THE TABLEAU METHOD FOR TEMPORAL LOGIC: AN OVERVIEW

Pierre WOLPER

Abstract: An overview of the tableau decision method for propositional temporal logic is presented. The method is described in detail for linear time temporal logic. It is then discussed how it can be applied to other variants of temporal logic like branching time temporal logic and extensions of linear time temporal logic. Finally, applications of temporal logics to computer science are reviewed.

1. *Introduction*

Temporal logic (TL) has been studied as a branch of logic for several decades. It was developed as a logical framework to formalize reasoning about time, temporal relations such as "before" or "after", and related concepts such as tenses. TL is closely related to the modal logic of necessity ([HC68]) which attempts to formalize the notions of possible and necessary truth. As temporal logic can in fact be viewed as a special case of modal logic, its origins can also be traced to those of that theory. Tableau decision procedures for temporal and modal logic have been known for some time. An account of earlier work on temporal logic can be found in either the book of Prior ([Pr67]) or of Rescher and Urquhart ([RU71]).

In [Pn77], it was suggested that temporal logic could be a useful tool to formalize reasoning about the execution sequence of programs and especially of concurrent programs. In that approach, the sequence of states a machine goes through during a computation is viewed as the temporal sequence of worlds described by TL. Since then, several researchers have been using TL to state and prove properties of concurrent programs (*e.g.* [GPSS80], [La80], [OL80], [BP80], [MP81], [CES83], [MP83], [MW84]), protocols (*e.g.* [Ha80], [HO81], [SM81], [SM82], [Vo82], [SS82], [SMV83], [SPE84], and hardware (*e.g.* [Bo82], [MO81], [HMM83], [Mo83]).

In this paper, we first define the propositional version of linear time temporal logic. We then describe the tableau decision procedure for PTL. We also review other variants of temporal logics and state results about their decision problems. And finally, we discuss the use of temporal logic in computer science.

2. *Propositional Temporal Logic*

We define here the propositional version of temporal logic (PTL). We have chosen one of the most common versions appearing in the computer science literature. In contrast to the temporal logics studied in philosophical logic it contains only temporal operators dealing with the "future" and no operators concerning the "past".

Syntax: PTL *formulas* are built from

- A set $\mathscr{P}$ of atomic propositions: $p_1, p_2, p_3, \ldots$
- Boolean connectives: $\wedge$, $\neg$.
- Temporal operators: $\bigcirc$ ("next"), $\Diamond$ ("eventually"), U ("until").

The formation rules are:

- An atomic proposition $p \in \mathscr{P}$ is a formula.
- If f_1 and f_2 are formulas, so are $f_1 \wedge f_2$, $\neg f_1$, $\bigcirc f_1$, $\Diamond f_1$, $f_1 U f_2$.

We use $\Box$ ("always") as an abbreviation for $\neg\Diamond\neg$. We also use $\vee$ and $\supset$ as the usual abbreviations, and parentheses to resolve ambiguities.

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

- S is a finite or enumerable set of states.
- N: $(S \rightarrow S)$ is a total successor function that for each state gives a unique next state.
- π: $(S \rightarrow 2^{\mathscr{P}})$ 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 i^{th} state in the sequence

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

of successsors of a state s.

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

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". The satisfiability problem for PTL is, given a formula f, to determine if there is some interpretation that satisfies f (*i.e.*, a *model* of f). In the next section, we describe the tableau method for the satisfiability problem of PTL.

Examples:

1) The formula

$$p$$

is satisfied by all sequences in which p is true in the first state.

2) The formula

$$\Box(p \supset \bigcirc q)$$

is satisfied by all sequences where each state in which p is true is followed by a state in which q is true.

3) The formula

$$\Box(p \supset \bigcirc(\neg q\, U\, r))$$

is satisfied by all sequences where if p is true in a given state, then, from the next state on, q is always false until the first state where r is true.

4) The formula

$$\Box\Diamond p$$

is satisfied by all sequences in which p is true infinitely often.

5) The formula

$$\Box p \wedge \Diamond\neg p$$

is not satisfied by any sequence.

6) The formula

$$\Diamond p \supset (\neg p\, U\, p)$$

is satisfied by all sequences (it is valid).

3. *The Tableau Method for PTL*

The tableau method for PTL is an extension of the tableau method for propositional logic (see [Sm68]). Boolean connectives are handled exactly as for propositional logic and temporal connectives are handled by decomposing them into a requirement on the "current state" and a requirement on "the rest of the sequence". This decomposition of the temporal connectives makes the tableau into a state by state search for a model of the formula being considered. The decomposition rules for the temporal operators are based on the following identities:

$$\Diamond f \equiv f \vee \bigcirc\Diamond f \tag{1}$$

$$f_1\, U\, f_2 \equiv (f_2 \vee (f_1 \wedge \bigcirc(f_1\, U\, f_2))). \tag{2}$$

Given that decomposing formulas using identities (1-2) does not make them smaller, the tableau might be infinite. However, we will insure that the tableau is finite by identifying the nodes of the tableau that are labeled by the same set of formulas. Note that in the tableau method for propositional logic this is not necessary as the tableau is always a finite tree. Another difference is that obtaining a tableau with no propositional inconsistencies for a formula f is not a guarantee that f is satisfiable. Indeed, formulas of the form $\Diamond f_1$ or $(f_1 \, U \, f_2)$ which we call *eventualities* might not be satisfied. The problem is that, for instance for a formula $\Diamond f_1$, the tableau rule based on (1) will allow us to continuously postpone the point at which f_1 is satisfied. This corresponds to always choosing the $\bigcirc \Diamond f$ disjunct in (1). We will thus have to add an extra step to the tableau method which will eliminate nodes containing eventualities that are not satisfiable.

More precisely, to test a PTL formula f for satisfiability, the tableau method constructs a directed graph. Each node n of the graph is labeled by a set of formulas T_n. Initially, the graph contains exactly one node, labeled by $\{f\}$. To describe the construction of the graph, we distinguish between elementary and non-elementary formulas. Elementary formulas are those whose main connective is $\bigcirc$ (we call these $\bigcirc$-formulas) or that are either atomic propositions or negations of atomic propositions. The construction of the graph proceeds by using the following decomposition rules which map each non-elementary formula f into a set Σ of sets S_i of formulas f_j:

$$
\begin{array}{lcl}
\neg\neg f & \rightarrow & \{\{f\}\} \\
\neg\bigcirc f & \rightarrow & \{\{\bigcirc \neg f\}\} \\
f_1 \wedge f_2 & \rightarrow & \{\{f_1, f_2\}\} \\
\neg(f_1 \wedge f_2) & \rightarrow & \{\{\neg f_1\}, \{\neg f_2\}\} \\
\Diamond f & \rightarrow & \{\{f\}, \{\bigcirc \Diamond f\}\} \\
\neg\Diamond f & \rightarrow & \{\{\neg f, \neg\bigcirc \Diamond f\}\} \\
f_1 \, U \, f_2 & \rightarrow & \{\{f_2\}, \{f_1, \bigcirc (f_1 \, U \, f_2\}\} \\
\neg(f_1 \, U \, f_2) & \rightarrow & \{\neg f_2, \neg f_1 \vee \neg\bigcirc (f_1 \, U \, f_2)\}
\end{array}
$$

During the construction, to avoid decomposing the same formula twice, we will mark the formulas to which a decomposition rule has been applied (we don't simply discard them as we will need them when checking if eventualities are satisfied). 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 non-elementary formula f and the tableau rule for f is $f \rightarrow \{S_i\}$, 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 and 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 node 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 subformulas or negations of subformulas of the initial formula or such formulas preceded by $\bigcirc$. The number of these formulas is equal to $4l$, where l is the length of the initial formula. 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 formula of the form $\Diamond f$ or $f_1 \, U \, f_2$ that is not satisfiable (see below), eliminate that node.

To determine if a formula $\Diamond f$ or $f_1 \, U \, f_2$ is satisfiable, one uses the following rule:

F1: A formula $\Diamond f_2$ or $f_1 \, U \, f_2$ is satisfiable in a pre-state, if there is a path in the tableau leading from that pre-state to a node containing the formula f_2.

The decision procedure ends after all unsatisfiable nodes have been eliminated. If the initial node has been eliminated, then the initial formula is unsatisfiable, if not it is satisfiable.

Example:

Consider the formula $\Box p \wedge \Diamond \neg p$. The tableau obtained for this formula by the algorithm we have just described is the one appearing in figure 1.

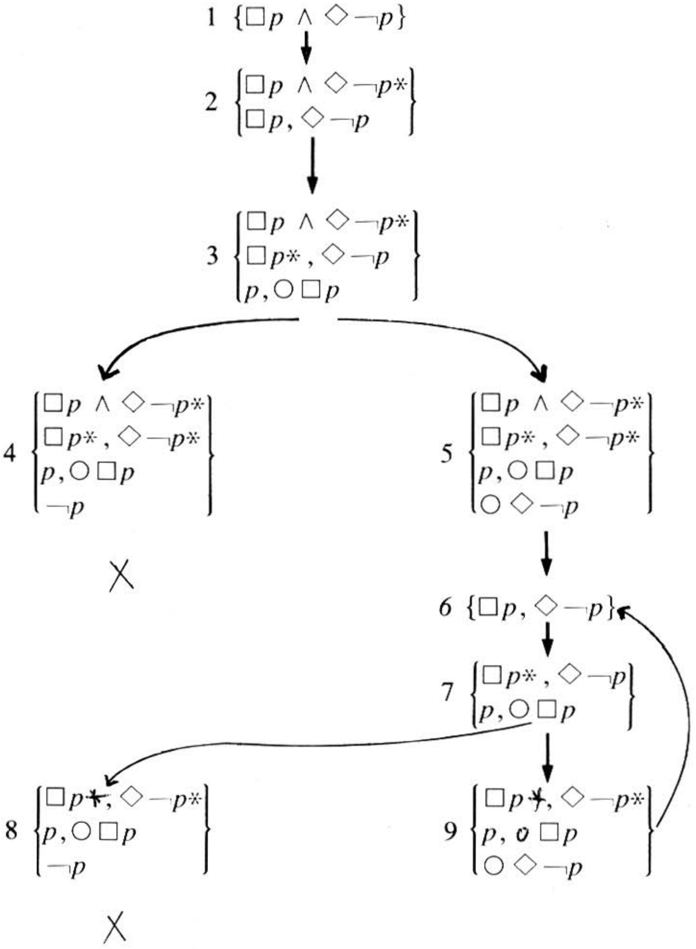
1 $\{\Box p \wedge \Diamond \neg p\}$
2 $\Box p \wedge \Diamond \neg p*$, $\Box p, \Diamond \neg p$
3 $\Box p \wedge \Diamond \neg p*$, $\Box p*, \Diamond \neg p$, $p, \bigcirc \Box p$
4 $\Box p \wedge \Diamond \neg p*$, $\Box p*, \Diamond \neg p*$, $p, \bigcirc \Box p$, $\neg p$
X
5 $\Box p \wedge \Diamond \neg p*$, $\Box p*, \Diamond \neg p*$, $p, \bigcirc \Box p$, $\bigcirc \Diamond \neg p$
6 $\{\Box p, \Diamond \neg p\}$
7 $\Box p*, \Diamond \neg p$, $p, \bigcirc \Box p$
8 $\Box p*, \Diamond \neg p*$, $p, \bigcirc \Box p$, $\neg p$
X
9 $\Box p*, \Diamond \neg p*$, $p, \bigcirc \Box p$, $\bigcirc \Diamond \neg p$

Figure 1

In this tableau, the initial node is 1; the states are 4, 5, 8 and 9; and the pre-states are 1 and 6. Nodes 4 and 8, contain a proposition (p) and its negation, they are thus unsatisfiable and are eliminated by rule E1. Node 6 is a pre-state and contains an eventuality formula ($\Diamond \neg p$) that is not satisfiable as there is no path leading from 6 to a node containing $\neg p$ (node 8 is eliminated). The other nodes are then eliminated by rule E2 and the formula is found to be unsatisfiable.

It is easy to see that the decision procedure requires time and space exponential in the length l of the initial formula. Actually, it is possible to test a PTL formula for satisfiability using only polynomial space. The satisfiability problem for PTL is in fact complete in PSPACE. For a discussion of the complexity of PTL, see [SC82] and [WVS83]. The correctness of the tableau method we have just described is established by the following theorem:

Theorem 1: An PTL 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 $\{f_1, \ldots, f_s\}$ is eliminated, then $\{f_1, \ldots, f_s\}$ 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 $f \rightarrow \{S_i\}$. 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 initial 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 an eventuality in the node that is not satisfiable on any path in the tableau. As any model corresponds to some path in the tableau, the eventuality 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 satisfying evenualities, 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 satisfied. It can be done as follows.

For each pre-state in the graph, unwind a path from that pre-state such that all the eventualities it contains are satisfied on that path. This is done by satisfying the eventualities one by one. If one of the eventualities is selected, it is possible to find a path on which that eventuality is satisfied and whose last node is a pre-state (if not the node would have been eliminated). Now, given the tableau rules, the eventualities that are not satisfied will appear in the last node of that path and hence the path can be extended to satisfy a second eventuality. By repeating this construction, one obtains a path on which all eventualities are satisfied. Once all these paths are constructed, we link them together. The model obtained has the following form:

$$\begin{tikzcd} s_0 \arrow[r] & s_1 \arrow[r] & s_2 \arrow[r] & \ldots . s_j \arrow[r] & \ldots . s_m \arrow[l, bend right] \end{tikzcd}.$$

A complete axiomatization of PTL can be obtained from the tableau method in the usual way. The only point worthy of interest is the axiom corresponding to the elmination rule E3. That axiom is basically an induction axiom for the $\diamond$ operator. [Wo83] contains a complete axiomatization of PTL and a proof of completeness.

4. *Other Temporal Logics*

Extensions of Linear Time Temporal Logic

A simple property like *even* (p), meaning that p is true in every even state of a sequence and might be true or false in odd states is not expressible in the temporal logic we have discussed in the previous section. This led to the definition of an extended temporal logic (ETL) in [Wo83]. The extended temporal logic is based on the observation that one can extend PTL with operators corresponding to arbitrary right-linear grammars. The operators $\bigcirc$, $\diamond$, and U of PTL are in fact special cases of this more general class of operators. One of the important features of ETL is that it also has a tableau decision procedure similar to the one we have described for PTL (see [Wo83]). In [WVS83], several alternative definitions of ETL are considered and their expressiveness and complexity are characterized.

Quantified Temporal Logic

Another way to extend linear time propositional temporal logic is to allow quantification of propositional variables. The resulting logic, quantified propositional temporal logic (QPTL) is decidable, but unfortunately is of non-elementary complexity. The known decision procedures are by reduction to SnS or by using automata theoretic techniques ([Si83], [Wo82]). Interestingly, the expressiveness of QPTL and of ETL is the same [WVS83].

Branching Time Temporal Logics

In the interpretations of linear time temporal logic, each state has exactly one successor. That is the reason for the name "linear time". If one deals with a situation where there are several possible futures (in computer science, this is the case if one deals with a non-deterministic program), the "linear time" assumption no longer holds. This led to the development of branching time temporal logics (BTL) that are interpreted over structures where each state can have several successors.

Formulas of branching time temporal logic are similar to formulas of linear time temporal logic with the addition of *path quantifiers*. Path quantifiers ($\forall$ and $\exists$) are used to specify to which paths the temporal logic formula applies. For example, $\forall \Diamond p$ is true in a state, if on all paths from that state there is a state satisfying p. Depending on how the path quantifiers and temporal logic formulas are allowed to interact, one defines a variety of branching time temporal logics.

One of the simplest of the branching time temporal logics is the logic *UB* described in [BMP81]. In that logic, path quantifiers and temporal operators are required to always appear together. In other words, *UB* can be viewed as PTL where each of the temporal operators $\bigcirc$, $\Diamond$, and U is replaced by two operators, $\forall\bigcirc$ and $\exists\bigcirc$, $\forall\Diamond$ and $\exists\Diamond$, $\forall U$ and $\exists U$. The logic *UB* also has a tableau decision procedure similar to the one described here for PTL, though slightly more complicated as the logic is interpreted over branching rather than linear structures. *UB* also has a simple complete axiomatization. However, the complexity of the decision problem for *UB* is EXPTIME rather than PSPACE as it is for the linear time temporal logics. In [EH82], one can find a thorough description of tableau-like decision procedures for branching time temporal logic.

At the other end of the spectrum is the logic *CTL** described in [EH83]. In *CTL**, one allows path quantifiers to apply to arbitrary linear time temporal logic formulas. For instance $\forall(\Box\Diamond p \wedge q\,U(\bigcirc r))$ is a *CTL** formula. *CTL** is strictly more expressive than *UB*. Also, it is possible to define several logics that are in between *CTL** and *UB* as far as expressiveness. [EH83] discusses all these logics and compares their expressiveness. As far as decision procedures, the tableau method that we presented here does not extend naturally to *CTL**. At this point the only decision procedures that are known for *CTL** are based on automata theoretic methods and require time triply or quadruply exponential (see [ES84], [VW83] and [VW84] for a description of these decision procedures). No simple complete axiomatization is known for *CTL**. Also, no precise caracterization of its complexity is known. The best lower bound obtained is exponential, whereas the best upperbound is triply exponential.

Interval Temporal Logic

A variation of temporal logic that has appeared recently is interval temporal logic. Interval temporal logic is a variant of linear time temporal logic that allows explicit description of intervals. Two variants are currently in existence. One described in [SMV83] was developed as a more convenient higher level extension of PTL for the specification and verification of protocols. Its formulas combine the description of an interval and a temporal logic statement concerning that interval. For instance the formula $[I]\,\Box p$ states that the first time the interval I appears, it satisfies $\Box p$ (*i.e.*, all its states satisfy p). It is not more expressive than PTL and there is a translation from its formulas to PTL formulas. This gives a decision procedure for the logic though it is no more a tableau decision procedure closely linked to the syntax of the logic [Pl83]. This interval temporal logic has no known simple complete axiomatization.

A second interval temporal logic is described in [HMM83] and

[Mo83]. It was designed with the description of hardware as a goal. In this logic, all statements are about intervals. Its fundamental operations are ○ which here maps an interval into its tail (the same interval with the first state removed) and concatenation of intervals (;). These two simple constructs make it into a very powerful language. Unfortunately, it is undecidable in the general case and, the only known decidable subset is of non-elementary complexity (see [Mo83]).

Probabilistic Temporal Logic

Yet another variant of temporal logic is probabilistic temporal logic. Its development has been motivated by the appearance of probabilistic algorithms. As branching time temporal logic, probabilistic temporal logic is interpreted over structures in which states have more than one successor. The difference is that a probability is associated with each transition. The formulas of the logic then state that a given linear time temporal logic formula holds on a set of paths that has probability one. Three different variants of probabilistic temporal logic are described in [LS82]. For each of these a complete axiomatization is given. In [KL83] double exponential decision procedures are given for these logics.

First Order Temporal Logic

Up to this point we have been talking about propsitional temporal logics. However, first order temporal logics are often used when for instance stating properties of programs. Unfortunately, though axiomatizations for first order temporal logics have been proposed (*e.g.*, [Ma81]), they are not complete.

5. *Use of Temporal Logic in Computer Science*

In the last few years, temporal logic has been used in several different areas of computer science. The main ones are the following.

Stating and Proving Properties of Concurrent Programs

This was the initial motivation when temporal logic was introduced to the computer science community in [Pn77]. When reasoning about a concurrent program, it is not sufficient to deal with its input output behavior, one has to consider the entire computation sequence. As temporal logic is geared towards describing sequences, it appeared well suited for this problem.

In this approach, one views the execution sequence of a program as the sequence of states TL describes and one can state properties of that sequence. As TL formulas do not allow us to explicitely represent the program, it has to be encoded in a set of statements that basically represent the allowable transitions in each state. This approach has been further developed in [MP81], and [MP83].

Related methods for specifying and proving the correctness of concurrent programs are described in [OL82] and [La83]. In [OL82] a proof method called *proof lattices* was introduced.

A method to check that finite state programs satisfy some temporal logic specifications was proposed in [CES83]. The idea is that a finite state program can be viewed as a structure over which temporal logic formulas can be interpreted. The problem of checking that the program satisfies a given temporal formula is then equivalent to the problem of checking that the structure corresponding to that program is a model of the formula. In [CES83], it was pointed out that if one uses the branching time logic *UB*, this problem is in polynominal time, which leads to attractive algorithms. Similar ideas were developed in [QS82].

Another application related to the one we are now describing is the study of fairness conditions and properties. When several processes are running concurrently, the outcome of executing the program can depend on how ressources are allocated to the various processes. For example, if ressources are allocated evenly to all processes, the program might terminate whereas if one of the processes does not receive any ressources the program might get blocked. This had led to specifying *fairness conditions* on the execution of concurrent programs. These fairness conditions require a somewhat even distribution of ressources among the various processes. A large number of different conditions have been proposed. Temporal logic has appeared to be a useful tool for stating and reasoning about these conditions (see [LPS81], [QS82b], and [Pn83]).

Synthesis of Concurrent Programs

A direct use of the decision procedure we described in this paper has been the synthesis of the synchronization part of concurrent programs. If one assumes that the various parts of a concurrent program only interact through a finite number of signals, then their interaction can be specified in propositional temporal logic. Now, if one applies the tableau decision procedure to this specification, one obtains a graph that can be viewed as a program satisfying those specifications. Indeed, all executions of the program (paths through the graph) satisfy the specification (if one ensures that eventualities are satisfied). This approach was developed in [Wo82] and [MW84] using a linear time temporal logic and in [CE81] using a branching time temporal logic. A more informal approach to synthesis from temporal logic specifications appears in [RK80].

Specification and Correctness of Protocols

Communication protocols are another area where temporal logic has been applied. Protocols can often be hard to implement correctly and analyse. This is due to the fact that they are often quite intricate and can exhibit unexpected behaviors. This has led to a lot of interest in formal methods to specify and reason about protocols. Among these methods temporal logic has played an important role. Again it is its ability to describe sequences (*e.g.*, the sequence of communications a protocol performs) that has made it attractive for this application (see [Hai80], [HO81], [SM81], [SS82], [SMV83], [SPE84]).

Hardware

The development of always larger integrated circuits has made the need for tools to specify and reason about such circuits more and more necessary. Several researchers have tried to apply temporal logic to this problem ([Bo82], [MO81], [HMM83[, [Mo83]). The technique of model checking introduced in [CES83] for concurrent programs was applied to circuits in [CM84].

AT&T Bell Laboratories
Murray Hill, NJ 07974

Pierre WOLPER

REFERENCES

[BMP81] M. Ben-Ari, Z. Manna, A. Pnueli, "The Temporal Logic of Branching Time". *Eighth ACM Symposium on Principles of Programming Languages*, Williamsburg, VA, January 1981, pp. 164-176.

[Bo82] G.V. Bochmann, "Hardware Specification with Temporal Logic: An Example", *IEEE Transactions on Computers*, C-31(3), March 1982, pp. 223-231.

[BP80] M. Ben-Ari, A. Pnueli, "Temporal Logic Proofs of Concurrent Programs", Technical Report, Department of Mathematical Sciences, Tel-Aviv University, 1980.

[CE81] E.M. Clarke, E.A. Emerson, "Synthesis of Synchronization Skeletons from Branching Time Temporal Logic", in *Logics of Programs*, Lecture Notes in Computer Science vol. 131, Springer-Verlag, Berlin, 1982, pp. 52-71.

[CES83] E.M. Clarke, E.A. Emerson, A.P. Sistla, "Automatic Verification of Finite-state Concurrent Systems Using Temporal Logic Specifications: A Practical Approach", *Proc. of the 10th ACM Symposium on Principles of Programming Languages*, Austin, January 1984, pp. 117-126.

[CM84] E. Clarke, S. Mishra, "Automatic Verification of Asynchronous Circuits", in *Logics of Programs*, Springer-Verlag Lecture Notes in Computer Science, Vol. 164, Berlin, 1984, pp. 101-115.

[EC80] E.A. Emerson, E.M. Clarke, "Characterizing Correctness Properties of Parallel Programs as Fixpoints", *Proc. 7th Int. Colloquium on Automata, Languages and Programming*, Lecture notes in Computer Science vol. 85, Springer-Verlag, Berlin, 1981, pp. 169-181.

[EH82] E.A. Emerson, J.Y. Halpern, "Decision Procedures and Expressiveness in the Temporal Logic of Branching Time", *Proceedings of the 14th Symposium on Theory of Computing*, San Francisco, CA, May 1982, pp. 169-180.

[EH83] E.A. Emerson, J.Y. Halpern, "Sometimes and Not Never Revisited: On Branching Versus Linear Time", *Proceedings of the 10th Symposium on Principles of Programming Languages*, Austin, January 1983, pp. 127-140.

[ES84] E.A. Emerson, A.P. Sistla, "Deciding Branching Time Logic", *Proc. 16th ACM Symposium on Theory of Computing*, Washington, May 1984, pp. 14-24.

[GPSS80] D. Gabbay, A. Pnueli, S. Shelah and J. Stavi, "The Temporal Analysis of Fairness", *Seventh ACM Symposium on Principles of Programming Languages*, Las Vegas, NV, January 1980, pp. 163-173.

[Hai80] B.T. Hailpern, "Verifying Concurrent Processes Using Temporal Logic", Ph.D. Thesis, Stanford University, 1980.

[HC68] G.E. Hughes, M.J. Cresswell, *An Introduction to Modal Logic*, Methuen and Co, London, 1968.

[HMM83] J. Halpern, Z. Manna, B. Moszkowski, "A Hardware Semantics Based on temporal Intervals", *Proc. 10th International Colloquium on Automata Languages and Programming*, Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1983.

[HO81] B.T. Hailpern, S. Owicki, "Modular Verification of Computer Communication Protocols", Research Report RC 8726, IBM T.J. Watson Research Center, March 81.

[KL83] S. Kraus, D. Lehman, "Decision Procedures for Time and Chance", *Proc. 24th IEEE Symposium on Foundations of Computer Science*, Tucson, November 1983, pp. 202-209.

[Lam80] L. Lamport, "Sometimes is Sometimes Not Never", *Seventh ACM Symposium on Principles of Programming Languages*, Las Vegas, NV, January 1980, pp. 174-185.

[LPS81] D. Lehman, A. Pnueli, J. Stavi, "Impartiality, Justice and Fairness: The Ethics of Concurrent Termination", *Proc. 8th International Colloquium on Automata languages and Programming*, Lecture Notes in Computer Science, Springer-Verlag, Berlin 1981, pp. 264-277.

[LS82] D. Lehman, S. Shelah, "Reasoning with Time and Chance", *Information and Control*, Vol. 53, 1982, pp. 165-198.

[Ma81] Z. Manna, "Verification of Sequential Programs: Temporal Axiomatization", *Theoretical Foundations of Programming Methodology* (F.L. Bauer, E.W. Dijkstra, C.A.R. Hoare, eds.), NATO Scientific Series, D. Reidel Pub. Co., Holland, 1981.

[Mo83] B. Moszkowski, "Reasoning about Digital Circuits", Ph.D. Thesis, Department of Computer Science, Stanford University, July 1983.

[MO81] Y. Malachi and S. Owicki "Temporal Specifications of Self-Timed Systems" in *VLSI Systems and Computations*, H.T. Kung, Bob Sproul, and Guy Steele editors, Computer Science Press 1981, pp. 203-212.

[MP81] Z. Manna, A. Pnueli, "Verification of Concurrent Programs: the Temporal Framework", *The Correctness Problem in Computer Science* (R.S. Boyer and J.S. Moore, eds.), International Lecture Series in Computer Science, Academic Press, London, 1981, pp. 215-273.

[MP83] Z. Manna, A. Pnueli, "How to Cook a Temporal Proof System for your Pet Language, *Proc. of the 10th ACM Symposium on Principles of Programming Languages*, Austin, January 1984, pp. 141-154.

[MW84] Z. Manna, P. Wolper, "Synthesis of Communicating Processes from Temporal Logic Specifications", *ACM Transactions on Programming Languages and Systems*, Vol. 6, No. 1, January 1984, pp. 68-93.

[OL82] S. Owicki, L. Lamport, "Proving Liveness Properties of Concurrent Programs", *ACM Transactions on Programming Languages and Systems*, Vol. 4, No. 3, 1982, pp. 455-496.

[Pl83] D. Plaisted, "An Intermediate-Level Language for Obtaining Decision Procedures for a Class of Temporal Logics", Technical Report, Computer Science Laboratory, SRI, 1983.

[Pn77] A. Pnueli, "The Temporal Logic of Programs", *Proceedings of the Eighteenth Symposium on Foundations of Computer Science*, Providence, RI, November 1977, pp. 46-57.

[Pn83] A. Pnueli, "On the Extremely Fair Treatment of Probabilistic Algorithms", *Proc. 15th ACM Symposium on Theory of Computing*, Boston, April 1983, pp. 278-289.

[Pr67] A. Prior, *Past, Present and Future*, Oxford University Press, 1967.

[QS82] J.P. Queille, J. Sifakis, "Specification and Verification of Concurrent Systems in CESAR", Lecture Notes in Computer Science, Vol. 137 Springer-Verlag, Berlin, 1982, pp. 337-351.

[QS82b] J.P. Queille, J. Sifakis, "A Temporal Logic to Deal with Fairness in Transition Systems", *Proc. IEEE Symposium on foundations of Computer Science*, Chicago, 1982, pp. 217-225.

[RK80] K. Ramamritham, R.M. Keller, "Specification and Synthesis of Synchronizers", *Proceedings International Symposium on Parallel Processing*, August 1980, pp. 311-321.

[RU71] N. Rescher, A. Urquart, *Temporal Logic*, Springer-Verlag, 1971.

[SC82] A.P. Sistla, E.M. Clarke, "The Complexity of Propositional Linear Temporal Logic", *Proceedings of the 14th ACM Symposium on Theory of Computing*, San Francisco, CA, May 1982, pp. 159-168.

[Si83] A.P. Sistla, "Theoretical Issues in the Design and Verification of Distributed Systems", Ph.D. Thesis, Harvard University, 1983.

[Sm68] R.M. Smullyan, *First Order Logic*, Springer-Verlag, Berlin, 1968.

[SM81] R.L. Schwartz, P.M. Melliar-Smith, "Temporal Logic Specification of Distributed Systems", *Proceedings of the Second International Conference on Distributed Systems*, Paris, France, April 1981.

[SM82] R.L. Schwartz, P.M. Melliar-Smith, "From State Machines to Temporal Logic: Specification Methods for Protocol Standards", *IEEE Transactions on Communications*, December 1982.

[SMV83] R.L. Schwartz, P.M. Melliar-Smith, F.H. Vogt, "An Interval Logic for Higher-Level Temporal Reasoning", *Proc. 2nd ACM Symposium on Principles of Distributed Computing*, Montreal, Canada, August 1983, pp. 173-186.

[SPE84] D.E. Shasha, A.Pnueli, W. Ewald, "Temporal Verification of Carrier-Sense Local Area Network Protocols", *Proc. 11th ACM Symposium on Principles of Programming Languages*, Salt Lake City, January 1984, pp. 54-65.

[SS82] K. Sabnani, M. Schwartz, "Verification of a Multidestination Protocol Using Temporal Logic", in *Protocol Specification Testing and Verification*, C. Sunshine ed., North-Holland, 1982, pp. 21-42.

[Vo82] F.H. Vogt, "Event-Based Temporal Logic Specification of Services and Protocols", in *Protocol Specification, Testing and Verification*, North-Holland Publishing, 1982.

[VW83] M.Y. Vardi, P. Wolper, "Yet Another Process Logic", in *Logics of Programs*, Springer-Verlag Lecture Notes in Computer Science, Vol. 164, Berlin, 1984, pp. 501-512.

[VW84] M.Y. Vardi, P. Wolper, "Automata Theoretic Techniques for Modal Logics of Programs", *Proc. 16th ACM Symposium on Theory of Computing*, Washington, May 1984, pp. 446-456.

[Wo82] P. Wolper, "Synthesis of Communicating Processes from Temporal Logic Specifications", Ph.D. Thesis, Stanford University, August 1982.

[Wo83] P. Wolper, "Temporal Logic Can Be More Expressive", *Information and Control*, Vol. 56, Nos. 1-2, 1983, pp. 72-99.

[WVS83] P. Wolper, M.Y. Vardi, A.P. Sistla, "Reasoning about Infinite Computation Paths", *Proc. 24th IEEE Symposium on Foundations of Computer Science*, Tucson, 1983, pp. 185-194.