Why temporal logic

- a formal specification language

\[ \text{a way to specify what we want mathematically (unambiguously!)} \]

*Amir Pnueli* (1941 - 2009) won the *ACM Turing Award*, in part for proposing to use temporal logic for specification.

Amir Pnueli (1941 - 2009) won the ACM Turing Award, in part for proposing to use temporal logic for specification.
Example: Specification of the SpaceWire Protocol (European Space Agency standard)

8.5.2.2 ErrorReset

a. The ErrorReset state shall be entered after a system reset, after link operation is terminated for any reason or if there is an error during link initialization.

b. In the ErrorReset state the Transmitter and Receiver shall all be reset.

c. When the reset signal is de-asserted the ErrorReset state shall be left unconditionally after a delay of 6,4 μs (nominal) and the state machine shall move to the ErrorWait state.

d. Whenever the reset signal is asserted the state machine shall move immediately to the ErrorReset state and remain there until the reset signal is de-asserted.

From Sanjit Seshia.

Temporal Logics

- Many variants: for linear, branching, timed, continuous, security, ..., properties
- We will look at LTL (for so-called linear-time properties) and CTL (for so-called branching-time properties).
LTL (Linear Temporal Logic) – Syntax

LTL\(^1\) formulas are defined by the following grammar:

\[
\phi ::= p \mid q \mid \ldots \\
   \mid \phi_1 \land \phi_2 \mid \neg \phi_1 \\
   \mid G\phi_1 \\
   \mid F\phi_1 \\
   \mid X\phi_1 \\
   \mid \phi_1 U \phi_2
\]

Intuition:

- \(G\phi\): **globally** \(\phi\) (*always* \(\phi\)), also written \(\Box \phi\).
- \(F\phi\): in the **future** \(\phi\) (*eventually* \(\phi\)), also written \(\Diamond \phi\).
- \(X\phi\): **next** \(\phi\), also written \(\lhd \phi\).
- \(\phi_1 U \phi_2\): \(\phi_1\) **until** \(\phi_2\).

\(^1\)This is PLTL: Propositional LTL (there is also first-order LTL with quantifiers \(\forall, \exists\)).

LTL – Semantics: Intuition

LTL formulas are evaluated over infinite sequences (execution **traces**).

Satisfaction relation looks like this – for LTL formula \(\phi\) and infinite trace \(\sigma\):

\[
\sigma \models \phi
\]

<table>
<thead>
<tr>
<th>formula</th>
<th>mnemonic</th>
</tr>
</thead>
<tbody>
<tr>
<td>(p)</td>
<td>proposition (must hold now)</td>
</tr>
<tr>
<td>(G\phi)</td>
<td>always, globally</td>
</tr>
<tr>
<td>(F\phi)</td>
<td>finally, future, eventually</td>
</tr>
<tr>
<td>(X\phi)</td>
<td>next step</td>
</tr>
<tr>
<td>(\phi_1 U \phi_2)</td>
<td>until</td>
</tr>
</tbody>
</table>

Intuitive semantics
LTL: examples

$$\text{GF} \ p$$

$$\text{G}(p \rightarrow \text{F}q)$$

$$p \text{U} (q \text{U} (p \land r))$$

What do these formulas intuitively mean?

LTL – Semantics: Formally

We want to define formally the satisfaction relation: $$\sigma \models \phi$$.

What kind of object is $$\sigma$$?

An infinite trace of sets of atomic propositions:

$$\sigma \in (2^P)^\omega.$$  

That is,

$$\sigma = \sigma_0, \sigma_1, \sigma_2, \ldots$$

where $$\sigma_i \subseteq P$$ for all $$i$$. $$P$$ is the set of all atomic propositions.

Let $$P = \{p, q\}$$. Examples of traces:

$$\sigma = \{p\}, \{q\}, \{p\}, \{q\}, \{p\}, \ldots$$

$$\rho = \{p\}, \{p\}, \{p\}, \{p\}, \{p\}, \ldots$$

$$\tau = \{p\}, \{q\}, \{p, q\}, \{\}, \{p, q\}, \ldots$$

$$\ldots$$
LTL – Semantics: Formally

Let

\[ \sigma = \sigma_0, \sigma_1, \sigma_2, \cdots \]

Notation: \[ \sigma[i..] = \sigma_i, \sigma_{i+1}, \sigma_{i+2}, \cdots \]

Satisfaction relation defined recursively on the syntax of a formula:

\[
\begin{align*}
\sigma \models p & \quad \text{iff } p \in \sigma_0 \\
\sigma \models \phi_1 \land \phi_2 & \quad \text{iff } \sigma \models \phi_1 \text{ and } \sigma \models \phi_2 \\
\sigma \models \neg \phi & \quad \text{iff } \sigma \not\models \phi \\
\sigma \models G\phi & \quad \text{iff } \forall i = 0, 1, \ldots : \sigma[i..] \models \phi \\
\sigma \models F\phi & \quad \text{iff } \exists i = 0, 1, \ldots : \sigma[i..] \models \phi \\
\sigma \models X\phi & \quad \text{iff } \sigma[1..] \models \phi \\
\sigma \models \phi_1 U \phi_2 & \quad \text{iff } \exists i = 0, 1, \ldots : \sigma[i..] \models \phi_2 \land \\
& \quad \forall 0 \leq j < i : \sigma[j..] \models \phi_1
\end{align*}
\]


**LTL Semantics: Summary**

**LTL formulas:** Statements about an execution trace

$q_0, q_1, q_2, q_3, \ldots$

<table>
<thead>
<tr>
<th>formula</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$</td>
<td>$p$ holds in $q_0$</td>
</tr>
<tr>
<td>$G\phi$</td>
<td>$\phi$ holds for every suffix of the trace</td>
</tr>
<tr>
<td>$F\phi$</td>
<td>$\phi$ holds for some suffix of the trace</td>
</tr>
<tr>
<td>$X\phi$</td>
<td>$\phi$ holds for the trace $q_1, q_2, \ldots$</td>
</tr>
<tr>
<td>$\phi_1 U \phi_2$</td>
<td>$\phi_1$ holds for all suffixes of the trace until a suffix for which $\phi_2$ holds.</td>
</tr>
</tbody>
</table>

Here $p$ is a propositional logic formula and $\phi, \phi_1, \phi_2$ are propositional logic formulas or LTL formulas.

---

**State machines and temporal logic**

State machine = implementation (the *system* we want to verify).

LTL formula = specification (the *property* that we want the system to satisfy).

The **model checking problem**: does a given system (e.g., state machine) satisfy a given temporal logic specification (e.g., LTL formula)?

Meaning: **all** execution traces of the system must satisfy the LTL formula.
Execution traces of a state machine

A run of a Mealy machine \((I, O, S, s_0, \delta, \lambda)\) is a (finite or infinite) sequence of states / transitions:

\[
s_0 \xrightarrow{x_0/y_0} s_1 \xrightarrow{x_1/y_1} s_2 \xrightarrow{x_2/y_2} s_3 \cdots
\]

such that

- \(\forall i : x_i \in I, y_i \in O\)
- \(\forall i : s_{i+1} = \delta(s_i, x_i)\)
- \(\forall i : y_i = \lambda(s_i, x_i)\)

The observable I/O behavior (trace) corresponding to the above run is

\[
\{x_0, y_0\} \rightarrow \{x_1, y_1\} \rightarrow \{x_2, y_2\} \rightarrow \cdots
\]

Here we assume that only I/O are observable. We could also define traces that expose the internal state of the machine. E.g., we may want to state the requirement that a certain register never has a certain value.

System models can also be Transition Systems

**Transition system**: an even more basic model than state machines:

Transition system = states + transitions (+ labels)

**Possibly infinite** sets of states/transition.

- Can describe infinite-state systems (e.g., programs with integer or real variables).
- Can also be used in non-discrete systems (e.g., timed or hybrid automata, as we will see later).
- Form the basis for the semantics of temporal logics (LTL, CTL, ...) and other equivalences between systems such as (bi-)simulation.

Many variants: Labeled Transition Systems, Kripke Structures, ...
Example: Labeled Transition System

In a LTS the labels are on the transitions.

Example: Kripke Structure

In a KS the labels are on the states. Each state is labeled with a set of atomic propositions (those that hold on that state).
The model-checking problem for LTL

Given a transition system $M$ and an LTL formula $\phi$, check that all traces of $M$ satisfy $\phi$.

We write this as:

$$M \models \phi$$

(read “$M$ satisfies $\phi$”).

Examples

Let’s find transition systems satisfying or violating the following LTL formulas:

- $Gp$
- $Fp$
- $GFp$
- $G(p \rightarrow Fq)$
- $p \cup q$
Interesting facts about LTL

• Can we express $Gp$ using only $F$, $p$, and boolean operators?
• Vice versa, can we express $F$ in terms of $G$?
• Can we express $F$ in terms of $U$?
• Can we express $X$ in terms of $G$, $F$, $U$?
  ▶ Cannot be done!

LTL – more examples

See http://embedded.eecs.berkeley.edu/eecsx44/lectures/Fall2013/TemporalLogic.pdf.

Errata:
• Slides 13-14: “if and only if it holds” should be “if and only if $p$ holds”.
• Slide 19: $F(p \Rightarrow (XXq))$ should be $G(p \Rightarrow (XXq))$. 
Safety and Liveness

Two important classes of properties.

- **Safety** property: *something “bad” does not happen.*
  - E.g., system never crashes, division by zero never happens, voltage stays always $\leq K$ (never exceeds $K$), etc.
  - Finite length error trace.

- **Liveness** property: *something “good” must happen.*
  - E.g., every request must eventually receive a response.
  - Infinite length error trace.
Safety and Liveness

Are these LTL properties safety, liveness, something else?

- \(Gp\): safety.
- \(Fp\): liveness.
- \(Xp\): safety.
- \(p \cup q\): “both”.
- \(GFp\): liveness.
- \(G(p \Rightarrow Fq)\): liveness.
- \(G(p \Rightarrow Xq)\): safety.

Safety and Liveness – Formally

Let \(P\) be a set of atomic propositions.
\(2^P\) is the powerset of \(P\).
\((2^P)^*\) is the set of all finite sequences over \(P\).
\((2^P)^\omega\) is the set of all infinite sequences over \(P\).

What is a property, formally?

A property \(A\) is a set of traces: \(A \subseteq (2^P)^\omega\).

Examples:

- \(A = (2^P)^\omega\): \(A\) holds on all traces (every trace is in \(A\), i.e., every trace satisfies property \(A\)).
- \(A = \emptyset\): no trace satisfies \(A\).
- \(A = \ldots\) the set of all traces satisfying \(GFp\).
Safety and Liveness – Formally

Let $A$ be a property = set of (infinite) traces.

For a trace $\sigma$, and length $k \in \mathbb{N}$, we denote by $\sigma[1..k]$ the finite prefix $\sigma_1 \cdots \sigma_k$ of $\sigma$.

- $A$ is a safety property if
  \[ \forall \sigma \notin A : \exists k \in \mathbb{N} : \forall \rho \in (2^P)^{\omega} : \sigma[1..k] \cdot \rho \notin A \]
  i.e., for any $\sigma$ violating the safety property, there exists a **bad prefix** $\sigma[1..k]$, such that no matter how we extend this prefix we can no longer satisfy the safety property.

- $A$ is a liveness property if
  \[ \forall \sigma \in (2^P)^* : \exists \rho \in (2^P)^{\omega} : \sigma \cdot \rho \in A \]
  i.e., every finite trace can be extended, by appending a **good suffix**, into an infinite trace which satisfies the liveness property.

**Theorem ([Alpern and Schneider, 1985])**

*Every property is the intersection of a safety property and a liveness property.*
Bibliography

Defining liveness.

Principles of Model Checking.
MIT Press.

Model Checking.
MIT Press.

Handbook of theoretical computer science (vol. b).
chapter Temporal and modal logic, pages 995–1072. MIT Press.

Logic in Computer Science: Modelling and Reasoning about Systems.
Cambridge University Press.

Springer-Verlag, New York.

Temporal Verification of Reactive Systems: Safety.
Springer-Verlag, New York.

A temporal logic of concurrent programs.