System specification with temporal logic
[Automated Reasoning, 2012/2013 1b — Lecture 3]

Doina Bucur

RuG

Nov 2012
In this lecture...

Specifications and Linear Temporal Logic (LTL)
  Temporal logics
  Defining LTL operators and formulas
  Checking LTL formulas inductively
  Writing LTL formulas. Dualities, patterns and scopes
  Safety, liveness, fairness
  Alternative formalism to temporal logics
  Writing specifications in PROMELA

Historical and bibliographical notes

Assignment

We learn the most widespread language to write system specifications, Linear Temporal Logic (LTL); we show induction rules to verify a Kripke structure against a given LTL formula. Finally, we categorize specifications.
Remember: labelled state spaces
Temporal logics

**Need:** specify causal and temporal properties of (in)finite executions.

How to use temporal logic specifications:

- We take a Kripke structure
- ...and unwind it from any initial state.
- The result is an infinite tree showing all possible executions.
- Temporal-logic formulas express properties for states and paths.
Linear-Time Logic (LTL)

A simple, useful temporal logic is LTL: it writes properties for a single path\(^1\).

Examples of LTL path formulas \( p \):

- \( p \equiv (\text{turn} = 0) \):
  In the first state \( s \) on this path, \( p \in L(s) \).
  Atomic propositions from AP are LTL formulas.

- \( p \equiv (q \land \neg r) \):
  Standard **logical operators** compose LTL formulas out of other LTL formulas.

- \( p \equiv (Gq) \):
  New **temporal operators** do the same.
  \( Gq \): Globally on this path (i.e. for all states), \( q \) holds.

\(^1\)The more general variant is Computation Tree Logic (CTL\(^*\)): its formulas write properties for more paths at a time. For software checking, LTL is sufficient.
LTL temporal operators (1)

**Xₚ** “next time”.
Property $p$ holds at the second state on the path.

```
Xₚ:
p

!p
```

**Fₚ** “in the future” or “eventually”, also denoted $\Diamond p$.
Property $p$ holds at some state on the path (after a finite number of states).

```
Fₚ:
p

!p
```

**Gₚ** “globally” or “always”, also denoted $\Box p$.
Property $p$ holds at every state on the path.

```
Gₚ:
p

!p
```
LTL temporal operators (2)

\( p \mathbf{U} q \) “until”.

This holds if a state exists on the path where \( q \) holds, and at every preceding state on the path \( p \) holds.
**LTL temporal operators (3)**

$pRq$ \textit{“release”}.

Dual of “until”.

This holds if a state exists on the path where $p$ holds, and at every preceding state on the path (including that state) $q$ holds.

Alternatively, $q$ holds all the time.
LTL formulas

Definition (LTL formula)

1. If $p \in AP$, then $p$ is a state formula.
2. If $f_1$ and $f_2$ are state formulas, then $\neg f_1, f_1 \lor f_2, f_1 \land f_2$ are state formulas.
3. A state formula is also a path formula.
4. If $g_1$ and $g_2$ are path formulas, then $\neg g_1, g_1 \lor g_2, g_1 \land g_2, Xg_1, Fg_1, Gg_1, g_1 U g_2, g_1 R g_2$ are path formulas.

Note: Logical implication and equivalence are the usual shorthands:

- $p \to q$ is $(\neg p) \lor q$, and
- $p \leftrightarrow q$ is $(p \to q) \land (q \to p)$. 
Induction rules (1): state formulas

Say $M$ is a Kripke structure, $f_i$ state formulas and $g_i$ path formulas. We write $s \models f_1$ to say that $f_1$ holds on state $s$ in $M$, and $\pi \models g_1$ to say that $g_1$ holds along path $\pi$ in $M$.

(Reminder: you already know how to check $s \models f_1$, i.e., an assertion, over a Kripke structure).

You can imagine that $s \models$ in LTL is formalized simply:

\[
\begin{align*}
    s \models p & \iff p \in L(s) \\
    s \models \neg f_1 & \iff s \not\models f_1 \\
    s \models f_1 \land f_2 & \iff s \models f_1 \text{ and } s \models f_2 \\
    s \models f_1 \lor f_2 & \iff s \models f_1 \text{ or } s \models f_2
\end{align*}
\]
Induction rules (2): path formulas

\( \pi \models \) is defined inductively:

\[
\begin{align*}
\pi \models f_1 & \iff \pi \text{ starts in } s \text{ and } s \models f_1 \\
\pi \models \neg g_1 & \iff \pi \not\models g_1 \\
\pi \models g_1 \lor g_2 & \iff \pi \models g_1 \text{ or } \pi \models g_2 \\
\pi \models Xg_1 & \iff \pi^1 \models g_1 \\
\pi \models Fg_1 & \iff \exists k \geq 0. \pi^k \models g_1 \\
\pi \models Gg_1 & \iff \forall k \geq 0. \pi^k \models g_1 \\
\pi \models g_1 U g_2 & \iff \exists k \geq 0. \pi^k \models g_2 \text{ and } \forall i, 0 \leq i < k. \pi^i \models g_1 \\
\pi \models g_1 R g_2 & \iff \forall j \geq 0. \text{ if } \forall i \leq j. \pi^i \models g_1 \text{ then } \pi^j \models g_2
\end{align*}
\]
Some intuition

Question: An unwound Kripke structure $M$ has any number of different executions. How can a LTL path formula $f$, with $f \equiv F(pc_0 = cr_0 \land pc_1 = cr_1)$ (which describes one path) specify the entire $M$?

Answer: It can, only when you add a path quantifier to $f$: “For all computation paths in $M$, $f$”.

$A$ (“for all computation paths”) and $E$ (“for some computation path”) are path quantifiers, part of the Computation Tree Logic (CTL*), a superset of LTL. They act like the logical quantifiers $\forall x. P$ and $\exists x. P$, but on executions instead of on variables. (And one of them is sufficient to write any formula.)

$Af$ is a state formula.
Some examples

\(pc_0 = 0, pc_1 = l_1\)
\(turn = 0\)

\(pc_0 = l_0, pc_1 = nc_1\)
\(turn = 0\)

\(pc_0 = l_0, pc_1 = nc_1\)
\(turn = 0\)

\(pc_0 = nc_0, pc_1 = l_1\)
\(turn = 0\)

\(pc_0 = l_0, pc_1 = l_1\)
\(turn = 1\)

\(pc_0 = l_0, pc_1 = nc_1\)
\(turn = 1\)

\(pc_0 = l_0, pc_1 = nc_1\)
\(turn = 1\)

\(pc_0 = cr_0, pc_1 = l_1\)
\(turn = 1\)

\(pc_0 = nc_0, pc_1 = cr_1\)
\(turn = 0\)

\(pc_0 = nc_0, pc_1 = cr_1\)
\(turn = 0\)

\(pc_0 = nc_0, pc_1 = cr_1\)
\(turn = 1\)

\(pc_0 = nc_0, pc_1 = cr_1\)
\(turn = 1\)

With the initial state where \(turn = 0\), are the following properties true for all computation paths \(A\)?

\[XX(turn = 0)\]
\[F((pc_0 = cr_0) \land (pc_1 = cr_1))\]
\[(turn = 0)U(turn = 1)\]
\[GF(pc_1 = cr_1)\]
Operator minimality

The set of temporal operators in LTL is not minimal:

- \( Gp \) is shorthand for: \( p \ W \) false\(^2\)
  (‘false’ is a state label which is false in any state);
- \( Fp \) is shorthand for: true \( U \) \( p \)
  (‘true’ is a state label which is true in any state).

\(^2\)This new operator \( W \) is called a ‘Weak until’, and is somehow different than Until; can you tell how?
Some interesting dualities

You can prove that:

\[
\begin{align*}
\mathsf{G}p & \iff \mathsf{GG}p \\
\neg \mathsf{G}p & \iff \mathsf{F}\neg p \\
\mathsf{G}(p \land q) & \iff \mathsf{G}p \land \mathsf{G}q \\
\mathsf{F}(p \lor q) & \iff \mathsf{F}p \lor \mathsf{F}q \\
pUq & \iff pU(pUq) \\
\neg (pUq) & \iff (\neg q)U(\neg p \land \neg q) \\
\mathsf{GF}(p \lor q) & \iff \mathsf{GF}p \lor \mathsf{GF}q \\
\mathsf{A}p & \iff \neg \mathsf{E}\neg p
\end{align*}
\]
Writing LTL from natural-language specifications... (1)

...requires guesswork on the correct semantics of natural language.

- Take the informal causal property (in English) “p implies q”.
- You write $p \rightarrow q$: a state formula, necessarily true only in the first state.
- Perhaps we should apply this to all states, $G(p \rightarrow q)$?
- But perhaps this should be a temporal implication: $G(p \rightarrow Fq)$.
- Still, this allows $p$ never to become true (with the formula being meaninglessly true). To expect that $p$ does become true eventually, we write $G(p \rightarrow Fq) \land Fp$. 

Writing LTL from natural-language specifications... (2)

...can be daunting.

The specification “Between the time an elevator is called at a floor and the time it opens its doors at that floor, the elevator can arrive at that floor at most twice”\(^3\) is in LTL:

\[
\Box((\text{call} \land \Diamond \text{open}) \rightarrow
(\neg \text{atfloor} \land \neg \text{open}) \cup
(\text{open} \lor ((\text{atfloor} \land \neg \text{open}) \cup
(\text{open} \lor ((\neg \text{atfloor} \land \neg \text{open}) \cup
(\text{open} \lor ((\text{atfloor} \land \neg \text{open}) \cup
(\text{open} \lor ((\neg \text{atfloor} \cup \text{open})))))))))
\]


http://patterns.projects.cis.ksu.edu/documentation/patterns.shtml (same authors) is also more informative than this overview.
Patterns and scopes to the rescue

Specifications naturally fall into patterns; on the right, a survey of 500+ collected specifications:

<table>
<thead>
<tr>
<th>Patterns</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Response</td>
<td>245</td>
</tr>
<tr>
<td>Universality</td>
<td>119</td>
</tr>
<tr>
<td>Absence</td>
<td>85</td>
</tr>
<tr>
<td>UNKNOWN</td>
<td>44</td>
</tr>
<tr>
<td>Precedence</td>
<td>26</td>
</tr>
<tr>
<td>Existence</td>
<td>26</td>
</tr>
</tbody>
</table>

**Formula** (with \( p \) a state formula)

- \( Gp \): \( p \) is always true
- \( Fp \): \( p \) is eventually true
- \( p \rightarrow Fq \): \( p \) implies eventually \( q \)
- \( \neg qWp \): \( p \) precedes \( q \)
- \( GFp \): always eventually \( p \)
- \( FGp \): eventually always \( p \)
- \( Fp \rightarrow Fq \): eventually \( p \) implies eventually \( q \)

**Pattern names**

- universality/absence/invariant
- existence/guarantee
- response
- precedence
- recurrence (progress)
- stability (non-progress)
- correlation
Frequently used scopes

For more complex properties, place the previous patterns into scopes.

Pattern \( P \) happens:

- **globally:** \( G P \)
- **before** \( Q \): \( FQ \rightarrow (PUQ) \)
- **after** \( Q \): \( G(Q \rightarrow GP) \)
- **between** \( Q \) and \( R \): \( G((Q \land \neg R \land FR) \rightarrow (PUR)) \)

---

### Scopes

<table>
<thead>
<tr>
<th>Scope</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global</td>
<td>438</td>
</tr>
<tr>
<td>Between</td>
<td>29</td>
</tr>
<tr>
<td>After</td>
<td>25</td>
</tr>
<tr>
<td>Until</td>
<td>11</td>
</tr>
<tr>
<td>Before</td>
<td>8</td>
</tr>
</tbody>
</table>

---

- **Global**
- **Before** \( Q \)
- **After** \( Q \)
- **Between** \( Q \) and \( R \)
- **After** \( Q \) until \( R \)
- **State Sequence** \( Q \rightarrow R \rightarrow Q \rightarrow R \rightarrow Q \)
Safety specifications

“Nothing bad should happen”.

Examples:

- Mutual exclusion $G\neg (pc_0 = cr_0 \land pc_1 = cr_1)$;
- Deadlock freedom;
- (generally, any invariant $Gp$);
- But also: any property for which an error trace contains a finite bad prefix, such as precedence. I.e., safety properties are violated in finite time.
Liveness specifications

“Something good will eventually happen”.

Examples:

- A guarantee $F_p$ or progress $GF_p$ capture liveness;
- Starvation freedom (a form of fairness): each waiting process will eventually enter its critical section;
- Assume a system with infinite executions. Liveness properties are violated in infinite time.
- Safety and liveness properties are disjoint.
- Liveness complements safety. Any LTL formula can be rewritten as a conjunction of safety and liveness properties.
Fairness

A **fair** path is, intuitively, one on which certain states do appear regularly, instead of being ignored forever by the program scheduler. Can be expressed with a **progress** formula.

An example: for a communication channel, a fairness constraint is a state in which a message is received at an end of the channel, if it was sent at the other.

**Definition (Fair Kripke structure)**

A **fair** Kripke structure is $M = (S, S_0, T, L, F)$, where all is as before besides the new $F \subseteq 2^S$.

$F$ is the set of **fairness constraints**\(^4\).

$\pi$ is called a **fair path** if every state from $F$ appears in $\pi$ infinitely often.

---

\(^4\)Fairness constraints are often called generalized Büchi acceptance conditions. We see about Büchi automata next time.
Alternative: Automata as specifications

You can also express these (and more) specifications as automata\(^5\).

\[ \text{true} \quad \xrightarrow{p} \quad p \]

\[ q_0 \rightarrow p \rightarrow q_1 \]

automaton for \( FG \, p \)

However, the complexity of expressing a specification with automata is higher than with temporal logics. The same goes for operating on a specification (think how hard it is to complement an automaton). This is why temporal logics are more widely used as specification formalisms.

Nevertheless, automata are one of the main implementation techniques for temporal-logic model checking. We see this next time.

\(^5\) Also, with other logics, calculi, regular expressions.
**Promela constructs**

**Promela** also recently expresses LTL (latest version); in general, you can specify:

- **Assertions** `assert` (executable state formula `p`)
  Added at a particular program state; if state is reachable, `p` should not evaluate to false. The most commonly used specifications in practice, by far.

- **LTL formulas** with the `ltl` keyword;

- **Meta labels** for end states, progress states, accept states;

- **Never claims** `never`{labelled automata}, checked at each state i.e. `Gp` (both safety and liveness).
  `Spin` translates LTL formulas into `never` claims internally.

- **Traces** express allowed sequences of channel operations.
Historical notes

Amir Pnueli first proposed the use of *temporal logics* in the analysis of distributed systems in 1977; it took another decade for this to become mainstream.

Since then, various extensions of LTL have been found of use. A flavour of LTL with *past operators* can write more succinct formulas than standard LTL. LTL is the basis of the *Process Specification Language* (PSL), a standardized industrial specification language: [http://en.wikipedia.org/wiki/ISO_18629](http://en.wikipedia.org/wiki/ISO_18629).

The terms *safety* and *liveness* were first formalized by Leslie Lamport in 1983.
Bibliographical notes

The content in this lecture covers Ch. 3, *Temporal Logics* (the bits concerning LTL), from *Model Checking*, E. M. Clarke, O. Grumberg, D. A. Peled.

Go for the Temporal Logic section in Ch. 6, *Automata and Logic*, then Ch. 4, *Defining Correctness Claims* from *The SPIN Model Checker*, G. J. Holzmann.

You may also read on the topic from Ch. 3, *Linear Time Properties*, from *Principles of Model Checking*, C. Baier, J.-P. Katoen.
Most of the following assignments originate in G. Holzmann’s *Logic Model Checking* course at Caltech in 2011 [LMC’11]; you find these on Nestor under ‘Course material’.

(50%) Do [LMC’11] Assignment 1, all four points. This teaches you some basics of what assertions are useful for, how to check for final variable values and termination, and reminds you of how to look for deadlock and starvation.

(50%) Prove any two of the “interesting dualities” on page 15. Then, from [LMC’11] Assignment 6, go for points 1, 2 and 3. These ask you to write some LTL. In particular, point 2 in this assignment deals with writing LTL properties for algorithms sharing critical sections (mutual exclusion, progress, bounded waiting). Then, resurrect Peterson’s algorithm from your previous assignment, and have ./pan -a verify these LTL properties (you will need the -a flag, and to have a look through http://spinroot.com/spin/Man/ltl.html).
Assignment administration

Handing in:
▶ in lab session or electronically;
▶ either way, before the end of Fri Dec 7; that’s one and a half weeks.

Grading:
▶ this is worth 25% of your Practical grade (and 15% of the final grade).