L(F, X, U)$) formula.*

PROOF. Suppose $EFP \equiv Aq$ for some linear time formula q over F, X, U. Consider the **R**-generable structure $\mathbf{M} = (\mathbf{S}, \mathbf{X}, \mathbf{L})$ where $\mathbf{S} = \{s_1, s_2\}$, $\mathbf{X}$ is the set of paths generated by $\mathbf{R} = \{(s_1, s_1), (s_1, s_2), (s_2, s_2)\}$, and $\mathbf{L}$ is such that $\mathbf{M}, s_1 \models \neg P$ and $\mathbf{M}, s_2 \models P$. The **R**-generable "substructure" obtained by restricting $\mathbf{M}$ to $\mathbf{S}' = \{s_1\}$ is defined as $\mathbf{M}' = (\mathbf{S}', \mathbf{X}', \mathbf{L}')$ where $\mathbf{X}'$ is the set of paths generated by $\{(s_1, s_1)\}$ and $\mathbf{L}'(s_1) = \mathbf{L}(s_1)$. Plainly, $\mathbf{M}, s_1 \models EFP$ and $\mathbf{M}', s_1 \models \neg EFP$. By the supposed equivalence of EFP and Aq, we have $\mathbf{M}, s_1 \models Aq$. But then $\mathbf{M}', s_1 \models Aq$ because every path starting at s_1 in $\mathbf{M}'$ is also a path starting at s_1 in $\mathbf{M}$. Again using the supposed equivalence we get $\mathbf{M}', s_1 \models EFP$, a contradiction. □

[4] Because we are working in the context of **R**-generable structures, throughout this section we write just $\equiv$ for $\equiv^{\mathscr{R}}$ and $<$ for $<^{\mathscr{R}}$.

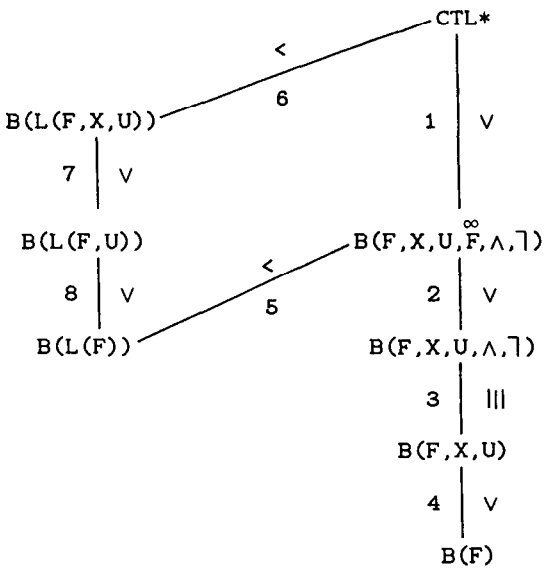


FIGURE 1

Our next result shows that Lamport's linear time system B($L(F)$) is expressible in a branching time logic:

THEOREM 4. $B(L(F)) \leq B(F, X, U, \overset{\infty}{F}, \wedge, \neg)$.

PROOF. This proof involves a complicated induction on the structure of B($L(F)$) formulas. Details are left to the Appendix. □

However, if we add either the nexttime operator or the until operator, the situation changes:

THEOREM 5. *The formula* $A[F(P \wedge XP)]$ *is not equivalent to any* $B(L(F, X, U, \overset{\infty}{F}, \wedge, \neg))$ *formula.*

PROOF. We inductively define two sequences $\mathbf{M}_1, \mathbf{M}_2, \mathbf{M}_3, \ldots$ and $\mathbf{N}_1, \mathbf{N}_2, \mathbf{N}_3, \ldots$ of models as follows. Define $\mathbf{M}_1$, $\mathbf{N}_1$ to have the graphs shown in Figure 2 where in $\mathbf{M}_1$, $a_1 \models P$, $b_1 \models P$, $d_1 \models \neg P$ and in $\mathbf{N}_1$, $a_1 \models P$, and $d_1 \models \neg P$.

Suppose we have defined $\mathbf{M}_i$ and $\mathbf{N}_i$. Then $\mathbf{M}_{i+1}$ and $\mathbf{N}_{i+1}$ have the graphs shown in Figure 3, where in both $\mathbf{M}_{i+1}$ and $\mathbf{N}_{i+1}$, $a_{i+1} \models P$, $b_{i+1} \models \neg P$, and $\mathbf{M}_i'$, $\mathbf{N}_i'$ are copies of $\mathbf{M}_i$, $\mathbf{N}_i$, respectively.

It should be clear that

(1) for all i, $\mathbf{M}_i$, $a_i \models A[F(P \wedge XP)]$ and $\mathbf{N}_i$, $a_i \models \neg A[F(P \wedge XP)]$.

We will also show that

(2) For any B(F, X, U, $\overset{\infty}{F}$, $\wedge$, $\neg$) formula p there is a B(F, X, U) formula q that is equivalent to p over these two sequences of models. That is, for all i and all states s in $\mathbf{M}_i$,

$$\mathbf{M}_i, s \models p \equiv q, \qquad \text{and similarly for } \mathbf{N}_i.$$

(3) For any B(F, X, U) formula p, if $|p| \leq i$, then $\mathbf{M}_i$, $a_i \models p$ iff $\mathbf{N}_i$, $a_i \models p$.

To see that the result follows, suppose that $A[F(P \wedge XP)]$ is equivalent to some B(F, X, U, $\overset{\infty}{F}$, $\wedge$, $\neg$) formula p. Then by (2) above, there is a B(F, X, U) formula p' equivalent to p over these models. Now $|p'| = i$ for some i. Then $\mathbf{M}_i$, $a_i \models$

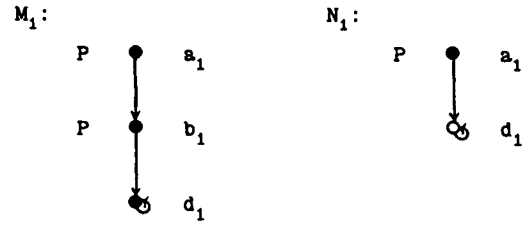


FIGURE 2

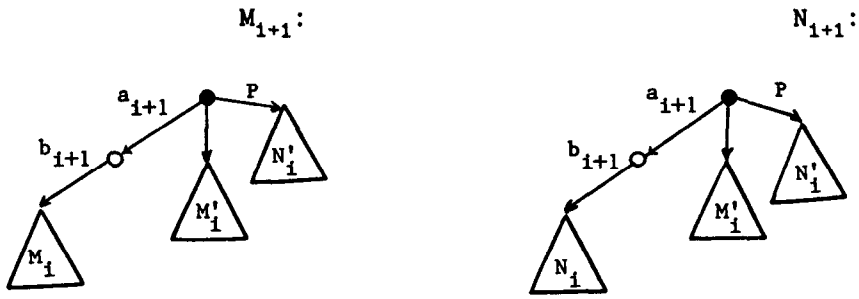


FIGURE 3

$A[F(P \wedge XP)]$, which, by supposition and (2), implies $\mathbf{M}_i, a_i \vDash p'$. By (3) this implies $\mathbf{N}_i, a_i \vDash p'$, which implies, again by supposition and (2), that $\mathbf{N}_i, a_i \vDash A[F(P \wedge XP)]$. But this contradicts the fact (1) above that $N_i, a_i \vDash \neg A[F(P \wedge XP)]$.

The details of the proof for (2) and (3) are provided in the Appendix. □

THEOREM 6. *The formula* $E[((P_1 \; U \; P_2) \vee (Q_1 \; U \; Q_2)) \; U \; R]$ *is not equivalent to any* $B(L(F, X, U, \overset{\infty}{F}, \wedge, \neg))$ *formula.*

PROOF. Left to the Appendix. □

Similar combinatorial techniques can also be used to prove the following theorem:

THEOREM 7. *The formula* $E\overset{\infty}{F}P$ *is not equivalent to any* $B(F, X, U, \wedge, \neg)$ *formula.*

PROOF. Left to the Appendix. □

Since $\rightsquigarrow\Box P$ in linear time corresponds to $A\overset{\infty}{G}P$, the dual of $E\overset{\infty}{F}P$, Theorem 7 reproves in the **R**-generable framework Lamport's result, Theorem 2. We further note that Theorem 7 also follows from the results of [7], which depend on recursion-theoretic techniques. However, the techniques there do not suffice to establish, for example, Theorem 6. Thus, the combinatorial proof techniques used here seem to provide a sharper tool than does recursion theory in applications such as this. (In [17], similar combinatorial techniques are used to differentiate between PDL-like languages (cf. [13]); they independently established as a corollary that $B(F, X, U) < \text{CTL}^*$.)

We also note the following results:

THEOREM 8 [9]

(*a*) $B(F, X, U) \equiv B(F, X, U, \wedge, \neg)$.
(*b*) *$E(P\ U\ Q)$ is not equivalent to any $B(F)$ formula.*

THEOREM 9 [16]

(*a*) *$(P\ U\ Q)$ is not equivalent to any $L(F)$ formula.*
(*b*) *XP is not equivalent to any $L(F, U)$ formula.*

We now explain how the expressiveness results indicated in the diagram follow from the Theorems above. That it is appropriate to label each of the arcs, except for arc 6, with (at least) $\leq$ follows directly from syntactic containment. For example, $B(F, X, U)$ is a syntactic sublanguage of $B(F, X, U, \wedge, \neg)$. Theorem 4 establishes that it is appropriate to label arc 6 with (at least) $\leq$. The label of arc 3 can be strengthened to be $\equiv$ by applying Theorem 8. The strengthening of the rest of the arcs to $<$ is accounted for in this way: arc 1 follows from Theorem 5, arc 2 from Theorem 7, arc 4 from Theorem 8b, arcs 5 and 6 from Theorem 3, arc 7 from Theorem 9b, and arc 8 from Theorem 9a.

It remains to justify the absence of any additional arcs connecting a pair of languages in the diagram. By Theorem 3, there is no $\leq$-arc from any branching time language to any linear time language. By Theorem 7 the dual of $E\overset{\infty}{F}P$, which is expressible as $AFGP$ in $B(L(F))$, is not expressible in any of the B(–) languages below and including $B(F, X, U, \wedge, \neg)$ in the diagram. This establishes that $B(L(F))$ is incomparable in expressive power to each of $B(F, X, U, \wedge, \neg)$, $B(F, X, U)$, and $B(F)$. To see that $B(L(F, X, U))$ and $B(L(F, U))$ are each incomparable to each of the logics $B(F, X, U, \overset{\infty}{F}, \wedge, \neg)$, $B(F, X, U, \wedge \neg)$, $B(F, X, U)$, and $B(F)$, we note that the dual of $E[((P_1\ U\ P_2) \vee (Q_1\ U\ Q_2))\ U\ R]$ is expressible in $B(L(F, X, U))$, but by Theorem 7 is not expressible in $B(F, X, U, \overset{\infty}{F}, \wedge, \neg)$ (or any logic below it in the diagram).

Remark. Although we have focused on a few important sublanguages of CTL*, it is easy to define a host of others sublanguages, such as $B(F, U)$, $B(F, U, \wedge, \neg)$, $B(F, U, \overset{\infty}{F}, \wedge, \neg)$, and $B(F, X, U, \overset{\infty}{F})$, in an analogous fashion. Using techniques similar to those presented here, we can get a complete taxonomy of linear time and branching time logics. Since the proofs of these results are quite similar to a number of the proofs we have already described, we just briefly mention a number of them here, omitting details.

It is easy to show that the formula AXP is not expressible in any branching time language that does not have an explicit X operator. Putting this fact together with Theorem 7, we can see that $B(F, U, \wedge, \neg) < B(F, U, \overset{\infty}{F}, \wedge, \neg) < B(F, X, U, \overset{\infty}{F}, \wedge, \neg)$, whereas $B(F, X, \wedge, \neg)$ and $B(F, U, \overset{\infty}{F}, \wedge, \neg)$ are incomparable; combining it with Theorem 8b, we get that $B(F) < B(F, U) < B(F, X, U)$. Using techniques similar to Theorem 6, we can also show that the formula $E(\overset{\infty}{F}P \wedge \overset{\infty}{F}Q)$ is not equivalent to any $B(F, X, U, \overset{\infty}{F})$ formula (this is actually proved in [10]). From this fact we immediately get that $B(F, X, U, \overset{\infty}{F}) < B(F, X, U, \overset{\infty}{F}, \wedge, \neg)$ and that $B(F, U, \overset{\infty}{F}) < B(F, U, \overset{\infty}{F}, \wedge, \neg)$. Note that our proof of Theorem 4 actually shows that $B(L(F)) < B(F, U, \overset{\infty}{F}, \wedge, \neg)$ (since we did not use the X operator anywhere in the proof). Finally, techniques similar to those used in [9] to show $B(F, X, U) \equiv B(F, X, U, \wedge \neg)$ can also be used to show $B(F, U) \equiv B(F, U, \wedge \neg)$. □

What do these expressiveness results say about branching versus linear time for reasoning about concurrency? First, any assertion made in the usual linear time logic L(F, X, U) can be trivially formulated in CTL*. We also note that for each assertion in Lamport's restricted linear time language L(F) there is a corresponding assertion in the restricted branching time language B(F, X, U, $\overset{\infty}{F}$, $\wedge$, $\neg$). In particular, Lamport's linear time assertion about fairness $\rightsquigarrow\Box\neg$*enabled* $\vee \rightsquigarrow$*executed* directly corresponds to the B(F, X, U, $\overset{\infty}{F}$, $\wedge$, $\neg$) assertion $A(\overset{\infty}{G}\neg$*enabled* $\vee$ *Fexecuted*). We further note that the linear time assertion $\Diamond P \vee \Box\neg P$ not only (as mentioned before) corresponds to the B(F, X, U, $\wedge$, $\neg$) assertion $A(FP \vee G\neg P)$, but also can be expressed in B(F, X, U) as $\neg E[\neg P\ U\ (P \wedge EGP)]$ ($\equiv \neg E[G\neg P \wedge FP] \equiv \neg E\neg(FP \vee G\neg P) \equiv A(FP \vee G\neg P)$).

Although the B(F, X, U) formulation is less transparent, it does show that, by adding only some additional operators (U and X; actually just U is needed) to Lamport's branching time system B(F) to get B(F, X, U), the principle in question can in fact be expressed. Further consequences of the expressiveness results are explored in Section 6.

5. *Relation to PL, MPL, and TC*

We assume that the reader is familiar with PL (the reader unfamiliar with PL will find a brief sketch of the syntax and semantics of PL in the Appendix). We can translate CTL* into PL in the following way: To each CTL* structure $\mathbf{M} = (\mathbf{S}, \mathbf{X}, \mathbf{L})$, we associate the PL structure $\mathbf{M}' = (\mathbf{S}, \models, \mathbf{R})$, where the set of paths of atomic program A, $\mathbf{R}_A$, is equal to $\mathbf{X}$, and for any atomic proposition P, $\mathbf{M}', s \models P$ iff $\mathbf{M}, s \models P$. We can then give a translation of a CTL* formula p into an equivalent PL formula p'. We define the translation inductively, taking the primitive temporal connectives of CTL* to be E, X, and U (cf. the remark in Section 3.1):

$P' = P$ for atomic propositions P.
$(\neg p)' = \neg(p')$.
$(p \wedge q)' = p' \wedge q'$.
$(p\ U\ q)' = q' \vee (p'$ **suf** $q')$.
$(Ep)' = \mathbf{f}(\langle A\rangle p')$.
$(Xp)' =$ **false suf** p'. (Note: This is equivalent to the PL formula $\mathbf{n}p'$.)

Then by a straightforward induction on the structure of CTL* formulas we can show

PROPOSITION 5. *For all $x \in \mathbf{X}$, and all path formulas p, $\mathbf{M}, x \models p$ iff $\mathbf{M}', x \models p'$ and for all $s \in \mathbf{S}$, and all state formulas q, $\mathbf{M}, s \models q$ iff $\mathbf{M}', (s) \models q'$.*

Note $(p\ U\ q)' \equiv q' \vee (p'$ **suf** $q')$ since the U operator considers the current path while the **suf** operator only depends on proper suffixes. Ep is a state formula; since in PL we have only path formulas, we force the truth of the formula to depend only on paths starting at the first state.

Since MPL has not been widely discussed in the literature, we briefly review its syntax and semantics here before describing the translation from CTL* into MPL (see [1] for more details). To simplify the exposition, we take the liberty of slightly altering Abrahamson's notation. In particular, we use the temporal connectives $\Diamond$, U, and X instead of their duals $\Box$, W, and Y, respectively. We also omit the H operator and view all paths as simply infinite sequences of states corresponding to legal sequences of transitions since blocking will not concern us here.

The syntax of MPL is as follows:

(1) Any atomic proposition is a formula.
(2) If p, q are formulas, then so are $\neg p$, $p \wedge q$, $\Diamond p$, Xp, and $p\ U\ q$.

We take $\Box p$ to be an abbreviation for $\neg\Diamond\neg p$.

A structure $\mathbf{M}$ is a triple $(\mathbf{S}, \mathbf{X}, \mathbf{L})$ as before. An MPL formula is true or false of a triple $\mathbf{M}, x, y$ where $\mathbf{M}$ is a structure $(\mathbf{S}, \mathbf{X}, \mathbf{L})$, $x \in \mathbf{X}$, and y is a finite prefix of x (called a *stage*). If y, z are stages or paths, we write $y \leq z$ if y is a prefix of z. We define $\models$ inductively as follows:

(1) $\mathbf{M}, x, y \models P$ iff $P \in \mathbf{L}(\text{last}(y))$.
(2) $\mathbf{M}, x, y \models p \wedge q$ iff $\mathbf{M}, x, y \models p$ and $\mathbf{M}, x, y \models q$;
$\mathbf{M}, x, y \models \neg p$ iff not$(\mathbf{M}, x, y \models p)$.
(3) $\mathbf{M}, x, y \models p\ U\ q$ iff $\exists z(y \leq z \leq x$ and $\mathbf{M}, x, z \models q$
and $\forall w(y \leq w \leq z \Rightarrow \mathbf{M}, x, w \models p))$;
$\mathbf{M}, x, y \models Xp$ iff $\exists z(\mathbf{M}, x, z \models p$ and $y \leq z \leq x$ and $\neg\exists w(y < w < z))$.
(4) $\mathbf{M}, x, y \models \Diamond p$ iff $\exists x'(x' \in \mathbf{X}, y \leq x'$, and $\mathbf{M}, x', y \models p)$.

We intend to show that CTL* interpreted over suffix closed, fusion closed structures is embeddable in MPL. These restrictions are necessary since there are CTL* formulas that are satisfiable only in structures that are not suffix closed (e.g., $EGX\mathbf{true} \wedge \neg EXEGX\mathbf{true}$) or not fusion closed (e.g., $EFEFp \wedge \neg EFp$), whereas every MPL formula is satisfiable in a structure that is both suffix closed and fusion closed. This latter fact arises from the use of stages in defining the semantics of MPL. Indeed, we have

LEMMA 1. *For every structure* $\mathbf{M} = (\mathbf{S}, \mathbf{X}, \mathbf{L})$, *path* $x \in \mathbf{X}$, *and stage* y *of* x, *there exists a suffix closed and fusion closed structure* $\mathbf{M}' = (\mathbf{S}', \mathbf{X}', \mathbf{L}')$, *path* $x' \in \mathbf{X}'$, *and stage* y' *of* x' *such that for all MPL formulas* q, $\mathbf{M}, x, y \models q$ *iff* $\mathbf{M}', x', y' \models q$.

PROOF. Left to the Appendix. □

If y is a stage of x, write x/y to indicate the suffix of x obtained by deleting all but the last state of the prefix y, that is, $y \cdot (x/y) = x$. Then we get

LEMMA 2. *If* $\mathbf{M} = (\mathbf{S}, \mathbf{X}, \mathbf{L})$ *and* $\mathbf{X}$ *is suffix closed and fusion closed, then for all MPL formulas* p *and* $x \in \mathbf{X}$,

$$\mathbf{M}, x, y \models p \text{ iff } \mathbf{M}, x/y, \text{first}(x/y) \models p.$$

PROOF. A straightforward induction on the structure of p suffices. Note that we need suffix closure to ensure that $x/y \in \mathbf{X}$ and fusion closure in order to show that $\mathbf{M}, x/y, \text{first}(x/y) \models \Diamond p$ implies $\mathbf{M}, x, y \models \Diamond p$. Details are left to the reader. □

The preceding lemma shows that, in a suffix closed and fusion closed structure, we can essentially omit mention of the stages. Thus, we will write $\mathbf{M}, x \models p$ as an abbreviation for $\mathbf{M}, x, \text{first}(x) \models p$. We can then translate a CTL* formula p to an MPL formula p'' simply by replacing all occurrences of E by $\Diamond$. We now get

THEOREM 10. *Given a structure* $\mathbf{M} = (\mathbf{S}, \mathbf{X}, \mathbf{L})$ *where* $\mathbf{X}$ *is suffix closed and fusion closed, and path* $x \in \mathbf{X}$,

if p *is a CTL* path formula, then* $\mathbf{M}, x \models p$ *iff* $\mathbf{M}, x \models p''$ *and*
if p *is a CTL* state formula, then* $\mathbf{M}, \text{first}(x) \models p$ *iff* $\mathbf{M}, x \models p''$.

Finally, it is interesting to observe the close relation between CTL*, MPL, and TC, the logic of time and chance introduced by Lehmann and Shelah [20]. The system of TC is essentially the same as that of CTL*, except that, instead of A, the operator ∇ is used, with the interpretation: for a set of paths of measure 1. It is known that TC and MPL have the same complete axiomatizations, and thus the same sets of valid and satisfiable formulas. We would thus expect that there is translation from CTL* (or MPL) to TC.

6. *Conclusion*

We believe that linear time logics are generally adequate for verifying the correctness of preexisting concurrent programs. For manual verification purposes, we do not usually care which computation path is actually followed or that a particular path exists because we are typically interested in properties that hold of all computation paths. It is thus satisfactory to pick an arbitrary path and reason about it. Indeed, Owicki and Lamport [23] give convincing evidence of the power of this approach. In these situations, the simplicity of linear time logic is a strong point in its favor. Nevertheless, we do see a number of advantages in using branching time logic.

Perhaps the most important is that of *model checking*. Given a finite state concurrent program, the global state graph of such a program can be viewed as a CTL* structure (with finitely many states). The problem of checking whether the program has a certain property reduces to that of checking whether the formula describing that property holds for the CTL* structure corresponding to the program. Model checking for large subclasses of CTL* can be done very efficiently, and, in general, it seems that model checking is easier for branching time than for linear time logics (cf. [4, 11]).

Linear time logic also suffers from the problem that, when we view a linear time logic L as the branching time logic B(L) (i.e., all formulas of the form Aq where q is a formula of L), it is not closed under negation. Although it may be possible to prove that a property holds for all executions of a correct program, if a program is incorrect because the property does not hold along some execution, it will be impossible to disprove the property for the program as a whole. Thus, it seems to us that a logic that cannot express a fact such as "property P does not hold along some execution of the program" suffers from a serious disadvantage (cf. [1]).

There are also situations for which we want the ability to explicitly assert the existence of alternative computation paths and must use some system of branching time logic. This arises from the nondeterminism—beyond that used to model concurrency—present in many concurrent programs. Consider an instance of the mutual exclusion problem where each process P_i is functioning as a terminal server. At any moment, P_i (nondeterministically) may or may not receive a character. A key attribute of a correct solution is that it should be possible for one particular P_i to remain in its noncritical section, NCS_i, forever (awaiting but never receiving a character from the keyboard), while other P_j continue to receive and process characters. It should also be possible for P_i to receive a character and then enter its trying region, TRY_i. From there it eventually enters the critical section, CS_i, where the character is processed before returning to NCS_i. But, no matter what happens, once P_i is in NCS_i it either remains there forever or eventually enters TRY_i. To express this property one can use a branching time logic formula involving a term (intended to hold whenever P_i is in NCS_i) of the form $EG(inNCS_i) \wedge EF(inTRY_i) \wedge A(G(inNCS_i) \vee F(inTRY_i))$. However, using Theorem 1, this is provably not expressible in linear time logic, that is, in a language of the form B($L(-)$). The

natural candidate formula, $A(G(inNCS_i) \vee F(inTRY_i)$, allows a "degenerate" model, where all paths satisfy $F(inTRY_i)$ and no path satisfies $G(inNCS_i)$.

This ability to existentially quantify over paths is particularly useful in applications such as automatic program synthesis from temporal logic specifications (cf. [3, 8]) where very precise and thorough specifications are needed. Of course, it is possible to synthesize a class of interesting programs successfully using only linear time logic (cf. [22, 33]); but, as the remarks above demonstrate, some means external to the logic must be used if we wish to ensure the existence of alternative computation paths. We also note that explicit path quantification can be helpful in ensuring that a program exhibits an adequate degree of parallelism (i.e., that it can follow any one of a number of computation paths and is not a degenerate solution with only a single path).

In general, our feeling is that one should use the subset of CTL* most appropriate to the application, where "appropriateness" is measured by expressive power and complexity of testing satisfiability and truth in finite models. We have concentrated on issues of expressive power in this paper. For results on complexity of CTL* and its sublanguages, we refer the reader to [6], [9], [12], and [30–32].

Appendix

PROOF OF THEOREM 4. We first define the set of basic fomulas, B, as follows:

(1) Any propositional formula (i.e., Boolean combination of atomic propositions) is a B formula.
(2) If $p_1, \ldots, p_n$ are propositional formulas, then $p_1\ U\ (p_2\ U \cdots (p_{n-1}\ U\ Gp_n) \cdots)$ is a B formula which we abbreviate $[p_1, \ldots, p_n]$. Intuitively, $[p_1, \ldots, p_n]$ means that there is a finite segment (possibly of length 0) where p_1 holds, followed by a segment where p_2 holds, ..., followed by a segment where p_{n-1} holds, and then p_n holds ever after.
(3) If p is a propositional formula, then GFp is a B formula.
(4) If p is a propositional formula, and $[p_0^0, \ldots, p_{n_0}^0], \ldots, [p_0^m, \ldots, p_{n_m}^m]$, $Fr_1, \ldots, Fr_n$ are B formulas, then $F(p \wedge [p_0^0, \ldots, p_{n_0}^0] \wedge \cdots \wedge [p_0^m, \ldots, p_{n_m}^m] \wedge Fr_1 \wedge \cdots \wedge Fr_n)$ is also a B formula. (Any of the terms in the conjunction may or may not be present.)

Let B^+ be the closure of B under conjunction and disjunction. Note that the formulas of B^+ can be written in conjunctive or disjunctive normal form, where the literals are formulas of B.

CLAIM. *For every linear time formula over F and G, there is an equivalent formula of B^+.*

PROOF OF CLAIM. First note that given any linear time formula over F and G, we can use deMorgan's laws and duality (e.g., $\neg Fp \equiv G\neg p$) to drive the negations inward until only the atomic propositions appear negated. Since B^+ contains the propositional formulas and is closed under conjunction and disjunction, it then suffices to show that, if $p \in B^+$, then Fp and Gp are equivalent to some B^+ formula.

For Fp note that, since $F(q_1 \vee q_2) \equiv Fq_1 \vee Fq_2$, it suffices to show Fp is equivalent to some B^+ formula just when p is a conjunction of formulas in B. This follows directly from (4) above and the observation that $F(q_1 \wedge GFq_2) \equiv Fq_1 \wedge GFq_2$.

Similarly, since $G(q_1 \wedge q_2) \equiv Gq_1 \wedge Gq_2$, it suffices to show Gp is equivalent to some B^+ formula just when p is a disjunction of formulas in B. This follows using

the observation below (where p' and the q'_k are propositional formulas):

$$G(p' \vee (\bigvee_i [p_0^i, \ldots, p_{n_i}^i]) \vee (\bigvee_j Fq_j) \vee (\bigvee_k GFq'_k))$$

$$\equiv$$

$$Gp' \vee (\bigvee_i [p', p_0^i, \ldots, p_{n_i}^i]) \vee (\bigvee_k GFq'_k) \vee (\bigvee_j GFq_j)$$
$$\vee (\bigvee_{i,j} [F(q_j \wedge [q_j, p']) \vee F(q_j \wedge [q_j\, p', p_0^i, \ldots, p_{n_i}^i])]).$$

It is easy to check that the right-hand side of the equivalence implies the left-hand side. To see the converse, note that the first three disjuncts of the right-hand side take care of the case that no q_j is ever true, and the fourth disjunct covers the case that some q_j is true infinitely often (or at the last state of a finite path). The last disjunct corresponds to the last possibility: all q_j being true only finitely often (and not at the last state of a finite path). In this case, the last time any q_j is true, either Gp' or one of the $[q_j, p', p_0^i, \ldots, p_{n_i}^i]$ will be true at the next state. This would be a B^+ formula except that q_j in some GFq_j may not be a propositional formula.

If q in GFq is not a propositional formula, note that q still must be in the form of (4) above since it is the argument to F. Note also the equivalence below:

$$GF(p \wedge [p_0^0, \ldots, p_{n_0}^0] \wedge \cdots \wedge [p_0^m, \ldots, p_{n_m}^m] \wedge Fr_1 \wedge \cdots \wedge Fr_n)$$

$$\equiv$$

$$GFp \wedge F([p_0^0, \ldots, p_{n_0}^0]) \wedge \cdots \wedge F([p_0^m, \ldots, p_{n_m}^m]) \wedge GFr_1 \wedge \cdots \wedge GFr_n.$$

By repeatedly applying this equivalence, we can get down to the case where GF only takes a propositional formula as an argument. This completes the proof of the claim. □

It remains to show that if p is a B^+ formula, then Ep is equivalent to a B(F, X, U, $\overset{\infty}{F}$, $\wedge$, $\neg$) formula. Since $E(q \vee q') \equiv Eq \vee Eq'$, it suffices to prove the result in the case where p is a conjunction of B formulas. We proceed by induction on the number of subformulas of p of the form Fr (corresponding to rule 4 above).

If p has no F's, then it is of the form

$$q \wedge [p_0^0, \ldots, p_{n_0}^0] \wedge \cdots \wedge [p_0^m, \ldots, p_{n_m}^m] \wedge \bigwedge_i GFr_i$$

where q is propositional. We first show that a conjunction of formulas of the form $[p_0, \ldots, p_n]$ is equivalent to a disjunction of such formulas. Given $[p_0, \ldots, p_n]$, $[q_0, \ldots, q_m]$, we say that the ordering of terms in $[p_0 \wedge q_0, \ldots, p_{i_k} \wedge q_{j_k}, \ldots, p_n \wedge q_m]$ is *consistent* provided that, if $p_{i_k} \wedge q_{j_k}$ appears before $p_{i_h} \wedge q_{j_h}$, then $i_k \leq i_h$ and $j_k \leq j_h$. Now observe that

$$[p_0, \ldots, p_n] \wedge [q_0, \ldots, q_m] \equiv \bigvee \{[p_0 \wedge q_0, \ldots, p_{i_k} \wedge q_{j_k}, \ldots, p_n \wedge q_m]$$
with consistent ordering of terms$\}$.

Thus, we can assume (if p has no F's) that p is of the form $q \wedge [p_0, \ldots, p_n] \wedge \bigwedge_i GFr_i$ by again using the fact that $E[q \vee q'] \equiv Eq \vee Eq'$. But

$$E[q \wedge [p_0, \ldots, p_n] \wedge \bigwedge_i GFr_i]$$
$$\equiv q \wedge E[p_0\, U\, E[p_1\, U \cdots E[p_{n-1}\, U\, E[Gp_n \wedge \bigwedge_i GFr_i]]\cdots]].$$

This is an B(F, X, U, $\overset{\infty}{F}$, $\wedge$, $\neg$) formula as desired since $GF \equiv \overset{\infty}{F}$.

In general, p has the form

$$q \wedge [p_0, \ldots, p_n] \wedge Fr_1 \wedge \cdots \wedge Fr_m \wedge \bigwedge_h GFs_h,$$

where q is propositional. Observe that

$$Ep \equiv q \wedge \bigvee_{i,j} f_{ij},$$

where $f_{ij} =$

$$E[p_0\ U\ E[p_1\ U \cdots E[p_j\ U\ E([p_j, \ldots, p_n] \wedge r_i \wedge \bigwedge_{k \neq i} Fr_k \wedge \bigwedge_h GFs_h)] \cdots]].$$

(Intuitively, we are disjuncting over which r_i gets satisfied first and in which segment p_j this occurs.) $E([p_j, \ldots, p_n] \wedge r_i \wedge \bigwedge_{k \neq i} Fr_k \wedge \bigwedge_h GFs_h)$ has one fewer F, so we can apply the induction hypothesis. This completes the proof of Theorem 4. □

PROOF OF THEOREM 5 (continued). Recall that we must prove that

(2) for any B(F, X, U, $\overset{\infty}{F}$, $\wedge$ $\neg$) formula p there is a B(F, X, U) formula q that is equivalent to p over these two sequences of models—that is, for all i and all states s in $\mathbf{M}_i$,

$$\mathbf{M}_i, s \models p \equiv q, \qquad \text{and similarly for } \mathbf{N}_i;$$

(3) for any B(F, X, U) formula p, with $|p| \leq i$, $\mathbf{M}_i, a_i \models p$ iff $\mathbf{N}_i, a_i \models p$.

We first prove (3), arguing by induction on $|p|$, that for all B(F, X, U) formulas p,

(*) if $|p| \leq i$, then ($\mathbf{M}_i, a_i \models p$ iff $\mathbf{N}_i, a_i \models p$).

Note that (*) trivially implies that, if $|p| \leq i$, then ($\mathbf{M}_{i+1}, a_i \models p$ iff $\mathbf{N}_{i+1}, a_i \models p$), which in turn can be seen to imply that

(**) if $|p| \leq i$, then ($\mathbf{M}_{i+1}, b_{i+1} \models p$ iff $\mathbf{N}_{i+1}, b_{i+1} \models p$).

We take EXq, $E[q\ U\ r]$ and $A[q\ U\ r]$ as our primitive operators in the induction, since any B(F, X, U) formula is equivalent to one using only these modalities. The argument proceeds in cases based on the structure of the B(F, X, U) formulas p. The cases in which p is an atomic proposition, a conjunction $q \wedge r$, or a negation $\neg q$ are easy and left to the reader.

If p is of the form EXq then,

$\mathbf{M}_{i+1}, a_{i+1} \models EXq$

if and only if

$\mathbf{M}_{i+1}, b_{i+1} \models q$ or $\mathbf{M}_i, a_i \models q$ or $\mathbf{N}_i, a_i \models q$

if and only if

$\mathbf{N}_{i+1}, b_{i+1} \models q$ (by **) or $\mathbf{M}_i, a_i \models q$ or $\mathbf{N}_i, a_i \models q$

if and only if

$\mathbf{N}_{i+1}, a_{i+1} \models EXq$.

If $p = E[q\ U\ r]$ then,

$\mathbf{M}_{i+1}, a_{i+1} \models E[q\ U\ r]$ if and only if

(1) $\mathbf{M}_{i+1}, a_{i+1} \models r$ or
(2) $\mathbf{M}_{i+1}, a_{i+1} \models q$, $\mathbf{M}_{i+1}, b_{i+1} \models r$ or
(3) $\mathbf{M}_{i+1}, a_{i+1} \models q$, $\mathbf{M}_i, a_i \models E[q\ U\ r]$ or
(4) $\mathbf{M}_{i+1}, a_{i+1} \models q$, $\mathbf{N}_i, a_i \models E[q\ U\ r]$

if and only if

(1) $\mathbf{N}_{i+1}, a_{i+1} \models r$ (by (*)) or
(2) $\mathbf{N}_{i+1}, a_{i+1} \models q$, $\mathbf{N}_{i+1}, b_{i+1} \models r$ (by (*), (**), respectively) or
(3) $\mathbf{N}_{i+1}, a_{i+1} \models q$ (by (*)), $\mathbf{M}_i, a_i \models E[q\ U\ r]$ or
(4) $\mathbf{N}_{i+1}, a_{i+1} \models q$ (by (*)), $\mathbf{N}_i, a_i \models E[q\ U\ r]$

if and only if

$\mathbf{N}_{i+1}, a_{i+1} \models E[q\ U\ r]$.

In the last case, if $p = A[q\ U\ r]$ then

$\mathbf{M}_{i+1}, a_{i+1} \models A[q\ U\ r]$ if and only if

(1) $\mathbf{M}_{i+1}, a_{i+1} \models r$ or
(2) $\mathbf{M}_{i+1}, a_{i+1} \models q$, $\mathbf{M}_{i+1}, b_{i+1} \models r$,
$\mathbf{M}_i, a_i \models A[q\ U\ r]$, $\mathbf{N}_i, a_i \models A[q\ U\ r]$ or
(3) $\mathbf{M}_{i+1}, a_{i+1} \models q$, $\mathbf{M}_{i+1}, b_{i+1} \models q$,
$\mathbf{M}_i, a_i \models A[q\ U\ r]$, $\mathbf{N}_i, a_i \models A[q\ U\ r]$

if and only if

(1) $\mathbf{N}_{i+1}, a_{i+1} \models r$ (by (*)) or
(2) $\mathbf{N}_{i+1}, a_{i+1} \models q$, $\mathbf{N}_{i+1}, b_{i+1} \models r$ (by (**)),
$\mathbf{M}_i, a_i \models A[q\ U\ r]$, $\mathbf{N}_i, a_i \models A[q\ U\ r]$ or
(3) $\mathbf{N}_{i+1}, a_{i+1} \models q$, $\mathbf{N}_{i+1}, b_{i+1} \models q$ (by (**)),
$\mathbf{M}_i, a_i \models A[q\ U\ r]$, $\mathbf{N}_i, a_i \models A[q\ U\ r]$

if and only if

$\mathbf{N}_{i+1}, a_{i+1} \models A[q\ U\ r]$.

It remains to establish our claim that B(F, X, U) and B(F, X, U, $\overset{\infty}{F}$, $\wedge$, $\neg$) are of equivalent expressive power on the two sequences of models. For any B(F, X, U, $\overset{\infty}{F}$, $\wedge$, $\neg$) formula Eq, q can be placed in disjunctive normal form. Since $E(q' \vee q'') \equiv Eq' \vee Eq''$ and $\overset{\infty}{G}p' \wedge \overset{\infty}{G}p'' \equiv \overset{\infty}{G}(p' \wedge p'')$, it suffices to show the equivalence for any B(F, X, U, $\overset{\infty}{F}$, $\wedge$, $\neg$) formula of the form $p_1 = E[p \wedge \overset{\infty}{F}q_1 \wedge \cdots \wedge \overset{\infty}{F}q_n \wedge \overset{\infty}{G}r]$ where $Ep, q_1, \ldots, q_n$ are B(F, X, U, $\wedge$, $\neg$) formulas. To show this, we observe that every path in one of the structures $\mathbf{M}_i$ or $\mathbf{N}_i$ ends in a self-loop at the state d. Using c to denote either a_i or b_i, we thus have that $\mathbf{M}_i, c \models p_1$ iff $\mathbf{M}_i, c \models Ep$ and $\mathbf{M}_i, d \models q_1 \wedge \cdots \wedge q_n \wedge r$. Moreover, $\mathbf{M}_i, d \models q_1 \wedge \cdots \wedge q_n \wedge r$ iff $\mathbf{M}_i, c \models EFAG(q_1 \wedge \cdots \wedge q_n \wedge r)$. Thus, $\mathbf{M}_i, c \models p_1$ iff $\mathbf{M}_i, c \models Ep \wedge EFAG(q_1 \wedge \cdots \wedge q_n \wedge r)$ and similarly, for $\mathbf{N}_i$. The latter formula is in B(F, X, U, $\wedge$, $\neg$). By Theorem 9, we know that, for any B(F, X, U, $\wedge$, $\neg$) formula q', there exists an equivalent B(F, X, U) formula q. So we are done. (This completes the proof of Theorem 5). □

PROOF OF THEOREM 6. We inductively define two sequences $\mathbf{M}_1, \mathbf{M}_2, \mathbf{M}_3, \ldots$ and $\mathbf{N}_1, \mathbf{N}_2, \mathbf{N}_3, \ldots$ of models as follows. Define $\mathbf{M}_1, \mathbf{N}_1$ to have the graphs shown in Figure 4 where in $\mathbf{M}_1$, $a_1 \models P_1 \wedge \neg P_2 \wedge \neg Q_1 \wedge \neg Q_2 \wedge \neg R$, $b_1 \models \neg P_1 \wedge \neg P_2 \wedge \neg Q_1 \wedge \neg Q_2 \wedge \neg R$, $c_1 \models \neg P_1 \wedge \neg P_2 \wedge Q_1 \wedge \neg Q_2 \wedge \neg R$, $d_1 \models \neg P_1 \wedge \neg P_2 \wedge \neg Q_1 \wedge Q_2 \wedge \neg R$, and $e_1 \models \neg P_1 \wedge \neg P_2 \wedge \neg Q_1 \wedge \neg Q_2 \wedge R$ and in $\mathbf{N}_1$, $a_1 \models P_1 \wedge \neg P_2 \wedge \neg Q_1 \wedge \neg Q_2 \wedge \neg R$, $b_1 \models \neg P_1 \wedge \neg P_2 \wedge Q_1 \wedge \neg Q_2 \wedge \neg R$, $c_1 \models \neg P_1 \wedge P_2 \wedge \neg Q_1 \wedge \neg Q_2 \wedge \neg R$, $d_1 \models \neg P_1 \wedge \neg P_2 \wedge \neg Q_1 \wedge Q_2 \wedge \neg R$, and $e_1 \models \neg P_1 \wedge \neg P_2 \wedge \neg Q_1 \wedge \neg Q_2 \wedge R$.

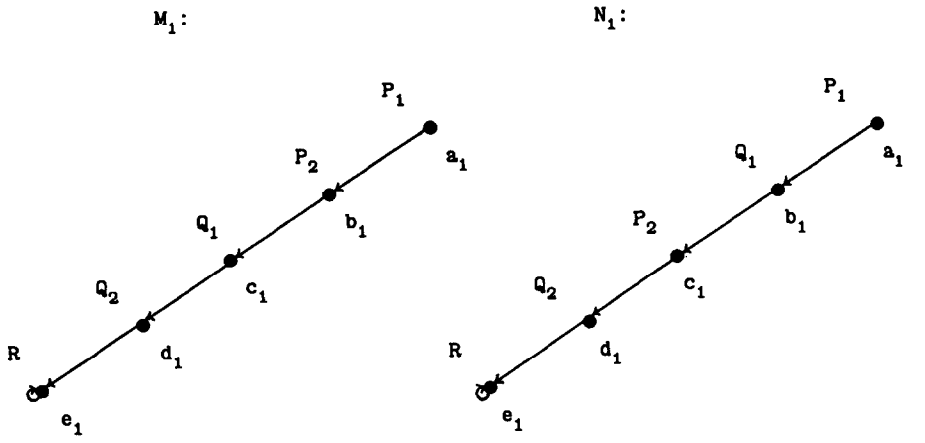


FIGURE 4

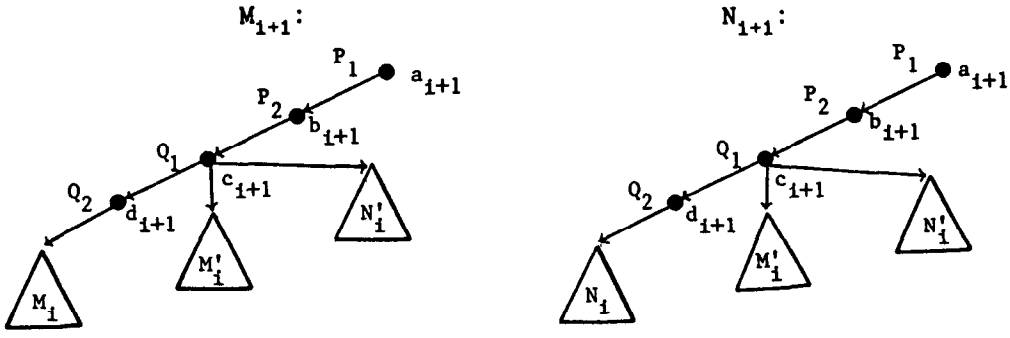


FIGURE 5

Suppose we have defined $\mathbf{M}_i$ and $\mathbf{N}_i$. Then $\mathbf{M}_{i+1}$ and $\mathbf{N}_{i+1}$ have the graphs shown in Figure 5 where in both $\mathbf{M}_{i+1}$ and $\mathbf{N}_{i+1}$, $a_{i+1} \models P_1 \wedge \neg P_2 \wedge \neg Q_1 \wedge \neg Q_2 \wedge \neg R$, $b_{i+1} \models \neg P_1 \wedge P_2 \wedge \neg Q_1 \wedge \neg Q_2 \wedge \neg R$, $c_{i+1} \models \neg P_1 \wedge \neg P_2 \wedge Q_1 \wedge \neg Q_2 \wedge \neg R$, and $d_{i+1} \models \neg P_1 \wedge \neg P_2 \wedge \neg Q_1 \wedge Q_2 \wedge \neg R$, and finally $\mathbf{M}_i'$, $\mathbf{N}_i'$ are copies of $\mathbf{M}_i$, $\mathbf{N}_i$, respectively.

It should be clear that

(1) for all i, $\mathbf{M}_i, a_i \models E[((P_1 \; U \; P_2) \vee (Q_1 \; U \; Q_2)) \; U \; R]$ and
$\mathbf{N}_i, a_i \models \neg E[((P_1 \; U \; P_2) \vee (Q_1 \; U \; Q_2)) \; U \; R]$.

We can also show that

(2) For any B(F, X, U, $\overset{\infty}{F}$, $\wedge$, $\neg$) formula p, there is a B(F, X, U) formula q, which is equivalent to p over these two sequences of models. That is, for all i and all states s in $\mathbf{M}_i$,
$$\mathbf{M}_i, s \models p \equiv q, \quad \text{and similarly for } \mathbf{N}_i.$$

(3) For any B(F, X, U) formula p, with $|p| \leq i$, $\mathbf{M}_i, a_i \models p$ iff $\mathbf{N}_i, a_i \models p$.

The details of the remainder of the proof follow along exactly the same lines as that of Theorem 5. Details are left to the reader. □

FIGURE 6

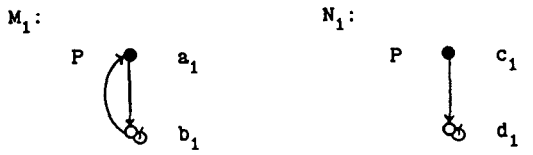


FIGURE 7

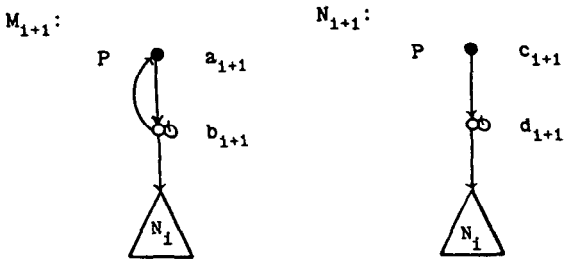
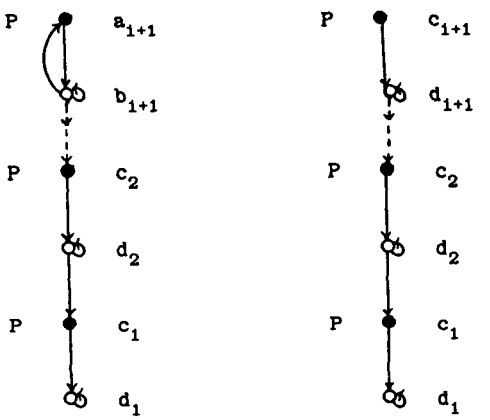


FIGURE 8



PROOF OF THEOREM 7. We inductively define two sequences of models $\mathbf{M}_1, \mathbf{M}_2, \mathbf{M}_3, \ldots$ and $\mathbf{N}_1, \mathbf{N}_2, \mathbf{N}_3, \ldots$ such that for all i, $\mathbf{M}_i, a_i \models E^{\infty}FP$ and $\mathbf{N}_i, c_i \models \neg E^{\infty}FP$. We show that B($F, X, U$) is unable to distinguish between the two sequences of models; that is, for all B(F, X, U) formulas p with $|p| \leq i$, $\mathbf{M}_i, a_i \models p$ iff $\mathbf{N}_i, c_i \models p$. The result follows since, if $E^{\infty}FP$ were equivalent to some B($F, X, U, \wedge, \neg$) formula p, by Theorem 8 it would also be equivalent to some B(F, X, U) formula p' of length i. But $\mathbf{M}_i, a_i \models p'$ iff $\mathbf{N}_i, c_i \models p'$, contradicting the fact that $\mathbf{M}_i, a_i \models E^{\infty}FP$ and $\mathbf{N}_i, c_i \models \neg E^{\infty}FP$. (Since a_i, b_i appear only in $\mathbf{M}_i$ and c_i, d_i in $\mathbf{N}_i$, we omit the models from our assertions.)

We define $\mathbf{M}_1, \mathbf{N}_1$ to have the graphs shown in Figure 6 where $a_1 \models P$, $b_1 \models \neg P$, $c_1 \models P$, and $d_1 \models \neg P$. Assume that $\mathbf{M}_i, \mathbf{N}_i$ are defined. The $\mathbf{M}_{i+1}$ and $\mathbf{N}_{i+1}$ have the graphs shown in Figure 7 where $a_{i+1} \models P$, $b_{i+1} \models \neg P$, $c_{i+1} \models P$, and $d_{i+1} \models \neg P$.

Remark. We see that if we unwind the inductive definitions, the models have the form depicted in Figure 8.

We first argue by induction on $|p|$, that for any B(F, X, U) formula p,

(*) $[(\exists j \geq |p|\ c_j \models p)$ iff $(\forall j \geq |p|, c_j \models p)]$ and
$[(\exists j \geq |p|\ d_j \models p)$ iff $(\forall j \geq |p|, d_j \models p)]$.

The basis case when $|p| = 1$ is obvious, since all c_j agree on the atomic propositions as do all d_j. For the induction step, we assume (*) for formulas of length I and try to show it for $I + 1$. Note that the $\Leftarrow$ direction is obvious. To establish the $\Rightarrow$ direction, it suffices to show that, if p is of length $I + 1$ and $j \geq I + 1$, then

(i) $c_j \models p$ implies $c_{j+1} \models p$,
(ii) $c_j \models p$ and $j > I + 1$ imply $c_{j-1} \models p$,
(iii) $d_j \models p$ implies $d_{j+1} \models p$,
(iv) $d_j \models p$ and $j > I + 1$ imply $d_{j-1} \models p$.

We break the argument into cases depending on the form of p. If p is of the form $q \wedge r$, or $\neg q$ the argument is straightforward and left to the reader.

Case 1: $p = EXq$. Assume $c_j \models p$. Note that $c_j \models p$ iff $c_j \models q$ or $d_j \models q$. By the induction assumption twice, $c_{j+1} \models q$ or $d_{j+1} \models q$. Similarly, if $j > |p| = I + 1$ so that $j - 1 > I > |q|$, we also see that $c_{j-1} \models q$ or $d_{j-1} \models q$. Thus, $c_{j+1} \models p$ ((i)) and $c_{j-1} \models p$ ((ii)).

Now assume $d_j \models p$. Note that $d_j \models p$ iff

(1) $d_j \models q$ or
(2) $c_{j-1} \models q$.

By the induction assumption twice, (1) implies $d_{j+1} \models q$ and (2) implies $c_j \models q$ whence $d_{j+1} \models p$ ((iii)). If $j > |p| = I + 1$ then $j - 1, j - 2 \geq |q|$ so by the induction assumption twice, $d_{j-1} \models q$ and $c_{j-2} \models q$. We conclude that $d_{j-1} \models p$ ((iv)).

Case 2: $p = E[q\ U\ r]$. Assume $c_j \models p$. Now $c_j \models p$ iff

(1) $c_j \models r$ or
(2) $c_j \models q$, $d_j \models r$ or
(3) $c_j \models q$, $d_j \models q$, $c_{j-1} \models p$.

This implies

(1′) $c_{j+1} \models r$ (by the induction assumption) or
(2′) $c_{j+1} \models q$, $d_{j+1} \models r$ (by the induction assumption twice) or
(3′) $c_{j+1} \models q$, $d_{j+1} \models q$, $c_j \models p$ (by the induction assumption twice and the assumption $c_j \models p$).

It follows that $c_{j+1} \models p$ ((i)). If we assume that $j > I + 1$, then we can also argue that

(1″) $c_{j-1} \models r$ (by the induction assumption) or
(2″) $c_{j-1} \models q$, $d_j \models r$ (by the induction twice) or
(3″) $c_{j-1} \models p$ (as a special case of (3)).

Thus, in all cases, we have $c_{j-1} \models p$ ((ii)).

Next assume that $d_j \models p$. Note that $d_j \models p$ iff

(1) $d_j \models r$ or
(2) $d_j \models q$, $c_{j-1} \models r$ or
(3) $c_j \models q$, $c_{j-1} \models q$, $d_{j-1} \models p$.

By repeated application of the induction hypothesis it follows that

(1′) $d_{j+1} \models r$ or
(2′) $d_{j+1} \models q$, $c_j \models r$ or
(3′) $d_{j+1} \models q$, $c_j \models q$, $d_j \models p$ ($d_j \models p$ follows by our assumption).

Thus $d_{j+1} \models p$ ((iii)).

If we also assume that $j > I + 1$, then we can use the induction assumption to argue that

(1″) $d_{j-1} \models r$ or
(2″) $d_{j-1} \models q$, $c_{j-2} \models r$ or
(3″) $d_{j-1} \models p$ ($d_{j-1} \models p$ follows directly from (3)).

Thus, in all cases, we get $d_{j-1} \models p$ ((iv)).

Case 3: $p = A[q\ U\ r]$: Assume $c_j \models p$. Now $c_j \models p$ iff

(1) $c_j \models r$ or
(2) $c_j \models q$, $d_j \models r$.

By the induction hypothesis, it follows that

(1′) $c_{j+1} \models r$ or
(2′) $c_{j+1} \models q$, $d_{j+1} \models r$,

whence $c_{j+1} \models p$ ((i)).

Assuming $j > I + 1$, we also get that

(1″) $c_{j-1} \models r$ or
(2″) $c_{j-1} \models q$, $d_{j-1} \models r$,

and we conclude that $c_{j-1} \models p$ ((ii)).

Next assume that $d_j \models p$. Note that $d_j \models p$ iff $d_j \models r$. By the induction hypothesis, it follows that $d_{j+1} \models r$ and (assuming $j > I + 1$) $d_{j-1} \models r$. We conclude that $d_{j+1} \models p$ ((iii)) and $d_{j-1} \models p$ ((iv)).

This completes the proof of (*). □

We now argue by induction on $|p|$ that

(**) $i \geq |p|$ implies $(a_i \models p$ iff $c_i \models p)$ and $(b_i \models p$ iff $d_i \models p)$.

We break the argument into cases depending on the structure of p. The cases in which p is an atomic proposition, a conjunction $q \wedge r$, or a negation $\neg q$ are easy and left to the reader. We present the cases where p is of the form EXq, $E[q\ U\ r]$, or $A[q\ U\ r]$.

Case 1: $p = EXq$. We first note that $a_i \models p$ iff

(1) $a_i \models q$ or $b_i \models q$,
(2) $c_i \models q$ or $d_i \models q$ (by the induction hypothesis twice),
(3) $c_i \models p$.

We next note that $b_i \models p$ iff

(1) $a_i \models q$ or
(2) $b_i \models q$ or
(3) $c_{i-1} \models q$,

and that $d_i \models p$ iff

(4) $d_i \models q$ or
(5) $c_{i-1} \models q$.

Now (1) implies $c_i \models q$ (by induction hypothesis) which in turn (by (*)) implies (5). Also, (2) implies (4) (by induction hypothesis) and (3) coincides with (5). Thus, $b_i \models p$ implies $d_i \models p$. For the converse, note that (4) implies (2) (by the induction hypothesis) and (5) coincides with (3).

Case 2: $p = E[q\ U\ r]$. Note that $a_i \models p$ iff

(1) $a_i \models r$ or
(2) $a_i \models q$, $b_i \models r$ or
(3) $a_i \models q$, $b_i \models q$, $c_{i-1} \models p$

and that $c_i \models p$ iff

(4) $c_i \models r$ or
(5) $c_i \models q$, $d_i \models r$ or
(6) $c_i \models q$, $d_i \models q$, $c_{i-1} \models p$.

By repeated application of the induction hypothesis we see that $a_i \models p$ iff $c_i \models p$.

Next note that $b_i \models p$ iff

(1) $b_i \models r$ or
(2) $b_i \models q$, $a_i \models r$, or
(3) $b_i \models q$, $c_{i-1} \models p$

and that $d_i \models p$ iff

(4) $d_i \models r$ or
(5) $d_i \models q$, $c_{i-1} \models p$.

Observe that (2) implies (3) since $a_i \models r$ implies $c_i \models r$ (by induction hypothesis), $c_i \models r$ implies $c_{i-1} \models r$ (by (*)), and $c_{i-1} \models r$ implies $c_{i-1} \models p$. By repeated application of the induction hypothesis we also see that (1) iff (4) and (3) iff (5). We conclude that $b_i \models p$ iff $d_i \models p$. □

Case 3: $p = A[q\ U\ r]$. First note that $a_i \models p$ iff

(1) $a_i \models r$ or
(2) $a_i \models q$, $b_i \models r$

and that $c_i \models p$ iff

(3) $c_i \models r$ or
(4) $c_i \models q$, $d_i \models r$.

By repeated application of the induction hypothesis we see that

$$a_i \models p \quad \text{iff} \quad c_i \models p.$$

Next note that

$$b_i \models p \quad \text{iff} \quad b_i \models r$$

and that

$$d_i \models p \quad \text{iff} \quad d_i \models r.$$

Again by repeated application of the induction hypothesis, we see that $b_i \models p$ iff $d_i \models p$.

This completes the proof of (**) and of the Theorem 7.

SUMMARY OF PL. For the reader's convenience, we summarize the syntax and semantics of PL here. (The reader should consult [16] for additional details.) The first and last states of a path x are denoted by first(x) and last(x), respectively (last(x) does not exist for infinite paths). If x, y are two paths such that last(x) = first(y), then $x \cdot y$ denotes the fusion of x and y.[5] For example, if $x = (s_1, s_2, s_3)$

[5] This was called simply the concatenation of x and y in [16].

and $y = (s_3, s_4, s_5)$, then $x \cdot y = (s_1, s_2, s_3, s_4, s_5)$. If last$(x) \neq$ first(y), then $x \cdot y$ is undefined.

All formulas in PL are path formula. The language of PL, however, is the same as (test-free) PDL [13] augmented with two additional operators **f** and **suf**. Intuitively, **f**p means that p holds in the unique initial prefix of length 0 (i.e., at the first state) of a path while **suf** is analogous to the ordinary until operator of temporal logic.

PL fomulas are interpreted over a *path model* $\mathbf{M} = (\mathbf{S}, \vDash, \mathbf{R})$ where $\mathbf{S}$ is a set of states, $\vDash$ is a satisfiability relation for atomic propositions, and $\mathbf{R}$ is an assignment of sets of paths to atomic programs. A path satisfies an atomic proposition iff its first state does. We write $x \vDash P$ if path x satisfies atomic proposition P, and $x \in \mathbf{R}_a$ if x is a member of the set of paths assigned to atomic program a. We then inductively extend $\vDash$ and $\mathbf{R}$ to compound formulas and programs as follows:

$\mathbf{R}_{\alpha\beta} = \{x \cdot y \mid x \in \mathbf{R}_\alpha \text{ and } y \in \mathbf{R}_\beta\}$;
$\mathbf{R}_{\alpha \cup \beta} = \mathbf{R}_\alpha \cup \mathbf{R}_\beta$;
$\mathbf{R}_{\alpha^*} = \bigcup_i \mathbf{R}_\alpha^i$;

$x \vDash p \vee q$ iff $x \vDash p$ or $x \vDash q$;
$x \vDash \neg p$ iff not $(x \vDash p)$;
$x \vDash \langle \alpha \rangle p$ iff $\exists y \in \mathbf{R}_\alpha\ x \cdot y \vDash p$;
$x \vDash \mathbf{f}p$ iff first$(x) \vDash p$;
$x \vDash p$ **suf** q iff there exists a path y such that

(i) y is a proper suffice of x and $y \vDash q$, and
(ii) $\forall$ z, if z is a proper suffix of x and y is a proper suffix of z, then $z \vDash p$.

Note that the operator **n**p defined as (**false suf** p) is analogous to the next time operator Xp of linear temporal logic.

PROOF OF LEMMA 1. Suppose $\mathbf{M} = (\mathbf{S}, \mathbf{X}, \mathbf{L})$. Fix a path $x \in \mathbf{X}$, and a stage y of x. Let $\mathbf{X}_1 = \{z \in \mathbf{X} \mid y \leq z\}$ and $\mathbf{M}_1 = (\mathbf{S}, \mathbf{X}_1, \mathbf{L})$. It is easy to check that $\mathbf{M}_1, x, y \vDash p$ iff $\mathbf{M}, x, y \vDash p$. To simplify the notation, assume for now that $\mathbf{X}_1$ is countable (the case where $\mathbf{X}_1$ is uncountable is considered below) and consists of the distinct paths $x = x_0, x_1, x_2, x_3, \ldots$. We now unwind $\mathbf{M}_1$ into a "treelike" model. Define a set $\mathbf{T} = \{t_{ij} \mid i, j \geq 0\}$ of "fresh" states distinct from the states in $\mathbf{S}$. We inductively define a set of paths $\mathbf{X}' = \{y_0, y_1, y_2, \ldots\}$ over $\mathbf{T}$ which is fusion closed along with a mapping h: $\mathbf{T} \to \mathbf{S}$ as follows: Suppose $x_0 = (s_0, s_1, \ldots, s_k, \ldots)$ (which could be finite or infinite). Then define $y_0 = (t_{00}, t_{01}, \ldots, t_{0k}, \ldots)$ and $h(t_{0j}) = s_j$ for all j. We can extend h so that if $y = (u_0, \ldots, u_m, \ldots)$ then $h(y) = h(u_0), \ldots, h(u_m), \ldots)$. Note that $h(y_0) = x_0$. Now suppose we have constructed the paths y_j for all $j < i$ so that $h(y_j) = x_j$. We now define $y_i = (t_{k_j j})$, where, for all $j < 1 + |x_i|$, k_j is the least k such that the length j stage of x_k is also a stage of x_i. Now, extend h so that $h(y_i) = x_i$. Let $\mathbf{T}'$ consist of those states of $\mathbf{T}$ that occur in y_i for some i. Also let $\mathbf{L}'$ be a labeling of states in $\mathbf{T}'$ such that $\mathbf{L}'(t) = \mathbf{L}(h(t))$. Now define $\mathbf{M}' = (\mathbf{T}', \mathbf{X}', \mathbf{L}')$. Then we can show, by a straightforward induction on the structure of formulas, that for any formula q, if $|w| \geq |y|$, then $\mathbf{M}', z, w \vDash q$ iff $\mathbf{M}, h(z), h(w) \vDash q$.

Next define $\mathbf{X}'' = \{z^i \mid z \in \mathbf{X}'\}$. Using the observations that no state occurs twice along any path, and that two paths have a state in common iff they have a common prefix including the state, it is easy to check that $\mathbf{X}''$ is fusion closed and suffix

closed. Let $\mathbf{M}'' = (\mathbf{T}', \mathbf{X}'', \mathbf{L}')$. Then we can argue by induction on the length of formula q that, for $z \in \mathbf{X}'$, $\mathbf{M}'$, z, $w \models q$ iff $\mathbf{M}''$, z, $w \models q$. Thus, $\mathbf{M}''$ is a fusion-closed and suffix-closed model of p. Note that our original path $x \in \mathbf{X}''$. The above argument shows that for all MPL formulas q, $\mathbf{M}$, x, $y \models q$ iff $\mathbf{M}''$, x, $y \models q$.

If $\mathbf{X}_1$ is not countable, a similar argument goes through (although we seem to need the well-ordering principle—which is equivalent to the axiom of choice—to order the paths first). □

ACKNOWLEDGMENT. We wish to thank the referees for their careful reading of the paper and their helpful criticisms which led to improvements over the earlier version. We also thank C. L. Lei for helpful comments.

REFERENCES

1. ABRAHAMSON, K. Decidability and expressiveness of logics of processes. Ph.D. dissertation, Univ. of Washington, Seattle, 1980.
2. BEN-ARI, M., MANNA, Z., AND PNUELI, A. The temporal logic of branching time. In *Proceedings of the 8th Annual ACM Symposium on Principles of Programming Languages* (Williamsburg, Va., Jan. 26–28). ACM, New York, 1981, pp. 164–176.
3. CLARKE, E. M., AND EMERSON, E. A. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proceedings of the IBM Workshop on Logics of Programs*. Lecture Notes in Computer Science, vol. 131. Springer-Verlag, New York, 1981, pp. 52–71.
4. CLARKE, E. M., EMERSON, E. A., AND SISTLA, A. P. Automatic verification of finite-state concurrent systems using temporal logic specifications: A practical approach. In *Proceedings of the 10th Annual ACM Symposium on Principles of Programming Languages* (Austin, Tex., Jan. 24–26). ACM, New York, 1983, pp. 117–126.
5. EMERSON, E. A. Alternative semantics for temporal logics. *Theor. Comput. Sci. 26* (1983), 121–130.
6. EMERSON, E. A. Automata, tableaux, and temporal logics. In *Proceedings of the Brooklyn College Conference on Logics of Programs 1985*. Lecture Notes in Computer Science, vol. 193. Springer-Verlag, New York, 1985, pp. 79–87.
7. EMERSON, E. A., AND CLARKE, E. M. Characterizing correctness properties of parallel programs as fixpoints. In *Proceedings of the 7th International Colloquium on Automata, Languages, and Programming*. Lecture Notes in Computer Science, vol. 85. Springer-Verlag, New York, 1980, pp. 169–181.
8. EMERSON, E. A., AND CLARKE, E. M. Using branching time logic to synthesize synchronization skeletons. *Sci. Comput. Prog. 2* (1982), 241–266.
9. EMERSON, E. A., AND HALPERN, J. Y. Decision procedures and expressiveness in the temporal logic of branching time. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing* (San Francisco, Calif., May 5–7). ACM, New York, 1982, pp. 169–180.
10. EMERSON, E. A., AND HALPERN, J. Y. "Sometimes" and "not never" revisited: On branching vs. linear time. Res. Rep. RJ 4197, IBM Research, San Jose, Calif., 1984 (Also Tech. Rep. 84-01, Comput. Sci. Dept., Univ. of Texas, Austin, Tex. 1984).
11. EMERSON, E. A., AND LEI, C. L. Modalities for model checking: Branching time strikes back. In *Proceedings of the 12th Annual ACM Symposium on Principles of Programming Languages* (New Orleans, La., Jan. 14–16). ACM, New York, 1985, pp. 84–96.
12. EMERSON, E. A., AND SISTLA, A. P. Deciding full branching time logic. *Inf. Control 61*, 3 (1984), 175–201.
13. FISCHER, M. J., AND LADNER, R. E. Propositional dynamic logic of regular programs. *J. Comput. Syst. Sci. 18* (1979), 194–211.
14. GABBAY, D., PNUELI, A., SHELAH, S., AND STAVI, J. On the temporal analysis of fairness. In *Proceedings of the 7th Annual ACM Symposium on Principles of Programming Languages* (Las Vegas, Nev., Jan. 28–30). ACM, New York, 1980, pp. 163–173.
15. HAILPERN, B., AND OWICKI, S. Modular verification of concurrent programs using temporal logic. Tech. Rep., Stanford Univ. Stanford, Calif., 1981.
16. HAREL, D., KOZEN, D., AND PARIKH, R. Process logic: Expressiveness, decidability, and completeness. *J. Comput. Syst. Sci. 25* (1982), 144–170.
17. HAREL, D., AND SHERMAN, R. Looping vs. repeating in dynamic logic. *Inf. Control 55* (1982), 175–192.

18. KAMP, H. Tense logic and the theory of linear order. Ph.D. dissertation, UCLA, Los Angeles, 1968.
19. LAMPORT, L. "Sometime" is sometimes "not never"—On the temporal logic of programs. In *Proceedings of the 7th Annual ACM Symposium on Principles of Programming Languages* (Las Vegas, Nev., Jan. 28–30). ACM, New York, 1980, pp. 174–185.
20. LEHMANN, D., AND SHELAH, S. Reasoning with time and chance. *Inf. Control 53* (1982), 165–198.
21. MANNA, Z., AND PNUELI, A. The modal logic of programs. In *Proceedings of the 6th International Colloquium on Automata, Languages, and Programming*. Lecture Notes in Computer Science, vol. 71. Springer-Verlag, New York, 1979, pp. 385–408.
22. MANNA, Z., AND WOLPER, P. Synthesis of communicating processes from temporal logic specifications. *ACM Trans. Prog. Lang. Syst. 6*, 1 (Jan. 1984), 68–93.
23. OWICKI, S., AND LAMPORT, L. Proving liveness properties of concurrent programs. *ACM Trans. Prog. Lang. Syst. 4*, 3 (July 1982), 455–495.
24. PNUELI, A. The temporal logic of programs. In *Proceedings of the 19th Annual Symposium on Foundations of Computer Science*. IEEE, New York, 1977, pp. 46–57.
25. PNUELI, A. The temporal semantics of concurrent programs. *Theor. Comput. Sci. 13* (1981), 45–60.
26. PRATT, V. R. Process logic. In *Proceedings of the 6th Annual ACM Symposium on Principles of Programming Languages* (San Antonio, Tex., Jan. 29–31). ACM, New York, 1979, pp. 93–100.
27. PRIOR, A. *Time and Modality*. Oxford Univ. Press, London, 1957.
28. PRIOR, A. *Past, Present, and Future*. Oxford Univ. Press, London, 1967.
29. RESCHER, N., AND URQUHART, A. *Temporal logic*. Springer-Verlag, Berlin, 1971.
30. SISTLA, A. P., AND CLARKE, E. M. The complexity of propositional linear temporal logic. *J. ACM 32*, 3 (1985), 733–749.
31. VARDI, M., AND STOCKMEYER, L. Improved upper and lower bounds for modal logics of programs. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing* (Providence, R. I., May 6–8). ACM, New York, 1985, pp. 240–251.
32. VARDI, M., AND WOLPER, P. Yet another process logic. In *CMU Workshop on Logics of Programs*. Lecture Notes in Computer Science, vol. 164. Springer-Verlag, New York, 1983, pp. 501–512.
33. WOLPER, P. Specification and synthesis of communicating processes using an extended temporal logic (preliminary version). In *Proceedings of the 9th Annual ACM Symposium on Principles of Programming Languages* (Albuquerque, N. M., Jan. 25–27). ACM, New York, 1982, pp. 20–33.
34. WOLPER, P. Temporal logic can be more expressive. *Inf. Control 56*, 1/2 (1983), 72–99.

RECEIVED NOVEMBER 1983; REVISED MAY 1985; ACCEPTED JUNE 1985