"SOMETIMES" AND "NOT NEVER" REVISITED: ON BRANCHING VERSUS LINEAR TIME (PRELIMINARY REPORT)

E. Allen Emerson[1] and Joseph Y. Halpern[2]

1. Computer Sciences Department, University of Texas, Austin, TX 78712
2. IBM Research Laboratory, San Jose, CA 95193

1. INTRODUCTION

Temporal logic ([PR57], [PR67]) provides a formalism for describing the occurrence of events in time which is suitable for reasoning about concurrent programs (cf. [PN77]). 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, tree-like nature: at each moment, time may split into alternate courses representing different possible futures. Depending upon which view is chosen, we classify (cf. [RU71]) 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 operators are provided for describing events along a single time path (cf. [GPSS80]). In contrast, in a logic of branching time the operators reflect the branching nature of time by allowing quantification over possible futures (cf. [AB80],[EC80]).

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 [LA80] 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 formulae 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. Based on his comparison and other arguments, he concludes that, while branching time logic is suitable for reasoning about nondeterministic programs, linear time logic is preferable for reasoning about concurrent programs.

In this paper, we re-examine Lamport's arguments and reach somewhat different conclusions. We first point out some technical difficulties with the formalism of [LA80]. For instance, the definition of expressive equivalence leads to paradoxical situations where satisfiable formulae are classified as equivalent to false. Moreover, the proofs of the results comparing expressive power do not apply in the case of structures

1. This author was partially supported by a University of Texas URI Summer Research Award and a departmental grant from IBM.

2. Some of this work was performed while the author 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 NSF grant MCS80-10707.

generated by a binary relation like those used in the logics of [FL79] and [BMP81]. We give a more refined basis for comparing expressive power that avoids these technical difficulties. It does turn out that expressibility results corresponding to Lamport's still hold. However, it should be emphasized that these results apply only to the two particular systems that he defines. Sweeping conclusions regarding branching versus linear time logic in general are not justified on this basis.

We will argue that there are several different aspects to the problem of designing and reasoning about concurrent programs. While 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 pre-existing concurrent programs. For verification purposes, 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. 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 corresponding 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 [CE81] and studied in [EH82]. 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.

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 that (cf. Theorem 4.1) 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 [AB80] and PL [HKP80]. Finally, section 6 concludes with a comparison of the utility of branching and linear time logic.

2. LAMPORT'S APPROACH AND ITS LIMITATIONS

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

2.1 Definition. A structure $M = (S,X,L)$ where

S is a nonempty set of states,
X is a nonempty set of paths, i.e., a nonempty set of nonempty sequences of states, and
L is a labelling which assigns to each state a set of atomic propositions true in the state.

We use $s,t,s',t_1,\ldots$ etc. to denote states in S and $x,y,x',y_1,\ldots$ etc. to denote sequences of states (with repetitions allowed) over S. A path x is a nonempty sequence of states. We say that a path is of length k, and write $|x| = k$, if it consists of $1 + k$ states. Thus, if x is finite then $|x| = k$ for some $k \geq 0$, and x has the form $(s_0,\ldots,s_k)$. If x is infinite then $|x| = \omega$ and has the form $(s_0,s_1,s_2,\ldots)$. If x can be either finite or infinite it is sometimes convenient to write $x = (s_0,\ldots,s_k,.)$ or even $x = (s_i)$ where, implicitly, $0 \leq i < 1+|x|$. We use first(x) to denote the first state, s_0, of x, and last(x) to denote the last state, s_k, of x. If x is infinite, last(x) does not exist. If $|x| > 0$, we define $x^{\backprime} = (s_1,\ldots,s_k,.)$; otherwise $x^{\backprime} = x$. We define the suffixes of x, $x^0 = x$, $x^{m+1} = (x^m)^{\backprime}$. 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. If x is a finite sequence and y is a sequence, then the concatenation of x and y, written xy, is the sequence obtained by appending y to x. (E.g., if $x = (s_1,s_2)$ and $y = (s_3,s_4,s_5)$ then $xy = (s_1,s_2,s_3,s_4,s_5)$. Similarly, if $x = x's$ is a finite path and $y = sy'$ is a path then the fusion of x and y, written $x*y$, is the path $x'sy'$ (the fusion is undefined if $last(x) \neq first(y)$).

Remark: Various constraints can placed on the set of paths X. In particular, Lamport [LA80] requires that X be suffix closed meaning that if $x \in X$ then $x' \in X$. Similarly, we say that X is fusion closed (cf. [PR79]) if $x_1sy_1 \in X$ and $x_2sy_2 \in X$ imply $x_1sy_2 \in X$, We also say that X is limit closed (cf. [AB80]) provided that if there is an infinite sequence of paths $y_0x_0, y_0y_1x_1, y_0y_1y_2x_2, \cdots \in X$ and each y_i is nonempty then the "limit" path $y_0y_1y_2\cdots \in X$. In the subsequent sections, we shall also consider the case where X is required to be R-generable meaning that there is a (total, nonempty) binary relation R such that X consists precisely of the infinite sequences $(s_0,s_1,s_2,\cdots)$ such that $(s_i,s_{i+1}) \in R$ for all i. This is a natural condition which has been assumed in many previous papers including [FL79], [EC80], [BMP81], and [EH82]. It is shown in [EM81] that the above three closure properties are exactly equivalent to R-generability. These closure properties are important in ensuring that certain commonly accepted identities are valid (see sections 4,5 and [EM81]).

2.2 **Syntax.** Lamport inductively defines the syntax of a class of temporal formulae:

1. Any atomic proposition P is a temporal formula.
2. If p,q are temporal formulae then so are
   $p \wedge q$ ("conjunction"), and $\sim p$ ("negation").
3. If p is a temporal formula then so are
   []p (meaning "always p") and
   ->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 $M,s \models_B p$ to indicate that formula p is interpreted as true in structure M at state s. We define $\models_B$ inductively:

1. $M,s \models_B P$ iff $P \in L(s)$
2. $M,s \models_B p \wedge q$ iff $M,s \models_B p$ and $M,s \models_B q$
   $M,s \models_B \sim p$ iff not($M,s \models_B p$)
3. $M,s \models_B []p$ iff $\forall$ path $x \in X$ with $first(x) = s$
   $\forall n \geq 0$, $M,first(x^n) \models_B p$
   $M,s \models_B ->p$ iff $\forall$ path $x \in X$ with $first(x) = s$
   $\exists n \geq 0$, $M,first(x^n) \models_B p$

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

1. $M,x \models_L P$ iff $P \in L(first(x))$
2. $M,x \models_L p \wedge q$ iff $M,x \models_L p$ and $M,x \models_L q$
   $M,x \models_L \sim p$ iff not $(M,x \models_L p)$
3. $M,x \models_L []p$ iff $\forall n \geq 0$, $M,x^n \models_L p$
   $M,x \models_L ->p$ iff $\exists n \geq 0$, $M,x^n \models_L p$

For both interpretations, the modality <>p is introduced as an abbreviation for $\sim[]\sim p$ and the other logical connectives are introduced as abbreviations in the usual way.

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:

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

Next, Lamport defines his notion of equivalence:

2.5 **Definition.** Formula p under interpretation X is strongly equivalent to formula q under interpretation Y, written $p \equiv_s q$, provided that for every structure M, $M \models_X p$ iff $M \models_Y q$

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

2.6 **Theorem ([LA80]).** <>P in branching time is not strongly equivalent to any assertion of linear time.

2.7 **Theorem ([LA80]).** ->[]P in linear time is not strongly equivalent to any assertion of branching time.

We have several criticisms of the formalism. Note that defining a formula as true or false of an entire model causes useful information to be lost. For example, in the branching time interpretation although there is a model M with states s,s' such that $M,s \models_B ->P \wedge \sim P$ and $M,s' \models_B \sim(->P \wedge \sim P)$, there is no model M such that $M \models_B ->P \wedge \sim P$. Similar remarks apply for the linear time interpretation. Thus, we get

2.8 **Proposition.** In linear time or in branching time, $->P \wedge \sim P \equiv_s$ false.

This shows that $\equiv_S$ is too coarse an equivalence relation in that it classifies satisfiable formulae as equivalent to false. Moreover, since the same notation is used for both branching and linear time formulae, it is not clear from the syntax which interpretation is intended. This has the effect of obscuring the essential difference between the two interpretations, namely, that linear time formulae make assertions about paths and branching time formulae make assertions about states. It also causes difficulties when translating from English into the formalism.

We also disagree with Lamport's conclusion that linear time logic is superior to branching time logic for reasoning about concurrent programs. Lamport gives two specific arguments to justify this claim:

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 (->[] ~ENABLED) $\vee$ ->EXECUTED. However, 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 ->P $\vee$ []~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 appropriate operators (such as the "infinitary" quantifiers used in [EC80]) 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 section 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, it is possible to write a formula in a branching time system which accurately renders the principle as shown in section 3.

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*, which subsumes Lamport's branching and linear time systems as well as UB [BMP81] and CTL ([EH82], [CE81]). CTL* is closely related to MPL [AB80]. (CTL* is also used in [CES83].) 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 [EC80], $\overset{\infty}{F}$ ("infinitely often"), $\overset{\infty}{G}$ ("almost everywhere").

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

S1. Any atomic proposition P is a state formula.
S2. If p,q are state formulae
then so are p $\wedge$ q, ~p
S3. If p is a path formula
then Ap, Ep are state formulae
P1. Any atomic proposition P is a path formula
P2. If p,q are path formulae
then so are p $\wedge$ q, ~p
P3a. If p is a state formulae
then Gp, Fp are path formulae
P3b. If p is a path formulae
then Gp, Fp are path formulae
P4a. If p,q are state formulae
then Xp, (p U q) are path formulae
P4b. If p,q are path formulae
then Xp, (p U q) are path formulae
P5a. If p is a state formula
then $\overset{\infty}{F}$p, $\overset{\infty}{G}$p are path formulae
P5b. If p is a path formula
then $\overset{\infty}{F}$p, $\overset{\infty}{G}$p are path formulae

Remark: The other truth-functional connectives are introduced as abbreviations in the usual way. As we shall see, we could take the view that Ap

abbreviates ~E~p, Fp abbreviates (true U p), Gp abbreviates ~F~p, $\overset{\infty}{F}$p abbreviates G(Xtrue $\wedge$ Fp), and $\overset{\infty}{G}$p abbreviates ~$\overset{\infty}{F}$~p. 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,~, and $\wedge$. Also, we could consider state formulae as a special case of path formulae whose truth value depends on the first state of the path and thus view all formulae as path formulae. This is essentially what is done in PL (cf. [HKP80]) and also leads to a slightly easier formulation of the syntax and semantics. However, like Abrahamson [AB80] , 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 formulae generated by the above rules forms the language CTL*. We also consider a number of other languages generated by some combination of the above rules: The set of path formulae generated by rules P1,2,3b gives the language L(F,G), and the set of state formulae generated by rules S1-3,P3a yields the language BT. As we shall see, L(F,G) corresponds precisely to Lamport's linear time interpretation and BT corresponds precisely to Lamport's branching time interpretation. The set of path formulae generated by rules P1,2,3b,4b corresponds to the language L(F,G,X,U) used in many applications (cf. [GPSS80], [MW81]). The set of state formulae generated by rules S1-3,P3a,4a corresponds to the language CTL used in [CE81]. We define the language ECTL to be the set of state formulae generated by rules S1-3,P3a,4a,5. We can then define BT^+, CTL^+, and $ECTL^+$ to be the set of state formulae generated by adding the rule P2 to the rules for BT, CTL, and ECTL, respectively. CTL^+ was considered in [EH82] and $ECTL^+$ is essentially the language studied in [EC80]. Both ECTL and $ECTL^+$ provide us with an ability to make assertions about fair computations.

3.2 Semantics. We write M,s |= p (M,x |= p) to mean that state formula p (path formula p) is true in structure M at state s (of path x, respectively). When M is understood, we write simply s |= p (x |= p). We define |= inductively:

S1. s |= P iff P $\in$ L(s)
S2. s |= p $\wedge$ q iff s |= p and s |= q
    s |= ~p iff not (s |= p)
S3. s |= Ap iff for every path x $\in$ X with first(x) = s, x |= p
    s |= Ep iff for some path x $\in$ X with first(x) = s, x |= p
P1. x |= P iff P $\in$ L(first(x))
P2. x |= p $\wedge$ q iff x |= p and x |= q
    x |= ~p iff not (x |= p)
P3a. x |= Gp iff for all i $\geq$ 0, first(x^i) |= p
     x |= Fp iff for some i $\geq$ 0, first(x^i) |= p
P3b. x |= Gp iff for all i $\geq$ 0, x^i |= p
     x |= Fp iff for some i $\geq$ 0, x^i |= p
P4a. x |= Xp iff |x| > 0 and first(x^1) |= p
     x |= (p U q) iff for some i $\geq$ 0, first(x^i) |= q and for all j $\geq$ 0 [j $\leq$ i implies first(x^j) |= p]
P4b. x |= Xp iff |x| > 0 and x^1 |= p
     x |= (p U q) iff for some i $\geq$ 0, x^i |= q and for all j $\geq$ 0 [j $\leq$ i implies x^j |= p]
P5a. x |= $\overset{\infty}{F}$p iff |x| = ω and for infinitely many distinct i, first(x^i) |= p
     x |= $\overset{\infty}{G}$p iff for all but a finite number of i, first(x^i) |= p
P5b. x |= $\overset{\infty}{F}$p iff |x| = ω and for infinitely many distinct i, x^i |= p
     x |= $\overset{\infty}{G}$p iff for all but a finite number of i, x^i |= p

It is easy to check that all the equivalences mentioned in the remark in section 3.1 hold. Observe that the following equivalences establish the claimed correspondences between Lamport's linear time and L(F,G) and between Lamport's branching time and BT:

M,x |=$_L$ []p iff M,x |= Gp
M,x |=$_L$ ->p iff M,x |= Fp
M,s |=$_B$ []p iff M,s |= AGp
M,s |=$_B$ ->p iff M,s |= AFp

Note that under the linear time interpretation the formula discussed in the previous section, ->P $\vee$ []~P, corresponds to the L(F,G) formula FP $\vee$ G~P which is clearly valid. Under the branching time interpretation it corresponds to AFP $\vee$ AG~P which is not valid. However, the valid BT^+ formula A(FP $\vee$ G~P) (obtained by simply prefixing the L(F,G) formula with A) does capture the intended principle.

Clearly, a direct comparison of linear time (i.e. path) formulae with branching time (i.e. state) formulae is impossible. As we have seen, Lamport's approach of defining a formula as true or false of an entire structure leads to technical problems. How then can we compare linear time with branching time? Since in program verification applications, there is an implicit universal quantification over all possible futures when a linear time assertion is used, 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:

3.3 Definition Given any language L of path formulae we define the language of associated state formulae B(L) = {Ap : p $\in$ L}. (Note that B(L) is not closed under negation or disjunction (cf. [AB80]).)

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 B(L). This time, however, equivalence of the branching time formulae is measured by the "usual" notion:

3.4 Definition. Given state formulae p,q we say that p is equivalent to q, written p $\equiv$ q, provided that for every structure M, for every state s of M, M,s |= p iff M,s |= q.

It is easy to check that $\equiv$ is an equivalence relation which refines $\equiv_s$ and avoids the problems of Proposition 2.8. In fact, we have the following results which clarify the relation between $\equiv$ and $\equiv_s$:

3.5 Proposition. For any path formula p, p $\equiv_s$ Ap.

Proof: Let M = (S,X,L) be an arbitrary structure. We show M |= p iff M |= Ap. If M |= p then for all x $\in$ X, M,x |= p. So for all s $\in$ S, M,s |= Ap and thus M |= Ap. Conversely, if M |= Ap then for all s $\in$ S, M,s |= Ap and for all x $\in$ X starting at s, M,x |= p. Since each x $\in$ X starts at some s S, M,x |= p for all x $\in$ X. Thus, M |= p. []

3.6 Proposition. For any state formulae p,q, p $\equiv_s$ q iff AGp $\equiv$ AGq.

Proof: (=>:) Assume p $\equiv$ q. It will suffice to show that M,s |= AGp implies M,s |= AGq because, by a symmetric argument, we can then conclude AGp $\equiv$ AGq. So suppose M,s |= AGp where M = (S,X,L) is an arbitrary structure and s $\in$ S. Define X' = {x $\in$ X : x starts at s}. If X' is empty, then trivially M,s |= AGq as desired. Otherwise, define M' = (S',X',L') with S' = {s' $\in$ S : s' appears on some x' $\in$ X'} and L' = L|S. Note that for any state formula r, M,s |= AGr iff M',s |= AGr iff $\forall$s' $\in$ S', (M',s' |= r). Taking r=p, we get $\forall$s' $\in$ S', M',s' |= p. Since p $\equiv_s$ q, $\forall$s' $\in$ S', we have M',s' |= q. Now take r=q, to see that M,s |= AGq as desired.

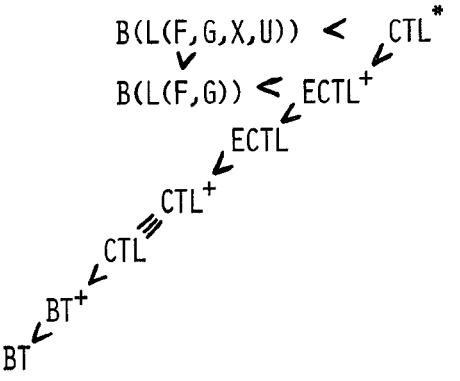
(<=:) Assume AGp $\equiv$ AGq, i.e. M,s |= AGp iff M,s |= AGq for all M and s in M. It will suffice to show that M |= p implies M |= q as a symmetric argument will yield p $\equiv_s$ q. Now suppose M |= p where M = (S,X,L). Then $\forall$s $\in$ S, we have M,s |= p whence $\forall$s $\in$ S, we also have M,s |= AGp. Since AGp $\equiv$ AGq, $\forall$s $\in$ S, M,s |= AGq and M,s |= q. Thus M |= q as desired. []

3.7 Corollary. For any path formula p and state formula q, p $\equiv_s$ q iff AGAp $\equiv$ AGq.

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

3.8 Definition. We say that L_2 is at least as expressive as L_1, written $L_1 \leq L_2$, provided that for every p $\in$ L_1 there exists q $\in$ L_2 such that p $\equiv$ q. We say that L_1 is exactly as expressive as L_2, written $L_1 \equiv L_2$, provided $L_1 \leq L_2$ and $L_2 \leq L_1$. Finally, L_1 is strictly less expressive than L_2, written $L_1 < L_2$, provided $L_1 \leq L_2$ and $L_1 \not\equiv L_2$.

Using this formalism, in the next section we compare the relative expressive power of the branching time languages defined above. We show that the following picture describes their relative expressive power:



where any two languages not connected by a chain of <'s and $\equiv$'s are of incomparable expressive power.

4. EXPRESSIVENESS RESULTS

In proving our expressibility results, we assume that all structures are R-generable. Without such an assumption even rudimentary equivalences such as EFEFp $\equiv$ EFp do not necessarily hold. Our "inexpressibility" results are stronger than those Lamport obtains in that ours apply in the case of R-generable structures as well as suffix-closed structures whereas his apply only to suffix-closed structures.

Our first result shows that Lamport's linear time system is expressible in the branching time logic $ECTL^+$:

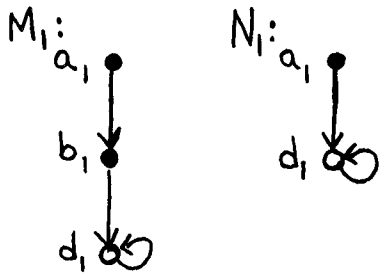
4.1 **Theorem.** $B(L(F,G)) \leq ECTL^+$.

Proof: This proof involves a complicated induction on the structure of B(L(F,G)) formulae. Details are left to the appendix . □

However, if we add the nexttime operator the situation changes:

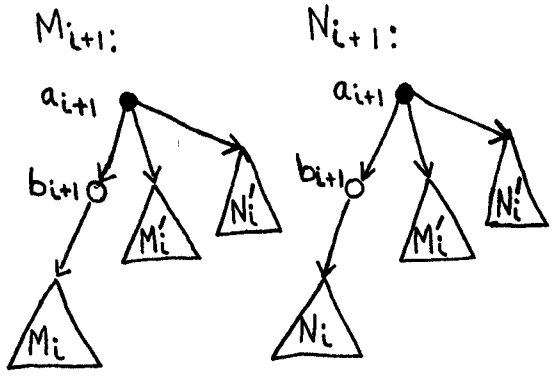
4.2 **Theorem.** The formula A[F(P ∧ XP)] is not equivalent to any formula $q \in ECTL^+$.

Proof: We inductively define two sequences $M_1,M_2,M_3,\ldots$ and $N_1,N_2,N_3,\ldots$ of models as follows. Define M_1,N_1 to have the graphs shown below



where in M_1, $a_1 \models P$, $b_1 \models P$, $d_1 \models \neg P$ and in N_1, $a_1 \models P$, and $d_1 \models \neg P$.

Suppose we have defined M_i and N_i. Then M_{i+1} and N_{i+1} have the following graphs



where in both M_{i+1} and N_{i+1}, $a_{i+1} \models P$, $b_{i+1} \models \neg P$, and M'_i, N'_i are copies of M_i, N_i, respectively.

It should be clear that

(1) for all i, $M_i,a_i \models A[F(P \wedge XP)]$ and $N_i,a_i \models \neg A[F(P \wedge XP)]$.

We will also show that

(2) For any $ECTL^+$ formula p there is a CTL formula q which is equivalent to p over these two sequences of models. That is, for all i and all states s in M_i, $M_i,s \models p \equiv q$, and similarly for N_i.

(3) For any CTL formula p, with $|p| < i$, $M_i,a_i \models p$ iff $N_i,a_i \models p$.

To see that the result follows, suppose that A[F(P ∧ XP)] is equivalent to some $ECTL^+$ formulae p. Then by (2) above, there is a CTL formula p′ equivalent to p over these models. Now $|p'| = i$ for some i. Then $M_i,a_i \models A[F(P \wedge XP)]$ which, by supposition and (2), implies $M_i,a_i \models p'$. By (3) this implies $N_i,a_i \models p'$, which implies, again by supposition and (2), that $N_i,a_i \models A[F(P \wedge XP)]$. But this contradicts the fact (1) above that $N_i,a_i \models \neg A[F(P \wedge XP)]$.

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

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

4.3 **Theorem.** The $ECTL^+$ formula $E[\overset{\infty}{F}P \wedge \overset{\infty}{F}Q]$ is not equivalent to any formula $q \in ECTL$.

Proof: Left to the appendix. □

4.4 **Theorem.** The ECTL formula $\overset{\infty}{EF}P$ is not equivalent to any formula $q \in CTL^+$.

Proof: Left to the full paper. □

Theorem 4.4 also follows from the results of [EC80] which depend on recursion-theoretic techniques. However, such techniques will not suffice to establish Theorem 4.3. Thus, the combinatorial proof techniques used here seem to provide a sharper tool than does recursion theory in applications such as this.

The following theorem shows that existential quantification over paths cannot be expressed with only universal quantification as provided in the languages of the form B(L(-)):

4.5 **Theorem.** The BT formula EFP is not equivalent to any B(L(F,G,X,U)) formula.

Proof: Suppose EFP ≡ Aq for some linear time formula q over F,G,X,U. Since ¬EFP is satisfiable, it must be that ¬q is satisfiable. Thus $M,x \models \neg q$ for some structure M = (S,X,L) with X = {x}. Add a successor s′ to s = first(x) which satisfies P to get a new structure. That is, let M′ = (S′,X′,L′) where $S' = S \cup \{s'\}$, $X' = X \cup \{ss'\}$, L′(s) =

L(s) for $s \in S$ and $L'(s') = \{P\}$. Then $M',s \models EFP$ and, by the supposed equivalence, $M',s \models Aq$. But then $M',x \models q$ and also $M,x \models q$, a contradiction. □

The portion of the diagram below CTL^+ follows from the results in [EH82].

5. RELATION TO PL AND MPL

We assume that the reader is familiar with PL (see [HKP80] for details). We can translate CTL^* into PL in the following way: To each CTL^* structure $M = (S,X,L)$, we associate the PL structure $M^t = (S,\models,R)$ where the set of paths of atomic program A, R_A, is equal to X, and for any atomic proposition P, $M^t,s \models P$ iff $M,s \models P$. We can then give a translation of a CTL^* formula p into an equivalent PL formula p^t. We define the translation inductively, taking the primitive temporal connectives of CTL^* to be E, X, and U (c.f. the remark in Section 3.1):

$P^t = P$ for atomic propositions P
$(\sim p)^t = \sim(p^t)$
$(p \wedge q)^t = p^t \wedge q^t$
$(p\ U\ q)^t = q^t \vee (p^t\ \underline{suf}\ q^t)$
$(Ep)^t = \underline{f}(\langle A\rangle p^t)$
$(Xp)^t = \underline{false}\ \underline{suf}\ p^t$
(Note this is equivalent to $(\underline{n}p^t)$)

Then by a straightforward induction on the structure of CTL^+ formulae we can show

5.1 Proposition. For all $x \in X$, $M,x \models p$ iff $M^t,x \models p^t$ and for all $s \in S$, $M,s \models p$ iff $M^t,(s) \models p^t$.

Note $(p\ U\ q)^t \equiv q^t$ or $(p^t\ \underline{suf}\ q^t)$ since the U operator considers the current path while the $\underline{suf}$ operator only depends on proper suffixes. Ep is a state formula; since in PL we have only path formulae, 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 [AB80] for more details). To simplify the exposition, we take the liberty of slightly altering Abrahamson's notation. In particular, we use the temporal connectives <>, U, and X instead of their duals [], W, and Y, respectively. We also omit the H operator and view all paths as simply 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 formulae then so are ~p, $p \wedge q$, <>p, Xp, and p U q.

We take []p to be an abbreviation for ~<>~p.

A structure M is a triple (S,X,L) as before. An MPL formula is true or false of a triple M,x,y where M is a structure (S,X,L), $x \in X$, 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. $M,x,y \models P$ iff $P \in L(last(y))$
2. $M,x,y \models p \wedge q$ iff $M,x,y \models p$ and $M,x,y \models q$

   $M,x,y \models \sim p$ iff not$(M,x,y \models p)$
3. $M,x,y \models p\ U\ q$ iff $\exists z(y \leq z \leq x$ and $M,x,z \models q$ and $\forall w(y \leq w \leq z \Rightarrow M,x,w \models p))$
   $M,x,y \models Xp$ iff $\exists z(M,z,x \models p$ and $y \leq z \leq x$ and $\sim\exists w(y < w < z))$
4. $M,x,y \models \langle\rangle p$ iff $\exists x'(x' \in X,\ y \leq x'$, and $M,x',y' \models p)$

While no restrictions are placed on the set of paths X in defining the semantics of MPL, we must restrict our attention to structures that are suffix closed as well as fusion closed in order to translate CTL^* into MPL. These restrictions are necessary since there are CTL^* formulae (e.g., $EGX\underline{true} \wedge \sim EXEGX\underline{true} \wedge EFEFp \wedge \sim EFp$) which are satisfiable only in structures that are neither suffix closed nor fusion closed 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 and is proved in

5.2 Lemma. An MPL formula is satisfiable iff it is satisfiable in a structure that is suffix closed and fusion closed.

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, i.e. $y \cdot (x/y) = x$. Then we get

5.3 Lemma. If $M = (S,X,L)$ and X is suffix closed and fusion closed then for all MPL formulae p and $x \in X$

$M,x,y \models p$ iff $M,x/y, first(x/y) \models p$

Proof: A straightforward induction on the structure of p suffices. Note that we need fusion closure of X in order to show that M,x/y,first(x/y) |= <>p implies M,x,y |= <>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 M,x |= p as an abbreviation for M,x,first(x) |= p. We can then translate a CTL* formula p to an MPL formula $p^{t'}$ simply by replacing all occurrences of E by <>. We now get:

5.4 **Theorem.** Given a structure M = (S,X,L) where X is suffix closed and fusion closed, and path x ∈ X,

if p is a CTL* path formula then
    M,x |= p iff M,x |= $p^{t'}$ and
if p is a CTL* state formula then
    M,first(x) |= p iff M,x |= $p^{t'}$.

What do these translations tell us about the existence of decision procedures for CTL*?. Note that a CTL* formula p is satisfiable in a structure M iff the corresponding PL formula p^t is satisfiable in the structure M^t. Moreover, by the definition of M^t in terms of M, p is satisfiable in a structure M meeting certain restrictions on its set of paths X (e.g., suffix closure or R-generability) iff iff p^t is satisfiable in a structure M^t where R_A meets the same restrictions. However, the definition of PL allows arbitrary sets of paths in the structure, and the original work [HKP80] on decidability of PL formula does not consider the question of restrictions on the sets of paths. It is true that we can modify the algorithm given in [HA82] to check if a formula is satisfiable in a structure where R_A is suffix closed and/or fusion closed. Alternatively, using Lemmas 5.2 and 5.3 and Theorem 5.4 together with the fact ([AB80]) that there is an algorithm for testing satisfiability of MPL formulae which runs in deterministic time $O(2^{2^{cn}})$, we get

5.5 **Theorem** There is an algorithm to decide if a CTL* formula p is satisfiable in a structure which is suffix closed, fusion closed, and for which every state is the first state of some path that runs in time $O(2^{2^{|p|}})$.

Proof: We simply check if $p^{t'}$ is satisfiable.

But, the problem of deciding if a formula is satisfiable in an R-generable structure seems much harder. It is the limit closure constraint which causes difficulties. For instance, there is no analogue of lemma 5.2 for limit closure in the case of MPL. Consider the formulas <>GXtrue and []G<>Xtrue given by Abrahamson ([AB80], p. 110). If M = (S,X,L) then M,x,y |= <>GXtrue iff there is an infinite path in X extending y and M,x,y |= []G<>Xtrue iff every finite path extending y can in turn be extended by another path in X. It is easy to check that ~<>GXtrue ∧ []G<>Xtrue is satisfiable in a structure that is suffix closed and fusion closed but not limit closed. However, as Abrahamson points out, his decision procedure will generate structures that are not limit closed for certain formulae satisfiable in limit closed structures, and there is no obvious modification of his algorithm for MPL to force it to generate limit closed models whenever possible. Similarly, there is no obvious modification of the [HA82] algorithm to force R_A to be limit closed. Thus, the problem of finding an elementary time algorithm to decide if the CTL* formula p is satisfiable in a limit closed (or R-generable model) remains open.

We remark that in [AB80] a complete axiomitization is given for MPL which also applies to CTL*, provided we restrict our attention to structures which are suffix closed and fusion closed. The problem of finding a complete axiomatization which applies to R-generable structures remains open.

6. CONCLUSION

We believe that linear time logics are generally adequate for verifying the correctness of pre-existing concurrent programs. For 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 [OL80] 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, and we see only one advantage in considering the use of a branching time logic. Namely, linear time logics, as interpreted over branching time structures, are not closed under negation. While 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. As Abrahamson [AB80] notes "It is out of the question to attempt to disprove a property when we can't even state its negation."

Furthermore, there are other 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 $EG\underline{in}NCS_i \wedge EF\underline{in}TRY_i \wedge A(G\underline{in}NCS_i \vee F\underline{in}TRY_i)$. However, using Theorem 4.5, this is provably not expressible in linear time logic, i.e., in a language of the form B(L(-)). The natural candidate formula, $A(G\underline{in}NCS_i \vee F\underline{in}TRY_i)$, allows a "degenerate" model where all paths satisfy $F\underline{in}TRY_i$ and no path satisfies $G\underline{in}NCS_i$.

This ability to existentially quantify over paths is particularly useful in applications such as automatic program synthesis from temporal logic specifications (cf. [CE81], [EC82]) where very precise and thorough specifications are needed. Of course, it is possible to successfully synthesize a wide class of interesting programs using only linear time logic (cf. [MW81], [WO82]); 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).

7. APPENDIX

Proof of Theorem 4.1: We first define the set of basic formulae, 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 formulae then $p_1 \; U \; (p_2 \; U \; \ldots \; (p_{n-1} \; U \; Gp_n)\ldots)$ 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 $[p_0^0,\ldots,p_{n_0}^0],\ldots,[p_0^m,\ldots,p_{n_m}^m],[q_0^0,\ldots,q_{k_0}^0]$, $Fr_1,\ldots,Fr_n$ are B formulae, then $F(p \wedge [p_0^0,\ldots,p_{n_0}^0] \wedge\ldots\wedge [p_0^m,\ldots,p_{n_m}^m] \wedge X[q_0^0,\ldots,q_{k_0}^0] \wedge Fr_1 \wedge\ldots\wedge Fr_n)$ 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 formulae of B^+ can be written in conjunctive or disjunctive normal form, where the literals are formulae 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. $\sim Fp \equiv G\sim p$) to drive the negations inward until only the atomic propositions appear negated. Since B^+ contains the propositional formulae 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 formulae 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 formulae in B. This follows using the observation below (where p′ and the q'_k are propositional formulae):

$$G(p' \vee \vee_i [p_0^i,\ldots,p_{n_i}^i] \vee \vee_j Fq_j \vee \vee_k GFq'_k)$$
$$\equiv$$
$$Gp' \vee \vee_i [p',p_0^i,\ldots,p_{n_i}^i] \vee \vee_k GFq'_k \vee \vee_j GFq_j \vee \vee_{i,j} [F(q_j \wedge XGp') \vee F(q_j \wedge X[p',p_0^i,\ldots,p_{n_i}^i]$$

Intuitively, the first line of the right hand side takes care of the case that no q_j is ever true, and the second line covers the case that some q_j is true infinitely often. The third line corresponds to all q_j being true only finitely often: the last time any q_j is true, either Gp′ or one of the $[p_0^i,\ldots,p_{n_i}^i]$ will be true at the next state. This would be a B^+ formula except that $q \equiv q_j$ 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:

$$GF(p \wedge [p_0^0,\ldots,p_{n_0}^0] \wedge\ldots\wedge [p_0^m,\ldots,p_{n_m}^m] \wedge X[q_0^0,\ldots,q_{k_0}^0] \wedge Fr_1 \wedge\ldots\wedge Fr_n)$$
$$\equiv$$
$$GFp \wedge F([p_0^0,\ldots,p_{n_0}^0]) \wedge\ldots\wedge F([p_0^m,\ldots,p_{n_m}^m]) \wedge F([q_0^0,\ldots,q_{k_0}^0]) \wedge GFr_1 \wedge\ldots\wedge GFr_n$$

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 an $ECTL^+$ 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 formulae. We proceed by induction on the number of formula of the form Fr (corresponding to rule 4 above) which appear as any subformula in p.

If p has no F's, then it is of the form $q \wedge [p_0^0,\ldots,p_{n_0}^0] \wedge\ldots\wedge [p_0^m,\ldots,p_{n_m}^m] \wedge \wedge_i GFr_i$ where q is propositional. We first show that a conjunction of formulae of the form $[p_0,\ldots,p_n]$ is equivalent to a disjunction of such formulae. 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

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

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

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

This is an $ECTL^+$ formula as desired since $GF \equiv F^{\infty}$.

In general, p has the form

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

where q is propositional. Observe that

$$Ep \equiv q \wedge \vee_{i,j} E[p_0 \,U\, E[p_1 \,U\, \ldots E[p_j \,U\, E[p_j,\ldots,p_n] \wedge r_i \wedge \wedge_{k\neq i} Fr_k \wedge \wedge_h GFs_h]\ldots]]$$

(Intuitively, we are disjuncting over which r_i gets satisfied first and in which segment p_j this occurs.) $E([p_j,\ldots,p_n] \wedge r_i \wedge \wedge_{k\neq i} Fr_k \wedge \wedge_h GFs_h)$ has one fewer F so we are almost ready to apply the induction hypothesis. Only one problem remains: one of the conjuncts forming r_i might be a formula of the form $X[q_0,\ldots,q_m]$. Note that the following equivalences hold:

$$GFs \equiv XGFs$$
$$[p_0,\ldots,p_n] \equiv \vee_j (p_j \wedge X[p_j,\ldots,p_n])$$

$$Fr_j \equiv r_j \vee XFr_j$$
$$Xq_1 \wedge Xq_2 \equiv X(q_1 \wedge q_2).$$

By repeatedly applying these equivalences, we can rewrite $[p_j,\ldots,p_n] \wedge r_i \wedge \wedge_{k\neq i} Fr_k \wedge \wedge_h GFs_h$ as a disjunction of formulae of the form $p' \wedge Xq'$ where p′ is a propositional formula and q′ is a B^+ formula. But $E(p' \wedge Xq') \equiv p' \wedge EXEq'$, and q′ will have less F's than the original p. By induction, we are done. □

Proof of Theorem 4.2 (continued:) We now argue by induction on $|p|$, that for all CTL formulae p,

(*) if $|p| \leq i$ then $(M_i,a_i \models p$ iff $N_i,a_i \models p)$.

Note that (*) trivially implies if $|p| \leq i$ then

($M_{i+1},a_i \models p$ iff $N_{i+1},a_i \models p$) which in turn can be seen to imply

(**) if $|p| \leq i$ then ($M_{i+1},b_{i+1} \models p$ iff $N_{i+1},b_{i+1} \models p$).

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

If p is of the form EXq then,

$M_{i+1},a_{i+1} \models EXq$

iff

$M_{i+1},b_{i+1} \models q$ or $M_i,a_i \models q$ or $N_i,a_i \models q$

iff

$N_{i+1},b_{i+1} \models q$ (by **) or $M_i,a_i \models q$ or $N_i,a_i \models q$

iff

$N_{i+1},a_{i+1} \models EXq$.

If p = E[q U r] then,

$M_{i+1},a_{i+1} \models E[q\ U\ r]$ iff

(1) $M_{i+1},a_{i+1} \models r$ or
(2) $M_{i+1},a_{i+1} \models q$, $M_{i+1},b_{i+1} \models r$ or
(3) $M_{i+1},a_{i+1} \models q$, $M_{i+1},b_{i+1} \models q$, $M_i,a_i \models E[q\ U\ r]$ or
(4) $M_{i+1},a_{i+1} \models q$, $M_i,a_i \models E[q\ U\ r]$ or
(5) $M_{i+1},a_{i+1} \models q$, $N_i,a_i \models E[q\ U\ r]$

iff

(1) $N_{i+1},a_{i+1} \models r$ (by (*)) or
(2) $N_{i+1},a_{i+1} \models q$, $N_{i+1},b_{i+1} \models r$ (by (*), (**) resp.) or
(3) $N_{i+1},a_{i+1} \models q$, $N_{i+1},b_{i+1} \models q$ (by (*), (**) resp.), $M_i,a_i \models E[q\ U\ r]$ or
(4) $N_{i+1},a_{i+1} \models q$ (by (*)), $M_i,a_i \models E[q\ U\ r]$ or
(5) $N_{i+1},a_{i+1} \models q$ (by (*)), $N_i,a_i \models E[q\ U\ r]$

iff

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

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

$M_{i+1},a_{i+1} \models A[q\ U\ r]$ iff

(1) $M_{i+1},a_{i+1} \models r$ or
(2) $M_{i+1},a_{i+1} \models q$, $M_{i+1},b_{i+1} \models r$, $M_i,a_i \models A[q\ U\ r]$, $N_i,a_i \models A[q\ U\ r]$ or
(3) $M_{i+1},a_{i+1} \models q$, $M_{i+1},b_{i+1} \models q$, $M_i,a_i \models A[q\ U\ r]$, $N_i,a_i \models A[q\ U\ r]$

iff

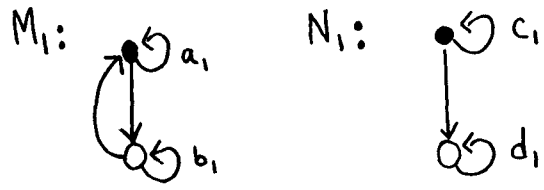
(1) $N_{i+1},a_{i+1} \models r$ (by (*)) or
(2) $N_{i+1},a_{i+1} \models q$, $N_{i+1},b_{i+1} \models r$ (by (**)), $M_i,a_i \models A[q\ U\ r]$, $N_i,a_i \models A[q\ U\ r]$ or
(3) $N_{i+1},a_{i+1} \models q$, $N_{i+1},b_{i+1} \models q$ (by (**)), $M_i,a_i \models A[q\ U\ r]$, $N_i,a_i \models A[q\ U\ r]$

iff

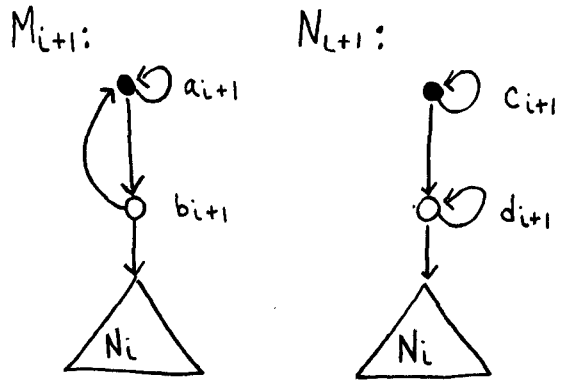
$N_{i+1},a_{i+1} \models A[q\ U\ r]$.

It remains to establish our claim that CTL and ECTL$^+$ are of equivalent expressive power on the two sequences of models. For any ECTL$^+$ 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 ECTL$^+$ formula of the form $p_1 = E[p \wedge \overset{\infty}{F} q_1 \wedge \ldots \wedge \overset{\infty}{F} q_n \wedge \overset{\infty}{G} r]$ where $Ep,q_1,\ldots,q_n$ are CTL$^+$ formulae. To show this, we observe that every maximal path in one of the structures M_i or 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 $M_i,c \models p_1$ iff $M_i,c \models Ep$ and $M_i,d \models q_1 \wedge \ldots \wedge q_n \wedge r$. Moreover, $M_i,d \models q_1 \wedge \ldots \wedge q_n \wedge r$ iff $M_i,c \models EFAG(q_1 \wedge \ldots \wedge q_n \wedge r)$. Thus, $M_i,c \models p_1$ iff $M_i,c \models Ep \wedge EFAG(q_1 \wedge \ldots \wedge q_n \wedge r)$ and similarly, for N_i. The latter formula is in CTL$^+$. By [EH82], we know that for any CTL$^+$ formula q' there exists an equivalent CTL formula q. So we are done. □

Proof of Theorem 4.3: We inductively define two sequences of models $M_1,M_2,M_3,\ldots$ and $N_1,N_2,N_3,\ldots$ such that for all i, $M_i,a_i \models E[\overset{\infty}{F}p \wedge \overset{\infty}{F}Q]$ and $N_i,c_i \models \sim E[\overset{\infty}{F}P \wedge \overset{\infty}{F}Q]$. We show that ECTL is unable to distinguish between the two sequences of models, i.e. for all ECTL formulae p with $|p| \leq i$, $M_i,a_i \models p$ iff $N_i,c_i \models p$. The result follows since if $E[\overset{\infty}{F}p \wedge \overset{\infty}{F}Q]$ were equivalent to some ECTL formula p' of length i then we would get a contradiction: $M_i,a_i \models p'$ iff $N_i,c_i \models p'$ while $M_i,a_i \models E[\overset{\infty}{F}p \wedge \overset{\infty}{F}Q]$ and $N_i,c_i \models \sim E[\overset{\infty}{F}P \wedge \overset{\infty}{F}Q]$. We define M_1, N_1 to have the graphs shown below:



where $a_1 \models P \wedge \neg Q$; $b_1 \models \neg P \wedge Q$, $c_1 \models P \wedge \neg Q$, and $d_1 \models \neg P \wedge Q$. Assume we have defined M_i, N_i. Then M_{i+1}, N_{i+1} have the graphs below:



where $a_{i+1} \models P \wedge \neg Q$, $b_{i+1} \models \neg P \wedge Q$, $c_{i+1} \models P \wedge \neg Q$, and $d_{i+1} \models \neg P \wedge Q$.

Details are given in the full paper. □

Proof of Lemma 5.2: Suppose $M = (S,X,L)$ and $M,x,y \models p$. Let $X_1 = \{z \in X \mid y \leq z\}$ and $M_1 = (S,X_1,L)$. It is easy to check that $M_1,x,y \models p$. To simplify the notation, assume for now that X_1 is countable (the case where 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 M_1 into a "tree-like" model. Define a set $T = \{t_{ij} \mid i,j \geq 0\}$ of "fresh" states distinct from S. We will inductively define a set of paths $X' = \{y_0,y_1,y_2,\ldots\}$ over T which is fusion closed along with a mapping $h:T \rightarrow S$ as follows: Suppose $x_0 = (s_0,s_1,\ldots,s_k,.)$ (which could be finite or infinite). Then define $y_0 = (t_{00},t_{01},\ldots t_{0k},.)$ and $h(t_{0j}) = s_j$ for all j. We can extend h so that if $y = (u_0,\ldots,u_m,.)$ then $h(y) = (h(u_0),\ldots,h(u_m),.)$. 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 T' consist of those states of T which occur in y_i for some i. Also let L' be a labelling of states in T' such that $L'(t) = L(h(t))$. Now define $M' = (T',X',L')$. Then we can show, by a straightforward induction on the structure of formulae, that for any formula q, if $|w| \geq |y|$ then $M',z,w \models q$ iff $M,h(z),h(w) \models q$.

Next define $X'' = \{x^i \mid x \in X'\}$. 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 X" is fusion closed and suffix closed. Let $M'' = (T',X'',L')$. Then we can argue by induction on the length of formulae q, that for $x \in X'$, $M',x,y \models q$ iff $M'',x,y \models q$. Thus, M" is a fusion closed and suffix closed model of p.

If 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). □

8. REFERENCES

[AB80] Abrahamson, K., Decidability and Expressiveness of Logics of Processes, PhD Thesis, Univ. of Washington, 1980.

[BMP81] Ben-Ari, M., Manna, Z., and Pnueli, A., The Temporal Logic of Branching Time. 8th Annual ACM Symp. on Principles of Programming Languages, 1981.

[CE81] Clarke, E. M., and Emerson, E. A., Design and Synthesis of Synchronization Skeletons using Branching Time Temporal Logic, Proceedings of the IBM Workshop on Logics of Programs, Springer-Verlag Lecture Notes in Computer Science #131, 1981.

[CES83] Clarke, E. M., Emerson, E. A., and Sistla, A. P., Automatic Verification of Finite State Concurrent Programs: A Practical Approach, this POPL conference, 1983.

[EC80] Emerson, E. A., and Clarke, E. M., Characterizing Correctness Properties of Parallel Programs as Fixpoints. Proc. 7th Int. Colloquium on Automata, Languages, and Programming, Lecture Notes in Computer Science #85, Springer-Verlag, 1981.

[EC82] Emerson, E. A., and Clarke, E. M., Using Branching Time Logic to Synthesize Synchronization Skeletons, Tech. Report TR-208, Univ. of Texas, 1982. (to appear in SCP)

[EH82] Emerson, E. A., and Halpern, J. Y., Decision Procedures and Expressiveness in the Temporal Logic of Branching Time. 14th Annual ACM Symp. on Theory of Computing, 1982.

[EM81] Emerson, E. A., Alternative Semantics for Temporal Logics, Tech. Report TR-182, Univ. of Texas, 1981. (To appear in TCS)

[FL79] Fischer, M. J., and Ladner, R. E, Propositional Dynamic Logic of Regular Programs, JCSS vol. 18, pp. 194-211, 1979.

[GPSS80] Gabbay, D., Pnueli, A., et al., The Temporal Analysis of Fairness. 7th Annual ACM Symp. on Principles of Programming Languages, 1980.

[HA82] Halpern, J. Y., Deterministic Process Logic is Elementary, to appear in FOCS, 1982.

[HO81] Hailpern, B., and Owicki, S., Modular Verification of Concurrent Programs Using Temporal Logic, Stanford TR, 1981.

[HKP80] Harel, D., Kozen, D., and Parikh, R., Process Logic: Expressiveness, Decidability, and Completeness, 12th Annual ACM Symp. on Theory of Computing, 1980.

[LA80] Lamport, L., "Sometimes" is Sometimes "Not Never." 7th Annual ACM Symp. on Principles of Programming Languages, 1980.

[MW81] Manna, Z., and Wolper, P., Synthesis of Communicating Processes from Temporal Logic Specifications, IBM Workshop on Logics of Programs, Springer-Verlag Lecture Notes in Computer Science #131, 1981.

[OL80] Owicki, S., and Lamport, L., Proving Liveness Properties of Concurrent Programs, Computer Systems Laboratory, Stanford Univ., 1980.

[PN77] Pnueli, A., The Temporal Logic of Programs, 19th Annual Symp. on Foundations of Computer Science, 1977.

[PN81] Pnueli, A., The Temporal Logic of Concurrent Programs, Theoretical Computer Science, V13, pp. 45-60, 1981.

[PR57] Prior, A., Time and Modality, Oxford Univ. Press, London, 1957.

[PR67] Prior, A., Past, Present, and Future, Oxford Univ. Press, London, 1967.

[RU71] Rescher, N., and Urquhart, A., Temporal Logic, Springer-Verlag, Berlin, 1971.

[WO81] Wolper, P., Temporal Logic can be more Expressive, 22nd Annual Symp. on Foundations of Computer Science, 1981.

[WO82] Wolper, P., Specification and Synthesis of Communicating Processes Using an Extended Temporal Logic (Preliminary Version), 9th Annual ACM Symp. on Principles of Programming Languages, 1982.