Information Systems Analysis

Temporal Logic and Timed Automata

(5)

UPPAAL timed automata

© Paweł Głuchowski, Wrocław University of Technology version 2.2
Contents of the lecture

Tools for automatic verification of a system

- Which tool to choose?
Contents of the lecture

UPPAAL

• What does UPPAAL serve for?
• UPPAAL editor, simulator and verifier
Contents of the lecture

Automata in UPPAAL

- States of the automaton
- Description of the automaton's transition
  - Declarations
  - Numeric variables and constants
  - Clock variables
- Channels and synchronisation
Contents of the lecture

Verification in UPPAAL

- Syntax of the language of formulas
- Verification of reachability, liveness and safety
  - What is possible?
  - What is not possible?
Tools for automatic verification of a system

• Which tool to choose?
### Tools for automatic verification of a system

#### Which tool to choose?

<table>
<thead>
<tr>
<th>tool</th>
<th>description of the system's model</th>
<th>temporal logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kronos</td>
<td>language ET-LOTOS</td>
<td>CTL (TCTL)</td>
</tr>
<tr>
<td>NuSMV</td>
<td>language SMV</td>
<td>LTL, CTL, RTCTL</td>
</tr>
<tr>
<td>Spin</td>
<td>language PROMELA</td>
<td>LTL</td>
</tr>
<tr>
<td>UPPAAL</td>
<td><strong>graphic</strong></td>
<td>CTL (TCTL)</td>
</tr>
<tr>
<td>Verus</td>
<td>language Verus</td>
<td>LTL, CTL, RTCTL, PRTCTL</td>
</tr>
</tbody>
</table>
What does UPPAAL serve for?

- UPPAAL editor, simulator and verifier
What does UPPAAL serve for?

Goal:

- modelling and analysis of real-time systems, including concurrent programs.

Possibilities:

- graphic modelling a system as finite state automata,
- using timed automata (automata with clocks),
- graphic simulating possible runs of the automata,
- specifying some properties of the system as CTL formulas (temporal operators F and G only, without nesting thereof),
- verifying some properties of the model.
UPPAAL

UPPAAL editor, simulator and verifier

Work order in UPPAAL:

1) Editor
2) Simulator
3) Verifier
UPPAAL editor, simulator and verifier

Step 1. **Modelling**
- build a model of a system as an automaton or automata.

Step 2. **Simulating**
- check, step by step, whether the model behaves correctly.

Step 3. Write properties of the system as logic CTL formulas.

Step 4. **Verifying**
- automatically verify truth of these formulas.
Automata in UPPAAL

- States of the automaton
- Description of the automaton's transition
  - Declarations
  - Numeric variables and constants
  - Clock variables
- Channels and synchronisation
Automata in UPPAAL

States of the automaton:

- normal,
- initial,
- urgent:
  - time of being in it equals zero (it is left immediately),
- committed:
  - time of being in it equals zero (it is left immediately),
  - leaving it has a higher priority than leaving the urgent state.

If more than one committed state is active, the order of leaving them is random.
Automata in UPPAAL

Description of the automaton's transition:

- **select** – a selection of a variable's value from a given range,
- **guard** – a condition to take the transition,
- **sync** – a synchronisation through a channel,
- **update** – a change of values of variables and an execution of functions.

![Diagram](image-url)
Automata in UPPAAL

Declarations

- Instances of automata are declared in the *System declarations*.
- Global variables are declared in the “upper” *Declarations*.
- Local variables (for one automaton) are declared in the *Declarations* “bellow” this automaton's *template*. 
Numeric variables and constants

Declarations of variables:

- int name;  //an int variable (range from -32768 to 32768)
- int [0,9] name;  //an int variable (range from 0 to 9)
- int name[3] = {1,2,3};  //a table of 3 int variables and their values
- bool name;  //a logic variable

Declaration of a constant:

- const int name = 3;  //an int constant and its value

Declaration of a type:

- typedef int [0,9] name;  //a definition of a type int[0,9]
Automata in UPPAAL

Clock variables

Declaration of a clock variable:

• clock name;

A clock variable, as a state's *invariant*, makes the state to be left.

![Diagram](image)
Channels and synchronisation

Binary channel

- Synchronisation between two automata.
- Lack of a receiver blocks the sender.
- For many available receivers 1 of them is chosen randomly.

Declaration of a channel:

- chan name;
Channels and synchronisation

Binary urgent channel

- Synchronisation between two automata.
- Lack of a receiver blocks the sender.
- For many available receivers 1 of them is chosen randomly.
- Instant synchronisation (waiting time equals 0).
- Any guard with clock variables on a transition with the channel is forbidden.

Declaration of a channel:

- urgent chan name;
Channels and synchronisation

Broadcast channel

- Synchronisation between one automaton and one or many at once.
- Lack of a receiver does not block the sender.
- Synchronisation applies to available receivers only.

Declaration of a channel:

- broadcast chan name;
Verification in UPPAAL

- Syntax of the language of formulas
- Verification of reachability, liveness and safety
  - What is possible?
  - What is not possible?
Verification in UPPAAL

Syntax of the language of formulas

**formula** ::= 'A[]' expression | 'E<>' expression | 'E[]' expression | A<> expression --> expression

**expression** ::= ID | NAT | expression '[' expression ']' | '(' expression ')' | expression '++' | '++' expression | expression '--' | '--' expression | expression assign expression | unary expression | expression binary expression | expression '?' expression ':' expression | expression '.' ID | expression '(' arguments ')' | 'forall' '(' ID ':' type ')' expression | 'exists' '(' ID ':' type ')' expression | 'deadlock' | 'true' | 'false'

**arguments** ::= [ expression ( ',' expression )* ]

**assign** ::= '=' | ':=' | '+=' | '-=' | '*=' | '/=' | '%=' | '|=' | '&=' | '^=' | '<<=' | '>>='

**unary** ::= '+' | '-' | '!' | 'not'

**binary** ::= '<' | '<=' | '==' | '!=' | '>=' | '>' | '+' | '-' | '*' | '/' | '%' | '&' | '|' | '^' | '<<' | '>>' | '&&' | '||' | '?=' | 'or' | 'and' | 'imply'

**type** – predefined or created type of data
Verification in UPPAAL

Verification of reachability, liveness and safety

- **Reachability:**
  
  \[ E<> D.s \] – the state \( s \) of the automaton \( D \) may be reached,
  
  \[ A<> D.s \] – the state \( s \) of the automaton \( D \) will be reached.

- **Liveness:**
  
  \[ D.s \rightarrow D.z==3 \]

  – if the state \( s \) of the automaton \( D \) is reached, it will result in reaching its
  local variable \( z == 3 \),

  in CTL: \[ AG(D.s \Rightarrow AF D.z==3) \].

- **Safety:**
  
  \[ E[] D.s \] – the automaton \( D \) may be still in the state \( s \),
  
  \[ A[] D.s \] – the automaton \( D \) is still in the state \( s \).
Verification in UPPAAL

What is possible?

- to use temporal operators F (as “<>”) and G (as “[ ]”),
- to check, whether a given state is/will be active and whether a given variable has/will have a declared value, e.g.:
  - $A[] \text{aut.s imply aut.z} \geq x$
    - certainly always $\text{aut.s}$ implies $\text{aut.z} \geq x$,
  - $E<> \text{aut.s and aut.z} \geq x$
    - possibly finally $\text{aut.s}$ and $\text{aut.z} \geq x$ at once,
  - $A[] \text{aut.s1 + aut.s2 + aut.s3} \leq 1$
    - certainly always at most one of the states $\text{aut.s1}$, $\text{aut.s1}$ and $\text{aut.s3}$ is active.
Verification in UPPAAL

What is possible?

- to check, whether the system of automata is blocked (the *deadlock* expression), i.e. it is not possible to change any state, e.g.:
  - $E<>\text{deadlock}$ – the deadlock may finally be possible,
  - $A[] \neg \text{deadlock}$ – the deadlock is never possible,

- to use quantifiers, e.g. for automata:
  - “for all”, e.g.: $E<> \forall (i:\text{range}) \text{aut}(i).s$
  - “exists”, e.g.: $E<> \exists (i:\text{range}) \text{aut}(i).s$
What is not possible?

- to use other temporal operators than G and F,
- to nest temporal operators,
- to use more than one temporal operator in one formula,
- to use the operator $\rightarrow$ together with a temporal operator,
- to use the operator $\rightarrow$ together with the *deadlock* expression.
The end

Literature: