A Tool-Chain for Monitoring Real-Time Properties

Shubham Tomar

A Tool-Chain for Monitoring Real-Time Properties

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

Master of Technology

in

Computer Science and Engineering

by

Shubham Tomar
20CS06016

Under the supervision of

Dr. Srinivas Pinisetty

SCHOOL OF ELECTRICAL SCIENCES
INDIAN INSTITUTE OF TECHNOLOGY BHUBANESWAR
MAY 2022

APPROVAL OF THE VIVA-VOCE BOARD

Date: 2^{nd} may 2022

Certified that the thesis entitled **A Tool-Chain for Monitoring Real-Time Properties** is submitted by **Mr. Shubham Tomar (20CS06016)** to the Indian Institute of Technology Bhubaneswar for the award of the degree Master of Technology has been accepted by the examiners and that the student has successfully defended the thesis in the viva-voce examination held today.

(Supervisor) (External Examiner)

(Internal Examiner I) (Internal Examiner II)

CERTIFICATE

Certified that the thesis entitled **A Tool-Chain for Monitoring Real-Time Properties** is submitted by **Mr. Shubham Tomar (20CS06016)** to the Indian Institute of Technology Bhubaneswar for the award of the degree Master of Technology has been accepted by the examiners and that the student has successfully defended the thesis in the viva-voce examination held today.

Date:

Dr. Srinivas Pinisetty
(Supervisor)

DECLARATION

I certify that

1. The work contained in the thesis is original and has been done by myself under the general supervision of my supervisor.
2. The work has not been submitted to any other Institute for any degree or diploma.
3. I have followed the guidelines provided by the Institute in writing the thesis.
4. I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.
5. Whenever I have used materials (data, theoretical analysis, and text) from other sources, I have given due credit to them by citing them in the text of the thesis and giving their details in the references.
6. Whenever I have quoted written materials from other sources, I have put them under quotation marks and given due credit to the sources by citing them and giving required details in the references.

Shubham Tomar

20CS06016

ACKNOWLEDGMENTS

I owe a debt of gratitude to all the people who helped, influenced and motivated me during my wonderful journey of two years.

First and foremost I would like to express my sincere gratitude to my thesis supervisor, Dr. Srinivas Pinisetty for his vital support and assistance. It has been great experience learning with him and his moral support in every direction any time when I required inspite of his busy schedule. His guidance and encouragement made it possible to achieve the goal. His high level knowledge and enthusiastic behavior motivated me to tackle the research work. He is my source of inspiration. He is the pioneer of my life.

I would like to show my utmost gratitude towards Dr. Padmalochan Bera for his valuable suggestion and inspiration throughout my master's degree. I would like to pay my regards to all the faculty, staff members and lab technicians of School of Electrical Sciences, whose services turned my research a success.

I am specially thankful to Miss Saumya Shankar and Mr. Abhinandan Panda for her help during implementation of my project. I extend my acknowledgement to my friends Mr. Avadesh Chamola, Mr. Samannay Mahanta, Mr. Abhishek Kumar Maurya, Mr. Varinder Kantoria, Mr. Aditya Jaiswal and all my batchmates for their never ending support, care and love which made this journey very memorable. This journey in this IIT would have not been complete without them. They helped me in enriching my knowledge not only in academic aspects but other domain of life. Their shared experience gave a much wider aspect of understanding about life. I rejoiced every experience of my life with them while being in my hard times also.

Last but not the least I would like to take this opportunity to embrace my parents and my beloved ones for everything they have done for me. I am greatly indebted towards them. I thank my parents for their support throughout my life. They not only assisted me financially but also extended their support morally and emotionally. Once again a special thanks to who extended their care and supportive hands in every aspects of my life in these two years of journey.

Shubham Tomar

ABSTRACT

Verification and validation of safety-critical systems are nowadays done using formal method approaches like static verification (model checking) and runtime monitoring in order to provide safety guarantees. Static verification approaches like model checking and theorem proving suffer from scalability issues and failure to provide assurances during execution (actual deployment) of the safety-critical systems. Therefore, we focus on lightweight monitoring methods for safety-critical systems, which are runtime verification (RV) and runtime enforcement (RE), which provide safety guarantees during the execution of the system.

The RV is the process of ensuring that system executions conform to some desired property, based on theories, techniques, tools, and methods designed to ensure compliance. In the case of the violation of some desired property, the enforcer performs certain evasive actions, such as delaying or suppressing, that are used to prevent the violation of the desired property. Because they allow one to clearly define how much time should elapse between two events, timed properties are always a more precise tool for determining the desired behaviour of systems.

Prototype tools for enforcement monitor generation were developed using other TA-based libraries. However, the TA-based libraries that were used are no longer maintained and pose difficulties in extending the proposed framework. Though there are few tools existing for monitoring real-time systems, they have certain limitations such as platform dependence, use of some libraries that are no longer maintained, and a lack of a proper user interface for modelling the policies.

A tool-chain for monitoring real-time properties has been developed with an intuitive UI for modelling policies, modularity of various components and maintainability (e.g., supporting easy integration of new monitoring schemes/algorithms for policies specified as TAs into the framework). Properties satisfied by the TA can be verified with the help of this tool. Enforcement monitoring is used to enforce the input sequence for correctness of the system property.

Keywords: Timed automata, Runtime verification, Runtime enforcement, Safety critical system.

Contents

List of Figures

Chapter 1

Introduction

The growth of ubiquitous embedded and safety-critical or life-critical systems leads to the requirement of high-level confidence in the behaviour of these systems. So, it becomes very important to consider the functionality of a software application with a sense to find whether the developed software meets the appointed requirements or not and to specify the faults to ensure that the product is defect-free [4]. A safety-critical system or a life-critical system is a system where a single wrong value can cost a life, the health of a person, wealth, etc. For example, a real-life instance of the Patriot Missile System happened at Dhahran during the Persian Gulf War in 1991.

This system was highly portable for surface-to-air missile systems to knock down enemy aircraft. It was further modified to knock down Scud missiles. Though this system was tested rigorously on many test cases in which it was a great success but still it was a huge failure. This failure cost 28 deaths and 100 other casualties. The main reason behind the failure of this system was a very small bug in the software system's clock. This clock has approximately 1/3 of the second which results in the difference of 600 meters difference from its target. Moreover, there were many updates to this system during the war which required shutting the system for 1 to 2 hours. A small decimal has changed the scenario of the whole war. There are many such failures like Perils of floating-point, Ariane 5, etc.

Multiple approaches for monitoring real-time properties have been proposed [2] [8]. Different approaches differs from one another on the basis of the logic/formalism used to formalize the specified properties and the the power of the monitor (the monitor can have the power of delaying an event, suppressing an event, inserting/ editing an

event, etc.). For real-time systems, timed automata [1] is one of the key formalisms for modeling and analysing real-time systems. Various formal verification approaches such as model checking, and tools based on them such as UPPAAL [6] that have been proposed are based on the theory of timed automata. Recently approaches such as [2] also have been proposed for RV [11] and RE [13] for real-time systems for properties defined as timed automata. There are tools proposed based on these such as Tipex [10] and GREP [14].

1.1 Traditional Formal Verification Approaches

Traditionally, formal verification techniques are:

1. Theorem Proving
2. Testing
3. Model Checking.

Theorem Proving: Theorem proving is generally done by hand and allows you to demonstrate the correctness of programmers in the same way that a proof in mathematics demonstrates the soundness of a theorem. It heavily relies on high-order logic and employs mathematical structures to create formulas that correspond to the system's behaviour. The examination of these formulas, then, is the verification process.

Testing: In testing, a test suite is made up of a finite number of finite input–output sequences. When the input sequence is given to the system under test, test-case execution is used to see if the system's output matches the predicted one. Testing scales better as system complexity increases.

Model Checking: Model checking [8] is the process of determining whether all computations of a model M satisfy a correctness property given the model M. A model checking tool accepts system requirements or designs (referred to as *models*) as well as a property (referred to as *specification*) that the final system must meet. If the given model meets the given parameters, the tool returns yes as shown in Figure 1.1. Otherwise, it returns a counter example. The counter example explains why the model fails to meet the requirements. By examining the counter example, one can identify the source of the model's inaccuracy, rectify it, and try again. The concept

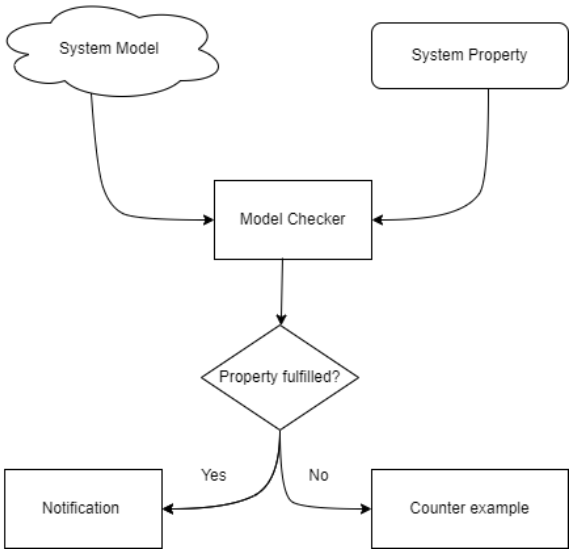

Figure 1.1: Model checking

is that by ensuring that the model meets a sufficient number of system attributes, we may boost our confidence in its correctness.

1.2 Runtime verification and enforcement approaches

Runtime verification: As a complement to theorem proving and model checking, runtime verification [15] (dynamic, i.e., executed at runtime) is being investigated. It is described as the branch of computer science that studies, develops, and applies verification techniques [12] to determine if a run of a system under inspection satisfies or violates a certain correctness property. In most cases , the runtime verification procedure is divided into four steps [8].

1. A monitor is created from a formal property. This phase is also known as monitor creation. It can consume events emitted by a running system and issue judgements depending on the property's present satisfaction and the history of events it has received as shown in Figure 1.2.

2. Instrumentation is present in the system under investigation. This stage's goal is to be able to generate relevant events that will be fed to the monitor. System instrumentation is another name for it.

3. In this stage is execution of the system, emitting events for the monitor.

4. The monitor analyses the happenings in the fourth stage and provides a verdict.

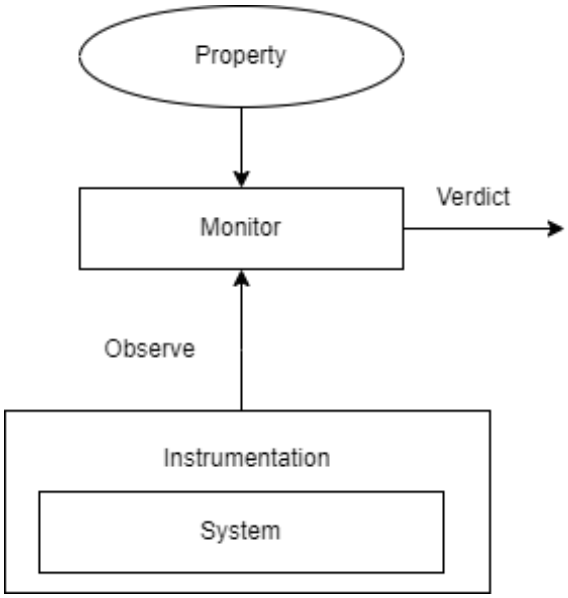

Figure 1.2: Runtime verification

Monitoring can be done online (when the system is running) or offline (when the system is not functioning) (while system is in running mode). And, depending on where the monitor is placed, it can be inline (when it's written into the system's code) or outline (monitor is an external entity). A correctness property is immediately transformed into a monitor in runtime verification, which reads a finite trace and returns a certain conclusion. The accuracy properties are commonly expressed using a linear temporal logic variation (LTL).

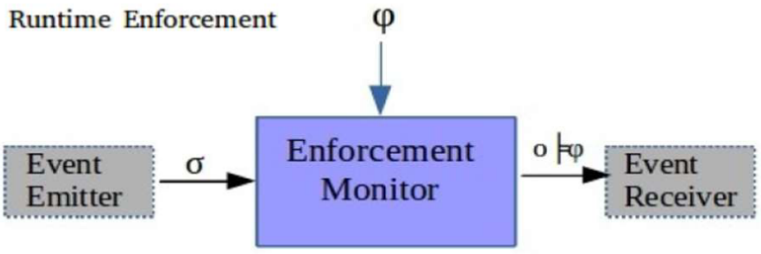

Figure 1.3: Runtime enforcement

Runtime Enforcement: Runtime Enforcement (RE) [12] builds on RV, in which monitors are automatically synthesised by modifying an (untrustworthy) input se-

quence into an output sequence that complies with a property, as shown in Figure 1.3. In the event of a violation, the enforcer takes evasive measures in order to avoid the violation. Blocking the execution, altering the input sequence by delaying, suppressing and/or introducing activities, and buffering input actions until a later time when they can be transmitted are examples of evasive actions. This is how the enforcement monitor works, it has an internal memory (buffer) that stores some events that it receives and then only releases them as output when the property is met.

Chapter 2

Literature Review

Foundation for solving the problem and building basics of all the concepts needed. We will first discuss the concepts, preliminary works related to runtime verification and enforcement. Then, runtime enforcement of untimed properties is explained briefly. After that we will discuss runtime enforcement of timed properties. We have also discussed TiPEX tool which is used for the enforcement of real-time properties. This representation depicts the software implementation of enforcement monitors(EM).

2.1 Literature Survey on Runtime Verification

For model checking and testing, runtime verification of attributes written in lineartime temporal logic (LTL) or timed lineartime temporal logic (TLTL) [3] is used to determine their differentiating features. It proposes a four-valued semantics (with truth values: conclusive (true, false), inconclusive (currently true, currently false) as a sufficient interpretation for determining if a partial observation of a running system satisfies an LTL or TLTL feature. A conceptually simple monitor generation procedure is provided for LTL, which is optimal in two ways: first, the size of the generated deterministic monitor is kept to a minimum; and second, the monitor identifies a continuously monitored trace as satisfying or falsifying a property as early as possible. A collection of real-world temporal logic specifications is used to demonstrate the practicality of the created methodology.

The US Food and Drug Administration (FDA) alerting people to a voluntary recall recently for about 465,000 pacemakers that were hackable. Hackers could either fast

pace the gadgets, causing arrhythmia, or drain the batteries, according to reports [13]. Such measures would put the patient's health and well-being in jeopardy. In light of this, strategies for ensuring the security of implantable medical devices are becoming increasingly popular. Existing methodologies lack the formal rigour required to assure such systems' safety and security. While there are ways for formal verification of pacemaker software, they are insufficient to prevent security flaws. A wearable device based on run-time verification proposes non-invasively sensing familiar ECG patterns to detect if a pacemaker has been compromised. A set of timed policies that can be monitored in real time [13], as well as a design technique for the wearable device.

2.2 Literature Survey on Runtime Enforcement

Discusses the policy space that can be enforced by monitoring programme run-time behaviour. An edit automaton combines the powers of suppression and insertion automata [9]. It is able to truncate action sequences and insert or suppress security-relevant actions at will. Truncation automata are too general and edit automata are too powerful. Mandatory results automata are obligated to return. A result to the target application before seeing the next action it wishes to execute. Runtime enforcement is a powerful technique to ensure that system must satisfy the required property. It must alter the input sequence in such a way that enforced timed word should be accepted by the TA.

2.3 Runtime Enforcement of Untimed Properties

A general RE framework for untimed properties is proposed in [5], and the procedure of runtime enforcement is systematically shown, in its general form, in the Figure 2.1 is illustrated here. e.g., an access control mechanism is input series which is produced by a system and output series is forwarded to a secured server. The runtime EM is a decision method committed to a property Π. It takes a finite as well as infinite order of events σ as input and produces in output a new infinite or finite sequence o. The enforcement monitor(EM) has a buffer memory and it also have some set of operations on the input events (possibly using the memory). The property given is

responsible for any modification done by the monitor. The enforced output sequence satisfies some constraints like transparency.

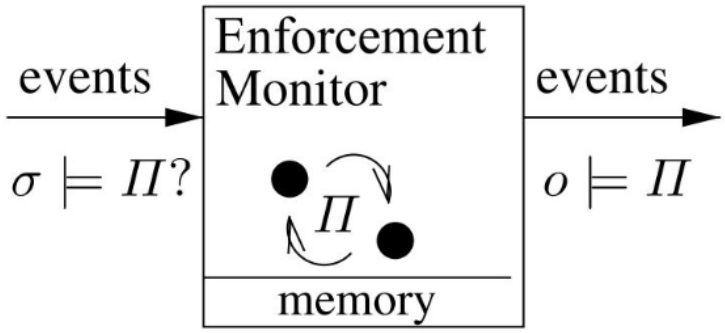

Figure 2.1: Runtime enforcement of the untimed automata

The transparency constraint guides the monitor to store some events of σ till it ensure that the produced output will be satisfying the given property irrespective of any future input sequence. On the other hand, at any moment when the property Π gets satisfied, the monitor simply dumps directly at any moment, when the events in the buffer followed by the new event observed at that moment will satisfy the property, then all the stored events along with the new received event will be released as output by the enforcer. In few distinct cases, by inspecting Π, The monitor may also decide that, whatever future events may occur, the input sequence will never satisfy the property at a certain time. This input sequence can be completely blocked in this scenario (resp. the monitor can be turned off, as it is not required anymore. Approaches such as [5] focus on how to automatically generate/synthesize a RE monitor that satisfies soundness and transparency from the given property formalized as an automaton.

2.4 Runtime enforcemen of Timed Properties

Timed properties are expressed by timed automata (TA). These TA accepts real time continuous input and enforcement mechanism (EM) works on it to satisfy output sequence w.r.t. given property. The runtime enforcement challenge is depicted in [5] Figure 2.2, where an EM reads a timed word as input and transforms and outputs

it so that it complies with a timed property used to get the EM, utilising a timed memory that accounts for the real time during which elements have been stored. The series of events is sent to EM via the event emitter. The property is enforced by EM, and the output sequence is sent to the Event Receiver.

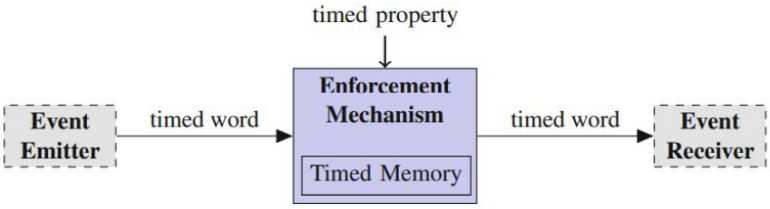

Figure 2.2: Enforcement mechanism (EM)

An EM can be a function or an operational hardware module. In any paradigm EM work is to enforce the result of system which satisfy the properties. Runtime enforcement of timed properties for subclasses of regular properties (i.e. for co-safety and safety Properties). How to get started with RE of safety and co-safety properties was explained briefly in [4]. Enforcement monitors has permission with an enforcement mechanism, which allow delaying or suppressing of events to satisfy the desired property. Because the only operation which is allowed is delaying the events, EM stores some actions in buffer for some computed time period as they are recieved. EMs aim is to ensured the output which satisfies the required property but to also produce the output in optimal time.

2.5 Tools: TiPEX

There are tools such as TiPEX for obtaining enforcement monitors from properties expressed using formalism such as TA. We take brief idea about tool called TiPEX[10], as this is the RE monitor generation tool based on the theory of RE for timed properties proposed in [1] which we consider as a reference for our work.

TiPEX has mainly two sections named as TAG and EMTA-EME. Its a open software which can be used by anyone. We can download all source files from https://srinivaspinisetty.github.io/Timed-Enforcement-Tools/. We have used Class Checker module from TAG section. Class Checker module checks whether the given

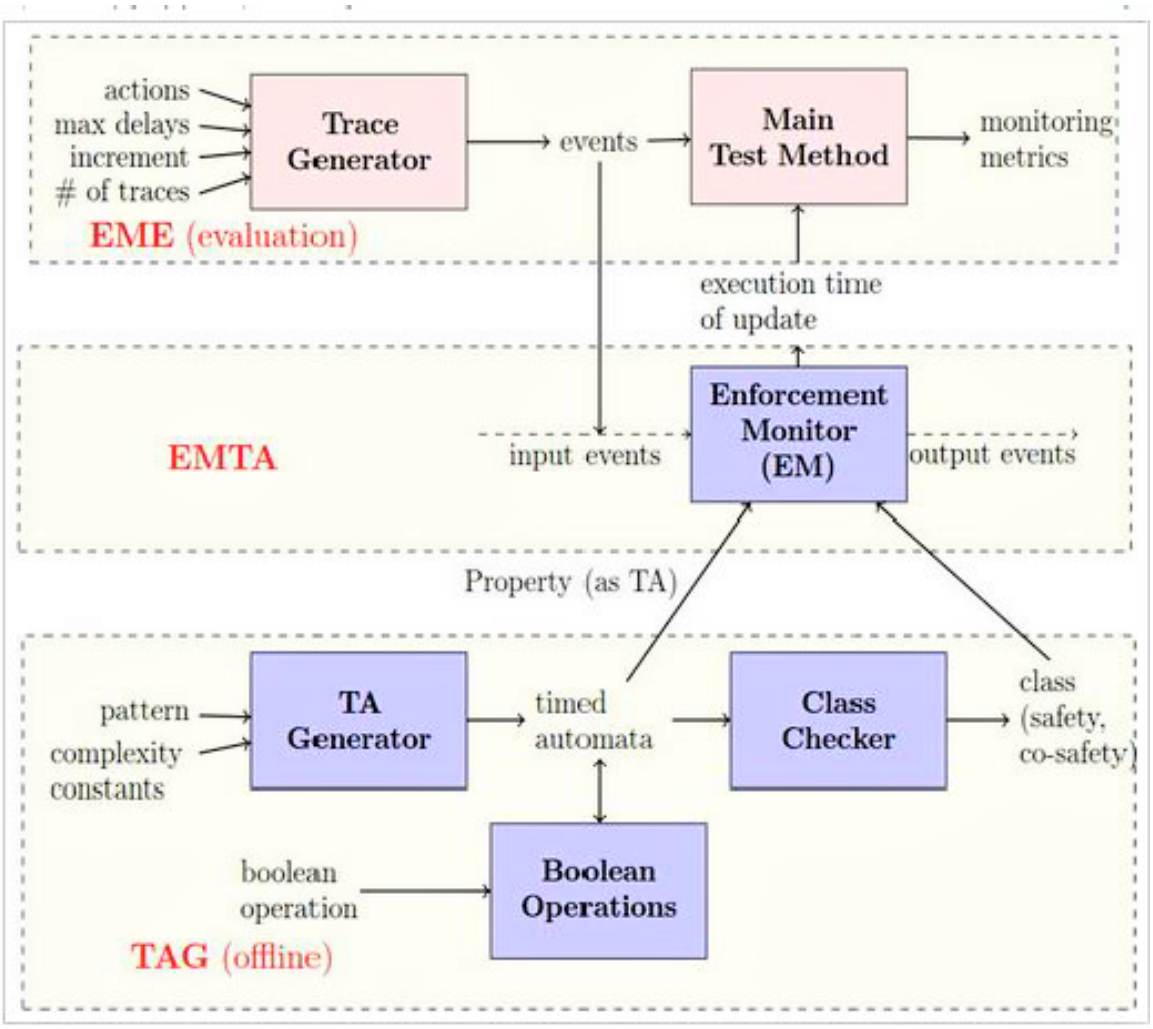

Figure 2.3: Overview of TiPEX [10]

timed automata belongs to safety, co-safety or regular class of property. EMTA-EME has implementation of enforcing timed properties as shown in Figure 2.3. EMTA-EME has implementation of enforcing timed properties.

Chapter 3

Motivation and proposed work

3.1 Motivation

Timed properties stand as a better precise tool to establish desired behaviors of systems since they permit to explicitly state how time should elapse between two events. Runtime enforcement has several application for the timed properties. As enforcement monitors can be used as firewalls to prevent denial of service attacks by ensuring a minimal delay between input events. For any complex safety-critical system, we generally have properties with real-time constraints to be enforced. The theory of runtime enforcement has been studied and frameworks for rendering enforcement monitors from properties characterized as Timed Automata (TA) are proposed. Prototype tools for enforcement monitor generation were developed using other TA-based libraries. However, the TA-based libraries that were used are no more maintained, and pose difficulties to extend the proposed framework.

3.1.1 Limitations of the Existing Tools

Some of the limitations of recent works are highlighted as follows:

- UPPAAL does not support the simulation of double values. To solve this problem designer need to multiply and divide by 100 if required. The simulator does not allow the designer to see clock values because clocks are a logical concept.
- UPPAAL libraries are no longer maintained. Hence sometime it create problem.

- KRONOS verifies clocks using the FDDI protocol, in which only 25 clocks are available. Hence, if the designer wants to use a large number of clocks, KRONOS can not handle it. Simulation states and transitions also have some limits, which is also a problem.

- TiPEX depends on some libraries of UPPAAL, which are no longer maintained. Dependency on UPPAAL creates problems while developing the safety-critical system.

3.2 Problem Statement

Though there are few tools existing for monitoring of real-time systems, they have certain limitations such as platform dependence, use of some libraries that are no longer maintained, and lack of proper user interface for modeling the policies. Thus the aim of this work is to build a tool-chain for monitoring of real-time properties. Some of the key focus aspects are developing an intuitive UI for modeling policies, modularity of various components and maintainability (e.g., supporting easy integration of a new monitoring schemes/algorithms for policies specified as TAs into the framework).

3.3 Proposed architecture

The proposed architecture does runtime verification, enforcement and have some other functionalities such as drawing timed automata, checking determinism and completeness of timed automata. We have design a graphical user interface(GUI) to take input as transition of the timed automata as shown in Figure 3.1. These transition are saved in .txt file and passed to Tchecker. With the help of Tchecker we can obtain zone graph. Tcheker solve reachability problem using zone graph in place of region automaton because of better time complexity. Tcheker gives output in the form of zone graph, which can be further used by the verification and enforcement monitor to carry out runtime verification and enforcement.

Runtime verification deals with the correctness of timed properties. In case of timed automata it verify whether a run is accepted by the automaton or not. It only deals with the detection or violation of the property. RV Monitor behaves like

a device, which reads finite trace and give a certain verdicts. Verdict can be in the form of *conclusive (true, false), and inconclusive (currently true, currently false)* [13], which tells whether property is satisfied or not.

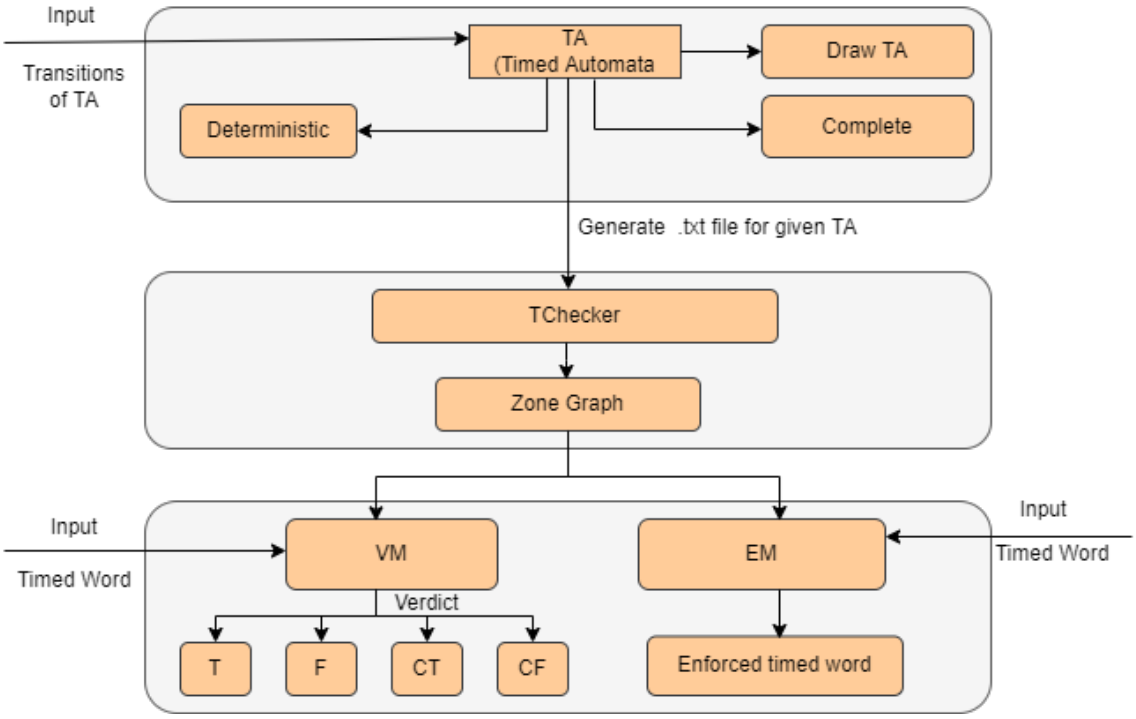

Figure 3.1: Proposed architecture

Runtime enforcement is a technique, which makes modifications accordingly to satisfy the desired property. Enforcement monitor reads timed word and outputs the enforced timed word which is satisfying the property. An enforcement monitor is used to make certain changes in the delays or suppression so that the enforced timed word is satisfying the property. The changes made by the enforcement monitor should be sound and transparent.

Soundness: It means that the resulting output trace should obey the property.

Transparency: It means that if the incoming trace already obeys the property, then the monitor should simply forward that trace to the system.

Chapter 4

Preliminaries and Notations

This chapter will cover all the required basic notations and definitions which are used in our research work.

4.1 Notations

Some important notation which will be useful in understanding the proposed methodology.

- Σ - It denote finite alphabet.
- $|w|$ It represents length of the given word.
- ϵ represents empty word.
- The set of all words over Σ is represented by $\Sigma*$ (respectively $\Sigma+$)
- A language over Σ is a subset of $\Sigma*$.
- pref(w)-set of prefixes of a word w.
- $\mathbb{R} \geq 0$ is used to represents the set of non-negative real numbers.
- An event is represented by a tuple (t,a), where date((t,a)) def = t ϵ $\mathbb{R} \geq 0$ is the absolute time and action act((t,a)) def = a $\epsilon\Sigma$ occurs.
- tw(Σ) represent set of timed words over Σ.
- L $\subseteq$ tw(Σ) - Represent timed language in any set.

- A finite sequence of events $\sigma = (t_1, a_1).\ (t_2, a_2)...(t_n, a_n)$ is a timed word over the finite alphabet Σ, where $(t_i)_{i\in[1,n]}$ is a non-decreasing sequence in $\mathbb{R}$. $\text{start}(\sigma) \stackrel{\text{def}}{=} t_1$ denotes the starting date of σ and $\text{end}(\sigma) \stackrel{\text{def}}{=} t_n$ denote its ending date (with the convention that for the empty timed word ϵ the starting and ending dates are null).

- If the ending date of the first timed word does not exceed the beginning date of the second, concatenation of two timed words is guaranteed.

- G(X) denote set of guards, i.e., guards are defined as Boolean combinations of simple constraints of the form x op c with x $\in$ variable(X), c $\in$ natural number(N) and op $\in \{<, <=, =, >=, >\}$

4.2 Basic Definitions

Definition 1 (Timed automata) A *timed automaton* (TA) is a tuple $\mathcal{A} = (L, l_0, X, \Sigma, \Delta, F)$, in which L is a finite set of *locations* and $l_0 \in L$ denotes the *initial location*. X is used to represents finite set of *clocks*. Σ denotes a finite set of *actions*, $\Delta \subseteq L \times \mathcal{G}(X) \times \Sigma \times 2^X \times L$ is the *transition relation*. $F \subseteq L$ is a set of *accepting or final locations*.

A timed automaton's semantics are defined as a transition system with each state consisting of the current location and clock values. A timed automaton has endless states since the possible values for a clock are limitless. The semantics of a TA is defined as follows.

Definition 2 (Semantics of timed automata) The *semantics* of a timed automata is a *timed transition system* $[\mathcal{A}] = (Q, q_0, \Gamma, \rightarrow, Q_F)$ where $Q = L \times \mathbb{R}^X{}_{\geq 0}$ is the (infinite) set of *states*. $q_0 = (l_0, \chi_0)$ denote the *initial state* where χ_0 is the valuation that maps every clock in X to 0. $Q_F = F \times \mathbb{R}^X{}_{\geq 0}$ represents the set of *final or accepting states*. $\Gamma = \mathbb{R}_{\geq 0} \times \Sigma$ denote set of transition *labels*, i.e., pairs composed of a delay and an action. The *transition relation* $\rightarrow \subseteq Q \times \Gamma \times Q$ is a set of transitions of the form $(l, \chi) \xrightarrow{(\delta,a)} (l', \chi')$ with $\chi' = (\chi + \delta)[Y \leftarrow 0]$ whenever there exists $(l, g, a, Y, l') \in \Delta$ such that $\chi + \delta \models g$ for $\delta \in \mathbb{R}$.

4.3 Preliminaries for Runtime Enforcement

Let $t \in \mathbb{R}$, and a timed word $\sigma \in (\mathbb{R} \times \Sigma)^*$, we define the *observation of* σ *at time* t as the timed word:

$obs(\sigma, t) \stackrel{\text{def}}{=} \max_{\preccurlyeq}\{\sigma'(\mathbb{R}_{\geq 0} \times \Sigma)^* \mid \sigma' \preccurlyeq \sigma \wedge time(\sigma') \leq t\}$,

here $\max_{\preccurlyeq}$ takes the maximal sequence on the basis to the prefix ordering $\preccurlyeq$ (which is unique in this case). That is $obs(\sigma, t)$ is the longest prefix of σ of duration smaller than t. By definition, $time(obs(\sigma, t)) \leq t$, meaning that the duration of an observation at time t never exceeds t.

The *maximal strict prefix of* σ *that belongs to* φ is denoted as $\max^{\varphi}_{\prec,\epsilon}(\sigma)$ and defined as: $\max^{\varphi}_{\prec,\epsilon}(\sigma) \stackrel{\text{def}}{=} \max_{\preceq}\big(\{\sigma'\epsilon(\mathbb{R}_{\geq 0} \times \Sigma)^* \mid \sigma' \prec \sigma \wedge \sigma' \in \varphi\} \cup \{\epsilon\}\big)$.

Orders on timed words. Apart from the prefix order $\preccurlyeq$, we define the following partial orders on timed words:

Delaying order $\preccurlyeq_d$: For $\sigma, \sigma' \in (\mathbb{R} \times \Sigma)^*$, we say that σ' *delays* σ (denoted $\sigma' \preccurlyeq_d \sigma$) iff

$$\Pi_\Sigma(\sigma') \preccurlyeq \Pi_\Sigma(\sigma) \text{ and } \forall i \leq |\sigma'| : delay(\sigma(i)) \leq delay(\sigma'(i)),$$

which means that σ' is "a delayed prefix" of σ. This order will be used to characterize outputs w.r.t. to inputs in enforcement monitoring.

Lexical order $\preccurlyeq_{lex}$: Given any two timed words σ, σ' with same untimed projection, i.e., $\Pi_\Sigma(\sigma) = \Pi_\Sigma(\sigma')$, and any two timed events with identical actions (δ, a) and (δ', a), we define $\preccurlyeq_{lex}$ inductively as follows: $\epsilon \preccurlyeq_{lex} \epsilon$, and $(\delta, a) \cdot \sigma \preccurlyeq_{lex} (\delta', a) \cdot \sigma'$ iff $\delta \leq \delta' \vee (\delta = \delta' \wedge \sigma \preccurlyeq_{lex} \sigma')$. This ordering is defined for timed words with identical actions, and is useful to choose a unique timed word among some with same actions.

4.4 Enforcement Mechanism

Several constraints, namely soundness and transparency, are required on how E_φ transforms timed words. Since our enforcement monitors are time retardants, the physical constraints in the following definition also apply to E_φ.

Definition 3 (Constraints on an Enforcement Mechanism) For a timed property φ, an enforcement mechanism behaves as a function E_φ from $(\mathbb{R}_{\geq 0} \times \Sigma)^* \times \mathbb{R}_{\geq 0}$ to $(\mathbb{R}_{\geq 0} \times \Sigma)^*$, satisfying the following constraints:

Physically time retardant:

$$\forall \sigma \in (\mathbb{R}_{\geq 0} \times \Sigma)^*, \forall t, t' \in \mathbb{R}_{\geq 0} : \ t \leq t' \implies E_\varphi(\sigma, t) \preccurlyeq E_\varphi(\sigma, t') \qquad (\textbf{Phy1}).$$

$$\forall \sigma \in (\mathbb{R}_{\geq 0} \times \Sigma)^*, \forall t \in \mathbb{R}_{\geq 0} : (E_\varphi(\sigma, t)) \leq t \qquad (\textbf{Phy2}).$$

Soundness:

$$\forall \sigma \in (\mathbb{R}_{\geq 0} \times \Sigma)^*, \ \forall t \in \mathbb{R}_{\geq 0} : \ E_\varphi(\sigma, t) \neq \epsilon \implies (\exists t' \geq t : E_\varphi(\sigma, t') \models \varphi) \qquad (\textbf{Snd}).$$

Transparency:

$$\forall \sigma \in (\mathbb{R}_{\geq 0} \times \Sigma)^*, \forall t \in \mathbb{R}_{\geq 0} : E_\varphi(\sigma, t) \preccurlyeq_d obs(\sigma, t) \qquad (\textbf{Tr}).$$

The requirements on the enforcement function are specified by three constraints: physically time retardant, soundness, and transparency. (Phy1) means that the outputs of the enforcement function are concatenated over time, i.e., what is output cannot be modified. (Phy2) expresses that the duration of the output never exceeds t. Soundness (Snd) means that if a timed word is released as output by the enforcement function, in the future, the output of the enforcement function should satisfy property φ. In other words, no event is output before being sure that the property will be satisfied by subsequent events. Transparency (Tr) expresses that, at any time t, the output is a delayed prefix of the observed input obs(σ, t). In [16] other important constraint is mentioned i.e. optimality, meaning that output sequences are produced as soon as possible.

The enforcement function can be described as a composition of functions, each performing the following steps: i) processing the input, ii) computing the delayed timed word satisfying the property, iii) and processing the output sequence. Moreover, the enforcement function describes how these functions are composed to transform an input sequence.

Definition 4 (Enforcement function) The enforcement function for a property φ is $E_\varphi : (\mathbb{R}_{\geq 0} \times \Sigma)^* \times \mathbb{R}_{\geq 0} \to (\mathbb{R}_{\geq 0} \times \Sigma)^*$ defined as:

$$E_\varphi(\sigma, t) = \ obs\Big(\Pi_1\big(store_\varphi(obs(\sigma, t))\big), t\Big).$$

where:

- $store_\varphi : (\mathbb{R}_{\geq 0} \times \Sigma)^* \to (\mathbb{R}_{\geq 0} \times \Sigma)^* \times (\mathbb{R}_{\geq 0} \times \Sigma)^*$ is defined as

$$\begin{aligned} store_\varphi(\epsilon) &= (\epsilon, \epsilon) \\ store_\varphi(\sigma \cdot (\delta, a)) &= \begin{cases} (\sigma_s \cdot \min \preccurlyeq_{lex,time} K, \epsilon) & \text{if } K \neq \emptyset, \\ (\sigma_s, \sigma_c \cdot (\delta, a)) & \text{otherwise,} \end{cases} \\ & \text{with} \\ & \quad (\sigma_s, \sigma_c) = store_\varphi(\sigma), \\ & \quad K = \kappa_\varphi(time(\sigma) + \delta, \sigma_s, \sigma_c \cdot (\delta, a)), \end{aligned}$$

- $\kappa_\varphi(T, \sigma_s, \sigma_c)$ is the set defined as:

$$\begin{aligned} \kappa_\varphi(T, \sigma_s, \sigma_c) \stackrel{\text{def}}{=} \{w \in (\mathbb{R}_{\geq 0} \times \Sigma)^* \mid & \; w \preccurlyeq_d \sigma_c \wedge |w| = |\sigma_c| \wedge \\ & \; \sigma_s \cdot w \models \varphi \wedge delay(w(1)) \geq T - time(\sigma_s))\}, \text{ and} \end{aligned}$$

- $min \preccurlyeq_{lex,time}$ stands for minimal timed word according to the lexical order among the timed words with minimal duration.

In [16] it is proved that the enforcement function as per definition 4 satisfies the soundness, transparency, physical time retardant and optimality constraints (as mentioned in definition 3)

Chapter 5

Designed Tool

The architecture discussed in chapter 3, Figure 3.1 has been implemented into three parts:

1. In section 5.1, we have discussed about the Graphical user interface(GUI) or front end for the user. Intuitive UI has been designed to give input in the form of transitions. We add feature to draw the TA, checking the determinism and completeness of the given timed automata.

2. In section 5.2 , we will discuss about the reachability problem, region automaton and zone graph.

3. In section 5.3, we have discussed about the verification of the timed word and enforcement for the input timed word to obtain enforced timed word.

5.1 Graphical User interface(GUI)

Front end refers to "user interface", which is visible to the user. User interface(UI) design is related to user experience. The aim of user interface design is to make digital interaction as simple, fluid, intuitive and efficient as possible. Key focus of intuitive UI is modeling policies, modularity of various components and maintainability (e.g., supporting easy integration of a new monitoring schemes/algorithms for policies specified as TAs into the framework). User can give input as transitions to as shown in the Figure 5.1. We can see few section as describe below:
Alphabets(Σ): Finite word or a finite sequence of action.

States(L): Name of the states in the TA.

Initial State(l0): It denote initial state of the TA, it can have only one state.

Final State(F): It represents final states of the TA.

Clocks(X): It is used to represents clocks, which we have used as a guard and reset for in the transitions of the TA.

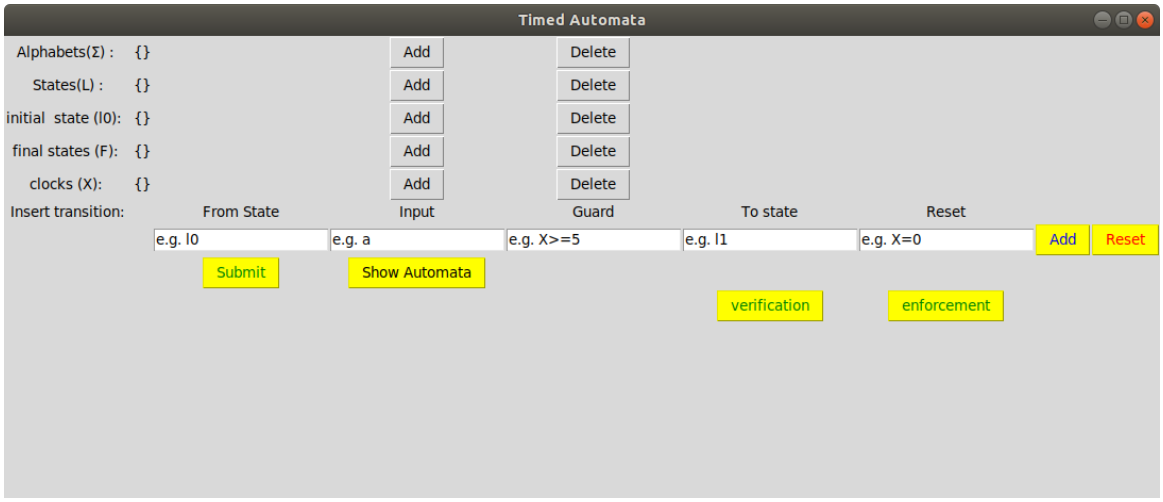

Figure 5.1: GUI fro transitions input of TA

We have added few buttons which is which is described as follow:

Add: Add button is to add particulars.

Delete: Add button is to delete particulars.

Reset: It is used to reset the input transition.

Insert transition: User can give the input in the form of transition as l,a,G,l',R . Here l if from state, a is action, G denote guard, l' denote to state and R is used to represent reset.

Show automata: To draw graphical representation for the entered transition of timed automata.

Submit: It is used to check determinism and completeness of TA.

*Verification :*To verify the input trace with the given timed automaton.

Enforcement: Used to do enforcement and get enforced timed word as output.

5.1.1 Timed automata

A timed automaton is a finite automaton is a graph with a finite number number of locations and a finite number of labelled edges that has real-valued variables added to it and having reset property. An automata can be considered a conceptual description of a timed system. The variables reflect the system's conceptual clocks, which begin at zero when the system is turned on and then increase in lockstep. The automaton's behaviour is limited by clock limitations, sometimes known as edge guards. A transition depicted by an edge can be taken when the clock values satisfy the guard labelled on the edge. Clocks may be reset to zero when a transition is accomplished.

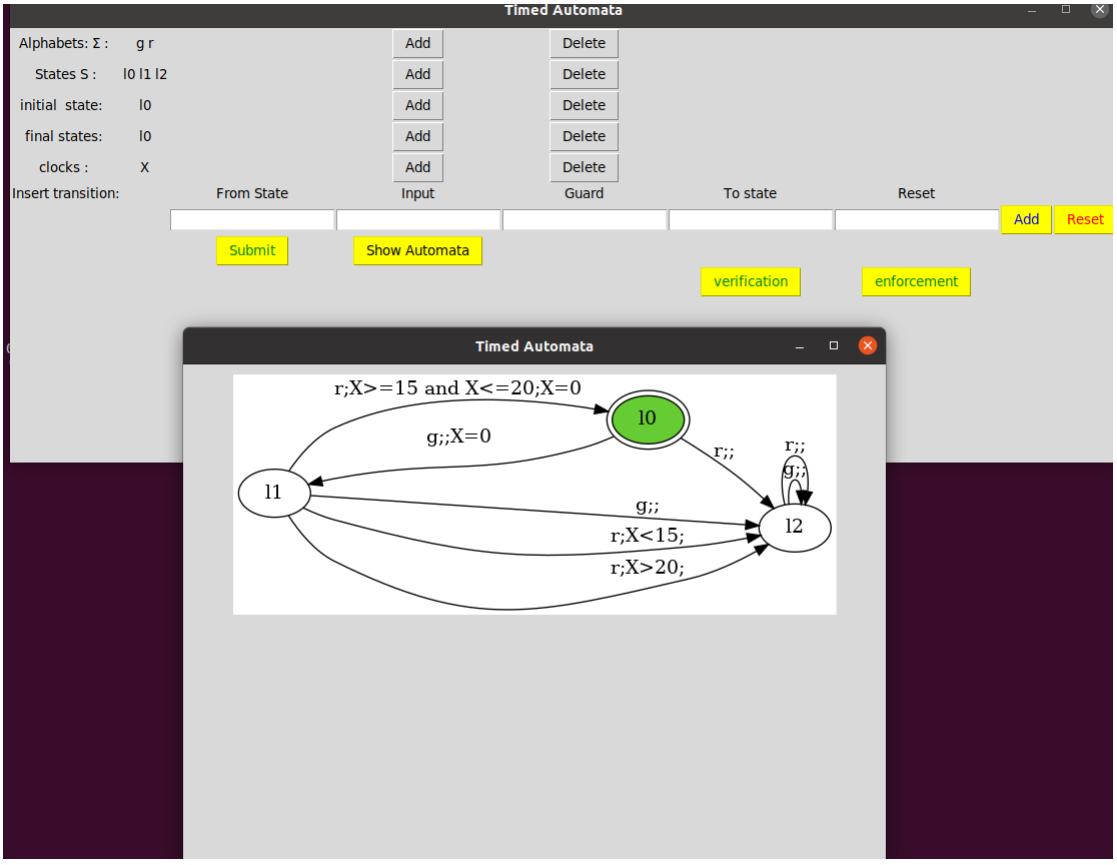

Figure 5.2: Timed automata Z_1 drawn with the help of designed tool.

The graphical user interface(GUI) drawn in Figure 5.1 is used to take input from the user. Users can enter input in the form of transitions with the help of add button. The timed automata shown in Figure 5.2 is obtained by clicking on the draw automata button with the help of designed tool, once all the transitions are entered. In Figure 5.2 state with green color denotes initial state and state having double circle denote

final states. States having only one circle are non-final states.

5.1.2 Determinism of timed automata

Deterministic timed automata are discussed in this section. Given an extended state and the next input symbol's time occurrences, the extended state following the next transition should be determined uniquely. Multiple transitions can start in the same state and have the same label, but their clock restrictions must be mutually exclusive.

We can conclude that only one these transitions are enabled at a perticular time.

Definition 5.1 A timed transition table < Σ, S, So,C , E> is called deterministic iff

(i) $|So| = 1$, It can have only one initial state and

(2) for all s∈S, for all a∈C, for every pair of edges of the form <s, , a, ,δ_1> and <s, -, a, -, δ_2>, here these clock constraints δ_1, and δ_2 should be mutually exclusive (i.e., $\delta_1 \wedge \delta_2$ is unsatisfiable). A timed automaton is deterministic iff its timed transition table is deterministic.

5.1.3 Completeness of timed automaton

Let at any location l∈L of timed automaton and any action a∈ Σ, the disjunction of the transition leaving l and action a evlauates to true. If this condition is satisfy to every state then automaton follow completeness.

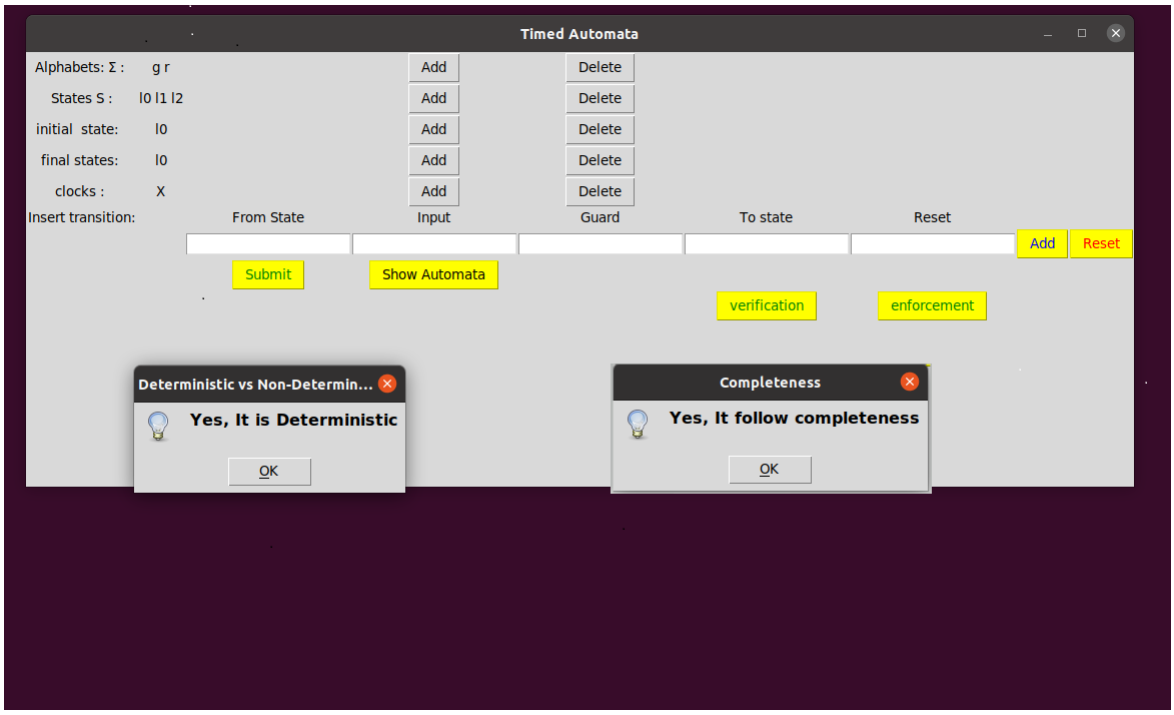

Figure 5.3: Verifying determinism and completeness of timed automaton Z_1.

Completeness and determinism are also verified with the help of the designed tool (by clicking on the submit button on the GUI) as shown in Figure 5.3 for the timed automata Z_1. Edge denotes the transition from one state to another with three parameters on it. The first parameter denotes action, the second one denotes guard and the third parameter denotes reset.

5.2 Reachability problem or emptiness of language

Given an automaton, check wheather language of that automaton is empty?
This problem is known as *reachability* problem or *emptiness* for given timed automaton. Here we need to check whether we can reach from initial state to final state. Solution to this problem is to check whether we can reach to dead state at point of time by any transition in the given timed automaton.

In Figure 5.5 we can see that there is infinite transition possible at a particular state. To solve reachability problem, we needs to sequence of enabled transition through which we can reach to final state. We need to check if the values of the

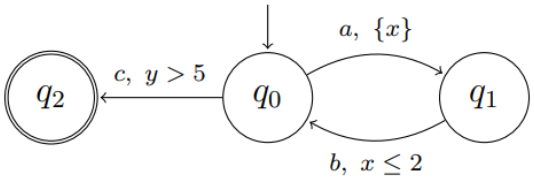

Figure 5.4: Timed automaton Z_1

clocks before taking the transition is satisfy the corresponding guard to know whether transition is possible or not.

In this way we can handle the problem of infinite transition effectively and efficiently.

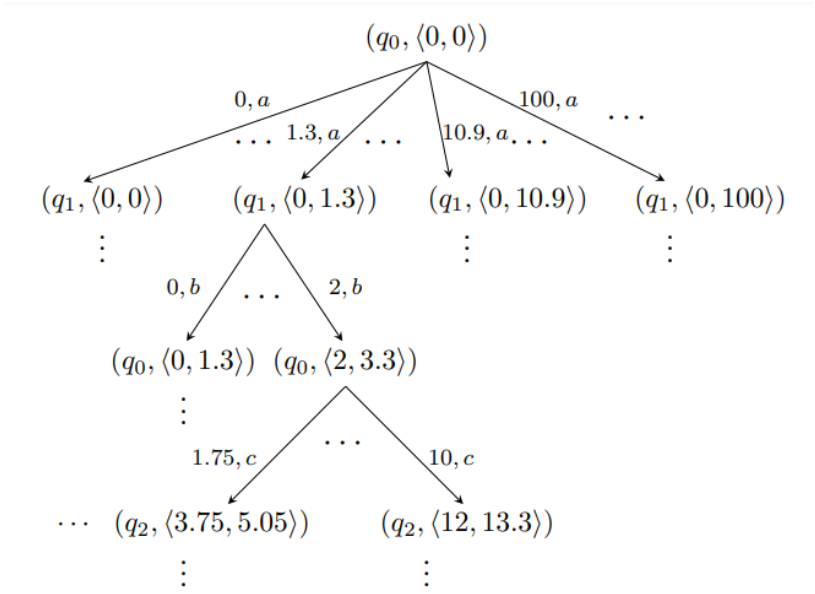

Figure 5.5: Part of the transition system Z_1 as shown in Figure 5.4[17]

Solution to this problem is to find a finite approximation transition system by grouping valuations together. The grouping should be done in sucha way that v and v' are in the same group if all the states of the automaton that are reachable from (q, v) and (q, v') are the same, for all q. Here we should not care about the intermediate time spent in a run. We focus only on v' should be able to take the same sequence of transition as v.

5.2.1 Region automaton

The grouping that we need to solve the rachability problem is given by *region automaton*[1]. The space valuations given by region automaton is partitioned into *regions*. In terms of reachability, two valuations within a region are indistinguishable. After forming a finite number of regions, a cross product with the automaton's states is used to produce state-augmented regions. These state-augmented regions serve as nodes in the automaton's region graph, which defines transitions in a natural fashion based on the valuations contained in the region. The analysis of the automaton is then perform using this finite region graph as shown in fig: 5.6.

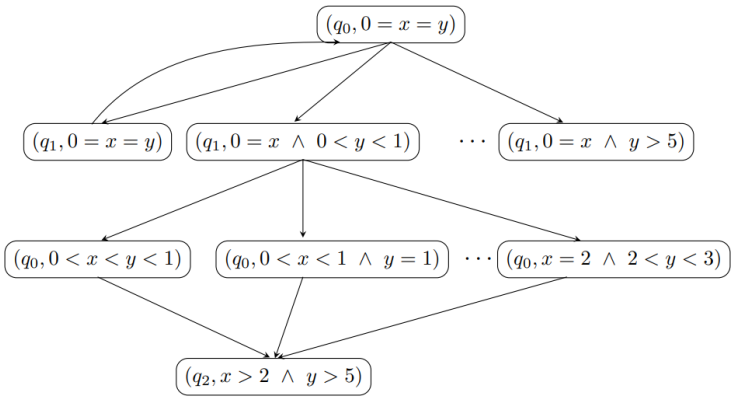

Figure 5.6: Region graph automaon Z_1 as shown in Figure 5.4[17]

Automaton Z_1 has an acceptoing run iff there is path in the region graph of Z_1 starting from its initial state to an accepting state. The region obtain using the bound function is $(|X|!.2^{|X|}.\Pi_{x \in X}(2M_x+2))$[17]. The complexity is very high so we have more efficient solutions for reachability problem is to solve using zone graph.

5.2.2 Reachability : Using zones

Region automaton is very complex, time complexity is not good enough to solve the problem. So to solve the problem by designing an algorithm in such a way that a timed automaton Z_1 constructs a finite graph [18] Graph(Z_1) with some accepting nodes as shown in Figure 5.7, which satisfy these two properties:

Soundness: If an accepting node in Graph(Z_1) is reachable, then there is a run of Z_1 that reaches an accepting state.

Completeness: If an accepting state is reachable in Z_1 then an accepting node is reachable in Graph(Z_1).

Let we have an Graph(Z_1) then we can search for accepting state using standard breadth-first search or depth-search search methods for an accepting run. The obtained Graph(Z_1) should be as small as possible, such that it can be computed efficiently.

Zone graph: Let us assume thet a timed automaton (Z_1)=(Q,q_0,X,T,Acc), the zone graph ZG((Z_1)) of a transition system. The nodes are of the form (q,Z) with q ∈ Q and Z a zone. The starting node is (q_0,Z_0) where $Z_0 = \{0+\delta \mid \delta \in R_{\geq 0}\}$ is the set of valuation obtained by elapsing time from 0.

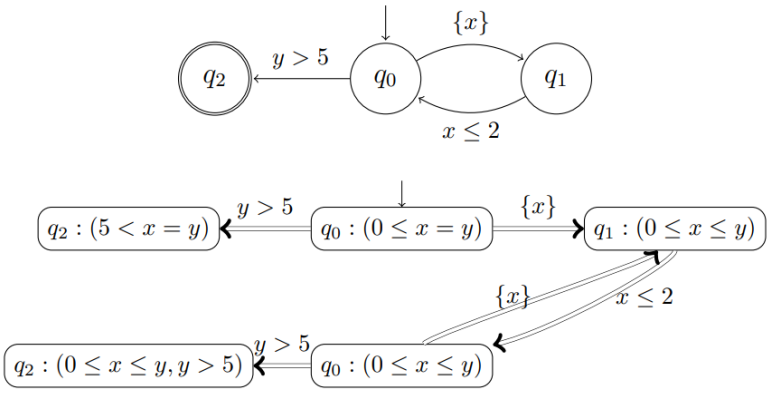

Figure 5.7: An automaton with a part of its zone graph [17]

5.2.3 Tchecker

Tchecker is a open source model-checking tool. It is designed for real timed systems developed at LabRI by Frédéric Herbreteau, Gerald Point and Thanh-Tung Tran. Tchecker has been designed for experimentation of the verification algorithms. It has collection of data structure, libraries and algorithms etc. to implement verifying algorithm. It implements emptiness and buchi emptiness algorithms for timed automata. We have used Tchecker to obtain zone graph in our tool. It can directly downloaded with the help of given link https://github.com/ticktac-project/tchecker.

5.3 Runtime verificaton and enforcement

We'll look at how to use the theory of timed automata to verify the correctness of finite-state real-time systems in this part. We've picked a simple verification issue formulation, but it's sufficient to demonstrate how a timed automata can be used to solve verification problems. We start with time in linear trace semantics for concurrent processes.

5.3.1 Trace semantics

In trace semantics, we associate a set of observable events with each process, and we model the process using the set of all its traces. A trace is a (linear) sequence of events that can be seen when a process is running. For example, an event could be the assignment of a value to a variable, the pressing of a control panel button, or the receiving of a message. All events are expected to occur simultaneously. Modeling activities [7] with a period uses events that represent the start and end of an action.

A trace is a collection of occurrences. As a result, if two events a and b happen at the same time in our model, the corresponding trace will have a set a, b. In normal interleaving models, this set will be replaced by all possible sequences, namely a followed by b and b followed by a. Furthermore, we exclusively look at infinite sequences, which depict the nonstop interaction of reactive systems with their environment.
Let set of A of events, a trace σ=$\sigma_1\sigma_2$... .

5.3.2 Adding timing to traces

Untimed processes replicate the sequence of events but not their exact times. We choose a collection of real numbers to represent time. We know that *time sequence* $\gamma = \gamma_1\gamma_2...$ is a infinite sequence of timed values $\gamma_1 \in R$ satisfying the strict monotonicity and progress constraints. A timed tracing of a sequence of events σ is a trace over A, and γ is a time sequence, and A is a pair (σ, γ). Because distinct events occurring at the same time appear in a single element in a trace, there's no need to enable adjacent elements in a trace to have the same associated time value.

In Figure 5.8 we can see trace $\rho_p = (a, 3.2) \rightarrow (c, 5.1) \rightarrow (b, 8.2)...$

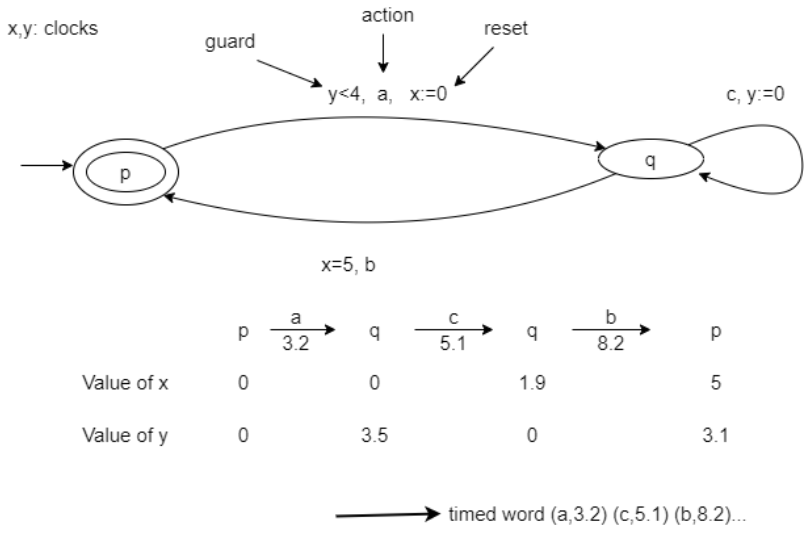

Figure 5.8: Example of trace with automaton

5.3.3 Runtime verification

Runtime verification[7] deals with the development of application of those verification technique that can check whether a run satisfy the property or violates. In case of transition system it tells whether a run is accepted by the automaton or reject. It only deals with the detection of the violation of the property. It does not change the program's execution. In case if timed automata it only check whether run is accepted by the automaton or not. It does not correct the input trace.

RV monitor: Monitor is behave like a device, which reads finite trace and give a certain verdict. Monitor can be operate either online or offline mode. In onliine mode the input is accepted in the lock-step manner while in offline mode by reading log of the system events [11]. Monitor observe the trace and apply the property whether property is satisfied or not outputting in the form of verdict in four form *True, False, currently true, currently false*[13]. True, false are conclusive verdicts and currently true, currently false are inconclusive verdicts.

If verdict is true, it state that run is accepted by the automaton and if verdict is false, it tells that run is reject by the automaton.

5.3.4 Enforcement

Verfication can only check whether run is accepted or rejected by the system. It can not change the input trace so that it can be accepted by the timed automata. In runtime enforcement, an enforcement monitor is used to make certain changed in the delays or suppression so that input trace is accepted by the system [5]. The changes made by enforcement monitor should be sound and transparent.

Soundness: It refers that resulting trace should be obey the property.

Transparency: It means that resulting trace already obey the property, then changes made by monitor should be minimal.

Tchecker is used to obtain a zone graph. Designed tool performs runtime verification as shown in Figure 5.9. We have a given timed word $\rho_p = (g, 10) \rightarrow (r, 16) \rightarrow (g, 1) \rightarrow (r, 15)$. This timed word is given input to the verification monitor, which is accepted by the timed automata Z_1 (as it reach at the final state). As the first event is (g,10), it takes to transition from l0 to l1 and the clock value of X is reset to 0. The second event is (r,6), which took the transition from l1 to l0. The actual clock value is 10+26=16 but the clock value of X is only 16 because it is reset to 0 while taking the previous transition l0 to l1 state. After reaching l1 clock value is reset to 0 again. Here relative ordering is used for the clock valuation. The third event is (g,1), the clock value of X becomes 1. It takes to transition from l1 to l0 and the clock value of X is reset to 0. The last event is (r,1), it took the transition from l1 to l0 as clock value of X is rest to 0.

Enforcement monitor take input as timed word and make certain changes such as delay and suppressing the event and give output as enforced timed word. Enforced timed word must satisfy the desired system properties. We have a given timed word $\rho_q = (g, 10) \rightarrow (r, 5) \rightarrow (g, 3) \rightarrow (r, 10)$. This timed word is given input to the enforcement monitor. The enforced timed word is $\rho_q = (g, 10) \rightarrow (r, 15) \rightarrow (g, 0) \rightarrow (r, 15)$.

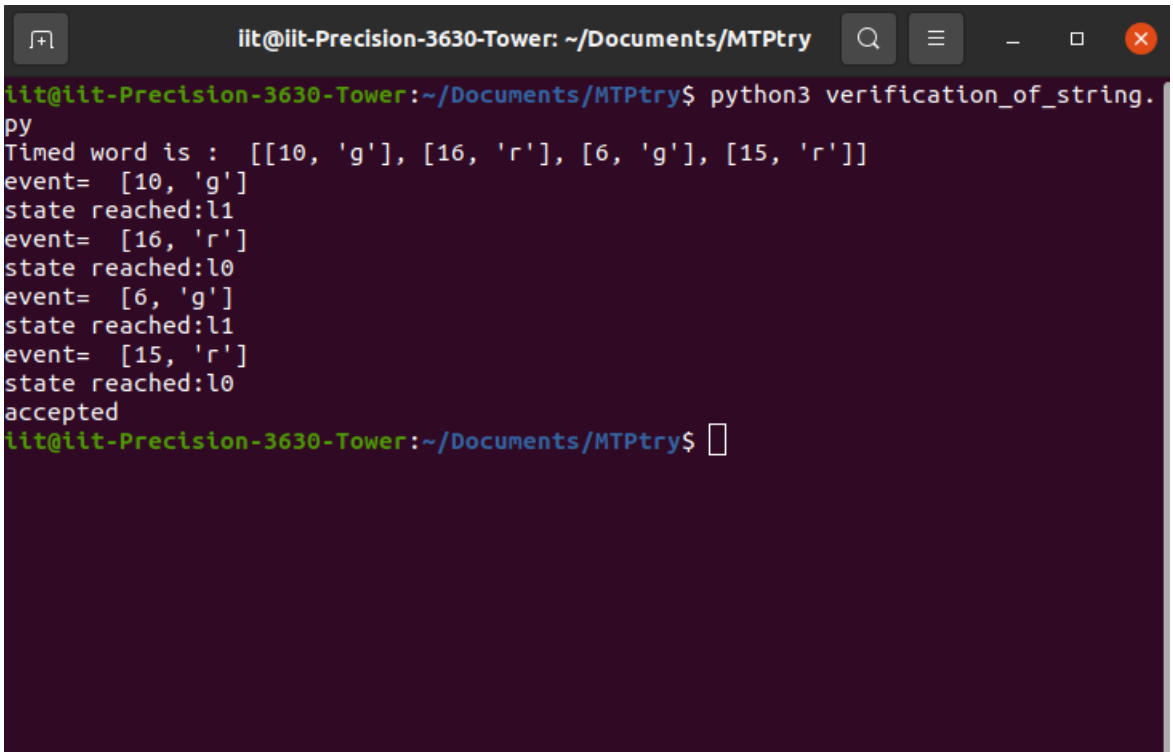

Figure 5.9: Verification of timed word ρ_p with timed automaton Z_1.

Chapter 6

Conclusion

Runtime verification and enforcement approaches are used to safeguard safety-critical systems these days. Timed automata are used to model the behavior of these systems. We have designed a framework which takes the timed properties, modelled by timed automata and constructs verification and enforcement monitors out of it, which will detect any violation of the property or even prevent those violations when the monitors are presented with an input trace.

We have developed a graphical user interface to take transitions of the timed automata as input from the user. We can further see the graphical representation of timed automaton and can also check for the determinism and completeness of that timed automata. We then employed Tchecker to build the zone graph to get the rechability analysis. In case of verification of a property, the zone graph is traversed and the verdicts are given for an input trace by the verification monitor and in case of enforcement of a property, the zone graph is traversed and the input trace is corrected according to the property by the enforcement monitor.

Chapter 7

Future work

The existing tools for monitoring of real-time systems have certain limitations such as platform dependence, use of some libraries that are no longer maintained, and lack of proper user interface for modeling the policies. Thus a tool-chain for monitoring of real-time properties have been designed. Some of the key focus aspects are developing an intuitive UI for modeling policies, modularity of various components and maintainability. Some features that can be added in the future are as follow:

- Integration of new methods to draw the timed automata e.g. Drag and Drop approach.
- Addition of modularity of various TA related operation such as cross product etc.
- Integration of a new monitoring schemes/algorithms for policies specified as TAs into the framework

References

[1] Rajeev Alur and David L Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994.

[2] David Basin, Felix Klaedtke, and Eugen Zălinescu. Algorithms for monitoring real-time properties. In *International Conference on Runtime Verification*, pages 260–275. Springer, 2011.

[3] Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime verification for ltl and tltl. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 20(4):1–64, 2011.

[4] Manfred Broy, Doron Peled, and Georg Kalus. *Engineering Dependable Software Systems*, volume 34. IOS Press, 2013.

[5] Yliès Falcone, Laurent Mounier, Jean-Claude Fernandez, and Jean-Luc Richier. Runtime enforcement monitors: composition, synthesis, and enforcement abilities. *Formal Methods in System Design*, 38(3):223–262, 2011.

[6] Kim G Larsen, Paul Pettersson, and Wang Yi. Uppaal in a nutshell. *International journal on software tools for technology transfer*, 1(1):134–152, 1997.

[7] Martin Leucker and Christian Schallhart. A brief account of runtime verification. *The Journal of Logic and Algebraic Programming*, 78(5):293–303, 2009.

[8] Jay Ligatti, Lujo Bauer, and David Walker. Run-time enforcement of nonsafety policies. *ACM Transactions on Information and System Security (TISSEC)*, 12(3):1–41, 2009.

[9] Srinivas Pinisetty, Yliès Falcone, Thierry Jéron, and Hervé Marchand. Runtime enforcement of regular timed properties. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, pages 1279–1286, 2014.

[10] Srinivas Pinisetty, Yliès Falcone, Thierry Jéron, and Hervé Marchand. Tipex: A tool chain for timed property enforcement during execution. In *Runtime Verification*, pages 306–320. Springer, 2015.

[11] Srinivas Pinisetty, Ylies Falcone, Thierry Jéron, Hervé Marchand, Antoine Rollet, and Omer Landry Nguena Timo. Runtime enforcement of timed properties. In *International Conference on Runtime Verification*, pages 229–244. Springer, 2012.

[12] Srinivas Pinisetty, Thierry Jéron, Stavros Tripakis, Yliès Falcone, Hervé Marchand, and Viorel Preoteasa. Predictive runtime verification of timed properties. *Journal of Systems and Software*, 132:353–365, 2017.

[13] Srinivas Pinisetty, Partha S Roop, Vidula Sawant, and Gerardo Schneider. Security of pacemakers using runtime verification. In *2018 16th ACM/IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE)*, pages 1–11. IEEE, 2018.

[14] Matthieu Renard, Antoine Rollet, and Yliès Falcone. Grep: games for the runtime enforcement of properties. In *IFIP International Conference on Testing Software and Systems*, pages 259–275. Springer, 2017.

[15] Koushik Sen and Sarfaz Khurshid. *Runtime Verification: Second International Conference, RV 2011, San Francisco, USA, September 27-30, 2011, Revised Selected Papers*, volume 7186. Springer, 2012.

[16] Thierry Jéron Hervé Marchand Antoine Rollet Omer Nguena-Timo Srinivas Pinisetty, Yliès Falcone. Runtime enforcement of timed properties revisited. *Formal Methods Syst. Des. 45(3): 381-422*, 2014.

[17] B Srivathsan. Language emptiness for timed automata.

[18] B Srivathsan. Reachability algorithm using zones.