Introduction to Lab 4
Modelling and Verification using UPPAAL

Jakaria Abdullah <jakaria.abdullah@it.uu.se>

13 October 2017
Lab 4: Modelling and Verification using UPPAAL

- **Lab goals:**
  - Practice formal modelling and verification of RTS
  - Work with timed automata and UPPAAL

- **Lab preparation:**
  - Work in your groups
  - Lab will be done on Tue, 17.10. in room 1515
  - Have a look at the lab homepage
    [http://www.it.uu.se/edu/course/homepage/realtid/ht17/lab4](http://www.it.uu.se/edu/course/homepage/realtid/ht17/lab4)
  - (“Small Tutorial” is recommended reading!)

- **Lab report:**
  - Answers (models, queries, values) to the questions
  - Via Student portal
  - **Deadline:** Sun, 22.10., 11:59
Part 1: Warm-Up
  ▶ Model 3 simple automata
  ▶ Use verification for simple properties

Part 2: Scheduling
  ▶ Setting: Schedule jobs to CPUs
  ▶ One automaton per job and per CPU
  ▶ Determine minimal execution time

Part 3: Deadlock detection
  ▶ Model Buffer, Producer and Consumer from Ada lab
  ▶ Use verifier to find deadlocks
    ★ “Deadlock” means: Only time may pass (for all future)
  ▶ Use simulator to analyze them
  ▶ Remove all deadlocks
Finite Automata

- Theoretic model for systems (or whatever else)
- Locations and transitions (drawn as nodes and edges)

State space: Set of locations

Trace semantics:
- One possible trace: \( p \rightarrow q \rightarrow p \rightarrow r \rightarrow q \rightarrow \ldots \)
- Another one: \( p \rightarrow r \rightarrow q \rightarrow p \rightarrow r \rightarrow \ldots \)
- *Not* a trace: \( p \rightarrow r \rightarrow p \rightarrow \ldots \)
Networks of Finite Automata

- Compose several automata into *networks*
- Use *synchronization* on edges/transitions

![Diagram]

- State space: Product of location sets
- Trace semantics:
  - Interleaving, i.e., one automaton at a time
  - Except: Synchronized edges are taken together
  - E.g.: 
    \[(p, s) \rightarrow (q, s) \rightarrow (p, s) \overset{a}{\rightarrow} (r, t) \rightarrow (r, s) \rightarrow \ldots\]
  - Not a trace: 
    \[(p, s) \rightarrow (r, s) \rightarrow \ldots\]
Networks of Finite Automata

- Compose several automata into *networks*
- Use *synchronization* on edges/.transitions

![Diagram of automata network](image)

- State space: Product of location sets
- Trace semantics:
  - Interleaving, i.e., one automaton at a time
  - Except: Synchronized edges are taken *together*
  - E.g.: \((p, s) \rightarrow (q, s) \rightarrow (p, s) \overset{a}{\rightarrow} (r, t) \rightarrow (r, s) \rightarrow \ldots\)
  - *Not* a trace: \((p, s) \rightarrow (r, s) \rightarrow \ldots\)
Finite Automata: Model Checking

- Does a model *satisfy* some property $\varphi$?

![Automaton A and Automaton B]

**Property:** “Does $A.r$ imply $B.t$?”

- $\varphi := A[] (A.r \implies B.t)$
- Means: “In each state of each trace, $B$ is in $t$ whenever $A$ in $r$”

- Is *satisfied* in above example
- (Not satisfied without $b$ synchronization!)
Temporal Logic (CTL, Computation Tree Logic)

- Temporal operators
  - $\text{A}[] \ p$: $p$ is an invariant
    - In all executions, $p$ always holds in all reachable states
  - $\text{E}[] \ p$: $p$ may hold globally
    - There is an execution in which $p$ always holds in all states
  - $\text{E}<> \ p$: $p$ is reachable/possible
    - There is an execution in which $p$ eventually holds in a state
  - $\text{A}<> \ p$: $p$ is guaranteed
    - In all executions, $p$ eventually holds in some states

- (UPPAAL cannot nest them)

Operator = Path quantifier + State operator

- $\text{A}, \text{E}$: Path quantifiers (Always, Eventually)
- $[], <>$: State operators (often written $\text{G}$, $\text{F}$: Globally, Finally)
Temporal Logic (CTL, Computation Tree Logic)

- Temporal operators
  - A[] p: p is an invariant
    - In all executions, p always holds in all reachable states
  - E[] p: p may hold globally
    - There is an execution in which p always holds in all states
  - E<> p: p is reachable/possible
    - There is an execution in which p eventually holds in a state
  - A<> p: p is guaranteed
    - In all executions, p eventually holds in some states

- (UPPAAL cannot nest them)

Operator = Path quantifier + State operator

- A, E: Path quantifiers (Always, Eventually)
- [], <>: State operators (often written G, F: Globally, Finally)
Timed Automata

- Extend finite automata with *clocks*:

![Timed Automata Diagram]

- Clocks have *real* values
  - All increasing at same pace
  - Can be reset and compared

- State space: Location $\times$ Clock valuations

- Trace semantics: Additional *delay* transitions
  
  
  $$
  (off, 0) \xrightarrow{\delta} (off, 1.2) \rightarrow (low, 0.0) \xrightarrow{\delta} (low, 5.7) \rightarrow (off, 5.7) \rightarrow (low, 0.0) \xrightarrow{\delta} (low, 2.3) \rightarrow (high, 2.3) \rightarrow \ldots
  $$

Jakaria

Lab 4: UPPAAL

13 October 2017 9 / 13
Networks of Timed Automata

- Compose just like before, using synchronized edges

**Lamp automaton**

**In UPPAAL:**

- Sync. channels need to be *declared*
- (As well as clocks and variables)
Model Checker for timed automata

- Developed at Aalborg University, Denmark and Uppsala University
- Started 1995, rather mature by now
- Different branches: Timed games, costs, statistical model checker, ...
- GUI in Java, verification engine C++
- Extensive online help. Use it!

Three panes:
1. Automata editor
2. Simulator
3. Verifier

Free for private/academic use (but closed-source)

You can run it at home: http://www.uppaal.org
Demo
The End

Questions?