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

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

ABSTRACT: We study the differences between and appropriateness of branching versus linear time temporal logic, issues first considered by Lamport (1980). To facilitate a careful examination of these issues, we define a language, CTL*, in which a universal or existential path quantifier can prefix an arbitrary linear time assertion. We then compare the expressive power of a number of sublanguages. We also relate CTL* to the logics MPL of Abrahamson (1980) and PL of Harel, Kozen, and Parikh (1982). Finally, we make some general observations regarding the choice between a system of branching or linear time.

Categories and subject desciptors: 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 Keywords and Phrases: temporal logic, concurrent programs, parallelism

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.

*A preliminary version of this paper was presented at the 1983 ACM Symposium on Principles of Programming Languages.

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 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 infinite then $|x| = \infty$ and has the form $(s_0,s_1,s_2,...)$. If x can be either finite or infinite it is sometimes convenient to write $x = (s_0,...,s_k,.)$ 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 $x^{succ} = (s_1,...,s_k,.)$; otherwise $x^{succ} = x$. We define the *suffixes* of x, $x^0 = x$, $x^{m+1} = (x^m)^{succ}$. 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. 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 $x \cdot y$, 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 $x^{succ} \in X$. 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 $y_0x_0, y_0y_1x_1, y_0y_1y_2x_2, ... \in X$ and each y_i is nonempty then the "limit" path $y_0y_1y_2... \in X$. 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 $(s_0,s_1,s_2,...)$ 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]). Finally, we say that X is *state complete* provided that for each $s \in S$ there is some $x \in X$ for which $first(x) = s$.

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 $\neg p$ ("negation").
3. If p is a temporal formula then so are $\Box p$ (meaning "always p")
   and $\leadsto p$ (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 $M,s \models_B q$
   $M,s \models_B \neg p$ iff not($M,s \models_B p$)
3. $M,s \models_B \Box p$ iff $\forall$ path $x \in X$ with $first(x) = s$, $\forall n \geq 0$, $M,first(x^n) \models_B p$

and linear time formulae, it is not clear from the syntax which interpretation is intended. This has the effect of obscuring an 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 ($\leadsto\Box$ $\neg$ENABLED) $\vee$ $\leadsto$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 $\leadsto P \vee \Box\neg P$ 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 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").

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.

We use $|p|$ to denote the *length* of formula p, ie., 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, G, *(*, *)*, etc.).

Remark: In rules 3-6a, the arguments p,q are state formulae whereas in rules 3-6b the arguments are path formulae.

3.2 Semantics. We write $M,s \models p$ ($M,x \models p$) 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 $s \models p$ ($x \models p$). We define $\models$ inductively:

S1. $s \models P$ iff $P \in L(s)$

S2. $s \models p \wedge q$ iff $s \models p$ and $s \models q$

$s \models \neg p$ iff not ($s \models p$)

S3. $s \models Ap$ iff for every path $x \in X$ with $first(x) = s$, $x \models p$

$s \models Ep$ iff for some path $x \in X$ with $first(x) = s$, $x \models p$

P1. $x \models P$ iff $P \in L(first(x))$

P2. $x \models p \wedge q$ iff $x \models p$ and $x \models q$

$x \models \neg p$ iff not ($x \models p$)

P3a. $x \models Gp$ iff for all $i \geq 0$, $first(x^i) \models p$

$x \models Fp$ iff for some $i \geq 0$, $first(x^i) \models p$

P3b. $x \models Gp$ iff for all $i \geq 0$, $x^i \models p$

$x \models Fp$ iff for some $i \geq 0$, $x^i \models p$

P4a. $x \models Xp$ iff $first(x^1) \models p$

P4b. $x \models Xp$ iff $x^1 \models p$

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, $first(x^i) \models p$

$x \models \overset{\infty}{G} p$ iff for all but a finite number of i, $first(x^i) \models p$

P6b. $x \models \overset{\infty}{F} p$ iff for infinitely many distinct i, $x^i \models p$

$x \models \overset{\infty}{G} p$ iff for all but a finite number of i, $x^i \models p$

Remark: The notions of M-validity and strong equivalence (defined in sections 2.4 and 2.5, respectively) generalize to apply to arbitrary state and path formulae.

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:

$M,x \models_L \Box p$ iff $M,x \models Gp$

L|S'. Since X is fusion closed, {s" $\in$ S: s" appears on some x' $\in$ X'} = S' and M' is thus a structure. Observe that for any state formula r, M,s $\models$ AGr iff M',s $\models$ AGr iff $\forall$s' $\in$ S', (M',s' $\models$ r). Taking r=p, we get $\forall$s' $\in$ S', M',s' $\models$ p. Since p $\equiv_s^{\mathcal{F}}$ q, $\forall$s' $\in$ S', we have M',s' $\models$ q. Now take r=q, to see that M,s $\models$ AGq as desired.

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

Remark: Both fusion closure and state completeness are needed for the previous result. Considering the formulae p = P $\wedge$ EFEX$\neg$P and q = *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*true* $\wedge$ EF*true* and q = p $\vee$ AG*false* we have AGp $\equiv$ AGq but also p $\not\equiv_s$ q if we allow structures that are not state complete.

3.7 Corollary. For any path formula p and state formula q, p $\equiv_s^{\mathcal{F}}$ q iff AGAp $\equiv^{\mathcal{F}}$ AGq.

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

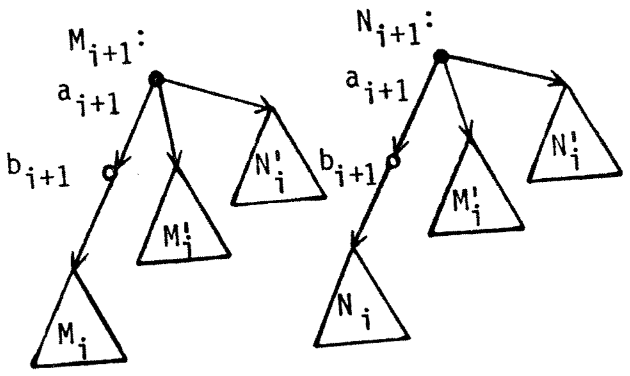
3.8 Definition. As measured with respect to a class of structures $\mathcal{C}$, we say that L_2 is *at least as expressive as* L_1, written $L_1 \leq^{\mathcal{C}} L_2$, provided that for every p $\in$ L_1 there exists q $\in$ L_2 such that p $\equiv^{\mathcal{C}}$ q. We say that L_1 is *exactly as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

4. EXPRESSIVENESS RESULTS

Using the formalism of the last section we compare the relative expressive power of the branching time languages defined there. 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.

We show that the following picture describes their relative expressive power:

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 $A[F(P \wedge XP)]$ 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 $N_i, a_i \models \neg A[F(P \wedge XP)]$. $\square$

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

4.3 Theorem. The formula $E[\ ((P_1 \ U\ P_2) \vee (Q_1 \ U\ Q_2))\ U\ R]$ is not equivalent to any formula $q \in$ ECTL$^+$. $\square$

Proof: Left to the appendix.

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

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

Proof: Left to the appendix.

4.5 Theorem. The ECTL formula $E\overset{\infty}{F} P$ is not equivalent to any formula $q \in$ CTL$^+$.

the **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 $\Diamond$, 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 formulae 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 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 $P \in L(\text{last}(y))$
2. $M,x,y \models p \wedge q$ iff $M,x,y \models p$ and $M,x,y \models q$
   $M,x,y \models \neg p$ 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 $\exists z(M,x,z \models p$ and $y \leq z \leq x$ and $\neg\exists w(y < w < z))$
4. $M,x,y \models \Diamond p$ 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 which are satisfiable only in structures that are not suffix closed (e.g., EGX*true* $\wedge \neg$ EXEGX*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 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. $\Box$

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 $x \in X$, obvious modification of the [HA82] algorithm to force R_A to be limit closed. In fact, no decision procedure for testing satisfiability of CTL* formulae in R-generable structures of elementary complexity was known for some time. Recently, however, decision procedures of triple ([ES83]) or quadruple ([VW83], [PS83]) exponential complexity have been announced.

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 [OL82] 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, a linear time logic, L, as interpreted over branching time structures, as language B(L) (i.e., all formulae of the form Aq where q is a formula of L) is 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 $\mathrm{EG}\,inNCS_i \wedge \mathrm{EF}\,inTRY_i \wedge \mathrm{A}(\mathrm{G}\,inNCS_i \vee \mathrm{F}\,inTRY_i)$. However, using Theorem 4.6, this is provably not expressible in linear time logic, i.e., in a language of the form B(L(-)). The natural candidate formula, $\mathrm{A}(\mathrm{G}\,inNCS_i \vee \mathrm{F}\,inTRY_i)$, allows a "degenerate" model where all paths satisfy $\mathrm{F}\,inTRY_i$ and no path satisfies $\mathrm{G}\,inNCS_i$.

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):

$$G(p' \vee (\vee_i[p^i_0,\ldots,p^i_{n_i}]) \vee (\vee_j Fq_j) \vee (\vee_k GFq'_k))$$

$$\equiv$$

$$Gp' \vee (\vee_i[p',p^i_0,\ldots,p^i_{n_i}]) \vee (\vee_k GFq'_k) \vee (\vee_j GFq_j) \vee$$

$$(\vee_{i,j}[F(q_j \wedge XGp') \vee F(q_j \wedge X[p',p^i_0,\ldots,p^i_{n_i}])])$$

Intuitively, the first three conjuncts of the right hand side take care of the case that no q_j is ever true, and the fourth conjunct covers the case that some q_j is true infinitely often. The last conjunct corresponds to all q_j being true only finitely often: the last time any q_j is true, either Gp' or one of the $[p^i_0,\ldots,p^i_{n_i}]$ 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:

$$GF(p \wedge [p^0_0,\ldots,p^0_{n_0}] \wedge\ldots\wedge [p^m_0,\ldots,p^m_{n_m}] \wedge X[q^0_0,\ldots,q^0_{k_0}] \wedge Fr_1 \wedge\ldots\wedge Fr_n)$$

$$\equiv$$

$$GFp \wedge F([p^0_0,\ldots,p^0_{n_0}]) \wedge\ldots\wedge F([p^m_0,\ldots,p^m_{n_m}]) \wedge F([q^0_0,\ldots,q^0_{k_0}]) \wedge GFr_1 \wedge\ldots\wedge GFr_n$$

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 subformulae of p of the form Fr (corresponding to rule 4 above).

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

$q \wedge [p^0_0,\ldots,p^0_{n_0}] \wedge\ldots\wedge [p^m_0,\ldots,p^m_{n_m}] \wedge \wedge_i GFr_i$

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

$[p_0,\ldots,p_n] \wedge [q_0,\ldots,q_m] \equiv \vee \{[p_0 \wedge q_0,\ldots,p_{i_k} \wedge q_{j_k},\ldots,p_n \wedge q_m]$ 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

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

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

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

iff

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

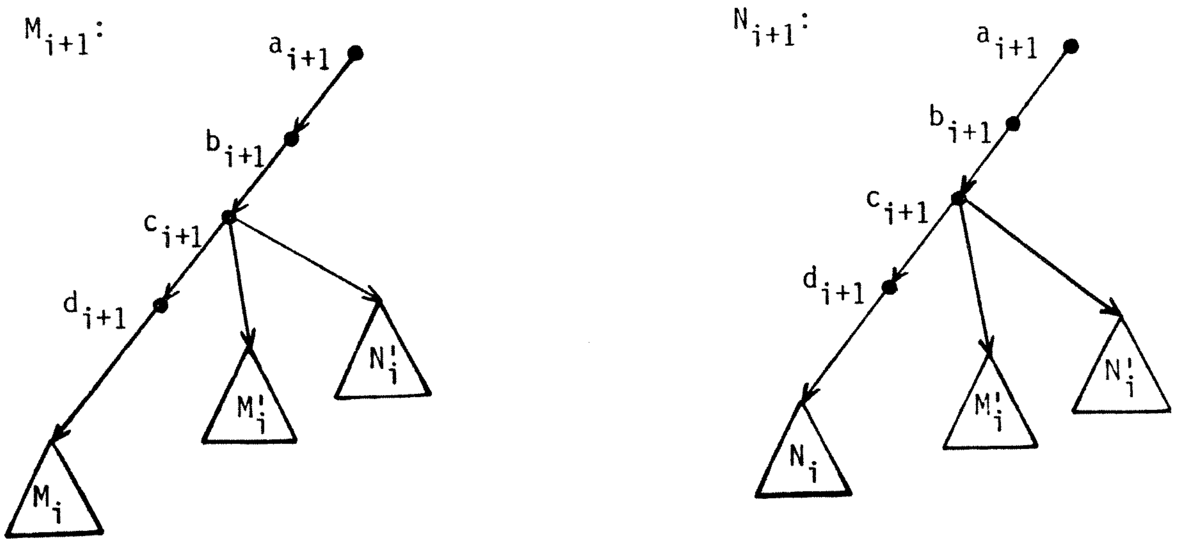
In the last case, if $p = A[q\ U\ r]$ then,

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



where in both M_{i+1} and 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 M'_i, N'_i are copies of M_i, N_i, respectively.

It should be clear that

(1) for all i, $M_i, a_i \models E[\,((P_1 \,U\, P_2) \vee (Q_1 \,U\, Q_2))\; U\, R]$ and
$N_i, a_i \models \neg E[\,((P_1 \,U\, P_2) \vee (Q_1 \,U\, Q_2))\; U\, R]$.

We can also show that

(2) For any 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 $N_i, a_i \models p$.

The details of the remainder of the proof follow along exactly the same lines as that of theorem 4.2. Details are left to the reader. □

Proof of Theorem 4.4: 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 $N_i, c_i \models \neg E[\overset{\infty}{F} P \wedge \overset{\infty}{F} Q]$. 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 $N_i, c_i \models \neg E[\overset{\infty}{F} P \wedge \overset{\infty}{F} Q]$. We define M_1, N_1 to have the graphs shown below:

We first argue by induction on $|p|$, that for any ECTL formula p,

(*) $[(\exists j \geq |p|\ c_j \models p \text{ iff } \forall j \geq |p|\ c_j \models p]$ and

$[(\exists j \geq |p|\ d_j \models p) \text{ iff } (\forall j \geq |p|\ d_j \models p)]$.

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 formulae of length I and try to show it for I+1. Note that the $\Leftarrow$ direction is obvious. To establish the $\Rightarrow$ direction it will suffice 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 I: $p = EXq$. 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 $j\text{-}1, j\text{-}2 \geq |q|$ 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 II: $p = E[q\ U\ r]$. Assume $c_j \models p$. Now $c_j \models p$ iff

(1) $c_j \models r$ or

(2) $c_j \models q$, $d_j \models r$

(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) $d_j \models q$ or

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

(3) $d_{j-1} \models p$

By the induction hypothesis, (1) implies $d_{j-1} \models q$ and hence $d_{j-1} \models p$. Since $j > I + 1 = |p|$, $j-2 \geq I \geq |q|$. Using the induction hypothesis we see that (2) implies $c_{j-2} \models q$ and hence $d_{j-1} \models p$. Whichever of (1), (2), or (3) obtains, we get $d_{j-1} \models p$ ((iv)).

Case V: $p = E\overset{\infty}{G}q$. The proof in this case is exactly like that for Case 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 where 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], A[q U r], or $E\overset{\infty}{F}q$.

Case 1: $p = EXq$: We first note that

$a_i \models p$ iff $a_i \models q$ or $b_i \models q$

iff $c_i \models q$ or $d_i \models q$ (by the induction hypothesis twice)

iff $c_i \models p$

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).

iff $d_i \models p$

Case 4: $p = E\overset{\infty}{F}q$. Observe that $a_i \models p$ iff

(1) $a_i \models q$ or

(2) $b_i \models q$ or

(3) $c_{i-1} \models p$

and that $c_i \models p$ iff

(4) $c_i \models q$ or

(5) $d_i \models q$ or

(6) $c_{i-1} \models p$.

By the induction hypothesis twice we see that (1) is equivalent to (4) and (2) is equivalent to (5). We conclude that $a_i \models p$ iff $c_i \models p$.

Next observe that $b_i \models p$ iff

(1) $b_i \models q$ or

(2) $a_i \models q$ or

(3) $c_{i-1} \models p$

and that $d_i \models p$ iff

(4) $d_i \models q$ or

(5) $c_{i-1} \models p$

(1) is equivalent to (4) by induction hypothesis. If $a_i \models q$ then $c_i \models q$ by induction hypothesis and then $c_{i-1} \models q$ by (*). Thus, $c_{i-1} \models p$ and (2) implies (5). It follows that $b_i \models p$ iff $d_i \models p$.

Case 5: $p = E\overset{\infty}{G}q$. The exact same argument as for Case 4 applies.

This completes the proof of (**) and of the Theorem 4.4. □

Proof of Theorem 4.5: 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, $M_i, a_i \models E\overset{\infty}{F}P$ and $N_i, c_i \models \neg E\overset{\infty}{F}P$. We show that CTL is unable to distinguish between the two sequences of models, i.e., for all CTL formulae p with $|p| \leq i$, $M_i, a_i \models p$ iff $N_i, a_{ii} \models p$. The result follows since if $E\overset{\infty}{F}P$ were equivalent to some CTL$^+$ formula p, it would also be equivalent to some CTL formula p' of length i. But $M_i, a_i \models p'$ iff $N_i, c_i \models p'$ contradicting the fact that $M_i, a_i \models E\overset{\infty}{F}P$ and $N_i, a_{ii} \models \neg E\overset{\infty}{F}P$. (Since a_i, b_i appear only in M_i and c_i, d_i in N_i, we omit the models from our assertions.)

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, pp. 164-176, 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, pp. 52-71, 1981.

[CES83] Clarke, E. M., Emerson, E. A., and Sistla, A. P., Automatic Verification of Finite State Concurrent Programs: A Practical Approach, 10th Annual ACM Symp. on Principles of Programming Languages, pp. 117-126, January 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, pp. 169-181, 1981.

[EC82] Emerson, E. A., and Clarke, E. M., Using Branching Time Logic to Synthesize Synchronization Skeletons, Science of Computer Programming, v. 2, pp. 241-266, 1982.

[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, pp. 169-180, 1982.

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

[ES83] Emerson, E. A., and Sistla A. P., Deciding Branching Time Logics: A Triple Exponential Decision Procedure for CTL*, 1983 Workshop on Logics of Programs, Springer-Verlag, to appear.

[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, pp. 163-173, 1980.