Sādhanā Vol. 27, Part 2, April 2002, pp. 163–180. © Printed in India

Min-max event-triggered computation tree logic

PALLAB DASGUPTA, P P CHAKRABARTI and JATINDRA KUMAR DEKA

Department of Computer Science and Engineering, Indian Institute of Technology, Kharagpur, India 721 302
e-mail: {pallab,ppchak,jatin}@cse.iitkgp.ernet.in

Abstract. Very often timing verification involves the analysis of the timings of discrete events such as signal changes, sending and receiving of signals, and sensitization of edge-triggered circuit components. The main bottleneck in verifying timing properties of timed finite state machines (FSM) has been the inherent complexity of verifying timed properties (PSPACE-complete for timed extensions of computational tree logic (CTL)). Often however, we are interested in the best case or worst case timings between events. In this paper we introduce a temporal query language called Min-max Event-Triggered Computational Tree Logic for expressing such extremal queries on the timings of events and show that such queries can be evaluated in time polynomial in the size of the system times the length of the formula.

Keywords. Computational tree logic; timing verification; temporal query language.

1. Introduction

Temporal logic model-checking (Clarke *et al* 1986) is one of the most popular and well studied paradigms for formal verification of hardware (see Clarke & Kurshan 1996, for a survey). In this approach the property to be verified on a given finite state machine (FSM) is specified in a temporal logic. Model checking has been extensively studied for two broad categories of temporal logics, namely *linear time temporal logic* and *branching time temporal logic* (Burch *et al* 1994; Clarke *et al* 1996).

In order to verify timing properties involving absolute quantums of time, researchers have extended *untimed* temporal logics such as *Linear Temporal Logic* (LTL), *Computation Tree Logic* (CTL), and CTL* (Clarke *et al* 1986) to *timed* temporal logics such as *Timed CTL* (TCTL) (Alur & Henzinger 1993) and *Real Time CTL* (RTCTL) (Emerson *et al* 1989). However, the verification of timed temporal logics has been found to be much more complex than their untimed counterparts (see Alur 1998, for a survey). For example, while LTL model checking is PSPACE complete, TLTL model checking is undecidable (Alur & Henzinger 1994). The problem is less severe in the case of branching time timed logics, where TCTL model checking is PSPACE complete (Alur & Henzinger 1993; Alur *et al* 1993) (whereas CTL model checking is possible in polynomial time). It has been shown

by Alur *et al* (1993) that TCTL model checking is PSPACE complete even in discrete-time models.

Often while evaluating the timing properties of an FSM, we are interested in evaluating the best case or worst case timing properties of signals (more formally, atomic propositions). For example, in an arbitrated bus model (such as AMBA or PCI) we are typically interested in the worst case delay between a request signal for the bus and the corresponding grant signal. In a previous work (Dasgupta *et al* 2001), we presented a temporal logic, Min-max CTL, for reasoning about such extremal timing properties and showed that Min-max CTL evaluation works in time polynomial in the size of the state space and the length of the formula. We also illustrated the expressibility of Min-max CTL over diverse problem domains.

While Min-max CTL can express queries related to the earliest and latest timings of signals, we are sometimes interested in the timings of non-extremal occurrences of signals as well. Specification of the earliest and latest occurrences of signals along a computation path is achieved in Min-max CTL by using the *until-min* ($U_{\min}$) and *until-max* ($U_{\max}$) operators in place of the *until* (U) operator of CTL. Since Min-max CTL is solely concerned with the extremal timings of signals, the syntax of Min-max CTL does not contain the usual *until* (U) operator of CTL. However in order to generalize Min-max CTL to be able to specify queries relating to the non-extremal timings of signals we need to add the CTL *until* operator, U, into the syntax of Min-max CTL.

In this paper, we investigate the implications of augmenting Min-max CTL with the usual *until* operator, U, of CTL. We show that as soon as this is done, the complexity of Min-max CTL evaluation increases significantly, and the algorithm for evaluation is at best pseudo-polynomial. We then show that reasoning about the timings of changes in signal values (which we call events) is significantly more efficient (polynomial time) as against reasoning about the timings of signals. This result is indeed good news since very often timing verification actually involves the analysis of the timings of discrete events such as signal changes, the sending and receiving of signals, and sensitization of edge triggered circuit components.

In order to express the above class of efficiently checkable event timing queries, we define an extension of Min-max CTL, called *Min-max Event Triggered Computation Tree Logic* (Min-max ETCTL). We then present an algorithm for Min-max ETCTL evaluation and show that it runs in time polynomial in the size of the state space times the length of the formula.

The paper is organized as follows. In § 2, we outline the syntax of Min-max CTL. Section 3 describes the formal model of computation. We introduce the syntax and semantics of Min-max ETCTL in §§ 4 and 5 respectively. Section 6 demonstrates the problems induced by using the CTL *until* operator in Min-max CTL, and how the problem is surpassed in Min-max ETCTL by using a restricted syntax. Section 7 presents an algorithm for Min-max ETCTL evaluation and establishes that the algorithm works in time polynomial in the size of the state space times the length of the formula.

2. Min-max CTL

The syntax of Min-max CTL (Dasgupta *et al* 2001) is an extension of CTL, where we embed the notion of costs and optimization functions over the computation tree. For example, consider the CTL formula $\varphi = E(f_1 U f_2)$ (where f_1 and f_2 are CTL formulas). φ is true at

a state s iff there exits a path π starting at s to some state t, such that f_2 is true at t and f_1 is true at each state (if any) in π preceding t. In Min-max CTL, we may pose the query $\psi = \min E_{C(g,h)}(f_1 U_{\min} f_2)$, where f_1 is a CTL formula and f_2 is a Min-max CTL formula, and $C(g, h)$ is a cost function over two variables g and h which are defined below. Evaluation of ψ on a state s returns a value. If $E(f_1 U f_2)$ is not true in s then the value returned is NULL. Otherwise, consider the set, W, of paths starting from s which satisfy $f_1 U f_2$. In each such path π, consider the *earliest* state, t, such that f_2 is true in t and f_1 is true in all states preceding t in π. (If we had used $U_{\max}$ instead of $U_{\min}$, then we would have considered the last state t in π where f_2 is true and f_1 is true in all preceding states). The state t is called the *closing state* of π for $f_1 U_{\min} f_2$. The distance from s to the closing state t is represented by g. The value returned by evaluating f_2 at t is represented by h. The Min-max value of path π for ψ is the value of $C(g, h)$. The Min-max value of state s for ψ is the *minimum* among the Min-max values of the paths in W. (If we had used max in place of min outside the E in ψ, then the Min-max value of s would have been the *maximum* among the Min-max values of paths in W.)

The syntax of Min-max CTL is as follows. B denotes boolean formulas, S denotes CTL formulas, and Z denotes Min-max CTL formulas. C and C' are user defined functions.

- $B = false|true|q|\neg q|B \wedge B|B \vee B|\neg B$
- $S = B|S \wedge S|S \vee S|E(S\ U\ S)|A(S\ U\ S)|\neg S$
- $Q = \min | \max$
- $Z = Z \wedge S|QE_{C(g,h)}(SU_Q Z)|QE_{C'(g)}(SU_Q S)|QA_{C(g,h)}(SU_Q Z)|QA_{C'(g)}(SU_Q S)$

We illustrate Min-max CTL through example 1 below.

Example 1. Consider the timed model shown in figure 1. The timed model represents the states of a client process in a closed system (where the behaviour of the server process is known). The states labeled *req* are states in which it makes a *request*, and the states labeled *gr* are states in which it gets a *grant*. Suppose we wish to determine the minimum among the worst-case response times for the earliest request along all possible computation paths of the client, where the worst-case response time for a *request* of a client is defined as the maximum delay by which it reaches a *grant* state. This query can be formulated in Min-max CTL as:

$$\varphi = \min\ E_h(true\ U_{\min}\ \max\ E_g(req\ U_{\min}\ gr))$$

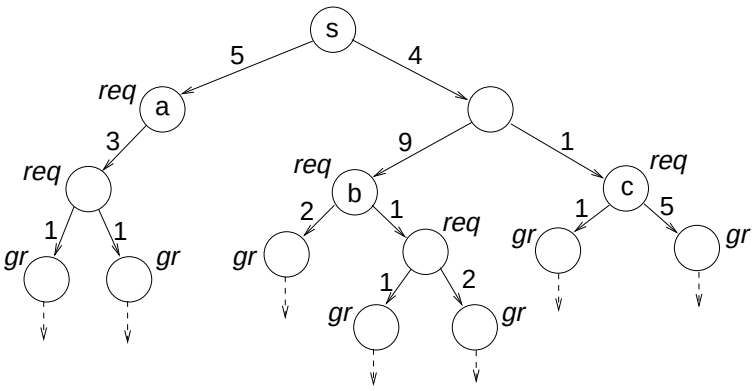

Figure 1. Sample timed model.

Thus, $\varphi = \min\ E_h(\mathit{true}\ U_{\min} f)$ where $f = \max\ E_g(\mathit{req}\ U_{\min}\ gr)$. The closing states for φ at s are the states a, b, and c. Evaluating f at a, b, and c returns 4, 3, and 5 respectively. These are the h-values respectively for the paths to the closing states a, b and c for evaluating φ at s. Since the cost function for φ is h and the objective function is min, the evaluation of φ at s returns 3. □

It should be noted that the syntax of Min-max CTL does not allow the use of the usual *until* operator, U, of CTL. Properties involving the CTL until operator may be useful in many cases. For example, in the context of the previous example, we may be interested in determining the minimum among the worst-case response times among requests along all possible computation paths – not necessarily the first one. This may be specified by the following property provided we allow the CTL until operator:

$$\varphi = \min\ E_h(\mathit{true}\ U\ \max\ E_g(req\ U_{\min}\ gr)).$$

In the next sections we shall augment Min-max CTL with the *until* operator of CTL and study the implications in terms of evaluation complexity.

3. Timed event structures

We first define the formal model used throughout this paper to represent timed finite state machines. We call each sequence of state transitions of the system a *computation* of the system. In a computation an *event* is said to occur if:

- Any variable of the system is possibly modified (like assignments).
- A control flow decision is taken based on the value of one or more variables (like a branching or waiting).

Events execute instantaneously (that is, in zero time) and multiple events can occur at the same instant. A computation can be characterized by a sequence of instances of event occurrences, interspersed by known delays during which the system is in transition. The variables of the system do not change during these transition periods. Thus we have two distinct types of *states* of the system:

Event states: States where an event may occur.
Non-event states: Transition states between event states. If the delay between two event states, v_i and v_j, is $\tau_{ij} > 0$, then we have $\tau_{ij} - 1$ non-event states (separated by unit delays) between them. If $\tau_{ij} = 0$, then there are no non-event states between v_i and v_j.

Since we consider finite representations of delays, we work on integer representations with a granularity of one unit. A *timed event structure* is an extension of the Kripke structure obtained by labeling each edge with an integer representing the delay of that edge. Each node of a timed event structure represents an *event state*, that is, a state resulting from the execution of an event along some computation path.

DEFINITION 1 (Timed event structure)

A timed event structure is a tuple $J = \langle \mathcal{AP}, S, \mathcal{R}, s_0, \mathcal{L} \rangle$, where:

- $\mathcal{AP}$ is a finite set of boolean variables (analogous to atomic propositions in Kripke structures),

- S is a set of event states,
- $\mathcal{R} \subseteq S \times S \times \mathcal{N}$ is a transition relation, where $\mathcal{N}$ denotes the set of integers, and $(\nu_i, \nu_j, \tau_{ij}) \in \mathcal{R}$ implies that the delay between successive event states ν_i and ν_j is τ_{ij} units of time,
- $s_0 \in S$ is the initial event state,
- $\mathcal{L} : S \rightarrow 2^{\mathcal{AP}}$ is a labeling of event states with the set of boolean variables true in that state. □

Each transition of a timed event structure having a delay $\tau > 1$ is a compaction of $\tau - 1$ non-event states, that is states where no event can occur. Each non-event state in a transition (ν_i, ν_j, τ) inherits the truth values of the boolean variables in $\mathcal{AP}$ from ν_i.

4. Syntax of Min-max ETCTL

The syntax of Min-max ETCTL is similar to CTL (Clarke *et al* 1986), except for three special types of *until* operators and two trigger constructs *posedge(f)* and *negedge(f),* where f is a boolean formula over the set of atomic propositions. Since we consider timed models, we exclude the next-time (X) operator of CTL. We first define the syntax and then illustrate the logic with examples.

B denotes the set of boolean formulas over the set of atomic propositions $q \in \mathcal{AP}$, S denotes CTL formulas, T denotes trigger formulas and Z denotes Min-max ETCTL formulas. C and C' are user defined functions.

- $B = \mathit{false} \mid \mathit{true} \mid q \mid \neg q \mid B \wedge B \mid B \vee B \mid \neg B$
- $S = B \mid S \wedge S \mid S \vee S \mid E(S\ U\ S) \mid A(S\ U\ S) \mid \neg S$
- $T = \mathit{posedge}(B) \mid \mathit{negedge}(B) \mid T \wedge T \mid T \vee T$
- $Q = \min \mid \max$
- $Z = Z \wedge S \mid QE_{C(g,h)}(S\ U_Q\ Z) \mid QE_{C'(g)}(S\ U_Q\ S) \mid QA_{C(g,h)}(S\ U_Q\ Z) \mid QA_{C'(g)}(S\ U_Q\ S) \mid QE_{C(g,h)}(S\ U\ (T \wedge Z)) \mid QE_{C'(g)}(S\ U\ (T \wedge S)) \mid QA_{C(g,h)}(S\ U\ (T \wedge Z)) \mid QA_{C'(g)}(S\ U\ (T \wedge S))$

Given a computation π, `posedge(f)` is true in an event state if in the preceding state f was false and the execution of the event makes f true, and `negedge(f)` is true in an event state if in the preceding state f was true and the execution of the event makes f false. Since truths of the atomic variables change only on event states, it follows that the trigger formulas can only be true on event states along specific computation paths. We use the following abbreviations:

Fq: *true U q*
$F_{\min}q$: *true* $U_{\min}$ *q*
$F_{\max}q$: *true* $U_{\max}$ *q*

Throughout this paper, Z-formulas, S-formulas and B-formulas refer to formulas derived out of Z, S and B respectively. We first present a few examples to informally explain the semantics of Min-max ETCTL and then proceed to define the formal semantics.

It may be observed that Min-max ETCTL is an extension of Min-max CTL where we allow the *until* operator of CTL in the Z-formulas. However, we restrict the syntax somewhat

by forcing any formula following the *until* operator to appear in conjunction with a trigger formula. The reason for this will be demonstrated after we familiarize the reader with the semantics of Min-max ETCTL.

5. Semantics of Min-max ETCTL

Min-max ETCTL has a semantics of evaluation which returns a numeric value whenever the formula is true at a state. This numeric value quantifies the value of the objective function C (or C') which we seek to optimize, and is the answer to the Min-max ETCTL query at that state. On the other hand, if the formula is false at a state of the model, then the evaluation of the formula at that state returns null.

A Min-max ETCTL formula is said to be true at a state when the *ETCTL restriction* of the Min-max ETCTL formula is true at that state.

DEFINITION 2 (ETCTL-restriction $CR(f)$ of a formula f)

The *ETCTL-restriction*, $CR(f)$ of a Min-max ETCTL formula, f, is the formula, which is obtained by dropping the min, max quantifiers and cost functions in f. Formally,

- If f is an S-formula or B-formula, then $CR(f) = f$.
- For a formula $f = z_1 \wedge z_2$ where z_2 is an S-formula, $CR(f) = CR(z_1) \wedge z_2$.
- For a formula $f = QE_{C'(g)}(z_1 U_{Q'} z_2)$, $CR(f) = E(z_1\ U\ z_2)$.
- For a formula $f = QE_{C(g,h)}(z_1 U_{Q'} z_2)$, $CR(f) = E(z_1\ U\ CR(z_2))$.
- For a formula $f = QA_{C'(g)}(z_1 U_{Q'} z_2)$, $CR(f) = A(z_1\ U\ z_2)$.
- For a formula $f = QA_{C(g,h)}(z_1 U_{Q'} z_2)$, $CR(f) = A(z_1\ U\ CR(z_2))$.
- For a formula $f = QE_{C'(g)}(z_1 U(z_2\ \wedge\ z_3))$ where z_2 is a trigger formula, $CR(f) = E(z_1\ U\ (z_2\ \wedge\ z_3)$.
- For a formula $f = QE_{C(g,h)}(z_1 U(z_2\ \wedge\ z_3))$ where z_2 is a trigger formula, $CR(f) = E(z_1\ U\ (z_2\ \wedge\ CR(z_3))$.
- For a formula $f = QA_{C'(g)}(z_1 U(z_2\ \wedge\ z_3))$ where z_2 is a trigger formula, $CR(f) = A(z_1\ U\ (z_2\ \wedge\ z_3)$.
- For a formula $f = QA_{C(g,h)}(z_1 U(z_2\ \wedge\ z_3))$ where z_2 is a trigger formula, $CR(f) = A(z_1\ U\ (z_2\ \wedge\ CR(z_3))$. □

A *path*, π, in a timed event structure $J = \langle \mathcal{AP}, S, \mathcal{R}, s_0, \mathcal{L} \rangle$ is an infinite sequence of states, $\nu_0, \nu_1, \ldots$, such that for all i, $\nu_i \in S$ and $(\nu_i, \nu_{i+1}, \delta_{i,i+1}) \in \mathcal{R}$. ν_0 is called the starting state of π. Since the timed model has a finite set of states, one or more states of the timed model will appear multiple number of times on a path. In other words, a path (as defined here) is an infinite walk over the state transition graph. π^i denotes the suffix of π starting from the i^{th} state, ν_i, of π.

If a state formula f is true in a state s we write $s \models f$. Likewise, if a path formula f' is true in a path π we write $\pi \models f'$.

- $\forall s \in S, s \models True$ and $s \not\models False$
- $s \models p$ iff $p \in \mathcal{L}(s)$
- $s \models \neg p$ iff $p \notin \mathcal{L}(s)$
- $s \models \varphi_1 \wedge \varphi_2$ iff $s \models \varphi_1$ and $s \models \varphi_2$
- $s \models \varphi_1 \vee \varphi_2$ iff $s \models \varphi_1$ or $s \models \varphi_2$

- $s \models E(\varphi U \psi)$ iff there exists a path $\pi = s_0, s_1, s_2, \ldots$ starting at $s = s_0$ and some $i \geq 0$ such that $s_i \models \psi$ and for all $j < i$, $s_j \models \varphi$. We also say that $\pi \models \varphi U \psi$, that is, the path formula $\varphi U \psi$ is true in the path π.
- $s \models E(\varphi U (posedge(f) \wedge \psi))$ iff there exists a path $\pi = s_0, s_1, s_2, \ldots$ starting at $s = s_0$ and some $i \geq 0$ such that $s_i \models \neg f$, $s_{i+1} \models f \wedge \psi$ and for all $j < i$, $s_j \models \varphi$. We also say that $\pi \models \varphi U (posedge(f) \wedge \psi)$, that is, the path formula $\varphi U (posedge(f) \wedge \psi)$ is true in the path π.
- $s \models E(\varphi U (negedge(f) \wedge \psi))$ iff there exists a path $\pi = s_0, s_1, s_2, \ldots$ starting at $s = s_0$ and some $i \geq 0$ such that $s_i \models f$, $s_{i+1} \models \neg f \wedge \psi$ and for all $j < i$, $s_j \models \varphi$. We also say that $\pi \models \varphi U (negedge(f) \wedge \psi)$, that is, the path formula $\varphi U (negedge(f) \wedge \psi)$ is true in the path π.

The ETCTL restriction of a Min-max ETCTL formula can be evaluated using a simple extension of CTL model checking algorithms. We do not elaborate the algorithm in this paper. For example, for formulas of the form $\eta = E(\varphi U (posedge(f) \wedge \psi))$ it suffices to start from the states where ψ holds consider only those incident transitions which satisfy $posedge(f)$ and work backwards along predecessors which satisfy φ to obtain the set of states satisfying η.

DEFINITION 3 (Closing state and g-value of a path)

Given a CTL path formula, $f' = f_1 \; U \; f_2$, and a path $\pi = \nu_0, \nu_1, \ldots$, where $\pi \models f'$, the boolean function $isclosing(i, \pi, f')$ is defined as follows. $isclosing(0, \pi, f')$ is true iff $\nu_0 \models f_2$. For $i > 0$, $isclosing(i, \pi, f')$ is true iff $\nu_i \models f_2$ and $\forall j, 0 \leq j < i, \nu_j \models f_1$. Let $\delta_{i,i+1}$ denote the delay between ν_i and ν_{i+1}, that is, $(\nu_i, \nu_{i+1}, \delta_{i,i+1}) \in \mathcal{R}$. Given a Min-max CTL path formula $f = z_1 \; U_Q z_2$, and a path π where $\pi \models CR(f)$, the closing state, $CLS(\pi, f)$, and the g-value $GVAL(\pi, f)$ of π with respect to f are defined as follows:

- If Q is min, then let $i = \min\{j : isclosing(j, \pi, CR(f))\}$. Then:

$$\begin{aligned} CLS(\pi, f) &= \nu_i, \\ GVAL(\pi, f) &= \begin{cases} 0, & \text{if } i = 0, \\ \sum_{j=0}^{i-1} \delta_{j,j+1}, & \text{otherwise.} \end{cases} \end{aligned}$$

- If Q is max, then we have two cases. If $\forall j, \exists i, i > j$, and $isclosing(i, \pi, CR(f))$ is true, then $CLS(\pi, f)$ is not well defined and $GVAL(\pi, f) = \infty$. Otherwise, let $i = \max\{j : isclosing(j, \pi, CR(f))\}$. Then:

$$\begin{aligned} CLS(\pi, f) &= \nu_i, \\ GVAL(\pi, f) &= \begin{cases} 0, & \text{if } i = 0, \\ \sum_{j=0}^{i-1} \delta_{j,j+1}, & \text{otherwise.} \end{cases} \end{aligned}$$

  The closing state for path formulas of the form $f' = f_1 \; U \; (posedge(f) \wedge f_2)$ and $f' = f_1 \; U \; (negedge(f) \wedge f_2)$ is defined in a similar way, except that the transition in π leading to the closing state must satisfy $posedge(f)$ and $negedge(f)$ respectively.

□

The Min-max value of a state s in a timed model $J = \langle \mathcal{AP}, S, \mathcal{R}, s_0, \mathcal{L} \rangle$ with respect to a Z-formula f will be denoted by $EvalS(f, s)$. The semantics of $EvalS(f, s)$ is as follows.

- If $s \not\models f$, then $EvalS(f, s) = null$.

- If $f = z_1 \wedge z_2$, where $s \models f$ and (wlog) z_1 is a S-formula and z_2 is a Z-formula, then $EvalS(f, s) = EvalS(z_2, s)$.
- If $f = QE_{C(g,h)}(z_1 U_{Q'} z_2)$, or $f = QA_{C(g,h)}(z_1 U_{Q'} z_2)$ then,

  (1) If Q' is min then:

  – If Q is min, then:

$$EvalS(f, s) = \min\{C(GVAL(\pi, f), EvalS(z_2, CLS(\pi, f))) \text{ among all paths } \pi = \nu_0, \nu_1, \ldots \text{ starting from } \nu_0 = s\},$$

  – If Q is max, then:

$$EvalS(f, s) = \max\{C(GVAL(\pi, f), EvalS(z_2, CLS(\pi, f))) \text{ among all paths } \pi = \nu_0, \nu_1, \ldots \text{ starting from } \nu_0 = s\}.$$

  (2) If Q' is max, then we have a special case to consider. If in some path $\pi = \nu_0, \nu_1, \ldots$ starting from $\nu_0 = s$, we have $\forall j, \exists i, i > j$ and *isclosing*$(i, \pi, CR(f))$ is true, then $CLS(\pi, f)$ is not well defined. However, $GVAL(\pi, f) = \infty$. In order to have a well defined semantics, we restrict the set of admissible cost functions to those which have the following limiting behaviour:

$$\lim_{g \to \infty} C(g, h) = \lim_{g \to \infty} C(g, k).$$

  where k is an arbitrary constant. In other words, we allow only cost functions where the value of $C(g, h)$ is independent of h when g approaches ∞. Under this restriction, we define $EvalS(f, s)$ as follows.

  – If Q is min, then:

$$EvalS(f, s) = \min\{C(GVAL(\pi, f), EvalS(z_2, \eta)) \text{ among all paths } \pi = \nu_0, \nu_1, \ldots \text{ starting from } \nu_0 = s\},$$

  where $\eta = CLS(\pi, f)$ if $CLS(\pi, f)$ is well defined, and some arbitrary constant k otherwise.

  – If Q is max, then:

$$EvalS(f, s) = \max\{C(GVAL(\pi, f), EvalS(z_2, \eta)) \text{ among all paths } \pi = \nu_0, \nu_1, \ldots \text{ starting from } \nu_0 = s\},$$

  where $\eta = CLS(\pi, f)$ if $CLS(\pi, f)$ is well defined, and some arbitrary constant k otherwise.

- If $f = QE_{C(g)}(z_1 U_{Q'} z_2)$, or $f = QA_{C(g)}(z_1 U_{Q'} z_2)$ then,

  (1) If Q is min, then

$$EvalS(f, s) = \min\{C(GVAL(\pi, f)) \text{ among all paths } \pi = \nu_0, \nu_1, \ldots \text{ starting from } \nu_0 = s\}.$$

(2) If Q is max, then

$$EvalS(f, s) = \max\{C(GVAL(\pi, f)) \text{ among all paths } \pi = \nu_0, \nu_1, \ldots \text{ starting from } \nu_0 = s\}.$$

From the above semantics it follows that $EvalS(f, s)$ returns the value returned by the cost function C for some *best* value path π, where *best* is either the minimum cost or the maximum cost. We shall refer to the set of *best* value paths with respect to $EvalS(f, s)$ as $BestP(f, s)$.

The following example illustrates the semantics of Min-max ETCTL.

Example 2. We illustrate the syntax and semantics of Min-max ETCTL through an example on the timed model shown in figure 1. In figure 1 the transition delays are shown on the edges, and the labelling of states with the set of atomic propositions true in them are shown besides the states. Consider the following Min-max ETCTL formulas:

$$f = \min E_h(F_{\min} \max E_g(\text{req } U_{\min} \text{ gr})),$$
$$y = \min E_{g+h}(F_{\min} \max E_g(\text{req } U_{\min} \text{ gr})).$$

Both of these formulas have the subformula $z = \max E_g(\text{req } U_{\min} \text{ gr})$. At each state t where $t \models z$ (that is, the ETCTL-restriction $z' = E(\text{req } U \text{ gr})$, of z is true at t), we get a non-null value for *EvalS(z,t)*. For example, in figure 1, *EvalS(z,a)* = *4*, *EvalS(z,b)* = *3*, and *EvalS(z,c)* = *5*.

- For evaluating f at s, note that the cost function is $C(g, h) = h$, that is, the cost function is independent of the distance g to the state t where z is true. On the other hand the cost depends only on the value h evaluated at the state t by *EvalS(z,t)*. Thus *EvalS(f,s)* = *3*, and the optimal path goes through state b.
- For evaluating y at s, note that the cost function is $C(g, h) = g + h$, that is, we require to minimize the sum of the h-value evaluated by *EvalS(z,t)* and the delay to the state t. It is easy to see that *EvalS(f,s)* = *9*, and the optimal path goes through state a.

□

The complexity of Min-max ETCTL evaluation depends on the nature of the cost functions $C(g)$ and $C(g, h)$ used in the formulas. Earlier (Dasgupta *et al* 2001) we showed that Min-max CTL evaluation is DP-hard in general. We had also established that when $C(g)$ and $C(g, h)$ are monotonic with respect to g and h, Min-max CTL evaluation works in polynomial time. Since Min-max CTL is a fragment of Min-max ETCTL, we restrict Min-max ETCTL to the following monotonic fragment. It may be noted that in all our examples we have actually used monotonic cost functions.

DEFINITION 4 (Monotonic Min-max ETCTL)

A function $f(x)$ is said to be monotonically increasing iff $f(a) > f(b)$ whenever $a > b$. The function is said to be monotonically decreasing iff $f(a) < f(b)$ whenever $a > b$. Also a function $f(x, y)$ is said to be monotonically increasing (decreasing) with respect to x, iff for each constant k, the function $f(x, k)$ is monotonically increasing (decreasing). A Min-max CTL formula is said to be monotonic iff its cost function $C(g, h)$ (or $C(g)$) is monotonically increasing or decreasing with respect to g, and each of it's subformulas are monotonic. Monotonic Min-max CTL is the language consisting of monotonic Min-max CTL formulas only. □

6. Advantages of event triggering

Timed event structures allow us to condense sequences of non-event states into lump transition delays. A delay $\tau > 1$ in a transition (ν_i, ν_j, τ) is a condensation of $\tau - 1$ non-event states into a single lump delay τ. The number of non-event states is exponential in the number of bits required to represent the delays, hence we expect to perform Min-max ETCTL evaluation without having to explicitly un-condense the non-event states in the transitions.

Interestingly, though the non-event states are identical in terms of the truth of the boolean variables in $\mathcal{AP}$, the non-event states are *not* identical in terms of the Min-max ETCTL formulas constructed out of these boolean variables. We shall illustrate this through an example, but we first show that intermediate non-event states are indeed identical with respect to path formulas involving $U_{\min}$ or $U_{\max}$ operators.

DEFINITION 5 (Intermediate non-event state)

In a timed event structure $J = \langle \mathcal{AP}, S, \mathcal{R}, s_0, \mathcal{L} \rangle$, for each $(\nu_i, \nu_j, \tau_{i,j}) \in \mathcal{R}$, there exists $\tau_{i,j} - 1$ non-event states between the event states ν_i and ν_j (unless $\tau_{i,j} = 0$, in which case there are no non-event states between ν_i and ν_j). $\nu_{i,j}^{\delta}$ denotes the state of the system δ units of time after the execution of the event at ν_i when the next event is ν_j. Except $\nu_{i,j}^{1}$ and $\nu_{i,j}^{\tau_{i,j}-1}$, all other non-event states between ν_i and ν_j are defined as intermediate non-event states. In other words, an intermediate non-event state is a non-event state which has a non-event state as a predecessor and a non-event state as a successor. □

We say that a timed event structure is *interval independent* with respect to a temporal logic, if all non-event states in any transition of the structure has the same truth value with respect to every logic that can be specified in that logic.

Theorem 1. *The closing state of a Min-max until path formula (involving $U_{\min}$ or $U_{\max}$) can never be an intermediate non-event state.*

Proof. Consider a path formula $f_1 \, U_Q \, f_2$ (where Q is min or max) and a path $\pi = s_0, s_1, \ldots$ satisfying the path formula. Since CTL-restriction of Min-max CTL formulas are interval-independent, if f_2 is true in one of the non-event states between s_i and s_{i+1}, then f_2 is true in all non-event states between s_i and s_{i+1}. Since we consider only the earliest (as in $U_{\min}$) state or the latest (as in $U_{\max}$) state where f_2 is true, intermediate non-event states can never be closing states for the formula $f_1 \, U_Q \, f_2$. However, the first or the last non-event state in a transition may qualify to be a closing state. □

By theorem 1, we have shown that the closing state of a Min-max *until* path formula can never be an intermediate non-event state. It is however possible that the first or the last non-event state in a transition qualifies to be a closing state. If we treat these first and last non-event states as event states as well, then we have a slightly modified event structure where Min-max ETCTL is interval independent in the strict sense. This situation is explained by example.

Example 3. Consider the timed event structure shown in figure 2a. Suppose we wish to evaluate the following Min-max CTL formula at state s_0:

$$\min E_g(\mathit{true} \; U_{\min} \; A(\mathit{true} \; U \; q)).$$

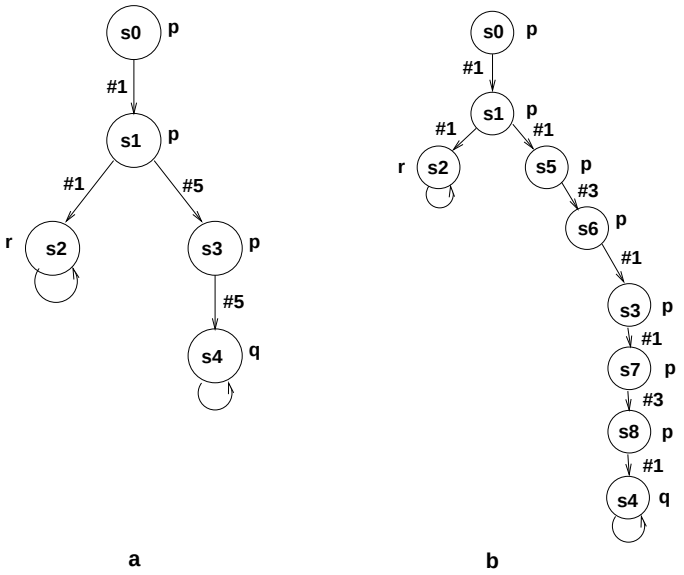

Figure 2. Sample timed event structure and its augmentation.

It is easy to see that the only candidate closing state for the $U_{\min}$-formula is the non-event state immediately following state s_1 in the transition from s_1 to s_3, since this non-event state is the earliest state where $A(true\ U\ q)$ is true.

As a second example, let us consider the following Min-max CTL formula at the state s_0 of the timed event structure of figure 2a:

$$\min E_g(true\ U_{\max}\ p).$$

Since the atomic proposition p is not true at the state s_4, the only candidate closing state for this formula is the non-event state immediately preceding s_4 (that is, the non-event state at a delay of 4 units from state s_3). In both these cases we have assumed the granularity of time to be one unit. Since the granularity of time is one unit, there is no non-event state for unit-delay transition and ϵ-delay transition.

Theorem 1 shows that intermediate non-event states can never be closing states for Min-max CTL formulas. Thus in order to circumvent the problem of handling non-event states, we need to define only the first and last non-event state in every transition (that is, if any such non-event state exists) as event states. Figure 2b shows the modified timed event structure after the redefinition of the first and last non-event states in the transitions. □

We are now in a position to demonstrate the reason for restricting the use of the CTL *until* operator in Min-max ETCTL. Min-max ETCTL does not allow path formulas of the form $f_1 U f_2$ involving the CTL *until* operator. Instead formulas of the form $f_1 U (\eta \wedge f_2)$, where η is a trigger formula are allowed. We first demonstrate the problem which arises out of allowing the CTL *until* operator in an unrestricted way.

Example 4. Consider the timed transition system shown in figure 3. Suppose we wish to evaluate the following query at state s_0:

$$\varphi = \min E_{g^2+h^2}(true\ U_{\min}\ (q \wedge \min E_g(p\ U_{\min}\ r))).$$

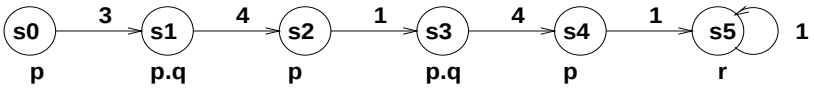

Figure 3. Sample timed transition system.

Let $\psi = (q \wedge \min E_g(p\ U_{\min}\ r))$. If the transition system is a timed event structure, then ψ is also true in the non-event states between s_1 and s_2, and in the non-event states between s_3 and s_4. Since in checking φ, we are looking for *true* $U_{\min}\ \psi$, only s_1 qualifies to be the closing state. Therefore, evaluating φ at s_0 yields $3^2 + 10^2 = 109$.

Suppose we wish to evaluate the following query at state s_0:

$$\varphi = \min E_{g^2+h^2}(\mathit{true}\ U_{\max}\ \psi).$$

The only state which qualifies to be a closing state is the last non-event state preceding s_4 in the transition from s_3 to s_4. As discussed previously, if we treat the first and last non-event states in a transition as event states, then we have interval independence in the strict sense and the query evaluates to $11^2 + 2^2 = 125$ at s_0.

Let us now study what happens when we use the CTL *until* operator in an unrestricted way as follows:

$$\varphi = \min E_{g^2+h^2}(\mathit{true}\ U\ \psi).$$

It may be observed that we no longer have interval independence. Each of the non-event states between s_1 and s_2, as well as all non-event states between s_3 and s_4 qualify to be closing states. The best closing state in this case is the non-event state at a delay of 6 units from s_0 (assuming the granularity of delays as 1 unit), and the value evaluated for φ at s_0 is $6^2 + 7^2 = 85$. □

Since the number of bits required to represent a delay d (which is a compaction of d non-event states in a timed event structure) is $\log_2 d$, having to inspect d non-event states separately is indeed an exponential blow-up in complexity, and we have a pseudo-polynomial algorithm at best. In a previous work on timed verification (Dasgupta *et al* 2000) we had encountered a similar problem, and had observed that reasoning about timings of events (that is, signal changes) does not suffer from this problem. This is demonstrated by the example 5.

Example 5. We continue with the timed event structure shown in figure 3. Now consider the following formula:

$$\varphi = \min E_{g^2+h^2}(\mathit{true}\ U\ (\mathit{posedge}(q) \wedge \min E_g(p\ U_{\min}\ r))).$$

Let $\psi = (\mathit{posedge}(q) \wedge \min E_g(p\ U_{\min}\ r))$. Along all paths starting from s_0, the trigger formula $\mathit{posedge}(q)$ is true only on the event states s_1 and s_3. Thus ψ is true only at these two event states, and only these two event states qualify as closing states for the path formula *true* $U\ \psi$. Therefore φ evaluates to 89 at s_0. □

Thus by forcing the subformula appearing after the CTL *until* operator to appear in conjunction with a trigger formula, we guarantee interval independence of Min-max ETCTL over the augmented timed event structures. The following section shows that Min-max ETCTL evaluation works in time polynomial in the size of the state space times the length of the formula.

Algorithm Evaluate(f,s)

1. Use CTL model checking techniques to label the states of the model with the sub-formulas of the ETCTL restriction of f. During this step, ignore the transition delays.

2. The evaluation at a state, s, of a Min-max ETCTL formula, $f = QE_C(f_1 U_{Q'} f_2)$, where Q and Q' are min or max quantifiers, is done as follows.
The procedure for evaluating formulas of the form $f = QA_C(f_1 U_{Q'} f_2)$ is exactly similar.

   2.1 Recursively evaluate all Min-max ETCTL subformulas of f at all states of the model.

   2.2 If Q' is max then define $\varphi = f_1$ else define $\varphi = f_1 \wedge \neg f_2$.

   2.3 Let H denote the set of states labeled f_2, which are reachable from s along φ-paths.

   2.4 If Q' is max then:

      2.4.1 Remove each state n from H such that:
         (a) n does not belong to any φ-cycle, and
         (b) For each successor n' of n, $n' \models A(f_1 U f_2)$.
      2.4.2 Label a state n from H with the symbol ∞ if:
         (a) n belongs to a φ-cycle, and
         (b) For each successor n' of n, $n' \models A(f_1 U f_2)$.

   2.5 For each state $t \in H$ do:

      2.5.1 If t is labeled ∞ then set $g = \infty$ and compute $W(t) = C(g, h)$, where $h = EvalS(f_2, t)$.
      2.5.2 Else consult table 1 to determine the required path type from s to t, determine the length g of that path, and compute $W(t) = C(g, h)$, where $h = EvalS(f_2, t)$.

   2.6 If Q is min, set $EvalS(f, s) = \min\{W(t) | t \in H\}$
      else set $EvalS(f, s) = \max\{W(t) | t \in H\}$.

3. The evaluation at a state, s, of a Min-max ETCTL formula, $f = QE_C(f_1 \; U \; T \wedge f_2)$, where Q is min or max quantifiers, and T is a trigger formula is done as follows.
The procedure for evaluating formulas of the form $f = QA_C(f_1 \; U \; T \wedge f_2)$ is exactly similar.

   3.1 Recursively evaluate all Min-max ETCTL subformulas of f at all states of the model.

   3.2 Let H denotes the set of states, t, labeled f_2, which are:
      (a) reachable from s along f_1-paths, and
      (b) the trigger formula T is true in the transition to t from the predecessor of t in the f_1-path.

   3.3 For each state $t \in H$ do:

      3.3.1 Consult table 2 to determine the required best path from s through t among those f_1 paths where the trigger formula T is true in the transition to t from the predecessor of t.
      3.3.2 In the chosen path, Compute $W(t) = C(g, h)$, where $h = EvalS(f_2, t)$.

   3.4 If Q is min, set $EvalS(f, s) = \min\{W(t) \mid t \in H\}$
      else set $EvalS(f, s) = \max\{W(t) \mid t \in H\}$.

Figure 4. Algorithm for Min-max ETCTL evaluation.

7. Algorithm for Min-max ETCTL evaluation

In this section we present an algorithm for Min-max ETCTL evaluation and analyse its complexity.

DEFINITION 6 (f-path and f-cycle)

A path, π, starting at a state s and going through a state s' is called a "f-path from s through s'" iff the state formula f holds in all states preceding s' in π. An f-cycle through a state t is an f-path from t through t. □

A shortest length f-path from a state s through a state s' is one where s' occurs as early as in any other f-path from s through s'. The longest length f-path from s through s' is defined similarly, where s' occurs at least as late as any other f-path from s through s'. Obviously, a shortest length f-path will have no f-cycles. Hence shortest length f-paths can be found using any standard shortest path algorithms on the state transition graph in polynomial time.

If any f-path from s through s' contains an intermediate state which is in an f-cycle, then it is possible to have f-paths of infinite length from s through s', and hence the length of the longest f-path from s through s' is ∞. Determining the set of states which belong to f-cycles can be done in polynomial time. Finding whether there exists any f-path from s through s' via any of these states can also be done in polynomial time. If no such f-path exists, then by dropping all states where f is false, we are left with finding a longest length path from s through s' in an acyclic graph, which is also solvable in polynomial time.

The algorithm which we describe is a labelling algorithm. A state, s, in the timed model is labelled by a sub-formula, f, iff its ETCTL restriction is true in that state. Further, if the sub-formula is a Min-max ETCTL formula, then the evaluation algorithm augments the label with the value $EvalS(f, s)$.

Once the first and last non-event states in the transitions of the original timed event structure is redefined as event states, we have full interval independence for Min-max ETCTL formulas. The algorithm shown in figure 4 assumes that such a pre-processing has already been done on the original timed event structure.

We prove the correctness of the algorithm with respect to Monotonic Min-max ETCTL formulas. We establish the correctness of evaluation for the formula $f = QE_C(f_1 U_{Q'} f_2)$. The correctness of evaluation for A formulas follows from the fact that the evaluation procedure for E formulas and A formulas are essentially the same.

Lemma 1. *If Q' is* min, *then a state, t, which cannot be reached from s by a $(f_1 \wedge \neg f_2)$-path is not a closing state of any path in BestP(f,s).*

Table 1. Best path types for $f = QE_C(f_1 U_{Q'} f_2)$.

C-type	Q-type	Q'-type	Best path type from s to t
Increasing	min	min	Shortest length $(f_1 \wedge \neg f_2)$-path
Increasing	min	max	Shortest length f_1-path
Increasing	max	min	Longest length $(f_1 \wedge \neg f_2)$-path
Increasing	max	max	Longest length f_1-path
Decreasing	min	min	Longest length $(f_1 \wedge \neg f_2)$-path
Decreasing	min	max	Longest length f_1-path
Decreasing	max	min	Shortest length $(f_1 \wedge \neg f_2)$-path
Decreasing	max	max	Shortest length f_1-path

Table 2. Best path types for $f = QE_C(f_1 \; U \; T \wedge f_2)$.

C-type	Q-type	Best path type from s to t
Increasing	min	Shortest length f_1-path
Increasing	max	Longest length f_1-path
Decreasing	min	Longest length f_1-path
Decreasing	max	Shortest length f_1-path

Proof. If t cannot be reached from s by an f_1-path, then by definition (semantics of Min-max CTL), t cannot be a closing state. If t can be reached from s by f_1-paths, but not by $f_1 \wedge \neg f_2$-paths, then every f_1-path from s through t has an intermediate state where f_2 is true. Since Q' is min, that intermediate state is the closing state. □

Lemma 2. *If Q' is* max*, then a state, t, which cannot be reached from s by a f_1-path is not a closing state of any path in BestP(f,s).*

Proof. If t cannot be reached from s by an f_1-path, then by definition (semantics of Min-max CTL), t cannot be a closing state. □

Lemma 3. *If Q' is* max *and t is a state which does not belong to any f_1-cycle, and for each successor t' of t, we have $t' \models A(f_1 U f_2)$, then t is not a closing state of any path in BestP(f,s).*

Proof. Since for each successor t' of t, $t' \models A(f_1 U f_2)$, it follows that any f_1-path from s through t in BestP(f,s) will also be an f_1-path from s through some state s' where f_2 holds, and t occurs earlier than s'. Since t does not belong to any f_1-cycle, $s' \neq t$. Since Q' is max, t cannot be a closing state. □

Lemma 4. *If Q' is* max *and t is a state such that there exists a f_1-path from s through t, $t \models f_2$, t belongs to a f_1-cycle, and for each successor t' of t, we have $t' \models A(f_1 U f_2)$, then t cannot be a closing state in any path having finite g-value, and there exists a path from s through t having g-value as ∞.*

Proof. Consider a path, π, having finite g-value. Then there exists a state s' which is the closing state of π. Clearly $s' \neq t$, since for each successor t' of t, $t' \models A(f_1 U f_2)$ and therefore every instance of t' is followed by some other candidate closing state.

Consider an f_1-path from s through t which repeatedly goes around in the f_1-cycle through t. In this path f_1 holds on all states and t occurs infinitely often. By definition, the g-value of such a path is ∞. □

Lemma 5. *If C-type is increasing, and Q-type is* min*, then any φ-path, P, from s through t which is longer than the shortest length φ-path, P^*, from s through t does not belong to BestP(f,s).*

Proof. Since C-type is increasing, the path cost of P^* is less than that of P. Since, Q-type is min, P^* is better than P and hence P cannot belong to BestP(f,s). □

Lemma 6. *If C-type is increasing, and Q-type is* max*, then any φ-path, P, from s through t which is shorter than the longest length φ-path, P^*, from s through t does not belong to BestP(f,s).*

Proof. Since C-type is increasing, the path cost of P^* is greater than that of P. Since, Q-type is max, P^* is better than P and hence P cannot belong to BestP(f,s). □

Lemma 7. *If C-type is decreasing, and Q-type is* min, *then any φ-path, P, from s through t which is shorter than the longest length φ-path, P^*, from s through t does not belong to BestP(f,s).*

Proof. Since C-type is decreasing, the path cost of P^* is less than that of P. Since, Q-type is min, P^* is better than P and hence P cannot belong to BestP(f,s). □

Lemma 8. *If C-type is decreasing, and Q-type is* max, *then any φ-path, P, from s through t which is longer than the shortest length φ-path, P^*, from s through t does not belong to BestP(f,s).*

Proof. Since C-type is decreasing, the path cost of P^* is greater than that of P. Since, Q-type is max, P^* is better than P and hence P cannot belong to BestP(f,s). □

Theorem 2. *Algorithm Evaluate correctly evaluates a Monotonic Min-max ETCTL formula at a state of a timed event structure.*

Proof. We establish the correctness of the algorithm for evaluating a Min-max ETCTL formula, $f = QE_C(f_1 U_{Q'} f_2)$, at a state, s, under the induction hypothesis that the algorithm correctly evaluates the subformulas f_1 and f_2 at all states of the model. Since evaluation for A formulas is exactly similar, the same proof applies to A formulas as well.

The algorithm determines the set of candidate closing states, and then proceeds to determine the g-value of the φ-path of appropriate type (longest or shortest) through the candidate closing states.

By lemmas 1 and 2, we have shown that a state can be a closing state only if it is reachable from s by a φ-path (where φ is defined in step 2.2 of the algorithm). Therefore, in step 2.3, we consider the set of states reachable from s by φ-paths.

By lemma 3, we have shown that if Q' is max and t is a state which does not belong to any f_1-cycle, and for each successor t' of t, $t' \models A(f_1 U f_2)$, then t is not a closing state of any path in BestP(f,s). In step 2.4.1 of the algorithm, we remove all such states from H.

By lemma 4, we have shown that if Q' is max and t is a state which belongs to a f_1-cycle, and for each successor t' of $t, t' \models A(f_1 U f_2)$, then for every path (shortest or longest) through t, either the g-value of the path is ∞, or there is some other closing state. Further we have shown that there exists at least one path through such states with g-value as ∞. Therefore, in step 2.4.2 of the algorithm, we label such states as ∞, and in step 2.5.1 we treat the g-values of paths through these states as ∞.

Lemmas 5–8 establish that only one path through each state in the set H needs to be considered for evaluation, and the path types are as shown in table 1. In step 2.5, for each state in H, the algorithm evaluates the cost $W(t)$ of the best path through t. Since these are the only candidate paths (by lemmas 5-8), the cost of the best path among these is the desired value of $EvalS(f, s)$. In step 2.6, the algorithm assigns the cost of the best path to $EvalS(f, s)$. □

Lemma 9. *The complexity of finding the length of a shortest f-path or a longest f-path from a state s to a state t in a timed model is $O(|\mathcal{R}| + |S| \log |S|)$, where $|S|$ is the number of states in the model and $|\mathcal{R}|$ is the size of the transition relation $\mathcal{R}$.*

Proof. Each f-path from s to t includes only states which are labelled f, and the state t. We first remove from the transition graph those states (except t) which are not labelled f and the set of transitions to and from these states. This can be done in $O(|\mathcal{R}| + |S|)$ time. All paths in the reduced transition graph are f-paths.

Finding the shortest path between a pair of nodes in a graph with non-negative edge costs requires $O(|\mathcal{R}|+|S| \log |S|)$ time where $|\mathcal{R}|$ denotes the number of edges in the graph (Cormen *et al* 1990).

For determining the longest path length, we require to consider the cycles in the graph. If we find a path from s to t through a state j which is self-reachable (that is, j belongs to a f-cycle), then the longest path length from s to t is ∞. Otherwise, we use the algorithm for acyclic graphs. This can be achieved in $O(|\mathcal{R}| + |S|)$ time. □

Theorem 3. *Algorithm Evaluate requires* $O(|f|.|S|^2.(|\mathcal{R}| + |S| \log |S|))$ *time to evaluate a Monotonic Min-max ETCTL formula* f *of length* $|f|$ *on a timed event structure* $J = \langle \mathcal{AP}, S, \mathcal{R}, s_0, \mathcal{L} \rangle$

Proof. Steps 2.3 and 3.2 can be done by a single depth-first traversal in $O(|\mathcal{R}| + |S|)$ time. Step 2.4 requires us to determine whether states in H belong to any φ-cycle. Since the worst case number of states in H is $|S|$, this step can be completed in $O(|S|.(|\mathcal{R}| + |S|))$ time. By virtue of lemma 9, the complexity of steps 2.5.2 and 3.3.2 is $O(|\mathcal{R}| + |S| \log |S|)$. Therefore, the total complexity of step 2.2 to step 2.6 and step 3.2 to step 3.4 is $O(|S|.(|\mathcal{R}|+|S| \log |S|))$. This is the complexity of evaluating the formula at one state when the Min-max values for the subformulas are given. The complexity of evaluating the formula at every state is given by $O(|S|^2.(|\mathcal{R}| + |S| \log |S|))$. By induction on the length of the formula, the complexity of Algorithm Evaluate is $O(|f|.|S|^2.(|\mathcal{R}| + |S| \log |S|))$. □

8. Conclusion

This paper shows that reasoning about the extremal timing properties of events can be done for efficiently than that of general extremal timing queries over timed event structures. The proposed logic, Min-max ETCTL, combines the ideas presented earlier (Dasgupta *et al* 2000, 2001) into a unified logic for reasoning about extremal timing properties of events.

PD acknowledges the partial support of the Indian National Science Academy, New Delhi for this work. PPC acknowledges the partial support of the Department of Science & Technology, Govt. of India, for this work.

References

Alur R 1998 Timed automata. Manuscript: www.cis.upenn.edu/∼alur/Nato97.ps.gz

Alur R, Courcoubetis C, Dill D 1993 Model checking in dense real-time. *Inf. Comput.* 104: 2–34

Alur R, Henzinger T A 1993 Real time logics: Complexity and expressiveness. *Inf. Comput.* 104: 35–77

Alur R, Henzinger T A 1994 A really temporal logic. *J. Assoc. Comput. Mach.* 41: 181–204

Burch J R, Clarke E M, Long D E, McMillan K L, Dill D L 1994 Symbolic model checking for sequential circuit verification. *IEEE Trans. Comput. Aided Design* 13: 401–424

Clarke E M, Kurshan R P 1996 Computer aided verification. *IEEE Spectrum* 33(6): 61–67
Clarke E M, Emerson E A, Sistla A P 1986 Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.* 8: 244–263
Cormen T H, Leiserson C E, Rivest R L 1990 *Introduction to algorithms* (Cambridge, MA: MIT Press and McGraw-Hill)
Dasgupta P, Deka J K, Chakrabarti P P 2000 Model checking on timed event structures. *IEEE Trans. Comput. Aided Design Integrated Circuits Syst.* 19: 601–611
Dasgupta P, Deka J K, Chakrabarti P P, Sriram S 2001 Min-max computation tree logic. *Artif. Intell.* 127: 137–162