Quantitative Temporal Reasoning[†]

(Extended Abstract)

E. Allen Emerson[1] *A. K. Mok*[1] *A. P. Sistla*[2] *Jai Srinivasan*[1]

1. Department of Computer Sciences,
The University of Texas at Austin,
Austin, TX 78712.

2. GTE Research Laboratories,
Waltham, MA 02254.

Abstract

A substantially large class of programs operate in distributed and real-time environments, and an integral part of their correctness specification requires the expression of time-critical properties that relate the occurrence of events of the system. We focus on the formal specification and reasoning about the correctness of such programs. We propose a system of temporal logic, RTCTL (Real-Time Computation Tree Logic), that allows the melding of qualitative temporal assertions together with real-time constraints to permit specification and reasoning at the twin levels of abstraction: qualitative and quantitative. We show that several practically useful correctness properties of temporal systems, which need to express timing as an essential part of their functionality requirements, can be expressed in RTCTL. We also develop a model-checking algorithm for RTCTL whose complexity is linear in the size of the RTCTL specification formula and in the size of the global state-space graph. Finally, we present an optimal, exponential time tableau-based decision procedure for the satisfiability of RTCTL formulae, which can be used as the basis of a technique to automate the synthesis of real-time programs from specifications.

1 Introduction

Motivated mainly by the virtue of separating concerns, most of the research into the formal specification and reasoning about the correctness of programs has paid little heed to dealing with quantitative temporal properties. In fact, this has proved to be an advantageous abstraction because, in many applications, the correctness properties of a program need to be stated independently of concerns of efficiency, performance, or features (e.g., the speed) of the underlying hardware implementation. Given this, a common characteristic of most temporal or modal logics heretofore proposed for program reasoning (cf. [Pn77], [FL79], [Ab80], [GPSS80], [BHP81], [BMP81], [Wo81], [EC82], [EH82], [Ko82], [EH83], [ES84], [VW84]) is that they provide a formalism for *qualitative* reasoning about change over time. For example, such formalisms allow the expression of assertions such as an event p will *eventually* occur (stated as $F\,p$); note that this assertion places no bound on the time that may elapse before the occurrence of p. Thus, with $p = halt$, $F\,halt$ asserts that a program terminates, and, indeed, such a qualitative sort of correctness temporal property is in fact the strongest one that may be desirable to state of many programs.

On the other hand, there is a substantially large class of programs that operate in distributed and real-time environments (for example, network communication protocols and embedded real-time control systems), an integral part of whose correctness specification requires the expression of time-critical properties that relate the occurrence of events of the system. For example, consider $p = respond$, in the context of a control system. There, we might want

[†]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.

to assert a quantitative correctness property such as $F^{\leq 50}$ *respond*, meaning that a response is guaranteed within *bounded* time, namely, 50 time units. By and large, the specification and verification of such systems has been *ad hoc*. One common technique of coping with such systems (particularly, network communication protocols) has been to abstract out their timing component, use formal techniques that handle qualitative temporal assertions to ensure that their behaviour is correct upto timing constraints, and then incrementally consider quantitative temporal specifications. This method does appear to have advantages in that it allows one to gain considerable intuition for why some of the specifications may be incorrectly conceived, independent of timing constraints, and that it also permits one to manually "fine-tune" time-critical parameters of a system using one's intimate understanding of the system's internals to guide the process.

We remark that methodologies for specifying and reasoning directly about real-time properties have been proposed, but suffer from some limitations. For one thing, such formalisms are often designed with a very specific application in mind and fail to consider or appropriately generalize the kinds of properties expressible, or to characterize the class of properties provably expressible. Also, the ability to describe properties of a system is only one component of a formalism: one also needs develop techniques to effectively, and tractably, reason about such assertions. The few methodologies that do not resort to doing this arbitrarily often use techniques that are *rigid*, in the sense that the correctness of the analysis depends on the specific values of constants in the assertions, and is not easily modified to still be correct for different values of these constants. Moreover, the two-phased capability of being able to reason, first qualitatively, and, then, refining the reasoning to be quantitative, is lost.

In this paper, we provide general techniques to augment systems of temporal logic to handle quantitative assertions. We focus primarily on one system of logic, RTCTL (Real-Time Computation Tree Logic), which extends CTL (Computation Tree Logic, cf. [EC82], [EH82]), a system of logic that has been widely applied to reasoning about program correctness. RTCTL is able to overcome many of the difficulties of the other approaches because it builds on the foundations of temporal logic. For example, it allows the melding of qualitative temporal assertions together with real-time constraints to permit specification and reasoning at the twin levels of abstraction: qualitative and quantitative. It supports not only efficient reasoning at both these levels, but also refinement from the qualitative level down to the quantitative level. Moreover, temporal logic has demonstrably proved to be useful to reason about a variety of discrete systems, and thus, an appropriate extension (such as RTCTL) would naturally allow one to deal with various kinds of applications. And, finally, part of the power of our approach is derived from the fact that standard problems in temporal logic such as *satisfiability* and *model-checking* have been shown to be applicable to automating the construction of, and reasoning about, temporal systems such as concurrent programs (cf. [EC82], [CES83], [MW84], [LP85]), and, furthermore, techniques to effectively tackle these problems are well-established.

Our method is based on the expression of RTCTL assertions in the propositional μ-calculus, extended with natural number ordinal-ranks. We call this new logic Real-μ. In the ordinary μ-calculus ([Ko82]), correctness properties are characterized as extremal fixpoints of predicate transformers similar to those considered by Dijkstra ([Di76]). Here, least fixpoints corresponding to eventualities are annotated with a natural number bound on when they must be fulfilled. We show that several interesting quantitative temporal properties are thereby expressible.

We go on to develop a model-checking algorithm for RTCTL which, like the algorithm for CTL, has complexity linear in the size of the RTCTL specification formula and in the size of the global state-space graph. The key observation that makes this possible is that the model-checking algorithm of [EL88] actually recovers not only whether an eventuality is fulfilled, but also when, based on calculating its rank in the Tarski-Knaster sequence of approximations. Hence, our model-checking algorithm can be generalized to the full language Real-μ in polynomial time complexity.

Next, we focus on the satisfiability problem for RTCTL. We exhibit an exponential time decision procedure for RTCTL, using a *tableau-based* approach (cf. [FL79], [Pr80], [BHP81], [BMP81], [Wo81], [EC82], [EH82], [LPZ85]), and show that it is optimal. The importance of a tableau-based procedure lies in the fact that, unlike the *automata-theoretic* approach (cf. [St81], [WVS83], [VW84], [Em85]), only the tableau-based method has been demonstrably extended to construct a small model of a satisfiable formula. For applications, such as automating synthesis of programs from their specifications, the model corresponds to the global flowgraph of the program, and the ability to generate it is crucial. (Note, therefore, that the alternative approach of deciding RTCTL by translating its formulae to logics—such as the μ-calculus—whose only known decision procedures are automata-theoretic does not suffice.) Thus, our algorithm is a basis for automating the synthesis of programs with timing-constraints, and we expect that the overall synthesis method would be similar to the ones described in [EC82] and [MW84].

The rest of this paper is organized as follows. In the next section, we present the logic RTCTL and some useful assertions expressible in it. Section 3 deals with real-time model-checking, and Section 4 with the satisfiability problem for RTCTL. Finally, Section 5 considers various other quantitative temporal logics derived from CTL.

2 The Logic RTCTL

The system of branching time temporal logic CTL (Computation Tree Logic) has been extensively used to specify and reason about correctness properties of concurrent programs (cf. [EC82], [EH82], [CES83]). One disadvantage of CTL and other extant temporal logics, however, is that they lack the ability to express properties of programs related to real-time. In this section, we define RTCTL (Real-Time CTL), an extension to CTL that permits reasoning about time-critical correctness properties of programs, and give a sample of the kinds of program properties RTCTL can express. We begin, however, with a formal definition of the syntax and semantics of CTL.

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

S1. Each atomic proposition P is a formula.

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

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

A formula of CTL is interpreted with respect to a temporal structure $M = (S, R, L)$ where S is a set of states, R is a binary relation on S that is total (so each state has at least one successor), and L is a labelling which assigns to each state a set of atomic propositions, those intended to be true at the state. Intuitively, the states of a structure could be thought of as corresponding to the states of a concurrent program, the state transitions of which are specified by the binary relation R. 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; intuitively, a fullpath captures the notion of an execution sequence. We write $M, s \models p$ to mean that "formula p is true at state s in structure M". When M is understood we write only $s \models p$. We define $\models$ by induction on formula structure:

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

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

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

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

We now consider some examples of CTL formulae useful to describe qualitative temporal properties of programs. $AF\,q$, for example, specifies the *inevitability* of q: q must eventually hold along all paths. Thus, $AG(p \Rightarrow AF\,q)$ says that p *inevitably leads-to* q: q eventually holds along every path stemming from a state at which p is true. Similarly, $EF\,q$ indicates that q could *potentially* become true: it is true along some one fullpath. Note that none of these modalities allows one to express that q will in fact become true within a certain number, say 10, of state transitions: they merely assert that q will eventually become true.

So we extend CTL to RTCTL. The set of RTCTL (Real-Time Computation Tree Logic) formulae is generated by the rules S1–S3 above together with the rule:

S4. If p, q are formulae and k is any natural number, then so are $A(p\ U^{\leq k}\ q)$ and $E(p\ U^{\leq k}\ q)$.

The temporal structures over which RTCTL formulae are interpreted are the same as CTL structures. The semantics of the new RTCTL modalities are given by:

S4. $s_0 \models A(p\ U^{\leq k}\ q)$ iff for all fullpaths $s_0, s_1, s_2, \ldots$ in M, $\exists i, 0 \leq i \leq k$, such that $s_i \models q$ and $\forall j, 0 \leq j < i, s_j \models p$
$s_0 \models E(p\ U^{\leq k}\ q)$ iff for some fullpath $s_0, s_1, s_2, \ldots$ in M, $\exists i, 0 \leq i \leq k$, such that $s_i \models q$ and $\forall j, 0 \leq j < i, s_j \models p$

Intuitively, k corresponds to the maximum number of permitted transitions along a path of a structure before the eventuality $p\,U\,q$ holds. We follow the convention that each transition takes unit time for execution (but see the remark near the end of Section 3), so k specifies a time bound.

Some other basic modalities of RTCTL are defined as abbreviations: $AF^{\leq k}\,q$ abbreviates $A(true\ U^{\leq k}\,q)$ and $EF^{\leq k}\,q$ abbreviates $E(true\ U^{\leq k}\,q)$. We also define the modality $G^{\leq k}$ (for each natural number k) as the dual of $F^{\leq k}$, i.e., $AG^{\leq k}\,p$ abbreviates $\neg EF^{\leq k}\,\neg p$, and $EG^{\leq k}\,p$ abbreviates $\neg AF^{\leq k}\,\neg p$.

It is worth pointing that the RTCTL modalities elegantly generalize the analogous CTL ones. Specifically, note that $A(p\,U\,q)$ abbreviates $\exists k : A(p\,U^{\leq k}\,q)$, and, similarly, $E(p\,U\,q)$ abbreviates $\exists k : E(p\,U^{\leq k}\,q)$. This motivates the following definition: If $A(p\,U\,q)$ is true at a state s of an RTCTL structure, we define the *rank* of $A(p\,U\,q)$ at s as the smallest natural number k such that $A(p\,U^{\leq k}\,q)$ holds at s. The rank of $AF\,q$, $E(p\,U\,q)$, and $EF\,q$ are defined similarly.

As usual, an RTCTL formula is said to be *valid* if it holds at all states of all structures. From the semantics above, it is easy to verify that the RTCTL formulae $A(p\,U^{\leq k}\,q) \equiv (q \vee (p \wedge AXA(p\,U^{\leq (k-1)}\,q)))$ and $E(p\,U^{\leq k}\,q) \equiv (q \vee (p \wedge EXE(p\,U^{\leq (k-1)}\,q)))$ are valid for $k \geq 1$. Also $A(p\,U^{\leq 0}\,q) \equiv q \equiv E(p\,U^{\leq 0}\,q)$ is valid. These formulae may be regarded as the analogues of the fixpoint characterizations of the CTL modalities AU and EU ([EH82]).

We conclude this section by illustrating how the basic RTCTL modalities could be used to express important correctness properties of programs that must place an explicit bound on the time between events. First, observe that $AF^{\leq k}\,q$, for example, specifies the *bounded inevitability* of q, i.e., q must hold within k steps along all fullpaths. Thus, the RTCTL formula $AG(p \Rightarrow AF^{\leq k}\,q)$ specifies that p always leads-to q within a bounded period of time, viz., k time units. This formula is therefore useful to specify, for example, that a system must respond (with the action q) to an environmental stimulus p within k units of time; the importance of specifications of this kind for temporal systems is underscored in [JM87].

As a second example, consider a family of m processes, the schedules of which are required to satisfy the property of *k-bounded fairness*, i.e., each process should be scheduled for execution at least once every k steps of the system. This can be expressed by the RTCTL formula $\bigwedge_{i=1}^{m} AF^{\leq k}P_i \wedge \bigwedge_{i=1}^{m} AG(P_i \Rightarrow AXAF^{\leq (k-1)}P_i)$, where P_i indicates that process i is executed. The first set of conjuncts ensures that each process is in fact executed along the first k steps, and the AG conjuncts ensure that, once executed, a process must be scheduled for execution again within k steps. We may remark, as an aside, that the property that there be at least one k-bounded fair execution sequence is expressed by the formula $E\big(\bigwedge_{i=1}^{m} F^{\leq k}P_i \wedge \bigwedge_{i=1}^{m} G(P_i \Rightarrow XF^{\leq (k-1)}P_i)\big)$, which does not conform to the syntax of RTCTL.

As a third and final example, consider a system specification that requires that, on sensing an alarm, all normal processes be suspended, and a vigilant mode be entered for at least the next k time units during which only a restricted set of critical activities is performed. The RTCTL formula $AG(alarm \Rightarrow AG^{\leq k}\,vigilant)$ expresses this requirement.

3 Real-Time Model-Checking

In this section, we present an algorithm for the model-checking problem for RTCTL that is linear in both the size of the structure being checked as well as the length of the input formula. Because of the simplicity of the model-checking problem and the efficiency of its solution, the model-checking approach has found several applications to the automatic verification of temporal systems. So far, model-checking algorithms for several temporal logics (cf. [CES83], [LP85], [EL85], [CG87], [SG87]) have been used to verify a large number of finite-state systems ranging from examples of concurrent programs presented in the academic literature (such as the mutual exclusion example in [OL82]) and network communication protocols ([Sif87]) to VLSI circuits ([Br86]). The capability of RTCTL to allow one to reason quantitatively about time in addition to the qualitative reasoning afforded by CTL can only enhance the utility of this problem to such applications, for timing constraints play a key role in both network protocols and hardware circuits.

The way model-checking is applied to program verification may be summarized as follows. The global state transition graph of a finite-state concurrent system may be viewed as a finite temporal structure, and a correctness specification whose truth is to be determined of the program is expressed as a formula in RTCTL. The model-checking algorithm is used to determine whether the formula is true in the structure, and, thereby, whether the given finite-state program meets a particular correctness specification. It is easily appreciated that this approach is potentially of wide applicability since a large class of concurrent programming problems have finite-state solutions, and the interesting properties of many such systems can be specified in a propositional temporal logic.

Formally, the model-checking problem for RTCTL may be stated as: *Given an* RTCTL *formula* p_0 *and a finite temporal structure* $M = (S, R, L)$, *is there a state* $s \in S$ *such that* $M, s \models p_0$? (Note that the RTCTL structure is said to be finite if its *size*, $|M|$, defined as $|S| + |R|$, is finite.)

Fig. 1 presents an algorithm that decides this problem. The goal is to determine, for each state s in M, whether $M, s \models p_0$. The algorithm is designed to operate in stages: the first stage processes all subformulae of p_0 of length 1, the second, of length 2, and so on. At the end of the ith stage, each state is labelled with the set of all subformulae of p_0 of length no more than i that are true at the state. As the basis, note that the labelling L of M initially contains the set of atomic propositions (i.e., all subformulae of p_0 of length 1) true at each state of M. To perform the labelling on subsequent iterations, information gathered in earlier iterations is used. For example, a subformula of the form $q \wedge r$, i.e., one whose main connective is $\wedge$, should be added to the labels of precisely those states already labelled with both q and r. Subformulae of the form $\neg q$ are handled in like fashion.

For the modal subformula $A(q\ U^{\leq k}\ r)$, information from the successor states of s as well as that from the state s itself is used. For now assume that the procedure *AU_check* is always invoked with k instead of $\min(k, |S|)$. Since $A(q\ U^{\leq k}\ r) \equiv r \vee (q \wedge AXA(q\ U^{\leq (k-1)}\ r))$, $A(q\ U^{\leq k}\ r)$ is initially added to the label of each state already labelled with r. The satisfaction of $A(q\ U^{\leq k}\ r)$ is then propagated outward, by repeatedly adding $A(q\ U^{\leq k}\ r)$ to the label of each state labelled by q and having $A(q\ U^{\leq (k-1)}\ r)$ in the labels of all successors. It is fairly easy to see that this propagating step need be repeated at most k times, and that states labelled on the ith step actually satisfy $A(q\ U^{\leq i}\ r)$. Finally, if a state s satisfies $A(q\ U^{\leq i}\ r)$ for some $i \leq k$, it also satisfies $A(q\ U^{\leq k}\ r)$; hence, the last foreach loop in *AU_check* adds $A(q\ U^{\leq k}\ r)$ to the labels of such states.

However, if *AU_check* is invoked with k instead of $\min(k, |S|)$, its complexity would be linear in k. Since k is represented in binary, rather than unary, the complexity of the algorithm would be *exponential* in the length of the binary representation of k, and, hence, in the length of p_0. To overcome this, the invocation to *AU_check* is made with the minimum of k and $|S|$. To see why this suffices, consider $k > |S|$. Note that for $A(q\ U^{\leq k}\ r)$ to hold at state s, there should not be a fullpath stemming from s along which r never holds. Since M is finite, any such fullpath must contain a loop. Thus, it suffices to check if $A(q\ U^{\leq k}\ r)$ holds along the loop-free initial segments of all paths out of s, and such segments have length at most $|S|$. Hence, it suffices to perform the iteration in the procedure *AU_check* just $|S|$ times. For much the same reason, to determine if the formula $A(q\ U\ r)$ holds at a state, it suffices to determine if $A(q\ U^{\leq |S|}\ r)$ holds there. The modalities EU and $EU^{\leq k}$ are handled similarly.

We note that this version of the algorithm can be naively implemented to run in time linear in the length of p_0 and quadratic in the size of the structure M. This is apparent for each of the cases when the main connective of the formula is one of AU, EU, $AU^{\leq k}$ or $EU^{\leq k}$. In the other three cases, the procedure is in fact linear in the size of the structure. However, the techniques of [EL88] are applicable here as well, and can be used to implement the algorithm in time linear in the sizes of both the structure and the formula, i.e., the complexity of the algorithm is $O(|p_0| \times |M|)$ We shall explain this further in the full paper. Thus, we have:

Theorem 1 *The model-checking problem for* RTCTL *is decidable in time linear in both the size of the input structure as well as the length of the input formula.* □

We should remark that, with minor modifications to the procedures *AU_check* and *EU_check*, the above algorithm can as efficiently handle more general temporal structures, ones in which each element of the binary relation R is associated with an integer label that intuitively corresponds to the amount of time taken to "execute" that transition. RTCTL structures as defined in the previous section may be thought of as labelling each element of R with a single unit of time.

As evident from the algorithm, the basic idea behind this mechanical model-checking approach to verification of finite-state systems is to make brute force graph reachability analysis efficient and expressive through the use of temporal logic as an assertion language. Of course, much research in protocol verification—to cite just one area—has attempted to exploit the fact that protocols are frequently finite-state, making exhaustive graph reachability analysis possible. The advantage offered by model-checking seems to be that it provides greater flexibility in formulating specifications through the use of temporal logic as a single, uniform assertion language that can express a wide variety of correctness properties. This makes it possible to reason about, for example, both safety and liveness properties with equal facility. And, now, with RTCTL, quantitative assertions can be handled.

4 Satisfiability for RTCTL

We now turn to the problem of determining the satisfiability of an RTCTL formula. This problem may be stated as: *Given an* RTCTL *formula* f, *is there a temporal structure* M *and a state* s *of* M *such that* $M, s \models f$? If f is true at

```
/* Input:  A structure M = (S, R, L) and an RTCTL formula p0.   */
/* Output: There is a state s ∈ S such that M, s ⊨ p0.          */

for i := 1 to length(p0) do begin
    foreach subformula p of p0 of length i do begin
        case structure of p is of the form
            P, an atomic proposition : /* Nothing to do as states of M already labelled with propositions. */;
            q ∧ r          :  foreach s ∈ S do
                                  if q ∈ L(s) and r ∈ L(s) then add q ∧ r to L(s);
            ¬q             :  foreach s ∈ S do
                                  if q ∉ L(s) then add ¬q to L(s);
            EX q           :  foreach s ∈ S do
                                  if q ∈ L(t) for some R-successor t of s then add EX q to L(s);
            A(q U^≤k r)    :  AU_check (min(k, |S|), q, r);
            A(q U r)       :  AU_check (|S|, q, r);
            E(q U^≤k r)    :  EU_check (min(k, |S|), q, r);
            E(q U r)       :  EU_check (|S|, q, r);
        end; /* case */
    end;     /* foreach */
end;         /* for */
if p0 ∈ L(s) for some s ∈ S then Output (true)
                            else Output (false);


procedure AU_check (maxrank, q, r);

begin
    foreach s ∈ S do
        if r ∈ L(s) then add A(q U^≤0 r) to L(s);
    for rank := 1 to maxrank do
        foreach s ∈ S do
            if q ∈ L(s) and A(q U^≤(rank-1) r) ∈ L(t) for every R-successor t of s then
                add A(q U^≤rank r) to L(s);
    foreach s ∈ S do
        if A(q U^≤j r) ∈ L(s) for some j ≤ maxrank then add A(q U^≤maxrank r) to L(s);
end;  /* AU_check */


procedure EU_check (maxrank, q, r);

begin
    foreach s ∈ S do
        if r ∈ L(s) then add E(q U^≤0 r) to L(s);
    for rank := 1 to maxrank do
        foreach s ∈ S do
            if q ∈ L(s) and E(q U^≤(rank-1) r) ∈ L(t) for some R-successor t of s then
                add E(q U^≤rank r) to L(s);
    foreach s ∈ S do
        if E(q U^≤j r) ∈ L(s) for some j ≤ maxrank then add E(q U^≤maxrank r) to L(s);
end;  /* EU_check */
```

Figure 1: A Model-Checking Algorithm for RTCTL.

state s of M, M is said to be a *model* of f. Note that the RTCTL formula f is satisfiable iff $\neg f$ is not valid; hence exhibiting a decision procedure for satisfiability amounts to deciding the validity problem (i.e., determining if a given RTCTL formula is valid) as well.

The satisfiability problem for temporal logics has been shown to have applications to synthesis of concurrent programs from their temporal specifications (cf. [EC82], [MW84], [ESS89], [PR89]). The method determines whether the temporal logic formula expressing the program specifications is satisfiable, and, if so, produces a model of the formula. The model may be viewed as the global flowgraph of a program implementing the specifications, and the program itself can be read off from the model. If the formula is not satisfiable, the specification is *inconsistent:* there is no program that implements it.

As mentioned in the introduction, only the tableau-based approach has been demonstrably extended to produce actual models of satisfiable formulae. Thus, we seek a tableau-based algorithm to decide the satisfiability of RTCTL formulae. A naive way to do this is to translate the given RTCTL formula, f, to an equivalent CTL one, g, by using the fixpoint characterizations of the $AU^{\leq k}$ and $EU^{\leq k}$ modalities to expand each occurrence of these modalities in f. The tableau-based decision procedure for CTL could then be used to determine the satisfiability of g. But the complexity of such an algorithm would be double exponential in $|f|$, as $|g|$ itself would be exponential in $|f|$, and the CTL decision procedure is exponential in the length of its input.

Instead, we outline a direct tableau-based decision procedure for RTCTL, whose complexity is only exponential in the size of its input. Let f be the RTCTL formula whose satisfiability needs to be determined. We first define several useful notions used in the description of the procedure, beginning with the *Fischer-Ladner closure*, $CL(f)$, of an RTCTL formula f (cf. [FL79], [EH82], [LPZ85]). For conciseness of presentation, we assume that f is strictly in the syntax presented, i.e., it does not have any of the abbreviations listed in Section 2. Identifying $\neg\neg p$ with p, and $A(p\ U^{\leq 0} q)$ and $E(p\ U^{\leq 0} q)$ with q for any RTCTL formulae p and q, $CL(f)$ is the smallest set of formulae containing f and satisfying the following eight conditions:

A. $\neg p \in CL(f) \quad \Leftrightarrow \quad p \in CL(f)$,
B. $p \wedge q \in CL(f) \quad \Rightarrow \quad p, q \in CL(f)$,
C. $EX\,p \in CL(f) \quad \Rightarrow \quad p \in CL(f)$,
D. $AX\,p \in CL(f) \quad \Rightarrow \quad p \in CL(f)$,
E. $A(p\ U\ q) \in CL(f) \quad \Rightarrow \quad p, q, AXA(p\ U\ q) \in CL(f)$,
F. $E(p\ U\ q) \in CL(f) \quad \Rightarrow \quad p, q, EXE(p\ U\ q) \in CL(f)$,
G. $A(p\ U^{\leq k} q) \in CL(f) \Rightarrow \quad p, q, AXA(p\ U^{\leq (k-1)} q) \in CL(f)$ for $k \geq 1$, and
H. $E(p\ U^{\leq k} q) \in CL(f) \Rightarrow \quad p, q, EXE(p\ U^{\leq (k-1)} q) \in CL(f)$ for $k \geq 1$.

Note that the size of $CL(f)$ is exponential in $|f|$. We shall call a formula in $CL(f)$ *elementary* if it is of the form $EX\,p$ or $AX\,p$. We define a subset S of $CL(f)$ to be *maximally consistent* iff S satisfies all the following conditions:

1. For each $p \in CL(f)$, $\neg p \in S \Leftrightarrow p \notin S$,
2. $p \wedge q \in S \quad \Leftrightarrow \quad p, q \in S$,
3. $A(p\ U\ q) \in S \quad \Leftrightarrow \quad q \in S$ or $p, AXA(p\ U\ q) \in S$,
4. $E(p\ U\ q) \in S \quad \Leftrightarrow \quad q \in S$ or $p, EXE(p\ U\ q) \in S$,
5. $A(p\ U^{\leq k} q) \in S \quad \Leftrightarrow \quad q \in S$ or $p, AXA(p\ U^{\leq (k-1)} q) \in S$ for $k \geq 1$,
6. $E(p\ U^{\leq k} q) \in S \quad \Leftrightarrow \quad q \in S$ or $p, EXE(p\ U^{\leq (k-1)} q) \in S$ for $k \geq 1$,
7. $A(p\ U^{\leq 0} q) \in S \quad \Leftrightarrow \quad q \in S$,
8. $E(p\ U^{\leq 0} q) \in S \quad \Leftrightarrow \quad q \in S$,
9. $A(p\ U^{\leq k} q) \in S \quad \Rightarrow$ for all $j \geq k$ such that $A(p\ U^{\leq j} q) \in CL(f)$, $A(p\ U^{\leq j} q) \in S$, and
10. $E(p\ U^{\leq k} q) \in S \quad \Rightarrow$ for all $j \geq k$ such that $E(p\ U^{\leq j} q) \in CL(f)$, $E(p\ U^{\leq j} q) \in S$.

We now show that the number of maximally consistent subsets of $CL(f)$ is only exponential in $|f|$. An *eventuality* is any formula of the form $A(p\ U\ q)$, $E(p\ U\ q)$, $A(p\ U^{\leq k} q)$, or $E(p\ U^{\leq k} q)$. We shall call a formula in $CL(f)$ *quantitative* if it is of the form $A(p\ U^{\leq k} q)$, $AXA(p\ U^{\leq k} q)$, $E(p\ U^{\leq k} q)$, or $EXE(p\ U^{\leq k} q)$. We let $\mathcal{H}$ denote the set of quantitative eventualities that appear in f as subformulae. We partition the positive formulae (i.e., formulae not of the form $\neg p$) in $CL(f)$ into $|\mathcal{H}| + 1$ sets: each quantitative eventuality $H = A(p\ U^{\leq k_H} q)$ (respectively, $H = E(p\ U^{\leq k_H} q)$) that appears in f has a corresponding partition, Y_H, which contains all formulae in $CL(f)$ of the form $A(p\ U^{\leq j} q)$ and $AXA(p\ U^{\leq j} q)$ (respectively, $E(p\ U^{\leq j} q)$ and $EXE(p\ U^{\leq j} q)$), where $j \leq k_H$. All other positive formulae in $CL(f)$ are members of a separate partition, Y_0. It is easy to see that $|Y_0|$ is linear in $|f|$, whereas, for any $H \in \mathcal{H}$, $|Y_H|$ is exponential in the number of bits in k_H, and hence, exponential in $|f|$.

Next, we note from Rule 1 above that in constructing any maximally consistent set S, we have two choices for each formula in Y_0: either include it in S or include its negation in S. For the formulae in Y_H, however, Rules 9 and 10 imply that we can effectively choose only one j, viz., the smallest one, which is no more than k_H, such that $A(p\ U^{\leq j}\ q)$ or $E(p\ U^{\leq j}\ q)$ is in S. Also, once this choice of the smallest j is made, Rules 5 and 6 determine the quantitative elementary formulae of H that must appear in S. Thus, the number of distinct maximally consistent sets is of the order of $2^{|Y_0|} \times \prod_{H \in \mathcal{H}}(k_H + 2)$, i.e., of the order of $2^{|f|}$. Note, also, that the number of elements in a maximally consistent set is also exponential in $|f|$.

The decision procedure we outline focusses on establishing a *small-model theorem* for RTCTL, i.e., on showing that each satisfiable RTCTL formula f has a model that is bounded by some small function of its length. The first step in the procedure is to construct the *initial tableau* for f. This tableau, which we denote by T_0, is a directed graph, whose nodes correspond to the maximally consistent sets of $CL(f)$. A node corresponding to the set S is labelled with the formulae in S. We use the elementary formulae in a node to guide us in determining the edges of T_0. An edge is added from node V to node W iff (a) for every formula of the form $AX\,p$ in V, p is in W, and (b) for every formula of the form $\neg EX\,p$ in V, $\neg p$ is in W.

The next step is to prune T_0 by deleting nodes the conjunction of the formulae in whose labels cannot ever label any state of any temporal structure. Despite the fact that RTCTL has more kinds of eventualities than CTL, this step is *identical* to the pruning step for CTL (cf. [EH82]). The main task of the pruning step in the CTL algorithm is to check for each eventuality in the label of each node, whether there is a directed acyclic subgraph of the tableau for that eventuality rooted at that node which certifies fulfillment of that eventuality at that node. For RTCTL, it would appear that such a directed acyclic subgraph would need to be detected for the quantitative eventualities in the label of a node as well; however, this is not required because the local structure of the tableau (i.e., the way the initial tableau is constructed) guarantees that such an acyclic subgraph can always be found. We shall describe the pruning procedure for CTL in the full paper and prove:

Proposition 2 *The above algorithm decides the satisfiability of its input* RTCTL *formula f correctly and in time $O(2^{|f|})$.* □

Thus, we have a deterministic decision procedure for RTCTL whose complexity is at most exponential in the length of f. Since the problem of determining the satisfiability of CTL formulae is deterministic exponential time complete ([EH82]), and since RTCTL subsumes CTL, our algorithm is optimal. Also, note that the techniques in [EC82] and [EH82] to construct the initial tableau "bottom-up" are applicable to RTCTL as well. Thus the exponential blow-up in $|f|$ need be incurred only in the worst case, rather than in the average case as would be done by the above naive construction of the initial tableau.

5 Other Quantitative Modalities and Temporal Logics

In this section, we briefly consider two other quantitative temporal modalities: $U^{\geq k}$ and $U^{=k}$. Intuitively, $A(p\ U^{\geq k} q)$ says that q is true after k or more time instants along each fullpath and p is true till then. Similarly, $A(p\ U^{=k} q)$ states that q is true exactly at the kth time instant along all fullpaths and p is true at each of the preceding $k-1$ time instants. More formally, we define the logic CRTCTL (Complete RTCTL) to comprise the formulae generated by the rules S1–S4 in Section 2 together with the rules:

S5. If p, q are formulae and k is any natural number, then so are $A(p\ U^{\geq k} q)$ and $E(p\ U^{\geq k} q)$, and

S6. If p, q are formulae and k is any natural number, then so are $A(p\ U^{=k} q)$ and $E(p\ U^{=k} q)$.

We also define two sublogics of CRTCTL: RTCTL$^{\geq}$, whose formulae are obtained by using Rules S1–S3 and S5, and RTCTL$^{=}$, whose formulae are generated by Rules S1–S3 and S6.

The semantics of the new quantitative modalities are given by:

S5. $s_0 \models A(p\ U^{\geq k} q)$ iff for all fullpaths $s_0, s_1, s_2, \ldots$ in M, $\exists i, i \geq k$, such that $s_i \models q$ and $\forall j, 0 \leq j < i, s_j \models p$
$s_0 \models E(p\ U^{\geq k} q)$ iff for some fullpath $s_0, s_1, s_2, \ldots$ in M, $\exists i, i \geq k$, such that $s_i \models q$ and $\forall j, 0 \leq j < i, s_j \models p$

S6. $s_0 \models A(p\ U^{=k} q)$ iff for all fullpaths $s_0, s_1, s_2, \ldots$ in M, $s_k \models q$ and $\forall j, 0 \leq j < k, s_j \models p$
$s_0 \models E(p\ U^{=k} q)$ iff for some fullpath $s_0, s_1, s_2, \ldots$ in M, $s_k \models q$ and $\forall j, 0 \leq j < k, s_j \models p$

Other abbreviations of these modalities can be defined as in Section 2. It is worth noting that $AF^{=k}p \equiv AG^{=k}p$ is a validity. So is $A(p\ U^{=k}\ q) \Rightarrow A(p\ U^{\leq k}\ q) \wedge A(p\ U^{\geq k}\ q)$, but the same formula with the implication reversed is not valid. We could also define modalities such as $U^{<k}$ and $U^{>k}$, but as we are dealing with discrete time, these are easily expressed in terms of $U^{\leq k}$ and $U^{\geq k}$ respectively.

From the semantics above, it is easy to verify that the following formulae are valid. First, for each $k \geq 1$: (*i*) $A(p\ U^{\geq k}\ q) \equiv p \wedge AXA(p\ U^{\geq (k-1)}\ q)$; (*ii*) $E(p\ U^{\geq k}\ q) \equiv p \wedge EXE(p\ U^{\geq (k-1)}\ q)$; (*iii*) $A(p\ U^{=k}\ q) \equiv p \wedge AXA(p\ U^{=(k-1)}\ q)$; and (*iv*) $E(p\ U^{=k}\ q) \equiv p \wedge EXE(p\ U^{=(k-1)}\ q)$. Secondly, for $k = 0$: (*v*) $A(p\ U^{\geq 0}\ q) \equiv A(p U\ q)$; (*vi*) $E(p\ U^{\geq 0}\ q) \equiv E(p U\ q)$; and (*vii*) $A(p\ U^{=k}\ q) \equiv E(p\ U^{=k}\ q) \equiv q$. These formulae may be regarded as the analogues of the fixpoint characterizations of the CTL modalities AU and EU ([EH82]).

We conclude this section with a summary of results (which we shall prove in the full paper) concerning the logics CRTCTL, RTCTL, RTCTL$^{\geq}$, and RTCTL$^{=}$. First, we note that the expressive power (cf., [GPSS80], [EH83]) of each of these logics is the same as that of CTL as each of them subsumes CTL and the basic quantitative modalities can be expanded into CTL formulae using their fixpoint characterizations. Thus, each of these logics is as expressive as every other.

Secondly, there is a polynomial time model-checking algorithm for each of these logics. In fact, an RTCTL$^{\geq}$ formula, p, can be model-checked over a structure M in time $O(|p| \times |M|^2)$ and an RTCTL$^{=}$ or a CRTCTL formula p can be model-checked in time $O(|p| \times |M|^3)$. The algorithm for RTCTL$^{\geq}$ is similar to that of RTCTL: to test whether a state s in a structure M satisfies $A(q\ U^{\geq k}\ r)$, say, we first rank all states in M that satisfy $A(q\ U\ r)$ with 0, and "radiate" the satisfaction of $A(q\ U^{\geq j}\ r)$, $j \leq k$, outward from these states. To test if s satisfies $A(q\ U^{=k}\ r)$, however, we need to compute all states in M that are k time units away from s. This can be done in time linear in k and polynomial in $|M|$ by an algorithm similar to the one that computes the transitive closure of a directed graph. So, model-checking CRTCTL and RTCTL$^{=}$ is somewhat more computationally intensive.

Finally, the satisfiability problem of RTCTL$^{\geq}$, like that of RTCTL is deterministic EXPTIME-complete; in fact, RTCTL's algorithm can be used with appropriate changes. Surprisingly enough, however, the satisfiability problem of RTCTL$^{=}$ (and, hence, of CRTCTL) is deterministic double exponential time complete. The algorithm outlined for RTCTL can also be modified to handle these logics. The resulting algorithm is double exponential in the number of bits used to represent the integer constants in the input formula, but only single exponential in the length of the remainder of the formula, i.e., in the length of the formula without these integer constants. Thus, this algorithm is likely to be more efficient than translating formulae of these logics to (exponentially longer) formulae in CTL and using CTL's decision procedure to determine their satisfiability.

References

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

[Br86] Browne, M.C., An Improved Algorithm for the Automatic Verification of Finite State Systems Using Temporal Logic, *Proc. Symp. on Logic in Computer Science*, Cambridge, pp. 260–266, 1986.

[BHP81] Ben-Ari, M., J.Y. Halpern, A. Pnueli, Finite Models for Deterministic Propositional Dynamic Logic, *Proc. 8th Annual International Colloquium on Automata, Languages and Programming*, LNCS#115, Springer–Verlag, pp. 249–263, 1981; *a revised version entitled* Deterministic Propositional Dynamic Logic: Finite Models, Complexity, and Completeness, *appears in Journal of Computer and System Sciences*, vol 25, no. 3, pp. 402–417, 1982.

[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; *also appeared in Acta Informatica*, vol. 20, no. 3, pp. 207–226, 1983.

[CES83] Clarke, E.M., E.A. Emerson, A.P. Sistla, Automatic Verification of Finite State Concurrent Systems Using Temporal Logic Specifications: A Practical Approach, *Proc. 10th Annual ACM Symp. on Principles of Programming Languages*, Austin, pp. 117–126, 1983; *also appeared in ACM Transactions on Programming Languages and Systems*, vol. 8, no. 2, pp. 244–263, 1986.

[CG87] Clarke, E.M., O. Grumberg, Avoiding the State Explosion Problem in Temporal Model Checking Algorithms, *Proc. of the 6th Annual ACM Symp. on Principles of Distributed Computing*, Vancouver, pp. 294–303, 1987.

[Di76] Dijkstra, E.W., *A Discipline of Programming*, Prentice–Hall, 1976.

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

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

[EH82] Emerson, E.A., J.Y. Halpern, Decision Procedures and Expressiveness in the Temporal Logic of Branching Time, *Proc. of the 14th Annual ACM Symp. on Theory of Computing*, San Francisco, pp. 169–180, 1982; *also appeared in Journal of Computer and System Sciences*, vol 30, no. 1, pp. 1–24, 1985.

[EH83] Emerson, E.A., J.Y. Halpern, "Sometimes" and "Not Never" Revisited: On Branching versus Linear Time, *Proc. 10th Annual ACM Symp. on Principles of Programming Languages*, Austin, pp. 127–140, 1983; *also appeared in Journal ACM*, vol 33, no. 1, pp. 151–178, 1986.

[EL85] Emerson, E.A., C.L. Lei, Modalities for Model Checking: Branching Time Logic Strikes Back, *Proc. 12th Annual ACM Symp. on Principles of Programming Languages*, New Orleans, pp. 84–96, 1985; *also appeared in Science of Computer Programming*, vol. 8, pp. 275–306, 1987.

[EL88] Emerson, E.A., C.L. Lei, Model-Checking in the Propositional Mu-Calculus, *unpublished manuscript*, 1988.

[ES84] Emerson, E.A., A.P. Sistla, Deciding Full Branching Time Logic, *Information and Control*, vol. 61, no. 3, pp. 175–201, 1984; *also appeared in Proc. of the 16th Annual ACM Symp. on Theory of Computing*, Washington D.C., pp. 14–24, 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.

[GPSS80] Gabbay, D., A. Pnueli, S. Shelah, J. Stavi, On the Temporal Analysis of Fairness, *Proc. 7th Annual ACM Symp. on Principles of Programming Languages*, Las Vegas, pp. 163–173, 1980.

[JM87] Jahanian, F., A.K. Mok, A Graph-Theoretic Approach for Timing Analysis and its Implementation, *IEEE Transactions on Computers*, vol. C–36, no. 8, pp. 961–975, 1987.

[Ko82] Kozen, D., Results on the Propositional μ-Calculus, *Proc. 9th Annual International Colloquium on Automata, Languages and Programming*, LNCS#140, Springer–Verlag, pp. 348–359, 1982; *also appeared in Theoretical Computer Science*, vol. 27, no. 3, pp. 333–354, 1983.

[LP85] Lichtenstein, O., A. Pnueli, Checking That Finite State Concurrent Programs Satisfy Their Linear Specification, *Proc. 12th Annual ACM Symp. on Principles of Programming Languages*, New Orleans, pp. 97–107, 1985.

[LPZ85] Lichtenstein, O., A. Pnueli, L. Zuck, The Glory of The Past, *Proc. Conf. on Logics of Programs*, Brooklyn, R. Parikh, editor, LNCS#193, Springer–Verlag, pp. 196–218, 1985.

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

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

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

[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.

[Pr80] Pratt, V., A Near-Optimal Method For Reasoning About Action, *Journal of Computer and System Sciences*, vol 20, no. 2, pp. 231–254, 1980.

[Sif87] J. Sifakis, *personal communication*, 1987.

[SC82] Sistla, A.P., E.M. Clarke, The Complexity of Propositional Linear Temporal Logics, *Proc. of the 14th Annual ACM Symp. on Theory of Computing*, San Francisco, pp. 159–168, 1982; *also appeared in Journal ACM*, vol. 32, no. 3, pp. 733–749, 1985.

[SG87] Sistla, A.P., S.M. German, Reasoning With Many Processes, *Proc. 2nd Annual Symp. on Logic in Computer Science*, Ithaca, pp. 138–152, 1987.

[St81] Streett, R.S., Propositional Dynamic Logic of Looping and Converse, Ph.D. Thesis, *MIT LCS* Technical Report TR-263, 1981; *alternatively, see:* Propositional Dynamic Logic of Looping and Converse is Elementarily Decidable, *Information and Control*, vol. 54, pp. 121–141, 1982.

[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; *also appeared in Journal of Computer and System Sciences*, vol 32, no. 2, pp. 183–221, 1984.

[Wo81] Wolper, P., Temporal Logic Can Be More Expressive, *22nd Annual Symp. on Foundations of Computer Science*, Nashville, pp. 340–348, 1981; *also appeared in Information and Control*, vol. 56, pp. 72–99, 1983.

[WVS83] Wolper, P., M. Vardi, A.P. Sistla, Reasoning about Infinite Computation Paths, *24th Annual Symp. on Foundations of Computer Science*, Tucson, pp. 185–194, 1983.