From Scenarios to Optimally Allocated Timed Automata

Sandeep Vuppula

May 29, 2017

University of Minnesota Duluth

Table of contents

Objectives of the Research

Our main focus of the research is,

1. To synthesize a timed automaton from a set of scenarios, and
2. To optimally allocate clocks in the constructed timed automaton.

Introduction

Introduction

- Model-based design is a very effective method for designing real-time systems.
- Modeling a system formally can help us to understand the desired and undesired behaviours of the system.
- Building formal models for systems is challenging because of the lack of good formal requirements specifications.

Introduction

- To construct a formal model, the following questions are to be answered first:
  1. How the requirements should be expressed formally, and
  2. How the formal model of a real-time system can be constructed from requirements.
- The formal model that we build is timed automata [1].

Introduction

- We use scenarios to build a formal model. A scenario is a partial description of the behaviour of a system,
- We propose Timed Event Sequences (TES) to formally represent the scenarios, and
- We use mode graphs to specify the legal events that can occur in the system.

We synthesize a *minimal*, *acyclic* and *deterministic* timed automaton using TES and mode graph.

Introduction

- The number of clocks in a given timed automaton has a direct impact on verification of the system.
- Given a timed automaton $\mathcal{A}$, the problem of deciding whether there exists another timed automaton $\mathcal{B}$ that accepts the same language as that of $\mathcal{A}$ but with fewer number of clocks is undecidable.

We propose a method to optimally allocate clocks in a timed automaton.

Background

Modeling Time

There are three approaches for modeling time [1].

- Discrete time model: Time is considered as discrete and monotonically increasing sequence of integers. Limits the preciseness: in real-time systems, the events do not occur at integer times.
- Fictitious-clock model: It is similar to that of discrete time model except that it assumes sequence of times to be non decreasing integers. Limits accuracy, as the exact time values at which the events occur are not considered.
- Dense time model: In this model, the domain of time is considered as a dense set and the time of occurrences of events as real numbers, which increase monotonically without any limit. Difficulty in transforming dense time traces into formal languages.

Finite State Automata

A finite state automaton (FSA) or a finite state machine (FSM) is an abstract machine which has a finite number of states. On an input, the machine changes from one state to another state.

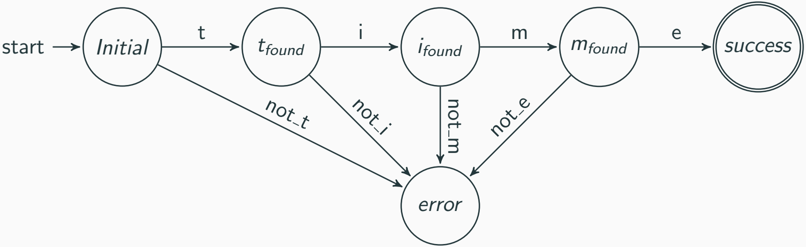


Figure 1: Simple FSM parsing the string “time”

Timed Automata

- A timed automaton [1] is a finite state automaton extended with a finite set of real-valued clocks.
- Upon an input, the selection of next state is based not only on the input symbol but also on the time of the current symbol with respect to the formerly read symbols.

Example: Consider a simple timed automaton in Figure 2. This automaton accepts an input sequence 'a' followed by 'b' such that, there is 2 units of time difference between any two consecutive a's and b's.

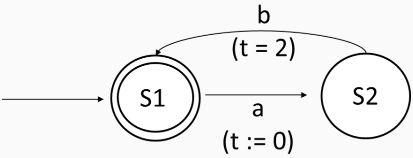


Figure 2: A simple timed automaton

Synthesis of Timed Automata from Scenarios

Synthesis of Timed Automata from Scenarios

Our method of synthesizing a timed automaton model of a real-time system from scenarios involves two steps:

1. Constructing a time annotated graph from scenarios, and
2. Constructing a timed automaton from time annotated graph.

Synthesis of Timed Automata from Scenarios

- Our approach for building a formal model is using scenarios.
  - A scenario is a partial description of the behaviour of a system.
  - A scenario not only describes the events, but also the timing relations among the events.
  - Set of scenarios together can capture the behaviour of the real-time system.
- We use Timed Event Sequences to describe the scenarios formally.
- We use *mode graphs* to specify the legal events that can occur in the system.

Mode Graph

A *mode graph* is a deterministic state machine in which the states are called modes and the transitions triggered by the events in the system. It is a tuple $\mathcal{M} = (M, m_0, m_f, \Sigma, T)$ where,

- M is a finite set of modes,
- m_0 is the initial mode,
- m_f is the final mode,
- Σ is a set of events, and
- $T : M \times \Sigma \to M$ is a transition function.

Mode Graph Example

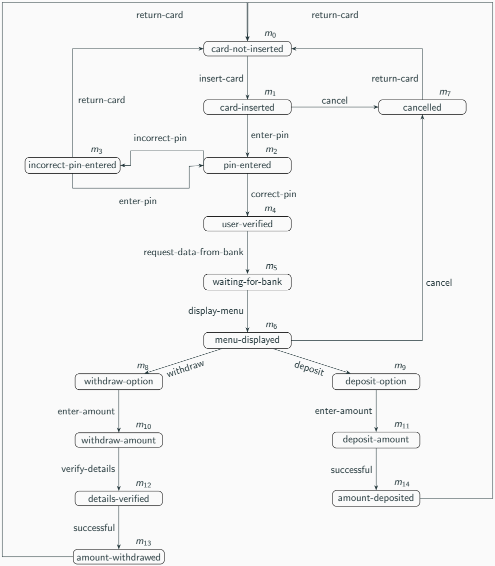


Figure 3: Mode graph of the ATM

Timed Event Sequences

The scenarios describing the partial behaviours of a real-time system are expressed formally in the form of Timed Event Sequences (TES).
A Timed Event Sequence ξ contains:

1. The initial mode of the scenario,
2. The final mode of the scenario,
3. A set of events and their corresponding time annotations.

$m^{initial}$: card-not-inserted

(insert-card, $\{\}$)
(enter-pin, $\{W - t_0 \geq 5,\ W - t_0 \leq 60\}$)
(incorrect-pin, $\{\}$)
(re-enter-pin, $\{W - t_0 \geq 5,\ W - t_0 \leq 60\}$)
(correct-pin, $\{\}$)
(request-data-from-bank, $\{\}$)
(display-menu, $\{W - t_4 \leq 5\}$)

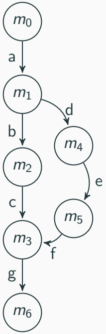
m^{final}: menu-displayed

TES of ATM scenario

Dominance Assumption

- Given two modes m_i and m_j, m_i is said to be the *dominating mode* of m_j iff all the paths to m_j from the initial mode in the mode graph pass through m_i [4]. We call this the *Dominance* relation and denote it as m_i DOM m_j.
- Dominance assumption ensures that time variables are well defined.

Dominance Assumption Example



$m^{initial}$: m_0
$(a,\ \{\})$
$(d,\ \{W - t_0 < 3\})$
$(e,\ \{W - t_1 > 1\})$
$(f,\ \{\})$
$(g,\ \{W - t_0 > 3, W - t_1 \leq 3\})$
m^{final} : m_6

Figure 4: A mode graph and a scenario satisfying the dominance assumption

- m_0 and m_1 are dominating modes of m_3,
- Transition g is dominated by all the modes that dominate m_3,
- t_0 and t_1, on transition g refer to the modes m_0 and m_1.

Constructing a Time Annotated Graph from Scenarios

Given a mode graph $\mathcal{M}$ and a set of Timed Event Sequences $\Xi = \{\xi_1, \xi_2, .., \xi_k\}$ as inputs, our algorithm synthesizes a time annotated graph (TAG) G [5]. Initially we start with an empty graph, G_0 and perform the following steps:

1. Build a partial graph G_1 using the first scenario ξ_1,
2. The algorithm repeatedly takes a partially built graph G_k, and a scenario ξ_{k+1} $(1 < k < n)$ and then generates a new partial graph G_{k+1}

Decision on whether to create new states and transitions is resolved with the help of state labels (modes). A new state s is created and labelled with a mode m_j if there is an event e from state q such that $L(q) = m_i$ and $(m_i, e, m_j) \in T$.

Properties of Constructed Time Annotated Graph

The graph constructed by algorithm has the following properties:

1. It is acyclic,
2. The graph is connected,
3. By construction, two states have the same label only if one is a predecessor of the other,
4. The graph is finite,
5. The graph is deterministic,
6. The graph is minimal,
7. After construction, every scenario is a partial run of the constructed graph, and
8. Every path in the final graph is identical to that of the mode graph.

Constructing a Timed Automaton from Time Annotated Graph

To convert a time annotated graph to a timed automaton we have to:

1. Determine the required number of clocks,
2. Add clock resets, and
3. Replace the time annotations with the clock constraints

Example: If there is a time annotation $W - t_0 > 5$ on a transition, then the clock constraint $c_0 > 5$ is added to that transition and clock c_0 is reset on all transitions from the state labelled with mode m_0.

Example

$m^{initial}$: card-not-inserted

(insert-card, $\{\}$)
(enter-pin, $\{W - t_0 \geq 5,\ W - t_0 \leq 60\}$)
(incorrect-pin, $\{\}$)
(re-enter-pin, $\{W - t_0 \geq 5,\ W - t_0 \leq 60\}$)
(correct-pin, $\{\}$)
(request-data-from-bank, $\{\}$)
(display-menu, $\{W - t_4 \leq 5\}$)

m^{final}: menu-displayed

TES of Scenario 1

$m^{initial}$: card-not-inserted

(insert-card, $\{\}$)
(enter-pin, $\{W - t_0 \geq 5,\ W - t_0 \leq 60\}$)
(correct-pin, $\{\}$)
(request-data-from-bank, $\{\}$)
(display-menu, $\{W - t_4 \leq 5\}$)

m^{final}: menu-displayed

TES of Scenario 2

Figure 5: Timed Event Sequences of the ATM

Example (Cont.)

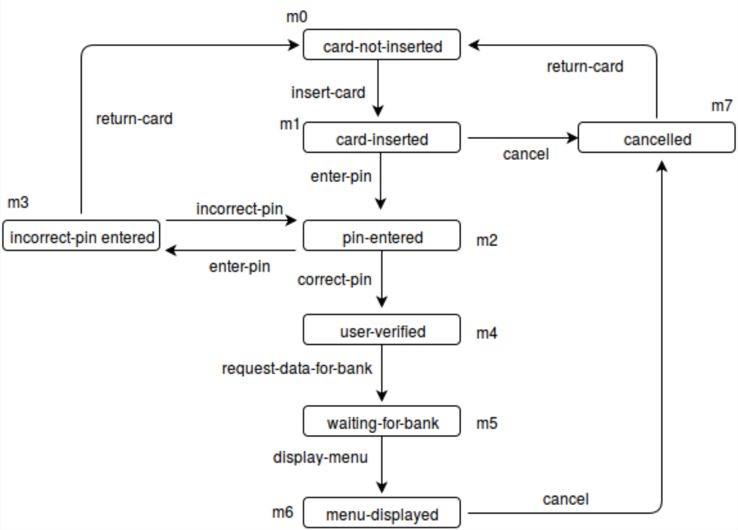


Figure 6: Mode Graph for ATM

Example (Cont.)

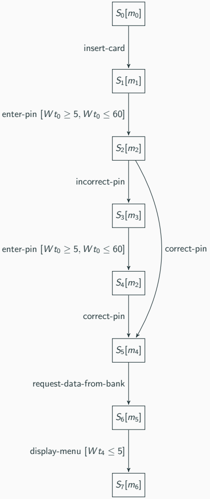


Figure 7: Time annotated graph synthesized from two TES in Figure 5

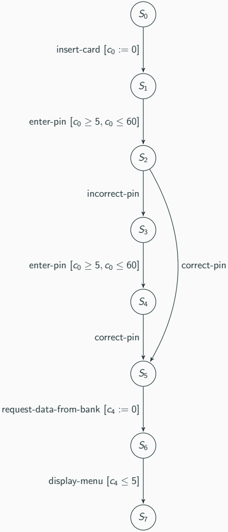


Figure 8: Timed automaton constructed from time annotated graph

Optimal Clock Allocation of Timed Automata

Class of Timed Automata

The timed automaton constructed as a result of our synthesis method belongs to the class of timed automata that satisfies these properties:

1. The automaton is connected and has a unique initial state s_0,
2. A clock constraint on a transition 'r' can only refer to the times of transitions from states that dominate the transition 'r', we call this *dominance assumption*,
3. A clock t_j can only be reset on a transition leaving a state s, where label is j, that is $L(s) = j$.

Optimal Clock Allocation of Timed Automata

To optimally allocate clocks in a timed automaton $\mathcal{A}$ that belongs to our class of timed automata, we need to perform the following steps in order:

1. Calculate the liveness ranges of clocks in the timed automaton $\mathcal{A}$,
2. Replace the original clocks in $\mathcal{A}$ with a set of new clocks,
3. Rewrite the clock constraints and clock resets in $\mathcal{A}$ in terms of new clock variables.

As the problem of deciding if there exists another timed automaton accepting the same language as $\mathcal{A}$ is *undecidable* in general, we do not take into account the satisfiability of clock constraints.

Liveness Range Analysis

Liveness Ranges of all the clocks in a timed automaton helps us determine if a particular clock is needed on these transitions.

- Let $\mathcal{A} = (E, Q, \{q^0\}, Q_f, V, R, L)$ be the timed automaton and $r = (s, s', e, \lambda_r, \phi_r) \in R$ be a transition.
- Let $N = \{j \mid t_j \sim a \in \phi \vee t_j \in \lambda$, where $(s, s', e, \lambda_r, \phi_r) \in R\}$, be a set of *clock numbers* used to denote subscripts of the clocks on all the transitions in R.

Liveness Range Analysis

The following are a set of functions used to calculate the liveness ranges:

- **clock_ref**: $\mathit{clock\_ref}(r)$ is the set of clocks which are referred to in the clock constraints on r.
- **born**: $\mathit{born}(r)$ identifies a clock that is reset on r whose value can be used on some transition reachable from r.
- **active**: $\mathit{active}(r)$ identifies clocks that are "alive" on r (i.e., their values may be subsequently used). Notice that $\mathit{born}(r) \subseteq \mathit{active}(r)$.
- **needed**: Maps transition r to $\mathit{active}(r) \cup \mathit{clock\_ref}(r)$.

Liveness Range Analysis Example

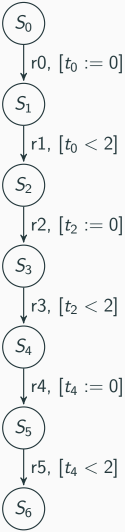


Figure 9: A simple timed automaton

Table 1: *born* and *active* values

Transition	Born	Active
r_0	$\{0\}$	$\{0\}$
r_1	ϕ	ϕ
r_2	$\{2\}$	$\{2\}$
r_3	ϕ	ϕ
r_4	$\{4\}$	$\{4\}$
r_5	ϕ	ϕ

Clock Allocation

The liveness analysis algorithm calculates liveness ranges and generates extended transitions of the form $(r, \mathit{born}(r), \mathit{active}(r))$. Our method to optimally allocate the clocks revolves around the idea that:

- A clock can be reused if the active range of the clock has ended,
- The clock cannot be reused on a transition if the transition belongs to active range of that clock.

Clock Allocation

Definition: Given a timed automaton $\mathcal{A}$ with the set R of (extended) transitions and the set N of clock numbers, a *clock allocation* for $\mathcal{A}$ is a relation $alloc \subset R \times P_0 \times N$ such that $(r, c, j) \in alloc \Rightarrow j \in active(r)$. Where, P_0 is the pool of new clock variables.

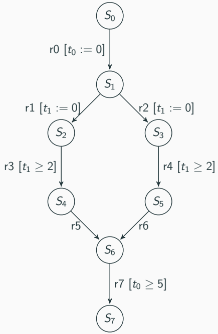


Figure 10: A timed automaton satisfying the dominance assumption

$$alloc = \{(r_0, c_0, 0), (r_1, c_0, 0), (r_2, c_0, 0), (r_3, c_0, 0), (r_4, c_0, 0), (r_5, c_0, 0), (r_6, c_0, 0), (r_1, c_1, 1), (r_2, c_1, 1), (r_3, c_1, 1), (r_4, c_1, 1)\}$$

Problematic States

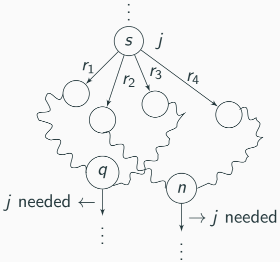


Figure 11: A timed automaton with problematic states

- Clock t_j is born on all outgoing transition of state s.
- r_1, r_3 meet at state q and r_2, r_4 meet at state n.

States q and n are the *problematic* states. So, r_1, r_3 should be assigned the same clock and r_2, r_4 should be assigned the same clock.

Handling the Problematic States

To handle the problematic states, we partition the transitions into mothers and others.

- *mothers*: $\{r \in out(s) \mid j \in born(r)\}$
- *others*: $\{r \in out(s) \mid born(r) = \emptyset\}$

We use the following functions:

- $reachable : Q \to 2^Q$ maps state q to the set of states that are reachable from q by some non-empty path.
- $reachable\_from : Q \to 2^Q$ maps state q to the set of states from which it can be reached by some non-empty path.

Handling the Problematic States (Contd..)

$PP := \emptyset$;   // potentially problematic states

foreach $r \in \mathit{mothers}$ **do**
  foreach $s \in \mathit{reachable}(\mathit{target}(r))$ **do**
    if $L(q) \in \mathit{active}(r')$, *where* r' *is an arbitrary transition of* $\mathit{in}(s)$
    then
      $PP := PP \cup \{s\}$

Those states in PP that can be reached from more than one mother are the problematic states.

Example: Optimal Clock Allocation

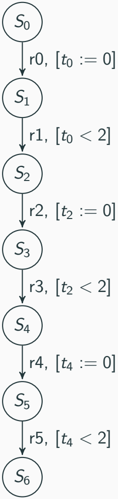


Figure 12: A simple timed automaton

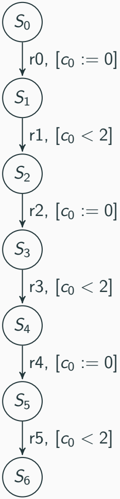


Figure 13: A simple optimally allocated timed automaton

Case Studies

Automated Teller Machine (ATM)

We use an invariant of an ATM:

1. Initially, the ATM waits for a user to insert his card,
2. User has to enter his PIN within 5 to 60 seconds. User has to enter correct PIN in 3 attempts,
3. ATM requests the bank to verify the user's PIN, and
4. If the PIN is correct, ATM displays menu to the user within 5 seconds.

Mode Graph

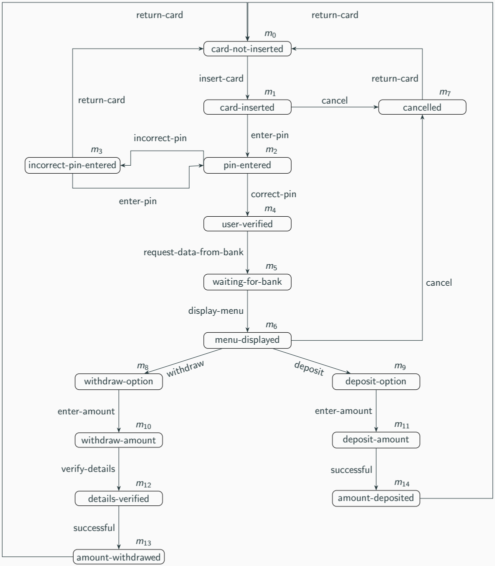


Figure 14: Mode graph of the ATM

Automated Teller Machine (ATM)

$m^{initial}$: card-not-inserted

(insert-card, $\{\}$)
(enter-pin, $\{W - t_0 \geq 5,\ W - t_0 \leq 60\}$)
(incorrect-pin, $\{\}$)
(re-enter-pin, $\{W - t_0 \geq 5,\ W - t_0 \leq 60\}$)
(correct-pin, $\{\}$)
(request-data-from-bank, $\{\}$)
(display-menu, $\{W - t_4 \leq 5\}$)

m^{final}: menu-displayed

TES of Scenario 1

$m^{initial}$: card-not-inserted

(insert-card, $\{\}$)
(enter-pin, $\{W - t_0 \geq 5,\ W - t_0 \leq 60\}$)
(correct-pin, $\{\}$)
(request-data-from-bank, $\{\}$)
(display-menu, $\{W - t_4 \leq 5\}$)

m^{final}: menu-displayed

TES of Scenario 2

Figure 15: Timed Event Sequences of the ATM

Automated Teller Machine (ATM)

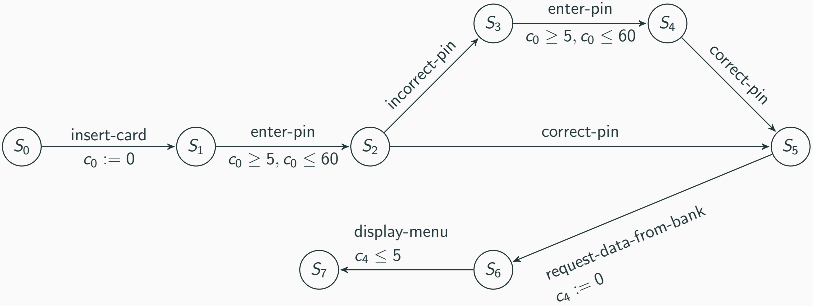


Figure 16: Timed automaton synthesized from Scenario 1 and Scenario 2

Automated Teller Machine (ATM)

Now consider an extended behaviour of the ATM machine. After the menu is displayed, assume that, the user can choose from the two available options:

Deposit:

1. User enters the amount to deposit, and
2. ATM returns success message if the deposit is successful.

Withdraw:

1. User enters the amount to withdraw,
2. Bank verifies the user details, and
3. Returns success message if the withdraw is successful.

Automated Teller Machine (ATM)

TES of Scenario 3	TES of Scenario 4
$m^{initial}$: menu-displayed	$m^{initial}$: menu-displayed
(deposit, $\{\}$)	(withdraw, $\{\}$)
(enter-amount, $\{W - t_6 \leq 20\}$)	(enter-amount, $\{W - t_6 \leq 20\}$)
(successful, $\{W - t_9 \leq 10\}$)	(verify-details, $\{\}$)
(return-card, $\{\}$)	(successful, $\{W - t_{10} \leq 10\}$)
	(return-card, $\{\}$)
m^{final}: card-not-inserted	m^{final}: card-not-inserted

Figure 17: Timed Event Sequences of the ATM with withdraw and deposit option

Automated Teller Machine (ATM)

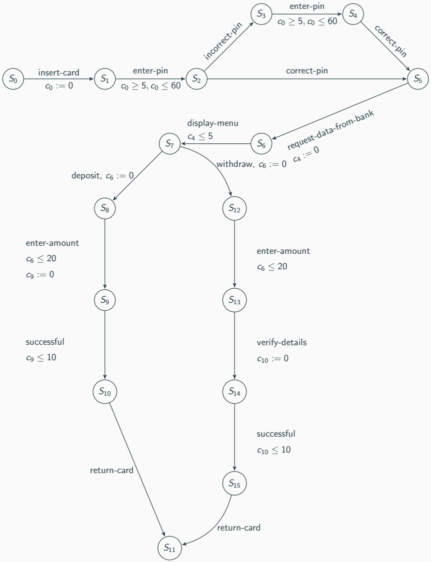


Figure 18: The synthesized timed automaton of the ATM

Light Control System

Consider the variant of the light control system [2]:

1. *Idle* is both the initial and final mode ,
2. The mode changes from *Idle* to *Light* upon issuing the command *ON*, and
3. If *ON* is issued again within 3 units of time, the light will go *Bright*. Else the light goes back to *Idle*.

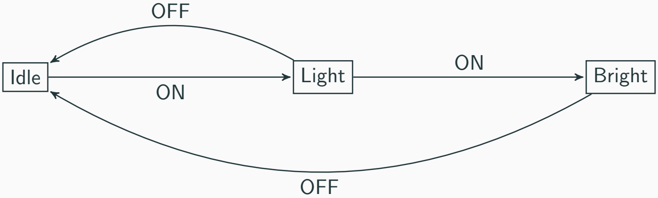


Figure 19: Mode graph of the Light Control System

Light Control System

Scenario 1	Scenario 2	Scenario 3
$m^{initial}$: *Idle*	$m^{initial}$: *Light*	$m^{initial}$: *Bright*
$(ON, \{\})$	$(ON, \{w - t0 <= 3\})$	$(OFF, \{\})$
$(OFF, \{w - t0 > 3\})$	$(OFF, \{\})$	$(ON, \{\})$
$(ON, \{\})$	$(ON, \{\})$	$(ON, \{w - t0 <= 3\})$
$(ON, \{w - t0 <= 3\})$	$(OFF, \{w - t0 > 3\})$	$(OFF, \{\})$
$(OFF, \{\})$	m^{final} : *Idle*	m^{final} : *Idle*
m^{final} : *Idle*		

Figure 20: Timed Event Sequences of the Light Control System

Light Control System

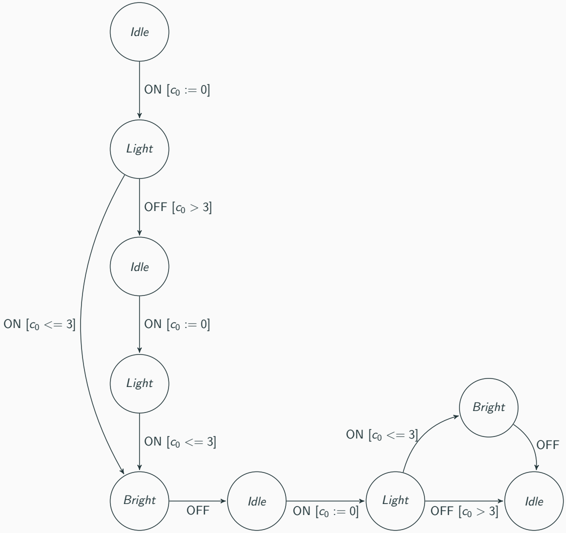


Figure 21: Timed automaton of the Light Control System

Traffic Light

Consider the behaviour of a traffic light [3].

- The system periodically moves from *Initial* $\rightarrow$ *Red*, *Red* $\rightarrow$ *Yellow*, *yellow* $\rightarrow$ *Green*, and then it is *Reset*,
- On the transitions *turn* $-$ *yellow* and *turn* $-$ *green* a clock constraint is checked and a new clock is born.

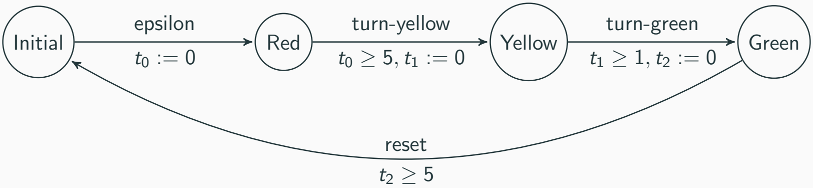


Figure 22: Timed automaton of the Traffic Light

Traffic Light

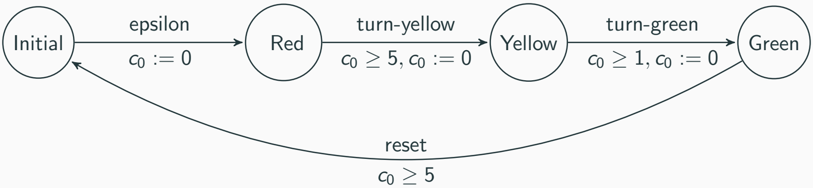


Figure 23: The optimally allocated timed automaton of the Traffic Light

CSMA/CD Protocol

Consider a variant of the CSMA/CD protocol [7].

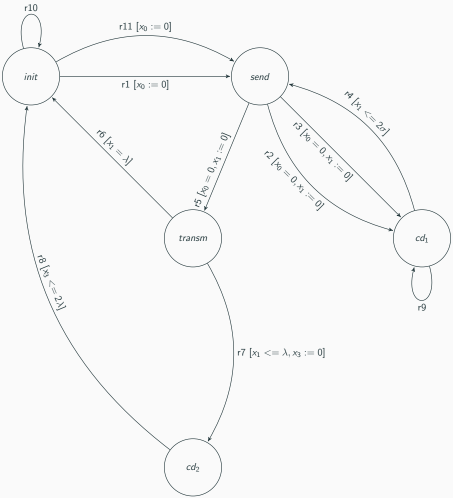


Figure 24: The timed automaton for the sender in CSMA/CD protocol

CSMA/CD Protocol

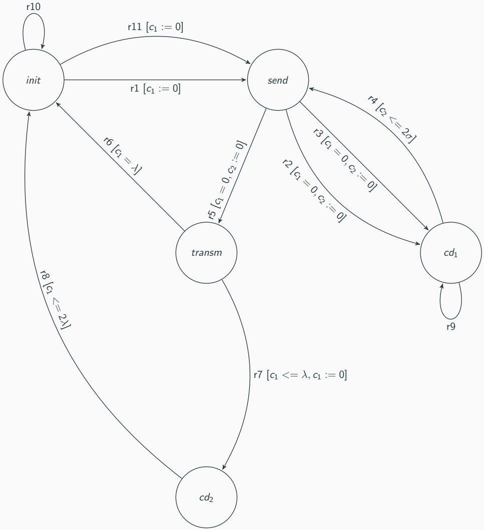


Figure 25: The optimally allocated timed automaton for the sender in CSMA/CD protocol

Conclusion

Conclusion

- Proposed Timed Event Sequences to formally represent scenarios,
- Developed and implemented an algorithm to construct timed automaton from a given set of scenarios expressed as TES and a mode graph,
- The synthesized timed automaton is minimal, deterministic and acyclic,
- The generated timed automaton belongs to a class of timed automata that satisfies the dominance assumption,
- Developed and implemented an optimal clock allocation algorithm to a class of timed automata that satisfies the dominance assumption, and
- Our algorithm, does not change the size of the original timed automaton and it's complexity is quadratic in the size of the graph.

THANK YOU!

Dr. Neda Saeedloei
Dr. Henry Wang
Dr. Ping Zhao
all the faculty members and students.

Questions?

R. Alur and D. L. Dill.
A theory of timed automata.
Theor. Comput. Sci., 126(2):183–235, Apr. 1994.

P. Bouyer.
An introduction to timed automata.
2011-2012.

A. Dubey, S. Nordstrom, T. Keskinpala, S. Neema, T. Bapty, and G. Karsai.
Towards a verifiable real-time, autonomic, fault mitigation framework for large scale real-time systems.
Innovations in Systems and Software Engineering, 3(1):33–52, 2007.

T. Lengauer and R. E. Tarjan.
A fast algorithm for finding dominators in a flowgraph.
volume 1, pages 121–141, New York, NY, USA, Jan. 1979. ACM.

N. Saeedloei.
From scenarios to timed automata.
Technical report, Department of Computer Science, University of Minnesota Duluth, Duluth, MN, jun 2016.

N. Saeedloei and F. Kluzniak.
Optimal clock allocation for a class of timed automata.
Technical report, Department of Computer Science, University of Minnesota Duluth, Duluth, MN, Sept 2016.

S. Yovine.
A case study: the csma/cd protocol.
Nov 1994.