INFORMATION AND CONTROL 56, 72–99 (1983)

Temporal Logic Can Be More Expressive

PIERRE WOLPER*

Computer Science Department,
Stanford University, Stanford, California 94305

It is first proved that there are properties of sequences that are not expressible in temporal logic, even though they are easily expressible using, for instance, regular expressions. Then, it is shown how temporal logic can be extended to express any property definable by a right-linear grammar and hence a regular expression. Finally, a complete axiomatization and a decision procedure for the extended temporal logic are given and the complexity of the extended logic is examined.

1. INTRODUCTION

To state or prove properties of concurrent programs, it is often necessary to deal not only with the input/output behavior of the program but also with its entire execution sequence. This has led to the development of specification languages for concurrent programs that are oriented toward the description of sequences. Among these languages, one can distinguish those based on regular expressions, like "path expressions" (Habermann, 1975) or "flow expressions" (Shaw, 1979) and those based on a logic of sequences like Temporal Logic (Manna and Pnueli, 1981; Gabbay, Pnueli, Shelah, and Stavi, 1980).

Temporal logic has been shown to be adequate for expressing a wide variety of properties of the execution sequences of concurrent programs such as partial correctness, termination, mutual exclusion, accessibility, or liveness (Manna and Pnueli, 1981). Based on the fact that the propositional version of temporal logic (PTL) is as expressive as the first-order theory of linear order, it has even been called *expressively complete.*

In this paper, we prove that there are, nevertheless, properties of sequences that cannot be expressed in PTL but that are easily expressible in languages based on regular expressions. An example of such a property is that a given

* Current address: Bell Laboratories, 600 Mountain Avenue, Murray Hill, N. J. 07974.
This research was supported in part by the National Science Foundation under Grant MCS80-06930, by the Office of Naval Research under Contract N00014-76-C-0687, by the United States Air Force Office of Scientific Research under Grant AFSOR-81-0014, by DARPA under Contract N00039-82-C-0250, and by an IBM Predoctoral Fellowship.

0019-9958/83 $3.00

event has to happen exactly every n steps of the computation and can either happen or not at other steps. This could, for instance, express that a program has to check for the occurrence of a given condition every n execution steps.

We then introduce an extended temporal logic (ETL) that can express any property of a sequence definable by a right-linear grammar. To do this, we give a method for defining a temporal operator corresponding to any right-linear grammar. We show how to extend the axiomatization of PTL to include these operators and prove the extended axiomatization complete. We also give a decision procedure for ETL and characterize its complexity which turns out to be the same as that of PTL (PSPACE-complete).

2. Propositional Temporal Logic: Definition

Propositional temporal logic (PTL) is classical propositional logic extended with four "temporal" operators: $\bigcirc, \Diamond, \Box$, and U. The first three are unary, the last binary. Intuitively, for a sequence, $\Box f$ is true if f is true in all future states of that sequence; $\bigcirc f$ is true if f is true in the next state in the sequence; $\Diamond f$ is true if f is true in some future state (is eventually true) and $f_1 U f_2$ is true if f_1 is true for all states until the first state where f_2 is true. More precisely, we have the following:

Syntax

PTL *formulas* are built from

(i) A set $\mathscr{P}$ of atomic propositions: $p_1, p_2, p_3, \ldots$.

(ii) Boolean connectives: $\wedge, \neg$.

(iii) Temporal operators: $\bigcirc$ ("next"), $\Box$ ("always"), $\Diamond$ ("eventually"), U ("until").

The formation rules are:

(i) An atomic proposition $p \in \mathscr{P}$ is a formula.

(ii) If f_1 and f_2 are formulas, so are

$$f_1 \wedge f_2, \quad \neg f_1, \quad \bigcirc f_1, \quad \Box f_1, \quad \Diamond f_1, \quad f_1 U f_2.$$

We also use $\vee$ and $\supset$ as the usual abbreviations and parentheses to resolve ambiguities.

Semantics

A *structure* for a PTL formula (with a set $\mathscr{P}$ of atomic propositions) is a triple $\mathscr{A} = (S, N, \pi)$ where

(i) S is a finite or enumerable set of states.

(ii) N: $(S \to S)$ is a total successor function that for each state gives a unique next state.

(iii) π: $(S \to 2^{\mathcal{P}})$ assigns truth values to the atomic propositions of the language in each state.

For a structure $\mathscr{A}$ and a state $s \in S$ we have

$$\begin{aligned}
\langle \mathscr{A}, s\rangle \models p \quad &\text{iff} \quad p \in \pi(s),\\
\langle \mathscr{A}, s\rangle \models f_1 \wedge f_2 \quad &\text{iff} \quad \langle \mathscr{A}, s\rangle \models f_1 \text{ and } \langle \mathscr{A}, s\rangle \models f_2,\\
\langle \mathscr{A}, s\rangle \models \neg f \quad &\text{iff} \quad \text{not } \langle \mathscr{A}, s\rangle \models f,\\
\langle \mathscr{A}, s\rangle \models \bigcirc f \quad &\text{iff} \quad \langle \mathscr{A}, N(s)\rangle \models f.
\end{aligned}$$

In the following definitions, we denote by $N^i(s)$ the ith state in the sequence

$$s, \quad N(s), \quad N(N(s)), \quad N(N(N(s))),\ldots$$

of successors of a state s:

$$\begin{aligned}
\langle \mathscr{A}, s\rangle \models \Box f \quad &\text{iff} \quad (\forall i \geqslant 0)(\langle \mathscr{A}, N^i(s)\rangle \models f)\\
\langle \mathscr{A}, s\rangle \models \Diamond f \quad &\text{iff} \quad (\exists i \geqslant 0)(\langle \mathscr{A}, N^i(s)\rangle \models f)\\
\langle \mathscr{A}, s\rangle \models f_1 \, U \, f_2 \quad &\text{iff} \quad (\forall i \geqslant 0)(\langle \mathscr{A}, N^i(s)\rangle \models f_1) \text{ or}\\
&\qquad (\exists i \geqslant 0)(\langle \mathscr{A}, N^i(s)\rangle \models f_2\\
&\qquad\qquad \text{and } \forall j \ (0 \leqslant j < i \supset \langle \mathscr{A}, N^j(s)\rangle \models f_1))
\end{aligned}$$

An *interpretation* $\mathscr{I} = \langle \mathscr{A}, s_0\rangle$ for PTL consists of a structure $\mathscr{A}$ and an initial state $s_0 \in S$. We will say that an interpretation $\mathscr{I} = \langle \mathscr{A}, s_0\rangle$ *satisfies* a formula f iff $\langle \mathscr{A}, s_0\rangle \models f$. Since an interpretation $\mathscr{I}$ uniquely determines a sequence

$$\sigma = s_0, N(s_0), N^2(s_0), N^3(s_0),\ldots$$

we will often say "the sequence σ satisfies a formula" instead of "the interpretation $\mathscr{I}$ satisfies a formula."

There are several variants of the definitions we have given here. For instance, in Gabbay, Pnueli, Shelah and Stavi (1980), an operator U_E that is similar to U but requires that f_2 is eventually true is defined. It is

$$\begin{aligned}
\langle \mathscr{A}, s\rangle \models f_1 \, U_E \, f_2 \quad &\text{iff} \quad (\exists i \geqslant 0)(\langle \mathscr{A}, N^i(s)\rangle \models f_2\\
&\qquad \text{and} \quad \forall j (0 \leqslant j < i \supset \langle \mathscr{A}, N^j(s)\rangle \models f_1)).
\end{aligned}$$

The relation between U and U_E is

$$f_1 U_E f_2 \equiv f_1 U f_2 \wedge \Diamond f_2.$$

3. PTL: Axiomatization, Decision Procedure, and Complexity

Propositional temporal logic has a simple complete axiomatization. The following is a variant of the system proved to be complete in Gabbay, Pnueli, Shelah, and Stavi, (1980) and also described in Manna, (1981).

Axiom schemas:

$$\vdash \Diamond p \equiv \neg \Box \neg p, \tag{A1}$$

$$\vdash \Box(p \supset q) \supset (\Box p \supset \Box q), \tag{A2}$$

$$\vdash \bigcirc \neg p \equiv \neg \bigcirc p, \tag{A3}$$

$$\vdash \bigcirc(p \supset q) \supset (\bigcirc p \supset \bigcirc q), \tag{A4}$$

$$\vdash \Box p \supset p \wedge \bigcirc p \wedge \bigcirc \Box p, \tag{A5}$$

$$\vdash \Box(p \supset \bigcirc p) \supset (p \supset \Box p), \tag{A6}$$

$$\vdash \Box p \supset p\, U\, q, \tag{A7}$$

$$\vdash p\, U\, q \equiv q \vee (p \wedge \bigcirc(p\, U\, q)). \tag{A8}$$

Inference rules:

$$\text{If } w \text{ is a propositional tautology, then } \vdash w, \tag{R1}$$

$$\text{If } \vdash w_1 \supset w_2 \text{ and } \vdash w_1, \text{ then } \vdash w_2, \tag{R2}$$

$$\text{If } \vdash w, \text{ then } \vdash \Box w, \tag{R3}$$

As will be seen later, our results on extended temporal logic imply that the two axioms concerning "until" (U) can be replaced by

$$\vdash p\, U\, q \supset q \vee (p \wedge \bigcirc(p\, U\, q)), \tag{A9}$$

$$\vdash [u \wedge \Box(u \supset q \vee (p \wedge \bigcirc u)] \supset p\, U\, q. \tag{A10}$$

The last axiom is an explicit induction axiom for U which makes proofs of statements involving U substantially easier.

PTL has what is often called the *small model property*. This means that if a PTL formula of length l is satisfiable, then it is satisfiable in a structure $\langle \mathscr{A}, s_0 \rangle$ of size at most k^l for a fixed k (i.e., it has a model of size at most

k^l). This gives an obvious decision procedure for satisfiability of PTL formulas (check all possible models with less than k^l states).

This decision procedure can be quite wasteful as it always looks at all possible models. It can be improved. The idea is to use a tableau-like procedure that is more goal directed in the sense that it tries to construct a model satisfying the formula state by state instead of searching through all possible models. In the next section we will give such a decision procedure for ETL. This decision procedure will also be applicable to PTL, since ETL contains PTL.

As far as characterizing the complexity of satisfiability for PTL, it turns out that one can apply to PTL the techniques developed in Halpern and Reif (1981) to prove that strict deterministic propositional dynamic logic is PSPACE-complete. We thus have the following result:

THEOREM 3.1. *Satisfiability for* PTL *is* PSPACE*-complete.*

This theorem is proved in Sistla and Clarke (1982).

4. EXPRESSIVENESS OF PTL

Following Pnueli (1977) and Manna and Pnueli (1981), temporal logic can be used to express various properties of programs. Consider a system of m concurrent processes in which each atomic instruction i of process j is labeled l_i^j. The "initial" label for process j is l_0^j and the "final" label l_e^j. If the input predicate for the program is ϕ and the output predicate is ψ and if we denote the fact that process j is at location l_i^j by atl_i^j then, total correctness can be expressed as

$$(at\bar{l}_0 \wedge \phi) \supset \Diamond(at\bar{l}_e \wedge \psi),$$

where $\bar{l}_0 = (l_0^1,\ldots, l_0^n)$, $\bar{l}_e = (l_e^1,\ldots, l_e^n)$ and for a set of labels $\bar{l}$, $at\bar{l} \equiv \bigwedge_i atl^i, l^i \in \bar{l}$.

Mutual exclusions between the instructions at locations l_i^j and $l_{i'}^{j'}$ can be expressed as

$$(at\bar{l}_0 \wedge \phi) \supset \Box\neg(atl_i^j \wedge atl_{i'}^{j'}).$$

That l_i^j will always be reached from l_k^l can be expressed as

$$(at\bar{l}_0 \wedge \phi) \supset \Box(atl_k^l \supset \Diamond atl_i^j).$$

In his doctoral thesis Kamp (1968) proved that a propositional temporal logic including the "Until" operator is as expressive as the first order theory of linear order. This theory is the first order theory of N (the natural

numbers) with equality, the binary relation $<$ and a set of unary predicates. Based on this result he called such temporal logics *expressively complete*. Let us immediately point out that though PTL and the first order theory of linear order are expressively equivalent, they have very different properties. For instance PTL is PSPACE-complete whereas the first order theory is nonelementary (Meyer, 1975).

In Gabbay, Pnueli, Shelah, and Stavi (1980) Kamp's result was restated and the term "expressively complete" was applied to the logic we have defined here. This characterization appears to be ill-chosen at least for the use of PTL in computer science. Indeed, there are properties that are not expressible in PTL and hence also not expressible in the first-order theory of linear order. Many of these properties can be of interest when dealing with execution sequences of programs. A simple example of such a property is that a given proposition p has to be true in every even state of a sequence. We will denote that property by *even*(p).

Note. A formula like

$$p \wedge \Box(p \supset \bigcirc \neg p) \wedge \Box(\neg p \supset \bigcirc p)$$

does not express *even*(p). Indeed it is not satisfied by the sequence where p is always true, which clearly satisfies *even*(p).

To prove that *even*(p) is not expressible in PTL we will prove a slightly more general result. Let us denote by $p^i(\neg p)\,p^\omega$ the sequence where p is true in the first i states, false in the state $i+1$ and true in all states after that. We can then prove

THEOREM 4.1. *Given a proposition p, any* PTL *formula $f(p)$ containing n "next" ($\bigcirc$) operators has the same truth value on all sequences of the form $p^i(\neg p)\,p^\omega$, $i > n$.*

Proof. We have to prove that the truth value of a formula $f(p)$ on a sequence $p^i(\neg p)\,p^\omega$, where $i > n$ does not depend on i. For convenience, let us denote the truth value of f on $p^i(\neg p)\,p^\omega$ by $|f|_i$. The proof proceeds by induction. We prove that the theorem holds for the formula f, assuming that it holds for its subformulas.

Case 1. $f(p)$ is an atomic proposition. Thus $f(p) \equiv p$ and always $|f(p)|_i \equiv T$.

Case 2. $f(p)$ is $f_1 \wedge f_2$, $f_1 \vee f_2$ or $\neg f_1$. This case follows immediately from the induction hypothesis.

Case 3. $f(p)$ is $\Box f$. By the definition of $\Box$,

$$|\Box f|_i \equiv |f|_i \wedge |f|_{i-1} \wedge \cdots \wedge |f|_{n+1} \wedge |\Box f|_n$$

and by the induction hypothesis,

$$|\Box f|_i \equiv |f|_{n+1} \wedge |\Box f|_n.$$

Case 4. $f(p)$ is $\bigcirc f$. We have that $|\bigcirc f|_i \equiv |f|_{i-1}$ and by the induction hypothesis $|f|_{i-1}$ is independent of i as $i-1 > n-1$ and f contains $n-1$ "next" ($\bigcirc$) operators.

Case 5. $f(p)$ is $f_1 U f_2$. Given (A8),

$$|f_1 U f_2|_i \equiv |f_2|_i \vee (|f_1|_i \wedge (|f_2|_{i-1} \vee (|f_1|_{i-1} \wedge \cdots \wedge (|f_2|_{n+1} \vee (|f_1|_{n+1} \wedge |f_1 U f_2|_n)) \cdots)))$$

and by the induction hypothesis,

$$|f_1 U f_2|_i \equiv (|f_2|_{n+1} \vee (|f_1|_{n+1} \wedge |f_1 U f_2|_n))$$

which is independent of i. ∎

COROLLARY 4.2. *For any given $m \geqslant 2$, the property "p is true in every state s_i, where $i = km$ (integer $k \geqslant 0$)" is not expressible in* PTL.

Proof. Consider a formula $f(p)$ that would express that property for a given m. It has a fixed number l of $\bigcirc$ operators. Then by the theorem, its truth value on $p^{km}(\neg p)p^{\omega}$ and $p^{km-1}(\neg p)p^{\omega}$, where k is such that $km-1 > l$, is the same. But, the required property holds for the former sequence but not for the latter. So, the formula $f(p)$ cannot express that property. ∎

5. EXTENDED TEMPORAL LOGIC: DEFINITION

As we saw in Section 4, a property like *even*(p) is not expressible in PTL. On the other hand, that property does not seem very difficult to express. This can be done in several ways.

First, in a language based on regular expressions, a formula like

$$(p; True)^{\omega},$$

where ω denotes infinite repetition expresses *even*(p). A survey of the use of regular expressions to describe properties of sequences appears in Shaw (1979).

If one wishes to stay within the framework of temporal logic, one could use a quantified version of PTL. The property *even*(p) would then be expressed by

$$\exists q(q \wedge \Box(q \supset \bigcirc\neg q) \wedge \Box(\neg q \supset \bigcirc q) \wedge \Box(q \supset p)).$$

Another alternative is to add to PTL an operator *even* denoting exactly the required property. This is a reasonable alternative as long as the new operator can be axiomatized and incorporated into the decision procedure for PTL.

The problem, really, is not to express *even*(p) but rather to be able to express it within a system that has useful properties like a complete axiomatization and a reasonable decision procedure. For instance, using a quantified version of tempotal logic seems to be a simple and elegant way of extending PTL. Unfortunately, the resulting language has a nonelementary decision problem (see Wolper, 1982b).

On the other hand, we found that we could add to PTL and operator *even*(p) without increasing the complexity of the decision problem. The natural question to ask of course, is whether other operators can be added in the same way, and if so, what is a characterization of the class of operators that can be added. It turns out that operators corresponding to any property definable by a right-linear grammar can be added to PTL without modifying the complexity of its decision problem.

Temporal logic with operators corresponding to right-linear grammars will be our extended temporal logic. To interpret grammars as operators, we have to establsih a correspondence between words generated by a grammar and sequences.

Given a word w (finite or infinite) over a finite alphabet Σ and an assignment of formulas to each of the letters of Σ, we will say that a sequence satisfies the word w for the given assignment if, for all i, the formula associated with the letter appearing in the ith position of the word w is true in the ith state of the sequence.

EXAMPLE. If $\Sigma = \{a, b\}$ and we assign the formula p to a and the formula True to b, the infinite word

$$ababababab\ldots,$$

is satisfied by any sequence

$$s = s_0, s_1, s_2, s_3, \ldots$$

in which p is true in every even state and nothing is required of odd states.

With each right-linear grammar $G = (V_{NT}, V_T, P, V_0)$ we associate a temporal operator $\mathscr{G}$ (called a *grammar operator*) in the following way. If $V_T = (v_1, v_2, \ldots, v_n)$, then the operator $\mathscr{G}$ has exactly n arguments and $\mathscr{G}(f_1, f_2, \ldots, f_n)$ is true of a sequence if there is some word (finite or infinite) generated by G that is satisfied by that sequence when f_1 is assigned to v_1, f_2 to $v_2, \ldots$, and f_n to v_n.

EXAMPLE. The grammar

$$V_0 \to v_1 V_1$$
$$V_1 \to v_2 V_0$$

generates only the infinite word $v_1 v_2 v_1 v_2 \ldots$. Thus if $\mathscr{G}(f_1, f_2)$ is the operator associated with that grammar, $\mathscr{G}(p, \mathit{True})$ expresses the property $\mathit{even}(p)$.

Before we formalize our definition of grammar operators, one point has to be clarified. We mentioned that we consider both finite and infinite words generated by right-linear grammars. The concept of a grammar generating a finite word is a familiar one but the concept of a grammar generating an infinite word needs some explanation. Given a right-linear grammar having productions of the form $V \to vV'$ or $V \to v$, a countably infinite word (ω-word) is obtained by applying the productions of the first type ω times. If, as in our preceding example, there are no productions of the second type, only infinite words are generated by the grammar.

One can find a discussion of grammars for ω-words in Cohen and Gold (1977). In this paper it is mentioned that the definition of an ω-regular grammar should also contain a *repetition set*. A repetition set is a set of nonterminals that have to appear an infinite number of times in any generation of an ω-word. Adding a repetition set to the definition of ω-regular grammars does indeed extend the sets of words definable by such grammars. We will, however, consider here grammars without repetition sets, as this does not restrict the expressiveness of our ETL. This is due to the fact that it is possible to express by an ETL formula the condition corresponding to the existence of a repetition set (Wolper, 1982b).

We will now give a precise definition of our grammar operators. Given a grammar $G^a = (V_{NT}, V_T, P, V_0)$, where

$V_{NT} = \{V_0, V_1, \ldots, V_i\}$,

$V_T = \{v_1, v_2, \ldots, v_n\}$,

P is a set of productions of the form $V_i \to v_{ij} V_{ij}$ or $V_i \to v_{ij}$.

We define $l+1$ grammar operators $\mathscr{G}_i^a$, one for each member of V_{NT}. Each $\mathscr{G}_i^a$ is an n-ary operator $\mathscr{G}_i^a(f_1, f_2, \ldots, f_n)$ $(n = |V_T|)$. Semantically, given a structure $\mathscr{A} = (S, N, \pi)$ and a state $s \in S$,

$$\langle A, s \rangle \models \mathscr{G}_i^a(f_1, \ldots, f_n)$$

if and only if there is a word $w = v_{w_0} v_{w_1} \cdots$ $(1 \leqslant w_i \leqslant n)$ generated by G^a from V_i such that for all $j < 1 + |w|$,

$$\langle \mathscr{A}, N^j(s) \rangle \models f_{w_j}.$$

As we always define a grammar operator for each nonterminal, we will from now on omit the start symbol from the definition of grammars. A grammar operator will be denoted by $\mathscr{G}_i^q(f_1,\ldots,f_n)$. The superscript refers to the grammar defining that operator and the subscript to the nonterminal to which it corresponds.

It is interesting to note that all the operators of PTL are expressible using grammar operators. The "always" operator ($\Box$) is the operator $\mathscr{G}_0^{\Box}(f_1)$ corresponding to the grammar having

$$V_0 \to v_1 V_0$$

as its only production. The "eventually" operator ($\Diamond$) can then be defined as the dual of $\Box$.

If $\mathscr{G}_0^{\bigcirc}(f_1, f_2)$ is the operator corresponding to the non-terminal V_0 of the grammar having

$$V_0 \to v_1 V_1,$$
$$V_1 \to v_2,$$

as productions, then $\mathscr{G}_0^{\bigcirc}(True, f)$ expresses $\bigcirc f$.

Finally $f_1 \, U \, f_2$ is definable by the grammar

$$V_0 \to v_1 V_0,$$
$$V_0 \to v_2.$$

We will still use $\bigcirc$, $\Box$, $\Diamond$, U in ETL, but they can simply be viewed as a notation for the corresponding grammar operators. However, as we will see in Section 6, $\bigcirc$ and $\Box$ will play a special role in the axiomatization of ETL.

6. ETL: Axiomatization

We will present here a complete axiomatic system for ETL. It contains two axioms for the $\bigcirc$ operator, two axioms for each of the other grammar operators and four rules of inference. The axioms for the grammar operators will be given in a general form that involves the $\bigcirc$ operator. That is why we need to axiomatize $\bigcirc$ separately. The axioms for $\bigcirc$ are the same as those appearing in the axiomatization of PTL. They are

$$\vdash \bigcirc \neg p \equiv \neg \bigcirc p, \tag{N1}$$
$$\vdash \bigcirc (p \supset q) \supset (\bigcirc p \supset \bigcirc q). \tag{N2}$$

The axioms for the other grammar operators are given in a generic form. That is, an instance of these axioms corresponds to each specific grammar

operator. They could be called "axiom schema schemas." Consider a grammar operator $\mathscr{G}_i^a(f_1,\ldots,f_n)$ corresponding to the nonterminal V_i of a grammar $G^a = (V_{NT}, V_T, P)$. Suppose that the productions of the grammar G^a are

$$V_i \to v_{ij}[\![V_{ij}]\!],$$

where $v_{ij} \in V_T$, $V_{ij} \in V_{NT}$ need not be present (we indicate this by double brackets), $0 \leqslant i \leqslant l$ $(=|V_{NT}|)$ and $1 \leqslant j \leqslant m_i$. That is, for each nonterminal V_i there are m_i productions. The axioms for the operator $\mathscr{G}_i^a$ are then

$$\vdash \mathscr{G}_i^a(p_1,\ldots,p_n) \supset \bigvee_{1 \leqslant j \leqslant m_i} (p_{ij} \wedge [\![\bigcirc \mathscr{G}_{ij}^a(p_1,\ldots,p_n)]\!]), \tag{G1}$$

$$\vdash \left[u_i \wedge \bigwedge_{0 \leqslant k \leqslant l} \Box\left(u_k \supset \bigvee_{1 \leqslant j \leqslant m_k} (p_{kj} \wedge [\![\bigcirc u_{kj}]\!])\right)\right] \supset \mathscr{G}_i^a(p_1,\ldots,p_n), \tag{G2}$$

where p_{ij} is the member of $\{p_1,\ldots,p_n\}$ associated with v_{ij}, and where $u_0,\ldots,u_l$ (one for each nonterminal in the grammar) are propositions not already appearing. Each u_i is associated with the nonterminal V_i, u_{kj} being the proposition associated with the nonterminal V_{kj} of the relevant production. The terms within double brackets are omitted if the corresponding nonterminals are missing in the production. One can interpret the first axiom as stating that the grammar operators are fixed points of the relations corresponding to the productions of the grammar and the second as stating that they are greatest fixed points of these relations. The definition of temporal operators as fixed points is discussed in Emerson and Clarke (1980).

The rules of inference of our axiomatic system are

If w is a propositional tautology, then $\vdash w$. (I1)

If $\vdash w_1 \supset w_2$ and $\vdash w_1$, then $\vdash w_2$. (I2)

If $\vdash w$, then $\vdash \bigcirc w$. (I3)

If $\vdash w$, then $\vdash \Box w$. (I4)

EXAMPLE 1. The operators of PTL are grammar operators. They can thus be axiomatized by (G1) and (G2). For $\Box$ the axioms are

$$\vdash \Box p \supset p \wedge \bigcirc \Box p,$$

$$\vdash [u \wedge \Box(u \supset p \wedge \bigcirc u)] \supset \Box p;$$

and for U they are

$$\vdash p\, U\, q \supset q \vee (p \wedge \bigcirc(p\, U\, q)),$$

$$\vdash [u \wedge \Box(u \supset q \vee (p \wedge \bigcirc u))] \supset p\, U\, q.$$

EXAMPLE 2. To illustrate the use of the axioms (G1) and (G2), let us prove

$$\vdash \mathcal{G}_i^a(p_1,\ldots,p_n) \equiv \bigvee_{1\leqslant j\leqslant m_i} (p_{ij} \wedge \bigcirc \mathcal{G}_{ij}^a(p_1,\ldots,p_n)). \tag{6.1}$$

For readability, we omit the double brackets indicating the possible missing terms. Given (G1), we only need to prove

$$\vdash \bigvee_{1\leqslant j\leqslant m_i} (p_{ij} \wedge \bigcirc \mathcal{G}_{ij}^a(p_1,\ldots,p_n)) \supset \mathcal{G}_i^a(p_1,\ldots,p_n). \tag{6.2}$$

This can be done by using (G2) and choosing u_i to be

$$\bigvee_{1\leqslant j\leqslant m_i} (p_{ij} \wedge \bigcirc \mathcal{G}_{ij}^a(p_1,\ldots,p_n)) \tag{6.3}$$

and all the other u_k to be $\mathcal{G}_k^a(p_1,\ldots,p_n)$. With this choice it is straightforward to establish

$$\vdash \bigvee_{1\leqslant j\leqslant m_i} (p_{ij} \wedge \bigcirc \mathcal{G}_{ij}^a(p_1,\ldots,p_n)) \supset \left[u_i \wedge \bigwedge_{0\leqslant k\leqslant l} \Box\left(u_k \supset \bigvee_{1\leqslant j\leqslant m_k} (p_{kj} \wedge \bigcirc u_{kj})\right)\right]; \tag{6.4}$$

and from (6.4) and (G2), (6.2) follows.

In Section 9 we will prove the completeness of this axiomatic system. We have postponed the proof as it is based on the decision procedure for ETL given in Section 7.

7. ETL: DECISION PROCEDURE

As we mentioned earlier ETL is decidable. Like PTL, it has the small model property and hence an obvious decision procedure. Here we will describe the sometimes more efficient tableau decision procedure. It is closely related to the tableau decision procedure for PTL (see Ben-Ari, Manna, and Pnueli, 1981; Rescher and Urquart, 1971) which itself is an extension of the tableau method for porpositional calculus (see Smullyan, 1968). The basic idea of the method is that an ETL formula can be decomposed into sets containing formulas that are either atomic (an atomic propositions or its negation) or have $\bigcirc$ as their main connective. Following Ben-Ari, Manna, and Pnueli (1981), we will call such formulas *elementary*. This decomposition serves to separate the requirements expressed by the formula into a requirement on the first state (the atomic formulas) and on

the "rest" of the sequence (the O-formulas). One can thus try to construct a model state by state and hence test for satisfiability. The tableau (or decomposition) rules map each nonelementary formula f into a set Σ of sets S_i of formulas f_j, the interpretation being that f is satisfiable iff all the formulas in at least one of the sets S_i are satisfiable. The rules include

$$\neg\neg f \to \{\{f\}\},$$
$$\neg \bigcirc f \to \{\{\bigcirc \neg f\}\},$$
$$f_1 \wedge f_2 \to \{\{f_1, f_2\}\},$$
$$\neg(f_1 \vee f_2) \to \{\{\neg f_1, \neg f_2\}\},$$
$$(f_1 \vee f_2) \to \{\{f_1\}\{f_2\}\},$$
$$\neg(f_1 \wedge f_2) \to \{\{\neg f_1\}\{\neg f_2\}\}.$$

For a grammar operator $\mathscr{G}_i^a(f_1,\ldots,f_n)$ corresponding to the nonterminal V_i of a grammar $G^a = (V_{NT}, V_T, P)$ whose productions are of the form

$$V_i \to v_{ij} V_{ij},$$

where V_{ij} need not be present, $0 \leqslant i \leqslant l$ $(= |V_{NT}|)$ and $1 \leqslant j \leqslant m_i$. The rules are the following (we have enclosed in double brackets the terms that are omitted when V_{ij} is missing from a production):

$$\mathscr{G}_i(f_1,\ldots,f_n) \to \bigcup_{1 \leqslant j \leqslant m_i} \{\{f_{ij}, [\![\bigcirc \mathscr{G}_{ij}(f_1,\ldots,f_n)]\!]\}\},$$

$$\neg\mathscr{G}_i(f_1,\ldots,f_n) \to \left\{ \bigcup_{1 \leqslant j \leqslant m_i} \{(\neg f_{ij} \vee [\![\bigcirc \neg \mathscr{G}_{ij}(f_1,\ldots,f_n)]\!])\} \right\}.$$

To test a formula f for satisfiability, we use these decomposition rules to construct a graph that is a systematic search for a model of f. Each node n of the graph is labeled by a set of formulas T_n. Initially, the graph contains exactly one node labeled by $\{f\}$. The graph is then constructed node by node using the decomposition rules. During the construction, to avoid decomposing the same formula twice, we will mark the formulas to which a decomposition rule has been applied (we do not simply discard them as we will need them when checking if eventualities are fulfilled). Once the graph is constructed, we eliminate unsatisfiable nodes.

The graph construction proceeds as follows:

(1) Start with a node labeled by $\{f\}$ where f is the formula to be tested. We will call f the *initial formula* and the corresponding node the *initial node*. Then repeatedly apply steps (2) and (3). In these steps, when we say "create a son of node n labeled by a set of formulas T," we mean create

a node if the graph does not already contain a node labeled by T. If it does, we just create an edge from n to the already existing node.

(2) If a node n labeled by T_n contains an unmarked nonelementary formula f and the tableau rule for f is $f \to \{S_i\}$, then, for each S_i, create a son of n labeled by $(T_n - \{f\}) \cup \{S_i\} \cup \{f^*\}$, where f^* is f marked.

(3) If a node n contains only elementary or marked formulas, then create a son of n labeled by the $\bigcirc$-formulas $\in T_n$ with their outermost $\bigcirc$ removed.

A node containing only elementary or marked formulas will be called a *state*. And, a node that is either the initial nodes or the immediate son of a state will be called a *pre-state*.

Given the form of the tableau rules, the formulas labeling the nodes of the graph are either

(i) Subformulas or negations of subformulas of the initial formula or such formulas preceded by $\bigcirc$.

(ii) Formulas of the form $[\![\bigcirc]\!][\![\neg]\!]\mathscr{G}_k(f_1,\ldots,f_n)$ (i.e., $\mathscr{G}_k$ possibly preceded by $\bigcirc$ and $\neg$), where $\mathscr{G}_k$ is an operator corresponding to some nonterminal V_k of a grammar G defining one of the operators appearing in the initial formula. The number of these formulas is equal to $4l$, where l is the length of the initial formula computed with the convention that the length of a grammar operator is equal to the number of nonterminals in the grammar defining it. The number of nodes in the graph is then at most equal to the number of sets of such formulas, that is 2^{4l}.

At this point, to decide satisfiability, we have to eliminate the unsatisfiable nodes of the graph. We repeatedly apply the following three rules.

(E1) If a node contains both a proposition p and its negation $\neg p$, eliminate that node.

(E2) If all the successors of a node have been eliminated, eliminate that node.

(E3) If a node which is a pre-state contains a set of formulas of the form $\neg\mathscr{G}$ that is not fulfillable (see below), eliminate that node.

The last rule is needed for the following reason. A property of the form $\neg\mathscr{G}$ is what we call an *eventuality property*. For such a property to be satisfied by a sequence, there has to be a point in that sequence at which it differs from any sequence satisfying $\mathscr{G}$. We call the point where this happens the point that *fulfills* the eventuality. But, the tableau rule for $\neg\mathscr{G}$ allows us to indefinitely postpone that point. We thus have to check that such a point can actually exist.

EXAMPLE. Consider the grammar

$$V_0 \to v_1 V_1,$$
$$V_1 \to v_2 V_0,$$

and the corresponding operators $\mathscr{G}_0(f_1, f_2)$ and $\mathscr{G}_1(f_1, f_2)$. The tableau rules for $\neg\mathscr{G}_0$ and $\neg\mathscr{G}_1$ are

$$\neg\mathscr{G}_0(f_1, f_2) \to \{\{\neg f_1 \vee \bigcirc\neg\mathscr{G}_1(f_1, f_2)\}\},$$
$$\neg\mathscr{G}_1(f_1, f_2) \to \{\{\neg f_2 \vee \bigcirc\neg\mathscr{G}_0(f_1, f_2)\}\}.$$

As we have mentioned earlier, $\mathscr{G}_0(p, \mathit{True})$ expresses the property *even* (p). Thus, for $\neg\mathscr{G}_0(p, \mathit{True})$ to be true there has to be an even state where p is false. The tableau rules, however, allow us to always postpone that state by satisfying the $\bigcirc\neg\mathscr{G}$ part of the disjunction. We need to make sure this does not happen.

Let us now examine how to determine if a set of eventuality properties is fulfillable. The tableau rule for a formula $\neg\mathscr{G}$ is of the form

$$\neg\mathscr{G}_i(f_1, \ldots, f_n) \to \left\{ \bigcup_{1 \leqslant j \leqslant m_i} \{(\neg f_{ij} \vee \bigcirc\neg\mathscr{G}_{ij}(f_1, \ldots, f_n))\} \right\}.$$

That is, $\neg\mathscr{G}$ is mapped into a set of disjunctions that we call the *tableau formulas* for $\neg\mathscr{G}$. In each of these disjunctions, we will distinguish between the term of the form $\neg f_{ij}$ and the term $\bigcirc\neg\mathscr{G}_{ij}$. We will call the former the *finite term* and the latter the *inductive term.* We have the following inductive definition:

(F1) A set of eventualities $\{\neg\mathscr{G}_i\}$ is *immediately fulfillable* in a pre-state n if there is a path from n such that the first state on that path contains the finite term of each of the tableau formulas for each of the eventualities $\neg\mathscr{G}_i$.

(F2) A set of eventualities $\{\neg\mathscr{G}_i\}$ is fulfillable in a pre-state if

(i) it is immediately fulfillable, or

(ii) there is a path from that node such that the first state on that path satisfies the following condition: it contains a term from each of the tableau formulas for each of the $\neg\mathscr{G}_i$ and all of these terms that are inductive terms are fulfillable in the next pre-state on that path.

Note that we consider here sets of eventualities rather than single eventualities as can be done for ordinary temporal logic. The reason for this is that, in ETL, the tableau rule for an eventuality can map it into several eventualities. Now, if one considers these separately, one could erroneously

conclude that the eventualities are fulfillable. Indeed, the presence of a path that fulfills each one individually does not imply the presence of a path that fulfills all of them simultaneously. This is due to the fact that when fulfilling one eventuality, another one might get mapped into a set that contains the eventuality that has just been fulfilled, hence eliminating all progress. The example we give below for the decision precedure illustrates this.

To determine if a set of eventualities is fulfillable one proceeds as follows:

(i) Mark the sets that are immediately fulfillable, i.e., the sets that are fulfillable on a path containing just one state. These are obtained by going through the graph and marking all sets for which condition F1 is satisfied.

(ii) Repeat the following step until no new sets are marked as fulfillable:

> Mark as fulfillable all sets of eventualities that can be determined to be fulfillable, given the sets already marked. That is, in each pre-state, mark all sets of eventualities such that, there is a path from the pre-state satisfying the following: the first state on that path contains a term from each of the tableau formulas for each eventuality in the set and, the set of these terms that are inductive is marked as fulfillable in the next pre-state on the path.

The algorithm thus proceeds by first finding all sets of eventualities fulfillable on paths of length 1, then on paths of length 2,.... Each node contains at most l (the length of the initial formula) eventualities. Thus, in each pre-state there are at most 2^l sets of eventualities and the tableau contains no more than $2^l \times 2^{4l}$ sets of eventualities. The elimination procedure thus iterates at most $2^l \times 2^{4l}$ times (at least one set of eventualities is marked at each iteration) and is thus polynomial in the size of the graph. Also note that when a set of eventualities is determined to be fulfillable, one can actally find a path through the graph that fulfills all eventualities in that set. The path can easily be constructed as follows. Each time a set of eventualities is marked to be fulfillable in a pre-state, a pointer is kept to the pre-state containing the set of fulfillable eventualities that enabled us to mark that set as fulfillable (as in the iterative step of the elimination procedure). To obtain the path, one then only needs to follow these pointers until reaching an immediately fulfillable set of eventualities.

The decision precedure ends after all unsatisfiable nodes are eliminated. If the initial node has been eliminated then the formula is unsatisfiable, if not it is satisfiable. It is easy to see that the decision procedure requires time and space exponential in the length l of the initial formula (computed according to our convention on the size of grammar operators).

EXAMPLE. Consider the operators $\mathscr{G}_0^e(f_1, f_2)$ and $\mathscr{G}_1^e(f_1, f_2)$ corresponding to a grammar G^e for which

$$V_{NT} = \{V_0, V_1\}, \qquad V_T = \{v_1, v_2\}, \qquad P = \left\{\begin{array}{l} V_0 \to v_1 V_0 \\ V_0 \to v_1 V_1 \\ V_1 \to v_2 V_0 \\ V_1 \to v_2 V_1 \end{array}\right\}.$$

Applying our satisfiability algorithm to the formula

$$\neg\mathscr{G}_0^e(p, \neg p) \wedge \neg\mathscr{G}_1^e(p, \neg p)$$

we obtain the graph given in Fig. 1.

For conciseness, we have denoted $\neg\mathscr{G}_0^e(p, \neg p)$ by $\neg\mathscr{G}_0^e$ and $\neg\mathscr{G}_1^e(p, \neg p)$ by $\neg\mathscr{G}_1^e$. We have combined the expansion of the last four formulas of node 4 and we have omitted the marked formulas in nodes 5, 6, 7, and 8. Node 8 actually corresponds to 4 different nodes. However all these nodes contain p and $\neg p$ and are thus unsatisfiable. As they will anyway be eliminated we are not interested in them. Also, the arrows leading to node 2 should really lead to a node labeled by $\{\neg\mathscr{G}_0^e, \neg\mathscr{G}_1^e\}$. However, this would simply lead to duplicating part of the tableau.

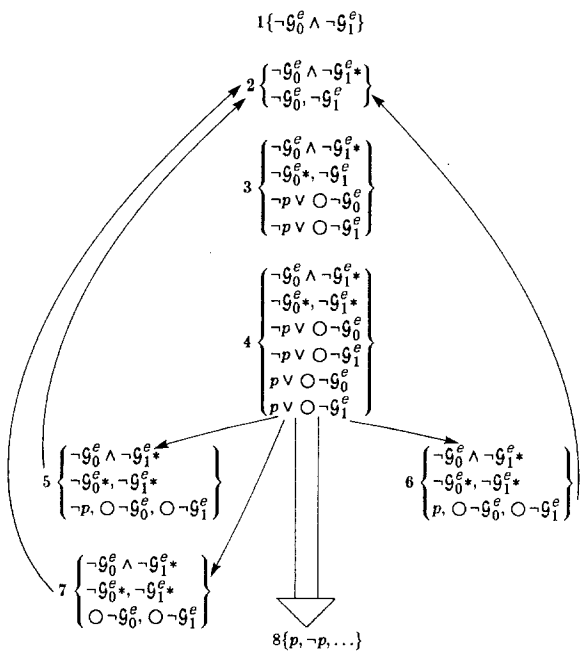

FIGURE 1

The elimination procedure will discard all nodes corresponding to node 8 as they contain p and $\neg p$. The pre-states are 1 and 2. Pre-state 2 is the only one containing formulas of the form $\neg\mathscr{G}$. In that pre-state, the sets $\{\neg\mathscr{G}^e_0\}$ and $\{\neg\mathscr{G}^e_1\}$ are immediately fulfillable. But, no other set of eventualities can be fulfilled. Thus $\{\neg\mathscr{G}^e_0, \neg\mathscr{G}^e_1\}$ is not fulfilled and node 2 is eliminated and hence node 1.

The conclusion is that the formula $\neg\mathscr{G}^e_0 \wedge \neg\mathscr{G}^e_1$ is unsatisfiable.

Before concluding this section we will prove that our decision precedure is sound and complete. We have

THEOREM 7.1. *An* ETL *formula f is satisfiable iff the initial node of the graph generated by the tableau decision procedure for that formula is not eliminated.*

Proof. (a) If the initial node is eliminated, then f is unsatisfiable. We prove by induction that if a node in the tableau labeled by $\{f_1,..., f_s\}$ is eliminated, then $\{f_1,..., f_s\}$ is unsatisfiable.

Case 1. The node was eliminated by rule (E1). It thus contains a proposition and its negation and is unsatisfiable.

Case 2. The node is eliminated by rule (E2) and is not a state. The sons of that node were created using a tableau rule $f \to \{S_i\}$. It is easy to check that for each of these tableau rules, f is satisfiable iff at least one of the S_i is satisfiable. As all the successor nodes have been eliminated, they all contain unsatisfiable sets of formulas and the node contains the unsatisfiable formula f.

Case 3. The node is eliminated by rule (E2) and is a state. Thus, the set of all the $\bigcirc$-formulas in the node is unsatisfiable and so is the set of all formulas in the node.

Case 4. The node was eliminated by using rule (E3). Hence, there is a set of eventualities in the node that it not fulfillable on any path in the tableau. As any model corresponds to some path in the tableau, the set of eventualities is unsatisfiable and so is the set of all formulas in the node.

(b) If the initial node is not eliminated, then f is satisfiable. To prove this, we have to show that if the initial node is not eliminated, there is a model of the initial formula. First notice that except for fulfilling eventualities, a path through the tableau starting with the initial node defines a model of the initial formula. We thus only have to show that we can construct a path through the tableau on which all eventualities are fulfilled. It can be done as follows:

For each pre-state in the graph, unwind a path from that pre-state such that all the properties of the form $\neg\mathscr{G}$ it contains are fulfilled on that path.

This is always possible as we pointed out after giving the elimination rule for unfulfillable sets of eventualities. The length of each of these paths is at most $2^l \times 2^{4l}$.

Once all these paths are constructed, we link them together. The model obtained has the form

$$s_0 \to s_1 \to s_2 \to \cdots \overset{\frown}{s_j \to \cdots s_m}.$$

It may have as many as $2^{4l} \times 2^{5l}$ states. This bound is obtained by multiplying the number of pre-states (2^{4l}) by the length of the path satisfying the eventualities in each pre-state (2^{5l}). ∎

8. ETL: Complexity

The main result is that, like PTL, ETL is PSPACE-complete. The fact that it is PSPACE-hard follows immediately from the fact that PTL is PSPACE-hard. We will give here an alternative proof that uses the greater expressivity of ETL.

Lemma 8.1. *Satisfiability for* ETL *is* PSPACE-*hard.*

Proof. The proof is by reduction from *finite automaton inequivalence* which is PSPACE-complete (see Garey and Johnson, 1979). The finite automaton inequivalence problem is to determine if two finite automata A and B over the same alphabet Σ recognize different languages.

Given A, B, and $\Sigma = \{v_1, ..., v_n\}$, we will built an ETL formula as follows. Consider the alphabet $\Sigma' = \{v_1^1, ..., v_n^1, v_1^2, ..., v_n^2\}$ that has two symbols v_i^1 and v_i^2 for every symbol v_i of Σ. Also consider the grammars G^A and G^B over Σ corresponding to the finite automata A and B, respectively. We will transform these grammars into grammars $G^{A'}$ and $G^{B'}$ over Σ' as follows. Each production of the form

$$V_i \to v_{ij} V_j$$

is replaced by

$$V_i \to v_{ij}^1 V_j$$

and each production of the form

$$V_i \to v_{ij}$$

is replaced by

$$V_i \to v_{ij}^2.$$

In other words the symbols of Σ' with superscript 1 will correspond to letters appearing inside a word and those with superscript 2 to letters appearing at the end of word.

Consider now a set of propositions $p_1,\ldots, p_k$, where $k = \lceil \log_2(|\Sigma|) \rceil$. This set of propositions can be used to encode the letters of Σ in the standard way. The letter v_1 is encoded by $f_0 \equiv \neg p_1 \wedge \cdots \wedge \neg p_k$, v_2 by $f_1 \equiv p_1 \wedge \neg p_2 \wedge \cdots \wedge \neg p_k$ and v_{2^k} by $f_{2^k-1} \equiv p_1 \wedge \cdots \wedge p_k$. If $2^k > n$, we do not use the formulas $f_n,\ldots, f_{2^k-1}$.

To the grammars $G^{A'}$ and $G^{B'}$ correspond two grammar operators: $\mathscr{G}_0^{A'}$ and $\mathscr{G}_0^{B'}$. They both have $2|\Sigma| = 2n$ arguments. If q is a new proposition and $f_0,\ldots, f_{n-1}$ are the encodings of the letters of Σ, then the following formula is satisfiable iff the finite automata A and B recognize different languages.

$$\Diamond q \wedge \neg(\mathscr{G}_0^{A'}(f_0 \wedge \neg q,\ldots, f_{n-1} \wedge \neg q, f_0 \wedge q,\ldots, f_{n-1} \wedge q) \equiv \mathscr{G}_0^{B'}(f_0 \wedge \neg q,\ldots, f_{n-1} \wedge \neg q, f_1 \wedge q,\ldots, f_{n-1} \wedge q)).$$

The formula $\Diamond q$ ensures that we only consider the finite words accepted by the automata. ∎

This proof partially answers the question of what operators can be added to PTL without increasing its complexity. Indeed it shows that satisfiability for ETL is at least as hard as the inequivalence problem for the languages corresponding to the operators. For instance, if we allowed grammar operators corresponding to context-free languages, then ETL would be undecidable. This is investigated further in the context of propositional dynamic logic in Harel, Pnueli, and Stavi (1981).

To prove that ETL is in PSPACE, the techniques developed in Halpern and Reif (1981) are applicable.

LEMMA 8.2. *Satisfiability for* ETL *is in* PSPACE.

Proof. We will show that satisfiability in ETL is in NSPACE and hence by a theorem due to Savitch (1970) in PSPACE. To show that that ETL is in NSPACE, we will give a nondeterministic version of the tableau method that only requires polynomial space. We saw in Section 7 that a satisfiable ETL formula has a model of the form

$$s_0 \to s_1 \to s_2 \to \cdots \overset{\frown}{s_j \to \cdots s_m},$$

where m is less than 2^{9l}. What we will do here is construct this model directly in a nondeterministic way. While doing this, the only formulas we need to remember are those in the current state we are building and in s_j. So the construction proceeds as in the tableau method except that we guess at each stage which successor node to consider and whether a state is s_j or not.

Once we are past s_j, we keep track of which of its eventualities are satisfied. Once all of these are satisfied, and we reach a state s_m that can have s_j as successor, the algorithm terminates and the formula is declared satisfiable. At each point we only need to remember the set of formulas in the current state and in s_j and the maximal length of the path. This can clearly be done in polynomial space. ∎

We can then trivially conclude that ETL is PSPACE-complete.

THEOREM 8.3. *Satisfiability for* ETL *is* PSPACE-*complete.*

9. COMPLETENESS OF THE AXIOMATIC SYSTEM FOR ETL

THEOREM 9.1. *For* ETL, *the axiomatic system consisting of* (N1), (N2); (G1), (G2) *for each grammar operator* $\mathscr{G}_i$ *and the rules of inference* (I1)–(I4) *is complete* (*the numbers refer to the axioms given in Section* 6).

Proof. The proof is based on the tableau decision procedure. We will prove that if the initial node of the graph built by the decision procedure for a formula $\neg f$ is eliminated, then f is provable. Given the completeness of the decision procedure, this implies the completeness of the deductive system.

We will prove by induction that for each eliminated node n labeled by a set of formulas $T_n = \{\neg f_1, \ldots, \neg f_s\}$, $f_1 \vee \cdots \vee f_s$ is provable. This technique is similar to the one introduced in Kozen and Parikh (1981) to prove completeness of the Segerberg axioms for propositional dynamic logic. It is also used in Ben-Ari, Manna, and Pnueli (1981) to prove the completeness of a branching time PTL.

There are four cases to consider:

Case 1. The node was eliminated by rule (E1). By (I1),

$$\vdash p \vee \neg p \vee \cdots \vee f_s. \tag{9.1}$$

Case 2. The node was eliminated by rule (E2) and is not a state. Thus the sons of that node were created using a tableau rule $\neg f \to \{\{\neg f_{ij}\}\}$, for some $\neg f \in T_n$. It can be proved in the axiomatic system (see (6.1)) that for each of these rules

$$\vdash \neg f \equiv \bigvee_i \left(\bigwedge_j \neg f_{ij} \right). \tag{9.2}$$

Thus

$$\vdash \bigwedge_i \left(\bigvee_j f_{ij} \right) \supset f \tag{9.3}$$

and as all the sons of the node have been eliminated, by the induction hypothesis

$$\vdash \bigvee_j f_{ij} \tag{9.4}$$

for all i. From (9.3) and (9.4) we get

$$\vdash f \tag{9.5}$$

from which (I1) and (I2) yield

$$\vdash f_1 \vee \cdots \vee f_s. \tag{9.6}$$

Case 3. The node is eliminated by rule (E2) and is a state. By the induction hypothesis, we have

$$\vdash f'_1 \vee \cdots \vee f'_r, \tag{9.7}$$

where $f'_1 \cdots f'_r$ are the $\bigcirc$-formulas in $T_n = \{f_1,..., f_s\}$ with their outermost $\bigcirc$ removed. From (9.7) it follows by (I3) that

$$\vdash \bigcirc(f'_1 \vee \cdots \vee f'_r) \tag{9.8}$$

and hence by (N1) and (N2)

$$\vdash \bigcirc f'_1 \vee \cdots \vee \bigcirc f'_r \tag{9.9}$$

which yields immediately

$$\vdash f_1 \vee \cdots \vee f_s. \tag{9.10}$$

Case 4. The node n was eliminated by rule (E3). Thus it is labeled by a set of formulas $\{\neg\mathscr{G}^{a_1}_{k_1}(p_{1_1},..., p_{n_1}),..., \neg\mathscr{G}^{a_r}_{k_r}(p_{1_r},..., p_{n_r}), \neg f_{r+1},..., \neg f_s\}$ including the unfulfillable set of eventualities $\{\neg\mathscr{G}^{a_1}_{k_1},..., \neg\mathscr{G}^{a_r}_{k_r}\}$ and we have to prove

$$\vdash \mathscr{G}^{a_1}_{k_1}(p_{1_1},..., p_{n_1}) \vee \cdots \vee \mathscr{G}^{a_r}_{k_r}(p_{1_r},..., p_{n_r}) \vee f_{r+1} \vee \cdots \vee f_s. \tag{9.11}$$

For this we will use the induction axiom (G2). We thus have to prove that for each grammar $G^a \in \{G^{a_1},..., G^{a_r}\}$, there are formulas $u_0,..., u_l$ for which we can prove

$$\vdash \bigwedge_{0 \leqslant k \leqslant l} \Box\left(u_k \supset \bigvee_{1 \leqslant j \leqslant m_k} (p_{kj} \wedge \bigcirc u_{kj})\right). \tag{9.12}$$

Each formula u_i essentially needs to state that $\neg\mathscr{G}^a_i$ is not fulfillable. So, it seems natural to take for u_i the formulas appearing in pre-states accessible

from the node n in which $\neg\mathcal{G}_i^a$ is not fulfillable. Unfortunately, this is not quite enough as there are pre-states where one of the eventualities appearing is not fulfillable but we do not know which one. We will thus use the formulas appearing in pre-states containing $\neg\mathcal{G}_i^a$ augmented with the explicit condition that $\neg\mathcal{G}_i^a$ is not fulfillable. The inductive formulas u_i will then be the disjunction of all such augmented pre-state formulas. More formally, if $\mathcal{N} = \{n_1, ..., n_t\}$ is the set of pre-states accessible from n in which $\neg\mathcal{G}_i^a$ occurs, $\neg f_{m1}, ..., \neg f_{ms_m}$ are the formulas appearing in pre-state $m \in \mathcal{N}$ and NF_i^m is the formula stating that $\neg\mathcal{G}_i^a$ is not fulfillable in m, then

$$u_i \equiv \bigvee_{m \in \mathcal{N}} \left(NF_i^m \wedge \bigwedge_{1 \leqslant k \leqslant s_m} \neg f_{mk} \right). \tag{9.13}$$

We now have to make explicit what the formulas NF_i^m are. First let us define the formula FF_i^m that states that $\neg\mathcal{G}_i^a$ is fulfillable in pre-state m. More precisely, FF_i^m states that $\neg\mathcal{G}_i^a$ is fulfillable on a *proper path* in the graph. A proper path satisfying an eventuality $\neg\mathcal{G}_i^a$ is a path through the graph that does not contain twice the same pre-state with the same set of eventualities to be fulfilled in order to fulfill $\neg\mathcal{G}_i^a$. From Section 7, it follows that a proper path has at most length 2^{5l}. We thus consider all proper paths from m that fulfill $\neg\mathcal{G}_i^a$. For each of these paths we write a formula FF_i^{mq} stating which atomic propositions are true in each state along that path. This formula thus has the form

$$\bigwedge_j (\neg) p_{j0} \wedge \bigcirc \bigwedge_j (\neg) p_{j1} \wedge \cdots \wedge \bigcirc^k \bigwedge_j (\neg) p_{jk}. \tag{9.14}$$

Now, to state that $\neg\mathcal{G}_i^a$ is fulfillable, we will simply take the disjunction of these formulas.

$$FF_i^m \equiv \bigvee_q FF_i^{mq}. \tag{9.15}$$

And we have that

$$NF_i^m \equiv \neg FF_i^m. \tag{9.16}$$

Now that we have defined the formulas u_i, let us prove (9.12). We will actually prove that (9.12) holds if we take as the left side of the implication any of the disjuncts in the definition of u_i. In other words, we will prove that

$$\vdash \left(\left(NF_i^m \wedge \bigwedge_{1 \leqslant k \leqslant s_m} \neg f_{mk} \right) \supset \bigvee_{1 \leqslant j \leqslant m_i} (p_{ij} \wedge \bigcirc u_{ij}) \right). \tag{9.17}$$

If $n_1, ..., n_x$ are the pre-states directly accessible from m, then we can prove, by using the formulas corresponding to the tableau rules that

$$\vdash \left(NF_i^m \wedge \bigwedge_{1 \leqslant n \leqslant s_m} \neg f_{mn} \right) \supset \bigvee_{1 \leqslant j \leqslant x} \bigcirc(\neg f_{j1} \wedge \cdots \wedge \neg f_{js_{nj}}). \tag{9.18}$$

Now, given that we have NF_i^m on the left side of the implication, we can restrict the disjunction on the right to the nodes where $\neg\mathcal{G}_i^a$ or rather the eventualities immediately derivable from that formula (i.e., obtained by one application of the tableau rules) are not fulfillable. Moreover, NF_i^m also implies the condition stating that at least one of those eventualities is not fulfillable. Which means that, using (9.13), we can prove

$$\vdash \left(\left(NF_i^m \wedge \bigwedge_{1 \leqslant k \leqslant s_m} \neg f_{mk} \right) \supset \bigvee_{1 \leqslant j \leqslant m_i} (\bigcirc u_{ij}) \right). \tag{9.19}$$

Finally, we can only have NF_i^m and the corresponding disjunct u_{ij} if also p_{ij}. Thus we have

$$\vdash \left(\left(NF_i^m \wedge \bigwedge_{1 \leqslant k \leqslant s_m} \neg f_{mk} \right) \supset \bigvee_{1 \leqslant j \leqslant m_i} (p_{ij} \wedge \bigcirc u_{ij}) \right) \tag{9.20}$$

and by (I4)

$$\vdash \Box \left(\left(NF_i^m \wedge \bigwedge_{1 \leqslant n \leqslant s_m} \neg f_{mn} \right) \supset \bigvee_{1 \leqslant j \leqslant m_i} (p_{ij} \wedge \bigcirc u_{ij}) \right). \tag{9.21}$$

As this holds for every i, using (9.13) we get our goal (9.12).

As node n was eliminated, at least one of the eventualities $\neg\mathcal{G}_{k_1}^{a_1}, \ldots, \neg\mathcal{G}_{k_r}^{a_r}$ appearing in that node is not fulfillable. Thus we can prove

$$\vdash NF_{k_1}^n \vee \cdots \vee NF_{k_r}^n \tag{9.22}$$

and from this it follows that

$$\vdash (\neg\mathcal{G}_{k_1}^{a_1} \wedge \cdots \wedge \neg\mathcal{G}_{k_r}^{a_r} \wedge \neg f_{r+1} \wedge \cdots \wedge \neg f_s) \supset (u_{k_1} \vee \cdots \vee u_{k_r}). \tag{9.23}$$

Thus using (9.23), (9.12) and the induction axiom (G2) we get

$$\vdash (\neg\mathcal{G}_{k_1}^{a_1} \wedge \cdots \wedge \neg\mathcal{G}_{k_r}^{a_r} \wedge \neg f_{r+1} \wedge \cdots \wedge \neg f_s) \supset (\mathcal{G}_{k_1}^{a_1} \vee \cdots \vee \mathcal{G}_{k_r}^{a_r}) \tag{9.24}$$

from which our final goal (9.11) follows trivially. ∎

To illustrate the preceding proof, we will use our axiomatic system to prove

$$\vdash \mathcal{G}_0^e(p, \neg p) \vee \mathcal{G}_1^e(p, \neg p) \tag{9.25}$$

which is the negation of the formula we used in Section 7 to illustrate the tableau satisfiability algorithm. That formula was found to be unsatisfiable, so (9.25) should be provable. We will try to make clear the connection between the formal proof and the tableau that is the basis of our

completeness proof. In the tableau, the only eliminated nodes are 2 and 8. Node 8 is eliminated because it contains p and $\neg p$. This corresponds simply to the fact that by (I1) we have

$$\vdash p \vee \neg p. \tag{9.26}$$

Node 2 on the other hand is eliminated because it contains a set of eventualities that is not fulfillable ($\{\neg\mathcal{G}_0^e(p, \neg p), \neg\mathcal{G}_1^e(p, \neg p)\}$). This corresponds to the fourth case of our completeness proof and we will be using the induction axiom (G2). Our first task is to choose the formulas u_0 and u_1. We will use the definition (9.13). Let us start with u_0. The only prestate in which $\neg\mathcal{G}_0^e(p, \neg p)$ occurs is node 2. The disjunction appearing in (9.13) then only contains one term and

$$u_0 \equiv NF_0^2 \wedge \neg\mathcal{G}_0^e(p, \neg p) \wedge \neg\mathcal{G}_1^e(p, \neg p). \tag{9.27}$$

To determine NF_0^2, let us consider the paths that fulfill $\neg\mathcal{G}_0^e(p, \neg p)$. There is only one such path, namely $2 \rightarrow 5$. Thus the expression corresponding to (9.14) is simply $\neg p$. And, by (9.15) and (9.16) $NF_0^2 \equiv p$. Hence

$$u_0 \equiv p \wedge \neg\mathcal{G}_0^e(p, \neg p) \wedge \neg\mathcal{G}_1^e(p, \neg p). \tag{9.28}$$

Similarly, we obtain that

$$u_1 \equiv \neg p \wedge \neg\mathcal{G}_0^e(p, \neg p) \wedge \neg\mathcal{G}_1^e(p, \neg p). \tag{9.29}$$

Using the axioms for $\mathcal{G}_0^e(p, \neg p)$ and $\mathcal{G}_1^e(p, \neg p)$, we get

$$\begin{aligned} \vdash u_0 \supset ((\neg p \vee \bigcirc\neg\mathcal{G}_0^e(p, \neg p)) \wedge (\neg p \vee \bigcirc\neg\mathcal{G}_1^e(p, \neg p)) \\ \wedge (p \vee \bigcirc\neg\mathcal{G}_0^e(p, \neg p)) \wedge (p \vee \bigcirc\neg\mathcal{G}_1^e(p, \neg p))). \end{aligned} \tag{9.30}$$

From this, using (I1) and (I2) we can obtain

$$\vdash u_0 \supset (\bigcirc\neg\mathcal{G}_0^e(p, \neg p) \wedge \bigcirc\neg\mathcal{G}_1^e(p, \neg p)). \tag{9.31}$$

Then by using (I1), (I2), (N1), and (N2) we obtain the statement corresponding to (9.18)

$$\vdash u_0 \supset \bigcirc(\neg\mathcal{G}_0^e(p, \neg p) \wedge \neg\mathcal{G}_1^e(p, \neg p)). \tag{9.32}$$

As from (I1) (using (9.28) and (9.29))

$$\vdash u_0 \vee u_1 \equiv (\neg\mathcal{G}_0^e(p, \neg p) \wedge \neg\mathcal{G}_1^e(p, \neg p)) \tag{9.33}$$

we get using (I1), (I2), (N1), and (N2)

$$\vdash u_0 \supset (\bigcirc u_0 \vee \bigcirc u_1). \tag{9.34}$$

And finally, by (9.28) and (9.34)

$$\vdash u_0 \supset ((p \wedge \bigcirc u_0) \vee (p \wedge \bigcirc u_1) \tag{9.35}$$

and hence by (I4)

$$\vdash \Box(u_0 \supset ((p \wedge \bigcirc u_0) \vee (p \wedge \bigcirc u_1)). \tag{9.36}$$

Similarly, we can obtain

$$\vdash \Box(u_1 \supset ((\neg p \wedge \bigcirc u_0) \vee (\neg p \wedge \bigcirc u_1)). \tag{9.37}$$

By (9.33), (9.36), (9.37) and (G2) (with the help of (I1) and (I2)), we get

$$\vdash (\neg \mathscr{G}_0^e(p, \neg p) \wedge \neg \mathscr{G}_1^e(p, \neg p)) \supset (\mathscr{G}_0^e(p, \neg p) \vee \mathscr{G}_1^e(p, \neg p)). \tag{9.38}$$

Which lets us conclude by (I1) and (I2)

$$\vdash \mathscr{G}_0^e(p, \neg p) \vee \mathscr{G}_1^e(p, \neg p). \quad \blacksquare \tag{9.39}$$

10. Conclusions

We have shown that temporal logic could be extended to express properties definable by rigth-linear grammars. This extension does actually increase the expressive power of PTL but without changing the complexity class of the decision procedure which turned out to be useful in some applications (Wolper, 1982a). As compared to using right-linear grammars or regular expressions directly, ETL has the advantage that Boolean combination of properties are directly expressible without the increase in the size of the specifications that can occur with regular expressions.

Since our work combines a modal logic (PTL) and regular languages, it is tempting to try and relate it to dynamic logic (Harel, 1979; Fisher and Ladner, 1979). In dynamic logic, the regular expressions are used to express the control structure of the program and the formulas of the logic only deal with the input/output behaviour of the program, not with general properties of its execution sequence. In our work on the other hand we use the regular language precisely to express properties of these execution sequences. Also, as in all temporal logics, our ETL deals with infinite behaviours. This makes it more similar to PDLΔ (i.e., PDL with an infinite looping operator) for which the best known decision procedure requires time proportional to three exponentials in the length of the formula (Streett, 1981).

Finally, we should mention that the process logic described in Harel, Kozen, and Parikh (1980) also combines regular expressions and PTL. But,

it uses the regular expressions exclusively for describing the program while using PTL for talking about the execution sequence. Also, as far as is currently known it is of nonelementary complexity.

ACKNOWLEDGMENTS

I wish to thank Chris Goad, Ben Moszkowski, Yoni Malachi, Zohar Manna, Peter Pepper, and Frank Yellin for helpful discussions, insightful comments and/or reading a draft of the paper. An anonymous referee also provided some valuable comments.

RECEIVED: February 22, 1983; ACCEPTED: August 5, 1983

REFERENCES

BEN-ARI, M., MANNA, Z., AND PNUELI, A. (1981), The temporal logic of branching time, *in* "Eighth ACM Symposium on Principles of Programming Languages," Williamburg, Va., pp. 164–176.

COHEN, R., AND GOLD, A. (1977), Theory of ω-languages 1. Characterization of ω-context-free languages, *J. Comput. System Sci.* **15**, 169–184.

EMERSON, E. A., AND CLARKE, E. M. (1980), Characterizing correctness properties of parallel programs as fixpoints, *in* "Proceeding, 7th Int. Colloquium on Automata, Languages and Programming," Lecture notes in Computer Science, Vol. 85, pp. 169–181, Springer-Verlag, Berlin.

FISHER, M., AND LADNER, R. (1979), Propositional dynamic logic of regular programs, *J. Comput. System Sci.* **18**, 194–211.

GABBAY, D., PNUELI, A., SHELAH, S., AND STAVI, J. (1980), The temporal analysis of fairness, *in* "Seventh ACM Symposium on Principles of Programming Languages," pp. 163–173, Las Vegas, Nev.

HABERMANN, A. N. (1975), "Path Expressions," Computer Science Report, Carnegie-Mellon University, Pittsburgh, Pa.

HAREL, D. (1979), "First Order Dynamic Logic," Lecture Notes in Computer Science, No. 68, Springer-Verlag, Berlin.

HAREL, D., KOZEN, D., AND PARIKH, R. (1980), Process logic: Expressiveness, decidability, completeness, "Proceedings of the 21st Symposium on Foundations of Computer Science," Syracuse, N. Y., pp. 129–142.

HAREL D., PNUELI, A., AND STAVI, J. (1981), Propositional dynamic logic of context-free programs, *in* "Proceedings of the Twenty-Second Symposium on Foundations of Computer Science," Nashville, Tenn., pp. 310–321.

HALPERN, J. Y., AND REIF, J. H. (1981), The propositional dynamic logic of determinisitc, well-structured programs, *in* "Proceedings of the Twenty-Second Symposium on Foundations of Computer Science," Nashville, Tenn., pp. 322–334.

KAMP, J. A. W. (1968), "Tense Logic and the Theory of Linear Order," PhD thesis, University of California, Los Angeles.

KOZEN, D., AND PARIKH, R. (1981), An elementary proof of the completeness of PDL, *Theoret. Comput. Sci.* **14**, 113–118.

MANNA, Z. (1981), Verification of sequential programs: Temporal axiomatization, *in* "Theoretical Foundations of Programming Methodology" (F. L. Bauer, E. W. Dijkstra, and C. A. R. Hoare, Eds.), NATO Scientific Series, Reidel, Holland.

MEYER, A. R. (1975), Weak monadic second order theory of successor is not elementary recursive, *in* "Proceedings Logic Colloquium, Lecture Notes in Mathematics," Vol. 453, pp. 132–154, Springer-Verlag, Berlin.

MANNA Z., AND PNUELI, A. (1981), Verification of concurrent programs: The temporal framework, *in* "The Correctness Problem in Computer Science" (R. S. Boyer and J. S. Moore, Eds.), pp. 215–273, International Lecture Series in Computer Science, Academic Press, London.

PNUELI, A. (1977), The temporal logic of programs, *in* "Proceedings of the Eighteenth Symposium on Foundations of Computer Science," Providence, RI, pp. 46–57.

PRATT, V. R. (1981), Using graphs to understand PDL, *in* "Proceedings of the Workshop on Logics of Programs," Yorktown-Heights, N. Y., Lecture Notes in Computer Science, Vo. 131, pp. 387–396, Springer-Verlag, Berlin.

RESCHER, N., AND URQUART, A. (1971), "Temporal Logic," Springer-Verlag, Berlin.

SAVITCH, W. J. (1970), Relationship between nondeterministic and deterministic tape complexities, *J. Comput. System Sci.* **4**, 177–192.

SISTLA, A. P., AND CLARKE, E. M. (1982), The complexity of propositional linear temporal logic, "Proceedings of the 14th ACM Symposium on Theory of Computing," San Francisco, Calif.

SHAW, A. C. (1979), "Software Specification Languages Based on Regular Expressions," Technical Report, ETH Zurich.

SMULLYAN, R. M. (1968), "First Order Logic," Springer-Verlag, Berlin.

STREETT, R. (1981), Propositional dynamic logic of looping and converse, *in* "Proceedings of the 13th Symposium on Theory of Computing," Milwaukee, Wisc., pp. 375–383.

WOLPER, P. (1982a), Specification and synthesis of communicating processes using and extended temporal logic, *in* "Ninth Symposium on Principles of Programming Languages," Albuquerque, N. Mex. pp. 20–33.

WOLPER, P. (1982b), "Synthesis of Communicating Processes from Temporal Logic Specifications," PhD thesis, Stanford University, Calif.