# Efficient Temporal Satisfiability[†]

*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, Eindhoven University of Technology, The Netherlands

*Electronic mail address:* `emerson@cs.utexas.edu`

June 29, 1991

**Abstract**

There has been much interest in decision procedures for testing satisfiability in various systems of temporal logic. This is due to the potential applications of these decision procedures to reasoning about the correctness of concurrent programs. In this paper, we exhibit a temporal logic that is both useful in applications to program reasoning and is efficiently decidable in quadratic time. A surprising corollary of our results is that the fragment of CTL (Computation Tree Logic) used by Emerson & Clarke [EC82] to synthesize concurrent programs from temporal specifications is actually decidable in polynomial time. Our results demonstrate that many useful correctness properties can be expressed with a rather restricted syntax, and provide insight into the relation between the syntactic complexity of temporal logics and the complexity of testing 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. A summary of some these results was presented at the 16th Annual ACM Symposium on the Principles of Programming Languages in 1989.

## 1 Introduction

Over the last decade there has been a great deal of research into the complexity of decision procedures for testing satisfiability in various systems of modal or temporal logic, owing to their potential applications to reasoning about concurrent programs (cf. [Pn77], [FL79], [Ab80], [BMP81], [EH82], [VW84], [ES84], [PR89]). For all extant propositional logics of programs, the complexity of testing satisfiability is at least $\mathcal{NP}$-hard since these logics subsume ordinary propositional logic. Assuming $\mathcal{P} \neq \mathcal{NP}$, it thus seems most likely that the best deterministic decision procedure we could hope for is of exponential time 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 have formulated a logic, Simplified Computation Tree Logic (SCTL), which $(i)$ is decidable in polynomial time, and $(ii)$ still has applications to reasoning about concurrent programs. Thus, we demonstrate that many useful correctness properties can be expressed with a rather restricted syntax. Our results show, for example, that the satisfiability problem for the fragment of the logic used for program synthesis in [EC82] is actually decidable in polynomial time.

The basic intuition in formulating SCTL 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 essentially 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]). The standard paradigm for developing a decision procedure for most temporal logics 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 logics 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 size of the automaton is usually exponential 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, these logics are decidable in polynomial time.

Section 2 presents the syntax and the semantics of one such efficiently decidable temporal logic, Simplified Computation Tree Logic (SCTL), and Section 3 details a quadratic time decision procedure that tests the satisfiability of SCTL formulae. As is typical of many temporal logics, the construction of the transition diagram of the automaton (the tableau) for an SCTL formula is effected through propositional reasoning and can be done merely by inspecting the formula. The syntactic restrictions on SCTL ensure that the size of the automaton so obtained is polynomial in the formula's size. The next step is to test the automaton for non-emptiness; this is done by graph reachability analysis. It 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.

We then turn to applications of SCTL's satisfiability problem to program synthesis. As has been shown in [EC82] and [MW84], the decision procedure for the satisfiability problem of certain temporal logics is useful in mechanically synthesizing concurrent programs from their specifications. In Section 4, we show that the synthesis method described in [EC82] is applicable to program specifications expressed in SCTL as well. In general, however, program specifications are stated compositionally (i.e., in terms of the program's constituent processes), whereas SCTL is designed to describe the behaviour of the global program. Therefore, in Section 4, we define a restricted temporal logic, which we call cSCTL (compositional SCTL), to express specifications on processes. We also show how to mechanically translate process assertions expressed in this syntax to an equivalent SCTL formula that describes the global behaviour of the program. The translation algorithm as well as the synthesis method itself are both illustrated with the classical two-process mutual exclusion problem. We may note here that we can determine if a two process specification is consistent (i.e., if it can be realized by a program) in polynomial time ($O(n^4)$). In contrast, if the entire Computation Tree Logic (CTL) is permitted as the specification language, as it is in [EC82], the best provable upper bound on this problem is exponential in $n$. In practice, however, it appears that a fragment of CTL often suffices to express process specifications, and one such useful fragment is cSCTL.

## 2 Preliminaries

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 satisfiability problem for CTL, however, is deterministic exponential time complete, and, as noted in the introduction, this is a serious handicap to its use in reasoning about programs. So, our goal is to formulate a temporal logic which is, both, decidable in low polynomial time and expressive enough to state important temporal properties of programs.

An efficient algorithm for satisfiability requires that we must factor out aspects of propositional reasoning from those of temporal reasoning. This is essential because determining the satisfiability of a propositional formula is $\mathcal{NP}$-hard. As best we know, with $n$ ordinary, "overlapping" propositions, a decision procedure that tests satisfiability must consider each of the $2^n$ combinations of these propositions individually. So, we choose to construct the formulae of our logics from a set of mutually exclusive and exhaustive atomic propositions. (This restriction does not conflict with our expressiveness needs, for, in reasoning about programs, the propositions would usually correspond to program states, or to disjoint blocks in a partition of the program's state space.)

The other restrictions on the syntax of formulae permitted by our logics are guided by the kinds of program assertions that are frequently used in practice. Since program specifications are most often stated as conjunctions of simpler properties, we restrict the formulae to allow only conjunctions of simpler temporal assertions. These assertions, themselves, can describe initiality constraints (the program's starting states, for example), successors of program states, liveness properties, safety properties, and precedence properties.

In this section, we define a temporal logic, Non-Euclidean Simplified Computation Tree Logic (NESCTL), that abides by these syntactic restrictions. [‡] (Since the formulae of NESCTL inherit their semantics from CTL, we present CTL's syntax and semantics before defining NESCTL.) Despite NESCTL's restricted syntax, however, it can be shown that the time complexity of any decision procedure for testing the satisfiability of an NESCTL formula would still be exponential in the length of the formula. Thus, we place an additional restriction, which we call the euclidean syntactic constraint, on NESCTL formulae. The resulting logic is Simplified Computation Tree Logic (SCTL), and we show in the following section that SCTL is indeed efficiently decidable.

### 2.1 Computation Tree Logic: Syntax and Semantics

Let $\Sigma$ 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 (i.e., each state in $S$ has at least one $R$-successor), and $L$ is a labelling which assigns to each state a set of atomic propositions, those intended to be true at the state. Note, therefore, that a structure can be viewed as a vertex-labelled directed graph, and so, we shall use standard graph-theoretic notions (e.g., paths, reachability, etc.) over structures. 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 as follows:

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 $s_0 \models p$.

[‡]The term *Euclidean* will be explained subsequently.

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: $AF\, q$ abbreviates $A(\mathit{true}\ U\ q)$, $EF\, q$ abbreviates $E(\mathit{true}\ U\ q)$, $AG\, q$ abbreviates $\neg EF \neg q$, $EG\, q$ abbreviates $\neg AF \neg q$, and $AX\ q$ abbreviates $\neg EX \neg q$.

Examples of CTL formulae include: $AG(\neg C_1 \vee \neg C_2)$ 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 $AG(T_1 \Rightarrow AF\, C_1)$ which states that process 1 does not starve (whenever it is in its trying region, it eventually enters its critical region).

### 2.2 The Logics NESCTL and SCTL

Formulae of NESCTL and SCTL are also composed from a set, $\Sigma$, of atomic propositions. In this case, however, $\Sigma$ is constrained to be a finite alphabet, and, as we shall see later, the atomic propositions are mutually exclusive and exhaustive, i.e., precisely one proposition is true at each state of a structure. We shall use $P$, $Q$, etc., to denote atomic propositions in $\Sigma$, greek letters ($\alpha$, $\beta$, etc.) to denote subsets of $\Sigma$, and $\vee\alpha$ to denote the disjunction of propositions in $\alpha$.

We first spell out the syntax of formulae of NESCTL. An NESCTL formula is a conjunction of one or more of the following five types of assertions:

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

Note that the two CTL formulae cited as examples in the previous sub-section conform to this restricted syntax.

We now define a few useful terms. First, for simplicity of exposition, we shall refer to the antecedent and the consequent of the implication in an assertion of types 3, 4, or 5 above as the antecedent and the consequent, respectively, of the assertion itself. Also, a sub-formula of the form $AF(\vee\gamma)$ will be termed an *eventuality*, and one of the form $A((\vee\theta)\ U\ (\vee\gamma))$, an *assurance*. Thirdly, subformulae of the form $AX(\vee\alpha)$ (which are conjuncts of the consequents of successor assertions) will be called *universal conjuncts* and those of the form $EX(\vee\beta_i)$, *existential conjuncts*. And, for any NESCTL formula, $f$, composed from the alphabet $\Sigma$, and any pair of atomic propositions $P, Q \in \Sigma$, we shall say that $Q$ is a *permitted successor* of $P$ if $Q$ appears in the universal conjunct of every successor assertion in $f$ in which $P$ is the antecedent. (Note that this is vacuously true if $f$ has no successor assertion of which $P$ is the antecedent; in this case, every atomic proposition in $\Sigma$ is a permitted successor of $P$.) Finally, we shall adopt the following notational convention: Since every 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. Also, 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 $\wedge\mathcal{F}$.

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 $\Sigma$ 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 the state $s$ of a model, the non-initial assertions of $f$ are true at all states of the model reachable from $s$. Also, as $\Sigma$ is exhaustive, it is easily verified that the leads-to assertion $AG(P \Rightarrow AF(\vee\gamma))$ is equivalent to the ensures assertion $AG(P \Rightarrow A((\vee\Sigma)\ U\ (\vee\gamma)))$.

Despite the syntactic simplicity of NESCTL, it can be shown that its satisfiability problem is deterministic EXPTIME-complete ([EES90]). To understand why, we define one more term. Let the NESCTL formula $f$ have the ensures assertion $AG(P \Rightarrow A((\vee\theta)\ U\ (\vee\gamma)))$ (or the leads-to assertion $AG(P \Rightarrow AF\ (\vee\gamma)))$ as a conjunct. Consider a structure, $M$, and let $s$ be a state in $M$ which is labelled with $P$. The assurance $A((\vee\theta)\ U\ (\vee\gamma))$ (respectively, eventuality $AF\ (\vee\gamma)$) is said to be *unfulfilled* or *pending* at the state $t$ of $M$ if there is path in $M$ from $s$ to $t$ such that all propositions labelling the states along the path are in $\theta \setminus \gamma$ (respectively, are not in $\gamma$). Now, the intuition for the intractability of NESCTL is that, while it is no longer required to consider various subsets of atomic propositions that could be true at a state of a structure (as one must do with "overlapping" atomic propositions), it is still necessary to consider different sets of unfulfilled eventualities and assurances at different states of the structure labelled with the same atomic proposition. We explain this in more detail at the end of the next section.

So, we define the logic SCTL, which is a sublogic of NESCTL. Formally, an NESCTL formula $f$ is also an SCTL formula if it satisfies the *euclidean syntactic constraint,* which requires that:

($a$) for every leads-to assertion $AG(P \Rightarrow AF\ (\vee\gamma))$ that is a conjunct of $f$ and such that $P \notin \gamma$, $f$ also has the conjunct $AG(Q \Rightarrow AF\ (\vee\gamma))$ for every permitted successor $Q$ of $P$ that is not in $\gamma$, and

($b$) for every ensures assertion $AG(P \Rightarrow A((\vee\theta)\ U\ (\vee\gamma)))$ that is a conjunct of $f$ and such that $P \in (\theta \setminus \gamma)$, $f$ also has the conjunct $AG(Q \Rightarrow A((\vee\theta)\ U\ (\vee\gamma)))$ for every permitted successor $Q$ of $P$ that is in $\theta \setminus \gamma$.

For example, if $f$ has the conjuncts $AG(P \Rightarrow AF\ Q)$ and $AG(P \Rightarrow AX\ R)$, the euclidean syntactic constraint requires that $f$ also have $AG(R \Rightarrow AF\ Q)$ as a conjunct.[§] We remark, as an aside, that clause ($b$) of the definition subsumes clause ($a$) because ensures assertions generalize leads-to assertions. Also, note that we can test whether an NESCTL formula satisfies the euclidean syntactic constraint in time quadratic in the length of the formula.

In essence, for each SCTL formula $f$, the euclidean syntactic constraint *allows the unfulfilled eventualities and assurances at each state of each model of $f$ to be recovered from just the atomic proposition labelling that state;* the unfulfilled assurances (eventualities) at a state labelled $P$ are precisely those that appear as consequents in those ensures assertions (respectively, leads-to assertions) in $f$ of which $P$ is the antecedent. This restriction, together with SCTL's restricted syntax make a polynomial time decision procedure for testing satisfiability possible, as demonstrated in the next section.

For each SCTL formula $f$, we shall call the associated set, $\mathcal{F}$, of $f$'s conjuncts *an SCTL set,* and given an SCTL set $\mathcal{F}$, we write $\wedge\mathcal{F}$ to denote its corresponding SCTL formula. If the underlying set of atomic propositions from which $f$ is composed is $\Sigma$, we say that $f$ is an SCTL formula *over* $\Sigma$.

[§]In this simplest form, the constraint resembles the definition of an *euclidean* binary relation [Ch80]; hence, our choice of the name.

*Remark* **1** Let $f$ be an SCTL formula over $\Sigma$ and let $\Sigma_f$ be the set of atomic propositions that actually appear in $f$. In the sequel, we shall assume that $\Sigma$ has at most one proposition which is not in $\Sigma_f$. We require this assumption because it might appear that the complexity of some of the results we state below are dependent on $|\Sigma|$; as a consequence of this assumption, $|\Sigma|$ is $O(|f|)$, and the size of $\Sigma$ does not affect our complexity arguments. The justification for this assumption lies in the fact that the truth of $f$ at any state of any structure remains unchanged if all the atomic propositions which label that structure and which are not in $\Sigma_f$ are uniformly changed to one atomic proposition which is not in $\Sigma_f$. Thus, $f$ is satisfiable iff it has a model $M$ such that at most one of the propositions labelling states of $M$ is not in $\Sigma_f$. $\Box$

We conclude this section with a definition and a technical lemma concerning formulae of SCTL. (Analogous versions of these can be stated for all NESCTL formulae.) An SCTL set $\mathcal{G}$ (and the corresponding SCTL formula, $\wedge\mathcal{G}$) over $\Sigma$ is said to be in *canonical form* iff:

(*i*) $\mathcal{G}$ has precisely one initial assertion and one invariance assertion, and

(*ii*) every proposition $P \in \Sigma$ is the antecedent of at most one successor assertion in $\mathcal{G}$, say $AG\big(P \Rightarrow AX(\vee\alpha) \wedge (\bigwedge_i EX \vee \beta_i)\big)$, where $i$ ranges over a finite, possibly empty, index set. Moreover, every set $\beta_i$ of each existential conjunct in the consequent of this successor assertion must be a subset of the set $\alpha$ that appears in its universal conjunct.

We shall call the atomic propositions in the initial assertion of $\mathcal{G}$ the *initial propositions* of $\mathcal{G}$, and the successor assertion of which $P$ is the antecedent, the *successor assertion for* $P$. Also, we shall say that $P$ does not have a successor assertion in $\mathcal{G}$ if it is not the antecedent of any successor assertion in $\mathcal{G}$.

**Lemma 1** *Let $\mathcal{F}$ be an SCTL set over $\Sigma$. Then, there exists an SCTL set $\mathcal{G}$ over $\Sigma$ 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 $\wedge\mathcal{F}$, and the length of $\wedge\mathcal{G}$ is linear in the length of $\wedge\mathcal{F}$.*

*Proof:* Given $\mathcal{F}$, the set $\mathcal{G}$ is computed as follows. First, if $\mathcal{F}$ has no invariance assertion in it, the assertion $AG(\vee\Sigma)$ is added to $\mathcal{G}$; this is justified because $\Sigma$ is exhaustive. Also, if $\mathcal{F}$ has one or more invariance assertion, say $AG(\vee\delta_1)$, $AG(\vee\delta_2)$, …, $AG(\vee\delta_n)$, the single invariance assertion $AG \vee (\delta_1 \cap \delta_2 \cap \ldots \cap \delta_n)$ is added to $\mathcal{G}$. Similarly, the initial assertion $\vee\Sigma$ is added to $\mathcal{G}$ if $\mathcal{F}$ has no initial assertions, and, if $\mathcal{F}$ has one or more initial assertions, say, $(\vee\theta_1)$, $(\vee\theta_2)$, …, $(\vee\theta_k)$, the single initial assertion $\vee(\theta_1 \cap \theta_2 \cap \ldots \cap \theta_k)$ is added to $\mathcal{G}$.

Next, we consider the successor assertions in $\mathcal{F}$. Consider the proposition $Q$. Suppose that $\mathcal{F}$ has more than one successor assertion of which $Q$ is the antecedent, say, $AG(Q \Rightarrow AX(\vee\alpha) \wedge (\bigwedge_i EX \vee\beta_i))$, and $AG(Q \Rightarrow AX(\vee\alpha') \wedge (\bigwedge_j EX \vee\beta'_j))$. These two assertions are removed from $\mathcal{F}$ and replaced by the single assertion $AG(Q \Rightarrow AX \vee (\alpha \cap \alpha') \wedge (\bigwedge_i EX \vee\beta_i) \wedge (\bigwedge_j EX \vee\beta'_j))$. This is repeated until $\mathcal{F}$ has just one successor assertion whose antecedent is $Q$, and then, this assertion is copied to $\mathcal{G}$. It is easy to see that the set $\alpha$ that appears in the universal conjunct of this assertion contains precisely the permitted successors of $Q$ in $\wedge\mathcal{F}$. We then ensure that each successor assertion in $\mathcal{G}$ meets the constraint placed on it in the definition of a canonical formula: every assertion of the form $AG(Q \Rightarrow AX(\vee\alpha) \wedge (\bigwedge_i EX \vee\beta_i))$ in $\mathcal{G}$ is replaced by $AG(Q \Rightarrow AX(\vee\alpha) \wedge (\bigwedge_i EX \vee(\alpha \cap \beta_i)))$; note that the propositions in $\beta_i \setminus \alpha$ are not permitted successors of $Q$.

To complete the construction of $\mathcal{G}$, the leads-to and the ensures assertions of $\mathcal{F}$ are added to $\mathcal{G}$ without modification. Note that $\mathcal{G}$ satisfies the euclidean syntactic constraint if $\mathcal{F}$ does as the permitted successors of each atomic proposition are unchanged. It is easily seen that $\wedge\mathcal{G}$ and $\wedge\mathcal{F}$ are equivalent, that $\mathcal{G}$ can be obtained from $\mathcal{F}$ in time linear in the length of $\wedge\mathcal{F}$, and that the length of $\wedge\mathcal{G}$ is 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 can be stated as follows: *Given a finite set $\Sigma$ of mutually exclusive and exhaustive atomic propositions and an SCTL set $\mathcal{F}$ over $\Sigma$, 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* from $\mathcal{F}$. 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 in $\Sigma$. 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). Each AND node is labelled with the eventualities and assurances of which they are the antecedents in the leads-to and ensures assertions of $\mathcal{F}$. Also, any initial proposition of $\mathcal{F}$ is termed an *initial node* of the tableau.

We now describe how the edges in $T_0$ are added. First, consider any successor assertion, say $AG(Q \Rightarrow AX(\vee\alpha) \wedge (\bigwedge_i EX\vee\beta_i))$, in $\mathcal{F}$. The AND node $Q$ has one OR node successor for each conjunct in the consequent of this assertion. The OR node corresponding to the universal conjunct has as its successors the AND nodes in $\alpha$, and the OR node corresponding to the $i$th existential conjunct has as its successors the AND nodes which are elements of $\beta_i$.

We now consider atomic propositions in $\Sigma$ which do not have any successor assertion in $\mathcal{F}$; let $\Theta$ be the set of such atomic propositions. Let $\Psi$ be the subset of $\Theta$ containing those propositions whose AND nodes are labelled with at least one assurance and let $\Phi = \Theta \setminus \Psi$. For each AND node, $Q$, in $\Psi$, we add one OR node successor, $x_Q$, to $T_0$. The successors of $x_Q$ are the elements of $\bigcap_H(\theta_H \cup \gamma_H)$, where $H = A((\vee\theta_H)\ U\ (\vee\gamma_H))$ ranges over the assurances labelling $Q$. The AND nodes in $\Phi$, on the other hand, have a common OR node successor whose successors are all the AND nodes in $T_0$.

Essentially, $T_0$ accounts for the constraints imposed by the 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 the state $t$) and $P$ has a successor assertion in $\mathcal{F}$, then $s$ must have one successor state $r_x$ for each OR node successor $x$ of $P$ in $T_0$ and $r_x$ must be labelled with some AND node successor of $x$ in $T_0$. The reason for adding OR node successors for the elements of $\Theta$ is that we require that $M$ be total, e.g., if $P \in \Theta$ (i.e., $P$ does not have any successor assertion in $\mathcal{F}$), then $s$ is still required to have a successor state, say $t$, in $M$. If $P \in \Phi$, $t$ may be labelled with any element of $\Sigma$. On the other hand, if $P \in \Psi$, the proposition labelling $t$ is constrained by the assurances pending at state $s$, i.e., the ones labelling the AND node $P$ in $T_0$.

*Remark* **2** It is easy to see that the size of $T_0$ (i.e., the sum of the number of nodes and edges in it) is linear in the length of $\wedge\mathcal{F}$. In fact, it is linear in the sum of the lengths of the successor assertions and the assurances in $\mathcal{F}$, and, if $\mathcal{F}$ has a successor assertion for every proposition in $\Sigma$, it is linear in the length of the successor assertions alone. □

An important property of $T_0$ 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 $A((\vee\theta)\ U\ (\vee\gamma))$ (the eventuality $AF\ (\vee\gamma)$) is in the label of $Q$, and all AND nodes in $p$ are in $(\theta \setminus \gamma)$ (respectively, are not in $\gamma$), then $A((\vee\theta)\ U\ (\vee\gamma))$ (respectively, $AF\ (\vee\gamma)$) is also in the label of $R$.*

*Proof:* Follows from a simple induction argument on the length of $p$ using the fact that $\wedge\mathcal{F}$ satisfies the euclidean syntactic constraint. □

**Example 1** Let $\Sigma = \{P, Q, R, S, T, V, W\}$ and $\mathcal{F}$ have the assertions shown in Fig. 1. Note that $\wedge\mathcal{F}$ is in canonical form and it fulfills the euclidean syntactic constraint. For example, the leads-to assertion $AG(S \Rightarrow AF\ R)$ is in $\mathcal{F}$ and the permitted successors of $S$ are $P$, $W$, and $V$. So $\mathcal{F}$ also has an assertion (assertions 15–17) with each of these propositions as the antecedent of the eventuality $AF\ R$.

The initial tableau constructed from $\mathcal{F}$ is shown in Fig. 2. AND nodes are shown as boxes and OR nodes as circles. Two of the OR nodes have been named $x$ and $y$ for easy reference in a subsequent example. As the eventualities 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; the ensures assertion $AG(P \Rightarrow A((\vee\theta)\ U\ (\vee\gamma)))$ in $\mathcal{F}$ would be false at any state of any structure labelled with $P$. 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 $t$ succeeding a state $s$ labelled $P$ in some structure, the ensures assertion $AG(P \Rightarrow A((\vee\theta)\ U\ (\vee\gamma)))$ would be false at $s$. Note that this is the only deletion rule that removes edges alone (i.e., without removing incident nodes) from the tableau, and it only removes edges from OR to AND nodes. Moreover, edges from OR nodes whose predecessor AND nodes are in $\Theta$ are not deleted by this rule. We refer to this pruning rule as the *edge deletion rule.*

Now, the following two deletion rules are repeatedly applied. First, any AND node, some one of whose successors in $T_0$ has been deleted, and any OR node with no successors, is deleted. The second rule requires that eventualities and assurances labelling AND nodes be *fulfilled* in the tableau. Let the eventuality $AF\ (\vee\gamma)$ be in the label of an AND node $Q$. *$Q$ fulfills this eventuality* if the tableau contains a DAG (directed acyclic graph) rooted at $Q$, fulfilling this eventuality, i.e., 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). Also, any interior AND node in the DAG[¶] must have as its successors all its successors in the tableau, and every frontier node of the DAG must be an AND node $\gamma$. Similarly, *$Q$ fulfills the assurance* $A((\vee\theta)\ U\ (\vee\gamma))$ if the tableau contains a DAG (directed acyclic graph) rooted at $Q$, fulfilling this assurance; such a DAG must satisfy all the conditions listed above, and, in addition, every interior AND node in the DAG must be in $(\theta \setminus \gamma)$. If an AND node is labelled with an eventuality or assurance that it does not fulfill, it is deleted.

The pruning procedure terminates either by deleting all the initial nodes, in which case $\wedge\mathcal{F}$ is unsatisfiable, or by generating a tableau which has an initial AND node and from which no more nodes or edges can be deleted by the above rules. Such a tableau can be unwound to a model of $\wedge\mathcal{F}$; so, in this case, $\wedge\mathcal{F}$ is satisfiable.

[¶] An *interior* node of a directed acyclic graph is one that has a successor in the graph, and a *frontier* node is one that does not.

**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 $S$ (because $P$ must be a successor of the OR node $y$ in any such DAG, and $S$ in turn must be a successor of the OR node $x$). The deletion of $S$ causes the deletion of its OR node predecessor $x$, which is a successor of $P$, and, hence, $P$ is deleted. No other AND nodes can be deleted, and the pruned tableau is shown in Fig. 3. In this tableau, 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 $AF\,R$. (For example, the DAG fulfilling $AF(Q \vee V)$ rooted at $R$ is the subgraph of the pruned tableau obtained by deleting $W$, its OR node successor and the OR node successor of $V$.) Since the pruned tableau contains an initial node (viz., $W$), $\wedge\mathcal{F}$ is satisfiable. A model for $\wedge\mathcal{F}$ is shown in Fig. 4.

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 from the initial tableau. Similarly, 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 still satisfies the euclidean syntactic constraint and its initial tableau is the same as the one shown in Fig. 2. However, this set of assertions is not satisfiable. As before, $Q$, $P$ and $S$ are deleted. But now, $T$ is also deleted ($P$ is not reachable from $T$, so there is no DAG fulfilling $AF\,P$ rooted at $T$), causing the only surviving initial node, $W$, to be deleted. $\square$

For the formula $\wedge\mathcal{F}$ of Example 1, the model we exhibited is, in a sense, *contained* in the pruned tableau: to construct the model, all that we did was to drop the OR nodes from the pruned tableau and replace their incoming edges with an edge from their AND node predecessor to each of their AND node successors (and then delete parallel edges). In general, however, this technique of constructing a model will not work as it may result in a structure that has cycles along which an assurance remains unfulfilled. The proof of the following theorem shows, among other things, how to use the DAGs that fulfill assurances in the pruned tableau to construct a model in general.

**Theorem 1** *The above algorithm decides the satisfiability of its input, $\wedge\mathcal{F}$, correctly and in time quadratic in the length of $\wedge\mathcal{F}$.*

*Proof:* The proof of correctness of the algorithm is structured as follows. We first show that if the pruning procedure 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 $s$. Then, we show the converse, viz., if $P$ is an AND node in the pruned tableau, there is a structure $M$ that has a state $s$ labelled with $P$ at which all non-initial assertions of $\mathcal{F}$ are true. It follows that $\wedge\mathcal{F}$ has a model iff the pruned tableau contains an initial node.

Let the AND node $P$ be deleted by the pruning procedure from the initial next time tableau constructed from $\mathcal{F}$. We shall prove that, in all structures $M$ and states $s$ of $M$ labelled with $P$, some non-initial assertion of $\mathcal{F}$ is false. We do this by induction on the order of deletion of the AND nodes. This is clear if $P$ is deleted is deleted by Rule 1 (i.e., because it does not occur in the invariance assertion of $\wedge\mathcal{F}$), and, similarly, if it is deleted by Rule 2 because it is labelled with the assurance $A((\vee\theta)\ U\ (\vee\gamma))$ and $P$ itself is not in $(\theta \cup \gamma)$.

Suppose $P$ is deleted because some one of its successor OR nodes, say $u$, is deleted. Now, if $P$ does not have a successor assertion in $\mathcal{F}$, it follows that $P$ must be in $\Psi$, i.e., there is an assurance in the label of $P$. (For if $P \in \Phi$, there is an edge from $u$ to $P$ in $T_0$ which is not deleted by the edge deletion rule; thus $u$ cannot be deleted unless $P$ is.) Let $P$ label a state, say $s$, of a structure $M$. As $M$ must be total, $s$ has a successor state, say $t$, labelled with $Q$. Now, if $Q$ is not a successor of $u$ in $T_0$, some ensures assertion of which $P$ is the antecedent is false at $s$. Otherwise, $Q$ has been deleted from the tableau (as the edge deletion rule does not delete edges out of $u$); thus, by the induction hypothesis, some non-initial assertion of $\mathcal{F}$ is false at $t$, and hence, at $s$.

So, let $P$ have a successor assertion in $\mathcal{F}$ and let $u$ correspond to the $i$th conjunct in the consequent of the successor assertion for $P$. Let the $i$th conjunct be $AX(\vee\delta)$ if it is a universal conjunct and $EX(\vee\delta)$ if it is an existential conjunct. Let $P$ label state $s$ in structure $M$. We consider two possibilities. The first is $\delta$ is empty, i.e., if $u$ has no successors in $T_0$. Now, because $M$ is total, $s$ must have at least one successor, and each successor must be labelled with an element of $\Sigma$. Thus, the successor assertion is false at $s$: if $u$ corresponds to a universal conjunct, it is false because there is a successor which is labelled with an element of $\Sigma$, and if $u$ corresponds to an existential conjunct, the assertion is false because every successor is labelled with an element in $\Sigma$. The second possibility is that $\delta$ is nonempty, but its elements, which are the successors of $u$ in $T_0$, are no longer its successors. Let $Q$ be a proposition in $\delta$ such that some successor state, $t$, of $s$ is labelled with $Q$. (If there is no such proposition, the successor assertion is clearly false at $s$ if $u$ corresponds to an existential conjunct. It is also false if $u$ corresponds to a universal conjunct because there must be a successor state of $s$ in $M$ and every successor state must be labelled with an element of $\delta$.) Now, $Q$ is no longer a successor of $u$ in the tableau. If $Q$ has been deleted, we are done: some non-initial assertion of $\mathcal{F}$ is false at $t$ (by the induction hypothesis), and, hence, is false at $s$. If $Q$ is still in the tableau, the edge from $u$ to $Q$ has been deleted. But this could have been done only by the edge deletion rule (Rule 2) because there is an assurance $A((\vee\theta)\ U\ (\vee\gamma))$ in the label of $P$ such that $P \in (\theta \setminus \gamma)$, but $Q \notin (\theta \cup \gamma)$. In this case, the ensures assertion $AG(P \Rightarrow A((\vee\theta)\ U\ (\vee\gamma)))$ is false at $s$.

Finally, let $P$ be deleted because it has an assurance $A((\vee\theta)\ U\ (\vee\gamma))$ in its label but a DAG rooted at $P$ cannot be constructed to fulfill this assurance. (The proof when $P$ is deleted because it has the eventuality $AF\ (\vee\gamma)$ in its label but does not have a DAG rooted at $P$ is similar.) It follows that $P \notin \gamma$ and, as $P$ has not been deleted by Rule 2, that $P \in (\theta \setminus \gamma)$. Again, let $P$ label state $s$ of structure $M$. If the invariance assertion or a successor assertion of $\mathcal{F}$ is false at a state reachable from $s$, then that assertion is false at $s$, and we are done. So, suppose that the invariance assertion and the successor assertions of $\mathcal{F}$ are true at all states reachable from $s$. Similarly, if a state in $M$ which is reachable from $s$ is labelled with a proposition that has been deleted from the tableau, we have shown by the induction hypothesis that some non-initial assertion is false at that state, and, hence, is false at $s$. So we assume that all states reachable from $s$ in $M$ are labelled only with AND nodes that have not been deleted as yet.

We show that the ensures assertion $AG(P \Rightarrow A((\vee\theta)\ U\ (\vee\gamma)))$ is false at $s$. Now, if every OR node successor of $P$ had an AND node successor in the tableau which has a DAG fulfilling the assurance $A((\vee\theta)\ U\ (\vee\gamma))$ rooted at it, then $P$ would have a DAG rooted at it fulfilling this assurance. So, let $u$ be an OR node successor of $P$ such that none of its AND node successors has a DAG rooted at it for the assurance $A((\vee\theta)\ U\ (\vee\gamma))$. Now, $P \in (\theta \setminus \gamma)$ and none of the AND node successors of $u$ is in $\gamma$; so, because of the edge deletion rule (Rule 2), all these AND node successors are in $(\theta \setminus \gamma)$. (Recall that the edge deletion rule is applied to the tableau before the DAG deletion rule is.) Because of the euclidean syntactic constraint on $\mathcal{F}$, all the AND node successors of $u$ have $A((\vee\theta)\ U\ (\vee\gamma))$ in their labels. Now, $s$ must be followed by a state $t$, labelled with a proposition, say $Q$, which is an AND node successor of $u$ (this is clearly necessary if $u$ corresponds to an existential conjunct in the successor assertion for $P$, and if $u$ corresponds to a universal conjunct, it follows because $s$ must have a successor state and every successor state must be labelled with some AND node successor of $u$). The above argument can be repeated with $Q$ for $P$ and $t$ for $s$, and so on, arbitrarily many times. Thus, there is a fullpath in $M$ that starts at $s$ and at each of whose states the assurance $A((\vee\theta)\ U\ (\vee\gamma))$ is pending. Thus the ensures assertion $AG(P \Rightarrow A((\vee\theta)\ U\ (\vee\gamma)))$ is false at $s$.

We now show that the pruning procedure can be implemented to run in time quadratic in the length of $\wedge\mathcal{F}$. In fact, we prove a tighter upper bound here, one which we use to establish an upper bound on the synthesis method. We note, from Remark 2 that the size of the initial next time tableau (i.e., the number of vertices and edges in it) is linear in the length of $\wedge\mathcal{F}$.

We implement the pruning procedure as follows. First, deletion rules 1 and 2 are applied to the initial next time tableau, $T_0$. Clearly, this can be done in time linear in the size of $T_0$.

Next, suppose $\mathcal{F}$ has $m$ assurances, $H_1, H_2, \ldots, H_m$, where $H_j = A((\vee\theta_j)\ U\ (\vee\gamma_j))$. After the application of deletion rule 2, we note that if the assurance $H_j$ labels an AND node $P$ in the tableau, then $P$ occurs in either $\theta_j$ or $\gamma_j$; Rule 2 would have deleted $P$ if this were not the case.

The rest of the procedure requires us to apply the successor deletion rule (Rule 3) and the DAG fulfillment rule (Rule 4) in any order till no more nodes can be deleted. We optimize this instead. Initially, we apply the successor deletion rule once, and the DAG fulfillment rule (Rule 4) once for each assurance and each eventuality in $\mathcal{F}$. The successor deletion rule can be applied in time linear in the size of the tableau. Also, when Rule 4 is applied to check fulfillment of the assurance $H_j$ (respectively, eventuality $E_k = AF\ (\vee\gamma'_k)$, $1 \leq k \leq m'$), we can easily determine all the AND nodes in the tableau that have $H_j$ (respectively, $E_k$) in their labels but do not have a DAG that fulfills it in time linear in the size of the tableau.

Now, suppose in the initial application of Rule 3 or in one of the $m$ initial applications of Rule 4 (that is, once for each of the $m$ assurances), an AND node, say $P$, is deleted. We must re-apply the successor deletion rule to the new tableau (the one without $P$) to ensure all nodes have the requisite successors. Thus, we would need to apply the successor deletion rule at most $O(|T_0|)$ times; in fact, the maximum number of applications is one more than the number of AND nodes in $T_0$. The cost of each application in linear in the size of $T_0$.

Now, consider applications of Rule 4. Again, suppose that some AND node, say $P$, is deleted. It might appear that we also need to apply Rule 4 once for each of the $m$ assurances and $m'$ eventualities in $\mathcal{F}$. However, we argue that we need to apply Rule 4 only for those assurances in which $P$ occurs and once for each eventuality in which $P$ occurs or of which $P$ is the antecedent in some leads-to assertion in $\mathcal{F}$. First, consider an assurance, say $H_j = A((\vee\theta_j)\ U\ (\vee\gamma_j))$, such that $P$ is not in either $\theta_j$ or $\gamma_j$. Let $T$ be the tableau before the deletion of $P$. Now, if the AND node $Q$ in $T$ is labelled with $H_j$ and has a DAG rooted at it in $T$ for the assurance $H_j$, then this DAG cannot contain $P$ because, from the definition of a DAG, every AND node in the DAG must be in $\theta_j \cup \gamma_j$. So the deletion of $P$ does not cause the fulfillment of $H_j$ at $Q$ to be violated. So, when $P$ is deleted, it suffices to apply Rule 4 for only those assurances in which $P$ occurs. Hence, the maximum number of times the assurance $H_j$ is tested for fulfillment is $1 + |\theta_j| + |\gamma_j|$, and the time it takes for each test is $O(|T_0|)$.

Next, consider an eventuality, say $E_k = AF\ (\vee\gamma'_k)$, such that $P$ is not in $\gamma'_k$ and is not the antecedent of $E_k$ in any leads-to assertion in $\mathcal{F}$. Let $T$ be the tableau before the deletion of $P$. Now, if the AND node $Q$ in $T$ is labelled with $E_k$ and has a DAG rooted at it in $T$ for the eventuality $E_k$, then this DAG cannot contain $P$ because this would contradict Lemma 2. So the deletion of $P$ does not cause the fulfillment of $E_k$ at $Q$ to be violated. Hence, the maximum number number of times the eventuality $E_k$ is tested for fulfillment is $1 + n_{E_k} + |\gamma'_k|$ where $n_{E_k}$ is the number of leads-to assertions in $\mathcal{F}$ whose consequent is $E_k$, and the time it takes for each test is $O(|T_0|)$.

Thus, the total time taken to implement the pruning rules is $O(|T_0| + |T_0|^2 + |T_0| \times K)$, where $K$ is the sum of the lengths of all the assurances, the lengths of all the eventualities and the number of the leads-to assertions in $\mathcal{F}$. The first component in the sum is the time taken to test Rules 1 and 2, the second component is the time taken to test Rule 3 (the successor deletion rule), and the last component is the time to test Rule 4. As $|T_0|$ is linear in the length of $\wedge\mathcal{F}$, the result follows.

It remains to show that if the procedure leaves behind an AND node $R$, there is a structure $M$ containing a state labelled with $R$ at which all non-initial assertions are true, and, further, $M$ is model of $\wedge\mathcal{F}$ if the pruned tableau has an initial node. We construct $M$ using a method described in [Em90]. $M$ is best visualized as a two-dimensional matrix whose rows correspond to the eventualities and the assurances in $\mathcal{F}$ and whose columns, to AND nodes. If the proposition $P$ is the antecedent of the assurance $A((\vee\theta)\ U\ (\vee\gamma))$ (eventuality $AF\ (\vee\gamma)$) in some ensures assertion (respectively, leads-to assertion) in $\mathcal{F}$, and $P \notin \gamma$, the matrix element in the row corresponding to $A((\vee\theta)\ U\ (\vee\gamma))$ (respectively, $AF\ (\vee\gamma)$) and in the column for $P$ is a DAG rooted at $P$ fulfilling the assurance (respectively, eventuality). Otherwise, this entry is the subgraph of the pruned tableau induced by the AND node $P$, all its OR node successors and all the AND node successors of these OR nodes, except that if an OR node successor $u$ of $P$ has an edge back to $P$, that edge is deleted and directed instead to a fresh AND node also labelled with $P$. Note that such a subgraph is also acyclic and has only one root. Call each matrix entry a DAGG. Now, coalesce each frontier node $Q$ of each DAGG of the $i$th row with the root of the DAGG in the column corresponding to $Q$ and the $(i+1)$th row (with the last row wrapping around to the first). Finally, the OR nodes are deleted, and, for each such OR node $u$, edges are added from the AND node predecessor of $u$ to all successor AND nodes of $u$. The resulting structure is $M$.

It is easy to see that each AND node in the pruned tableau labels some state of $M$. We show that all non-initial assertions of $\mathcal{F}$ are true at all states of $M$. First, the states of $M$ are labelled by AND nodes of the pruned tableau, and so the invariance assertion is true at every state. Also, the construction of $M$ ensures that each state, $s$, of $M$, labelled with $P$, say, has a successor state, $t_u$, for each OR node successor, $u$, of $P$ in $T_0$, and $t_u$ is labelled with one of the AND node successors of $u$ in $T_0$. Since $T_0$ enforces the successor assertion constraints, all the successor assertions are true at each state of $M$.

So we show that each state of $M$ satisfies every ensures assertion in $\mathcal{F}$. (The proof that each state of $M$ satisfies every leads-to assertion in $\mathcal{F}$ is similar.) Consider the assertion $AG(P \Rightarrow A((\vee\theta)\ U\ (\vee\gamma)))$ in $\mathcal{F}$, and let state $s$ of $M$ be labelled with $P$. We show that the assurance $H = A((\vee\theta)\ U\ (\vee\gamma))$ is fulfilled along every fullpath starting at state $s$. Note that any fullpath in $M$ goes through every row of the matrix, and, hence, through a state labelled with the root node of some DAGG of the assurance $H$ and then to one labelled with a frontier node of that DAGG. Consider any fullpath $p$ out of $s$. Let $s'$ be the first node, if any, after $s$ on $p$ to be labelled with an element of $\gamma$. Every node between $s$ and $s'$ (or after $s$, if $s'$ does not exist) is labelled with an element of $\theta \setminus \gamma$ (because of the edge deletion rule). So it remains to show that some element of $\gamma$, actually labels a state $s'$ that occurs in $p$ after $s$. But after it visits $s$, $p$ goes through a state, call it $s''$, that is labelled with $R$ and is the root node of some DAGG of the assurance $H$. If no node between $s$ and $s''$ (including both) is labelled with an element of $\gamma$, then, by the euclidean syntactic constraint on $\mathcal{F}$ and the fact that all AND nodes between $s'$ and $s''$ are in $\theta \setminus \gamma$, the assurance $H$ is in the label of $R$ in the pruned tableau. So the DAGG of which $R$ is the root is really a DAG fulfilling $H$ at its frontier nodes. Now, $p$ goes through a state labelled with one of the frontier nodes of this DAG, and, hence, fulfills the assurance $H$.

Hence, if $P$ is an AND node in the pruned tableau, a state labelled with $P$ occurs in $M$, and all non-initial assertions of $\mathcal{F}$ are true at this state. Also, if the pruned tableau has any initial AND node, $M$ is a model for $\wedge\mathcal{F}$ as it has a state labelled with an initial proposition and all non-initial assertions of $\mathcal{F}$ are true at this state. This completes the proof of the theorem. □

**Theorem 2** *The satisfiability problem for SCTL is $\mathcal{P}$-hard.*

*Proof:* The hardness proof is sketched here by reducing an arbitrary language in $\mathcal{P}$ to the satisfiability problem for SCTL in deterministic log space. Let $L$ be any language in $\mathcal{P}$ and let $M$ be a log space bounded alternating Turing machine that accepts $L$. (The reader is referred to [CKS81] for a definition of alternating Turing machines.) Without loss of generality, let $M$ have one input and one work tape, and let $M$ always have some move from each configuration; further, let the only successor configuration of any accepting configuration $C$ of $M$ be $C$ itself. Given $M$ and an input word $x$, we construct an instance of the SCTL satisfiability problem that is satisfiable iff $M$ accepts $x$, i.e., iff $x \in L$. The atomic propositions in $\Sigma$ are the configurations of $M$ on input $x$ (its state, its work tape contents, and the input and work tapes' head positions). The initial assertion has the initial configuration of $M$ as its only disjunct, and the successor assertions are obtained in the obvious manner from the next move function of $M$. For example, if $P$ is an existential configuration and $\alpha_P$ is the set of successor configurations of $P$, the successor assertion is $AG(P \Rightarrow AX(\vee\alpha_P))$. On the other hand, if $P$ is a universal configuration and $\alpha_P$ is the set of successor configurations of $P$, the successor assertion is $AG(P \Rightarrow AX(\vee\alpha_P) \wedge \bigwedge_{Q\in\alpha_P} EX\ Q)$.

There is one leads-to assertion, $AG(P \Rightarrow AF(\vee\gamma))$, for each non-accepting configuration $P$; here, $\gamma$ is the set of all the accepting configurations of $M$. Clearly, this set of assertions is an SCTL set: specifically, it satisfies the euclidean syntactic constraint. Also, it has a model (intuitively, the accepting "tree" of $M$ on $x$) iff $M$ accepts $x$. Finally, for each $M$, it is easy to see that there is a log space bounded deterministic Turing machine that transforms its input $x$ to this instance of the satisfiability problem. □

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 SCTL's initial, invariance, and successor assertions over the (mutually exclusive and exhaustive) alphabet $\Sigma$ ensures that the tableau built for an SCTL formula, 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 eventualities and 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 eventuality and 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 eventualities and assurances, and the ordinary 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 eventualities and assurances that are required to be fulfilled there. Different sets of eventualities and assurances are required because the set of unfulfilled eventualities and assurances depends on the path followed to get to a node in the next time tableau.[‖] But by the euclidean syntactic constraint on SCTL, there is a unique set of eventualities and 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 Applications of SCTL to Program Reasoning

The way we typically use SCTL to reason about concurrent programs is to associate with each atomic proposition a unique block in a partition of the program's state space. For simplicity, in the sequel, we shall assume that these blocks contain a single state of the program. Thus, it is readily seen that the reachability graph of a program defines a temporal structure: the vertices of the reachability graph are the states of the structure and the program's state at a vertex is the label of the corresponding state in the structure. Thus, each kind of assertion allowed in SCTL formulae corresponds to one class of program properties. For example, the initial assertion $P \vee Q$ says that the program starts in one of the states $P$ or $Q$, and $AG(P \Rightarrow AF(Q \vee R))$ says that whenever the program is in state $P$, it will always get to one of the two states, $Q$ or $R$, along every execution sequence.

[‖]We may remark that this explanation, till here, applies equally well to tableaux constructed for NESCTL formulae. This provides some intuition as to why NESCTL's satisfiability problem is deterministic EXPTIME-complete, despite the fact that its syntactic restrictions ensure that the local automaton is linear in the formula size.

Our model of computation assumes that the concurrent program itself is composed of a fixed number of processes. Each of these processes, in turn, is specified by its states (again, in general, these could be blocks of states) and state transitions. Thus, the program's state at any instant, is determined by the states of the component processes at that instant. As is usual, we shall assume that the transitions of the processes are atomic and can be interleaved to obtain an execution sequence of the program. Therefore, an edge in the temporal structure of a concurrent program corresponds to a transition of one of the processes. In this paper, we permit all possible execution sequences, i.e., we assume that the processes are scheduled nondeterministically. We note, however, that we could use fairness constraints to restrict the permitted set of execution sequences, and the techniques presented here to reason about programs can be extended using the methods developed in [EL86].

While SCTL is used to describe the behaviour of the global program, specifications on a concurrent program are commonly expressed compositionally, i.e., in terms of its component processes. Thus, we have to derive the SCTL formula that describes the program's behaviour from these compositional assertions. In the next sub-section, we describe cSCTL (compositional SCTL) which we use to specify processes and, in a following sub-section, show how to mechanically translate such assertions to an SCTL formula that describes the behaviour of the concurrent program.

The last sub-section demonstrates how the satisfiability procedure for SCTL can be applied to automate program synthesis. The synthesis procedure is illustrated with an example from the literature.

### 4.1 Compositional Assertions

Each process can be thought of as being described by its local flowgraph and by other properties (e.g., liveness properties) it is required to satisfy. The flowgraph itself specifies the states the process is permitted to be in, the starting state of the process, and its state transitions. The interpretation of states and transitions in the flowgraph may vary depending on the application. For the synthesis problem, for example, the flowgraph represents the process' *synchronization skeleton*. Each state models a local computation, and the transitions represent permitted sequences of local computations. In this case, the flowgraph serves as a *specification* of the process' desired behaviour.

Formally, the flowgraph of the process and the other properties it is required to satisfy are expressed as formulae of cSCTL (compositional SCTL), which corresponds to the fragment of CTL originally used for synthesis in [EC82]. We now describe cSCTL. Like SCTL, cSCTL formulae are conjunctions of simpler kinds of assertions. The cSCTL assertions for process $i$ are composed from a set $\Sigma_i$ of atomic propositions. $\Sigma_i$ has one proposition for each permitted state that process $i$ is allowed to be in; intuitively, this proposition holds when process $i$ is in that state. Hence, the propositions of $\Sigma_i$ may justifiably be assumed to be exhaustive and mutually exclusive.

The specification on process $i$ is a conjunction of assertions 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.

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 must visit some state of $\gamma$ in the future, irrespective of the execution sequence followed by the program. As before, we shall call subformulae of the kind $AF(\vee\gamma)$ *eventualities,* and assertions of this kind *leads-to* assertions.

C. $AG(Q \Rightarrow AX_i(\vee\alpha) \wedge (\bigwedge EX_i\beta))$, where $Q \in \Sigma_i$, $\beta \subseteq \alpha \subseteq \Sigma_i$, and $EX_i\beta \stackrel{\text{def}}{=} \{EX_i\ b \mid b \in \beta\}$. Exactly one assertion of this kind is required for each antecedent $Q \in \Sigma_i$. Again, we shall call these kinds of assertions *successor* assertions and subformulae of the form $AX_i(\vee\alpha)$ *universal conjuncts.* We call the elements of $\alpha$ the *permitted successors* of $P$. Also, we shall call a subformula of the form $EX_i\ b$ where $b \in \beta_i$ an *existential conjunct.*

Informally, $AX_i f$ means that if the first transition along an execution sequence is a transition by process $i$, then $f$ holds at the successor state of that execution sequence. Similarly, $EX_i f$ means that there is an execution sequence such that the first transition along the sequence is a transition by process $i$ and such that $f$ holds at the successor state of the execution sequence. So, this assertion specifies the successors of process $i$ when it is in state $Q$: its successors are permitted to be only those contained in $\alpha$, and, each time the process is in state $Q$, it must be possible for it to move to any state of $\beta$, irrespective of the states of other processes. Note that the set $\beta$ and, hence, $EX_i\beta$, could be empty.

We note that the flowgraph of a process can be constructed from its successor assertions. Formally, this is a directed graph whose nodes correspond to the process' states, and in which there is an edge from $P$ to $Q$ iff $Q$ is a permitted successor of $P$. (Note that the flowgraph only captures the information in the universal conjuncts of successor assertions.) In the sequel, we assume that each process state is reachable in the flowgraph from the initial state of the process.

Note that the syntax of cSCTL formulae is very like that of SCTL, the major difference being that the universal and existential conjuncts in successor assertions are relativized to an individual process. (The other variations are less significant: for example, we could permit disjunctions of propositions in the initial assertion and in the existential conjuncts of successor assertions, and we could add ensures assertions.) As before, we shall also represent a cSCTL formula by a set of its conjuncts.

In addition to the restriction on the kinds of assertions that may be used to describe processes, we require that each process specification satisfy the *compositional euclidean syntactic constraint:* for every leads-to assertion $AG(P \Rightarrow AF\,(\vee\gamma))$ in the process' specification and for every permitted successor, $Q$, of $P$ which is not in $\gamma$, the process specification must also have the conjunct $AG(Q \Rightarrow AF\,(\vee\gamma))$.

*Remark* **3** Effectively, the compositional euclidean syntactic constraint requires processes to be *history-free*. Consider a path, $p$, in the process' flowgraph from its initial state to the state $R$. We shall say that the eventuality $AF(\vee\gamma)$ is *incurred along $p$ but remains unfulfilled at $R$* if there is a state $Q$ on the path such that the assertion $AG(Q \Rightarrow AF(\vee\gamma))$ is an assertion in the process' definition and none of the states between (the last occurrence of) $Q$ and $R$ on the path is in $\gamma$. Now, a process is *history-free* iff for every state $R$ of the process and every path $p$ in the process' flowgraph from its initial state to $R$, the set of eventualities which are incurred along $p$ but which remain unfulfilled at $R$ is the same, irrespective of the path $p$. Thus, no matter what path we follow from the initial state to $R$, the set of eventualities that are incurred along the path but which remain unfulfilled at $R$ is precisely the set of unfulfilled eventualities at $R$.

In a sense, the notion of history-freedom for processes of reactive programs (cf. [Pn77]) ties in to the concept of a state of classical transformational (or functional) programs. The current state (i.e., the values of all variables and location counters) of a transformational program completely determines its set of possible behaviours. The behaviours of reactive programs, on the other hand, are infinite, and, therefore, are additionally restricted by the set of eventualities its processes are obliged to fulfill. In a history-free reactive process, this set of eventualities is dependent solely on the process' current state rather than on the path followed to it. Thus, we can ensure that the set of behaviours of a reactive program that is composed from history-free processes is completely determined by the state it is in, just as it is for transformational programs.

It is easily seen that if a process specification satisfies the compositional euclidean syntactic constraint, it is history-free. Conversely, it can be shown that, if every state of a history-free process is reachable from its initial state, the process has an equivalent specification that satisfies the compositional euclidean syntactic constraint. Consequently, it is possible to test if a process specification is history-free in time quadratic in its length. □

**Example 3** Consider the classical two process mutual exclusion problem as specified in [EC82]. The flowgraphs of the two 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 assertions defining process $i$ (where $i$ is either 1 or 2) 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. $AG(T_i \Rightarrow AF\ C_i)$. For all execution sequences of the program, process $i$ will eventually get to its critical region if it visits its trying region.

Note that both the process specifications satisfy the compositional euclidean syntactic constraint. It follows from the remark above that both processes are history-free. To see this directly, note that the only eventuality for process $i$ is $AF\ C_i$. No matter what path process $i$ follows from its initial state, $N_i$, to $T_i$, this eventuality is incurred along the path but remains unfulfilled along it. As for the state $N_i$, there are two kinds of paths from the initial state to it: one, the empty path along which the eventuality is not incurred at all, and the other in which the path goes through $T_i$ and then through $C_i$ before coming to $N_i$. In these paths, the eventuality is incurred but is also fulfilled before the process returns to $N_i$. The state $C_i$ can similarly be shown to have no unfulfilled eventualities.

We note that if the leads-to assertion $AG(C_1 \Rightarrow AF\ T_1)$ were added to the definition of process 1, it would not be history-free (and its process specification would not satisfy the compositional euclidean syntactic constraint). For now, the empty path from $N_1$ to $N_1$ has no eventuality incurred or unfulfilled at $N_1$. But every other path from $N_1$ to $N_1$ incurs the eventuality $AF\ T_1$ and the eventuality remains unfulfilled at $N_1$. □

### 4.2 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 be in their critical regions at the same time. So cSCTL also allows assertions of the following kind:

D. $AG(\neg(Q_1 \wedge Q_2 \wedge \ldots \wedge Q_m))$, where each $Q_i \in \bigcup_{j=1}^{k} \Sigma_j$ and at most one of the $Q_i$'s is in any one $\Sigma_j$. (This restriction is reasonable as the atomic propositions of each process are mutually exclusive.)

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

### 4.3 Translating Compositional Assertions

We now show how to translate a compositional specification in cSCTL into an SCTL formula that globally describes the behaviour of the concurrent program, assuming that the process steps are interleaved. The proof of the following theorem shows how this may be done mechanically.

**Theorem 3** *Consider a concurrent program with a fixed number, say $k$, of processes. For $1 \leq i \leq k$, let $\Sigma_i$ be the set of atomic propositions of process $i$ and $\mathcal{F}_i$ be a set of cSCTL assertions of types* A–C (*in Section* 4.1) *above such that $\wedge\mathcal{F}_i$, the specification of process $i$, satisfies the compositional euclidean syntactic constraint. Let $\mathcal{S}$ be the set of cSCTL system assertions* (*i.e., assertions of type* D *in Section* 4.2) *that the concurrent program must satisfy. Then, we can compute an SCTL formula $\wedge\mathcal{G}$ which globally describes the behaviour of the concurrent program using interleaving of process transitions to model parallelism. Moreover, if $L$ is the sum of the lengths of all the process and system assertions, $\wedge\mathcal{G}$ can be computed in time $O(L^{2k})$.*

*Proof:* We assume that the alphabets $\Sigma_i$ are mutually disjoint. We show how to construct the set $\mathcal{G}$ from the $\mathcal{F}_i$ and $\mathcal{S}$, and that $\wedge\mathcal{G}$ is an SCTL formula.

The alphabet of the assertions of $\mathcal{G}$ is $\Sigma = \{Q_1 \wedge Q_2 \wedge \ldots \wedge Q_k \mid Q_i \in \Sigma_i\}$. Note that, because each $\Sigma_i$ is mutually exclusive, so is $\Sigma$, and it is exhaustive because at each state of the concurrent program, each constituent process is in some state and because each $\Sigma_i$ is exhaustive. Also, $|\Sigma| = \prod_{i=1}^{k} |\Sigma_i| = O(L^k)$. We measure the length (or size) of a formula as a string (or syntax diagram) over the alphabet $\Sigma$.

We now define some notation. First, we write $Q_1Q_2\ldots Q_k$ instead of $Q_1 \wedge Q_2 \wedge \ldots \wedge Q_k$, and we let $\Gamma_i = \{Q_1Q_2\ldots Q_{i-1}Q_{i+1}\ldots Q_k \mid Q_j \in \Sigma_j\}$; the elements of $\Gamma_i$ specify the states of all processes except that of process $i$. If $P \in \Sigma_i$ and $Q = Q_1Q_2\ldots Q_{i-1}Q_{i+1}\ldots Q_k \in \Gamma_i$, we write $PQ$ to denote the global state $Q_1Q_2\ldots Q_{i-1}PQ_{i+1}\ldots Q_k$. We also let $P\Gamma_i$ (or $\Gamma_iP$) denote the set $\{PQ \mid Q \in \Gamma_i\}$. If $\alpha \subseteq \Sigma_i$ and, for each $j, 1 \leq j \leq k$, $Q_j \in \Sigma_j$, we write $\alpha\Gamma_i$ to denote $\bigcup_{P\in\alpha} P\Gamma_i$ (so $\alpha\Gamma_i = \emptyset$ if $\alpha = \emptyset$), and $Q_1Q_2\ldots Q_{i-1}\alpha Q_{i+1}\ldots Q_k$ to denote $\bigcup_{P\in\alpha}\{Q_1Q_2\ldots Q_{i-1}PQ_{i+1}\ldots Q_k\}$.

The following rules add assertions to (an initially empty) $\mathcal{G}$:

E. First, we add the assertion $Q_1Q_2\ldots Q_k$, where $Q_i$ is the initial state of process $i$, to $\mathcal{G}$. This is the initial state of the concurrent program.

F. Next, we compute the invariance assertion of $\mathcal{G}$. This is done using the system assertions. Initially, let the set $\delta$ equal $\Sigma$. Consider the following assertion in $\mathcal{S}$: $AG(\neg(Q_{i_1} \wedge Q_{i_2} \wedge \ldots \wedge Q_{i_m}))$, where each $Q_{i_j} \in \Sigma_j$ and $i_j \neq i_l$. Let $I = \{i_1, i_2, \ldots, i_m\}$. This assertion forbids a program state in which, for each $i_l \in I$, process $i_l$ is in state $Q_{i_l}$. We delete all such program states from $\delta$, and repeat this for each assertion in $\mathcal{S}$. Finally, we add the invariance assertion $AG(\vee\delta)$ to $\mathcal{G}$.

G. The third step is to compute the successor assertions of $\mathcal{G}$. We do this in turn for each $Q \in \Sigma$. Let $Q = Q_1Q_2\ldots Q_k$. Let the successor assertion of process $i$ when it is in state $Q_i$ be $AG(Q_i \Rightarrow AX_i(\vee\alpha_i) \wedge (\bigwedge EX_i\beta_i))$. The successor assertion for $Q$ is $AG\big(Q \Rightarrow AX(\vee\gamma) \wedge (\bigwedge EX\delta)\big)$, where, as before, $EX\ \delta \stackrel{\text{def}}{=} \{EX\ d \mid d \in \delta\}$, and $\gamma$ and $\delta$ (both subsets of $\Sigma$) are obtained as follows:

   - $\gamma$ has all successors of $Q$ that are obtained from $Q$ by allowing one of the $k$ processes to move at a time (recall that we use interleaving of process transitions to model parallelism). So $\gamma = \bigcup_{i=1}^{k} Q_1Q_2\ldots Q_{i-1}\alpha_iQ_{i+1}\ldots Q_k$; note that each state in $Q_1Q_2\ldots Q_{i-1}\alpha_iQ_{i+1}\ldots Q_k$ is obtained from $Q$ by changing the state of process $i$ to one of the permitted successors of $Q_i$ and leaving the states of the other processes unchanged.

- The set $\delta$ accounts for the existential conjuncts in the successor assertions of the processes. The successor assertion in $\mathcal{F}_i$ with $Q_i$ as its antecedent has the conjunct $\bigwedge(EX_i\ \beta_i)$ in its consequent. This conjunct requires that process $i$ be able to visit each state of $\beta_i$ each time it is in $Q_i$ irrespective of the states of the other processes of the concurrent program. So, $\delta = \bigcup_{i=1}^{k} Q_1Q_2 \ldots Q_{i-1}\beta_i Q_{i+1} \ldots Q_k$.

*Remark* **4** The sum of the lengths of all the successor assertions in the SCTL formula $\wedge\mathcal{G}$ is $O(L^k)$. To see this, we sum the lengths of the antecedents and the consequents of the successor assertions separately, and show that each of these sums is $O(L^k)$ long. Note that there are $|\Sigma|$ (i.e., $O(L^k)$) successor assertions in $\mathcal{G}$; hence, the sum of the antecedents of the successor assertions is $O(L^k)$. Next, we compute the sum of the lengths of the consequents of the successor assertions. Consider the successor assertion in $\mathcal{G}$ which has $Q = Q_1Q_2 \ldots Q_k$ as its antecedent. We note that each disjunct in the universal conjunct of this assertion and each existential conjunct in the consequent of this assertion is added because of a corresponding term in the consequent of the successor assertion of one of the propositions $Q_1, Q_2, \ldots, Q_k$. Thus, the length of the consequent of the successor assertions for $Q$ is the sum of the lengths of the consequents of the successor assertions of the $Q_i$, $1 \le i \le k$. Let $\widehat{\Sigma} = \bigcup_{i=1}^{k} \Sigma_i$. Then, the sum of the lengths of the consequents of the global successor assertions is $\sum_{\widehat{Q}\in\widehat{\Sigma}} n_{\widehat{Q}} \times l_{\widehat{Q}}$ where $n_{\widehat{Q}}$ is the number of propositions in $\Sigma$ that contain the proposition $\widehat{Q}$ and $l_{\widehat{Q}}$ is the length of the consequent of the successor assertion of which $\widehat{Q}$ is the antecedent. Consider any proposition $\widehat{Q} \in \widehat{\Sigma}$ and let $\widehat{Q}$ be in $\Sigma_i$. Note that $n_{\widehat{Q}} \le (|\Sigma|/|\Sigma_i|) \le L^{k-1}$. Thus, the sum of the lengths of the consequents of the successor assertions is at most $L^{k-1} \times (\sum_{\widehat{Q}\in\widehat{\Sigma}} l_{\widehat{Q}})$. Now, note that $(\sum_{\widehat{Q}\in\widehat{\Sigma}} l_{\widehat{Q}}) \le L$, So, the sum of the lengths of the consequents of the successor assertions is $O(L^k)$. □

H. Finally, we compute the leads-to assertions of $\mathcal{G}$. We consider each process in turn. For each $i$, $1 \le i \le k$, leads-to assertion $AG(Q_i \Rightarrow AF(\vee\gamma_i))$ in $\mathcal{F}_i$ and element $R$ in $\Gamma_i$, we add the assertion $AG(Q_iR \Rightarrow AF(\vee\gamma_i\Gamma_i))$ to $\mathcal{G}$. Thus, each time the concurrent program is in a state in which process $i$ is in state $Q_i$ (and, hence, the eventuality $AF(\vee\gamma_i)$ has to be fulfilled), this assertion requires the program to eventually get to a state in which process $i$ is in a state that is an element of $\gamma_i$. This is justified because the only way of fulfilling the eventuality $AF(\vee\gamma_i)$ on process $i$ is by making process $i$ move to a state in $\gamma_i$.

*Remark* **5** Note that, for any $i$, $1 \le i \le k$, $|\Gamma_i| = (|\Sigma|/|\Sigma_i|) = O(L^{k-1})$. Using this, it can be shown by an argument similar to the one in Remark 4 above, that: $(a)$ the sum of the lengths of all the eventualities in $\mathcal{G}$ is $O(L^k)$ and $(b)$ the number of leads-to assertions in $\mathcal{G}$ is $O(L^k)$. To see $(a)$, note that each global eventuality of the form $AF(\vee\gamma_i\Gamma_i)$ is derived from the local eventuality $AF(\vee\gamma_i)$. So the length of all global eventualities is at most $L^{k-1}$ times the total length of all local eventualities (which is at most $L$), and is thus at most $L^k$. To see $(b)$, note that each leads-to assertion in $\mathcal{G}$ is of the form $AG(Q_iR \Rightarrow AF(\vee\gamma_i\Gamma_i))$, where $R \in \Gamma_i$ and $AG(Q_i \Rightarrow AF(\vee\gamma_i))$ is a local leads-to assertion. Since there are at most $L$ local leads-to assertions and $L^{k-1}$ global leads-to assertions (one for each $R$) for each local leads-to assertion, there are at most $L \times L^{k-1} = L^k$ global leads-to assertions in all.

As for the total length of all the leads-to assertions in $\mathcal{G}$, it follows from $(b)$ that the sum of the lengths of the antecedents of the leads-to assertions is $O(L^k)$. However, each eventuality appears in the consequent of several leads-to assertions. Assuming, in the worst case, that every eventuality appeared in the consequent of a leads-to assertion in $\mathcal{G}$ for every proposition in $\Sigma$, the sum of the lengths of the consequents of the leads-to assertions in $\mathcal{G}$ is $O(L^{2k})$. Thus, the length of the leads-to assertions in $\mathcal{G}$ is $O(L^{2k})$. We note, however, that we could consolidate common subformulae (the eventualities in the leads-to assertions) so that the total length (more precisely, the total size of all the syntax diagrams, which after consolidation of common subformulae would be dags rather than trees) of all the leads-to assertions in $\mathcal{G}$ would be $O(L^k)$. □

Note that $\wedge\mathcal{G}$ is an NESCTL formula. To see that it satisfies the euclidean syntactic constraint, consider the leads-to assertion $AG(Q_iR \Rightarrow AF(\vee\gamma_i\Gamma_i))$ in $\mathcal{G}$ and let $Q_iR \notin \gamma_i\Gamma_i$, i.e., $Q_i \notin \gamma_i$. Consider any permitted successor, $P$ of $Q_iR$. From Rule G, $P$ is either in $Q_i\Gamma_i$ or is of the form $Q_i'R$. In the former case, Rule H adds the assertion $AG(P \Rightarrow AF(\vee\gamma_i\Gamma_i))$ to $\mathcal{G}$. If $P = Q_i'R$, then it follows from Rule G that $Q_i'$ is a permitted successor of $Q_i$. As $\mathcal{F}_i$ satisfies the compositional euclidean syntactic constraint, $AG(Q_i' \Rightarrow AF(\vee\gamma_i))$ is in $\mathcal{F}_i$ and so, Rule H adds $AG(P \Rightarrow AF(\vee\gamma_i\Gamma_i))$ to $\mathcal{G}$, as required.

Thus, $\wedge\mathcal{G}$ is an SCTL formula. From the explanations accompanying the rules above, it is easy to see that a concurrent program satisfies all the assertions in $\mathcal{G}$ iff its constituent processes satisfy their respective process assertions and the system assertions in $\mathcal{S}$, i.e., $\wedge\mathcal{G}$ is equivalent to the conjunction of the process and the system assertions. We also note that $\wedge\mathcal{G}$ is in canonical form and that every proposition in $\Sigma$ has a successor assertion in $\mathcal{G}$.

As for the complexity of the procedure, we note that the number of propositions, $|\Sigma|$, in $\mathcal{G}$ is no more than $L^k$. Now, the invariance assertion is at most $O(|\Sigma|)$ long. As shown in Remark 4 above, the sum of the lengths of all the successor assertions in $\mathcal{G}$ is $O(L^k)$, and Remark 5 shows that the sum of the lengths of all the leads-to assertions in $\mathcal{G}$ is $O(L^{2k})$. Thus, $\mathcal{G}$ can be computed in time $O(L^{2k})$. (Note that if we were to consolidate subformulae as mentioned in Remark 5, the length of $\wedge\mathcal{G}$ and the time taken to compute it would be no more than $O(L^k)$.) $\square$

**Example 4** The set $\Sigma$ of global atomic propositions for the two process mutual exclusion problem (Example 4) 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; note that it is in canonical form. The initial state assertion, $N_1N_2$, is got from the initial state assertions of the two processes, and the invariance assertion is obtained by dropping $C_1C_2$ from $\Sigma$ as the only system assertion is $AG(\neg(C_1 \wedge C_2))$.

Each global successor assertion can be written by inspecting the permitted successors of each of the processes. Consider the successor assertion for $N_1T_2$ for example. Process 1 is in state $N_1$ and process 2 is in $T_2$. The only permitted successor of process 1 is $T_1$ and that of process 2 is $C_2$. So the universal conjunct in the consequent of this assertion is $AX(T_1T_2 \vee N_1C_2)$. Note that interleaving of process steps is implicitly assumed: when one process moves, the other process does not change state. Next, we consider the existential conjuncts in the consequent. Now, the successor assertion for $N_1$ has an existential conjunct, viz., $EX_1\ T_1$. So, there is an existential conjunct $EX\ T_1T_2$ in the consequent of the successor assertion of $N_1T_2$. What this ensures is that, whenever the concurrent program is in the state $N_1T_2$, it will always be possible for it to transit to the state $T_1T_2$, i.e., process 1 will always be able to move, as desired. The successor assertion for $T_2$ has no existential conjuncts, and so the successor assertion for $N_1T_2$ has no more existential conjuncts.

Finally, there are two process eventualities, and, hence, two eventualities in the SCTL formula. The eventuality of process 1, for example, is $AF\ C_1$. The corresponding eventuality in the SCTL formula is $AF(C_1N_2 \vee C_1T_2 \vee C_1C_2)$, i.e., when process 1 fulfills its eventuality, process 2 could be in any state. The leads-to assertions in the SCTL formula are now obtained in the obvious fashion. For example, consider the leads-to assertions for the eventuality of process 1. We require that 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. There are three atomic propositions which represent the fact that process 1 could be in state $T_1$. These are $T_1N_2$, $T_1T_2$ and $T_1C_2$. Hence, there are three leads-to assertions for the first SCTL eventuality, one for each of these three propositions. We note that each of the permitted successors of these three propositions is either one of the others or a proposition which represents process 1 being in state $C_1$. The eventuality of process 1 is not pending at these latter propositions, and hence no other leads-to assertions are required in the formula (with this eventuality as the consequent) to satisfy the euclidean syntactic constraint. □

We may remark that we could expand the syntax of the cSCTL assertions in Section 4.1 without losing the ability to generate an SCTL formula which describes the entire program and which satisfies all the complexity measures in Theorem 3. For example, the initial assertion of the processes could allow a disjunction of atomic propositions and each of the $EX_i$ conjuncts in the process successor assertions could have a disjunction of atomic propositions. We could also permit ensures assertions as in SCTL.

### 4.4 Application of the Satisfiability Problem to Program Synthesis

Given a set of cSCTL process specifications for a $k$-process concurrent system, the synthesis problem is to construct the flowgraphs or ***synchronization skeletons*** of each of the processes, and to label their edges with guards (boolean predicates on the process' states and some auxiliary variables that are shared by all processes), which indicate when the process may make the transition, and actions, which update the auxiliary variables when the process makes the transition. A process is allowed to execute a transition only when the corresponding guard is true, and when it does execute a transition, it performs the associated action atomically. A valid execution sequence of the program is a sequence of transitions of processes such that each transition is made only when its corresponding guard is true. The synthesized guards should be such that all the program specifications are true of all valid execution sequences.

In this section, we adapt the synthesis method of [EC82] to SCTL, and illustrate it with the two process mutual exclusion problem that is presented in Example 4. The first step is to use the algorithm in the proof of Theorem 3 to translate the cSCTL process specifications to an SCTL formula, $\wedge\mathcal{G}$. For the mutual exclusion problem, this has been done in Example 5 (see Fig. 6). The next step is to test $\mathcal{G}$ for satisfiability, so the initial next time tableau is constructed from $\wedge\mathcal{G}$ and pruned.

**Example 5** The global next time tableau constructed out of the assertions in Fig. 6 is shown in Fig. 7. As before, AND nodes are shown as boxes and OR nodes as circles. The AND nodes $N_1T_2$, $T_1N_2$, $N_1C_2$, and $C_1N_2$ are shown twice only for clarity.

The pruning procedure deletes the AND node $C_1C_2$ (because it is absent from the invariance assertion; assertion 2 of Fig. 6), but no other node. A DAG, $D$, fulfilling the eventuality, $H_1 = AF(C_1N_2 \vee C_1T_2 \vee C_1C_2)$ rooted at the AND node $T_1N_2$ is shown in the figure, enclosed within dotted lines. A DAG for $H_1$ rooted at $T_1T_2$ can be obtained from $D$ by retaining only the AND nodes $T_1T_2$ and $C_1T_2$ and the the OR node successor of $T_1T_2$. A DAG for $H_1$ rooted at $T_1C_2$ can be obtained by making $T_1N_2$ the root and directing the edges of its two OR node successors to the root of $D$. Also, DAGs for the eventuality $H_2 = AF(N_1C_2 \vee T_1C_2 \vee C_1C_2)$ can be similarly constructed.

The formula is satisfiable as the initial AND node, $N_1N_2$, is not deleted. □

We note that if $\wedge\mathcal{G}$ is not satisfiable, the given process specifications are ***inconsistent,*** i.e., there is no concurrent program that realizes these specifications, or alternatively, there is no valid execution sequence that realizes the specification no matter what guards and actions are used to label the process transitions. If $\mathcal{G}$ is satisfiable, we can automatically generate guards and actions that yield valid execution sequences. The first step to do that is to construct a model for $\wedge\mathcal{G}$. This can be done as described in the proof of Theorem 1, by splicing together the DAGs fulfilling eventualities in the pruned tableau. For conciseness, however, we shall use the hand constructed model shown in Fig. 8

to explain the rest of the synthesis method. (Details of how this model can be obtained systematically from the DAGs in the pruned tableau may be found in [Sr91].)

The global model in Fig. 8 is the reachability graph of the concurrent program that we intend to synthesize. The final task of our synthesis procedure is to factor out the synchronization skeletons of the processes from the global model, i.e., to construct the synchronization skeletons and label their edges with appropriate guards and actions. The general procedure is as follows (see also [EC82]). First, the edges on the model are labelled with process numbers: if vertex $P$ is followed by $P'$ and process $i$ made the transition in going from $P$ to $P'$, the edge between $P$ and $P'$ is labelled with $i$. Next, if the model has $n > 1$ distinct vertices for the atomic proposition $P$, all edges coming in to $j$th such vertex (for $1 \leq j \leq n$) are labelled with an assignment $x_P := j$ where $x_P$ is an auxiliary variable, and the vertex itself is labelled with $x_P = j$ to distinguish it from the other vertices labelled with the proposition $P$. Applying this procedure to the model in Fig. 8, we obtain the labelled model in Fig. 9.

Finally, the synchronization skeletons for the processes may be obtained as follows. Let process $i$ have $m$ nodes $R_i^1, \ldots, R_i^m$, where each $R_i^j \in \Sigma_i$, in its synchronization skeleton. The initial synchronization skeleton is a graph with $m$ nodes and no edges. Let there be an edge in the global model from $P$ to $P'$ labelled with the process number $i$ such that the state of process $i$ in $P$ is $R_i^j$ and in $P'$ is $R_i^l$ and the state of process $t$ ($t \neq i$) in both these global states is $Q_t$. Add an edge from $R_i^j$ to $R_i^l$ corresponding to this global edge. Suppose state $P$ occurs only once in the model and so has no auxiliary variable labelling it. Then $Q_1 \wedge \ldots \wedge Q_{i-1} \wedge Q_{i+1} \wedge \ldots \wedge Q_k$ is the guard on the edge just added to the synchronization skeleton. Otherwise, if $x_P = a$ is the auxiliary variable in state $P$, $(Q_1 \wedge \ldots \wedge Q_{i-1} \wedge Q_{i+1} \wedge \ldots \wedge Q_k) \wedge (x_P = a)$ is the guard. In either case, if the edge from $P$ to $P'$ in the global model is labelled with $x_{P'} := b$, then $x_{P'} := b$ is the action associated with this edge; otherwise no action is associated with this edge.

Applying the above procedure to the labelled global model in Fig. 9, we obtain the synchronization skeletons shown in Fig. 10. The label $B? \rightarrow A$ on an edge means that its guard is $B$ and the action performed on the transition is $A$; the label $B?$ means that its guard is $B$ and that it has no associated action.

We now examine the complexity of determining whether we can synthesize a solution for a given set of process specifications which are expressed in cSCTL, i.e., whether the process specifications can be realized by a concurrent program or whether they are inconsistent. In Remark 2, we noted that given an SCTL set $\mathcal{F}$ in canonical form which has a successor assertion for every proposition in $\Sigma$, the size of its initial next time tableau, $|T_0|$, is linear in the sum of the lengths of the successor assertions in $\mathcal{F}$. In the proof of Theorem 1, we saw that the pruning procedure could be implemented to run in time $O(|T_0| + |T_0|^2 + |T_0| \times K)$, where $K$ is the sum of the lengths of all the assurances in $\mathcal{F}$, the lengths of all the eventualities in $\mathcal{F}$ and the number of leads-to assertions in $\mathcal{F}$. The proof of Theorem 3 shows that the SCTL formula, $\wedge\mathcal{G}$, derived from process and system assertions has a successor assertion for every proposition in $\Sigma$ and has no ensures assertions. Moreover, Remark 4 shows that the sum of the lengths of the successor assertions in $\mathcal{G}$ is $O(L^k)$ and Remark 5 shows that the sum of the lengths of all the eventualities in $\mathcal{G}$ is $O(L^k)$ and that the number of leads-to assertions in $\mathcal{G}$ is $O(L^k)$. Thus, we may conclude that, both, $|T_0|$ and $K$ are $O(L^k)$. Hence, the pruning procedure can determine if $\wedge\mathcal{G}$ is satisfiable in time $O(L^{2k})$.

From the above, it follows that the fragment of CTL originally used in [EC82] to determine if one can synthesize a solution to synchronization problems involving 2 processes is decidable in polynomial time $O(n^4)$, where $n$ is the sum of the lengths of the process and system assertions. In contrast, the best upper bound provable using full CTL as in [EC82] is $O(2^{2n})$. In general, for any fixed number $k$ of processes, it now follows that synthesis can be done in time $O(n^{2k})$, compared to the best provable upper bound of $O(2^{kn})$ in [EC82]. Thus, using SCTL reduces the complexity from exponential to polynomial, for a fixed number $k$ of processes, and the synthesis method may be feasible for applications when $k$ is small.

## 5 Conclusion

We have shown that by suitably restricitng the syntax of the logic CTL we can get a simplified logic SCTL which is still useful in applications and yet has its satisfiability problem decidable in polynomial time. We also define a compositional version cSCTL, which corresponds to the fragment of CTL originally used for synthesis in [EC82], and give a translation of it into SCTL. This shows that the temporal synthesis method of [EC82] can be made to run in polynomial time.

We remark that certain liberalizations of the SCTL syntax are permissible while still retaining polynomial time satisfiability. In particular, if we allow each proposition in SCTL to be replaced by a disjunction of propositons (where the propositions are still disjoint and exhaustive), the logic is still decidable in polynomial time. However, additional syntactic relaxations appear very problematic if we wish to preserve polynomial time satisfiability. We refer the reader to [EES90] for an in-depth discussion of these issues.

Finally, we remark that the state explosion problem, associated with translating a multi-process cSCTL specification into an equivalent global SCTL specification, remains an important unsolved issue. For a possible line of attack that appears promising in the context of synthesis, we refer the reader to [AE89]. Related work in the context of model checking may be found in [BCG87], [CG87], [SG87] and [SW89].

## Acknowledgement & Historical Remarks

This paper had been germinating for a number of years. 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, *Proc. 16th Annual ACM Symp. on Principles of Programming Languages,* Austin, pp. 191–201, 1989.

[BMP81] Ben-Ari, M., Z. Manna, A. Pnueli, The Temporal Logic of Branching Time, *Proc. 8th Annual ACM Symp. on Principles of Programming Languages,* Williamsburg, pp. 164–176, 1981.

[BCG87] Browne, M.C., E.M. Clarke, O. Grumberg, Characterizing Kripke Structures in Temporal Logic, Tech Report CMU–CS–87–104, January 1987; to appear in *Theoretical Computer Science*.

[Ch80] Chellas, B.F., Modal Logic—An Introduction, Cambridge University Press, 1980.

[CGB86] Clarke, E.M., O. Grumberg, M.C. Browne, Reasoning About Networks With Many Identical Finite-State Processes, *Proc. of the 5th Annual ACM Symp. on Principles of Distributed Computing*, Vancouver, pp. 240–348, 1986.

[CG87] Clarke, E.M., O. Grumberg, Avoiding the State Explosion Problem in Temporal Logic Model Checking Algorithms, Tech Report CMU–CS–87–137, July 1987; *also appears in Proc. of the 6th Annual ACM Symp. on Principles of Distributed Computing*, 1987.

[CKS81] Chandra, A.K., D.C. Kozen, L.J. Stockmeyer, Alternation, *Journal of the ACM*, **vol.** 28, pp. 114–133, 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.

[Em90] Emerson, E.A., Temporal and Modal Logic, in the *Handbook of Theoretical Computer Science*, **vol.** B: *Formal Models and Semantics*, J. van Leeuwen, editor, Elsevier Science Publishers, pp. 995–1072, 1990.

[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.

[EES90] Emerson, E.A., W.M. Evangelist, J. Srinivasan, On the Limits of Efficient Temporal Decidability, *Proc. of the 5th Annual IEEE Symp. on Logic in Computer Science*, Philadelphia, pp. 464–475, 1990.

[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.

[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.

[ESS89] Emerson, E.A., T.H. Sadler, J. Srinivasan, Efficient Temporal Reasoning, *Proc. 16th Annual ACM Symp. on Principles of Programming Languages,* Austin, pp. 166–178, 1989.

[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.

[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.

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

[PR89] Pnueli, A., R. Rosner, On the Synthesis of a Reactive Module, *Proc. 16th Annual ACM Symp. on Principles of Programming Languages,* Austin, pp. 179–190, 1989.

[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.

[SG87] Sistla, A.P., S.M. German, Reasoning about Systems with Many Processes, LICS87 (to appear *JACM*)

[Sr91] Srinivasan, J., Reasoning About Concurrent Programs With Simplified Temporal Logics, Ph. D. Thesis, *manuscript in preparation*, Univ. of Texas at Austin, 1991.

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

[St91] Stirling, C., Modal and Temporal Logic, *manuscript,* 1991; to appear in the *Handbook of Logic in Computer Science,* D. Gabbay, editor.

[SW89] Stirling, C., D.J. Walker, Local Model Checking in the Modal Mu-calculus, *Proc. of the 1989 Int'l Joint Conf. on Theory and Practice of Software Development,* LNCS#351–352, Springer–Verlag, 1989.

[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.

[Zu86] Zuck, L. Past Temporal Logic, Ph.D. Thesis, Weizmann Institute, 1986.

$$\Sigma = \{P, Q, R, S, T, V, W\}$$

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. $AG(S \Rightarrow AX(P \vee W \vee V) \wedge EX\ P)$
7. $AG(T \Rightarrow AX\ V)$
8. $AG(V \Rightarrow AX\ R)$
9. $AG(W \Rightarrow AX\ T)$
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: The SCTL assertions for Example 1

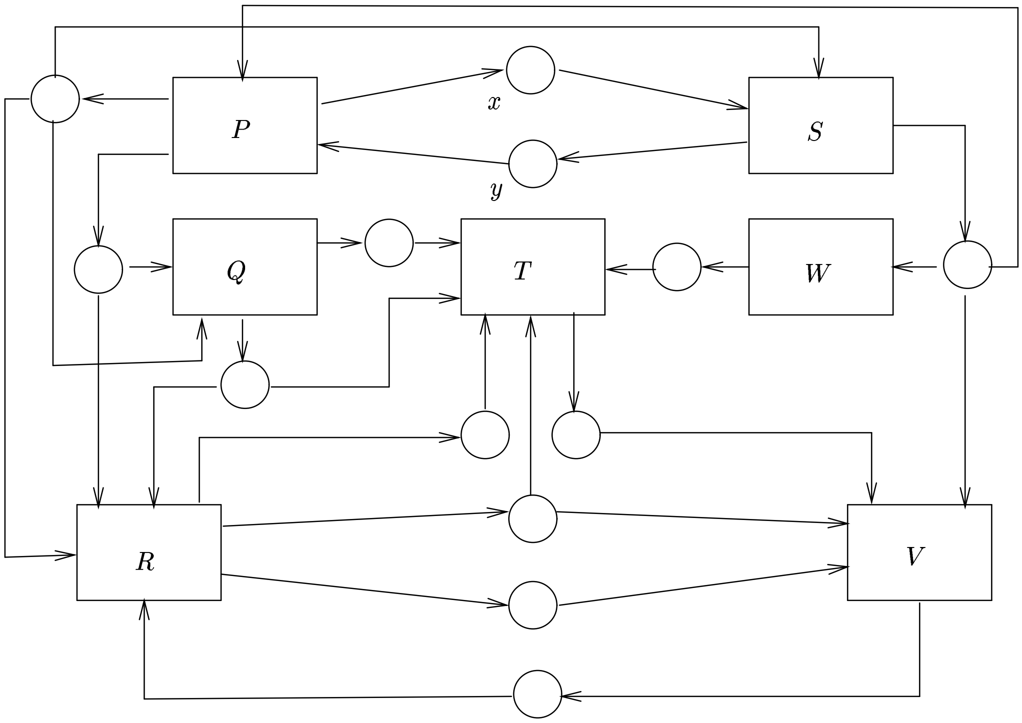


Figure 2: The initial tableau for the SCTL assertions in Example 1

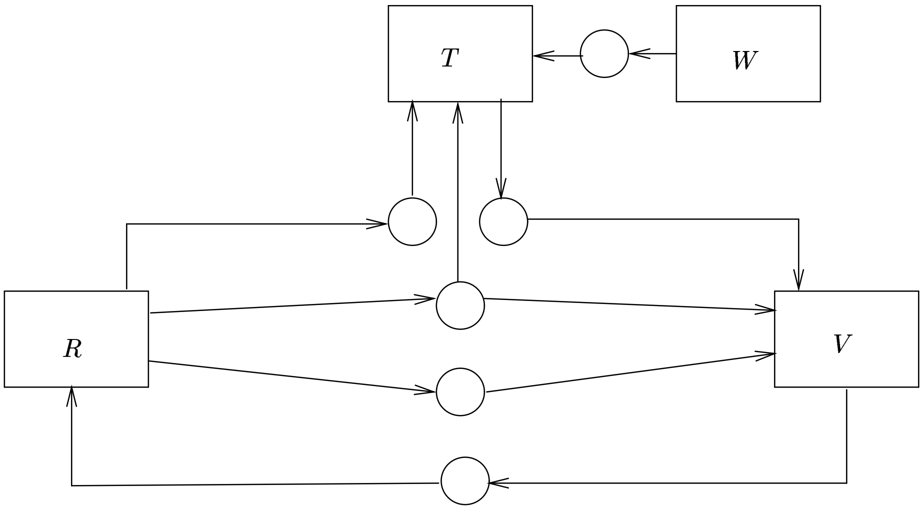


Figure 3: The pruned tableau for the SCTL assertions in Example 1

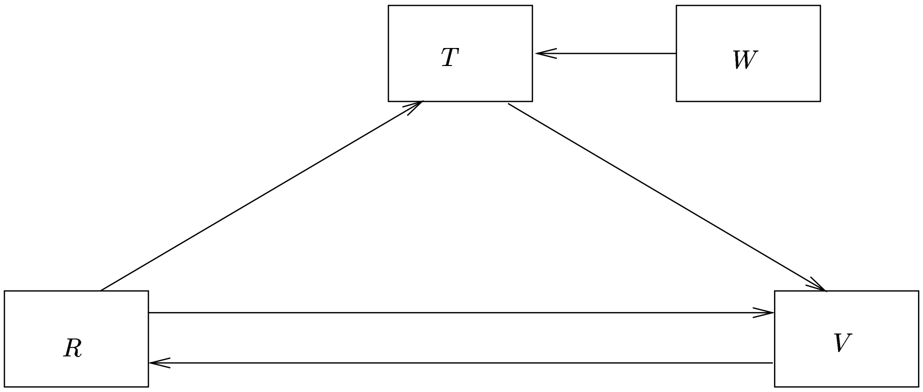


Figure 4: A model for the SCTL assertions in Example 1

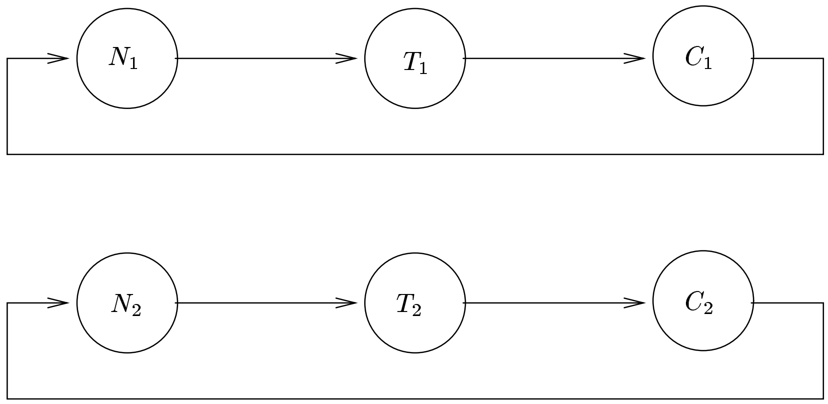


Figure 5: Flowgraphs of mutual exclusion processes (Examples 3 and 4)

$$\Sigma = \{N_1N_2, N_1T_2, N_1C_2, T_1N_2, T_1T_2, T_1C_2, C_1N_2, C_1T_2, C_1C_2\}$$

1. The program's initial state: $N_1N_2$.

2. The program may be in any state other than $C_1C_2$:

   $AG(\vee\{N_1N_2, N_1T_2, N_1C_2, T_1N_2, T_1T_2, T_1C_2, C_1N_2, C_1T_2\})$

3. The successor assertions, one for each proposition in $\Sigma$:

   3*a*. $AG(N_1N_2 \Rightarrow AX(T_1N_2 \vee N_1T_2) \wedge EX\ T_1N_2 \wedge EX\ N_1T_2)$

   3*b*. $AG(N_1T_2 \Rightarrow AX(T_1T_2 \vee N_1C_2) \wedge EX\ T_1T_2)$

   3*c*. $AG(N_1C_2 \Rightarrow AX(T_1C_2 \vee N_1N_2) \wedge EX\ T_1C_2 \wedge EX\ N_1N_2)$

   3*d*. $AG(T_1N_2 \Rightarrow AX(C_1N_2 \vee T_1T_2) \wedge EX\ T_1T_2)$

   3*e*. $AG(T_1T_2 \Rightarrow AX(C_1T_2 \vee T_1C_2))$

   3*f*. $AG(T_1C_2 \Rightarrow AX(C_1C_2 \vee T_1N_2) \wedge EX\ T_1N_2)$

   3*g*. $AG(C_1N_2 \Rightarrow AX(N_1N_2 \vee C_1T_2) \wedge EX\ N_1N_2 \wedge EX\ C_1T_2)$

   3*h*. $AG(C_1T_2 \Rightarrow AX(N_1T_2 \vee C_1C_2) \wedge EX\ N_1T_2)$

   3*i*. $AG(C_1C_2 \Rightarrow AX(N_1C_2 \vee C_1N_2) \wedge EX\ N_1C_2 \wedge EX\ C_1N_2)$

4. Whenever process 1 is in its trying region, irrespective of the state process 2 is in then, it should eventually move to its critical region. When it does so process 2 could be in any state:

   4*a*. $AG(T_1N_2 \Rightarrow AF(C_1N_2 \vee C_1T_2 \vee C_1C_2))$

   4*b*. $AG(T_1T_2 \Rightarrow AF(C_1N_2 \vee C_1T_2 \vee C_1C_2))$

   4*c*. $AG(T_1C_2 \Rightarrow AF(C_1N_2 \vee C_1T_2 \vee C_1C_2))$

5. Whenever process 2 is in its trying region, irrespective of the state process 1 is in then, it should eventually move to its critical region. When it does so process 1 could be in any state:

   5*a*. $AG(N_1T_2 \Rightarrow AF(N_1C_2 \vee T_1C_2 \vee C_1C_2))$

   5*b*. $AG(T_1T_2 \Rightarrow AF(N_1C_2 \vee T_1C_2 \vee C_1C_2))$

   5*c*. $AG(C_1T_2 \Rightarrow AF(N_1C_2 \vee T_1C_2 \vee C_1C_2))$

Figure 6: The SCTL assertions for the mutual exclusion problem (Example 4)

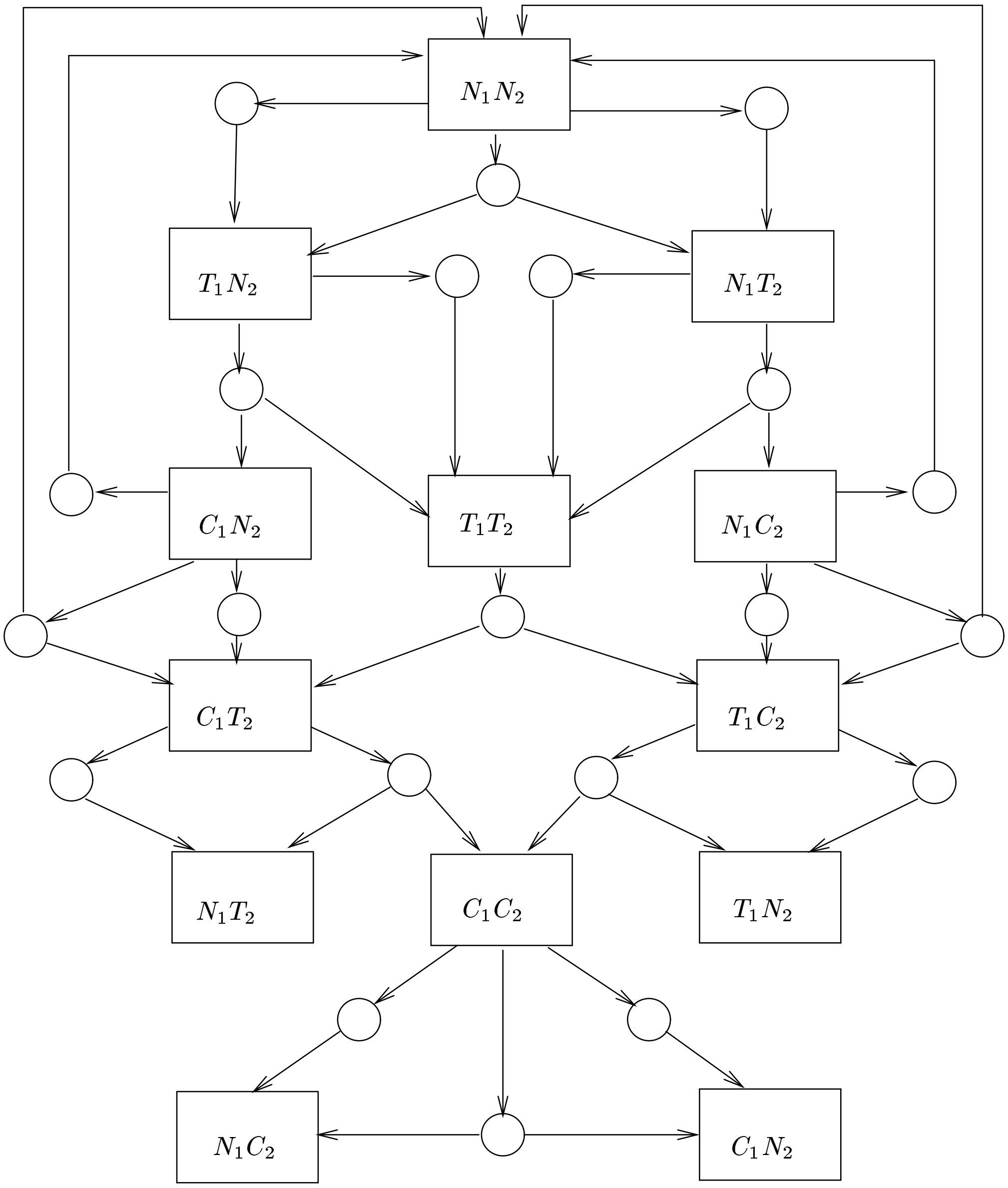


Figure 7: The initial next time tableau for the mutual exclusion problem (Example 5)

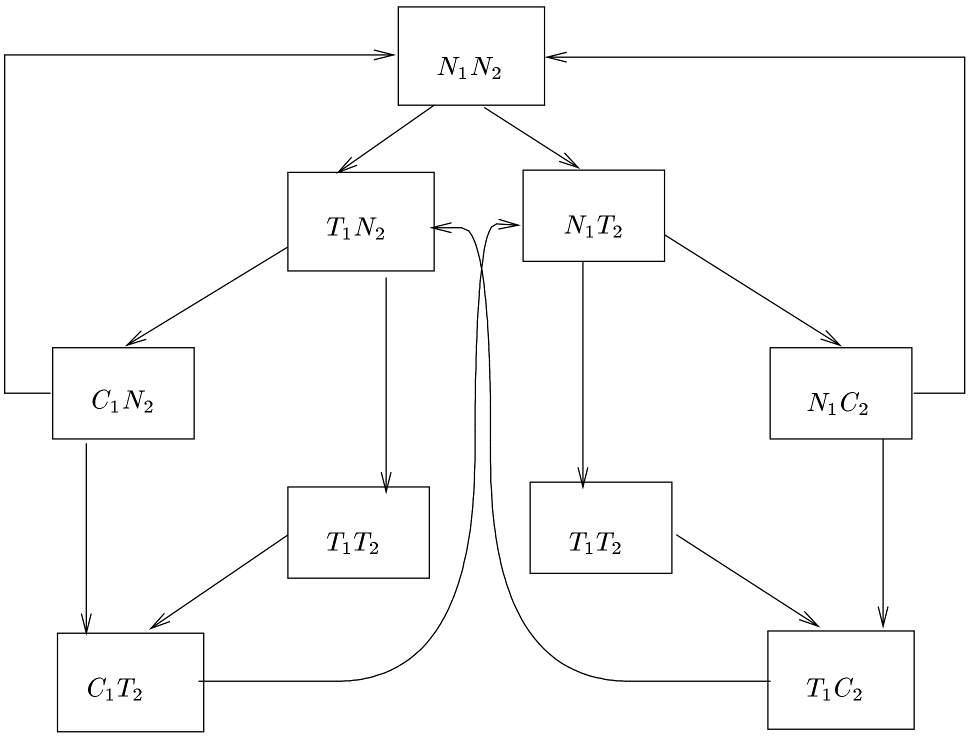


Figure 8: A model for the mutual exclusion problem

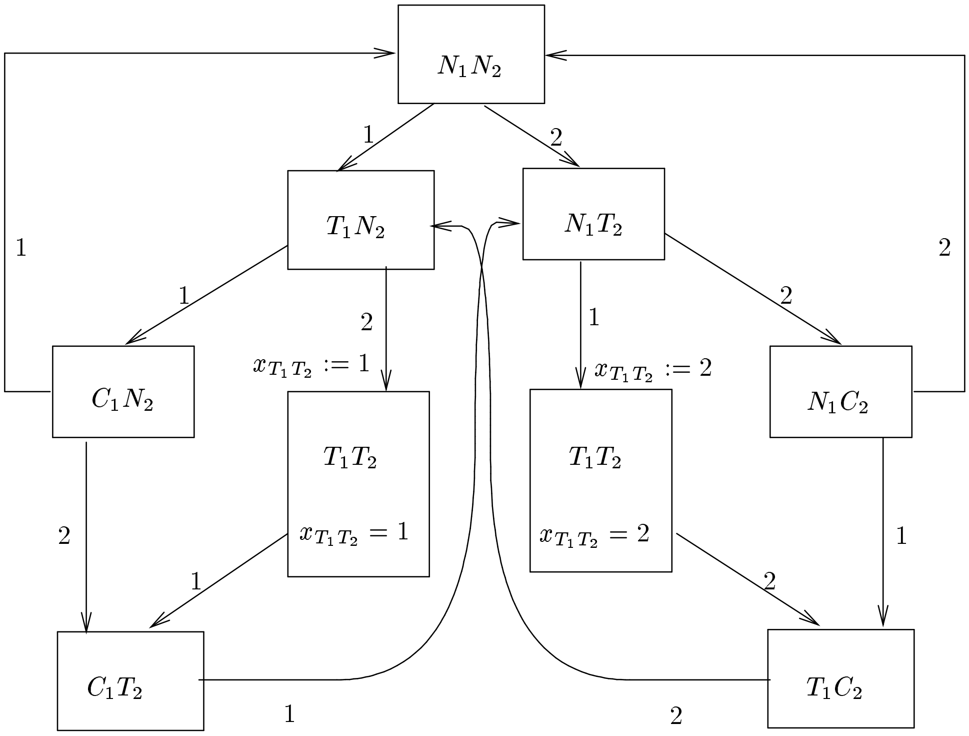


Figure 9: The labelled model

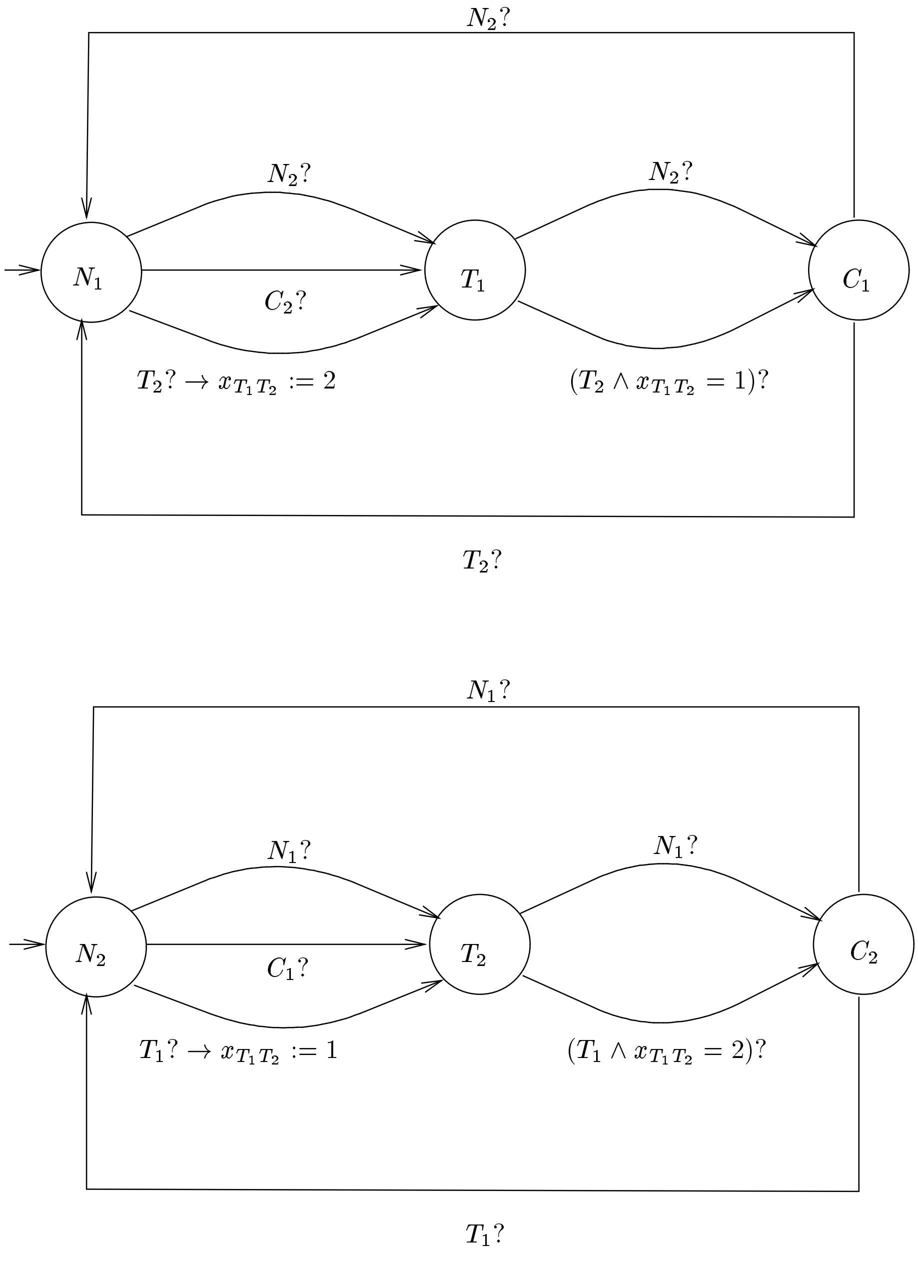


Figure 10: The synchronization skeletons