System Specification, Verification and Synthesis (SSVS) – CS 4830/7485, Fall 2019

9: Formal Specification: Temporal logic LTL

Stavros Tripakis

Northeastern University
Khoury College of Computer Sciences

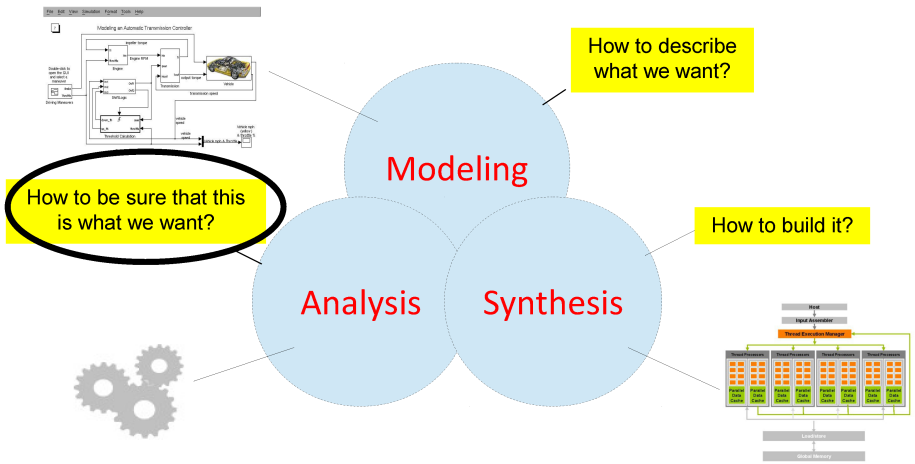
Where we stand

- We are done with the first part of the course: **systems**!
- We now know how to model systems, formally.

- We are ready to begin the second part: **specification**!
- Specification tries to answer questions like:
  What are the system requirements?
  What is my system supposed to do?
  What does it mean for my system to be correct?
  What properties does my system have?
  ...

Recall our ultimate goal: verification

Model-based design

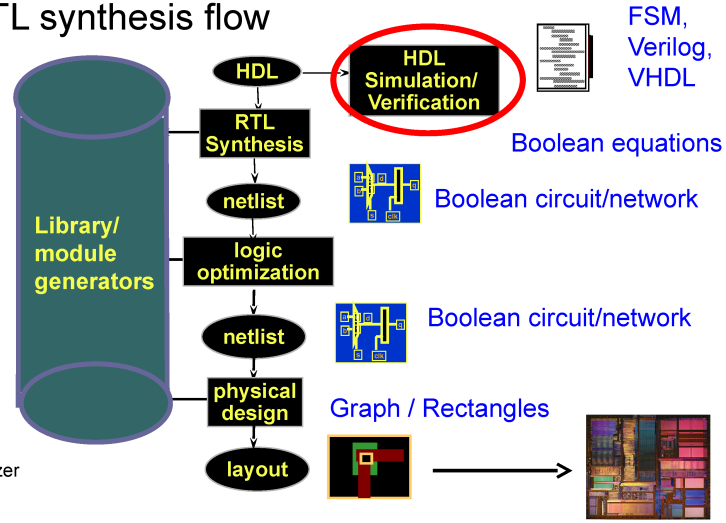


This version of the course focuses on Modeling and Analysis

33

Example of a successful model-based design flow

- RTL synthesis flow



K. Keutzer

39

We have designed a system.

We want to **verify** that it is **correct**.

But what does "correct" mean?

We need to **specify** correctness $\Rightarrow$ we need a specification language.

Current practice

Specifications often written in natural language, e.g., English.

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 Standard ECSS-E-ST-50-12C, *SpaceWire – Links, nodes, routers and networks*, 31 July 2008.

But English is often imprecise: recall our quiz

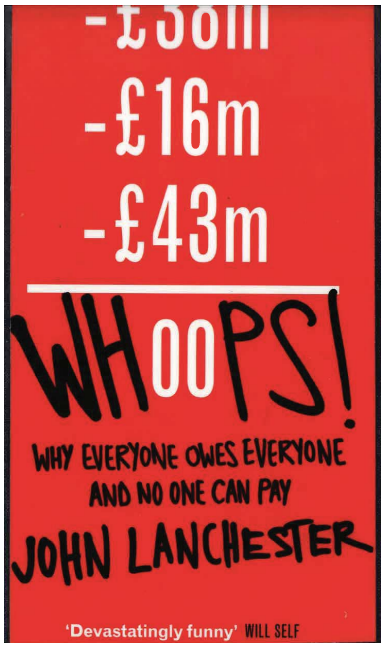
Express the following English statements in your favorite mathematical formalism:

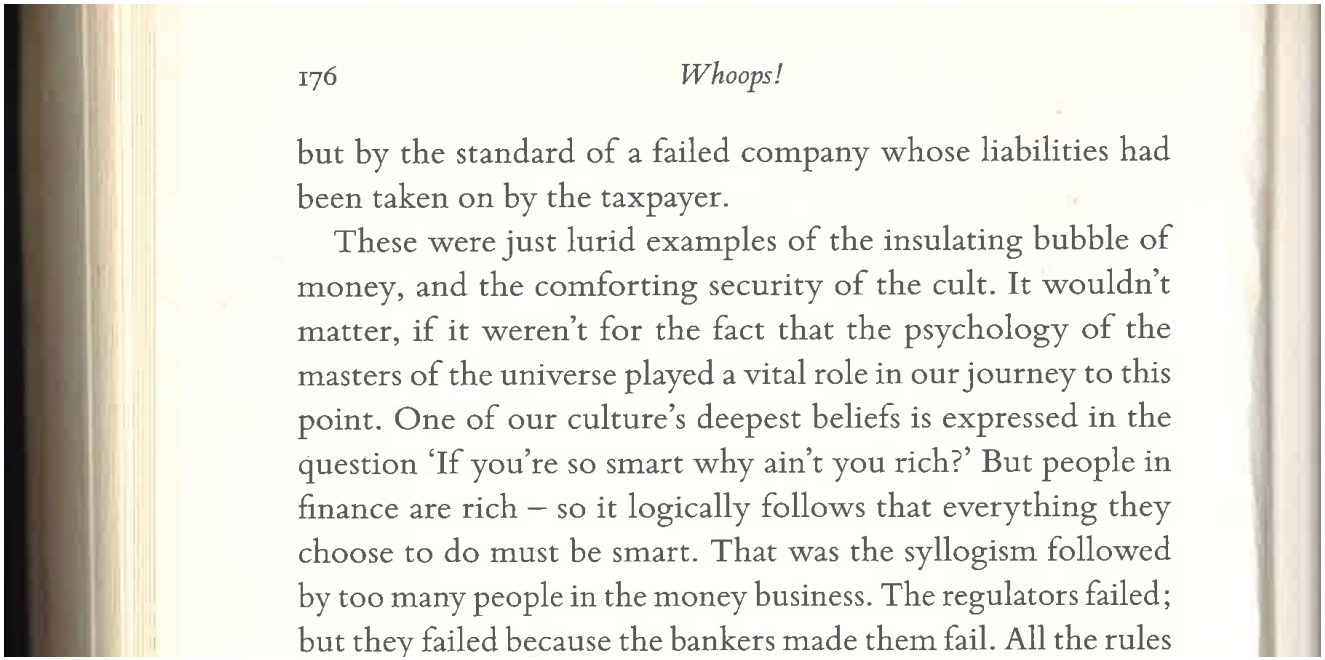
1. You can fool some people sometimes
2. You can fool some people all the time
3. You can fool some people sometimes but you can't fool all the people all the time [Bob Marley]
4. You can fool some of the people all of the time, and all of the people some of the time, but you cannot fool all of the people all of the time [Abraham Lincoln]

We need a **formal** (mathematical language) $\Rightarrow$ precise, unambiguous, amenable to automation.

We need a **logic**!

Language and logic





176 *Whoops!*

but by the standard of a failed company whose liabilities had been taken on by the taxpayer.

These were just lurid examples of the insulating bubble of money, and the comforting security of the cult. It wouldn't matter, if it weren't for the fact that the psychology of the masters of the universe played a vital role in our journey to this point. One of our culture's deepest beliefs is expressed in the question 'If you're so smart why ain't you rich?' But people in finance are rich – so it logically follows that everything they choose to do must be smart. That was the syllogism followed by too many people in the money business. The regulators failed; but they failed because the bankers made them fail. All the rules

A Logic Primer

- Knowledge of basic logic is important (in this course and beyond).
- *A Logic Primer (DRAFT)* posted on course web site.
- Go over it to refresh your logic background.
- Ask me if and when things are unclear.

Outline for the entire Specification part

- Temporal logic
  - Behaviors and properties
  - Linear-time behaviors: LTL
  - Safety and liveness
  - Branching-time behaviors: CTL
  - The model-checking problems for LTL and CTL
- Automata-based specifications
  - Finite vs. infinite behaviors
  - Deadlocks
  - Finite automata: DFA and NFA
  - Omega automata (ω-automata): Büchi automata
- Specification, abstraction, refinement:
  - Equivalences and preorders
  - Trace inclusion, trace equivalence
  - Simulation, bisimulation
  - Refinement

TEMPORAL LOGIC

Temporal logic

a **formal specification language**

=

a way to state properties of our system mathematically
(precisely and unambiguously!)
(as opposed to natural language)

Becoming more and more widespread in the industry
(hardware, robotics, ...)

Temporal logic

Amir Pnueli (1941 - 2009) won the ACM Turing Award in 1996.

> *"For seminal work introducing temporal logic into computing science and for outstanding contributions to program and system verification."*

Temporal logics

- Many variants: for linear-time, branching-time, real-time, probabilistic, security, ..., properties
- We will look at
  - **LTL** (*linear temporal logic*) for linear-time properties.
  - **CTL** (*computation tree logic*) for branching-time properties.

LINEAR-TIME BEHAVIORS and PROPERTIES

What is a "behavior"?

- We can think of a system, mathematically, as simply a set of behaviors.
- But what exactly is a behavior?
- The linear-time view: a behavior is an **infinite sequence** (of states, actions, sets of atomic propositions, ...)

What is a "behavior"?

- We can think of a system, mathematically, as simply a set of behaviors.
- But what exactly is a behavior?
- The linear-time view: a behavior is an **infinite sequence** (of states, actions, sets of atomic propositions, ...)
- Why not a finite sequence?
- **Reactive** systems $\Rightarrow$ they never stop!
- What if I have a system that might stop?
- No problem: add **stuttering** transitions (self-loops) to the legal stop states.
- We will return to this discussion later.

What is a "property"?

- The linear-time view: a property is a set of behaviors, i.e., a set of infinite sequences.
- Every formula in LTL defines a property, i.e., a set of infinite behaviors.
- We will make all this more mathematical when we talk about safety and liveness.

LTL: SYNTAX and SEMANTICS

LTL (Linear Temporal Logic) – Syntax

LTL[1] formulas are defined by the following grammar:

$$\begin{aligned}\phi \quad ::= \quad & p \mid q \mid ..., \text{ where } p, q, ... \in \text{AP (atomic propositions)} \\ & \mid \phi_1 \wedge \phi_2 \mid \neg\phi_1 \\ & \mid \mathbf{G}\phi_1 \mid \mathbf{F}\phi_1 \mid \mathbf{X}\phi_1 \mid \phi_1 \,\mathbf{U}\, \phi_2\end{aligned}$$

$\phi_1 \wedge \phi_2$:	ϕ_1 **and** ϕ_2 (logical conjunction)
$\neg\phi_1$:	**not** ϕ_1 (logical negation)
$\mathbf{G}\phi$:	**globally** ϕ (*always* ϕ), also written $\Box\phi$.
$\mathbf{F}\phi$:	in the **future** ϕ (*eventually* ϕ), also written $\Diamond\phi$.
$\mathbf{X}\phi$:	**next** ϕ, also written $\bigcirc\phi$.
$\phi_1 \,\mathbf{U}\, \phi_2$:	ϕ_1 **until** ϕ_2.

[1] We will only look at *propositional* LTL (PLTL). There is also first-order LTL with quantifiers $\forall, \exists$.

LTL – Syntax

We will also use

$\phi_1 \vee \phi_2$: ϕ_1 **or** ϕ_2 (logical disjunction)
can be defined as $\neg(\neg\phi_1 \wedge \neg\phi_2)$

$\phi_1 \rightarrow \phi_2$: ϕ_1 **implies** ϕ_2 (logical implication)
can be defined as $\neg\phi_1 \vee \phi_2$

$\phi_1 \leftrightarrow \phi_2$: ϕ_1 **iff** ϕ_2 (logical equivalence)
can be defined as $\phi_1 \rightarrow \phi_2 \wedge \phi_2 \rightarrow \phi_1$

LTL – Syntax

Recall LTL syntax:

$$\phi \quad ::= \quad p \mid q \mid ... \mid \phi_1 \wedge \phi_2 \mid \neg\phi_1 \mid \mathbf{G}\phi_1 \mid \mathbf{F}\phi_1 \mid \mathbf{X}\phi_1 \mid \phi_1 \,\mathbf{U}\, \phi_2$$

Examples: let's look at some syntactically correct (and some incorrect!) LTL formulas.

$$p \rightarrow q \qquad p \rightarrow \mathbf{G}p \qquad \mathbf{GF}p \qquad p\mathbf{G}$$

$$\mathbf{G} \wedge \mathbf{F}p \qquad \mathbf{G}(p \rightarrow \mathbf{F}q) \qquad \mathbf{G}(p \rightarrow \mathbf{F}) \qquad p\,\mathbf{U}\,(q\,\mathbf{U}\,(p \wedge r))$$

$$p\,\mathbf{U}\,(\mathbf{G}q) \qquad p\,\mathbf{U}\,(\,\mathbf{U}\,q) \qquad p\mathbf{X}q \qquad p \rightarrow \mathbf{XX}q$$

LTL – Syntax

syntactically correct	incorrect
$p \rightarrow q$	$p \rightarrow$
$\mathbf{G}p$	$p\mathbf{G}$
$\mathbf{GF}p$	$\mathbf{G} \wedge \mathbf{F}p$
$\mathbf{G}(p \rightarrow \mathbf{F}q)$	$\mathbf{G}(p \rightarrow \mathbf{F})$
$p\,\mathbf{U}\,(q\,\mathbf{U}\,(p \wedge r))$	$p\,\mathbf{U}\,(\,\mathbf{U}\,q)$
$p\,\mathbf{U}\,(\mathbf{G}q)$	$p\mathbf{X}q$
$p \rightarrow \mathbf{XX}q$	

LTL – Semantics

LTL formulas are evaluated over infinite sequences of sets of atomic propositions (execution **traces**).

$$\sigma = P_0,\ P_1,\ P_2,\ \cdots$$

where $P_i \subseteq$ AP for all i.

For instance, let AP $= \{p, q\}$. Examples of traces:

$$\begin{aligned} \sigma_1 &= \{p\}, \{q\}, \{p\}, \{q\}, \{p\}, ... \\ \sigma_2 &= \{p\}, \{p\}, \{p\}, \{p\}, \{p\}, ... \\ \sigma_3 &= \{p\}, \{q\}, \{p, q\}, \{\}, \{p, q\}, ... \\ &... \end{aligned}$$

LTL – Semantics

LTL formulas are evaluated over infinite sequences of sets of atomic propositions (execution **traces**).

$$\sigma = P_0,\ P_1,\ P_2,\ \cdots$$

where $P_i \subseteq \text{AP}$ for all i.

For instance, let $\text{AP} = \{p, q\}$. Examples of traces:

$$\begin{aligned} \sigma_1 &= \{p\}, \{q\}, \{p\}, \{q\}, \{p\}, ... \\ \sigma_2 &= \{p\}, \{p\}, \{p\}, \{p\}, \{p\}, ... \\ \sigma_3 &= \{p\}, \{q\}, \{p, q\}, \{\}, \{p, q\}, ... \\ &\dots \end{aligned}$$

What do these traces mean?

LTL – Semantics

LTL formulas are evaluated over infinite sequences of sets of atomic propositions (execution **traces**).

$$\sigma = P_0,\ P_1,\ P_2,\ \cdots$$

where $P_i \subseteq$ AP for all i.

For instance, let AP $= \{p, q\}$. Examples of traces:

$$\begin{aligned} \sigma_1 &= \{p\}, \{q\}, \{p\}, \{q\}, \{p\}, ... \\ \sigma_2 &= \{p\}, \{p\}, \{p\}, \{p\}, \{p\}, ... \\ \sigma_3 &= \{p\}, \{q\}, \{p, q\}, \{\}, \{p, q\}, ... \\ &... \end{aligned}$$

What do these traces mean? p holds at step i iff $p \in P_i$.

LTL – Semantics

LTL formulas are evaluated over infinite sequences of sets of atomic propositions (execution **traces**).

$$\sigma = P_0,\ P_1,\ P_2,\ \cdots$$

where $P_i \subseteq \text{AP}$ for all i.

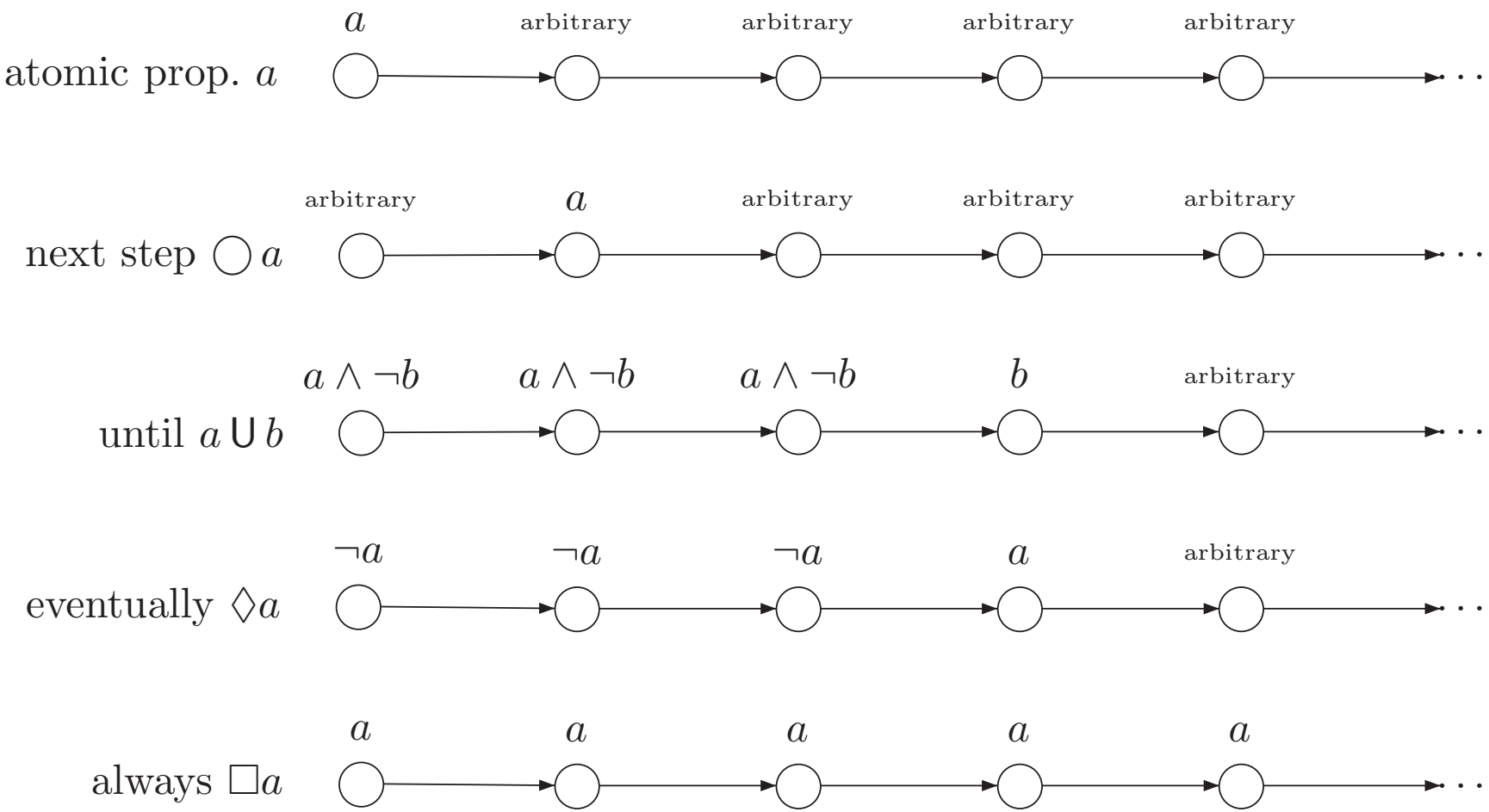
For instance, let $\text{AP} = \{p, q\}$. Examples of traces:

$$\begin{aligned} \sigma_1 &= \{p\}, \{q\}, \{p\}, \{q\}, \{p\}, ... \\ \sigma_2 &= \{p\}, \{p\}, \{p\}, \{p\}, \{p\}, ... \\ \sigma_3 &= \{p\}, \{q\}, \{p, q\}, \{\}, \{p, q\}, ... \\ &\dots \end{aligned}$$

What do these traces mean? p holds at step i iff $p \in P_i$.
Where do these traces come from? From (Kripke-style) transition systems (we'll see exactly how later).

LTL Semantics – Illustration

Figure taken from [Baier and Katoen, 2008]



LTL: examples

Let's find some traces that satisfy (and some that violate!) these formulas:

$$\mathbf{G}p \tag{1}$$

$$\mathbf{F}p \tag{2}$$

$$\mathbf{X}p \tag{3}$$

$$p\,\mathbf{U}\,q \tag{4}$$

$$\mathbf{GF}p \tag{5}$$

$$\mathbf{FG}p \tag{6}$$

$$\mathbf{G}(p \rightarrow \mathbf{F}q) \tag{7}$$

$$\mathbf{G}(p \rightarrow \mathbf{XX}q) \tag{8}$$

$$p\,\mathbf{U}\,(q\,\mathbf{U}\,(p \wedge r)) \tag{9}$$

LTL – Semantics: Formally

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

$$\sigma = P_0, P_1, P_2, \cdots$$

Notation (suffix): $\sigma[i..] = P_i, P_{i+1}, P_{i+2}, \cdots$.

Satisfaction relation defined recursively on the syntax of a formula:

$$\begin{array}{lll}
\sigma \models p & \text{iff} & p \in P_0 \\
\sigma \models \phi_1 \wedge \phi_2 & \text{iff} & \sigma \models \phi_1 \text{ and } \sigma \models \phi_2 \\
\sigma \models \neg\phi & \text{iff} & \sigma \not\models \phi \\
\sigma \models \mathbf{G}\phi & \text{iff} & \forall i = 0, 1, \ldots : \sigma[i..] \models \phi \\
\sigma \models \mathbf{F}\phi & \text{iff} & \exists i = 0, 1, \ldots : \sigma[i..] \models \phi \\
\sigma \models \mathbf{X}\phi & \text{iff} & \sigma[1..] \models \phi \\
\sigma \models \phi_1 \,\mathbf{U}\, \phi_2 & \text{iff} & \exists i = 0, 1, \ldots : \sigma[i..] \models \phi_2 \wedge \\
 & & \quad \forall 0 \leq j < i : \sigma[j..] \models \phi_1
\end{array}$$

LTL – Semantics: Formally

Let

$$\sigma = P_0, P_1, P_2, \cdots$$

Satisfaction relation defined recursively on the syntax of a formula:

$\sigma \models p$ iff $p \in P_0$ p holds at the first (current) step

$\sigma \models \phi_1 \wedge \phi_2$ iff $\sigma \models \phi_1$ and $\sigma \models \phi_2$

$\sigma \models \neg\phi$ iff $\sigma \not\models \phi$

$\sigma \models \mathbf{G}\phi$ iff $\forall i = 0, 1, \ldots : \sigma[i..] \models \phi$ ϕ holds for every suffix of σ

$\sigma \models \mathbf{F}\phi$ iff $\exists i = 0, 1, \ldots : \sigma[i..] \models \phi$ ϕ holds for some suffix of σ

$\sigma \models \mathbf{X}\phi$ iff $\sigma[1..] \models \phi$ ϕ holds for the suffix starting at the next step

$\sigma \models \phi_1 \,\mathbf{U}\, \phi_2$ iff $\exists i = 0, 1, \ldots : \sigma[i..] \models \phi_2 \wedge \forall 0 \leq j < i : \sigma[j..] \models \phi_1$

ϕ_2 holds for some suffix of σ and

ϕ_1 holds for all previous suffixes

Logic recap: basic vocabulary

- A formula ϕ is **valid** if it is "always" true, i.e., true in all models. In the case of LTL, it means that $\forall \sigma \in \Sigma^\omega : \sigma \models \phi$.
- A formula ϕ is **satisfiable** if it is "sometimes" true, i.e., true in some model. In the case of LTL, it means that $\exists \sigma \in \Sigma^\omega : \sigma \models \phi$. Otherwise, ϕ is **unsatisfiable**.
- A formula ϕ_1 is **stronger** than another formula ϕ_2 if the formula $\phi_1 \rightarrow \phi_2$ is valid.[2] Equivalently: $\forall \sigma \in \Sigma^\omega : \sigma \models \phi_1 \Rightarrow \sigma \models \phi_2$.
- A formula ϕ_1 is **weaker** than another formula ϕ_2 if ϕ_2 is stronger than ϕ_1. Equivalently: if $\phi_2 \rightarrow \phi_1$ is valid.
- Formulas ϕ_1, ϕ_2 are **equivalent** if the formula $\phi_1 \leftrightarrow \phi_2$ is valid, i.e., if both $\phi_1 \rightarrow \phi_2$ and $\phi_2 \rightarrow \phi_1$ are valid, i.e., both ϕ_1 is stronger than ϕ_2 and ϕ_2 is stronger than ϕ_1.
- A condition ϕ_1 is **necessary** for ϕ_2 if ϕ_2 implies (is stronger than) ϕ_1.
- A condition ϕ_1 is **sufficient** for ϕ_2 if ϕ_1 implies ϕ_2.

[2] So "stronger" really means "stronger or equivalent".

Interesting facts about LTL

- Can we express $\mathbf{G}p$ using only $\mathbf{F}$, p, and boolean operators?

Interesting facts about LTL

- Can we express $\mathbf{G}p$ using only $\mathbf{F}$, p, and boolean operators?

$$\mathbf{G}p \Leftrightarrow \neg\mathbf{F}\neg p$$

Interesting facts about LTL

- Can we express $\mathbf{G}p$ using only $\mathbf{F}$, p, and boolean operators?

$$\mathbf{G}p \Leftrightarrow \neg\mathbf{F}\neg p$$

- Vice versa, can we express $\mathbf{F}$ in terms of $\mathbf{G}$?

Interesting facts about LTL

- Can we express $\mathbf{G}p$ using only $\mathbf{F}$, p, and boolean operators?

$$\mathbf{G}p \Leftrightarrow \neg\mathbf{F}\neg p$$

- Vice versa, can we express $\mathbf{F}$ in terms of $\mathbf{G}$?

$$\mathbf{F}\phi \Leftrightarrow \neg\mathbf{G}\neg\phi$$

Interesting facts about LTL

- Can we express $\mathbf{G}p$ using only $\mathbf{F}$, p, and boolean operators?

$$\mathbf{G}p \Leftrightarrow \neg\mathbf{F}\neg p$$

- Vice versa, can we express $\mathbf{F}$ in terms of $\mathbf{G}$?

$$\mathbf{F}\phi \Leftrightarrow \neg\mathbf{G}\neg\phi$$

- Can we express $\mathbf{F}$ in terms of $\mathbf{U}$?

Interesting facts about LTL

- Can we express $\mathbf{G}p$ using only $\mathbf{F}$, p, and boolean operators?

$$\mathbf{G}p \Leftrightarrow \neg\mathbf{F}\neg p$$

- Vice versa, can we express $\mathbf{F}$ in terms of $\mathbf{G}$?

$$\mathbf{F}\phi \Leftrightarrow \neg\mathbf{G}\neg\phi$$

- Can we express $\mathbf{F}$ in terms of $\mathbf{U}$?

$$\mathbf{F}\phi \Leftrightarrow true\,\mathbf{U}\,\phi$$

What is "$true$" ?

Interesting facts about LTL

- Can we express $\mathbf{G}p$ using only $\mathbf{F}$, p, and boolean operators?

$$\mathbf{G}p \Leftrightarrow \neg\mathbf{F}\neg p$$

- Vice versa, can we express $\mathbf{F}$ in terms of $\mathbf{G}$?

$$\mathbf{F}\phi \Leftrightarrow \neg\mathbf{G}\neg\phi$$

- Can we express $\mathbf{F}$ in terms of $\mathbf{U}$?

$$\mathbf{F}\phi \Leftrightarrow true\,\mathbf{U}\,\phi$$

What is "$true$" ? Can be defined as a primitive formula, or as $p \vee \neg p$.

Interesting facts about LTL

- Can we express $\mathbf{G}p$ using only $\mathbf{F}$, p, and boolean operators?

$$\mathbf{G}p \Leftrightarrow \neg\mathbf{F}\neg p$$

- Vice versa, can we express $\mathbf{F}$ in terms of $\mathbf{G}$?

$$\mathbf{F}\phi \Leftrightarrow \neg\mathbf{G}\neg\phi$$

- Can we express $\mathbf{F}$ in terms of $\mathbf{U}$?

$$\mathbf{F}\phi \Leftrightarrow true\,\mathbf{U}\,\phi$$

What is "$true$" ? Can be defined as a primitive formula, or as $p \vee \neg p$.

- Can we express $\mathbf{X}$ in terms of $\mathbf{G}$, $\mathbf{F}$, $\mathbf{U}$?

Interesting facts about LTL

- Can we express $\mathbf{G}p$ using only $\mathbf{F}$, p, and boolean operators?

$$\mathbf{G}p \Leftrightarrow \neg\mathbf{F}\neg p$$

- Vice versa, can we express $\mathbf{F}$ in terms of $\mathbf{G}$?

$$\mathbf{F}\phi \Leftrightarrow \neg\mathbf{G}\neg\phi$$

- Can we express $\mathbf{F}$ in terms of $\mathbf{U}$?

$$\mathbf{F}\phi \Leftrightarrow true\,\mathbf{U}\,\phi$$

What is "$true$" ? Can be defined as a primitive formula, or as $p \vee \neg p$.

- Can we express $\mathbf{X}$ in terms of $\mathbf{G}$, $\mathbf{F}$, $\mathbf{U}$? No!

LTL – more examples

Let's try to express the following requirements in LTL:

1. *No more than one processor (in a 2-processor system) shall have a cache line in write mode.*

LTL – more examples

Let's try to express the following requirements in LTL:

1. *No more than one processor (in a 2-processor system) shall have a cache line in write mode.*
   Let AP $= \{p_1, p_2\}$, with p_i meaning "processor i has the cache line in write mode."

$$\mathbf{G}\neg(p_1 \wedge p_2)$$

LTL – more examples

Let's try to express the following requirements in LTL:

1. *No more than one processor (in a 2-processor system) shall have a cache line in write mode.*
   Let AP $= \{p_1, p_2\}$, with p_i meaning "processor i has the cache line in write mode."
   $$\mathbf{G}\neg(p_1 \wedge p_2)$$
2. *The grant signal must be asserted some time after the request signal is asserted.*

LTL – more examples

Let's try to express the following requirements in LTL:

1. *No more than one processor (in a 2-processor system) shall have a cache line in write mode.*
   Let AP $= \{p_1, p_2\}$, with p_i meaning "processor i has the cache line in write mode."
   $$\mathbf{G}\neg(p_1 \wedge p_2)$$
2. *The grant signal must be asserted some time after the request signal is asserted.*
   Let AP $= \{r, g\}$, with r meaning "request signal is asserted" and g meaning "grant signal is asserted."
   $$\mathbf{G}(r \rightarrow \mathbf{F}g)$$

LTL – more examples

Let's try to express the following requirements in LTL:

1. *No more than one processor (in a 2-processor system) shall have a cache line in write mode.*
   Let AP $= \{p_1, p_2\}$, with p_i meaning "processor i has the cache line in write mode."
   $$\mathbf{G}\neg(p_1 \wedge p_2)$$
2. *The grant signal must be asserted some time after the request signal is asserted.*
   Let AP $= \{r, g\}$, with r meaning "request signal is asserted" and g meaning "grant signal is asserted."
   $$\mathbf{G}(r \rightarrow \mathbf{F}g)$$
3. *A request must receive an acknowledgement, and the request should stay asserted until the acknowledgment is received.*

LTL – more examples

Let's try to express the following requirements in LTL:

1. *No more than one processor (in a 2-processor system) shall have a cache line in write mode.*
   Let AP $= \{p_1, p_2\}$, with p_i meaning "processor i has the cache line in write mode."
   $$\mathbf{G}\neg(p_1 \wedge p_2)$$
2. *The grant signal must be asserted some time after the request signal is asserted.*
   Let AP $= \{r, g\}$, with r meaning "request signal is asserted" and g meaning "grant signal is asserted."
   $$\mathbf{G}(r \rightarrow \mathbf{F}g)$$
3. *A request must receive an acknowledgement, and the request should stay asserted until the acknowledgment is received.*
   Let AP $= \{r, a\}$, with r request and a acknowledgment.
   $$\mathbf{G}(r \rightarrow (r\,\mathbf{U}\,a))$$

LTL in the industry

Several industrial standard languages based on LTL, e.g.,

- PSL (*Property Specification Language*), an IEEE standard.
- PSL/Sugar (IBM variant).

Example properties written in PSL/Sugar:

```
assert always req -> next (ack until grant);
```

$$\mathbf{G}(r \rightarrow \mathbf{X}(a \,\mathbf{U}\, g))$$

```
assert always req -> next[3] (grant);
```

$$\mathbf{G}(r \rightarrow \mathbf{XXX}g)$$

LTL expressiveness

- Are there properties that we **cannot** write in LTL?

LTL expressiveness

- Are there properties that we **cannot** write in LTL?
- There must be, because of cardinality arguments: Σ^ω is uncountable, so its powerset is even more uncountable!
  But the set of all LTL formulas is countable – why?

LTL expressiveness

- Are there properties that we **cannot** write in LTL?
- There must be, because of cardinality arguments: Σ^ω is uncountable, so its powerset is even more uncountable!
  But the set of all LTL formulas is countable – why? I can enumerate all formulas of length 1 (there's finitely many of them), then all those of length 2, then length 3, etc.

- You will explore this a bit more in the next **homework**.
- We will also return to it when we talk about Büchi automata.

THE MODEL-CHECKING PROBLEM FOR LTL

The verification problem

Specification (the "what") = the property that we want the system to have

Implementation (the "how") = the system that we want to verify

The **verification problem**: does the implementation satisfy the specification?

The verification problem for LTL = LTL model checking

The **LTL model checking problem**:

- Given:
  - Implementation: state machine or transition system M
  - Specification: LTL formula ϕ
- Check whether **every** trace of M satisfies ϕ:

$$\forall \sigma \in Traces(M) : \sigma \models \phi$$

We write this as:

$$M \models \phi$$

The verification problem for LTL = LTL model checking

The **LTL model checking problem**:

- Given:
  - Implementation: state machine or transition system M
  - Specification: LTL formula ϕ
- Check whether **every** trace of M satisfies ϕ:

$$\forall \sigma \in Traces(M) : \sigma \models \phi$$

We write this as:

$$M \models \phi$$

- In case $M \not\models \phi$ we would also like to get a **counter-example**: most model-checkers provide that
- In case $M \models \phi$ we might want to get a "proof": this is typically **not** provided (what would that proof look like?)

Parenthesis: how to be precise about the problem you are solving

Thesis: **Every CS problem can be cast in this form**:

- Given: X
- Find: Y
- Such that: Z

Parenthesis: how to be precise about the problem you are solving

Thesis: **Every CS problem can be cast in this form**:

- Given: X
- Find: Y
- Such that: Z

Make sure you follow that format when you present your papers: you should be able to explain to us what problem each paper is trying to solve in the above terms!

Understanding **what** problem is being solved is much more important than understanding **how** it is being solved!

Traces of a transition system

An infinite path in a Kripke structure $(\text{AP}, S, S_0, L, R)$ is an infinite sequence of states linked by transitions:

$$s_0, s_1, s_2, \cdots$$

such that $s_0 \in S_0$ and $\forall i : (s_i, s_{i+1}) \in R$.

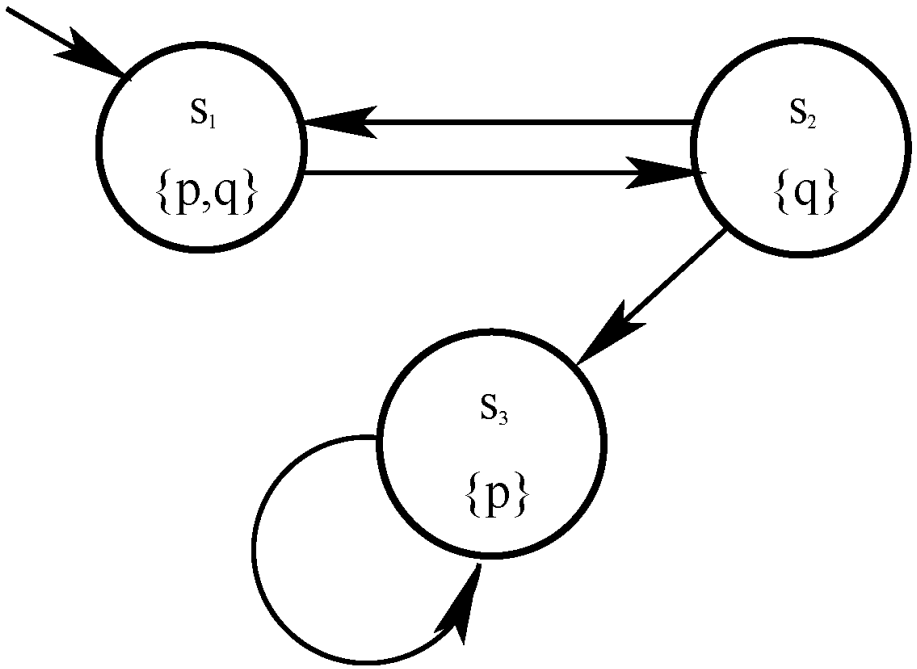
The corresponding observable **trace** σ is the corresponding infinite sequence of sets of atomic propositions:

$$\sigma = L(s_0), L(s_1), L(s_2), \cdots$$

If M is a Kripke structure then $Traces(M)$ is the set of all observable traces over all infinite paths of M.

Example

List some of the traces of the following transition system:



How many traces are there in total?

Traces of a state machine

An infinite run of a Mealy machine $(I, O, S, s_0, \delta, \lambda)$ is an 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 $\forall i : x_i \in I, y_i \in O$, $\forall i : s_{i+1} = \delta(s_i, x_i)$, and $\forall i : y_i = \lambda(s_i, x_i)$.

Traces of a state machine

An infinite run of a Mealy machine $(I, O, S, s_0, \delta, \lambda)$ is an 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 $\forall i : x_i \in I, y_i \in O$, $\forall i : s_{i+1} = \delta(s_i, x_i)$, and $\forall i : y_i = \lambda(s_i, x_i)$.

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

$$\sigma = \{x_0, y_0\}, \{x_1, y_1\}, \{x_2, y_2\}, \cdots$$

where we assume AP $= I \cup O$ and interpret x_i as the atomic proposition "the value of the input is x_i", and similarly for y_i.

Traces of a state machine

An infinite run of a Mealy machine $(I, O, S, s_0, \delta, \lambda)$ is an 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 $\forall i : x_i \in I, y_i \in O$, $\forall i : s_{i+1} = \delta(s_i, x_i)$, and $\forall i : y_i = \lambda(s_i, x_i)$.

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

$$\sigma = \{x_0, y_0\}, \{x_1, y_1\}, \{x_2, y_2\}, \cdots$$

where we assume AP $= I \cup O$ and interpret x_i as the atomic proposition "the value of the input is x_i", and similarly for y_i.

(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.)

Examples

Let's check the following LTL formulas on the previous Kripke structure:

$$\mathbf{G}p$$

$$\mathbf{F}p$$

$$\mathbf{GF}p$$

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

$$p \,\mathbf{U}\, q$$

INVARIANTS IN LTL

Recall: Invariants

- In a transition system, an invariant is a superset of the set of reachable states.
- So, an invariant is a "property" that all reachable states must have, or in other words, a condition/constraint that all reachable states must satisfy.
- How can we express that something is an invariant in LTL?

Invariants in LTL

- Let ϕ be an LTL formula without temporal operators, i.e., ϕ can contain only atomic propositions and Boolean operators ($\wedge, \vee, \neg, ...$), but no $\mathbf{G}, \mathbf{F}, \mathbf{X}, \mathbf{U}$.
  Such a ϕ is called a **state formula**.

- Then, a linear-time property is an invariant if it can be expressed as an LTL formula $\mathbf{G}\phi$, where ϕ is a state formula.

LTL in Spin

LTL in Spin: a toy example

```
bool a, b                    // initialized to false

active proctype system()
{
  do
  :: a = 1; b = 1; a = 0; b = 0
  od
}

// ltl p1 { ([] <> a) }
// ltl p2 { ([] <> (a && b)) }
// ltl p3 { [] (a -> (<> b)) }
// ltl p4 { always (a -> (eventually b)) }
// ltl p5 { always (a -> (next b)) }
ltl p5 { always (a -> (next a)) }
```

More Spin examples

See, for instance, `http://spinroot.com/spin/Man/1_Exercises.html`

Bibliography

Baier, C. and Katoen, J.-P. (2008).

Principles of Model Checking.
MIT Press.

Clarke, E., Grumberg, O., and Peled, D. (2000).

Model Checking.
MIT Press.

Huth, M. and Ryan, M. (2004).

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

Pnueli, A. (1981).

A temporal logic of concurrent programs.
Theoretical Computer Science, 13:45–60.