Efficient Temporal Reasoning[†]

(Extended Abstract)

E. Allen EMERSON[1,2] *Tom SADLER*[1] *Jai SRINIVASAN*[1]

1. Department of Computer Sciences, The University of Texas at Austin, USA
2. Mathematics and Computing Science Department, Technical University of Eindhoven, The Netherlands

Abstract

There has been much interest in decision procedures for testing satisfiability (or validity) of formulae in various systems of Temporal Logic. This is due to the potential applications of such decision procedures to mechanical reasoning about correctness of concurrent programs. We show that there exist Temporal Logics that are (*i*) decidable in polynomial time, and (*ii*) still useful in applications. One surprising corollary of our results is that the fragment of CTL (Computation Tree Logic) actually used by Emerson & Clarke [EC82] to synthesize concurrent programs from temporal specifications is decidable in polynomial time. Another is that the verification of many correctness properties of concurrent programs (such as in Owicki & Lamport [OL82]) can be efficiently automated. This demonstrates that many useful correctness properties can be expressed with a rather restricted syntax. Finally, our results provide insight into the relation between the structural (i.e., syntactic) complexity of temporal logics and the complexity of their decision problems.

1 Introduction

Over the last decade there has been a great deal of research into the complexity of decision procedures for testing satisfiability (or validity) in various systems of modal or temporal logic, owing to their potential applications to reasoning about concurrent programs (cf. [Pn77], [FL79], [Ab80], [BHP81], [EH82], [VW84], [ES84], [VS85], [CVW86], [HV86], [EJ88]). For all extant propositional logics of programs, the complexity of testing satisfiability is at least $\mathcal{NP}$-hard since these logics subsume pure propositional logic. Assuming $\mathcal{P} \neq \mathcal{NP}$, it thus seems most likely that the best deterministic time decision procedure we can hope for is of exponential complexity. The trend, therefore, has been to consider logics of increasingly greater expressive power, and recent work has concentrated on the problem of obtaining "efficient" exponential time decision procedures for logics of quite high expressive power. For logics of very high expressive power, it is difficult to get even a double or multi-exponential decision procedure.

Despite the fact that even single exponential time complexity can be a formidable obstacle to serious applications, consideration of such exponential time logics for potential use in applications has been justified on several grounds. First, it does seem to be the best we can do in general. Secondly, these are the worst case complexity measures; the "average" complexity may be lower. Empirical evidence does show that the complexity on many interesting example specifications is often much better than the exponential worst case. In this paper, we provide a surprising and convincing argument explaining why the observed complexity is much better than expected.

Contrary to the trend toward greater expressive power, we focus on the basic correctness properties essential to the majority of applications, and restrict our logic's expressive power accordingly. By appropriately restricting the syntax, we can get a logic that (*i*) is decidable in polynomial time, and (*ii*) still has useful applications to reasoning about concurrent programs. Our results show, for example, that the satisfiability

[†]This work was supported in part by NSF grant DCR-8511354, ONR URI contract N00014-86-K-0763, and Netherlands NWO grant nf-3/nfb 62-500.

problem for the fragment of the logic actually used for program synthesis in [EC82] is decidable in polynomial time, and that the proofs of some liveness properties established for example programs in [OL82] can be efficiently automated.

The basic intuition is to restrict the syntax so that not all of propositional logic is included as a sublogic. Instead, we allow formulae to be built up from Horn propositional logic, interacting with each other so as to limit the depth of nesting of the basic temporal operators F (sometime), G (always), X (next time), and U (until). Among the temporal formulae still allowed are such "simple" but important ones as Gp (always p, the prototypical safety property), $G(p \Rightarrow Fq)$ (temporal implication or the leads-to assertion, a fundamental liveness property), and $G(p \Rightarrow Xq)$ (temporal succession; this describes immediate successor states).

The success of our approach hinges on our ability to separate the problem of propositional reasoning from that of temporal reasoning. To understand this, it is helpful to think in terms of the automata-theoretic temporal framework (cf. [St82], [VW84], [Em85], [VW86], [Va87]). The standard paradigm for developing a decision procedure for a temporal logic involves two steps:

a) Reduce the satisfiability problem to the problem of testing non-emptiness of a finite state automaton on infinite objects. In the case of linear temporal logic this is an automaton on infinite strings, while in branching time, it is a tree automaton. The automaton is typically built by constructing a tableau. Intuitively, the tableau is a finite graph which encodes all potential models of the formula, and is derived from the syntax of the formula. It may be viewed as the transition diagram of an automaton that accepts as input the models of the formula.

b) Test the automaton for non-emptiness. This involves a graph reachability analysis, which is performed by a pruning procedure to ensure that each eventuality is fulfilled. For example, if a node of the automaton indicates that Fp (sometime p) should hold, then it is checked that there is indeed a path through the automaton leading to a node where p holds; if not, the node is deleted. This process is repeated until no more nodes can be deleted. For many logics, such a test for non-emptiness can be done in time polynomial in the automaton's size.

In general, the time to construct the automaton is proportional to its size. Thus, the total complexity is governed by the size of the automaton and the time to test non-emptiness. Since the automaton is usually of exponential size in the length of the formula, even though testing non-emptiness can be done in time polynomial in the automaton's size, the total time complexity is exponential. However, we show that there exist useful logics—still able to express important correctness properties of concurrent programs—for which the automaton is of size polynomial in the formula length, and, hence, is decidable in polynomial time.

The construction of the transition diagram of the automaton is effected through propositional reasoning. The syntactic restrictions on the logic ensure that the size of the automaton is polynomial in the formula's size. Testing non-emptiness of the automaton is done by graph reachability analysis, and corresponds to the actual temporal portion of the reasoning in the sense that it relates the truth of the assertions at the present state to their truth at states that will be reached in the future. This is significant because it means much of temporal reasoning simply amounts to graph reachability analysis and can be done very efficiently, in linear or quadratic time.

The rest of this paper is organized as follows. In Section 2, we describe SCTL (Simplified Computation Tree Logic), one of the systems of logic that we have developed that permit efficient reasoning. Section 3 details the decision procedure that tests satisfiability of an SCTL formula, while Section 4 shows how to decide the validity of inference problem for SCTL. Finally, as SCTL is designed to handle assertions of a concurrent program as a whole, Section 5 describes how to convert the compositional input specifications (in terms of processes) to SCTL assertions, and indicates how the decision procedures developed in the previous sections could be applied to practical examples.

2 Preliminaries: The Logic SCTL

Numerous systems of temporal logic of differing expressive power have been introduced in the literature. One that is commonly used for reasoning about concurrent programs is CTL (Computation Tree Logic) (cf. [EC82], [EH82]) as it can express many interesting properties of programs. The basic CTL modalities are of the form A ("for all paths") or E ("for some path") followed by a single occurrence of F (sometime), G (always), X (next time), or U (until). These eight basic modalities can be combined using boolean connectives and nesting. The reader is referred to the appendix for the details of CTL's syntax and semantics. Examples of CTL formulae include: $AG(\neg CS_1 \vee \neg CS_2)$ which expresses the property of mutual exclusion of two processes (at any time, either process 1 or process 2 is not in its critical region), and $AG(TRY_1 \Rightarrow AF\ CS_1)$ which states that process 1 does not starve (whenever it is in its trying region, it eventually enters its critical region).

In this paper, we focus on the automation of temporal reasoning using a decision procedure for the following problems:

- testing satisfiability: Given a formula p, does there exist a structure M and a state s of M such that $M, s \models$

p (i.e., so that M is a *model* of p)? This has applications to mechanical program synthesis (cf. [EC82], [MW84]).

• testing validity: given a formula p, is it the case that in all structures M and all states s of M, $M, s \models p$ (so p is "universally valid"). This has applications to mechanical program verification as explained below. Note that p is valid iff $\neg p$ is not satisfiable. So, if the logic is closed under negation, validity is reducible to satisfiability.

A particularly important special case of the problem of testing validity is: *Given assertions* $p_1, \ldots, p_k$ *and* q, *is the inference* $(p_1 \wedge \ldots \wedge p_k) \Rightarrow q$ *valid?* This problem is useful in providing mechanical support for program verification, which might otherwise be performed by hand using a temporal logic deductive system. By inspection, one would first determine that the program meets, say, specifications p_1, p_2, and p_3. Then one would prove, using the deductive system, that p_4 holds. In general, one would prove from the antecedent assertions $p_1, \ldots, p_k$ already established that the assertion p_{k+1} follows. This inference step amounts to showing that $(p_1 \wedge \ldots \wedge p_k) \Rightarrow p_{k+1}$ is valid. Thus each step could be automated using a decision procedure for validity, but the complexity for each step would, for most extant systems of logic, be exponential in the size of the assertions. However, if the assertions are in an appropriately restricted syntax (such as the one detailed below), we can show that the validity of each inference step can be done in small polynomial time. We may note, parenthetically, that: (a) our polynomial time algorithms for the validity of inference problem may be viewed as a mechanization (and a considerable generalization) of the informal proof lattice methodology in [OL82] and [MP82]; note that the proof lattices resemble the transition diagrams of tree automata (cf. [Em85]), and (b) this method can be applied even to infinite state programs (in some cases, by using an appropriate first order language to define monadic predicates corresponding to atomic propositions).

For problems such as mechanical synthesis and verification, the deterministic exponential time complete decision procedure for determining the satisfiability of a CTL formula is a serious handicap. One tractable system of logic that suffices to express such problems is SCTL (*S*implified CTL). But as SCTL is a variant of another logic, NESCTL (*N*on *E*uclidean SCTL), we first state the syntax and semantics of NESCTL formulae.

Let Σ be a finite alphabet of atomic propositions, P, Q, ..., which are assumed to be exhaustive and mutually exclusive. The syntax is restricted to allow only conjunctions of the following five types of assertions (α, β_i, γ, δ and θ are subsets of Σ; $\vee\alpha$ denotes the disjunction of the propositions in α):

1. $\vee\theta$, the *initial* assertions,
2. $AG(\vee\delta)$, the *invariance* assertions,
3. $AG(P \Rightarrow AX(\vee\alpha) \wedge (\bigwedge_i EX \vee\beta_i))$, where i ranges over a finite, possibly empty, index set, the *successor* assertions; $AX(\vee\alpha)$ is called the *universal* conjunct and $EX(\vee\beta_i)$, an *existential* conjunct,
4. $AG(P \Rightarrow AF(\vee\gamma))$, the *leads-to* assertions; the sub-formula $AF(\vee\gamma)$ is called an *eventuality*, and
5. $AG(P \Rightarrow A((\vee\theta)\ U\ (\vee\gamma)))$, the *ensures* assertions; the sub-formula $A((\vee\theta)\ U\ (\vee\gamma))$ is called an *assurance*.

Note that the two CTL formulae cited above as examples conform to this restricted syntax. For simplicity of exposition, we shall refer to the antecedent and the consequent of implications of assertions as the antecedent and the consequent respectively of the assertion itself.

As NESCTL formulae are also CTL formulae, their semantics is defined as it is for CTL formulae, with one proviso: NESCTL formulae are interpreted over only those structures whose labellings assign exactly one proposition to each state of the structure. Hence, the atomic propositions of Σ are termed *mutually exclusive* and *exhaustive*. As usual, a structure M is said to be a *model* of an NESCTL formula f iff f is true at some state of M. It is apparent that if f is true at state s of a model, the non-initial conjuncts of f are true at all states of the model reachable from s.

In the sequel, we adopt two simplifying notational conventions. First, as Σ is exhaustive, the leads-to assertion $AG(P \Rightarrow AF(\vee\gamma))$ is equivalent to the ensures assertion $AG(P \Rightarrow A((\vee\Sigma)\ U\ (\vee\gamma)))$; hence, we shall assume that formulae have only ensures assertions. Leads-to assertions can be handled likewise. Secondly, since any NESCTL formula is a conjunction of the above five kinds of assertions, we associate with each NESCTL formula f, a set that has f's conjuncts as its elements. So any set of assertions of the above five kinds, say, $\mathcal{F}$, corresponds to the NESCTL formula obtained by conjoining its elements; we denote this formula by $\wedge\mathcal{F}$.

The reason for choosing the atomic propositions in Σ to be mutually exclusive and exhaustive is to help factor out the propositional aspects of the reasoning from the temporal aspects. This is essential because with n "ordinary, overlapping" propositions $P_1, P_2, \ldots, P_n$, we have 2^n different subsets of propositions which could be true at a state, all of which must be considered by the decision procedure. Despite this simplification, and NESCTL's restricted syntax, a decision procedure for testing satisfiability of an NESCTL formula would still be exponential; the intuition for this (explained more fully in the next section) is that, while it is no longer required to consider sets of atomic propositions that could be true at a state of a structure, it may be necessary to consider different sets of assurances that must be fulfilled at different states of the structure labelled with the same atomic proposition.

So, we define the logic SCTL. Like all NESCTL formulae, SCTL formulae are also conjunctions of the above five kinds of assertions, but with an additional restriction which we shall state in terms of the sets of assertions that correspond to SCTL formulae. A formula f is an SCTL formula iff the associated set $\mathcal{F}$ of f's conjuncts is a set of SCTL assertions. A set $\mathcal{F}$ of assertions of the types 1–5 above is a *set of SCTL assertions* iff (a) for every proposition P in Σ, $\mathcal{F}$ has at least one successor assertion whose antecedent is P, and (b) $\mathcal{F}$ satisfies the *euclidean syntactic constraint:* for every ensures assertion $AG(P \Rightarrow A((\vee\theta)\ U\ (\vee\gamma)))$ in $\mathcal{F}$, either P is in γ, or for every proposition $Q \in (\theta \setminus \gamma)$ that is also in the sets α of the universal conjuncts of all successor assertions of which P is the antecedent, the ensures assertion $AG(Q \Rightarrow A((\vee\theta)\ U\ (\vee\gamma)))$ is in $\mathcal{F}$.

The first restriction, that of requiring every atomic proposition P to have a successor assertion, is not critical as the assertion $AG(P \Rightarrow AX(\vee\Sigma))$ is implicit in any set of assertions (because Σ is exhaustive). The euclidean syntactic constraint, however, is crucial: *it allows the pending assurances to be recovered from an atomic proposition alone.* This restriction together with SCTL's restricted syntax—the mutually exclusive and exhaustive set of propositions from which formulae are built, the limited nesting of modalities within each assertion plus the fact that all non-initial assertions hold globally everywhere—make a polynomial time decision procedure for testing satisfiability possible as demonstrated in the next section.

We conclude this section with a definition and a technical lemma (analogous versions of which can be stated for all NESCTL formulae). A set $\mathcal{G}$ of SCTL assertions is said to be in *canonical form* iff:

(i) it has precisely one initial and one invariance assertion, and

(ii) every proposition Q in Σ is the antecedent of precisely one successor assertion. This successor assertion must have at least one existential conjunct and each set β_i of each existential conjunct should be a subset of the set α that appears in the universal conjunct.

Lemma 1 *For each set $\mathcal{F}$ of SCTL assertions, there exists a set $\mathcal{G}$ of SCTL assertions in canonical form such that $\wedge\mathcal{G}$ is equivalent to $\wedge\mathcal{F}$, i.e., for every structure M and state s of M, $M, s \models \wedge\mathcal{G}$ iff $M, s \models \wedge\mathcal{F}$. Moreover, $\mathcal{G}$ can be computed from $\mathcal{F}$ in time linear in the length of $\wedge\mathcal{F}$.*

Proof: Given $\mathcal{F}$, the set $\mathcal{G}$ can be computed as follows. The invariance assertions $AG(\vee\delta_1)$ and $AG(\vee\delta_2)$ are replaced by $AG \vee (\delta_1 \cap \delta_2)$ and the assertion $AG(\vee\Sigma)$ is added if there are no invariance assertions—this is justified because Σ is exhaustive. Similarly, the initial assertions $\vee\theta_1$ and $\vee\theta_2$ are replaced by $\vee(\theta_1 \cap \theta_2)$ and the assertion $\vee\Sigma$ is added if there are no initial assertions in $\mathcal{F}$.

Note that, for each proposition Q in Σ, $\mathcal{F}$ has one successor assertion with Q as its antecedent (by the definition of SCTL formulae). To obtain precisely one such successor assertion in the set $\mathcal{G}$, the successor assertions $AG(Q \Rightarrow AX(\vee\alpha) \wedge (\bigwedge_i EX \vee \beta_i))$ and $AG(Q \Rightarrow AX(\vee\alpha') \wedge (\bigwedge_j EX\vee\beta'_j))$ in $\mathcal{F}$ are replaced by $AG(Q \Rightarrow AX \vee (\alpha \cap \alpha') \wedge (\bigwedge_i EX\vee\beta_i) \wedge (\bigwedge_j EX\vee\beta'_j))$. To ensure that each successor assertion has at least one existential conjunct, an assertion of the form $AG(Q \Rightarrow AX(\vee\alpha))$ is replaced by $AG(Q \Rightarrow AX(\vee\alpha) \wedge EX(\vee\alpha))$; this is justified because every state of a structure must have a successor state. Finally, when there is only one successor assertion with Q as the antecedent, each set β_i (of the ith existential conjunct in the assertion) is replaced by $\alpha \cap \beta_i$ where α appears in the universal conjunct of the assertion.

The resulting transformed set is $\mathcal{G}$; note that the leads-to and ensures assertions of $\mathcal{F}$ are in $\mathcal{G}$ without modification and that $\mathcal{G}$ satisfies the euclidean syntactic constraint if $\mathcal{F}$ does. It is easily seen that $\wedge\mathcal{G}$ and $\wedge\mathcal{F}$ are equivalent and that $\mathcal{G}$ can be obtained from $\mathcal{F}$ in time linear in the length of $\wedge\mathcal{F}$. □

3 Testing Satisfiability in SCTL

This section outlines an algorithm to decide the satisfiability problem for SCTL, which is: *Given a set Σ of mutually exclusive and exhaustive atomic propositions and a set $\mathcal{F}$ of SCTL assertions over Σ, does there exist a structure M and a state s in M such that $M, s \models \wedge\mathcal{F}$, i.e., such that M is a model of $\wedge\mathcal{F}$?*

Without loss of generality, we assume that the set $\mathcal{F}$ is in canonical form. The first step in checking the satisfiability of $\wedge\mathcal{F}$ is constructing a tableau called the *next time tableau* out of the successor assertions in $\mathcal{F}$. The initial next time tableau, T_0, is a bipartite directed graph whose vertex set is partitioned into the sets of AND and OR nodes. T_0 has one AND node for each atomic proposition. We identify each AND node with its proposition (so we might say "a state of a structure is labelled with an AND node" to mean that it is labelled with the corresponding atomic proposition). Consider the successor assertion $AG(Q \Rightarrow AX(\vee\alpha) \wedge (\bigwedge_i EX\vee\beta_i))$ in $\mathcal{F}$. Q has one OR node successor for each conjunct $EX(\vee\beta_i)$. The OR node corresponding to the ith existential conjunct has as its successors the AND nodes which are elements of β_i. Any AND node that occurs in the initial assertion of $\wedge\mathcal{F}$ is termed an *initial node* of the tableau. The AND nodes are labelled with the assurances of which they are the antecedents in the ensures assertions of $\mathcal{F}$.

T_0 accounts for the constraints imposed by successor assertions in $\mathcal{F}$ on potential models of $\wedge\mathcal{F}$. Of course, only its AND nodes may label states of structures, and so, the OR nodes in it may appear superfluous. The intuitive meaning of an OR node is as follows. If the non-initial assertions of $\mathcal{F}$ are true at a state s labelled with P in a structure M (as they must be if $M, t \models \wedge\mathcal{F}$ and s is reachable from t), then s must have one successor r_u for each OR node successor u of P in T_0 and r_u must be labelled with some AND node successor of u in T_0. Note that though s could have successors labelled with other atomic propositions (e.g., those that appear in the set α of the universal conjunct of the successor assertion with antecedent P), it is not required to have such successors, and, as the decision procedure has to determine only the satisfiability of $\wedge\mathcal{F}$, it can exclude such models from its candidate models of $\wedge\mathcal{F}$. For this reason, there are no OR nodes in T_0 that represent the universal conjuncts of successor assertions.

An important property of the tableau T_0, which follows from the fact that $\mathcal{F}$ satisfies the euclidean syntactic constraint, is captured by the following lemma:

Lemma 2 *If there is a path p from the AND node Q to the AND node R in the tableau T_0, the assurance $A((\vee\theta)\ U\ (\vee\gamma))$ is in the label of Q, and all AND nodes in p are in $(\theta \setminus \gamma)$, then $A((\vee\theta)\ U\ (\vee\gamma))$ is also in the label of R.*

Example 1 Let $\Sigma = \{P, Q, R, S, T, V, W\}$ and $\mathcal{F}$ have the assertions shown in Fig. 1. For clarity, leads-to assertions are shown as such rather than as ensures assertions. Note that $\mathcal{F}$ fulfills the euclidean syntactic constraint and is in canonical form. The tableau constructed from $\mathcal{F}$ is shown in Fig. 2. AND nodes are shown as boxes and OR nodes as circles. As the assurances labelling the AND nodes can be inferred from the set of assertions, they are not shown in the figure. □

The next step is to prune T_0 to delete inconsistent nodes. The procedure repeatedly attempts to delete nodes and edges in the current tableau that cannot occur in any model of $\wedge\mathcal{F}$. There are four deletion rules in all, the first two of which are applied only once, at the start, to the initial next time tableau. First, AND nodes which do not occur in the invariance assertion of $\mathcal{F}$ are deleted: this assertion cannot be true at any state of any structure labelled with such an AND node. Secondly, if the AND node P has the assurance $A((\vee\theta)\ U\ (\vee\gamma))$ in its label and P is not in $\theta\cup\gamma$, P is deleted. Also, if P has $A((\vee\theta)\ U\ (\vee\gamma))$ in its label, $P \in (\theta\setminus\gamma)$, and u is an OR node successor of P, the edge from u to any AND node Q that is not an element of $\theta\cup\gamma$ is deleted (for if Q were to label a state succeeding one labelled P in some structure, the ensures assertion $AG(P \Rightarrow A((\vee\theta)\ U\ (\vee\gamma)))$ would be false at the latter state). Note that this is the only deletion rule that removes edges of the tableau, and it only removes edges from OR to AND nodes.

Now, the following two deletion rules are repeatedly applied. First, any AND node, some one of whose successors in T_0 has been deleted, or any OR node with no successors, is deleted. Secondly, for every assurance $A((\vee\theta)\ U\ (\vee\gamma))$ in the label of an AND node Q, the tableau must contain a DAG (directed acyclic graph) rooted at Q, fulfilling this assurance. The DAG may have at most one copy of any node in the tableau and Q must be the only node without a predecessor in the DAG (i.e., its only root). Any interior AND node in the DAG must be in $(\theta\setminus\gamma)$ and have as its successors all its successors in the tableau, and every frontier node of the DAG must be an AND node that's an element of γ. If, for some assurance in its label, an AND node does not have a DAG rooted at it which fulfills that assurance, it is deleted. The pruning procedure terminates either by deleting all the initial nodes, in which case $\wedge\mathcal{F}$ is unsatisfiable, or leaves a tableau from which no more nodes or edges can be deleted by the above rules; such a tableau can be unwound to a model of $\wedge\mathcal{F}$.

Example 2 The pruning procedure applied to the tableau of Fig. 2 deletes Q, as it does not occur in the invariance assertion, and S, as the tableau does not have a DAG fulfilling $AF\ R$ rooted at S (as P must be a successor of the OR node between S and P, and S, a successor of the OR node between P and S). The deletion of S causes the deletion of its only OR node predecessor, and, hence, the deletion of P. No other nodes can be deleted: there are DAGs rooted at each of R, T, and V fulfilling $AF(Q \vee V)$ and $AF\ R$ and one rooted at W fulfilling $AF\ R$. Since the pruned tableau (shown in Fig. 3) contains an initial node (W), $\wedge\mathcal{F}$ is satisfiable. In fact, a model for $\wedge\mathcal{F}$ (Fig. 4) is contained in pruned tableau as each OR node in it has a single successor.

If the assertions in $\{AG(v \Rightarrow AF\ P) \mid v \in \{R, T, V\}\}$ were added to $\mathcal{F}$, the new set of assertions would not be satisfiable as T would be deleted (P is not reachable from T, so there is no DAG fulfilling $AF\ P$ rooted at T), causing the initial node W to be deleted. Similarly, if the initial assertion in $\mathcal{F}$ were changed to $P \vee Q \vee S$, the new set of assertions is unsatisfiable as all of P, Q and S are deleted. □

The following proposition is proved in the full paper:

Proposition 1 *The above pruning procedure decides the satisfiability of its input, $\wedge\mathcal{F}$, correctly. Specifically,* *if it deletes an AND node P from the initial next time tableau constructed from $\mathcal{F}$, then for any structure M and state s of M labelled with P, some non-initial assertion of $\mathcal{F}$ is false at s. Conversely, if P is an AND node in the pruned tableau, there is a structure M (which, if the pruned tableau has an initial node, is also a model of $\wedge\mathcal{F}$) that has a state s labelled with P at which all non-initial assertions of $\mathcal{F}$ are true. Thus $\wedge\mathcal{F}$ has a model iff the pruned tableau contains an initial node. Moreover, the pruning procedure can be implemented to run in time quadratic in the length of $\wedge\mathcal{F}$.*

Theorem 1 *SCTL satisfiability is in deterministic time $O(n^2)$. It is also $\mathcal{P}$-hard.*

Proof: Converting $\mathcal{F}$ to canonical form and building a tableau for it can be done in time linear in the length of $\wedge\mathcal{F}$. An implementation of the pruning procedure that runs in time quadratic in the length of the canonical form of $\wedge\mathcal{F}$ will be described in the full paper in the proof of Proposition 1. The proof of $\mathcal{P}$-hardness will be presented in the full paper. □

We conclude this section with a summary of the factors involved in getting a polynomial sized tableau for SCTL (and, hence, a polynomial time decision procedure for its satisfiability problem). First, the syntax of the SCTL's initial, invariance, and successor assertions over the (mutually exclusive and exhaustive) alphabet Σ ensures that the tableau built for an SCTL formula, called the next time tableau, is of size linear in the length of the formula. Here we have successfully factored out the propositional reasoning from the temporal reasoning. We must now account for the assurances. Viewed automata-theoretically, the ordinary tableau is the product of the local automaton (the next time tableau) with the global automaton, which is itself the product of the two-state automata for each assurance. The two-state automaton for the assurance $A(P\ U\ Q)$, say, has one state corresponding to $A(P\ U\ Q)$ being immediately fulfilled and one state for $A(P\ U\ Q)$ being pending. Therefore the global automaton is exponential in the number of assurances, and the ordinary CTL tableau is of exponential size. The exponential blow-up is due to the fact that each node of the next time tableau must be refined into possibly exponentially many nodes in the global tableau, corresponding to the different sets of assurances that are required to be fulfilled there. Different sets of assurances are required because the set of assurances depends on the path followed to get to the node in the next time tableau. But by the euclidean syntactic constraint on SCTL, there is a unique set of assurances that needs to be fulfilled at each node of the next time tableau: this set is determined by the (unique) proposition labelling the node. Hence, the next time tableau suffices for SCTL's decision procedure for satisfiability.

4 Testing Validity of Inference in SCTL

The validity of inference problem for SCTL is as follows. Let $\mathcal{A}$ be a set of SCTL assertions composed from the set Σ of atomic propositions. Let $\mathcal{C}$ be a set of ensures assertions built from Σ such that the set $\mathcal{A}\cup\mathcal{C}$ is also a set of SCTL assertions (specifically, it also satisfies the euclidean syntactic constraint). Then, it is required to determine whether $(\wedge\mathcal{A} \Rightarrow \wedge\mathcal{C})$ is valid, i.e., whether $\wedge\mathcal{C}$ (and, hence, $\wedge(\mathcal{A}\cup\mathcal{C})$) is true at every state of every structure at which $\wedge\mathcal{A}$ is true.

An algorithm to decide the validity of inference problem is outlined in this section. We assume that the ensures assertions in $\mathcal{C}$ are not in $\mathcal{A}$, and that for each ensures assertion $AG(P \Rightarrow A((\vee\theta)\ U\ (\vee\gamma)))$ in $\mathcal{C}$, P is not in γ (else, such assertions can be dropped from $\mathcal{C}$ as the inference of such assertions is trivially valid). We also assume, without loss of generality, that $\mathcal{A}$ is in canonical form. First, the next time tableau for $\wedge\mathcal{A}$ is built and pruned—let this pruned tableau be $T_{\mathcal{A}}$ and let the subgraph of the pruned tableau induced by nodes reachable from some initial node in it be $T'_{\mathcal{A}}$. The next time tableau for $\wedge(\mathcal{A}\cup\mathcal{C})$ is also constructed and pruned; let this pruned tableau be $T_{\mathcal{A}\cup\mathcal{C}}$ and let the subgraph of the pruned tableau induced by nodes reachable from some initial node in it be $T'_{\mathcal{A}\cup\mathcal{C}}$. The following lemma, a consequence of Proposition 1, is proved in the full paper.

Lemma 3 *$T'_{\mathcal{A}\cup\mathcal{C}}$ is a subgraph of $T'_{\mathcal{A}}$. If it is a proper subgraph, $\wedge\mathcal{A} \Rightarrow \wedge\mathcal{C}$ is an invalid inference.*

In the remainder of this section, we assume that $T'_{\mathcal{A}\cup\mathcal{C}}$ is not a proper subgraph of $T'_{\mathcal{A}}$. We extend $T'_{\mathcal{A}}$ to a tableau $T''_{\mathcal{A}}$, which contains $T'_{\mathcal{A}}$ as a subgraph, and, additionally, has one more OR node successor for each AND node. The additional OR node successor, call it u, of the AND node P represents the universal conjunct of the successor assertion of $\mathcal{A}$ that has P as its antecedent. Its successors are the AND nodes in $T''_{\mathcal{A}}$ that are in the set α of the universal conjunct of the successor assertion and are also in $\nu\cup\eta$ for those assurances $A((\vee\nu)\ U\ (\vee\eta))$ of $\mathcal{A}$ in the label of P for which $P \notin \eta$.

Let $\mathcal{E}$ be the set of assurances contained in $\mathcal{C}$, i.e., E is in $\mathcal{E}$ iff the ensures assertion $AG(P \Rightarrow E)$ is in $\mathcal{C}$ for some proposition P. For each $E \in \mathcal{E}$, let $\mathcal{C}_E$ be the set of ensures assertions in $\mathcal{C}$ whose consequents are E. It is easily seen that (i) for each $E \in \mathcal{E}$, $\mathcal{A}\cup\mathcal{C}_E$ satisfies the euclidean syntactic constraint (because $\mathcal{A}\cup\mathcal{C}$ does), and (ii) $\wedge\mathcal{A} \Rightarrow \wedge\mathcal{C}$ is valid iff, for each $E \in \mathcal{E}$, $\wedge\mathcal{A} \Rightarrow \wedge\mathcal{C}_E$ is valid. We now show how to test the validity of $\wedge\mathcal{A} \Rightarrow \wedge\mathcal{C}_E$ for an arbitrary assurance $E = A((\vee\theta)\ U\ (\vee\gamma))$ in $\mathcal{C}$ in time linear in the size of $T''_{\mathcal{A}}$.

First, some notation and definitions. Let B be any assurance in $\mathcal{A}\cup\mathcal{C}_E$. Let $H^B_{\mathcal{A}}$ ($H^B_{\mathcal{C}_E}$ respectively) be the set of AND nodes P in $T''_{\mathcal{A}}$ such that the ensures assertion $AG(P \Rightarrow B)$ is in $\mathcal{A}$ ($\mathcal{C}_E$ respectively). Note that $H^B_{\mathcal{A}}$ is merely the set of vertices in $T''_{\mathcal{A}}$ that the assurance B labels. Also, note that $H^B_{\mathcal{C}_E} = \emptyset$ unless B is the assurance E, and that $H^E_{\mathcal{A}}$ and $H^E_{\mathcal{C}_E}$ are disjoint because ensures assertions in $\mathcal{C}$ that were also in $\mathcal{A}$ were deleted from $\mathcal{C}$. Consider any infinite path p in the extended tableau $T''_{\mathcal{A}}$. We say that p *fulfills all assurances of $\mathcal{A}$ that it makes* iff, for each assurance $B = A((\vee\nu)\ U\ (\vee\eta))$ in $\mathcal{A}$, each time p goes through a vertex, say P, in $H^B_{\mathcal{A}}$, it subsequently goes through some AND node $Q \in \eta$, and every AND node between P and Q on p is in $(\nu \setminus \eta)$. Also, the assurance $E = A((\vee\theta)\ U\ (\vee\gamma))$ in $\mathcal{C}_E$ is said to be *made along p, but remains unfulfilled forever along p* iff p goes through a vertex, say P, in $H^E_{\mathcal{C}_E}$, and all subsequent AND nodes that it goes through are in $(\theta \setminus \gamma)$.

In the full paper, we prove the following proposition that relates the extended tableau $T''_{\mathcal{A}}$ to possible models of $\wedge\mathcal{A}$:

Proposition 2 *If there is an infinite path p starting from some initial node in the extended tableau $T''_{\mathcal{A}}$ that fulfills all assurances of $\mathcal{A}$ that it makes, there is a model M of $\wedge\mathcal{A}$ that contains a path r that starts at a state of M at which $\wedge\mathcal{A}$ is true and such that the sequence of propositions labelling its states is the same as the sequence of AND nodes in p. Conversely, let M be any structure, and, s, any of its states at which $\wedge\mathcal{A}$ is true. For any infinite path r in M starting at s, there is a path p in $T''_{\mathcal{A}}$ such that the sequence of AND nodes along p is the same as that of the propositions labelling the states of r, and all assurances of $\mathcal{A}$ made along p are fulfilled by p.*

Consider infinite paths p in $T''_{\mathcal{A}}$ with the following property:

> p starts at an initial node of $T''_{\mathcal{A}}$ and fulfills all assurances in $\mathcal{A}$ that it makes; however, the assurance E in $\mathcal{C}_E$, $A((\vee\theta)\ U\ (\vee\gamma))$, is made along p and remains unfulfilled forever along p.

We shall call such paths "bad" paths. From Proposition 2, there is a correspondence between infinite paths that start at an initial node of $T''_{\mathcal{A}}$ and fulfill all assurances of $\mathcal{A}$ that they make, and infinite paths in a structure that start at a state at which $\wedge\mathcal{A}$ is true; hence, we have:

Lemma 4 *$\wedge\mathcal{A} \Rightarrow \wedge\mathcal{C}_E$ is valid iff $T''_{\mathcal{A}}$ does not contain a bad path.*

Consider the subgraph of $T''_{\mathcal{A}}$, call it T, that is induced by AND nodes in $H^E_{\mathcal{C}_E}$ and those OR nodes whose predecessors is one of these AND nodes. Note that none of the AND nodes in T is in $\gamma \cup H^E_{\mathcal{A}}$ (because assertions in $\mathcal{C}$ that could have caused this were dropped from $\mathcal{C}$). Using Proposition 2 and the fact that $\mathcal{A} \cup \mathcal{C}_E$ satisfies the euclidean syntactic constraint, we prove in the full paper:

Lemma 5 *$T''_{\mathcal{A}}$ has a bad path iff T has a strongly connected component that has more than one vertex and such that, for all assurances $A((\vee\nu)\ U\ (\vee\eta))$ of $\mathcal{A}$ that occur in the labels of vertices of the component, an element of η is a vertex in the component.*

Hence, the problem reduces to determining if there exists a strongly connected component of T with more than one vertex that is "self-fulfilling", i.e., for each assurance $A((\vee\nu)\ U\ (\vee\eta))$ of $\mathcal{A}$ labelling some one of its vertices, some, and, hence, all, vertices in it satisfy $EF(\vee\eta)$; the inference $\wedge\mathcal{A} \Rightarrow \wedge\mathcal{C}_E$ is invalid if there is, and, otherwise, it is valid. So we have:

Theorem 2 *The validity of inference problem for SCTL is solvable in deterministic time $O(n^2)$. It is also $\mathcal{P}$-hard.*

Proof: Clearly, $T_{\mathcal{A}}$ and $T_{\mathcal{A}\cup\mathcal{C}}$ can be constructed and pruned in time quadratic in the lengths of $\wedge\mathcal{A}$ and $\wedge(\mathcal{A} \cup \mathcal{C})$ respectively. $T'_{\mathcal{A}}$ and $T'_{\mathcal{A}\cup\mathcal{C}}$ can be constructed from $T_{\mathcal{A}}$ and $T_{\mathcal{A}\cup\mathcal{C}}$ in time linear in the sizes of $T_{\mathcal{A}}$ and $T_{\mathcal{A}\cup\mathcal{C}}$ respectively and can be checked for equality in time linear in their sizes. $T''_{\mathcal{A}}$ can be constructed from $T'_{\mathcal{A}}$ in time linear in the length of $\wedge\mathcal{A}$. For each assurance in $\mathcal{C}$, the subgraph T of $T''_{\mathcal{A}}$ described above can be extracted from $T''_{\mathcal{A}}$ in time linear in the size of $T''_{\mathcal{A}}$. Finally, the strongly connected components of T can be computed in time linear in the size of T, and all of them which have more than one vertex can be checked for the self-fulfilling property in time proportional to the size of T. Since there are no more assurances in $\mathcal{C}$ than the length of $\wedge\mathcal{C}$, it follows that the complexity of the decision procedure is $O(n^2)$. The proof of $\mathcal{P}$-hardness will be presented in the full paper. □

Example 3 If the set $\mathcal{A}$ of antecedents contains the assertions in Fig. 1, and the set $\mathcal{C}$ of consequences is $\{AG(u \Rightarrow AF\ T) \mid u \in \{R, V\}\}$, $\wedge(\mathcal{A}\cup\mathcal{C})$ is unsatisfiable as all nodes of the tableau of Fig. 3 (which is the pruned tableau of $\wedge\mathcal{A}$) are deleted (because there is no DAG satisfying $AF\ T$ rooted at V or R). So the inference $\wedge\mathcal{A} \Rightarrow \wedge\mathcal{C}$ is not valid. On the other hand, if the set $\mathcal{C}$ of consequences is $\{AG(u \Rightarrow AF\ V) \mid u \in \{R, T, W\}\}$, the pruned tableau (and the subgraph of states reachable from the initial state) of $\wedge(\mathcal{A} \cup \mathcal{C})$ is the same as that of $\wedge\mathcal{A}$ (Fig. 3) and, by inspection, does not contain an infinite path along which the eventuality $AF\ V$ is pending forever because every infinite path in it goes through V infinitely often. Hence the inference is valid. □

5 Handling Compositional Assertions

SCTL, as developed, allows the expression of assertions that a concurrent system must satisfy. In practice, assertions are expressed compositionally, i.e., in terms of the processes of the concurrent system. We outline the kinds of assertions processes are permitted and show how to convert them to SCTL assertions for the concurrent system, using interleaving to model parallelism. In particular, we show that if processes satisfy a condition we term *history-freedom*, the SCTL assertions automatically satisfy the euclidean syntactic constraint. In the full paper, we shall demonstrate how the satisfiability and validity of inference procedures for SCTL can be applied to two practical problems. (*Remark:* We do not express fairness conditions here, but our techniques can be extended to do so using those developed in [CVW86] and [EL86].)

5.1 Process Assertions

Each process is specified by a set of temporal assertions that express the invariants the process satisfies and the state transitions of the process' flow graph. The interpretation of states and transitions in the flow graph may vary depending on the application. For the synthesis problem, for example, the flow graph represents the process' *synchronization skeleton*, each state of which models a terminating computation, and the transitions of which represent permitted sequences of terminating computations. In this case, the process assertions serve as *specifications*. For the validity of inference problem, the states may take on their traditional meaning of a process' state and the assertions express known facts about the program being verified.

The assertions of process i are composed from a set Σ_i of atomic propositions, which has one proposition for each state in process i's flow graph; intuitively, this proposition holds when process i is in that state. Hence, the propositions of Σ_i may justifiably be assumed to be exhaustive and mutually exclusive. The assertions of process i must have the restricted syntax of one of the following types:

A. Q, where $Q \in \Sigma_i$. This specifies process i's initial state. There must be exactly one assertion of this kind for each process. We assume that all states in the process' flow graph are reachable from its initial state.

B. $AG(Q \Rightarrow AF(\vee\gamma))$, where $Q \in \Sigma_i$ and $\gamma \subseteq \Sigma_i$. This asserts that whenever process i visits state Q, it incurs an obligation to visit some state of γ in the future.

C. $AG(Q \Rightarrow A((\vee\theta)\ U\ (\vee\gamma)))$, where $Q \in \theta$, $\theta \subseteq \Sigma_i$ and $\gamma \subseteq \bigcup_{j \neq i} \Sigma_j$. Every other element of θ should also have an ensures assertion with the same assurance as the consequent and with it as the antecedent. This collection of assertions forces process i to remain in the states of θ until one of the other processes has moved to a desired state (the states in γ); hence, synchronizations among processes can be expressed.

D. $AG(Q \Rightarrow AX_i(\vee\alpha) \wedge (\bigwedge EX_i\beta))$, where $Q \in \Sigma_i$, $\beta \subseteq \alpha \subseteq \Sigma_i$, and $EX_i\beta \overset{\text{def}}{=} \{EX_i\ b \mid b \in \beta\}$. Exactly one assertion of this kind is required for each antecedent $Q \in \Sigma_i$. $AX_i f$ means "f holds at the next instant of time along all those paths starting with a transition of process i" and $EX_i f$ means "there is an immediate successor at which f holds reachable by one step of process i". This assertion specifies the successors of state Q: process i picks a successor from α, and, each time it is in state Q, it can move to any state of β irrespective of the states of other processes. Note that the set β and, hence, $EX_i\beta$, could be empty.

E. $AG(Q \Rightarrow \bigvee_{j \neq i}(EX_j Q))$, where $Q \in \Sigma_i$. This asserts that each time process i is in state Q, there must be a successor that leaves it in state Q. (This does *not* necessarily imply that process i may remain in state Q forever.)

Additionally, we require that every process be *history-free*. A process is history-free iff for every state s of the process and every path p from its initial state to s, the set of eventualities incurred along p but which remain unfulfilled at s is the same, irrespective of the path p. The concept of history-freedom for non-terminating processes is merely an extension of the concept of a state of classical terminating processes. The current state of a terminating process completely determines its set of possible behaviours. For non-terminating processes, the behaviour is additionally restricted by the set of eventualities it is obliged to fulfill; by ensuring that this set is dependent solely on its current state, rather than on the path followed to it, we can ensure that the set of behaviours of a history-free process is completely determined by the state it is in, just as for terminating processes.

If a process is history-free, a canonical set of assertions, in which each unfulfilled eventuality at each state of the process appears in a leads-to assertion with the state as the antecedent and the eventuality as the consequent, can be computed as is shown in the proof (presented in the full paper) of the following theorem; we shall call such an augmented set of assertions a *complete* set. In the sequel, we shall assume that each process is specified completely in this manner.

Theorem 3 *Given a set of process assertions of the kinds A–E above, determining if it is history-free is in deterministic time $O(n^2)$.*

Example 4 Consider the classical two process mutual exclusion problem. The synchronization skeletons of the processes are shown in Fig. 5. The letters N, T and C are mnemonics for "non-critical region", "trying region", and "critical region" respectively. The subscripts indicate the process numbers. The specifications on process i are:

1. N_i, the initial state.
2. $AG(N_i \Rightarrow AX_i\ T_i \wedge EX_i\ T_i)$. Whenever process i moves out of its non-critical region, it moves into its trying region and it can always do so irrespective of the state of the other process.
3. $AG(T_i \Rightarrow AX_i\ C_i)$. Whenever process i moves out of its trying region, it transits to its critical region.
4. $AG(C_i \Rightarrow AX_i\ N_i \wedge EX_i\ N_i)$. Process i can always move into its non-critical region from its critical region irrespective of the state of the other process.
5. $AG(T_i \Rightarrow AF\ C_i)$. Process i will eventually get to its critical region if it visits its trying region.
6. $AG(N_i \Rightarrow EX_j\ N_i)$, where $j = 2$ if $i = 1$ and 1 otherwise. Whenever process i is in its non-critical region, there is a successor state in which process j has moved.

Note that both processes are history-free: the only unfulfilled eventuality is $AF\ C_i$ in state T_i and it is unfulfilled each time process i is in state T_i. Hence the above sets of specifications are complete. □

It is evident from the above example that if the flow graph of a process is tree-shaped (with the root being the initial state and the only cycles being caused by edges from the "leaves" to the root) and all eventualities incurred at each node are fulfilled along each path from that node to a leaf, the process is history-free. A surprisingly large class of processes that are "interesting" in practice satisfy this sufficiency condition.

5.2 System Assertions

In addition to process assertions, it is often necessary to ensure that the system as a whole satisfies certain safety properties. For example, in the mutual exclusion problem, both processes should not be allowed to enter their critical regions at the same time. So we allow assertions of the kind:

F. $AG(\neg(Q_1 \wedge Q_2 \wedge \ldots \wedge Q_m))$, where each $Q_i \in \bigcup_{j=1}^{k} \Sigma_j$ and at most one of the Q_i's is in any one Σ_j.

For example, $AG(\neg(C_1 \wedge C_2))$ expresses mutual exclusion of two processes.

5.3 Converting Compositional Assertions to Global Ones

Finally, we show how to convert the compositional assertions to assertions for the concurrent system. We do so using the mutual exclusion example here; in the full paper, we give an algorithm to do this. However, the main properties of the conversion are expressed in the following theorem which is also proved in the full paper:

Theorem 4 *For any fixed set of k history-free processes whose assertions are complete and of the types A–F above, the set of equivalent global assertions (using interleaving of process transitions to model parallelism) for the entire concurrent system is a set of SCTL assertions (specifically, it satisfies the euclidean syntactic constraint). Moreover, the global assertions can be computed in time polynomial in the length of the process assertions, with the degree of the polynomial being linear in k.*

Example 5 Consider again the two process mutual exclusion problem presented in the previous example. The set Σ of atomic propositions for the global system is $\{N_1N_2, N_1T_2, N_1C_2, T_1N_2, T_1T_2, T_1C_2, C_1N_2, C_1T_2, C_1C_2\}$. Intuitively, the proposition N_1T_2, say, is true when process 1 is in state N_1 *and* process 2 is in state T_2, and similarly for the other propositions. The global assertions are displayed in Fig. 6. The initial state assertion is got from the process initial state assertions, and the invariance assertion is obtained from Σ and the system assertions. The eventualities are obtained in the obvious fashion: if process 1 is in T_1, irrespective of the state of process 2, it should be possible for it to get to C_1 and process 2 can be in any state then. Finally, each successor assertion of a k-process concurrent system has $(k+2)$ components. Let process i be in state P_i when the concurrent system is in state R, and let the successor assertion of process i for state P_i be $AG(P_i \Rightarrow AX_i(\vee\alpha_i) \wedge (\bigwedge EX_i\beta_i))$. Consider the successor assertion of the concurrent system for state R. Its γ component is obtained by allowing one process to move at a time: the states to which process i may move are those in α_i excluding those states it is forbidden to move to because of assertions of type C of which P_i is the antecedent. If there is no assertion of type E with P_i as antecedent, θ_i is just γ. Otherwise, the θ_i component ensures process i can remain in state P_i, so it has each state in γ in which process i is in state P_i. The δ component accounts for the $\wedge(EX_i\beta_i)$ conjuncts in the successor assertions of processes: it ensures that process i can move to any state specified in β_i irrespective of the states of the other processes. Note that, as before, $EX\ \delta \overset{\text{def}}{=} \{EX\ d \mid d \in \delta\}$. □

5.4 Applications to Program Synthesis and Program Verification

In the full paper, we shall complete the example of the synthesis of the two process mutual exclusion problem presented above. (The remaining steps are similar to the procedure used in [EC82].) For now, it follows from the above that the fragment of CTL actually used in [EC82] to synthesize solutions to synchronization problems involving 2 processes is in time $O(n^4)$. In general, for any fixed number k of processes, it is in time $O(n^{2k})$. So it is feasible to use this method for small, fixed k. For a way to overcome the exponential blow-up in the number of processes incurred by the state explosion problem, see [AE89].

We shall also illustrate how the validity of inference problem can be used to verify concurrent programs, using the example program from [OL82] which implements mutual exclusion of two processes by allowing one process to have priority over the other. To do this, we use the linear time version of SCTL so that the algorithm may consider only the process-fair execution sequences, and show that all such execution sequences satisfy the two liveness properties proved for this example in [OL82].

Appendix: CTL Syntax and Semantics

Let Σ be an underlying alphabet of atomic propositions P, Q, etc. The set of CTL (Computation Tree Logic) formulae is generated by the following rules:

S1. Each atomic proposition P is a formula.

S2. If p, q are formulae then so are $p \wedge q$ and $\neg p$.

S3. If p, q are formulae then so are $A(p\ U\ q)$, $E(p\ U\ q)$, and EXp.

A formula of CTL is interpreted with respect to a structure $M = (S, R, L)$ where S is a set of states, R is a binary relation on S that is total (so each state has at least one successor), and L is a labelling which assigns to each state a set of atomic propositions, those intended to be true at the state. A *fullpath* $x = s_0, s_1, s_2, \ldots$ in M is an infinite sequence of states such that $(s_i, s_{i+1}) \in R$ for each i. We write $M, s \models p$ to mean that "formula p is true at state s in structure M". When M is understood, we write only $s \models p$. We define $\models$ by induction on formula structure:

S1. $s_0 \models P$ iff P is an element of $L(s_0)$.

S2. $s_0 \models p \wedge q$ iff $s_0 \models p$ and $s_0 \models q$.
$s_0 \models \neg p$ iff it is not the case that $s_0 \models p$.

S3. $s_0 \models A(p\ U\ q)$ iff for all fullpaths $s_0, s_1, s_2, \ldots$ in M, $\exists i \geq 0$ such that $s_i \models q$ and $\forall j, 0 \leq j < i, s_j \models p$.
$s_0 \models E(p\ U\ q)$ iff for some fullpath $s_0, s_1, s_2, \ldots$ in M, $\exists i \geq 0$ such that $s_i \models q$ and $\forall j, 0 \leq j < i, s_j \models p$.
$s_0 \models EXp$ iff there exists an R-successor t of s_0 such that $t \models p$.

The formulae p and q are said to be *equivalent* iff they are both true or both false at all states of all structures.

The other propositional connectives are defined as abbreviations in the usual way. The other basic modalities of CTL are also defined as abbreviations: AFq abbreviates $A(true\ U\ q)$, EFq abbreviates $E(true\ U\ q)$, AGq abbreviates $\neg EF\neg q$, EGq abbreviates $\neg AF\neg q$, and AXq abbreviates $\neg EX\neg q$.

Acknowledgement & Historical Remarks: This paper had been germinating for many years. When we saw the proof lattices in [OL82], and the closely related proof diagrams used by Manna & Pnueli (cf. [MP82], [MP83]), it was obvious to us that there was an efficient decision procedure implicit in their informal proof methods. These lattices resembled the tableau used in the CTL decision procedure of [Em81], [EC82], and [EH82]. In the mid 1980's, we gave a talk at MCC on the synthesis method where Mike Evangelist gave the most forceful arguments we had ever heard against the utility of exponential time decision procedures such as the one used for CTL. Yet, we knew that the synthesis method based on the CTL decision procedure worked efficiently in practice. This started us thinking.

References

[Ab80] Abrahamson, K., Decidability and Expressiveness of Logics of Processes, Ph.D. Thesis, Univ. of Washington, 1980.

[AE89] Attie, P.C., E.A. Emerson, Synthesis of Concurrent Systems With Many Similar Sequential Processes, *appears in this conference proceedings*, 1989.

[BHP81] Ben-Ari, M., J.Y. Halpern, A. Pnueli, Finite Models for Deterministic Propositional Dynamic Logic, *Proc. 8th Annual International Colloquium on Automata, Languages and Programming*, LNCS#115 Springer–Verlag, pp. 249–263, 1981.

[CVW86] Courcoubetis C., M.Y. Vardi, P. Wolper, Reasoning about Fair Concurrent Programs, *Proc. of the 18th Annual ACM Symp. on Theory of Computing*, Berkeley, pp. 283–294, 1986.

[Em81] Emerson, E.A., Branching Time Temporal Logic and the Design of Correct Concurrent

Programs, Ph.D. Thesis, Harvard University, 1981.

[Em85] Emerson, E.A., Automata, Tableaux, and Temporal Logics, *Proc. Conf. on Logics of Programs*, Brooklyn, R. Parikh, editor, Springer-LNCS#193, pp. 79–88, 1985.

[Em88] Emerson, E.A., Temporal and Modal Logic, *manuscript, to appear in* the *Handbook of Theoretical Computer Science*, J. van Leeuwen, editor, North–Holland.

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

[EH82] Emerson, E.A., J.Y. Halpern, Decision Procedures and Expressiveness in the Temporal Logic of Branching Time, *Proc. of the 14th Annual ACM Symp. on Theory of Computing*, San Francisco, pp. 169–180, 1982.

[EJ88] Emerson, E.A., C.S. Jutla, The Complexity of Tree Automata and Logics of Programs, *29th Annual Symp. on Foundations of Computer Science*, White Plains, pp. 328–337, 1988.

[EL85] Emerson, E.A., C.L. Lei, Modalities for Model Checking: Branching Time Logic Strikes Back, *Proc. 12th Annual ACM Symp. on Principles of Programming Languages*, New Orleans, pp. 84–96, 1985.

[EL86] Emerson, E.A., C.L. Lei, Temporal Reasoning under Generalized Fairness Constraints, *3rd Annual Symp. on Theoretical Aspects of Computer Science*, LNCS#210, Springer–Verlag, pp. 21–36, 1986.

[ES84] Emerson, E.A., A.P. Sistla, Deciding Full Branching Time Logic, *Information and Control*, **vol.** 61, no. 3, pp. 175–201, 1984.

[ES87] Emerson, E.A., T.H. Sadler, A Temporal Logic Decidable in Polynomial Time, *unpublished manuscript*, Sept. 1987.

[FL79] Fischer, M.J., R.E. Ladner, Propositional Dynamic Logic of Regular Programs, *Journal of Computer and System Sciences*, **vol.** 18, pp. 194–211, 1979.

[HV86] Halpern, J.Y., M. Vardi, The Complexity of Reasoning about Knowledge and Time, *Proc. of the 18th Annual ACM Symp. on Theory of Computing*, Berkeley, pp. 304–315, 1986.

[MP82] Manna, Z., A. Pnueli, Verification of Concurrent Programs: A Temporal Proof System, *Proc. 4th School on Advanced Programming*, Amsterdam, Holland, June 1982.

[MP83] Manna, Z., A. Pnueli, How to Cook a Proof System for Your Pet Language, *Proc. 10th Annual ACM Symp. on Principles of Programming Languages*, Austin, pp. 141–154, 1983.

[MW84] Manna, Z., P. Wolper, Synthesis of Communicating Processes from Temporal Logic Specifications, *ACM Transactions on Programming Languages and Systems*, **vol.** 6, no. 1, pp. 68–93, 1984.

[OL82] Owicki, S., L. Lamport, Proving Liveness Properties of Concurrent Programs, *ACM Transactions on Programming Languages and Systems*, **vol.** 4, no. 3, pp. 455–495, 1982.

[Pn77] Pnueli, A., The Temporal Logic of Programs, *18th Annual Symp. on Foundations of Computer Science*, Providence, pp. 46–57, 1977.

[Sa87] Sadler, T.H., Toward an Efficient Decision Procedure For Temporal Logics, M.S. Thesis (completed under the supervision of Prof. E.A. Emerson), CS Dept, UT–Austin, Dec. 1987.

[St82] Streett, R.S., Propositional Dynamic Logic of Looping and Converse is Elementarily Decidable, *Information and Control*, **vol.** 54, pp. 121–141, 1982.

[Va87] Vardi, M., Verification of Concurrent Programs: The Automata-Theoretic Framework, *Proc. 2nd Annual Symp. on Logic in Computer Science*, Ithaca, pp. 167–178, 1987.

[VS85] Vardi, M., L. Stockmeyer, Improved Upper and Lower Bounds for Modal Logics of Programs, *Proc. of the 17th Annual ACM Symp. on Theory of Computing*, Providence, pp. 240–251, 1985.

[VW84] Vardi M., P. Wolper, Automata Theoretic Techniques for Modal Logics of Programs, *Proc. of the 16th Annual ACM Symp. on Theory of Computing*, Washington D.C., pp. 446–456, 1984.

[VW86] Vardi, M., P. Wolper, An Automata-Theoretic Approach to Automatic Program Verification, *Proc. Symp. on Logic in Computer Science*, Cambridge, pp. 332–345, 1986.

1. $S \vee W$
2. $AG(\vee\{P, R, S, T, V, W\})$
3. $AG(P \Rightarrow AX(Q \vee R \vee S) \wedge EX(Q \vee R) \wedge EX\ S)$
4. $AG(Q \Rightarrow AX(R \vee T) \wedge EX\ T)$
5. $AG(R \Rightarrow AX(T \vee V) \wedge EX\ T \wedge EX\ V)$
6. $AG(S \Rightarrow AX(P \vee W \vee V) \wedge EX\ P \wedge EX(W \vee V))$
7. $AG(T \Rightarrow AX\ V \wedge EX\ V)$
8. $AG(V \Rightarrow AX\ R \wedge EX\ R)$
9. $AG(W \Rightarrow AX\ T \wedge EX\ T)$
10. $AG(R \Rightarrow AF(Q \vee V))$
11. $AG(T \Rightarrow AF(Q \vee V))$
12. $AG(V \Rightarrow AF(Q \vee V))$
13. $AG(S \Rightarrow AF\ R)$
14. $AG(T \Rightarrow AF\ R)$
15. $AG(V \Rightarrow AF\ R)$
16. $AG(W \Rightarrow AF\ R)$
17. $AG(P \Rightarrow AF\ R)$
18. $AG(Q \Rightarrow AF\ R)$

Figure 1: Set of assertions for Example 1

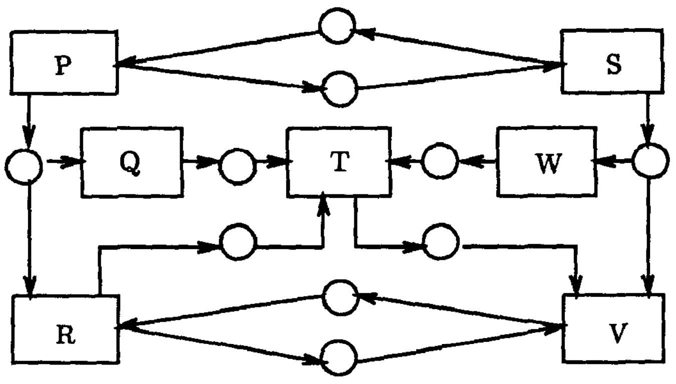

Figure 2: The tableau for the assertions of Example 1

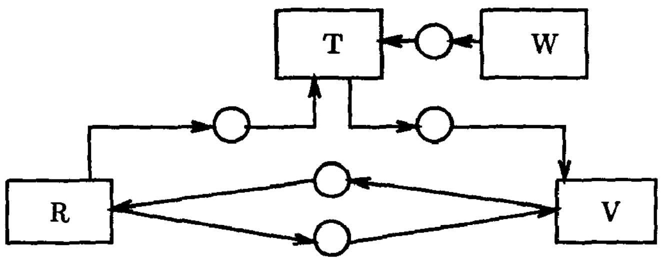

Figure 3: The pruned tableau for Example 1

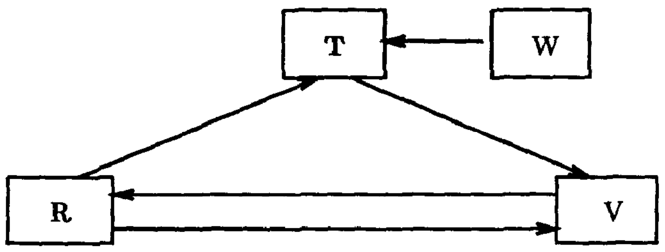

Figure 4: A model for the assertions of Example 1

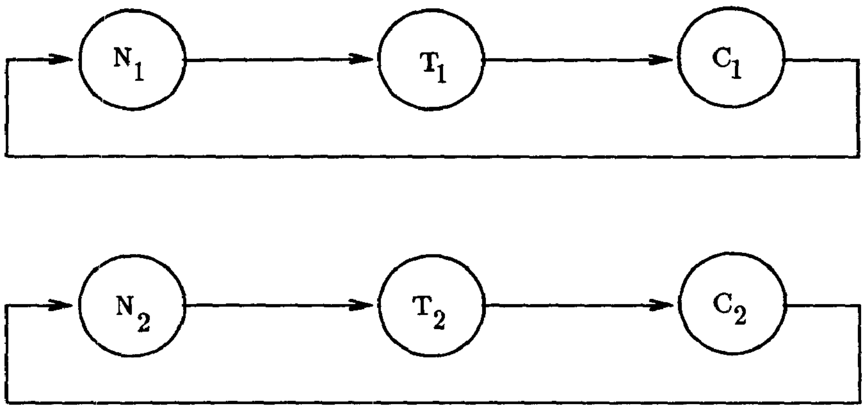

Figure 5: Synchronization skeletons for Examples 4 and 5

1. N_1N_2. This is the system's initial state.
2. $AG(R \Rightarrow AF(C_1N_2 \vee C_1T_2 \vee C_1C_2))$ for $R \in \{T_1N_2, T_1T_2, T_1C_2\}$. Process 1 must eventually get to its critical region from its trying region.
3. $AG(R \Rightarrow AF(N_1C_2 \vee T_1C_2 \vee C_1C_2))$ for $R \in \{N_1T_2, T_1T_2, C_1T_2\}$. Process 2 must eventually get to its critical region from its trying region.
4. $AG(\vee\{N_1N_2, N_1T_2, N_1C_2, T_1N_2, T_1T_2, T_1C_2, C_1N_2, C_1T_2\})$. The system may be in any state other than C_1C_2.

The successor assertions have the form

$$AG(R \Rightarrow AX(\vee\gamma) \wedge (\bigwedge EX\delta) \wedge EX(\vee\theta_1) \wedge EX(\vee\theta_2))$$

They are summarized in the following table:

R	γ	δ	θ_1	θ_2
N_1N_2	$\{T_1N_2, N_1T_2\}$	γ	$\{N_1T_2\}$	$\{T_1N_2\}$
T_1N_2	$\{C_1N_2, T_1T_2\}$	$\{T_1T_2\}$	γ	$\{C_1N_2\}$
N_1T_2	$\{N_1C_2, T_1T_2\}$	$\{T_1T_2\}$	$\{N_1C_2\}$	γ
C_1N_2	$\{C_1T_2, N_1N_2\}$	γ	γ	$\{N_1N_2\}$
N_1C_2	$\{N_1N_2, T_1C_2\}$	γ	$\{N_1N_2\}$	γ
T_1T_2	$\{T_1C_2, C_1T_2\}$	$\emptyset$	γ	γ
C_1T_2	$\{C_1C_2, N_1T_2\}$	$\{N_1T_2\}$	γ	γ
T_1C_2	$\{T_1N_2, C_1C_2\}$	$\{T_1N_2\}$	γ	γ
C_1C_2	$\{C_1N_2, N_1C_2\}$	γ	γ	γ

Figure 6: Global assertions for the mutual exclusion problem (Example 5)