From Scenarios to Optimally Allocated Timed Automata

Sandeep Vuppula

June 4, 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.

Background

Modeling Time

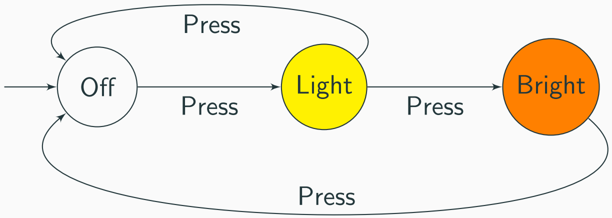
There are three approaches for modeling time:

- 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 a set of real numbers, and the sequence of times increases monotonically without any limit.

Timed Automata

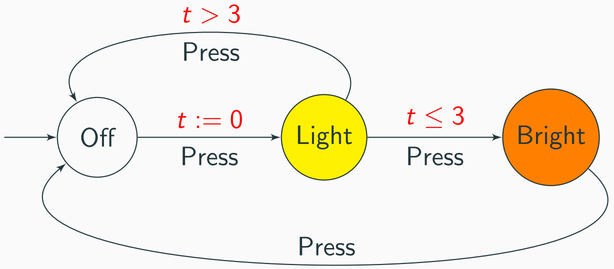
- A timed automaton is a finite state automaton extended with a finite set of real-valued clocks.
- Upon an input, the selection of next state is based not only on the input symbol but also on the time of the current symbol with respect to the formerly read symbols.

A Simple Light Control



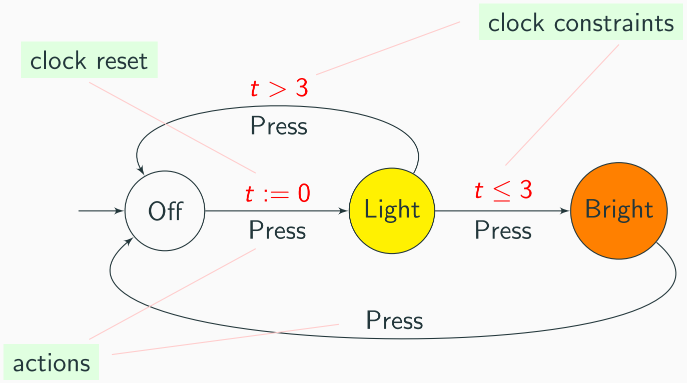
Desired behavior: if Press occurs twice quickly then the light gets brighter, otherwise the light turns off.

Timed Automata: A Simple Light Control



Solution: augment with real-valued clocks.

Timed Automata



Timed Automata: Syntax

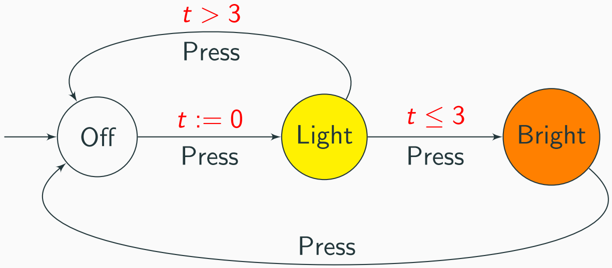
For a set V of clock variables:

- the set $\Phi(V)$ includes *clock constraints* of the form $t \sim a$, where
    - $t \in V$,
    - $\sim \in \{\leq, \geq, <, >, =\}$,
    - a is a constant in the set of rational numbers, $\mathbb{Q}$.

A *timed automaton* is a tuple $\mathcal{A} = \langle \Sigma, Q, Q_0, V, E \rangle$, where

- Σ is a finite set of labels/actions (alphabet)
- Q is a finite set of states
- $Q_0 \subseteq Q$ is a set of initial states
- V is a finite set of clocks
- $E \subseteq Q \times Q \times \Sigma \times 2^V \times 2^{\Phi(V)}$ is a finite set of edges of the form $(q, q', \sigma, \lambda, \phi)$, where
    - $\lambda \subseteq V$ is the clocks to be reset with this transition,
    - ϕ is a set of clock constraints over V.

Timed automata: Language Accepted



The languge accepted by the automaton:

⋮

$(press, 5)(press, 6.5)(press, 20)(press, 50)(press, 65)...$
$(press, 3)(press, 9)(press, 15)(press, 19)(press, 100)...$

⋮

Timed Automata: Undecidable Problem

- The number of clocks in a given timed automaton has a direct impact on verification of the system modeled as timed automaton.
- 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.

Motivation

Motivation

- 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.

Motivation

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

Contribution 1

- We use scenarios to specify the behaviour of a real-time system.
- We use mode graphs to specify the legal events that can occur in the system.

We synthesize a *minimal*, *acyclic* and *deterministic* timed automaton given a set of scenarios and a mode graph.

Contribution 2

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

- We perform liveness analysis of clocks in a timed automaton and use that information to minimize the number of clocks and optimally allocate them.
- Our proposed clock allocation method can be applied to any timed automaton that satisfies certain properties.
- Unlike the existing approaches, our algorithm does not change the graph of the original timed automaton and its complexity is quadratic in the size of the graph

Contributions

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 a set of scenarios, and
2. Transforming this graph to the final timed automaton.

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.
- A set of scenarios can capture the behaviour of a real-time system.
- We use *mode graphs* to specify the legal events that can occur in the system.
- We propose Timed Event Sequences (TES) to describe the scenarios *formally*.

Mode Graph: Definition

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
- $T : M \times \Sigma \rightarrow M$ is a transition function.

Mode Graph Example

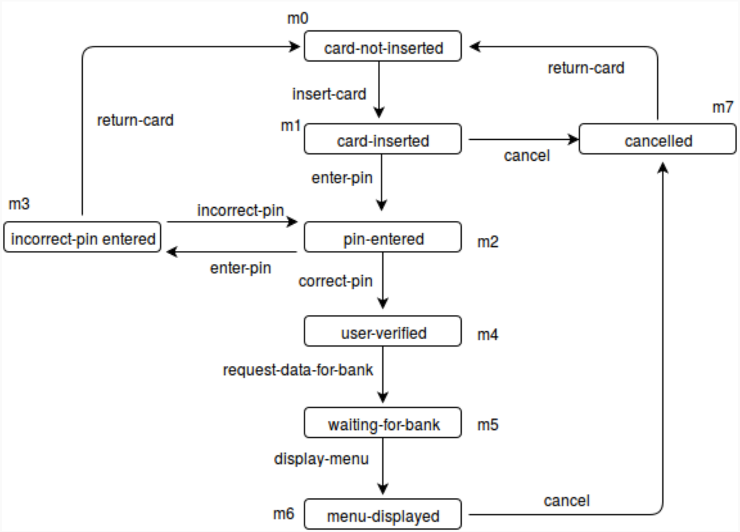


Figure 1: Mode Graph for ATM

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. We call this the *Dominance* relation and denote it as m_i DOM m_j.
- Dominance assumption ensures that time variables are well defined.

Timed Event Sequences (TES): Definition

A Timed Event Sequence ξ is a tuple $\left\langle m^{initial}, \Psi, m^{final} \right\rangle$ where,

- $m^{initial} \in M$ is the initial mode of the scenario,
- $\Psi = \{\psi_1, \psi_2, \psi_3, \dots\}$ is a non-empty sequence of timed events of the form (e_i, ϕ_i) where,
    - $e_i \in \Sigma$ is an event, and
    - $\phi_i \in 2^{\Phi(V)}$ is a set of time annotations associated with e_i.
- $m^{final} \in M$ is the final mode of the scenario.

Timed Event Sequences: Example

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.

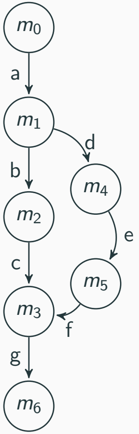
$m^{initial}$: card-not-inserted

(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

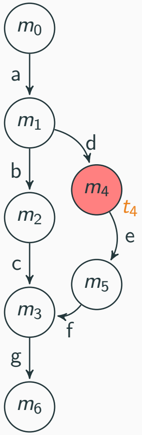
A Mode Graph and TES Satisfying the Dominance Assumption



$m^{initial}$: 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

- 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 time of leaving m_0 and m_1.

A Mode Graph and TES Not Satisfying the Dominance Assumption



$m^{initial}$: m_0
$(a, \{\})$
$(d, \{W - t_0 < 3\})$
$(e, \{W - t_1 > 1\})$
$(f, \{\})$
$(g, \{W - t_4 > 5, W - t_0 \leq 3\})$
m^{final} : m_6

- t_4 is used on the transition g.

Constructing a Time Annotated Graph from Scenarios

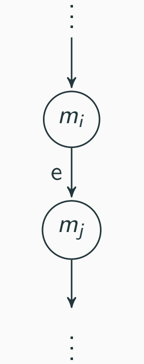
Given a mode graph $\mathcal{M}$ and a set of Timed Event Sequences $\Xi = \{\xi_1, \xi_2, .., \xi_k\}$ as inputs, we propose an algorithm for synthesizing a time annotated graph (TAG) G. Initially we start with an empty graph, G_0 and perform the following steps:

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}.

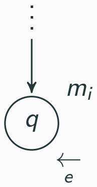
Decision on whether to create new states and transitions is resolved with the help of state labels (modes).

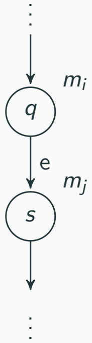
Decision on Creation of New States and Transitions

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 $(m_i, e, m_j) \in T$.



Mode Graph





Timed Annotated Graph

Properties of Constructed Time Annotated Graph

The graph constructed by our algorithm has the following properties:

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

Constructing a Timed Automaton from Time Annotated Graph

After constructing the time annotated graph, we need to transform it to the target timed automaton. For that, the following steps should be performed in order:

1. Determine the required number of clocks,
2. Generate clock resets and clock constraints, and
3. Add the clock resets and constraints to the appropriate transitions in the time annotated graph.

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 of Synthesis Method: TES of 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

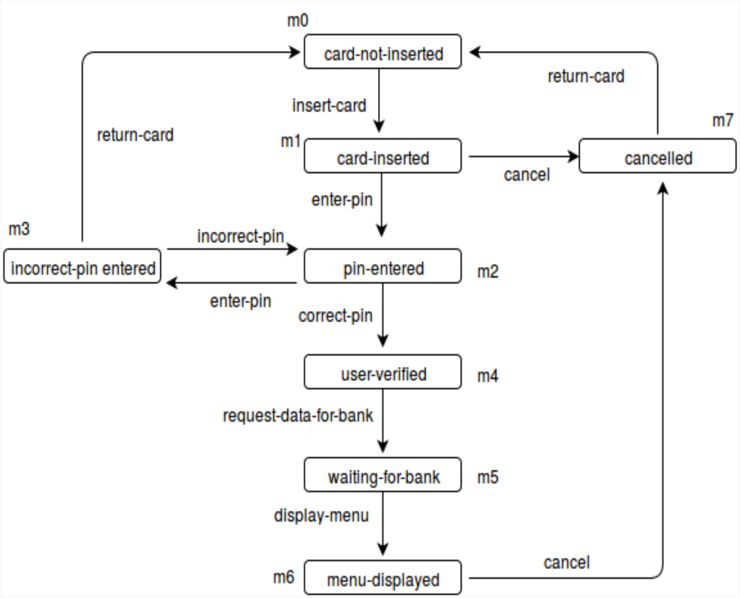
$m^{initial}$: card-not-inserted

(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

The Mode Graph for ATM



Synthesized Graph and Timed Automaton of the ATM

$$\begin{tikzcd}
\boxed{S_0[m_0]} \arrow[d, "\text{insert-card}"'] \\
\boxed{S_1[m_1]} \arrow[d, "\text{enter-pin } {[W - t_0 \geq 5, W - t_0 \leq 60]}"'] \\
\boxed{S_2[m_2]} \arrow[d, "\text{incorrect-pin}"'] \arrow[ddd, bend left, "\text{correct-pin}"] \\
\boxed{S_3[m_3]} \arrow[d, "\text{enter-pin } {[W - t_0 \geq 5, W - t_0 \leq 60]}"'] \\
\boxed{S_4[m_2]} \arrow[d, "\text{correct-pin}"'] \\
\boxed{S_5[m_4]} \arrow[d, "\text{request-data-from-bank}"'] \\
\boxed{S_6[m_5]} \arrow[d, "\text{display-menu } {[W - t_4 \leq 5]}"'] \\
\boxed{S_7[m_6]}
\end{tikzcd}$$

$$\begin{tikzcd}
S_0 \arrow[d, "\text{insert-card } {[c_0 := 0]}"'] \\
S_1 \arrow[d, "\text{enter-pin } {[c_0 \geq 5, c_0 \leq 60]}"'] \\
S_2 \arrow[d, "\text{incorrect-pin}"'] \arrow[ddd, bend left, "\text{correct-pin}"] \\
S_3 \arrow[d, "\text{enter-pin } {[c_0 \geq 5, c_0 \leq 60]}"'] \\
S_4 \arrow[d, "\text{correct-pin}"'] \\
S_5 \arrow[d, "\text{request-data-from-bank } {[c_4 := 0]}"'] \\
S_6 \arrow[d, "\text{display-menu } {[c_4 \leq 5]}"'] \\
S_7
\end{tikzcd}$$

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 previous transitions from states that dominate transition 'r'. We call this the *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, we need to perform the following steps in order:

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

Liveness Analysis of Clocks

Identifying the liveness ranges of all the clocks in a timed automaton helps us to 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 Analysis of Clocks

We also define the following set of functions:

- **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 Analysis Algorithm

Algorithm 1: Building the liveness ranges for clocks

```
Input   : A timed automaton $\mathcal{A} = \langle E, Q, \{q^0\}, Q_f, V, R, L\rangle$.
Output: An extended timed automaton
        $\mathcal{A}_e = \langle E, Q, \{q^0\}, Q_f, V, R_e, L\rangle$, where $R_e$ is the set of
        extended transitions.

$R_e := \emptyset$;
foreach transition $r = (s, q, e, \phi) \in R$ in $\mathcal{A}$ do
    $born(r) := active(r) := \emptyset$;
repeat
    foreach transition $r = (s, q, e, \lambda, \phi) \in R$ in $\mathcal{A}$ do
        foreach $r_o \in out(q)$ do
            $active(r) :=$
              $active(r) \cup ((active(r_o) \cup clock\_ref(r_o)) \setminus born(r_o))$;
        if $L(s) = j$ and $j \in active(r)$ then
            $born(r) := \{j\}$;
        $R_e := R_e \cup \{(r, born(r), active(r))\}$;
until there were no changes;
```

Liveness Analysis Example

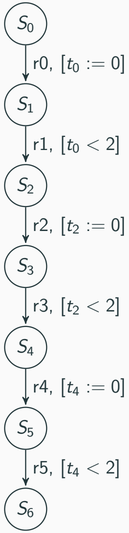

$$\begin{tikzcd}[row sep=large]
S_0 \arrow[d, "{r0,\ [t_0 := 0]}"] \\
S_1 \arrow[d, "{r1,\ [t_0 < 2]}"] \\
S_2 \arrow[d, "{r2,\ [t_2 := 0]}"] \\
S_3 \arrow[d, "{r3,\ [t_2 < 2]}"] \\
S_4 \arrow[d, "{r4,\ [t_4 := 0]}"] \\
S_5 \arrow[d, "{r5,\ [t_4 < 2]}"] \\
S_6
\end{tikzcd}$$

Table 1: *born* and *active* functions

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

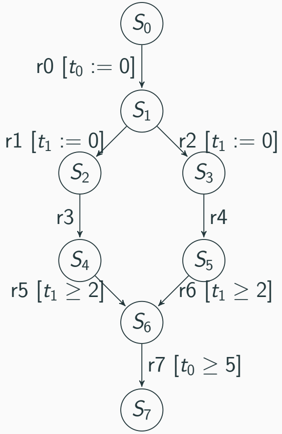
Our liveness analysis algorithm calculates the liveness ranges of clocks and generates extended transitions of the form $(r, \mathit{born}(r), \mathit{active}(r))$.

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.



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

Clock Allocation Algorithm

The algorithm annotates each state with:

- the set of available clocks, and
- the set of *clock assignments* of the form (c, j), where c is the clock that replaces the (old) clock t_j.

More precisely, we define the following functions:

- $\mathit{pool} : Q \to 2^{P_0}$ maps a state s to the set of clocks available at s;
- $\mathit{assignments} : Q \to 2^{P_0 \times N}$ maps a state s to the set of clock assignments at s.

Every time the algoritmm visits a transition where a clock, e.g., t_j is born, it associates a clock, say c, with j. Every time it visits a transition where the range of clock t_j ends, it restores c to the pool of available clocks.

Problematic States

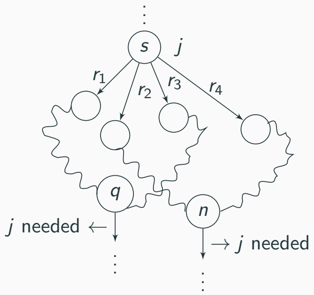


Figure 2: A timed automaton with problematic states

- Clock t_j is born on all outgoing transitions 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.

Clock Allocation Algorithm

```
**Procedure** compute-allocation(timed automaton $\mathcal{A}_e$, set of clocks $P_0$)
Input  : An extended timed automaton $\mathcal{A}_e = \langle E, Q, \{q^0\}, Q_f, V, R_e, L\rangle$
         and the initial pool of available clocks, $P_0$.
Output: An extended timed automaton $\mathcal{A}'_e = \langle E, Q_e, \{q^0\}, Q_f, V, R_e, L\rangle$,
         where $Q_e = Q \times 2^{P_0} \times 2^{P_0 \times N}$.
foreach state $s \in Q$ do
    Set the status of $s$ to Unseen;
annotate($q^0, P_0, \emptyset$);
visit($q^0$);
```

Clock Allocation Algorithm (Contd..)

```
Procedure annotate(state $q$, set of clocks $p$, set of assignments $\dashv$)
// Invoked only when status of $q$ is Unseen.
$pool(q) := p$;
$assignments(q) := \dashv$;
Set the status of $q$ to Seen;
```

```
Procedure visit(state $q$)
// Invoked only when the status of $q$ is Seen or Visited.
if status of $q$ is not Visited then
    Set the status of $q$ to Visited;
    annotate-immediate-successors-of($q$);
    foreach $r \in out(q)$ do
        $visit(target(r))$;
```

Clock Allocation Algorithm (Contd..)

```
**Procedure** annotate-immediate-successors-of(state $q$)

Partition $out(q)$ into $mothers$ and $others$;
foreach $r \in others$ do
    if $status\ of\ target(r)\ is\ Unseen$ then
        propagate($q, r, \emptyset$);
    // Otherwise $target(r)$ is already properly annotated
if $mothers \neq \emptyset$ then
    $Groups := partition\text{-}into\text{-}a\text{-}set\text{-}of\text{-}groups(q, mothers)$;
    foreach $group \in Groups$ do
        $c := find\text{-}clock(q, group)$;
        foreach $r \in group$ do
            // The target of $r$ is $Unseen$ (by the dominance assumption).
            $propagate(q, r, \{c\})$;
```

Clock Allocation Algorithm (Contd..)

```
Procedure propagate(state $q$, transition $r$, set of clocks $sc$)
// $q$ is the source of $r$. Propagate $pool(q)$ and $assignments(q)$ to
  $target(r)$, taking into account that some clock ranges
// may end on $r$. If $sc$ is not empty, it must be a singleton: in that case
  assign its member to clock number $L(q)$.
// Invoked only when the target of $r$ is $Unseen$.
$freed\_assignments := \{(d, j) \mid (d, j) \in assignments(q) \wedge j \notin active(r)\}$;
$freed\_clocks := \{d \mid (d, j) \in freed\_assignments\}$;
$tmp\_pool := pool(q) \cup freed\_clocks$;
$tmp\_assignments := assignments(q) \setminus freed\_assignments$;
if $sc \neq \emptyset$ then
    $tmp\_pool := tmp\_pool \setminus sc$;
    $tmp\_assignments := tmp\_assignments \cup \{(c, L(q))\}$, where $c \in sc$;
$annotate(target(r), tmp\_pool, tmp\_assignments)$;
```

Clock Allocation Algorithm (Contd..)

Procedure partition-into-a-set-of-groups(state q, set of transitions *mothers*)

```
$mother\_targets := \{target(r) \mid r \in mothers\}$;
// Initially, each mother is in its own group.
$Groups := \emptyset$;
foreach $r \in mothers$ do
    $Groups := Groups \cup \{r\}$;
$PP := \emptyset$;        // potentially problematic states
foreach $r \in mothers$ do
    foreach $s \in reachable(target(r))$ do
        if $L(q) \in active(r')$, where $r'$ is an arbitrary transition of $in(s)$
         then
            $PP := PP \cup \{s\}$;
// Those states in PP that can be reached from more than one mother
 are the problematic states.
foreach $s \in PP$ do
    $targets := reachable\_from(s) \cap mother\_targets$;
    Merge those members of Groups that contain transitions whose
     target is in targets;
return Groups;
```

Clock Allocation Algorithm (Contd..)

Procedure find-clock(state q, set of transitions *group*)

```
// Find a clock for $L(q)$ on transitions in $group$.
$live\_on\_entry := \{j \mid (c, j) \in assignments(q)\}$;
$dying\_all := \bigcap_{r \in group}(live\_on\_entry \setminus active(r))$;
// The set of clocks whose liveness ranges end in all the transitions in
  $group$:
$released\_all := \{c \mid (c, j) \in assignments(q) \wedge j \in dying\_all\}$;
$available := released\_all \cup pool(q)$;
Return the clock variable with the smallest number in $available$;
```

Example: Optimal Clock Allocation

$$\begin{tikzcd}
S_0 \arrow[d, "{r0,\ [t_0 := 0]}"] & S_0 \arrow[d, "{r0,\ [c_0 := 0]}"] \\
S_1 \arrow[d, "{r1,\ [t_0 < 2]}"] & S_1 \arrow[d, "{r1,\ [c_0 < 2]}"] \\
S_2 \arrow[d, "{r2,\ [t_2 := 0]}"] & S_2 \arrow[d, "{r2,\ [c_0 := 0]}"] \\
S_3 \arrow[d, "{r3,\ [t_2 < 2]}"] & S_3 \arrow[d, "{r3,\ [c_0 < 2]}"] \\
S_4 \arrow[d, "{r4,\ [t_4 := 0]}"] & S_4 \arrow[d, "{r4,\ [c_0 := 0]}"] \\
S_5 \arrow[d, "{r5,\ [t_4 < 2]}"] & S_5 \arrow[d, "{r5,\ [c_0 < 2]}"] \\
S_6 & S_6
\end{tikzcd}$$

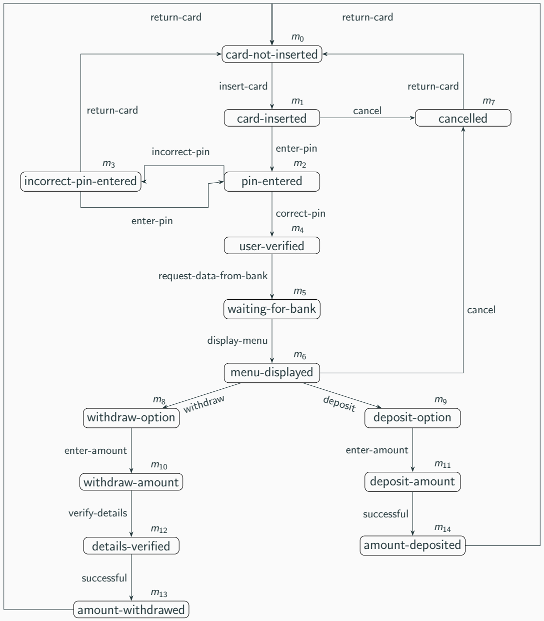
Case Studies

Synthesizing Timed Automaton for 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 of ATM



Timed Event Sequences of the 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

Synthesized Timed Automaton of the ATM

$$\begin{tikzcd}
 & & & S_3 \arrow[r, "\text{enter-pin}", "{c_0 \geq 5, c_0 \leq 60}"'] & S_4 \arrow[dr, "\text{correct-pin}"] & \\
S_0 \arrow[r, "\text{insert-card}", "c_0 := 0"'] & S_1 \arrow[r, "\text{enter-pin}", "{c_0 \geq 5, c_0 \leq 60}"'] & S_2 \arrow[ur, "\text{incorrect-pin}"] \arrow[rrr, "\text{correct-pin}"] & & & S_5 \arrow[dll, "\text{request-data-from-bank}", "c_4 := 0"'] \\
 & & S_7 & S_6 \arrow[l, "\text{display-menu}"', "c_4 \leq 5"] & &
\end{tikzcd}$$

Extended Behaviour of the 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.

TES of Extended Behaviour of the ATM

$m^{initial}$: menu-displayed

(deposit, $\{\}$)
(enter-amount, $\{W - t_6 \leq 20\}$)
(successful, $\{W - t_9 \leq 10\}$)
(return-card, $\{\}$)

m^{final}: card-not-inserted

TES of Scenario 3

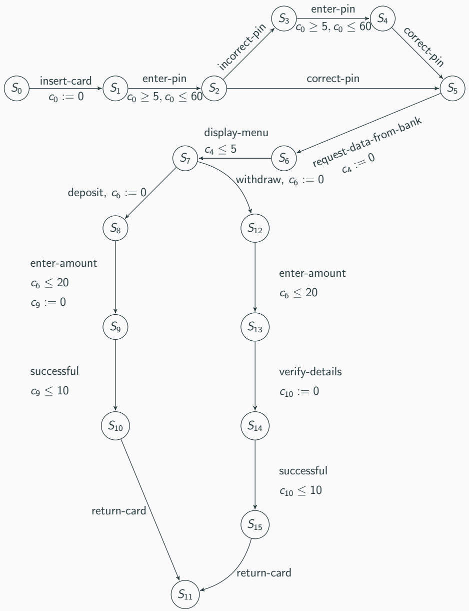
$m^{initial}$: menu-displayed

(withdraw, $\{\}$)
(enter-amount, $\{W - t_6 \leq 20\}$)
(verify-details, $\{\}$)
(successful, $\{W - t_{10} \leq 10\}$)
(return-card, $\{\}$)

m^{final}: card-not-inserted

TES of Scenario 4

Synthesized Timed Automaton of the ATM



Synthesizing Timed Automaton for Light Control System

Consider the variant of the light control system:

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*.

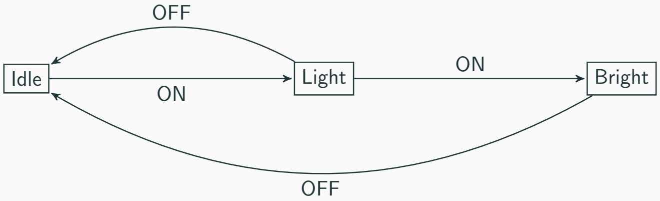


Figure 3: The mode graph of the Light Control System

TES of the Light Control System

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

Scenario 1

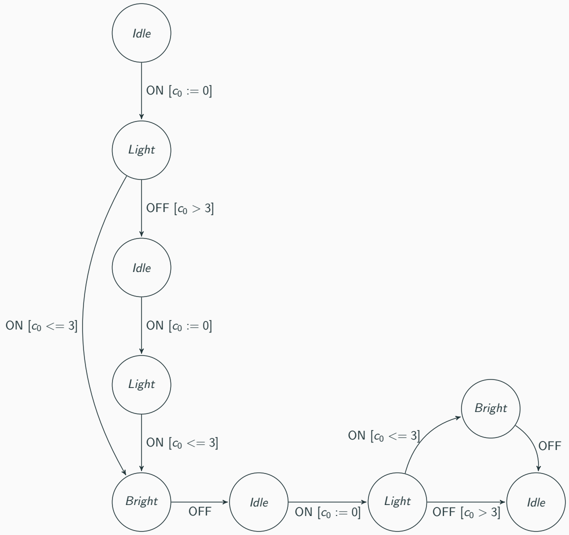
$m^{initial}$: *Light*
$(ON, \{w - t0 <= 3\})$
$(OFF, \{\})$
$(ON, \{\})$
$(OFF, \{w - t0 > 3\})$
m^{final} : *Idle*

Scenario 2

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

Scenario 3

Synthesized Timed Automaton of the Light Control System



Optimal Clock Allocation for Traffic Light

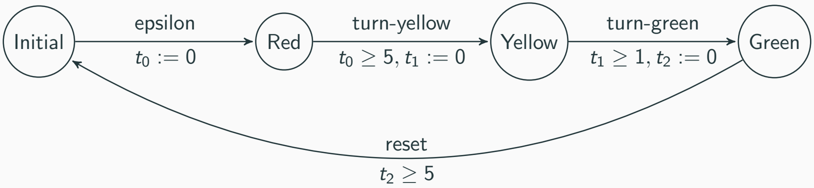


Figure 4: Timed automaton of the Traffic Light

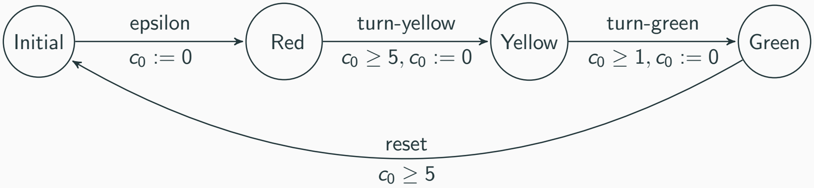
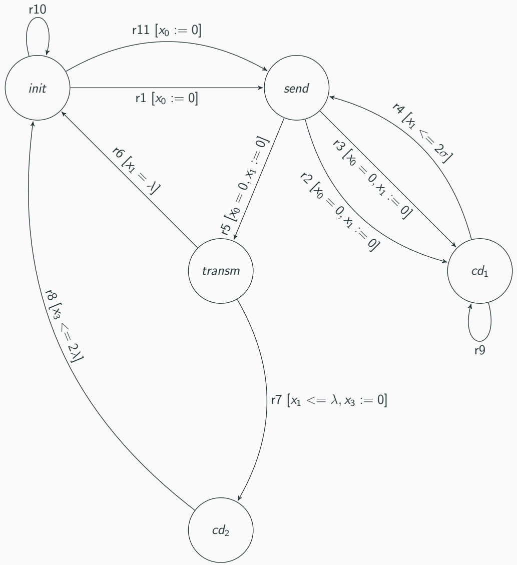


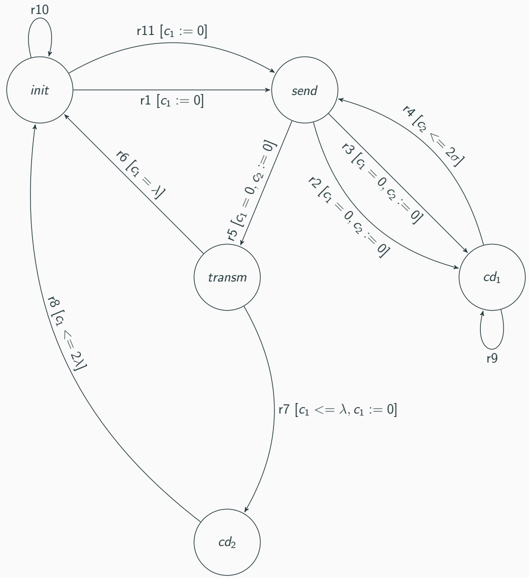
Figure 5: The optimally allocated timed automaton of the Traffic Light

Optimal Clock Allocation for Sender in CSMA/CD Protocol

Consider a variant of the CSMA/CD protocol.



Optimally Allocated Timed Automaton of 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 for a class of timed automata that satisfies the dominance assumption, and
- Our algorithm, does not change the graph of the original timed automaton and its complexity is quadratic in the size of the graph.

THANK YOU!

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

Questions?

References i

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.

References ii

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.