Weighted Quantified CTL : An efficient logic for verifying extremal timing queries in Timed Model

Thesis submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology (Honours)

in

Computer Science and Engineering

by

Krishnendu Chatterjee

Under the Guidance of

Dr. P.P.Chakrabarti and Dr.P.Dasgupta

Department of Computer Science and Engineering

Indian Institute of Technology

Kharagpur

May 2001

ACKNOWLEDGEMENT

I fall short of words to acknowledge my deep sense of gratitude to my supervisors **Dr.P.P.Chakrabarti and Dr.P.Dasgupta** for their invaluable guidance,motivation and constant encouragement throughout the course of this work.They have always been friendly and ever ready to spend their precious time to discuss and clear my trivial doubts and have always given invaluable suggestions how to proceed and improve the work. I would like to use this opportunity to extend my thanks to all my batchmates for making the four years here a wonderful experience and help me whenever I needed them.

Krishnendu Chatterjee
Dept. of Computer Sc. and Engg.
Indian Institute of Technology
Kharagpur- 721 302 , India.

Contents

List of Figures

List of Tables

Abstract

Computational Tree Logic(CTL) [7]is one of the most syntactic elegant and computationally attractive temporal logics for branching time model checking. CTL can be verified in time polynomial in the size of state space times the length of the formula . But there are other properties which can be verified in polynomial time but not expressed in CTL. We present a powerful extension of CTL with first order quantification over the set of reachable states. The extended logic QCTL preserves the syntactic elegance of CTL while enhancing its expressive power significantly . The general QCTL model-checking is PSPACE Complete, but a rich fragment of it which is more expressive than CTL can be verified in polynomial time .We then extend this logic over time model. This logic is called Weighted QCTL . While model checking using most timed model is PSPACE Complete or harder [1, 2], many practical timing queries involving extremal (best or worst case) timings can be answered in ploynomial time by quering the system using Weighted QCTL.

Chapter 1

Introduction

As Circuits and Protocols grow more and more complex to meet the ever growing computational needs of people, so does the time required to validate them. The *Pentium Bug* was just an indicator as to how detrimental such undetected design bugs can prove to be if they manage to remain unnoticed in the final product. The goal of automatic-verification is to validate the final products using a set of specifications that such a product should meet.

Over the past decade, *model checking* has evolved as one of the main tools for automated verification of hardware and more generally, concurrent systems. In this paradigm, the circuit or the system is given as a finite state machine and the correctness of a system is specified in the form of a temporal logic formula. The verification problem is then posed as a question as to whether the system satisfies the temporal property. Such a style of reasoning is particularly useful when the specification itself is quite abstract – such as the correctness specifications of PCI or AMBA Bus protocols.

Significant amount of research has been focussed on the design and analysis of temporal logics with respect to their expressibility and complexity of model checking. Two possible classifications of temporal logics are branching time propositional temporal logics such as *Computational Tree Logic*(CTL) and CTL*, and *Linear Time Temporal Logic* (LTL).

The temporal logic, CTL [7] has been particularly popular due to its elegant expressibility and polytime model checking complexity. However, TCTL, the timed extension of this logic (which can express quantitative timing properties) has been shown to be much more complex. While CTL model checking is polynomial in the size of the state space and the length of the formula, TCTL model checking has been shown to be PSPACE Complete [1, 2]. It is therefore an interesting and important objective to study extensions of CTL both in the timed and untimed domain, while preserving its computational efficiency and specificational elegance.

In this thesis, we work towards the development of a logic which has superior expressive powers as compared to CTL, but can still be verified efficiently in polytime. In particular we

focus on the verification of extremal timing properties, and show that a much wider class of problems than those addressed by traditional model checking can be evaluated. For example, we show that several interesting graph theoretic problems can be expressed when we extend CTL to the proposed logic.

1.1 Motivation

The popular temporal logics for model checking are CTL and LTL which are untimed logics. This logics can express several correctness property of a system. CTL is very popular due to its polytime checkablity and syntactic elegance. But there are properties which can be verified in polynomial time but cannot be expressed in CTL. Moreover this logic are not for timing properties of system. RTCTL,TCTL,TLTL are their timed counterparts to reason about the timing properties of a system. For example we have E(p U q) in CTL to denote that whether there is a path where p holds until we reach a state where q holds , where in the timed counterpart we have $E(p\ U_{[a,b]}\ q)$ which denotes that whether there is a path where p holds until we reach a state between time interval a and b where q holds.The complexity of model checking in timed systems is dictated by the magnitudes of the timing constraints and the number of clocks to represent the transition system and the temporal formula [2]. It has been shown that *reachability* in timed automata (which is the fundamental property for verification in timed systems) is rendered PSPACE hard independently by the number of clocks and the magnitude of the timing constraints [4, 10].This is because each can implicitly define an exponential number of possible time regions [2, 4].

But often we are interested to reason about the best and worst case timing properties of a system. We want to reason about the earliest or latest occurence of events. So the extremal or the Min-max properties forms very interesting and important ,practical timing queries on a system. So we try to give a logic for specifying the extremal timing queries of a system which can be verified efficiently.

With CTL formulas there is an implicit existential quantification on the reachable states. If we allow both existential and universal quantifiaction over reachable states can we express more interesting and practical queries which were not possible in CTL and what is the complexity of verification ? So we try to give a logic with first order quantification over reachable states which will have greater expressiblity than CTL along with syntactic elegance and efficient verifiaction algorithm.Then we bring the quantification along with extremal properties and show that many interesting queries can be expressed in the proposed logic.

1.2 Work Done

1.2.1 QCTL

- The QCTL Logic with First Order Quantification over reachable states:Its Syntax and Semantics
- Algorithm for Verification and Proof of its Correctness and Complexity Evaluation of Algorithm

1.2.2 Weighted QCTL

- Weighted QCTL : Timing in QCTL and Model Checking of Extremal Timing Properties:
  Its Syntax and Semantics
- Complexity Anlysis of General Weighted QCTL
- Algorithm for Monotonic Weighted QCTL : Proof of Correctness and Complexity Evaluation
- Showing the expressiblity of the logic with interesting practical queries and interesting Graph theoretic properties

1.2.3 Generalised QCTL

- Generalised QCTL : An generalisaton of QCTL : Complexity Evaluation showing this to be PSPACE Complete

1.2.4 Query Language

- Implementation of the Weighted QCTL as a Query language

1.3 Organization of the thesis

The thesis is organized as follows,

Chapter 2 In this chapter we introduce the basic concepts and the development of Quantified CTL.We also provide its syntax and semantics along with illustrations .We give algorithm for the Quantified CTL and show its correctness and discuss the complexity of Evaluation.

chapter 3 The chapter introduces Weighted Quantified CTL with illustrations.We give the formal syntax and semantics of Weighted Quantified CTL. We reduce *Exact Knapsack Problem* to general Weighted QCTL evaluation showing that it is DP-hard.We also show restrictions of general Weighted QCTL that can be evaluated in polynomial time. We develop algorithms for evaluation of this restriction and prove their correctness.

chapter 4 In this chapter we introduce Generalised Quantified CTL and show that Generalised Quantified CTL is PSPACE Complete.

chapter 5 In this chapter we present the implementation of a query language based on the Weighted Quantified CTL. We explain the data structures and some issues of the implementation.

chapter 6 We give some experimental results of our Query Language based on Weighted QCTL.

chapter 7 We discuss possible extensions and discuss future work.

Chapter 2

Quantified Computational Tree Logic

In Quantified CTL we use Quantification over the states where a formula is true. In CTL we had E and A which are quantification over paths. But here we explore that along with Qunatification over paths if we allow Quantifiacation over states where a fromula is true what properties can be expressed. In CTL there was an implicit existential quantifiaction , but here we allow universal quantifiaction as well. We show that with out logic we can express properties that are not expressible in CTL. Our logic preserves the syntactic elegance of CTL as well.

2.1 Examples

In this section we illustrate the proposed logic QCTL through a couple of motivating examples . Consider the transition system labeled by the atomic proposition :
t (the terminal states of the system) ,
c (control states of the system) and
o (observable states of the system) in the Fig 2.1.

We now illustrate some interesting properties which can be expressed in QCTL but not succinctly in CTL.

- Are all terminal states reachable from s_0 ? We express this in QCTL as :

  $\psi_1 = \forall x \in t$ [E (true U x)]

- Is there a terminal state reachable from s_0 along all paths ? This is done as

  $\psi_2 = \exists x \in$ t [A (true U x)]

- Are allterminal states reachable from s_0 through all paths ?

  $\psi_3 = \forall x \in$ t [A (true U x)]

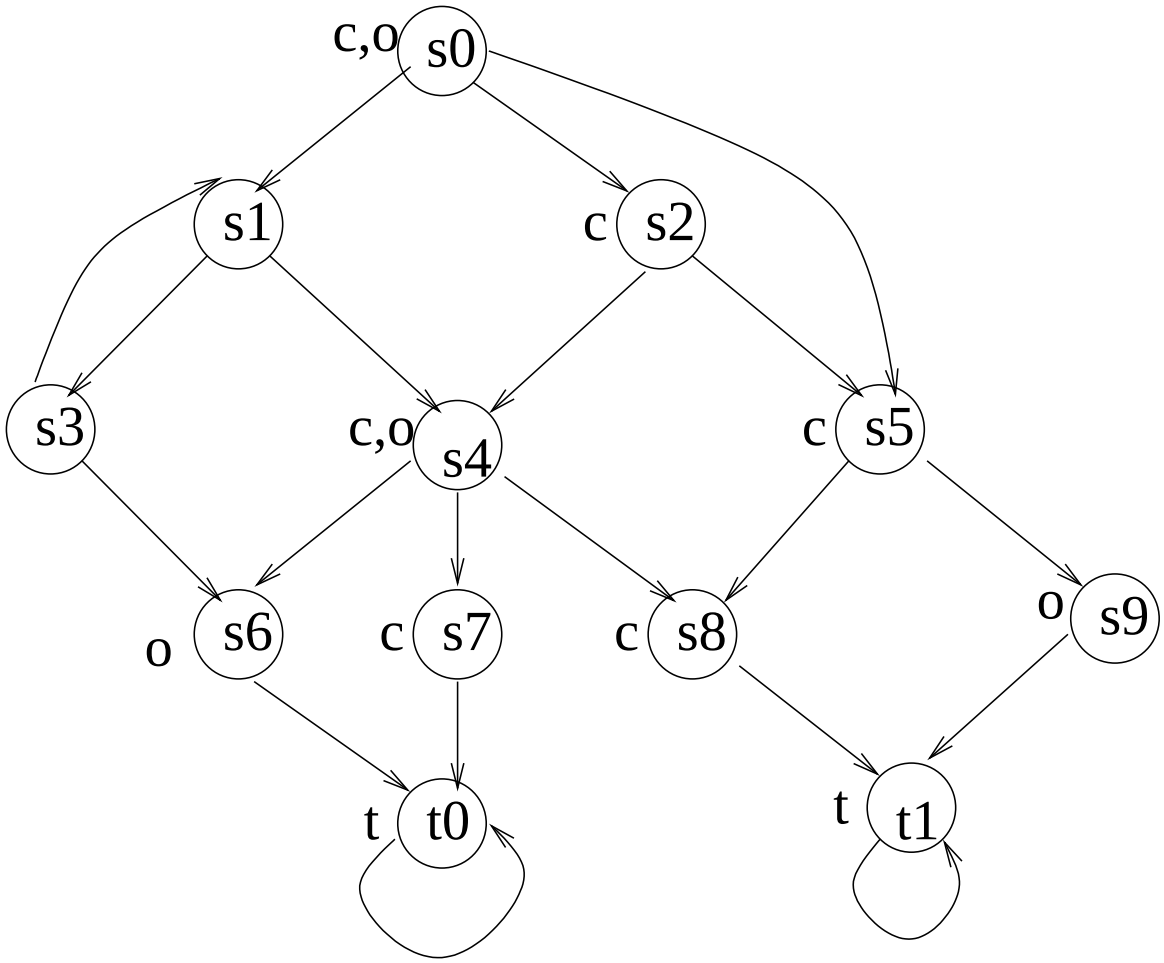

Figure 2.1: Sample transition system

- Is there a state which satisfy ψ_3 and can be reached only through control states ?

  $\psi_4 = \exists x \in (\psi_3)$ [E (c U x)]

KING OF A TOURNAMENT

QCTL can be also useful to express interesting graph theoretic queries . Consider the problem to identify Kings of a tournament graph [16]. A tournament is an orientation of a complete graph (clique). An edge from vertex x_i to vertex x_j indicates that x_i has defeated x_j . A king of a tournament is a vertex x_k such that for every other vertex x_i either there is an edge from x_k to x_i(x_k has defeated x_i) or there is an edge from x_k to some x_j and x_j has an edge to x_i(x_k has defated someone who has defeated x_i). It is easy to see that Kings of a tournament G=(V,E) are exactly those vertices that can be labeled by the following QCTL formula :

$\psi = \forall x \in$ true [x $\vee$ EX(x) $\vee$ EX (EX (x))]

If we wish to determine whether there is any node to have defeated all Kings, it would

be specified as

$\psi_1 = \forall x \in (\psi)$ [EX (x)]

2.2 Syntax and Semantics

In this section we present the formal syntax and semantics of QCTL

2.2.1 Syntax

Here we present the formal syntax and semantics of the QCTL . The formulas are interpreted over finite Kripke Structure J= $\langle \mathcal{AP}, \mathcal{S}, \mathcal{R}, s_0 , \mathcal{F} \rangle$ where

- $\mathcal{AP}$ is the set of atomic propositions
- $\mathcal{S}$ is the finite set of states
- $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S}$ is a transition where $(s_i, s_j) \in \mathcal{R}$ implies s_j is a suceesor of s_i
- $s_0 \in \mathcal{S}$ is the initial state
- $\mathcal{F} : \mathcal{S} \rightarrow 2^{\mathcal{AP}}$is a labeling of states with atomic propositions true in that state.

A path ,π, in the Kripke structure is an infinite sequence of states s_0, s_1 , ... such that for all i, $s_i \in \mathcal{S}$, $\mathcal{R}(s_i, s_{(i+1)})$. s_0 is called the starting state of π . Since the Kripke Structure is finite ,one or more states will appear multiple number of times on a path . In other words a path as defined here is an infinite walk on the state transition graph.

RULE SET 1:

$S ::= \neg S \mid S \wedge S \mid S \vee S \mid AX(\ S\ ) \mid EX(\ S\ ) \mid$
$A(\ S\ U\ S\ ) \mid E(\ S\ U\ S\ ) \mid p$
where p$\in \mathcal{AP}$

RULE SET 2:

$S(x) ::= \neg S(x) \mid S(x) \wedge S(x) \mid S(x) \vee S(x) \mid AX(\ S(x)\ ) \mid EX(\ S(x)\ ) \mid$
$A(\ QS\ U\ S(x)\ ) \mid E(\ QS\ U\ S(x)\ ) \mid S(x) \vee QS \mid S(x) \wedge QS \mid x$
where x is a free variable .

RULE SET 3 :

$QS ::= \exists y \in QS\ [\ S(\ y\ )\ ] \mid \forall y \in QS\ [\ S(\ y\ )\ ] \mid$
$QS \wedge QS \mid QS \vee QS \mid AX(\ QS\ ) \mid EX(\ QS\ ) \mid$

$A(\ QS\ U\ QS)\ \mid\ E(\ QS\ U\ QS)\ \mid\ \neg\ QS\ \mid\ S$

1. Here S are the CTL formulas .
2. S(x) is a Qunatified CTL subformula with a free variable x . When we write $S(x)\ \wedge S(x)$ we mean that both these involve the same free variable x and the composed formula is a Qunatified CTL subformula with the same free variable x.
3. QS is the Quantified CTL subformula with no free vaiable in it .We know the qunatifiers bind a free variable for the formula of the form:

$\forall\ x \in\ QS[\ S(x)\ ]$

$\exists\ x \in\ QS[\ S(x)\ ]$

and other terms do not contain free variables

2.2.2 Semantics

Symbol x is a free variable and S(x) represent QCTL subformula's with free variable. QS denote the QCTL formula which cannot have any free variable, even though subformulas can have.

The truth of a QCTL subformula at a state of the Kripke Structure is subject to an instantiation of the free variable. A free variable x can be instantiated to a state v, in which case we label v with x treating x as an atomic proposition true only in state v. So instantiation is a labeling function $\mathcal{I} : \{x\} \rightarrow \mathcal{S}$ which labels the free variable exactly at one state. For subformulas without free variables , $\mathcal{I}$ is always empty. Given a QCTL formula $\psi(x)$ and an instantiation $\mathcal{I}$ we use $s \models \psi(x)_{\mathcal{I}}$ to denote that $\psi(x)$ is true at state s under instantiation $\mathcal{I}$. If there is no free variable, we use $s \models \psi$ to indicate that ψ is true at s. Likewise a path formula $\psi(x)$ is true in path π under instantiation $\mathcal{I}$ we denote $\pi \models \psi(x)_{\mathcal{I}}$. Thus given a QCTL subformula with a variable and an insatntiation $\mathcal{I}$ we can treat the sub-formula as a formula like CTL with x as an atomic proposition true only at $\mathcal{I}(x)$.

- $\forall s \in\ \mathcal{S}\ \ s \models \mathit{true}$ and $s \not\models$ false
- $s \models p$ iff $p \in \mathcal{F}(s)$
- $s \models x_{\mathcal{I}}$ iff $\mathcal{I}(x) = s$
- $s \models\ \neg\psi(x)_{\mathcal{I}}$ iff $s \not\models \psi(x)_{\mathcal{I}}$
- $s \models\ \neg\psi$ iff $s \not\models \psi$
- $s \models (\psi_1(x)\ \wedge\ \psi_2(x))_{\mathcal{I}}$ iff $s \models \psi_1(x)_{\mathcal{I}}$ and $s \models \psi_2(x)_{\mathcal{I}}$

- $s \models (\psi_1(x) \ \wedge \ \psi_2)_{\mathcal{I}}$ iff $s \models \psi_1(x)_{\mathcal{I}}$ and $s \models \psi_2$
- $s \models \psi_1 \ \wedge \ \psi_2$ iff $s \models \psi_1$ and $s \models \psi_2$
- $s \models (\psi_1(x) \ \vee \ \psi_2(x))_{\mathcal{I}}$ iff $s \models \psi_1(x)_{\mathcal{I}}$ or $s \models \psi_2(x)_{\mathcal{I}}$
- $s \models (\psi_1(x) \ \vee \ \psi_2)_{\mathcal{I}}$ iff $s \models \psi_1(x)_{\mathcal{I}}$ or $s \models \psi_2$
- $s \models \psi_1 \ \vee \ \psi_2$ iff $s \models \psi_1$ or $s \models \psi_2$
- $s \models (EX(\ \psi(x\ )\ )_{\mathcal{I}}$ iff $\exists s_1, \mathcal{R}(s, s_1)$ such that $s_1 \models \psi(x)_{\mathcal{I}}$
- $s \models EX(\ \psi\ )$ iff $\exists s_1, \mathcal{R}(s, s_1)$ such that $s_1 \models \psi$
- $s \models (AX(\ \psi(x\ )\ )_{\mathcal{I}}$ iff $\forall s_1, \mathcal{R}(s, s_1)$ such that $s_1 \models \psi(x)_{\mathcal{I}}$
- $s \models AX(\ \psi\ )$ iff $\forall s_1, \mathcal{R}(s, s_1)$ such that $s_1 \models \psi$
- $\pi \models (\psi_1 \ U \ \psi_2(x))_{\mathcal{I}}$ iff there exist a state $t \in \pi$ such that $t \models \psi_2(x)_{\mathcal{I}}$ and for each state s_i preceding t in π $s_i \models \psi_1$
- $\pi \models \psi_1 \ U \ \psi_2$ iff there exist a state $t \in \pi$ such that $t \models \psi_2$ and for each state s_i preceding t in π $s_i \models \psi_1$
- $s \models (E(\psi_1 \ U \ \psi_2(x)))_{\mathcal{I}}$ iff there exists a path π with s as start state and $\pi \models (\psi_1 \ U \ \psi_2(x))_{\mathcal{I}}$
- $s \models E(\psi_1 \ U \ \psi_2)$ iff there exists a path π with s as start state and $\pi \models (\psi_1 \ U \ \psi_2)$
- $s \models (A(\psi_1 \ U \ \psi_2(x)))_{\mathcal{I}}$ iff forall path π with s as start state and $\pi \models (\psi_1 \ U \ \psi_2(x))_{\mathcal{I}}$
- $s \models A(\psi_1 \ U \ \psi_2)$ iff forall path π with s as start state and $\pi \models (\psi_1 \ U \ \psi_2)$
- $s \models \exists x \in \psi_1 \ [\ \psi_2(x)\ ]$ iff there exists an instantiation $\mathcal{I} : \{x\} \rightarrow \mathcal{S}$ such that $\mathcal{I}(x) \models \psi_1$ and $s \models \psi_2(x)_{\mathcal{I}}$
- $s \models \forall x \in \psi_1 \ [\ \psi_2(x)\ ]$ iff forall instantiation $\mathcal{I} : \{x\} \rightarrow \mathcal{S}$ such that $\mathcal{I}(x) \models \psi_1$ implies $s \models \psi_2(x)_{\mathcal{I}}$

QCTL generalizes the CTL while preserving its decomposable nature and QCTL significantly enhances the power of CTL.

2.3 Model Checking Algorithm

Suppose we wish to determine whether formula f is true at some state of the Kripke structure. Our algorithm operate at stages. At stage 1 we process all QCTL subformulas which are of length 1, at stage 2 we process all QCTL subformulas variable of length 2 and so on. At end of stage i each state is labeled by QCTL subformulas of length less than or equal to i that are true in that state . Whether a formula f is true in state s can be known after $|f|$ steps of our algorithm at which it terminates and $s \models f$ iff state s is marked by f .

Algorithm for verification

1.VerifyS (S formula f)

Use CTL Model Checking Algorithm and if the formula f is true in a state by the CTL Modle checking then it is state s is marked by f.

Algorithm for verification

2.VerifySVariable(S(x) formula $\psi(x)$, Instantiation $\mathcal{I}$)

Let $\mathcal{I}(x) = s_k \ \in \mathcal{S}$

Treat x as an atomic proposition true in s_k and false in all other states . Hence $\psi(x)$ becomes a CTL formula (call it ψ_0)

VerifyS(ψ_0).

Algorithm for verification

3.VerifyQS(QS formula f)

For formulas without quantifiers we can use CTL model checking algo as the Kripke structure is marked with the subfromulas.

For formula $f = \exists \in \psi\ [\psi_1(x)]$ where ψ is a QCTL subformula (length less than f)

Verifyexists(f)

For formula $f = \forall \in \psi\ [\psi_1(x)]$ where ψ is a QCTL subformula (length less than f)

Verifyforall(f)

Algorithm for verification

For formula $f = \exists x \in \psi[\psi_1(x)]$ we write $Arg1(f) = \psi$ and $Arg2(f) = \psi_1(x)$

Verifyexists

1.1 for all state $u \in S$

mark ψ to be false in u.

1.2 for all state $u \in S$

1.3 for each state $v \in S$

if(Arg1(f) is true in v)

let x be an atomic proposition true in v and false in all other in all other state

1.4 VerifySVariable $(Arg2(f), \mathcal{I})$

such that $\mathcal{I}(x) = v$

if($Arg2(f)$ is true in u) mark f to be true in u

Algorithm for verification

Verifyforall

For formula $f = \forall x \in \psi[\psi_1(x)]$ we write $Arg1(f) = \psi$ and $Arg2(f) = \psi_1(x)$

1.1 for all state $u \in S$

mark ψ to be true in u.

1.2 for all state $u \in S$

1.3 for each state $v \in S$

if(Arg1(f) is true in v)

let x be an atomic proposition true in v and false in all other in all other state

1.4 VerifySVariable $(Arg2(f), \mathcal{I})$

such that $\mathcal{I}(x) = v$

if($Arg2(f)$ is false in u) mark u to be false in u

Given a formula f of length $|f|$ we progressively verify formulas from length 1 to $|f|$ and it terminates after we reach length $|f|$. So the formula f is true at a state s if after termination s is marked with f.

Theorem 2.3.1 *Algorithm correctly verifies any QCTL fromula on the Kripke structure.*
Proof: Correctness of Algorithm.
We prove the correctness of the algorithm by induction on the length of the formula. For basis we need to show for the CTL formulas.
Let f be an QS subformula without any quantifiers involved (i.e) no subformula of f can have free variables in it. Since the semantics of QCTL without any free variable coincide with semantics of CTL VerifyS(f) will correctly verify f.
Consider a subformula $\psi(x)$ with free variable x . Under an instantiation $\mathcal{I}$ we know $s \models x_{\mathcal{I}}$ iff $\mathcal{I}(x) = s$ .Hence x acts as an atomic proposition true in $\mathcal{I}(x)$ and false in all other states. As composition of formulas with free variables follows the CTL semantics

VerifySVariable($\psi(x), \mathcal{I}$) will correctly verify a formula with free variable given an instantiation $\mathcal{I}$.
For formula of the form :
$f = \exists\ x \in\ \psi\ [\ \psi_1(x)\ ]$. It is true at a state s iff there is an instantiation $\mathcal{I} : \{x\} \rightarrow \mathcal{S}$ such that $\mathcal{I}(x) \models \psi$ and $s \models \psi_1(x)_{\mathcal{I}}$.
If f is true in s then there is an instantiation to satisy ψ(call it $\mathcal{I}_0$ and $\mathcal{I}_0(x) = v$). Hence ψ is true in v and by induction on the length of the formula ψ is marked in v. Also $s \models \psi_1(x)_{\mathcal{I}_0}$. As VerifySVariable works coorectly then s is marked by f.

Conversely if s is marked by f , then there is a state v where ψ is true and VerifySVariable has marked s with $\psi_1(x)$. Hence there exists an instantiation $\mathcal{I}_0$ such that $\mathcal{I}_0(x) = v$. Hence proved that Verifyexists work correctly.
For formula of the form :
$f = \forall\ x \in\ \psi\ [\ \psi_1(x)\ ]$. It is true at a state s iff forall instantiation $\mathcal{I} : \{x\} \rightarrow \mathcal{S}$ such that $\mathcal{I}(x) \models \psi$ implies $s \models \psi_1(x)_{\mathcal{I}}$.
In other words f is false in a state if there is a state $v \in \mathcal{S}$ such that $v \models \psi$ and with instantiation $\mathcal{I}(x) = v$ and $s \not\models \psi_1(x)_{\mathcal{I}}$ If f is false in s then there is an instantion (call it $\mathcal{I}_0$ and $\mathcal{I}_0(x) = v$). Hence ψ is true in v and by induction on the length of the formula ψ is marked in v. Also $s \not\models \psi_1(x)_{\mathcal{I}_0}$. As VerifySVariable works coorectly then s is not marked by f. Conversely if s is not marked by f , then there is a state v where ψ is true and VerifySVariable has not marked s with $\psi_1(x)$. Hence there exists an instantiation $\mathcal{I}_0$ such that $\mathcal{I}_0(x) = v$. Hence proved that Verifyforall work correctly .
Hence proved that our Algorithm works correctly . □

Theorem 2.3.2 *Our Algorithm requires $O(|\psi|.|S|^2.(|\mathcal{R}| + |S|))$ time to verify a QCTL formula f of length $|f|$ on a Kripke Structure $J = \langle \mathcal{AP}, S, \mathcal{R}, s_0, \mathcal{F} \rangle$ given an instantiation $\mathcal{I}$*
Proof: Consider $\psi = \exists x \in QS[S(x)]$ or $\psi = \forall x \in QS[S(x)]$. Now the step 1.1 will be accomplished in $O(|S|)$ time .Step 1.2 and 1.3 says that there will be atmost $|S|^2$ run of the step 1.4 . CTL model checking Algorithm works in $O(|f|(|\mathcal{R}| + |S|))$ time where f is the length of the formula to verify . Hence by induction hypothesis QS can be verified in $O(Length(QS)(|S|^2.(|\mathcal{R}| + |S|)))$ time .
Hence time to solve ψ is :
$O(Length(QS)(|S|^2.(|\mathcal{R}| + |S|))) + O(|S|^2(Length(S(x)(|\mathcal{R}| + |S|)))$ which is less
$O(Length(\psi)(|S|^2.(|\mathcal{R}| + |S|)))$ time .Hence proved that the verification Algorithm runs in time $O(|\psi|(|S|^2.(|\mathcal{R}| + |S|)))$ time . □

Chapter 3

Weighted Quantified Computataional Tree Logic

QCTL can be used to reason about the temporal behaviour of systems without explicitly quantifying time . This is inadequate to quantitatively raeson about timing properties of a system. In case of Branching Time timed logics, *TCTL* model checking is PSPACE Complete , even in case of discrete timed models. While analysing time transition systems , we often like to reason about the extremal (best or worst case) temporal properties of the system. For example, we may be interested in determining the worst case delay that may occur between a *request* and a *grant* in an arbiter circuit . Here we extend QCTL to Weighted QCTL which allows us the quantification of QCTL state and path properties in terms of a cost function over real time . Weighted QCTL is capable of expressing queries which involve a combination of extremal quantifiers along paths (or walks) in the timed model , as well across paths.

3.1 Examples

In Weighted QCTL if have two special *Until min* and *Until max* operator which indicates the earliest and latest occcurence along a path . We also have operators called MIN and MAX which are used to chose the Minimum and Maximum of a set of values . We now informally describe what their meanings are in our logic.

3.1.1 U_{min} and U_{max} operator

Let us illustrate through an example the meaning of U_{min} and U_{max} operator.

Suppose we are interested in determining the earliest occurence of a state where q is true , starting from a. Using the U_{min} operator indicates the earliest occurence of q along a path.

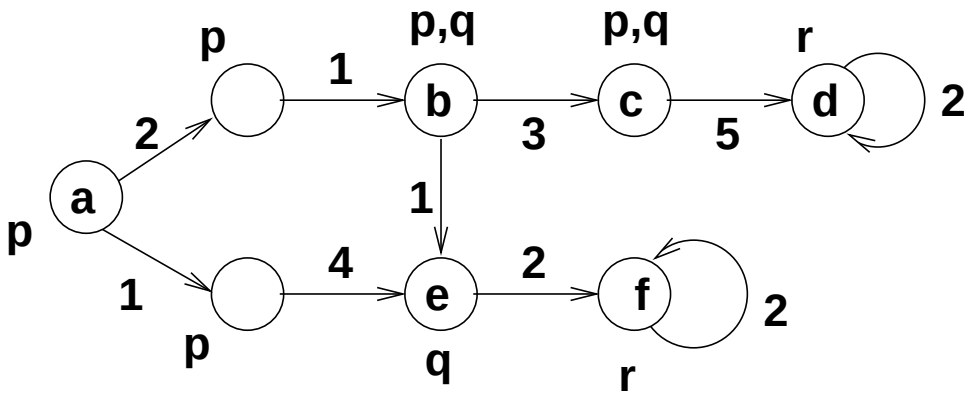

Figure 3.1: A Timed Model

Hence the path formula ($trueU_{min}q$) evaluates to 3 on all paths through b from state a. we refer b as the closing state of these paths .Similarly for all paths through e ($trueU_{min}q$) evalutes to 5 and e is the closing state . The delay from the statrt state to the closing state is called the g-value of tha path.

Using the U_{max} operator indicates the latest occurence of q along a path. For example ($trueU_{max}q$) at state a will evaluate to 6 for c as the closing state and 4 for the path to e through b , and 5 for the path to e through f. It is to be noted that ($trueU_{max}r$) cannot have a latest occurence . Hence it is undefined . Hence for U_{max} closing state can be undefined and the g-value is the ∞.

3.1.2 MIN and MAX Operators

This is to select the value of the path which has the minimum or maximum value as specified by the objective function starting at the given state. That is $MIN\ A_{C(g,h)}(\pi)$ or $MIN\ E_{C(g,h)}(\pi)$ where π is a path formula at a state s_i denotes the value of the minimum of all π paths starting with s_i. The length of the path determines the g-value and h-value is the value associated with the destination or closing state(explained later). So we wish the path length(g) and the value of the destination or closing state of the path such that C(g,h) is minimum. Similarly $MAX\ A_{C(g,h)}(\pi)$ or $MAX\ E_{C(g,h)}(\pi)$ where π is a path formula at a state s_i denotes the value of the maximum of all π paths starting with s_i. The length of the path determines the g-value and h-value is the value associated with the destination or closing state(explained later). So we wish the path length(g) and the value of the destination or closing state of the path such that C(g,h) is maximum. Given a set of values MIN will select the Minimum value of the set and MAX will select the Maximum of the set .

Here we illustrate the proposed logic Weighted QCTL through a couple of motivating examples . Consider the transition system labeled by the atomic proposition

t (the terminal states of the system) ,
c (control states of the system) and
o (observable states of the system) .The delays are shown on the edges of the graph.

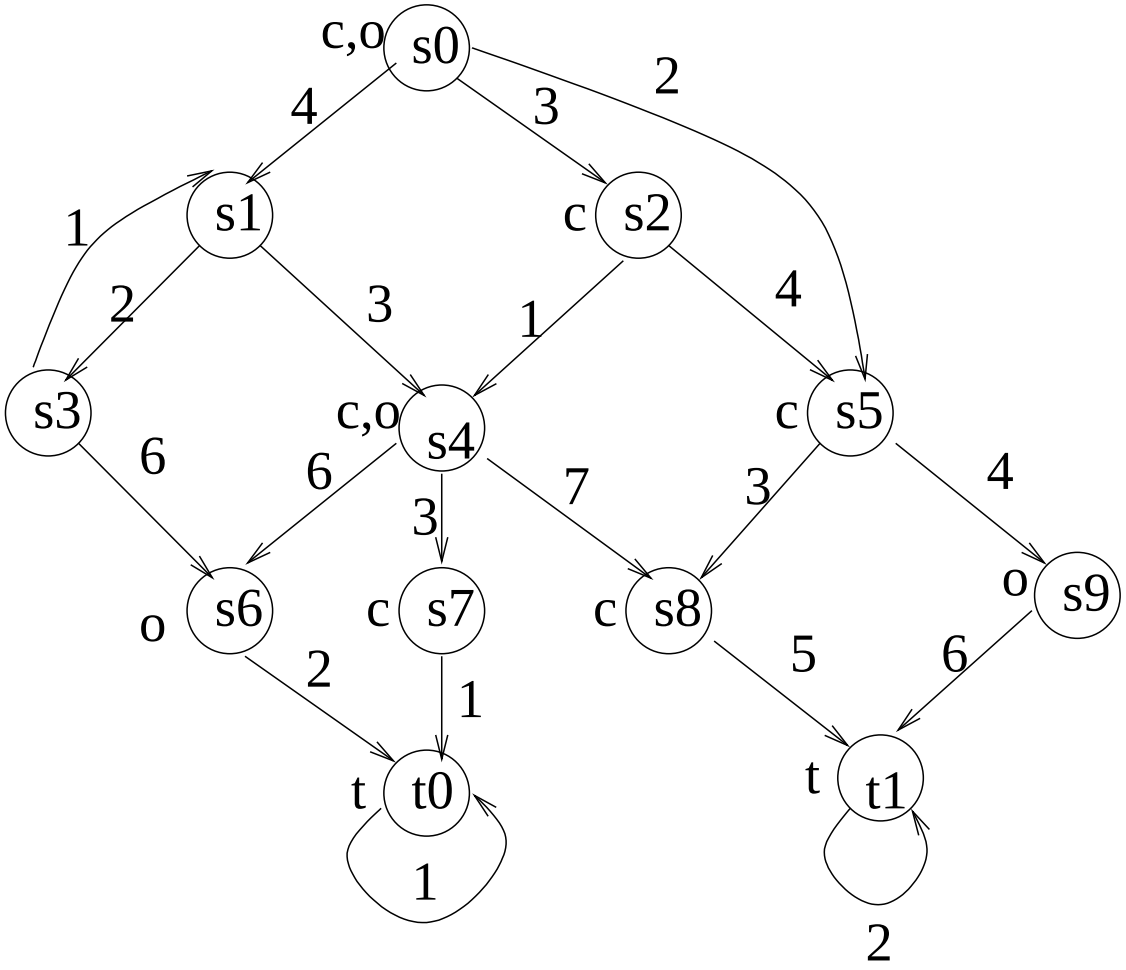

Figure 3.2: Sample transition system

We now illustrate some interesting properties which can be expressed in Weighted QCTL

.

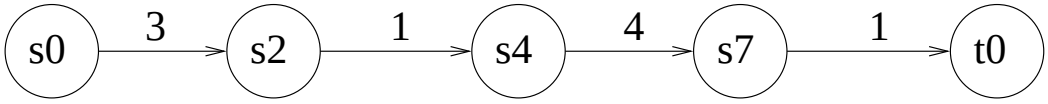

Figure 3.3: The paths satisfying control state until terminal state t0

Illustration As shown in Figure 4. and Figure 5. the minimum length path to t_0 is of length from s_0 8 and to t_1 through control states is 10 from s_0.
Hence $MIN\ E_g(c\ U_{min}\ t_0)$ is 8 and

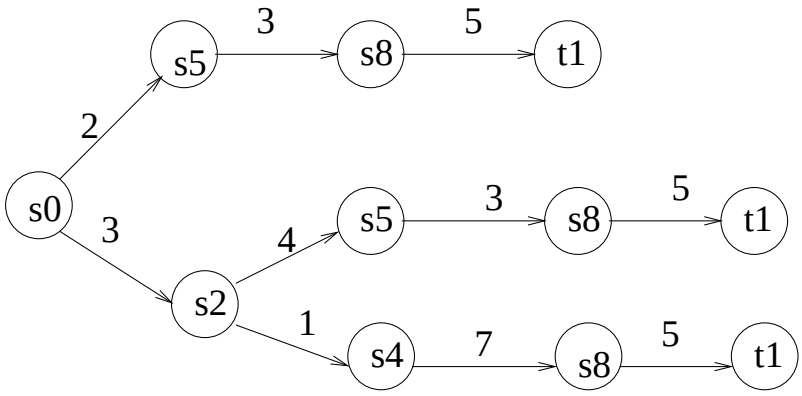

Figure 3.4: The paths satisfying control state until terminal state t1

$MIN\ E_g(c\ U_{min}\ t_1)$is 10 at s_0.

Hence $MIN\ (\forall\ x \in t[MIN\ E_g(c\ U_{min}\ x)]$ is 8 at s_0 .
Hence $MAX\ (\forall\ x \in t[MIN\ E_g(c\ U_{min}\ x)]$ is 10 at s_0 .

As shown in Figure 4. and Figure 5. the maximum length path to t_0 is of length from s_0 8 and to t_1 through control states is 16 from s_0.
Hence $MAX\ E_g(c\ U_{min}\ t_0)$ is 8 and
$MAX\ E_g(c\ U_{min}\ t_1)$is 16 at s_0.
Hence $MIN\ (\forall\ x \in t[MAX\ E_g(c\ U_{min}\ x)]$ is 8 at s_0 .
Hence $MAX\ (\forall\ x \in t[MAX\ E_g(c\ U_{min}\ x)]$ is 16 at s_0 .

Now we illustrate the same formula on a different transition system . Consider the transition system labeled by the atomic proposition
t (the terminal states of the system) ,
c (control states of the system) and
o (observable states of the system) .The delays are shown on the edges of the graph of Figure 3.5.

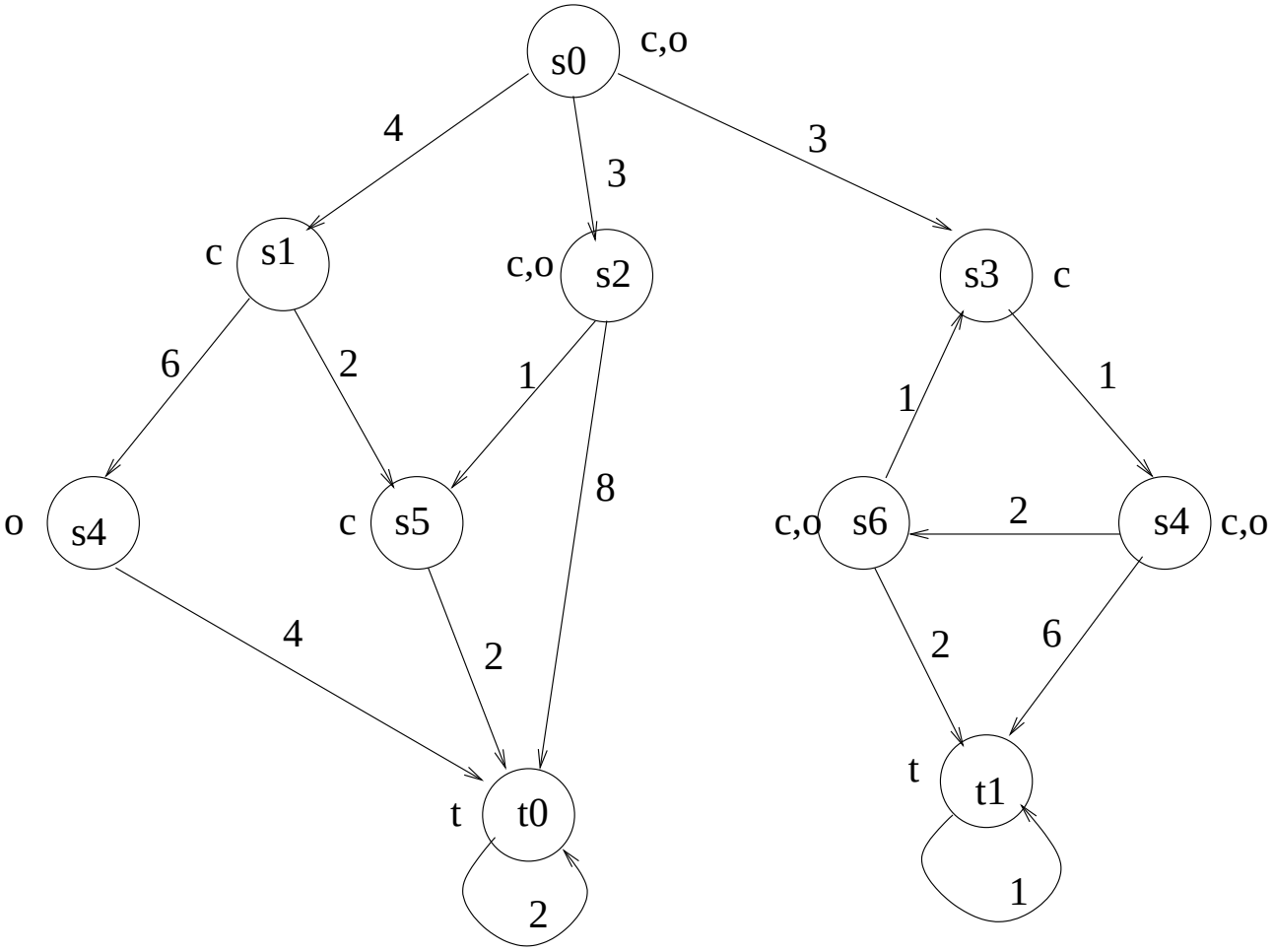

Figure 3.5: Sample transition system

As shown in Figure 3.6 and Figure 3.7 the minimum length path to t_0 is of length from s_0 6 and to t_1 through control states is 8 from s_0.
Hence $MIN\ E_g(c\ U_{min}\ t_0)$ is 6 and
$MIN\ E_g(c\ U_{min}\ t_1)$is 8 at s_0.
Hence $MIN\ (\forall\ x \in t[MIN\ E_g(c\ U_{min}\ x)]$ is 6 at s_0 .
Hence $MAX\ (\forall\ x \in t[MIN\ E_g(c\ U_{min}\ x)]$ is 8 at s_0 .

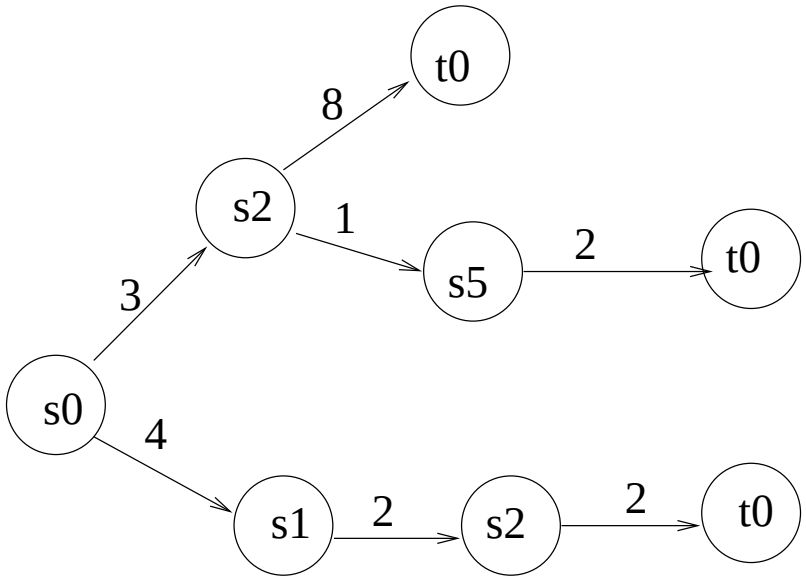

Figure 3.6: The paths satisfying control state until terminal state t0

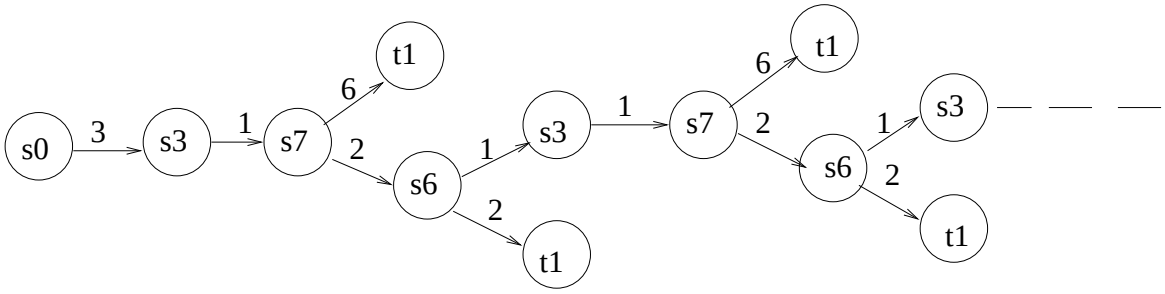

Figure 3.7: The paths satisfying control state until terminal state t1

As shown in Figure 3.6 and Figure 3.7 the maximum length path to t_0 is of length from s_0 11 and to t_1 through control states is ∞ from s_0.
Hence $MAX\ E_g(c\ U_{min}\ t_0)$ is 11 and
$MAX\ E_g(c\ U_{min}\ t_1)$is ∞ at s_0.
Hence $MIN\ (\forall\ x \in t[MAX\ E_g(c\ U_{min}\ x)]$ is 11 at s_0 .
Hence $MAX\ (\forall\ x \in t[MAX\ E_g(c\ U_{min}\ x)]$ is ∞ at s_0 .

Interesting Queries based on the above systems

- If all terminal states are reached by control states , what is the earliest time to reach a terminal state through control states ?
  $\psi = MIN(\forall\ x \in t[MIN\ E_g(c\ U_{min}\ x)])$

- If all terminal states are reached by control states , what is the least delay along worst paths ? (maximum delay)
  $\psi_1 = MIN(\forall\ x \in t[MAX\ E_g(c\ U_{min}\ x)])$

- Of all the terminal states , which are reached along all paths what is the worst case delay along best paths ?
  $\psi_2 = MAX(\exists x\ \in t[MIN\ A_g(cU_{min}x])$

- What is the maximum time upto which one can expect a observable state to be reached (i.e after that time never observable state can be reached)?
  $\psi_3 = MAX\ E_g(trueU_{max}o)$

3.2 Syntax and Semantics

Definition 3.2.1 *Timed Model : A Timed Model is a tuple* $J = \langle \mathcal{AP}, \mathcal{S}, \mathcal{R}, s_0, \mathcal{F} \rangle$ *where :*

- $\mathcal{AP}$ *is the set of atomic proposition*
- $\mathcal{S}$ *is the finite set of states*
- $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S} \times N^+$ *is a transition relation where* N^+ *denote the set of positive integers and* $(s_i, s_j, \delta_{ij}) \in \mathcal{R}$ *implies the delay between the sucessive states* s_i *and* s_j *is* δ_{ij} *units of time*
- $s_0 \in \mathcal{S}$ *is the initial state*
- $\mathcal{F} : \mathcal{S} \rightarrow 2^{\mathcal{AP}}$ *is labeling function of states with atomic propositions true in that state.*

3.2.1 Syntax

RULE SET 1:

$M ::= min \mid max \mid \lambda$

RULE SET 2:

$Ag ::= MIN \mid MAX$

RULE SET 3:

$S ::= \neg S \mid S \wedge S \mid S \vee S \mid AX(\ S\ ) \mid EX(\ S\ ) \mid$
$A(\ S\ U\ S\ ) \mid E(\ S\ U\ S\ ) \mid e_z\ relop\ e_z \mid p$

where p$\in \mathcal{AP}$

RULE SET 4:

$S(x) ::= \neg S(x) \mid S(x) \wedge S(x) \mid S(x) \vee S(x) \mid AX(\ S(x)\ ) \mid EX(\ S(x)\ ) \mid$
$A(\ QS\ U\ S(x)\ ) \mid E(\ QS\ U\ S(x)\ ) \mid S(x) \vee QS \mid S(x) \wedge QS \mid x$

where x is a free variable .

RULE SET 5 :

$QS ::= \exists y \in QS\ [\ S(\ y\ )\ ] \mid \forall y \in QS\ [\ S(\ y\ )\ ] \mid$
$QS \wedge QS \mid QS \vee QS \mid AX(\ QS\ ) \mid EX(\ QS\ ) \mid$
$A(\ QS\ U_M\ QS) \mid E(\ QS\ U_M\ QS) \mid \neg\ QS \mid S$

RULE SET 6:

$Q_z(x) ::= Ag\ E_{C(g,h)}\ (QS\ U_M Q_z(x))$
$\mid Ag\ A_{C(g,h)}\ (QS\ U_M Q_z(x))$
$\mid Ag\ E_{C(g)}\ (QS\ U_M S(x))$
$\mid Ag\ A_{C(g)}\ (QS\ U_M S(x))$
$\mid Q_z(x) \wedge QS$
$\mid Q_z(x) \wedge S(x)$

RULE SET 7:

$QZ ::= Ag(\exists\ x \in QS[\ Q_z(x)\ ])$
$\mid Ag(\forall\ x \in QS[\ Q_z(x)]\ )$
$\mid Ag\ E_{C(g,h)}\ (QS\ U_M QZ)$
$\mid Ag\ A_{C(g,h)}\ (QS\ U_M QZ)$
$\mid Ag\ E_{C(g)}\ (QS\ U_M QS)$
$\mid Ag\ A_{C(g)}\ (QS\ U_M QS)$
$\mid QZ \wedge QS$

RULE SET 8:

$e_z ::= e_z\ op\ e_z \mid QZ \mid n$ where $n \in Z$

RULE SET 9:

$op ::= + | -$

RULE SET 10:

$relop ::= < \mid > \mid <= \mid >=$

RULE SET 11:

$WQCTL ::= QZ \mid QS$

3.2.2 Semantics

In QCTL model checking we seek to determine the truth of a formula at a state of the model. In Weighted QCTL we have extended the syntax allowing the Quantified Until operators(U_{min}, U_{max}) , the MIN ,MAX Operators defined over cost functions C or C'. Thereby Weighted QCTL also has a semantics of evaluation which returns a numeric value whenever a formula is true at a state (given it is a Weigted QCTL formula a evaluation procedure needed). This numeric value quantifies the value of the objective function C or C' which we seek to optimize is the answer to the Weighted QCTL query at that state.If the formula is false at a state of the model then the value returned is *Null*. A Weighted QCTL formula is true at a state if its QCTL restriction is true at that state. The numeric value returned by the evaluation of Weighted QCTL formula f at states of the timed model are determined by the values (h-values) returned by evaluating the subformulas of f at the state of the model and the lengths (namely g-values) computed by the virtue of quantified Until operators over paths of the model. The objective function has two parameters g and h. The g value is associated with the path cost computed by the virtue of quantified Until opertors and h value associated with the evaluation of subformula following quantified Until operators. The user defined function C or C' compute the merits of paths as a function of g-value and h-value.

Let us informally describe what each of the non-terminal represent:
S and $S(x)$ are the CTL with constraints and QCTL subformulas with free variable x respectively.QS are the quantified QCTL formula(with constraints allowed).

$Q_z(x)$: It reperesnts the weighted QCTL sub-formula with value associated with it containing a free variable x.
Consider formulas :
$Ag\ E_{C(g,h)}(QS\ U_M\ Q_z(x))$ or $Ag\ A_{C(g,h)}(QS\ U_M\ Q_z(x))$ if Ag is MIN we are looking for the value of the path that satisfies $QS\ U_M\ Q_z(x)$ and has a g value of g_0 and the closing state has a h-value h_0 and the $C(g_0, h_0) \leq C(g', h')$ of any other path satisfying $QS\ U_M\ Q_z(x)$ and has a g-value g' and h-value h'.

QZ is the weighted QCTL formula .

if Ag is MAX we are looking for the value of the path that satisfies $QS\ U_M\ Q_z(x)$ and has a g value of g_0 and the closing state has a h-value h_0 and the $C(g_0, h_0) \geq C(g', h')$ of any other path satisfying $QS\ U_M\ Q_z(x)$ and has a g-value g' and h-value h'.
QZ is the weighted QCTL formula .

Definition 3.2.2 *QCTL restriction of $QCR(\psi(x))$/ $QCR(\psi)$ of a formula*

The *QCTL-restriction* $QCR(\psi(x)$ / $QCR(\psi)$ of a Weighted QCTL formula $\psi(x)$ / ψ is obtained by dropping all *Until min or Until max operator* ,*MIN* , *MAX operator* and *Cost functions*. Formally :

- If $\psi(x)$ is S(x) or ψ is QS or S then
  $QCR(\psi(x)) = \psi(x)$
  $QCR(\psi) = \psi$.

- $\psi(x) = Ag\ (A/E)_{C/C'}(\psi_1\ U_M\ \psi_2(x)\ )$ or
  $\psi = Ag\ (A/E)_{C/C')}(\psi_1\ U_M\ \psi_2\ )$ then
  $QCR(\psi(x)) = (A/E)(QCR(\psi_1)\ U\ QCR(\psi_2(x))\ )$ or
  $QCR(\psi) = (A/E)(QCR(\psi_1)\ U\ QCR(\psi_2))$

- ψ= Ag $((\exists/\ \forall)x \in QS\ [\ Q_z(x)\ ]$
  $QCR(\psi)$= $((\exists/\ \forall)x \in QCR(QS)\ [\ QCR(Q_z(x))\ ]$

- $\psi(x) = Q_z(x) \wedge S(x)$ then
  $QCR(\psi(x)) = QCR(Q_z(x)) \wedge QCR(S(x))$

- $\psi(x) = Q_z(x) \wedge QS$ then
  $QCR(\psi(x)) = QCR(Q_z(x)) \wedge QCR(QS))$

- $\psi = QZ \wedge QS$ then
  $QCR(\psi) = QCR(QZ) \wedge QCR(QS))$

The QCTL restriction of a Weighted QCTL formula is an untimed QCTL formula whose semantics is defined by QCTL formula over a timed model ignoring delays .

Definition 3.2.3 [Closing state and g-value of a path given an instantiation $\mathcal{I}$:] *Instantiation is a labeling function $\mathcal{I} : \{x\} \rightarrow \mathcal{S}$ which labels a free variable exactly at one state. Given an instantiation $\mathcal{I}$ we treat x as an atomic proposition true only in $\mathcal{I}(x)$ (i.e $s \models x_{\mathcal{I}}$ iff $\mathcal{I}(x) = s$) and $s \models \psi(x)_{\mathcal{I}}$ denote truth of $\psi(x)$ at s under $\mathcal{I}$ and $\pi \models \psi(x)_{\mathcal{I}}$ denote*

the truth of π under $\mathcal{I}$.
We use the notation INS for a instantaited form of a formula:
Given a path formula $\psi(x) = \psi_1\ U\ \psi_2(x)$ and an instantiation $\mathcal{I}$, $INS(\psi) = \psi(x)_{\mathcal{I}}$.
Given a path formula $\psi = \psi_1\ U\ \psi_2$ then $INS(\psi) = \psi$.

Given a QCTL path formula, $\psi(x) = \psi_1\ U\ \psi_2(x)$ and an instantiation $\mathcal{I}$, or $\psi = \psi_1\ U\ \psi_2$ and a path $\pi = \nu_0, \nu_1, \ldots$, where $\pi \models INS(\psi)$, the boolean function $isclosing(i, \pi, INS(\psi))$ is defined as follows :
$isclosing(0, \pi, INS(\psi))$ is true iff $\nu_0 \models \psi_2(x)_{\mathcal{I}}$ or $\nu_0 \models \psi_2$.
For $i > 0$, $isclosing(i, \pi, INS(\psi))$ is true iff $\nu_i \models \psi_2(x)_{\mathcal{I}}$ or $\nu_i \models \psi_2$ and $\forall j,\ 0 \leq j < i\ ,\ \nu_j \models \psi_1$.
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 Weighted QCTL path and instantiation $\mathcal{I}$ formula $\psi(x) = \psi_1\ U_M\ \psi_2(x)$ or $\psi = \psi_1\ U_M\ \psi_2$, and a path π where $\pi \models INS(QCR(\psi))$, the closing state, $CLS(\pi, INS(\psi))$, and the g-value $GVAL(\pi, INS(\psi))$ with respect to ψ are defined as follows: Hence under instantiation $\mathcal{I}$

- If M is min, then let $i = \min\{j :\ isclosing(j, \pi, INS(QCR(\psi)))\}$. Then:

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

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

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

- If M is λ then we have unbounded $Until$ and a state can occur many times in a path . Here the g-values,h-values associated with a path form a set which we call $GHVALSET$ and $GVALSET$ respectively which we define as follows :
  For the path formula $\psi(x) = \psi_1\ U_M\ \psi_2(x)$ and an Instantiation $\mathcal{I}$
  or $\psi = \psi_1\ U_M\ \psi_2$ where the formulas $\psi_2(x)$ or ψ_2 are of the form $Q_z(x)$ and QZ respectively

  $GHVALSET(\pi, INS(\psi))$= $\{\ C(g_i, h_i) | g_i = \sum_{j=0}^{j=i-1} \delta_{j,j+1}$ and $i : v_i \models\ QCR(\psi_2(x))_{\mathcal{I}}$ and $\forall k \leq i\ \ v_k \models\ QCR((\psi_1))$ and $h_i = EvalS(\psi_2(x)_{\mathcal{I}}, v_i)\ \}$

For the path formula $\psi(x) = \psi_1\ U_M\ \psi_2(x)$ and an Instantiation $\mathcal{I}$
or $\psi = \psi_1\ U_M\ \psi_2$ where the formulas $\psi_2(x)$ or ψ_2 are of the form $S(x)$ and QS respectively

$GVALSET(\pi, INS(\psi))$= $\{\ C(g_i) | g_i = \sum_{j=0}^{j=i-1} \delta_{j,j+1}$ and $i : v_i \models\ QCR((\psi_2(x))_{\mathcal{I}}$
and $\forall k \leq i\ \ v_k \models\ QCR(\psi_1)\ \}$

□

The Weighted QCTL 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 Weighted QCTL-formula $\psi(x)$ under an instantiation $\mathcal{I}$ at a state s will be denoted by $EvalS(\psi(x)_{\mathcal{I}}, s)$. The Semantics of $EvalS(\psi(x)_{\mathcal{I}}, s)$ and $EvalS(\psi, s)$ is as follows.
Given formula :
$\psi(x) = Ag\ (A/E)_{C/C'}(\psi_1\ U_M\ \psi_2(x))$ and an instantiation $\mathcal{I}$
or $\psi = Ag\ (A/E)_{C/C'}(\psi_1\ U_M\ \psi_2)$ we use
$Arg1(\psi) = \psi_1$ and
$Arg2(\psi) = \psi_2(x)_{\mathcal{I}}$ or $Arg2(\psi) = \psi_2$ respectively
and $INS(\psi) = \psi(x)_{\mathcal{I}}$ or $INS(\psi) = \psi$ respectively .

- If $s \not\models_{\mathcal{I}} QCR(\psi(x))$, then $EvalS(\psi(x)_{\mathcal{I}}, s) = NULL$.
- If $\psi(x) = Q_z(x) \wedge S(x)$, where $s \models QCR(\psi(x))_{\mathcal{I}}$ then $EvalS(\psi(x)_{\mathcal{I}}, s) = EvalS(Q_z(x)_{\mathcal{I}}, s)$.
- If $\psi(x) = Q_z(x) \wedge QS$, where $s \models QCR(\psi(x))_{\mathcal{I}}$ then $EvalS(\psi(x)_{\mathcal{I}}, s) = EvalS(Q_z(x)_{\mathcal{I}}, s)$.
- If $\psi(x) = Ag\ E_{C(g,h)}(QS\ U_M\ Q_z(x))$, or
  $\psi(x) = Ag\ A_{C(g,h)}(QS\ U_M\ Q_z(x))$ or
  $\psi = Ag\ E_{C(g,h)}(QS\ U_M\ QZ)$, or
  $\psi = Ag\ A_{C(g,h)}(QS\ U_M\ QZ)$ then

  1. If M is min then:
     - If Ag is MIN, then:

       $$EvalS(INS(\psi), s) = \min\{C(GVAL(\pi, INS(\psi)), EvalS(Arg2(\psi), CLS(\pi, INS(\psi)))$$
       among all paths $\pi = \nu_0, \nu_1, \ldots$ starting from $\nu_0 = s\}$

     - If Ag is MAX, then:

       $$EvalS(INS(\psi), s) = \max\{C(GVAL(\pi, INS(\psi)), EvalS(Arg2(\psi), CLS(\pi, INS(\psi))))$$
       among all paths $\pi = \nu_0, \nu_1, \ldots$ starting from $\nu_0 = s\}$

2. If M 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 $isclosing(i, \pi, QCR(INS(\psi)))$ is true, then $CLS(\pi, INS(\psi))$ is not well defined. However, $GVAL(\pi, INS(\psi)) = \infty$. In order to have a well defined semantics, we restrict the set of admissible cost functions to those which have the following limiting behavior:

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

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 $EvalS(\psi(x), s)$ as follows.

- If Ag is MIN, then:

$$EvalS(INS(\psi), s) = \min\{C(GVAL(\pi, INS(\psi)), EvalS(Arg2(\psi), \eta)) \text{ among all paths } \pi = \nu_0, \nu_1, \ldots \text{starting from } \nu_0 = s\}$$

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

- If Ag is MAX, then:

$$EvalS(INS(\psi), s) = \max\{C(GVAL(\pi, INS(\psi)), EvalS(Arg2(\psi), \eta)) \text{ among all paths } \pi = \nu_0, \nu_1, \ldots \text{starting from } \nu_0 = s\}$$

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

3. M is $\lambda(NullString)$

- If Ag is MIN : Given a path π the value of it is
  $EvalS(INS(\psi), s) = \min\ \{\ \min\ \ GHVALSET(\pi, \psi(x))$ among all paths $\pi = v_0, v_1, \ldots.$ with $v_0 = s\ \}$

- If Ag is MAX : Given a path π the value of it is
  $EvalS(INS(\psi), s) = \max\ \{\ \max\ \ GHVALSET(\pi, \psi(x))$ among all paths $\pi = v_0, v_1, \ldots.$ with $v_0 = s\ \}$

- If $\psi = Ag\ A_{C(g)}(QS\ U_M\ S(x))$, or
  $\psi = Ag\ E_{C(g)}(QS\ U_M\ S(x))$ or
  $\psi = Ag\ A_{C(g)}(QS\ U_M\ QS)$, or
  $\psi = Ag\ E_{C(g)}(QS\ U_M\ QS)$ then

1. If Ag is MIN, then

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

2. If Ag is MAX, then

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

3. M is $\lambda(NullString)$
   - If Ag is MIN : Given a path π the value of it is $EvalS(INS(\psi), s) = \min\ \{\ \min\ GVALSET(\pi, INS(\psi))$ among all paths $\pi = v_0, v_1, \ldots.$ with $v_0 = s\ \}$
   - If Ag is MAX : Given a path π the value of it is $EvalS(INS(\psi), s) = \max\ \{\ \max\ \ GVALSET(\pi, INS(\psi))$ among all paths $\pi = v_0, v_1, \ldots.$ with $v_0 = s\ \}$

- $\psi = Ag(\exists x \in QS[Q_z(x)]$ at s or
  $\psi = Ag(\forall x \in QS[Q_z(x)]$ at s
  if $QCR(\psi)$ is false then return $NULL$.
  else define V= $\{\ val_i | val_i = EvalS(Q_z(x)_{\mathcal{I}}, s)$ such that $\mathcal{I}(x) \models QCR(QS)$
  if Ag is MIN chose the minimum of the set V.
  if Ag is MAX chose the maximum of the set V.

From the above semantics it follows that $EvalS(\psi(x), s)_{\mathcal{I}}$ 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 $EvalS(\psi(x), s)_{\mathcal{I}}$ as $BestP(\psi(x), s)_{\mathcal{I}}$. From the above semantics it follows that $EvalS(\psi, s)$ 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 $EvalS(\psi, s)$ as $BestP(\psi, s)$.

The expression with values are represented by e_z . So every e_z has vlue associated with it for a given state s denoted by $val(e_z, s)$. Let us give the semantics :

- $val(e_z, s)$

- $e_z = QZ$
  $val(e_z, s) = EvalS(QZ, s)$

- $e_z = n$
  $val(e_z, s) = value\ of\ the\ numeral\ n$

- $e_z = e_{z1} + e_{z2}$
  $val(e_z, s) = val(e_{z1}, s) + val(e_{z2}, s)$

- $e_z = e_{z1} - e_{z2}$
  $val(e_z, s) = val(e_{z1}, s) - val(e_{z2}, s)$

Now we are ready to give semantics to the constraint part .(i.e S::= $e_z relope_z$)

- $s \models e_{z1} < e_{z2}$ iff $val(e_{z1}, s) < val(e_{z2}, s)$
- $s \models e_{z1} > e_{z2}$ iff $val(e_{z1}, s) > val(e_{z2}, s)$
- $s \models e_{z1} \leq e_{z2}$ iff $val(e_{z1}, s) \leq val(e_{z2}, s)$
- $s \models e_{z1} \geq e_{z2}$ iff $val(e_{z1}, s) \geq val(e_{z2}, s)$

The other semantics follow the same as QCTL semantics .

3.3 Complexity of Evaluation

In this section, we analyze the complexity of evaluating Weighted QCTL formulas over timed models. We first show that the problem is DP-hard in general. Then, we show that for most practical restrictions the problem is solvable in polynomial time.

Theorem 3.3.1 *Evaluating a Weighted QCTL query on a state of a timed model is DP-hard in general.*

Proof: We reduce the EXACT KNAPSACK problem to this problem to show that it is DP-hard. The EXACT KNAPSACK problem is defined as follows. We are given a set of N items, $i_1, i_2, \ldots, i_N$, where the item i_j has a weight, w_j and a profit p_j. Given a sack of capacity M, we are required to determine whether the maximum profit possible (by filling the sack with items without exceeding the sack capacity) is exactly P. The EXACT KNAPSACK problem is known to be DP-complete[12], and can be easily shown to be DP-complete even when $p_j = w_j$ for each item i_j.

Given an instance of EXACT KNAPSACK where each $p_j = w_j$, we create an instance of weighted QCTL query evaluation as follows. Consider the timed model shown in Fig 5.1. The

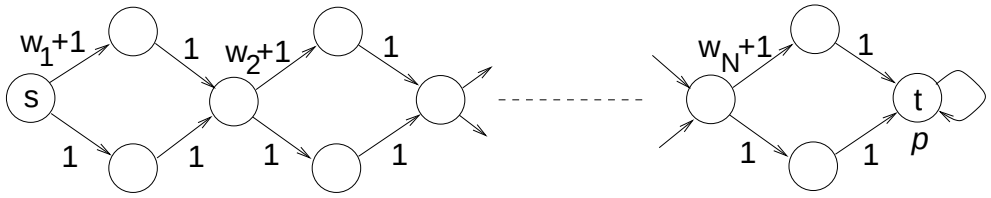

Figure 3.8: Timed model equivalent to EXACT KNAPSACK

transition delays are shown on the edges, and state t is labeled with the atomic proposition p. We are required to evaluate the following formula, f, at state s:

$$f = MAXE_{C(g)}(F_{\min} p) \quad \text{where} \quad C(g) = \begin{cases} g & \text{if } g \leq M + 2N \\ 0 & \text{otherwise} \end{cases}$$

Each path from s through t represents a selection of items whose weights get added in the path delay g. The cost function, $C(g)$, returns a non-zero value for only those paths where the delay g is bounded by $M + 2N$ (that is, the total weight of the selected items is bounded by M). Clearly $EvalS(f, s)$ returns $P^* + 2N$, where P^* is the maximum profit possible. Checking whether P^* is equal to the given value P, yields the answer to the EXACT KNAPSACK problem. Hence EXACT KNAPSACK reduces to Min-max CTL query evaluation. □

3.4 Monotonic Weighted QCTL

A function $C(g)$ is said to be monotonically increasing if

$$C(g_1) < C(g_2) \text{ iff } g_1 < g_2.$$

Likewise, $C(g)$ is called monotonically decreasing if

$$C(g_1) < C(g_2) \text{ iff } g_1 > g_2.$$

A function $C(g, h)$ is said to be monotonic in g if for a fixed value of h, $C(g, h)$ is monotonic in g. Likewise we define monotonicity in h.

A Weighted QCTL -formula is said to be monotonic if its cost function is monotonic in *all* its parameters (g and for two-parameter functions, h) and all its subformulae are monotonic. 'Monotonic Weighted QCTL' is the fragment of Weighted QCTL comprising monotonic Weighted QCTL formulae.

Many temporal queries of practical interest may be expressed elegantly in Weighted QCTL . And as we describe in the following section, they can also be evaluated efficiently.

It is to be recalled that all our cost functions of the form $C(g,h)$ need to satisfy the boundary condition

$$\lim_{g\to\infty} C(g,h) = \lim_{g\to\infty} C(g,k) \qquad \text{where } k \text{ is a constant}$$

This equals ∞ for functions increasing monotonically in g, and 0 for functions decreasing monotonically in g.

Definition 3.4.1 *:[f-path and f-cycle]*

A path π starting at a state s and going through a state s_1 is called a " f-path from s to s_1 " iff the state formula f holds in all states preceding s_1 in π. A f-cycle through a state t is a f-path from t through t .

A shortest length f-path from a state s through a state s_1 is one where s_1 occurs as early as in any other f-path from s through s_1 . Shortest path can be found using standard shortest path algorithms on state transition graph.

If any f-path from s to s_1 contains an intermediate state which is in a f-cycle , then it is possible to have f-paths[actually f-walk in state space] of infinite length from s through s_1. So the value returned is ∞.

Determination of set of states in f-cycles can be done in polynomial time . Finding wherther a f-path exists from s to s_1 via these states can also be done in polynomial time . If no such f-path exists then dropping all states where f is false, we are left with longest path in DAg, which can be done in polynomial time .

Algorithm $Evaluate(f,s)$

We Evaluate a formula f at a state s as follows:

1.Use QCTL model checking algo techniques to label the states of the model
with subformula of the QCTL restricted formula f .During this step,
we ignore delays of the transition of the timed model.

2. The evaluation a t a state s of a Value formula $f = GE_C(f_1 U_M f_2)$
where G is a graph Algo and M is min or max is done as follows . The procedure for evaluating $f = GA_C(f_1 U_M f_2)$ is exactly similar .

2.1Recursively evluate all Value sub-formula of f at
all the states of the model

2.2 If M is min $\psi = f_1 \wedge \neg f_2$ else $\psi = f_1$

2.3 H =set of states labeled f_2 which are reachable along
ψ paths from s .

2.4 If M is Max then :

2.4.1 Remove each state n from H such that

a. n does not belong to ψ cycle and

b. For each sucessor of n_1 of n ,$n_1 \models A(f_1 U f_2)$

2.4.2 Label state n from H with ∞ if

a. n belong to ψ cycle and

b. For each suceesor n_1 of n $n_1 \models A(f_1 U f_2)$

2.5 For each $t \in H$ do

2.5.1 If t is labeled ∞ $g = \infty$
and $W(t) = C(g, h)$ where $h = Evaluate(f_2, t)$

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 $h = Evaluate(f_2, t)$

2.6 If G is MIN $Evaluate(f, s) = min\{W(t)|t \in H\}$ else if
G is MAX $Evaluate(f, s) = max\{W(t)|t \in H\}$

For Weighted Q-CTL part for every instantiation of free variable we get a value for the Value formula . Using Aggregate(Minval and Maxval) of the set of Values we get a value of the Weighted Q-CTL formula .

Conisder formulas
$\psi = Ag(\exists x \in QS[Q_z(x)]$ at s or
$\psi = Ag(\forall x \in QS[Q_z(x)]$ at s
if $QCR(\psi)$ is false then return $NULL$.

C-Type	G-Type	M-Type	Best path type from s to t
Increasing	min	min	Shortest length $(f_1 \wedge \neg f_2)$-path
Increasing	min	max/λ	Shortest length f_1-path
Increasing	max	min	Longest length $(f_1 \wedge \neg f_2)$-path
Increasing	max	max/λ	Longest length f_1-path
Decreasing	min	min	Longest length $(f_1 \wedge \neg f_2)$-path
Decreasing	min	max/λ	Longest length f_1-path
Decreasing	max	min	Shortest length $(f_1 \wedge \neg f_2)$-path
Decreasing	max	max/λ	Shortest length f_1-path

Table 3.1: Best path types for $f = QE_C(f_1 U_{Q'} f_2)$

if Ag is MIN
$EvalS(\psi, s)$= min { $EvalS(Q_z(x)_{\mathcal{I}}, s)$ such that $\mathcal{I}(x) \models QCR(QS)$. }
if Ag is MAX
$EvalS(\psi, s)$= max { $EvalS(Q_z(x)_{\mathcal{I}}, s)$ such that $\mathcal{I}(x) \models QCR(QS)$. }

We prove the correctness of the algorithm with respect to Monotonic Weighted QCTL sub-formulas given an instantiation $\mathcal{I}$. We establish the correctness of evaluation for the formula $f = GE_C(f_1 U_M f_2)$. 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. Given the instantiation $\mathcal{I}$ we treat x as an atomic proposition true only in the state $\mathcal{I}(\S)$.

Lemma 3.4.1 *If M 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 $BestP(f, s)_{\mathcal{I}}$.*
Proof: If t cannot be reached from s by a f_1-path, then by definition (semantics of Weighted QCTL), 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 M is min, that intermediate state is the closing state. □

Lemma 3.4.2 *If M 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 $BestP(f, s)_{\mathcal{I}}$.*
Proof: If t cannot be reached from s by a f_1-path, then by definition (semantics of Weighted QCTL), t cannot be a closing state. □

Lemma 3.4.3 *If M 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 $t' \models_{\mathcal{I}} A(f_1 U f_2)$, then t is not a closing state of any path in $BestP(f,s)_{\mathcal{I}}$.*

Proof: Since for each successor t' of t, $t' \models_{\mathcal{I}} A(f_1 U f_2)$, it follows that any f_1-path from s through t in $BestP(f,s)_{\mathcal{I}}$ 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 M is max, t cannot be a closing state. □

Lemma 3.4.4 *If M is* max *and t is a state such that there exists a f_1-path from s through t, $t \models_{\mathcal{I}} f_2$, t belongs to a f_1-cycle, and for each successor t' of t, we have $t' \models_{\mathcal{I}} A(f_1 U f_2)$, 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, $t' \models_{\mathcal{I}} A(f_1 U f_2)$ and therefore every instance of t' is followed by some other candidate closing state.

Consider a 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 3.4.5 *If C-type is increasing, and G-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 $BestP(f,s)_{\mathcal{I}}$.*

Proof: Since C-type is increasing, the path cost of P^* is less than that of P. Since, G-type is MIN, P^* is better than P and hence P cannot belong to $BestP(f,s)_{\mathcal{I}}$. □

Lemma 3.4.6 *If C-type is increasing, and M-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 $BestP(f,s)_{\mathcal{I}}$.*

Proof: Since C-type is increasing, the path cost of P^* is greater than that of P. Since, M-type is max, P^* is better than P and hence P cannot belong to $BestP(f,s)_{\mathcal{I}}$. □

Lemma 3.4.7 *If C-type is decreasing, and G-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 $BestP(f,s)_{\mathcal{I}}$.*

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

Lemma 3.4.8 *If C-type is decreasing, and G-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 $BestP(f,s)_{\mathcal{I}}$.

Proof: Since C-type is decreasing, the path cost of P^* is greater than that of P. Since, G-type is MAX, P^* is better than P and hence P cannot belong to $BestP(f,s)_{\mathcal{I}}$. □

Similar Lemmas hold for BestP(f,s) (also follows from Min-Max CTL algorithm proof).

Theorem 3.4.1 *Algorithm Evaluate correctly evaluates a Monotonic Weighted QCTL formula at a state of a timed model given an instantiation* $\mathcal{I}$.

Proof: We establish the correctness of the algorithm for evaluating a Weighted QCTL formula, $f = GE_C(f_1 U_M f_2)$, 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 Lemma 3.4.1 and Lemma 3.4.2, 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 3.4.3, we have shown that if M is max and t is a state which does not belong to any f_1-cycle, and for each successor t' of t, $t' \models_{\mathcal{I}} A(f_1 U f_2)$, then t is not a closing state of any path in $BestP(f,s)_{\mathcal{I}}$. In Step 2.4.1 of the algorithm, we remove all such states from H.

By Lemma 3.4.4, we have shown that if M is max and t is a state which belongs to a f_1-cycle, and for each successor t' of t, $t' \models_{\mathcal{I}} A(f_1 U f_2)$, 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 ∞.

Lemma 3.4.5, Lemma 3.4.6, Lemma 3.4.7 and Lemma 3.4.8 establishes 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 3.1. 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 3.4.5-3.4.8), the cost of the best path among these is the desired value of $EvalS(f,s)_{\mathcal{I}}$. In Step 2.6, the algorithm assigns the cost of the best path to $EvalS(f,s)_{\mathcal{I}}$. □

Lemma 3.4.9 *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 labeled f, and the state t. We first remove from the transition graph those states (except t) which are not labeled 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[9].

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. $\square$

Theorem 3.4.2 *Algorithm Evaluate requires $O(|f|.|S|^2.(|\mathcal{R}| + |S| \log |S|))$ time to evaluate a Monotonic Weighted QCTL formula f of length $|f|$ on a timed model $J = \langle \mathcal{AP}, S, \mathcal{R}, s_0, \mathcal{L} \rangle$ given an instantiation $\mathcal{I}$*

Proof: Step 2.3 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 3.4.9, the complexity of Step 2.5.2 is $O(|\mathcal{R}| + |S| \log |S|)$. Therefore, the total complexity of Step 2.2 to Step 2.6 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 $O(|f|.|S|^2.(|\mathcal{R}| + |S| \log |S|))$. $\square$

Theorem 3.4.3 *Algorithm to Evaluate a Weighted QCTL formula requires $O(|f|.|S|^3.(|\mathcal{R}| + |S| \log |S|))$ time to evaluate a Monotonic Weighted QCTL formula f of length $|f|$ on a timed model $J = \langle \mathcal{AP}, S, \mathcal{R}, s_0, \mathcal{L} \rangle$ given an instantiation $\mathcal{I}$*

Proof: There can be at most $|\mathcal{S}|$ many instantiations . For each instantiation Evaluate takes $O(|f|.|\mathcal{S}|^2.(|\mathcal{R}| + |\mathcal{S}| \log |\mathcal{S}|))$ time by the above Theorem .Hence the complexity of Evaluation of a value of Weighted QCTL formula takes $O(|f|.|\mathcal{S}|^3.(|\mathcal{R}| + |\mathcal{S}| \log |\mathcal{S}|))$ time $\square$

3.5 Examples

This section illustrates problems of practical interest where Weighted QCTL is useful to evaluate extremal timing properties. The first example relates to path planning, the second relates to reasoning about grant arbiter.

3.5.1 Travel planning

We are given the air maps, rail maps and road maps of a country. Each map shows the connections between the cities, as well as the time required on each connection. It is easy to see that the information may be represented by a three layered graph, where the layers correspond to the air, rail and road graphs respectively.Every city is marked with the state it belongs to. So the State acts as atomic propositon by which the cities are marked as well.

QUERY1

- Quickest time to reach a city from which a city c_i by air can be reached before a city in Bengal can be reahed by air
  $f1 = MIN\ E_g(airU_{min}c_i)$
  $f2 = MIN\ E_g(airU_{min}Bengal)$
  $f = MIN\ E_g(trueU_{min}(f1 < f2))$

- Quickest time to reach a city from which a city c_i by air can be reached before all cities in Bengal by air

  $f1 = MIN\ E_g(airU_{min}c_i)$
  $f2 = MAX(\forall x \in Bengal[MIN\ E_g(airU_{min}x)])$
  $f = MIN\ E_g(trueU_{min}(f1 < f2))$

QUERY2

- How fast can a city of Bengal be reached by air that is also reachable by rail

  $f = MIN(\exists x \in Bengal[MIN\ E_g(airU_{min}x) \wedge E(railUx)])$

QUERY3

- What is the worst case time of reaching a city from where every path to a city of Bengal is either through rail or road

  $f = MAX\ E_g(trueU_{min}(\neg\exists x \in Bengal[E(trueU\neg(rail \vee road) \wedge E(trueUx) \wedge \neg x)]))$

3.5.2 Grant Arbiter

- We have *arbiter and grant* circuit . There are two types of request *req1 and req2* . There are some states which acts as *grants* . We present some interesting properties .

– What is the earliest time to get *grants* for *req1* from all granting states ?
$\psi_1 = MAX(\forall x \in grants[MIN\ E_g(req1\ U_{min}\ x)])$

– What is the earliest time to get *grants* for *req1* from any granting states ?
$\psi_2 = MIN(\exists x \in grants[MIN\ E_g(req1\ U_{min}\ x)])i$

– What is the quickest time to get reply from a *grant* state to *req1* that can also grant requests of *req2*
$psi_3 = MIN(\exists x \in grants[MIN\ E_g(req1 U_{min} x) \wedge E(req2 U x)])$

– What is the worst case delay of getting reply from all grants ?
$\psi_4 = MAX(\forall x \in grants[MAX\ E_g(req1\ U_{min}\ x)])$

– What is the worst case delay of getting reply from any grants ?

$\psi_4 = MIN(\exists x \in grants[MAX\ E_g(req1 U_{min}\ x)])$

3.5.3 NETLIST DATABASE QUERY

We have a netlist database where the nodes of the graph represnts modules , gates and can be marked with property like whetehr it is an AND , OR , NOT or NAND gate, whether it is a transmission gate , or a voltage repeater etc. On the following Netlist Graph we can have several queries.

- For Fault Tolerance we often use redundancy. So someone has designed a circuit with NAND gates and the same module with AND,OR and NOT gates. Now he is interested to know whether the fastest path to output in NAND realization is faster than the AND , OR and NOT realization. The following query express this:

$MIN\ E_g\ (NAND\ U\ output) < MIN\ E_g((AND \vee OR \vee NOT)\ U\ output)$

If the designer is interested to know whether the path with max delay in NAND realization is faster compared to the fastest in AND , OR and NOT realization his query would be :

$MAX\ E_g\ (NAND\ U\ output) < MIN\ E_g((AND \vee OR \vee NOT)\ U\ output)$

- In circuits the Transmission gates consumes very less power. So a designer may be interested to know whether there is a path through transmission gates to all the outputs serving the minimum delay less than a constant 'k'? This is expressed as :

  $MAX(\forall\ x\ \in\ output[MIN\ E_g(transmissiongate\ U\ x]) \leq k$

- It often requires that if a signal lives for long it will have to be strengthened in between. So a designer may query is it possible to reach the outputs in less than 'k' time delay or it can reach the output within 'k_1' units of delay provided it has gone through a "Voltage Enhancer" (" Repeater") ? This is represented as follows:

  $(MIN\ E(true\ U_{min} output) \leq\ k\ ) \vee (MIN\ E_{g+h}(true\ U_{min}\ ((MIN\ E_g(true U_{min} output) \wedge repeater)) \leq\ k_1)$

- A designer may want to find the minimum of the maximum delay to any output node that does not go through a node of property 'p' but all output must be reachable ? This is expressed as follows:

  $MIN\ (\forall\ x\ \in output[MAX\ E_g((true \wedge \neg p) U\ x)])$

  Property 'p' can be a "Clocked Input " or any other complex property verified as a nested query.

- The designer may also want to know what is the Minimum delay to a gate , which is the last occurence in a path through AND nodes and all outputs are reachable from it ?

  $MIN E_g(AND U_{max}(\forall x \in output[E(true U x)])$

Definition 3.5.1 *Given a graph $G = (V, E)$ where $E \subseteq V \times V \times N^+$ and some node marked as frontiers the node from which the all the frontiers can be reached in minimum time is the Centre of the Graph.*

Let us see what it can mean in real world . Some we give some information to the node in a graph , it passes the information to the neighbours but the passing of the information takes time equal to the specified edge weights . We want to give some information to a node and want them to reach all frontiers in the minimum time possible. So what we do is add a node called Aux to the graph and add edge (Aux,v,1) $\forall\ v \in V$ and thus the new graph is $G' = (V \cup \{Aux\}, E \cup E')$ where $E' = \{(Aux, v, 1) | \forall\ v \in\ V\}$. Suppose we want to know what is the time to reach all the frontiers from the Centre of the graph (G) . We evaluate the query ψ_2 in the state Aux and the result is the evaluated value .

$\psi_1 = MAX(\forall y \in fr[MIN\ E_g(true\ U_{min}\ y)]);$

$\psi_2 = MIN(\exists z \in (\psi_1)[MIN\ E_h(true\ U_{min}\ z)]);$

Thus we can capture interesting Graph Theoretic properties in our logic as well.

3.6 Appendix

Details of QCTL Restriction of Weighted QCTL formula

- If $\psi(x)$ is S(x) or ψ is QS or S then
  $QCR(\psi(x)) = \psi(x)$
  $QCR(\psi) = \psi$.
- $\psi(x) = Ag\ A_{C(g,h)}(QS\ U_M\ Q_z(x)\ )$ then
  $QCR(\psi(x)) = A(QCR(QS)\ U\ QCR(Q_z(x))\ )$
- $\psi(x) = Ag\ E_{C(g,h)}(QS\ U_M\ Q_z(x)\ )$ then
  $QCR(\psi(x)) = E(QCR(QS)\ U\ QCR(Q_z(x))\ )$
- $\psi = Ag\ A_{C(g,h)}(QS\ U_M\ QZ\ )$ then
  $QCR(\psi) = A(QCR(QS)\ U\ QCR(QZ)\ )$

- $\psi = Ag\ E_{C(g,h)}(QS\ U_M\ QZ\ )$ then
  $QCR(\psi) = E(QCR(QS)\ U\ QCR(QZ)\ )$

- $\psi(x) = Ag\ A_{C'(g)}(QS\ U_M\ S(x))$ then
  $QCR(\psi(x)) = A(QCR(QS)\ U\ QCR(S(x)))$

- $\psi(x) = Ag\ E_{C'(g)}(QS\ U_M\ S(x))$ then
  $QCR(\psi(x)) = E(QCR(QS)\ U\ QCR(S(x)))$

- $\psi = Ag\ A_{C'(g)}(QS\ U_M\ QS)$ then
  $QCR(\psi) = A(QCR(QS)\ U\ QCR(QS))$

- $\psi = Ag\ E_{C'(g)}(QS\ U_M\ QS)$ then
  $QCR(\psi) = E(QCR(QS)\ U\ QCR(QS))$

- $\psi(x)$= Ag $(\exists x \in QS\ [\ Q_z(x)\ ]$
  $QCR(\psi(x))$= $(\exists x \in QCR(QS)\ [\ QCR(Q_z(x))\ ]$

- ψ= Ag $(\forall x \in QS\ [\ Q_z(x)\ ]$
  $QCR(\psi)$= $(\forall x \in QCR(QS)\ [\ QCR(Q_z(x))\ ]$

- $\psi(x) = Q_z(x) \wedge S(x)$ then
  $QCR(\psi(x)) = QCR(Q_z(x)) \wedge QCR(S(x))$

- $\psi(x) = Q_z(x) \wedge QS$ then
  $QCR(\psi(x)) = QCR(Q_z(x)) \wedge QCR(QS)$

Chapter 4

Generalised QCTL

In this chapter we present the formal syntax and semantics of QCTL in its general form. In next we prsent the formal syntax and semantics of Generalised QCTL and in the later Section we shall establish that QCTL model checking is PSPACE-complete in general.

4.1 Syntax and semantics of Generalised QCTL

The truth of QCTL formulas are interpreted over a finite Kripke structure, $J = \langle \mathcal{AP}, S, \mathcal{R}, s_0, \mathcal{F} \rangle$, where:

- $\mathcal{AP}$ is a set of atomic propositions,
- S is a finite set of states,
- $\mathcal{R} \subseteq S \times S$ is a transition relation, where $(s_i, s_j) \in \mathcal{R}$ implies that s_j is a successor state of the state s_i.
- $s_0 \in S$ is the initial state,
- $\mathcal{F} : S \rightarrow 2^{\mathcal{AP}}$ is a labeling of states with atomic propositions true in that state.

A *path*, π, in the Kripke structure is an infinite sequence of states, $s_0, s_1, \ldots$, such that for all i, $s_i \in S$ and $\mathcal{R}(s_i, s_{i+1})$. s_0 is called the starting state of π. Since the Kripke structure has a finite set of states, one or more states 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. The formal grammar of QCTL is as follows:

$$\begin{aligned} C(\mathcal{L}) \;=\; & \neg C(\mathcal{L}) \mid C(\mathcal{L}) \wedge C(\mathcal{L}) \mid C(\mathcal{L}) \vee C(\mathcal{L}) \mid AX(C(\mathcal{L})) \mid EX(C(\mathcal{L})) \\ & \mid A(C(\mathcal{L})\ U\ C(\mathcal{L})) \mid E(C(\mathcal{L})\ U\ C(\mathcal{L})) \mid p \quad \text{where } p \in \mathcal{AP} \cup \mathcal{L} \end{aligned}$$

$$
\begin{aligned}
Q(\mathcal{L}) &= \exists x \in Q(\mathcal{L})\ [Q(x||\mathcal{L})] \quad \text{where } x \notin \mathcal{L} \\
&\mid \forall x \in Q(\mathcal{L})\ [Q(x||\mathcal{L})] \quad \text{where } x \notin \mathcal{L} \\
&\mid Q(\mathcal{L}) \wedge Q(\mathcal{L}) \mid Q(\mathcal{L}) \vee Q(\mathcal{L}) \mid AX(Q(\mathcal{L})) \mid EX(Q(\mathcal{L})) \\
&\mid A(Q(\mathcal{L})\ U\ Q(\mathcal{L})) \mid E(Q(\mathcal{L})\ U\ Q(\mathcal{L})) \mid \neg Q(\mathcal{L}) \mid C(\mathcal{L}) \\
QCTL &= Q([]) \quad \text{where [] stands for an empty list}
\end{aligned}
$$

The symbol $\mathcal{L}$ represents a list of *free* variables. $C(\mathcal{L})$ and $Q(\mathcal{L})$ respectively represent CTL and QCTL sub-formulas with free variables in $\mathcal{L}$. The string $x||\mathcal{L}$ represents the appending of the variable x in the list $\mathcal{L}$. The non-terminal $QCTL$ represents QCTL formulas. It should be noted that $QCTL = Q([])$, that is, a QCTL formula cannot have a free variable ($\mathcal{L}$ is empty), though its subformulas may have free variables. $C([])$ represents traditional CTL formulas (without free variables).

The truth of a QCTL sub-formula, $Q(\mathcal{L})$, at a state of the Kripke structure is subject to an *instantiation* of the free variables in its list $\mathcal{L}$. A free variable $x \in \mathcal{L}$ can be instantiated to a state ν, in which case we *label* ν with x and treat x as an atomic proposition true only in ν. Thus an instantiation of $\mathcal{L}$ can be viewed as a labeling function $\mathcal{I} : \mathcal{L} \to S$, where each label $x \in \mathcal{L}$ can be used on at most one state. For QCTL subformulas without free variables, $\mathcal{I}$ is always empty. Given a QCTL sub-formula $\varphi(\mathcal{L})$ and an instantiation $\mathcal{I}$, we use the notation $s \models_{\mathcal{I}} \varphi(\mathcal{L})$ to indicate that $\varphi(\mathcal{L})$ is true at state s under the instantiation $\mathcal{I}$. If $\mathcal{L}$ is empty, then we use the notation $s \models \varphi$ to indicate that φ is true at s. Likewise if a path formula ψ is true in a path π under instantiation $\mathcal{I}$, we denote this by $\pi \models_{\mathcal{I}} \psi$. The semantics of QCTL is as follows.

- $\forall s \in S, s \models True$ and $s \not\models False$
- $s \models p$ iff $p \in \mathcal{F}(s)$
- $s \models_{\mathcal{I}} x$ iff $\mathcal{I}(x) = s$
- $s \models_{\mathcal{I}} \neg\varphi(\mathcal{L})$ iff $s \not\models_{\mathcal{I}} \varphi(\mathcal{L})$
- $s \models_{\mathcal{I}} \varphi_1(\mathcal{L}_1) \wedge \varphi_2(\mathcal{L}_2)$ iff $s \models_{\mathcal{I}_1} \varphi_1(\mathcal{L}_1)$ and $s \models_{\mathcal{I}_2} \varphi_2(\mathcal{L}_2)$, where $\mathcal{I}_1$ (resp. $\mathcal{I}_2$) is $\mathcal{I}$ with domain restricted to $\mathcal{L}_1$ (resp. $\mathcal{L}_2$).
- $s \models_{\mathcal{I}} \varphi_1(\mathcal{L}_1) \vee \varphi_2(\mathcal{L}_2)$ iff $s \models_{\mathcal{I}_1} \varphi_1(\mathcal{L}_1)$ or $s \models_{\mathcal{I}_2} \varphi_2(\mathcal{L}_2)$, where $\mathcal{I}_1$ (resp. $\mathcal{I}_2$) is $\mathcal{I}$ with domain restricted to $\mathcal{L}_1$ (resp. $\mathcal{L}_2$).
- $s \models_{\mathcal{I}} EX\varphi(\mathcal{L})$ iff $\exists s'$, $\mathcal{R}(s, s')$ such that $s' \models_{\mathcal{I}} \varphi(\mathcal{L})$
- $s \models_{\mathcal{I}} AX\varphi(\mathcal{L})$ iff $\forall s'$, $\mathcal{R}(s, s')$ we have $s' \models_{\mathcal{I}} \varphi(\mathcal{L})$

- $\pi \models_{\mathcal{I}} \varphi_1(\mathcal{L}_1)\ U\ \varphi_2(\mathcal{L}_2)$ iff there exists a state $t \in \pi$ such that $t \models_{\mathcal{I}_2} \varphi_2(\mathcal{L}_2)$, and for each state s_i preceding t in π, $s_i \models_{\mathcal{I}_1} \varphi_1(\mathcal{L}_1)$, where $\mathcal{I}_1$ (resp. $\mathcal{I}_2$) is $\mathcal{I}$ with domain restricted to $\mathcal{L}_1$ (resp. $\mathcal{L}_2$).

- $s \models_{\mathcal{I}} E(\varphi_1(\mathcal{L}_1)\ U\ \varphi_2(\mathcal{L}_2))$ iff there exists a path π starting at s such that $\pi \models_{\mathcal{I}} \varphi_1(\mathcal{L}_1)\ U\ \varphi_2(\mathcal{L}_2)$.

- $s \models_{\mathcal{I}} A(\varphi_1(\mathcal{L}_1)\ U\ \varphi_2(\mathcal{L}_2))$ iff for each path π starting at s we have $\pi \models_{\mathcal{I}} \varphi_1(\mathcal{L}_1)\ U\ \varphi_2(\mathcal{L}_2)$.

- $s \models_{\mathcal{I}} \exists x \in \varphi_1(\mathcal{L}_1)\ [\varphi_2(\mathcal{L}_2)]$ iff there exists an instantiation $\mathcal{I}_x : \{x\} \to S$ of x such that $\mathcal{I}_x(x) \models_{\mathcal{I}_1} \varphi_1(\mathcal{L}_1)$ and $s \models_{\mathcal{I}_2 \cup \mathcal{I}_x} \varphi_2(\mathcal{L}_2)$, where $\mathcal{I}_1$ (resp. $\mathcal{I}_2$) is $\mathcal{I}$ with domain restricted to $\mathcal{L}_1$ (resp. $\mathcal{L}_2$).

- $s \models_{\mathcal{I}} \forall x \in \varphi_1(\mathcal{L}_1)\ [\varphi_2(\mathcal{L}_2)]$ iff for every instantiation $\mathcal{I}_x : \{x\} \to S$ of x such that $\mathcal{I}_x(x) \models_{\mathcal{I}_1} \varphi_1(\mathcal{L}_1)$ we have $s \models_{\mathcal{I}_2 \cup \mathcal{I}_x} \varphi_2(\mathcal{L}_2)$, where $\mathcal{I}_1$ (resp. $\mathcal{I}_2$) is $\mathcal{I}$ with domain restricted to $\mathcal{L}_1$ (resp. $\mathcal{L}_2$).

4.2 Complexity of QCTL model checking

The following theorem establishes that QCTL model checking is PSPACE-complete.

Theorem 4.2.1 *QCTL model checking is PSPACE-complete.*
Proof: We reduce QBF-SAT, that is, the satisfiability of *Quantified Boolean Formulas* (QBF) to QCTL model checking. QBF-SAT is known to be PSPACE-complete [12].

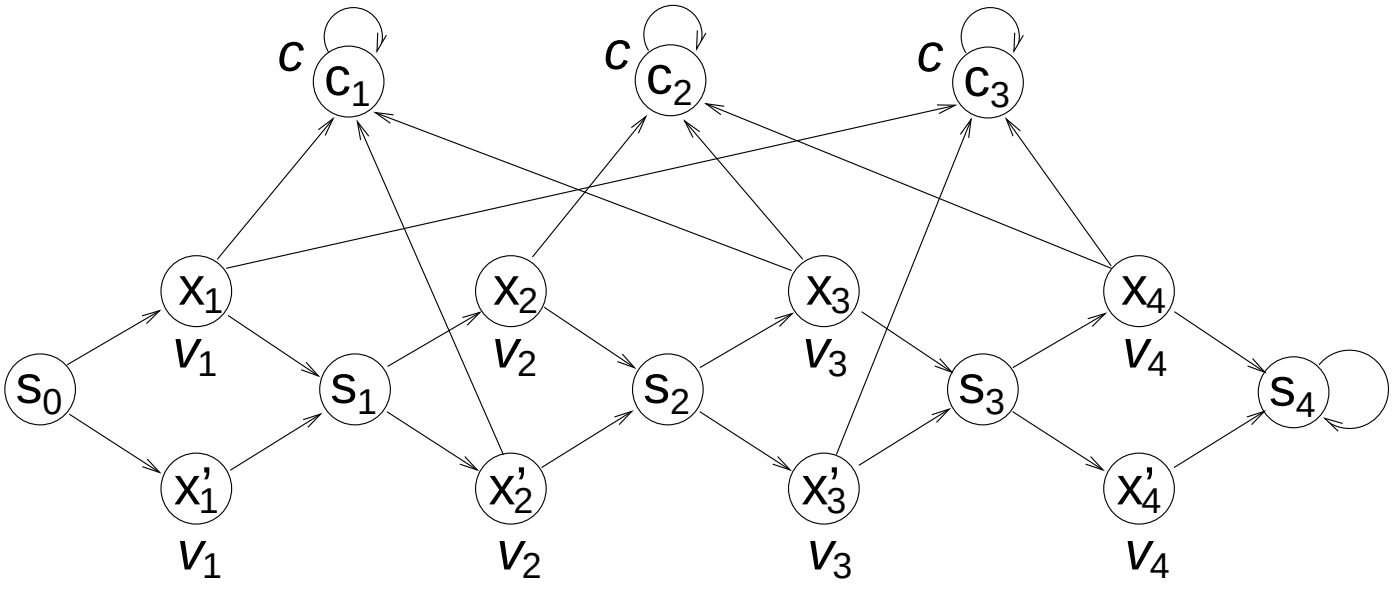

Figure 4.1: Model for $\exists x_1 \forall x_2 \exists x_3 \forall x_4 (x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee x_3 \vee x_4) \wedge (\neg x_3 \vee x_1 \vee x_4)$

Given a QBF of the form: $\exists x_1 \forall x_2 \exists x_3 \ldots \forall x_n Q(x_1, x_2, \ldots, x_n)$ we illustrate the construction of a Kripke structure, J, and a QCTL formula, ψ to be verified on the start state of J. Without loss

of generality [12], we assume that the boolean formula $Q(x_1, x_2, \ldots, x_n)$ is in *conjunctive normal form* (CNF) having k clauses $c_1, \ldots, c_k$, where each clause is a disjunction of literals. We construct a Kripke structure: $J = \langle \mathcal{AP}, S, \mathcal{R}, s_0, \mathcal{F} \rangle$, where:

- $\mathcal{AP} = \{V_1, \ldots, V_n\} \cup \{C\}$,
- $S = \{s_0, \ldots, s_n\} \cup \{x_1, \ldots, x_n\} \cup \{x'_1, \ldots, x'_n\} \cup \{c_1, \ldots, c_k\}$,
- $s_0 \in S$ is the initial state,
- The labeling relation $\mathcal{F} : S \to 2^{\mathcal{AP}}$ is as follows:
  - $\forall i, 1 \leq i \leq n$, $\mathcal{F}(x_i) = \mathcal{F}(x'_i) = V_i$, and
  - $\forall i, 1 \leq i \leq k$, $\mathcal{F}(c_i) = C$.
- The transition relation $\mathcal{R}$ is as follows:
  - $\forall i, 0 \leq i < n$, $\mathcal{R}(s_i, x_{i+1})$ and $\mathcal{R}(s_i, x'_{i+1})$,
  - $\forall i, 0 < i \leq n$, $\mathcal{R}(x_i, s_i)$ and $\mathcal{R}(x'_i, s_i)$,
  - $\forall i, 0 < i \leq n$ and $\forall j, 0 < j \leq k$, $\mathcal{R}(x_i, c_j)$ iff the clause c_j in the given QBF contains the literal x_i,
  - $\forall i, 0 < i \leq n$ and $\forall j, 0 < j \leq k$, $\mathcal{R}(x'_i, c_j)$ iff the clause c_j in the given QBF contains the literal $\neg x_i$,
  - $\forall j, 0 < j \leq k$, $\mathcal{R}(c_j, c_j)$,
  - $\mathcal{R}(s_n, s_n)$.

Fig 4.1 shows the Kripke structure corresponding to the following QBF:

$$\exists x_1 \forall x_2 \exists x_3 \forall x_4 (x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee x_3 \vee x_4) \wedge (\neg x_3 \vee x_1 \vee x_4)$$

The QCTL formula corresponding to this formula will be as follows:

$$\begin{aligned}
&\exists v_1 \in V_1\ [\forall v_2 \in V_2\ [\exists v_3 \in V_3\ [\forall v_4 \in V_4 \\
&\quad [\forall c \in C\ [E(true\ U\ (v_1 \wedge EX(c))) \\
&\qquad \vee E(true\ U\ (v_2 \wedge EX(c))) \\
&\qquad \vee E(true\ U\ (v_3 \wedge EX(c))) \\
&\qquad \vee E(true\ U\ (v_4 \wedge EX(c)))]]]]]
\end{aligned}$$

In general, the QCTL formula corresponding to the given QBF is of the form:

$$\psi = \exists v_1 \in V_1\ [\forall v_2 \in V_2\ [\ldots\ [\forall v_n \in V_n\ [\forall c \in C\ \left[\bigvee\nolimits_{j \in \{1,n\}} E(true\ U(v_j \wedge EX(c)))\right]]]\ldots]]$$

There is a one-to-one correspondence between the instantiations of the variables in the given formula $Q(x_1, \ldots, x_n)$, and the instantiations of the variables in the following QCTL subformula of ψ:

$$\varphi(v_1, \ldots, v_n) = \forall c \in C \ \left[\bigvee_{j \in \{1,n\}} E(true\ U(v_j \wedge EX(c))) \right]$$

Specifically, if x_i is instantiated to true in Q, then we instantiate v_i to the state x_i, and if x_i is instantiated to false in Q, then we instantiate v_i to the state x_i', and vice versa. Therefore we require to show that those and only those instantiations which satisfy Q are the ones under which s_0 models φ.

If x_i (resp. x_i') is chosen by the instantiation, then $E(true\ U(v_i \wedge EX(c)))$ is satisfied only for those clauses, c, which contain x_i (resp. $\neg x_i$) as a literal (this follows from the definition of the transition relation $\mathcal{R}$). Since we use $\forall c \in C$ to quantify the disjunction of $E(true\ U(v_i \wedge EX(c)))$, we require an instantiation where *each* clause is satisfied by at least one v_i. For the other direction, any instantiation of the v_is for which φ holds at s_0, must satisfy at least one of $E(true\ U(v_i \wedge EX(c)))$ at s_0 for every c. The instantiations of the v_is therefore yields an instantiation of x_is in Q which satisfies Q.

It is easy to see that QCTL is in PSPACE. Given the instantiations of each variable to a state of the Kripke structure, the verification task reduces to CTL model checking (including the verification of whether the instantiations are from the correct domains). This can be done in polynomial space. The instantiations can be generated recursively in polynomial space and verified as above. $\square$

Chapter 5

Implementation of WQCTL

With the Weighted QCTL in the heart we have implemented a query language where a series of complex formula can be given one after another . We give below the Grammar of the Query language that we have implemented :

The Grammar for Implementation of Query Language

- $S :' Query'()\{\ decl\_list\_opt \quad stmt\_list\_opt\ \}$
- $decl\_list\_opt :\ decl\_list \mid \lambda$
- $decl\_list :\ decl \mid \ decl\_list\ decl$
- $decl :\ type\ var\_list$
- $var\_list :\ id,\ var\_list \mid \quad id\ ;$
- $type :\ Bool\_set \mid Val\_set$
- $stmt\_list\_opt :\ stmt\_list \mid \lambda$
- $stmt\_list :\ stmt \mid stmt\_list\ stmt$
- $stmt :\ Bool\_stmt \mid Val\_stmt \mid Rep\_stmt$
- $Bool\_stmt :\ id\ =\ Bf$
- $Val\_stmt :\ id\ =\ Vf$
- $Rep\_stmt :\ result(id);$
- $Bf : \neg Bf \mid Bf \wedge Bf \mid Bf \vee Bf \mid AX(Bf) \mid EX(Bf) \mid id \mid AP \mid exp\ relop\ exp \mid \exists\ var\ id\ (Bf_x) \mid\ \forall\ var\ id\ (Bf_x) \mid A(BfU_M\ Bf) \mid E(BfU_M\ Bf)$

- $Bf_x : \neg Bf_x \mid Bf_x \wedge Bf_x \mid Bf \wedge Bf_x \mid Bf_x \wedge Bf \mid Bf_x \vee Bf_x \mid Bf \vee Bf_x \mid \quad Bf_x \vee Bf \mid AX(Bf_x) \mid EX(Bf_x) \mid G_{sx} \mid var$
- $G_{sx} :\ A(BfU_M\ Bf_x) \mid E(BfU_M\ Bf_x)$
- $G_{zx} :\ A(BfU_M\ Vf_x) \mid E(BfU_M\ Vf_x)$
- $Vf_x :\ G\ G_{sx} \mid G\ G_{zx} \mid Vf_x \wedge Bf_x \mid Vf_x \wedge Bf$
- $Vf :\ Vf \wedge Bf \mid Ag(\ \exists\ var\ id[Vf_x]) \mid Ag(\ \forall\ var\ id[Vf_x]) \mid G(\ \exists\ var\ id[G_{sx}]) \mid G(\ \forall\ var\ id[G_{sx}]) \mid\ G\ Ag(\ \exists\ var\ id[G_{zx}]) \mid G\ Ag(\ \forall\ var\ id[G_{zx}]) \mid G\ A(BfU_M\ Vf) \mid G\ E(BfU_M\ Vf) \mid G\ A(BfU_M\ Bf) \mid\ G\ E(BfU_M\ Bf) \mid id :$
- $M :\ min \mid max \mid \lambda$
- $G :\ LP \mid SP$
- $Ag :\ minval \mid maxval$
- $exp :\ exp\ op\ exp \mid id \mid const$
- $op : + \mid -$
- $relop : < \mid > \mid >= \mid <=$

5.1 Implementation of the Query Language

We now explain the data structures used and how we store the Timed Model .
The Timed Model is a labeled Weighted Graph which we store as follows :
The adjacency list for a vertex is stored as an array of ⟨ vertex , weight ⟩ pair of the size of outdgree of the vertex. The state are numbered from 0 to n-1 where there are n states .
The state lebeled by an atomic proposition is stored in form of bit-vector. A bit vector is associated to every atomic proposition which has 1 in its i-th bit if the i-th state is marked by the corresponding atomic proposition .
Given the graph has n states the length of each bitvector is n .
Representation of formula is also as bit vector. A formula with value has a bit vector and an array of integers(value vector) assocaited with it . Bit vector's i-th bit denotes the truth of the formula in the i-th state and i-th value of the value vector denotes the Evaluated value of the formula in the i-th state .

Now we illustrate with diagram the storage of a graph and the atomic proposition.

Consider the following Graph.
It has 7 states which are numbered from 0 to 6 and there are 3 atomic propostion and the states marked by atomic proposition is shown in the diagram . The weight with each edge is shown in the diagram .

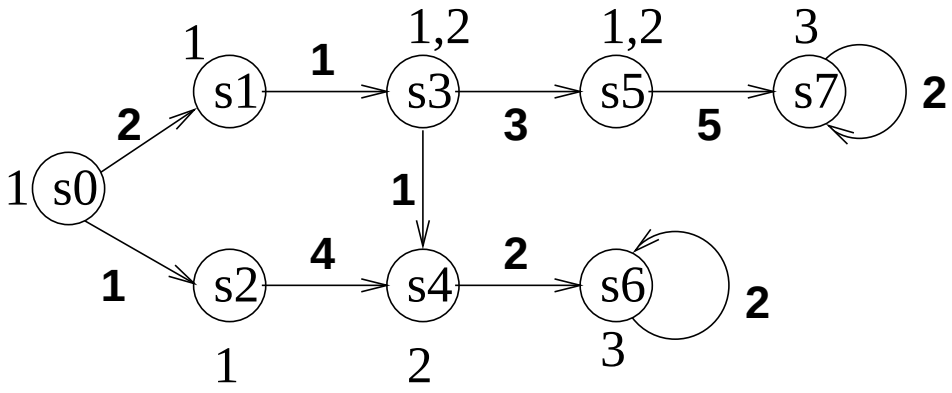

Figure 5.1: A Timed Model

The way we store the graph is shown in Fig 5.2:

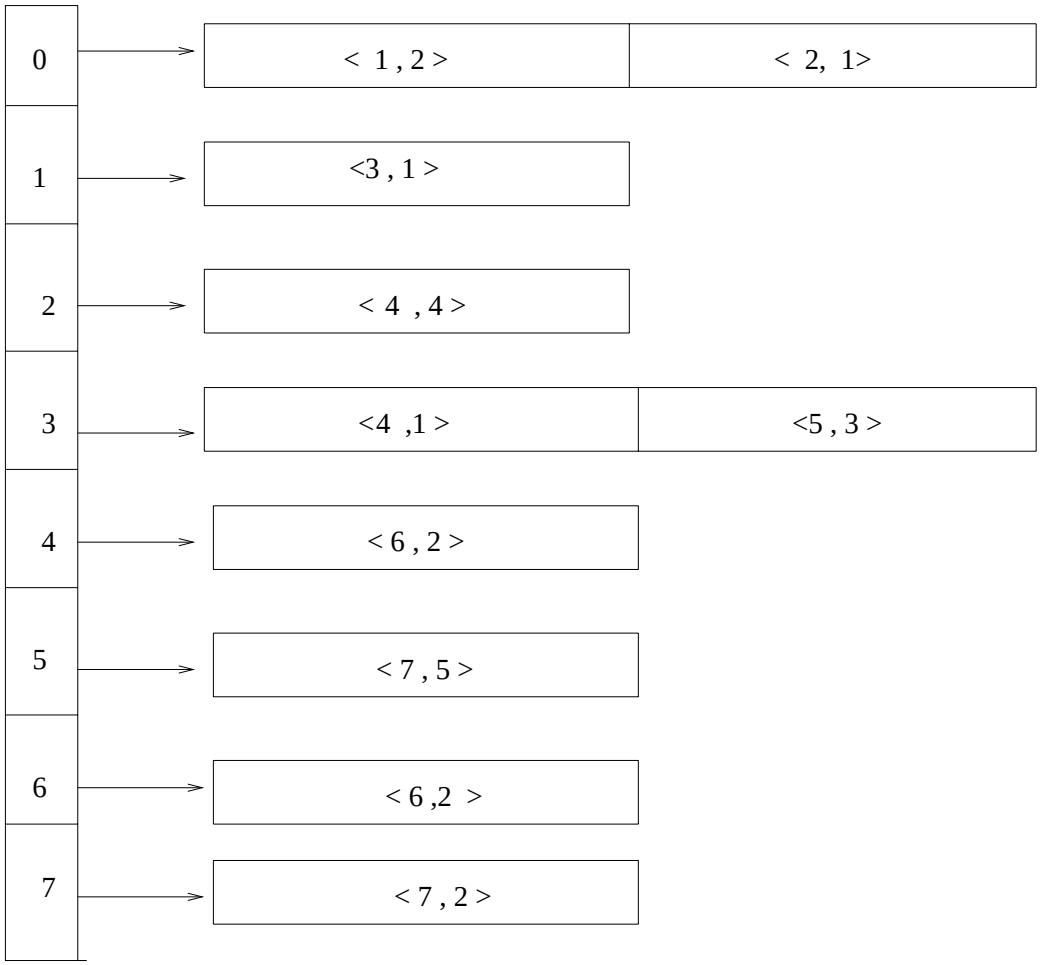

Figure 5.2: The storage of the Weighted Graph as ⟨ vertex, weight ⟩ pairs

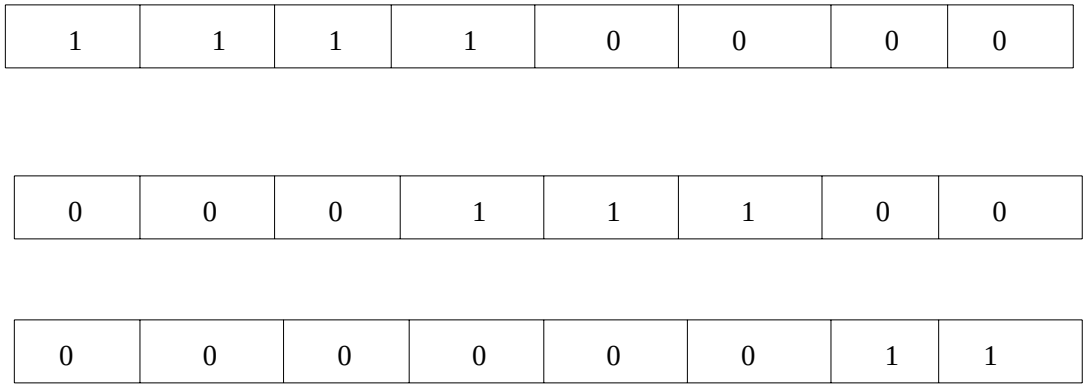

Figure 5.3: The storage of the label of the Nodes as bitvectors.

As shown in the Fig 5.3 atomic propositon 1 true in state no. 0,1,2,3 which is shown in the first bit-vector.
Similarly atomic proposition 2 true in state no. 3,4,5.
Atomic proposition 3 true in state no 6,7.

5.2 Storage of Parse Tree to save in Space

We explain that how we handle the space problem associated with the formulas with Quantification over states . A simple way to implement is to treat each state has a unique atomic proposition . But this would have caused a lot of storage as the number of atomic proposition would have been of the order of states.

Consider the formula:
$\forall\ x \in\ p2[\ A(\ p1\ U\ x\ )\ \wedge\ E(\ p3\ U\ x)\ ]$.
We illustrate what happens if we take action instantly when it is being parsed and then we illustrate if we store the parse tree of the formula and later take action how we can save in space.

The atomic proposition 2(denoted as p2) is true in three state . The three bit-vectors in the LEVEL 3 corresponds to the truth vectors of the several instantiation of x. (i.e . 2 is true in state 3,4,5 . So in the first bit-vector only 3rd bit is 1 rest all 0's , in the second bit-vector only 4th bit is 1 and rest all 0's and in the third bit-vector only 5th bit is 1 and all other 0's.). Corresponding to each of this bit-vector we at LEVEL2: find the truth vector of A(p1 U x) and E(p3 U x) . Now we *AND* the first bit-vector of AU node with the first-bitvector of EU , the second bit-vector of AU with second bit-vector of EU and so on to get the bit-vectors at LEVEL 1. Now as the Quantifier is *forall* we *AND* the three bit-vectors of LEVEL 1 to get the truth of the formula f.Had the Quantifier been *exists*

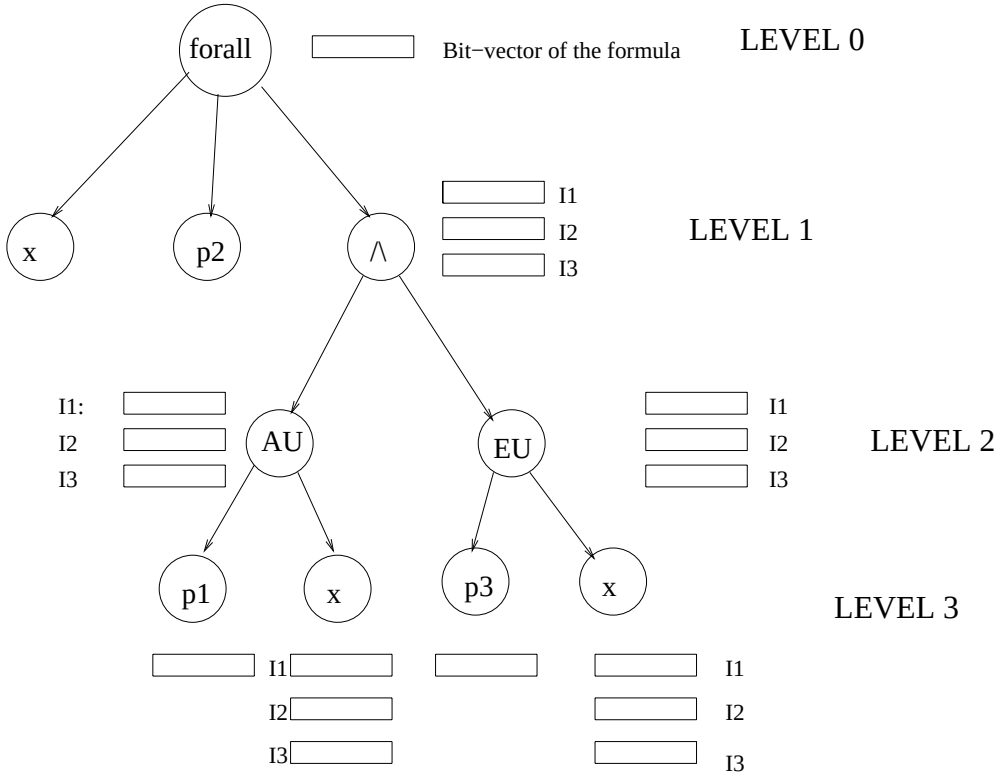

Figure 5.4: The parse tree and the space required to store bit-vectors if action taken when parsed.

we would have *OR* the bit-vectors of LEVEL1. to get the truth of f. As in general the number of instatitaion is of the order of the states we require a lot of space to store so many bitvectors.

The earlier method requires the storage of a lot of bit-vectors . Instead we store the whole pasre tree and at LEVEL 0 we instantiate the bit-vectors and then evaluate the truth of the formula corresponding to the instantiation.So here we instantiate the first bit-vector and get the bitvector of A(p1 U x)∧ E(p3 U x) . Then we again instantiate the second bit-vector get the bitvector and *AND* it with the result to save space .So here we run the procedure for the number of instantiation required .

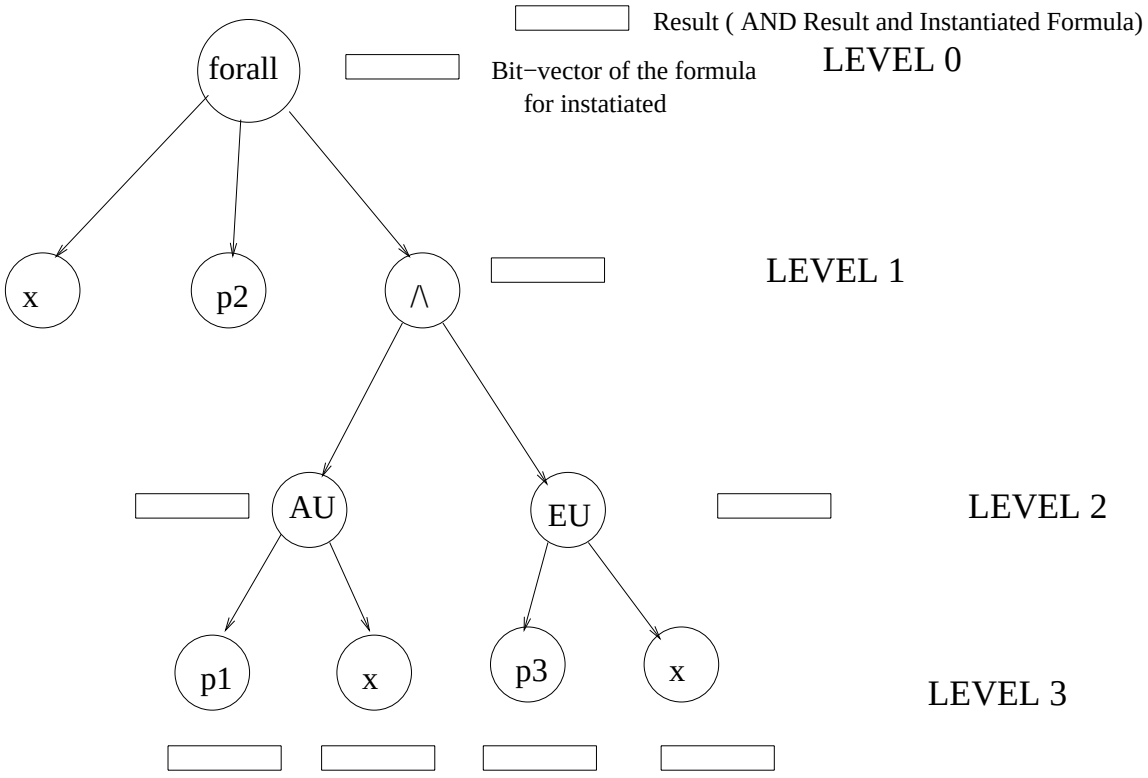

Figure 5.5: The parse tree and the space required to store bit-vectors if action defered storing the parse tree.

The problem is more acute with formulas with a evaluation procedure . We illustrate as below :

Consider the formula:
$MIN(\forall\ x \in\ p2[\ MIN\ A(\ p1\ U\ x\ )\ \wedge\ E(\ p3\ U\ x)\ ])$.

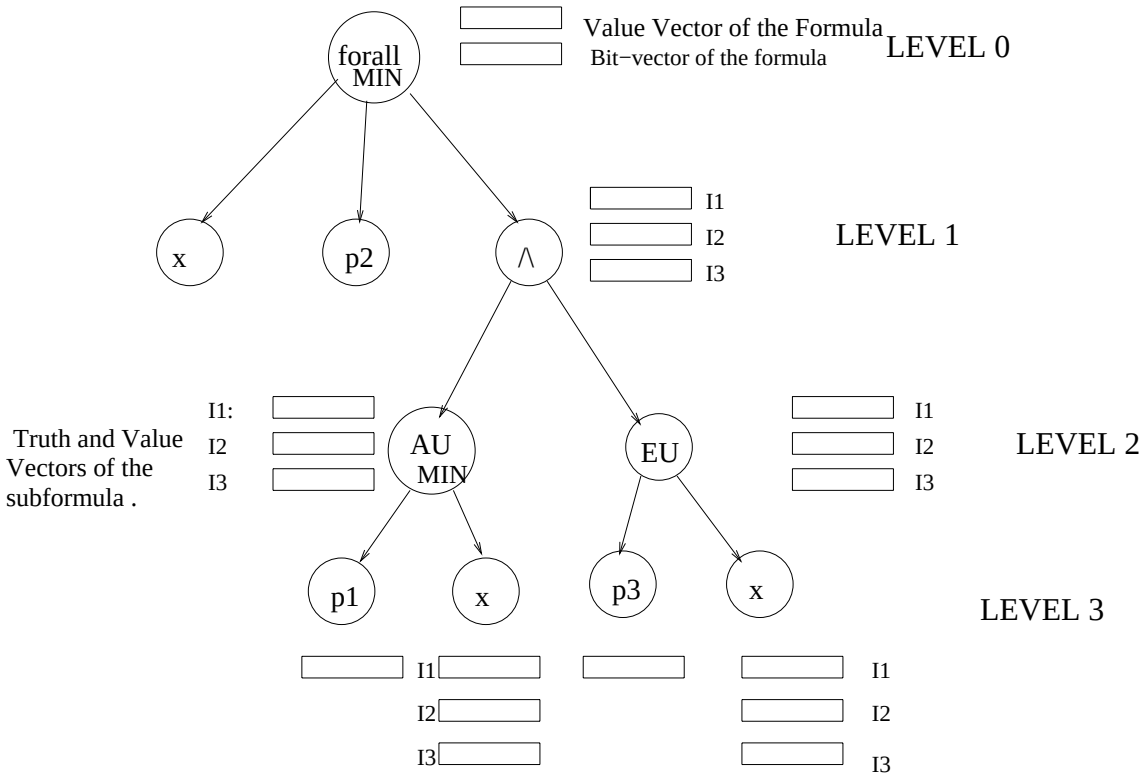

Figure 5.6: The parse tree and the space required to store bit-vectors and value vectors if action taken when parsed.

If we take action when they are being parsed then we have to store bitvectors and value vectors of the number of which in general can be the order of states as shown in Fig 5.6.

As before we have a bitvector for the Result of the truth of the formula and one value vector for the Result of the value of the formula . Since the Quantifier is $\forall$ we instantiate and keep *AND*ing with the Result truth vector and as the operator is MIN we keep taking the Minimum of the present value vector and the value vector of the Result. Similar strategy follows with $\exists$ Quantifier and MAX operator. So we run the process of instatiation and evaluating the parse tree for the number of times as many as the number of states where $p2$ is true and each time only storing the present updated Result and throw away rest all information.

This is shown in Fig 5.7.

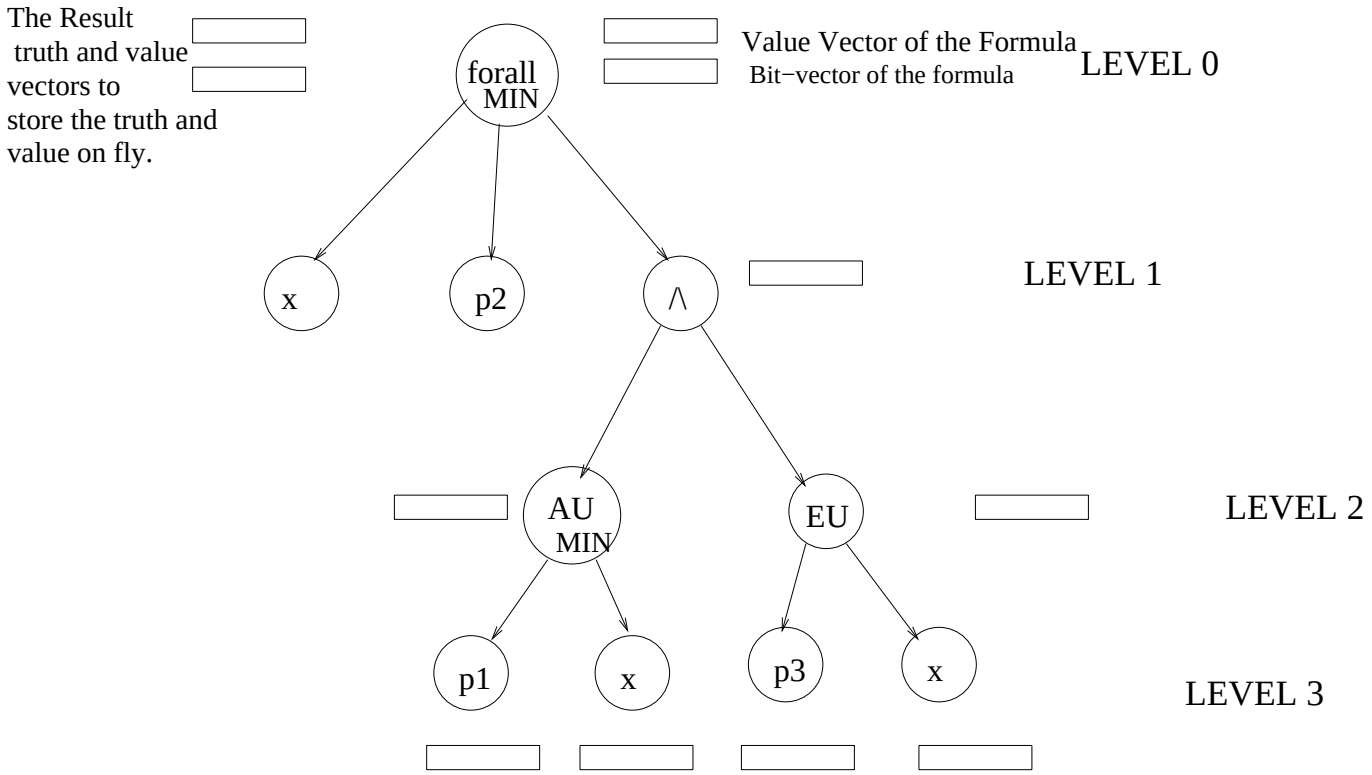

Figure 5.7: The parse tree and the space required to store bit-vectors if action defered storing the parse tree.

Chapter 6

Experimentation Results

Sample Query 1: How does a Query program looks like :

Query ()
{

- $Val\_set f_1, f_3$;
- $Bool\_set f_2$;
- $f_1 = EX(p_2)$;
- $f_3 = \exists\ x\ in\ f_1 E(p_1 U x)$
- $result(f_3)$;
- $f_3 = \exists\ x\ in\ f_1 A(p_1 U x)$
- $result(f_3)$;
- $f_3 = \forall\ x\ in\ f_1 E(p_1 U x)$
- $result(f_3)$;
- $f_3 = \forall\ x\ in\ f_1 A(p_1 U x)$
- $result(f_3)$;
- $f_2 = LPA(p_2 U p_1)$
- $result(f_2)$;
- $f_1 = A(p_2\ U\ p_1)$
- $result(f_1)$;

}

Query ()
{

- $Val\_set f_2$;
- $Bool\_set f_1$;
- $f_2 = SPA(true U_{min} p_2)$
- $result(f_2)$;
- $f_2 = SPE(p_1 U_{min} f_2 :)$
- $result(f_2)$;

}

EXPERIMENTAL RESULTS FOR SAMPLE QUERY 2

State	Deg.	Edges	Space	User Time msec	System Time msec
100	10	959	17.3 KB	12	5
200	10	2368	39.7 KB	57	8
500	10	5954	99 KB	362	23
1000	10	11984	199 KB	1538	45
2000	15	34228	521 KB	7046	124
5000	15	84597	1.29 MB	48516	291
10000	15	169770	2.58 MB	214497	1543
20000	20	437188	6.34 MB	1013575	8655
30000	20	656783	9.53 MB	2382190	18279
50000	20	1056233	12.9 MB	6612884	53337
60000	20	1324561	15.5 MB	9718738	78955
90000	20	1924326	23.3 MB	23200284	193551

Sample Query 3

Query ()
{

- $Val\_set f_2$;
- $Bool\_set f_1, f_3$;
- $f_1 = AX(p_2)$;
- $result(f_1)$;
- $f_3 = E(p_1 U f_1)$;
- $result(f_3)$;
- $f_3 = A(p_1 U f_1)$;
- $result(f_3)$;
- $f_2 = SPA(p_2 U p_1)$;
- $result(f_2)$;
- $f_2 = LPA(p_2 U p_1)$;
- $result(f_2)$;
- $f_1 = A(p_2 U p_1)$;
- $result(f_1)$;

}

EXPERIMENTAL RESULTS FOR SAMPLE QUERY 3

State	Deg.	Edges	Space	User Time msec	System Time msec
100	10	959	16.7 KB	6	6
200	10	2368	38.6 KB	24	9
500	10	5954	96 KB	122	25
1000	10	11984	194 KB	489	42
2000	15	34228	511 KB	2257	119
5000	15	84597	1.26 MB	14354	317
10000	15	169770	2.53 MB	63075	680
20000	20	437188	6.24 MB	310865	2577
30000	20	656783	9.3 MB	720048	5271
50000	20	1056233	12.73 MB	1840063	10766
60000	20	1324561	15.25 MB	2680585	24877
90000	20	1924326	22.8 MB	6303143	40879

Sample Query 4

Query ()
{

- $Val\_set f_2, f_4$;
- $Bool\_set f_1$;
- $f_1 = p_1$;
- $f_2 = LPE(p_2 U_{min} p_1)$;
- $result(f_2)$;
- $f_4 = LPE(p_1 U_{min} p_2)$;
- $result(f_4)$;
- $f_1 = f_2 < f_4$;
- $result(f_1)$;
- $f_1 = f_2 <= f_4$;
- $result(f_1)$;
- $f_1 = f_2 - 2 <= f_4$;
- $result(f_1)$;

}

EXPERIMENTAL RESULTS FOR SAMPLE QUERY 4

State	Deg.	Edges	Space	User Time msec	System Time msec
100	10	959	16.6 KB	8	5
200	10	2368	38.1 KB	29	6
500	10	5954	95 KB	150	25
1000	10	11984	191 KB	579	49
2000	15	34228	505 KB	2647	111
5000	15	84597	1.25 MB	16302	287
10000	15	169770	2.51 MB	68489	647
20000	20	437188	6.19 MB	334892	1490
30000	20	656783	9.29 MB	770160	2879
50000	20	1056233	12.69 MB	1932268	9446
60000	20	1324561	15.09 MB	2740223	22432
90000	20	1924326	22.7 MB	6524129	38772

Query()

{

- $Bool\_set f_3, f_1;$
- $f_1 = p_2;$
- $f_3 = \exists x \in f_1(E(p_1 U x));$
- $f_3 = \forall x \in f_1(E(p_1 U x));$
- $result(f_3, 7);$

}

EXPERIMENTAL RESULTS FOR SAMPLE QUERY 5

State	Deg.	Edges	Space	User Time msec	System Time msec
100	10	959	17 KB	9	7
200	10	2368	34.5 KB	16	5
500	10	5954	86 KB	53	19
1000	10	11984	173 KB	181	43
2000	15	34228	470 KB	954	96
5000	15	84597	1.16 MB	3001	114
10000	15	169770	2.33 MB	4923	153
20000	20	437188	5.83 MB	12084	378
30000	20	656783	8.77 MB	27259	851
50000	20	1056233	11.69 MB	66904	1151
60000	20	1324561	14.02 MB	94640	1626
90000	20	1924326	22.36 MB	113263	2153

Query()
{

- $Bool\_set f_3, f_1$;
- $Value\_set f_2$;
- $f_1 = p_1$;
- $f_2 = maxval(\exists x \in f_1[SPA(p_2 Ux)])$;
- $f_2 = minval(\forall x \in f_1[SPA(p_2 Ux)])$;
- $result(f_2, 7)$;

}

EXPERIMENTAL RESULTS FOR SAMPLE QUERY 6

State	Deg.	Edges	Space	User Time msec	System Time msec
100	10	959	19 KB	180	3
200	10	2368	40.3 KB	1356	6
500	10	5954	100.2 KB	10917	49
1000	10	11984	202 KB	76544	209
2000	15	34228	527 KB	107146	1396
5000	15	84597	1.24 MB	197326	2047
10000	15	169770	2.6 MB	447866	2544
20000	20	437188	6.1 MB	904366	2978
30000	20	656783	9.3 MB	1913261	3372
50000	20	1056233	12.1 MB	2344123	3688

Chapter 7

Extension and future work

It is possible to extend Weighted QCTL in several directions without sacrificing the computational efficiency of Weighted QCTL evaluation. In this chapter, we describe two interesting extensions.

Adding on Conjunction and Disjunction: In Weighted QCTL, we have not allowed the conjunction and disjunction of Weighted QCTL state formulas. It is easy to extend Weighted QCTL to allow conjunction and disjunction of Weighted QCTL state formulas by appropriately defining the semantics of evaluation. For example, given two Weighted QCTL formulas, f_1 and f_2, we can define the syntax of evaluation of $f = f_1 \wedge f_2$, with respect to a specified cost function C as:

$$EvalS(f, s) = C\left(EvalS(f_1, s), EvalS(f_2, s)\right)$$

If $f = f_1 \vee f_2$, we can define $EvalS(f, s)$ in a similar way. However, in the second case, it is possible that $s \models f_1$ but $s \not\models f_2$. Since $s \models f_1 \vee f_2$, $EvalS(f, s)$ must return a value even though $EvalS(f_2, s)$ returns null. We believe that this difficulty can be overcome by defining appropriate semantics for evaluation with possibly null return values.

Weighted QCTL U Weighted QCTL: In Weighted QCTL, we have allowed only QCTL formulas for f_1 in every $f_1 U f_2$ formula, (that is, we have allowed only f_2 to be Weighted QCTL). It is possible to extend Weighted QCTL to allow f_1 to be Weighted QCTL as well. In that case, we require to appropriately define the evaluation syntax over the set of states where f_1 holds until f_2 holds. This may require different interpretations of the same Weighted QCTL sub-formula, f_1, depending on whether the given formula involves $f_1 U f_2$ or $f_2 U f_1$. We have not attempted to define any such semantics for simplicity. However there can be interesting properties which can be expressed in the extended language.

Another future direction of work would be to look into what other important Graph Theoretic properties can be captured through the temporal properties. One other direction is to see that if we change the way we define the g-value of a path. It is the sum of the edge weights as we define presently. Changing the definition of the g-value to some other form may help us to express many more interesting queries without incurring much loss in the Computational efficiency. On the other hand we can study the complexity class where Weighted QCTL lies with different definitions of g-values and generalising the MIN-MAX operators to some General Operator.

Automata theoretic approaches for evaluation of Weighted QCTL queries need to be studied. This will also help us study the connection between the complexity of evaluation of Weighted QCTL queries and the nature of the function being maximized or minimized.For example general MinMaxCTL was proved to be DP-hard.However the exact complexity class to which belongs to would be of great theoretical interest.This may lead us to a closer understanding of the complexity involved in Timed verification.The exact class of queries verifiable in a given time resource also will be of great theoretical interest.

Chapter 8

Conclusion

Model checking with temporal logics such as TCTL, TLTL and RTCTL which explicitly reason about timing properties of transition systems is known to be more complex than reasoning about untimed temporal logics such as CTL. Specifically, CTL is attractive because it can be checked in time polynomial in the size of the transition system.But there are properties which can be verified in polynomial time but cannot be expressed in CTL. We explore such properties and have shown that using Quantification over states we can express interesting properties which are not possible in CTL. We have shown that quantitative reasoning about timing properties can often be done in polynomial time, specifically when we are interested in the extremal timing properties of the system. We have shown that the proposed logic, Weighted QCTL, can be used to express interesting queries about the best case and worst case temporal behaviors of timed models. We have also shown that for many useful cost functions, Weighted QCTL can be evaluated in polynomial time. Wighted QCTL allows to quantify over paths , across paths and across quantification of the states. This helps us to express many intersting timing properties about system realted to Timed Model.

Bibliography

[1] Alur, R., and Henzinger, T., Real time logics: Complexity and Expressiveness. *Information and Computation*, **104**, 1, 35-77, 1993.

[2] Alur, R., Courcoubetis, C., and Dill, D., Model checking in dense real-time. *Information and Computation*, **104**, 1, 2-34, 1993.

[3] Alur, R., and Henzinger, T.A., A really temporal logic. *JACM*, **41**, 1, 181-204, 1994.

[4] Alur, R., Timed Automata. Manuscript: www.cis.upenn.edu/∼alur/Nato97.ps.gz, 1998.

[5] Burch, J.R., Clarke, E.M., Long, D.E., McMillan, K.L., and Dill, D.L., Symbolic model checking for sequential circuit verification. *IEEE Trans. on Computer Aided Design*, **13**, 4, 401-424, 1994.

[6] Campos, S.V., Clarke, E.M., Marrero, W., and Minea, M., Verus: A tool for quantitative analysis of finite-state real-time systems. In *Workshop on Languages, Compilers and Tools for Real-Time Systems*, 1995.

[7] Clarke, E.M., Emerson, E.A., and Sistla, A.P., Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. on Program. Lang. & Systems*, **8**, 2, 244-263, 1986.

[8] Clarke, E.M., and Kurshan, R.P., Computer aided verification. *IEEE Spectrum*, **33**, 6, 61-67, 1996.

[9] Cormen, T.H., Leiserson, C.E., and Rivest, R.L., *Introduction to Algorithms*. The MIT Press, Cambridge and McGraw-Hill, 1990.

[10] Courcoubetis, C., and Yannakakis, M., Minimum and maximum delay problems in real-time systems. In *Proc. of the Third Workshop on Computer-Aided Verification*, LNCS 575, 399-409, 1991.

[11] Emerson, E.A., Mok, A.K., Sistla, A.P. and Srinivasan, J., Quantitative temporal reasoning, In *First Annual Workshop on Computer-Aided Verification*, France, 1989.

[12] Papadimitriou, C.H., *Computational Complexity*, Addison-Wesley, 1994.

[13] R.Alur and M.Yannakakis. Model checking of hierarchical state machines In *Sixth ACM Symposium on the Foundations of Software Engineering*,1998.

[14] K.Laster and O.Grumberg. Modular model checking of software. In *Proceedings of International Conference on Tools and Algorithms for Construction and Analysis of System (TACAS'98),Lisbon.*

bitemvardi:82 Vardi, M., Complexity of relational query languages, *STOC'82*, 137-146, 1982.

[15] Vardi, M., Alternating automata and program verification. *Computer Science Today: Recent trends and developments*, LNCS Vol 1000, 1995.

[16] West, D.B., *Introduction to Graph Theory*, Prentice Hall, 1996.