Temporal logic in system analysis
(3)
Applications of LTL logic

© Paweł Głuchowski
Politechnika Wrocławska

All rights reserved.
Reproduction or distribution in whole or in part is prohibited.
## Contents of the lecture

- Introduction to temporal logic
- LTL logic

### Prior lectures

<table>
<thead>
<tr>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applications of LTL logic in system analysis</td>
</tr>
<tr>
<td>Finite state automata</td>
</tr>
<tr>
<td>Properties of concurrent programs</td>
</tr>
<tr>
<td>System verification with LTL and automata</td>
</tr>
<tr>
<td>Automatic verification in UPPAAL</td>
</tr>
<tr>
<td>Modelling and verifying concurrency</td>
</tr>
<tr>
<td>Next lectures</td>
</tr>
</tbody>
</table>
Contents of the lecture

- What is LTL logic applied to in the system analysis

### Prior lectures

<table>
<thead>
<tr>
<th>Applications of LTL logic in system analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finite state automata</td>
</tr>
<tr>
<td>Properties of concurrent programs</td>
</tr>
<tr>
<td>System verification with LTL and automata</td>
</tr>
<tr>
<td>Automatic Verification in UPPAAL</td>
</tr>
<tr>
<td>Modelling and verifying concurrency</td>
</tr>
<tr>
<td>Next lectures</td>
</tr>
</tbody>
</table>
Contents of the lecture

- What is a finite state automaton
- Example: an automaton closing a door
- Example: an automaton setting the light on and off

Prior lectures
Applications of LTL logic in system analysis
Finite state automata
Properties of concurrent programs
System verification with LTL and automata
Automatic Verification in UPPAAL
Modelling and verifying concurrency
Next lectures
## Contents of the lecture

<table>
<thead>
<tr>
<th>Prior lectures</th>
<th>Properties of concurrent programs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applications of LTL logic in system analysis</td>
<td>• What is a concurrent program</td>
</tr>
<tr>
<td>Finite state automata</td>
<td>• Properties of concurrent programs</td>
</tr>
<tr>
<td>Properties of concurrent programs</td>
<td>• Examples of invariant properties</td>
</tr>
<tr>
<td>System verification with LTL and automata</td>
<td></td>
</tr>
<tr>
<td>Automatic Verification in UPPAAL</td>
<td></td>
</tr>
<tr>
<td>Modelling and verifying concurrency</td>
<td></td>
</tr>
<tr>
<td>Next lectures</td>
<td></td>
</tr>
</tbody>
</table>

2013/2014
Contents of the lecture

- System model verification
- Most important properties
- A system model verification algorithm
- A simple example

<table>
<thead>
<tr>
<th>Prior lectures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applications of LTL logic in system analysis</td>
</tr>
<tr>
<td>Finite state automata</td>
</tr>
<tr>
<td>Properties of concurrent programs</td>
</tr>
<tr>
<td>System verification with LTL and automata</td>
</tr>
<tr>
<td>Automatic Verification in UPPAAL</td>
</tr>
<tr>
<td>Modelling and verifying concurrency</td>
</tr>
<tr>
<td>Next lectures</td>
</tr>
</tbody>
</table>

2013/2014 Applications of LTL logic 6
## Contents of the lecture

### Prior lectures
- Applications of LTL logic in system analysis
- Finite state automata
- Properties of concurrent programs
- System verification with LTL and automata

### Automatic Verification in UPPAAL
- Tools for automatic verification
- What does UPPAAL allow to
- How to use the UPPAL modeller, simulator and verifier
Contents of the lecture

- Critical sections
- Synchronisation of processes by a clock
- Synchronisation of processes by a communicate
- Complex synchronisation of processes

Prior lectures
- Applications of LTL logic in system analysis
- Finite state automata
- Properties of concurrent programs
- System verification with LTL and automata
- Automatic Verification in UPPAAL

Next lectures
- Modelling and verifying concurrency
Contents of the lecture

- CTL logic
- Applications of CTL logic in system verification
- RTCTL and PRTCTL logic
- Temporal databases

Prior lectures

Applications of LTL logic in system analysis

Finite state automata

Properties of concurrent programs

System verification with LTL and automata

Automatic Verification in UPPAAL

Modelling and verifying concurrency

Next lectures

2013/2014
Applications of LTL logic in system analysis

• What is LTL logic applied to in the system analysis
What is LTL logic applied to in the system analysis?

- to specify deterministic algorithms,
- to specify and verify properties of deterministic or indeterministic algorithms.

In both cases an LTL description can be related to finite state automata.
Finite state automata

- What is a finite state automaton
- Example: an automaton closing a door
- Example: an automaton setting the light on and off
What is a finite state automaton?

- is an abstract state machine,
- consists of a finite number of states and transitions between them,
- has initial states and may have final states,
- is an automaton, through which transitions between states are fired deterministically (interchangeably described by a transition function).
Automaton $A=\{\Sigma, S, S_0, \rho, F\}$ is defined by:

- $\Sigma$ – an alphabet (set of program's states),
- $S$ – a set of automaton's states,
- $S_0$ – a set of initial states,
- $\rho$ – a transition function ($S \times \Sigma \times S$),
- $F$ – a set of final states.

A trace is a path of states, e.g. $s_1, s_2, s_2, s_3$.

A state can be described by propositions, that are true in it.
Example: an automaton closing a door

Goal: to automatically close the door when opened.

Assumptions:

- $a_1$ — at the beginning the door is opened,
- $a_2$ — the door closes instantly (time of closing equals zero).

Proposition $p$ — the door is closed

At the moment ($s_1$) of turning the automaton on it is true, that:

$$\neg p \land XGp$$
Example: an automaton setting the light on and off

Goal: to automatically set the light on at 18:00 and off at 6:00.

Assumptions:

- $a_1$ – in the beginning the light is off,
- $a_2$ – the light is set on at 18:00 and off at 6:00,
- $a_3$ – the light is set on and off instantly.
- $a_4$ – the timer H starts at 12:00.

proposition $p$ – the light is on
Finite state automata

Example: an automaton setting the light on and off

At the moment \( (s_1) \) of turning the automaton on it is true that:

\[
\neg p \land H=12 \land G(\neg p \land H \neq 18 \Rightarrow X \neg p) \land G(\neg p \land H=18 \Rightarrow Xp) \land G(p \land H \neq 6 \Rightarrow Xp) \land G(p \land H=6 \Rightarrow X \neg p)
\]

The automaton must satisfy the following:

- the lamp works: \( GFp \land GF \neg p \)
- the lamp does not twinkle: \( \neg F(p \land X \neg p \land XXp) \)
Properties of concurrent programs

- What is a concurrent program
- Properties of concurrent programs
- Examples of invariant properties
Properties of concurrent programs

What is a concurrent program?

A concurrent program consists of processes in which:

- processes run at the same time;
- processes may share some resources, e.g. variables;
- processes may interact with each other.
Properties of concurrent programs

A concurrent program $P$ consists of concurrently running processes: $P_1$, $P_2$, $P_3$, ...

Properties of $P$ are: $q_1$, $q_2$, $q_3$, $q_4$, ...

A property $q_i$ of $P$ may be specified by a formula $\phi_i$, which will be satisfied for each run of $P$ (from the first state in which $s_0 \models \phi_i$), independently of the way of managing the processes.
Properties of concurrent programs

safety — $q_1$ is satisfied at every moment: $\phi_1 \equiv Gq_1$
  e.g. access to the critical section CS is always possible.

satisfiability — $q_2$ will be satisfied at some moment: $\phi_2 \equiv Fq_2$
  e.g. every process will eventually get access to CS.

answer — $q_3$ is satisfied from time to time: $\phi_3 \equiv GFq_3$
  e.g. every process is set to sleep from time to time.

persistence — from some moment: $q_4$ is satisfied at every moment: $\phi_4 \equiv FGq_4$
  e.g. from some moment: no process wants to get access to CS.
Examples of invariant properties

The process $P_1$ places lines of text into a table “lines”.

The process $P_2$ removes lines from “lines”, if they are incorrect.

$P_1$ and $P_2$ run concurrently.

$P_1$: while (true) do { lines.add(new line); }

$P_2$: while (true) do { if(lines.last.incorrect) {lines.last.remove(); }}
Examples of invariant properties

Will these properties be satisfied?

• “lines” never contains more than one incorrect line:

\[ \neg F(\text{lines.incorrect.count} > 1) \]

• every line exists in “lines” for a time:

\[ (\forall \text{line})(F(\text{lines.contains(line)})) \]
System verification with LTL and automata

- System model verification
- Most important properties
- A system model verification algorithm
- A simple example
System model verification

Input data:

- a formal model of system functions given as an automaton;
- properties, which must be satisfied by the system, given as LTL formulas.

Output:

- answer whether the system satisfies the properties.

Does every/any possible sequence of states of the system (program) satisfy the properties?
System verification with LTL and automata

Most important properties

- **reachability**
  \[ F_p \] — the “wanted” state \( p \) of the system will eventually be reached,

- **safety**
  \[ G\neg q \] — the “unwanted” state \( q \) of the system will never be reached.

Verify (automatically) whether the formula is valid (true) in a given model.
A system model verification algorithm

1) Build a finite state automaton $A_S$ for the system's model $S$.
2) Write the system's properties as an LTL formula $f$.
3) Build a finite state automaton $A_{\neg f}$ for the formula $\neg f$.
4) Build a product automaton for $A_P = A_S \times A_{\neg f}$.
5) Verify (automatically) whether $A_P$ exists.

The problem of the model verification is decidable.
A system model verification algorithm

- $A_{\neg f}$ should accept only such sequences of states of the program of the system $S$ that the formula $\neg f$ is satisfied.
- If there exists such a sequence of states in $A_s$ that corresponds to the formula $\neg f$, then such a run of the program is possible that does not satisfy the properties.
A simple example

- The automaton $A_s$ models a system, where each state is labelled by all these propositions, which are true in it: $a$ and $b$. ($\emptyset$ – $a$ and $b$ are false)

- Is the property $f \equiv Fb$ satisfied for $A_s$?

- The automaton $A_{\neg f}$ models the formula $\neg f \equiv \neg Fb \equiv G\neg b$.

- No infinite trace of $A_{\neg f}$ exists in $A_s$ – the property $f$ is satisfied.
Automatic verification in UPPAAL

- Tools for automatic verification
- What does UPPAAL allow to
- How to use the UPPAL modeller, simulator and verifier
### Tools for automatic verification

<table>
<thead>
<tr>
<th>tool</th>
<th>system model description</th>
<th>temporal logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kronos</td>
<td>language ET-LOTOS</td>
<td>CTL (TCTL)</td>
</tr>
<tr>
<td>SMV</td>
<td>language SMV</td>
<td>LTL, CTL</td>
</tr>
<tr>
<td>Spin</td>
<td>language PROMELA</td>
<td>LTL</td>
</tr>
<tr>
<td><strong>UPPAAL</strong></td>
<td>graphic</td>
<td><strong>CTL</strong></td>
</tr>
<tr>
<td>Verus</td>
<td>language Verus</td>
<td>LTL, CTL, RTCTL, PRTCTL</td>
</tr>
</tbody>
</table>

*CTL, RTCTL and PRTCTL will be described in later lectures.*
What does UPPAAL allow to?

- graphically model a system as finite state automata,
- use time automata (automata with clocks),
- graphically simulate possible runs of the model,
- specify some properties of the system as CTL formulas (LTL is a “subset” of CTL),
- verify the properties for the model.

UPPAAL may be applied to real-time systems, including concurrent programs.
Automatic verification in UPPAAL

What does UPPAAL allow to?

UPPAAL is comfortable in use.

All you need to know is how to build a finite state automaton, write CTL formulas and write simple basic code.

1) Model automata
2) Simulate them
3) Verify formulas
How to use the UPPAL modeller, simulator and verifier?

Step 1. **Modelling** — build a system model as an automaton or automata (graphically).

Step 2. **Simulating** — step by step check whether the model acts as expected (graphically).

Step 3. Write the system's properties as logic formulas.

Step 4. **Verifying** — automatically verify truth of these formulas.
Modelling and verifying concurrency

- Critical sections
- Synchronisation of processes by a clock
- Synchronisation of processes by a communicate
- Complex synchronisation of processes
Critical sections

A program consists of two “identical” concurrent processes.

Assumptions:

- **mutual exclusion** — the two processes cannot be both in the critical section at the same time.
- **progress** — no process can access the critical section once again, while the other process is waiting to access it.

Conclusion:

- the processes will be in the critical section one after another and so on.
Modelling and verifying concurrency

Critical sections

Petterson's algorithm for mutual exclusion for 2 processes:
(example from “UPPAAL 4.0 Small Tutorial”)

Process P1:
\[
\begin{align*}
\text{req1} &= 1; & \quad & \text{// the “idle” state} \\
\text{turn} &= 2; & \quad & \text{// the “want” state} \\
\text{while}(\text{turn} \neq 1 \&\& \text{req2} \neq 0); & \quad & \text{// the “wait” state} \\
\text{req1} &= 0; & \quad & \text{// the “CS” state}
\end{align*}
\]

Process P2:
\[
\begin{align*}
\text{req2} &= 1; & \quad & \text{// the “idle” state} \\
\text{turn} &= 1; & \quad & \text{// the “want” state} \\
\text{while}(\text{turn} \neq 2 \&\& \text{req1} \neq 0); & \quad & \text{// the “wait” state} \\
\text{req2} &= 0; & \quad & \text{// the “CS” state}
\end{align*}
\]

\text{req1, req2, turn} – control variables; \quad \text{CS} – critical section
Critical sections

Model of processes P1 (left) and P2 (right):
(example from “UPPAAL 4.0 Small Tutorial”)

Modelling and verifying concurrency
Critical sections

**safety** — mutual exclusion of both processes:
\[ G\neg(P1.CS \land P2.CS) \]

in UPPAAL: \[ A\[] \neg (P1.CS \land P2.CS) \]

**reachability** — access to the critical section for every process:
\[ F(P1.CS) \land F(P2.CS) \]

in UPPAAL:
\[ E<>(P1.CS) \land E<>(P2.CS) \]
Critical sections

**safety** — mutual exclusion of both processes:

\[ G \neg (P1.CS \land P2.CS) \]

in UPPAAL: \[ A[] \neg (P1.CS \land P2.CS) \] satisfied

**reachability** — access to the critical section for every process:

\[ F(P1.CS) \land F(P2.CS) \]

in UPPAAL: \[ E<>(P1.CS) \land E<>(P2.CS) \] satisfied

What other properties can you find?
Modelling and verifying concurrency

Synchronisation of processes with a clock

The automata A1 and A2, controlled by a clock variable x, are almost identical. What are the differences?

A1: the clock controls the transition between states S1 and S2:
- the transition S1 → S2 is available until 5 time units have passed but may never be fired:
  \(\neg F(A1.S1 \land x>5 \land X A1.S2)\)

A2: the clock controls the state S1:
- the state must be left until 5 time units have passed:
  \(\neg F(A2.S1 \land x>5)\)
Modelling and verifying concurrency

Synchronisation of processes with a clock

The automaton \textit{Lecture} is controlled by a clock variable \(x\).

The lecture lasts exactly 105 minutes.

\textbf{Safety}: \( G(\text{Lecture.on} \land x<105 \Rightarrow X \text{Lecture.on}) \)

\textbf{Reachability}: \( F(\text{Lecture.off} \land x>105) \)

What other properties can you find?
Synchronisation of processes by a communicate

Processes *Policeman* and *Criminal* are synchronised with each other:

- When the *Policeman* spots the *Criminal*, he sends him a communicate: **don't move!** and then the *Criminal* hears it and stops.

C in the spots state means that the state must be left instantly at the same moment the state is entered (such state is called *committed*).
Modelling and verifying concurrency

Complex synchronisation of processes

In the summer, when a day lasts 16 hours, and night lasts 8 hours, the Mosquito hunts for your blood.

The Time process is always in one of two states: day and night. Transitions between them are controlled by a clock $x$.

On a day Time tells Mosquito to sleep.

In the night, Time tells Mosquito to fly.

The Mosquito listens to the Time.
Modelling and verifying concurrency

Complex synchronisation of processes

Notice that the *Mosquito* has no clock, it only observes the *Time* (day, night).

Correct life of the *Mosquito* can be specified by the following formulas:

- during a day, the *Mosquito* sleeps: 
  \[ G(Time.day \Rightarrow Mosquito.sleeps) \]

- during a night, the *Mosquito* flies: 
  \[ G(Time.night \Rightarrow Mosquito.flies) \]

What other properties can you find?
Next lectures

- CTL logic
- Applications of CTL logic in system verification
- RTCTL and PRTCTL logic
- Temporal databases
The end

Literature: