QEES lecture 5
Timed Automata

Marielle Stoelinga
Formal Methods & Tools

Welcome
Agenda

1. Solution to exercises
2. Timed Automata @ QEES
   - content of 4 lectures
3. Formal definitions
4. Work on Assignment:
   - Jobshop scheduling problem
1. Model a traffic light as a TA

- The light can show three colors: red, yellow and green
- It cycles through these colors in the usual order
- Initially, the light is yellow
- The red light is shown for exactly 4 minutes
- The green light is shown for exactly 1 minute
- The yellow light is shown for a duration between 0.1 and 0.2 minutes

```
x := 0
x = 4
x := 0
x := 0
x = 0
```
1. Model a traffic light as a TA

- The light can show three colors: red, yellow and green
- It cycles through these colors in the usual order
- ...

**Modeling principle #1:**

- Build your model in a step-wise fashion
- Solve one problem at the time
Exercise solutions

Consider two of these traffic light models, $T$ and $U$

- $T$ is initially green, $U$ initially red

Formulate correctness properties formulating that

- $T$ and $U$ can be red at the same time
- $T$ and $U$ can never be green at the same time
- Think one (or more) other useful properties of the traffic lights

- $E<> (T.Red \& U.Red)$
- Not $(E<> T.Green \& U.Green)$
- $A[] (T.green \text{ implies } U.yellow \text{ OR } U.red)$
- If a car arrives at the crossing the street when the light $T$ is green, then he will leave the crossing before light $U$ becomes green, provided that he drives at a reasonable state
What can we do with TA?

Verify properties on TA

- **Invariance: A[P]**
  - does a property hold for all states?

![Diagram](image)
What can we do with TA?

Parallel composition
- Composition TA
- Product of state space
- Synchronize on joint actions
- (bilateral)

**Gate**
- `open` (A_signal! x:=0)
- `lowering` (x ≤ 15)
- `raising` (x ≤ 16)
- `closed` (x ≥ 11)

**Train**
- `far` (A_signal! y:=0)
- `at_crossing` (y > 10)
- `near` (E_signal)

**Controller**
- `Approach? x:=0` (open, far,0)
- `Leave? z:=0` (open, near,2)
- `closed` (z<2)

1. **Approach?** x:=0
2. **Leave?** z:=0
3. **Approach!** z<2
4. **Exit!** z<2
Agenda

1. Solution to exercises

2. Timed Automata @ QEES
   - content of 4 lectures

3. Formal definitions

4. Work on Assignment:
   - Jobshop scheduling problem
## schedule 2013

<table>
<thead>
<tr>
<th>Date</th>
<th>at home (exercises / Video)</th>
<th>theory in class</th>
<th>exercises in class</th>
</tr>
</thead>
<tbody>
<tr>
<td>TA #1</td>
<td>Mon 25 Nov intro TA</td>
<td>intro TA</td>
<td>small resource scheduling</td>
</tr>
<tr>
<td></td>
<td>Fri 29 Nov Guest lecture @ TU Delft!</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TA #2</td>
<td>Mon 2 Dec Uppaal under the hood</td>
<td>Uppaal under the hood (algos)</td>
<td>exercises</td>
</tr>
<tr>
<td>TA #3</td>
<td>Fri 6 Dec Uppaal under the hood</td>
<td></td>
<td>WC: oefenen met RA</td>
</tr>
<tr>
<td>TA #4</td>
<td>Mon 9 Dec paper datapaths</td>
<td>- paper datapaths - assignment</td>
<td>oefenen RA, wrap up</td>
</tr>
</tbody>
</table>
- QEES lecture by Dr Arjan Mooij
- Embedded Systems Innovation by TNO
  - Former: Embedded Systems Institute
- Model-driven development @ Philips Med
Course objectives

You will learn

Practice

Modeling & Analysis
1. How to model real-time systems with TAs
2. How to formulate correctness properties in temporal logic
3. How to check them with model checkers (Uppaal)

Case studies
1. Learn about application of TA in state-of-the-art case studies in industrial practice.

Theory
1. Formal definitions of TA + semantics
2. How these model checking algorithms work
3. Data structures
**TOPICS**

**Modeling & Analysis**
1. TA model
2. TA property specification logic
3. The tool Uppaal

**Theory**
1. Parallel composition
2. Semantics
   1. Timed labeled transitions system
3. Model checking
   1. Recap: standard model checking algorithms
   1. Region automaton (RA)
   2. Zone automaton (ZA)
   3. TA model checking (based on ZA)

**Examples & Case studies**
1. Railroad crossing (lecture)
2. Traffic light (exercise)
3. Jobshop scheduling problem (exercise)
4. Fishers’ Mutex protocol (lecture)
5. Scheduling of datapath in OCE printer (paper)
6. Wireless HART protocol (assignment)
Timed Automata: Semantics, Algorithms and Tools, by Johan Bengtsson and Wang Yi

A Tutorial on Uppaal 4.0 (Updated November 28, 2006) Gerd Behrmann, Alexandre David, and Kim G. Larsen

NOTE: slides are an aid to read these papers
INTRODUCTION

- Mariëlle Stoelinga
  - MSc, PhD from RU Nijmegen
  - Postdoc at UC Santa Cruz
  - @UT since 2004
- Assoc Prof of ICT Risk Management
  - Fault tree analysis
  - Stochastic model checking
  - Model-driven testing
- TA modeling & analysis
  - IEEE WireWire
  - Parametric analysis
  - SDF2TA
1. Solution to exercises
2. Timed Automata @ QEES
   - content of 4 lectures
3. Formal definitions
   1. TA
   2. logic
4. Work on Assignment:
   - Jobshop scheduling problem
Timed Automata: mathematical definition

A TA is a 6-tuple \((L, l_0, \Sigma, C, E, \text{Inv})\) given by

- \(L\) is the set of locations (nodes)
- \(l_0 \in L\) is the initial location
- \(\Sigma\) is the action alphabet
  - \(\tau \in \Sigma\) notation for internal actions (absent in Uppaal)
- \(C\) is the set of clocks
- \(E \subseteq L \times B(C) \times \Sigma \times 2^C \times L\) is the edge set (transition relation)

Tiny mistake in paper:
\(\in\) should be \(\subseteq\)

Variants:
- \(\Sigma, C\) are global in paper. Here in tuple;
- \(I\) instead of \(\text{Inv}\)
- \(\tau\) vs no label
Timed Automata: mathematical definition

A TA is a 6-tuple \((L, l_0, \Sigma, C, E, \text{Inv})\) given by

- \(L\) is the set of locations (nodes)
- \(l_0 \in L\) is the initial location
- \(\Sigma\) is the action alphabet, \(\tau \in \Sigma\)
- \(E \subseteq L \times B(C) \times \Sigma \times 2^C \times L\) transition rel.
- \(\text{Inv}: L \rightarrow B(C)\) is the invariant function

Exercise:
write down this TA as a tuple

- \(L = \{\text{off}, \text{on}, \text{bright}\}\)
- \(l_0 = \text{off}\)
- \(\Sigma = \{\text{click}\}\)
- \(E = \{(\text{off}, \text{true}, \text{click}, \{x\}, \text{on}),\)
  \((\text{on}, x \leq 2, \text{click}, \emptyset, \text{bright}),\)
  \((\text{on}, x \geq 2, \tau, \emptyset, \text{off}),\)
  \((\text{bright}, \text{true}, \text{click}, \emptyset, \text{off})\}\)
- \(\text{Inv}(\text{off}) = \text{true},\)
- \(\text{Inv}(\text{on}) = x \leq 300\)
- \(\text{Inv}(\text{bright}) = \text{true}\)
Official syntax for clock constraints

Syntax:

\[ F ::= x - y \leq c \mid x - y < c \mid x \leq c \mid x < c \mid F \& F \]

x, y clocks, c integer

Encoding more constraints (syntactic sugar)

- \( x - y > c \) \( \rightarrow \) \( y - x < c \) (etc)
- True \( \rightarrow \) \( x \geq 0 \)
- \( x - y < p/q \) \( \rightarrow \) multiply all constants in TA by q
- \( x - y < c \) OR \( x' - y' < c \) \( \rightarrow \) use two transitions,
  one with \( x - y < c \), one with \( x' - y' < c' \)

Theory vs practice

Practice / modeling

- Syntax as rich as possible
  - AND, OR, …<, > ,…
- Concise /convenient modeling

Theory

- Minimal syntax
- Prove properties for all TA / for all constraints

Solution

- Minimal core syntax
- Maximal syntactic sugar
The logic TCTL

Error in screen casts!
reachability
• wrong: ◊p
• correct: E◊p
◊p is correct in LTL, not CTL

Logic in Uppaal: restricted TCTL

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E◊p</td>
<td>there exists a path where p eventually holds (reachability)</td>
</tr>
<tr>
<td>A□p</td>
<td>for all paths, p always holds (invariance, safety)</td>
</tr>
<tr>
<td>E□p</td>
<td>there exists a path where p always holds</td>
</tr>
<tr>
<td>A◊p</td>
<td>for all paths, p will eventually hold</td>
</tr>
<tr>
<td>p → q</td>
<td>whenever p holds, q will eventually hold (leadsto/response)</td>
</tr>
</tbody>
</table>

Correctness properties:

Safety / invariance
• A[ ] (P1.cs implies not: P2.cs)

Liveness/response
• P1.req → P1.cs

Reachability
• E<> P1.cs

No deadlocks
• A[ ] no deadlock

Invariance
• wrong: □p
• correct: A□p
□p is correct in LTL, not CTL

property p:
• location
• clock constraint
• Boolean combi
1. Solution to exercises
2. Timed Automata @ QEES
   - content of 4 lectures
3. Formal definitions
4. Work on Assignment:
   - Jobshop scheduling problem
Exercise: simple jobshop scheduling problem

- Classical example
- Simple simple resource scheduling
- Basis of more complex scheduling problems,
- WCET (worse case execution time) analysis of data paths in printers
- See next lecture

Example by Frits Vaandrager
Context

- Two people are two tools: hammer and mallet
- Purpose: make objects from simple components.
- Each object is made by driving a peg into a block. We call a pair consisting of a peg and a block a job;
- Jobs arrive sequentially on conveyor belt; completed objects depart on a conveyor belt.
- (The jobshop could involve any number of people, call jobbers, sharing more or fewer tools. Here, we assume a system with two jobbers and a hammer and a mallet.)

Schedulability analysis

Given arrival pattern for jobs
- Find feasible schedule
  - i.e. assignment of jobs to workers (jobbers)
  - Such that all jobs are completed
- Does it exist?
- Are all deadlines made?
- What is the fastest schedule?
Problem description

- Jobs come in different types: *easy*, *hard* and *average*.
  - Easy jobs can be done by hand,
  - Hard jobs with the hammer,
  - Average jobs with either hammer or mallet.

- Jobber as timed automaton
  - No clocks, for now
  - Use labels `jobE?`, `jobA?`, `jobH?` to model the arrival of easy, average and hard jobs respectively.
  - Use labels `get_hammer!`, `put_hammer!` to take / put back hammer
  - Similarly, use labels `get_mallet!`, `put_mallet!` to take / put back mallet

Exercise

1. Model the behavior of a jobber
2. Model the behavior of a hammer
The jobber

- Easy jobs by hand; Hard jobs with hammer, Average jobs: hammer or mallet.
- Use `get_hammer!`, `put_hammer!` to take / put back hammer
- Similarly, use labels `get_mallet!`, `put_mallet!`
Sanity checks (important!)
- A[] not deadlock
- E<> Jobber1.hard_done (etc)
- A[] Mallet.taken == jobber1.work_mallet || jobber2.work_mallet
- A[] Hammer.taken == ……??
A jobber needs (at least)
- 5 seconds for an easy job,
- 10 seconds for an average job using the hammer,
- 15 seconds for an average job using the mallet, and 20 seconds
  for a hard job.

Jobs arrive in the following order: H, A, H, H, H, E, E, A, A,
- where E denotes an easy job,
- A an average job, and
- H a hard job.

How much time do two jobbers need (at least) to complete 10 jobs?
Model the timing requirements in the Jobber
Model the arrival of jobs
Analyze schedulability
- A jobber needs (at least)
  - 5 seconds for an easy job,
  - 10 seconds for an average job using the hammer,
  - 15 seconds for an average job using the mallet, and 20 seconds for a hard job.
- Jobs arrive in the following order: H, A, H, H, H, E, E, A, A, A
  - where E denotes an easy job,
  - A an average job, and
  - H a hard job.

**Exercise**
1. Model timing behavior of a jobber
2. Find a feasible schedule
3. Find the fastest schedule
The tool Uppaal

- powerful model checker for TAs
- used in numerous industrial/academic applications
- developed in Uppsala and Aalborg
- TA, extended with
  - variables
  - committed/urgent locations
  - C-code
  - ...
- Can also be used for
  - test case generation
  - optimization
  - code synthesis
  - ...
<table>
<thead>
<tr>
<th>Date</th>
<th>at home (exercises / Video)</th>
<th>theory in class</th>
<th>exercises in class</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TA #1</strong></td>
<td><strong>Mon 25 Nov</strong></td>
<td>intro TA</td>
<td>intro TA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>small resource scheduling</td>
</tr>
<tr>
<td><strong>Fri 29 Nov</strong></td>
<td><strong>Guest lecture @ TU Delft !</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>TA #2</strong></td>
<td><strong>Mon 2 Dec</strong></td>
<td>Uppaal under the hood</td>
<td>exercises</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Uppaal under the hood (algos)</td>
<td></td>
</tr>
<tr>
<td><strong>TA #3</strong></td>
<td><strong>Fri 6 Dec</strong></td>
<td>Uppaal under the hood</td>
<td><strong>WC:</strong> oefenen met RA</td>
</tr>
<tr>
<td><strong>TA #4</strong></td>
<td><strong>Mon 9 Dec</strong></td>
<td>paper datapaths</td>
<td>- paper datapaths - assignment</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>oefenen RA, wrap up</td>
</tr>
</tbody>
</table>
What are timed automata?

.... And why do we need them?

Mariëlle Stoelinga
Formal Methods & Tools
2. Model the same traffic light except that
   - It is green initially
   - It switches to yellow only if a pedestrian arrives

What happens if a pedestrian arrives in Red or Green?
   - Incomplete specification!
Exercise solutions

Modeling principle #2:
- Make your models input-complete

What happens if a pedestrian arrives in Red or Green?
- Incomplete specification!
- Very common in practice

What happens if a pedestrian arrives in Red or Green?
- Incomplete specification!
- Very common in practice