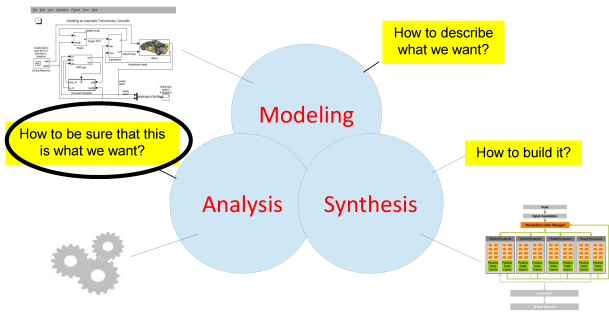
EE 244: Fundamental Algorithms for System Modeling, Analysis, and Optimization
Fall 2016

Temporal logic

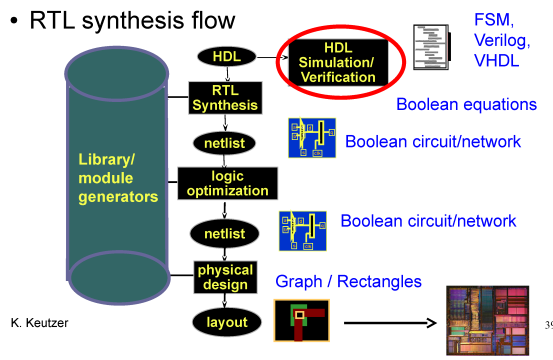
Stavros Tripakis
University of California, Berkeley

Recall one of our goals: verification

Model-based design



Example of a successful model-based design flow



We have designed a system.

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

But what does "correct" mean?

We need to **specify** correctness ⇒ 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.

Temporal logic

a **formal specification language**

=

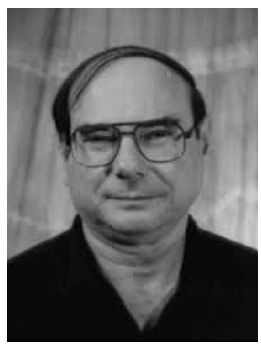
a way to state properties of our system mathematically
(precisely and unambiguously!)
(as opposed to natural language)

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

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, ..., properties
- We will look at
  - **LTL** (*linear temporal logic*) for linear-time properties.
  - **CTL** (*computation tree logic*) for branching-time properties.

LTL (Linear Temporal Logic) – Syntax

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

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

$\phi_1 \wedge \phi_2$: 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] This is *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.

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

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

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

LTL – Syntax

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

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:

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

What do these traces mean? p holds at step i iff $p \in P_i$.
Where do these traces come from? From state machines or **transition systems** (we'll see later).

LTL – Semantics: Intuition

Given LTL formula ϕ and infinite trace

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

we say that σ **satisfies** ϕ, written

$$\sigma \models \phi$$

when

formula	meaning
p	p holds now (at first step), i.e., $p \in P_0$
$\phi_1 \wedge \phi_2$	σ satisfies both ϕ_1 and ϕ_2
$\neg\phi_1$	σ does not satisfy ϕ_1
$\mathbf{G}\phi_1$	every suffix $P_i, P_{i+1}, \cdots$ of σ satisfies ϕ_1
$\mathbf{F}\phi_1$	some suffix of σ satisfies ϕ_1
$\mathbf{X}\phi_1$	the suffix $P_1\, P_2\, \cdots$ satisfies ϕ_1
$\phi_1\, \mathbf{U}\, \phi_2$	ϕ_2 holds for the suffix starting at position i, for some $i \geq 0$, and ϕ_1 holds for all suffixes prior to that

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:

$$\begin{array}{lll}
\sigma \models p & \text{iff} & p \in P_0 \quad p \text{ holds at the first (current) step} \\
\sigma \models \phi_1 \wedge \phi_2 & \text{iff} & \sigma \models \phi_1 \text{ and } \sigma \models \phi_2 \\
\sigma \models \neg\phi & \text{iff} & \sigma \not\models \phi \\
\sigma \models \mathbf{G}\phi & \text{iff} & \forall i = 0, 1, \ldots : \sigma[i..] \models \phi \quad \phi \text{ holds for every suffix of } \sigma \\
\sigma \models \mathbf{F}\phi & \text{iff} & \exists i = 0, 1, \ldots : \sigma[i..] \models \phi \quad \phi \text{ holds for some suffix of } \sigma \\
\sigma \models \mathbf{X}\phi & \text{iff} & \sigma[1..] \models \phi \quad \phi \text{ holds for the suffix starting at the next step} \\
\sigma \models \phi_1 \,\mathbf{U}\, \phi_2 & \text{iff} & \exists i = 0, 1, \ldots : \sigma[i..] \models \phi_2 \wedge \\
 & & \quad \forall 0 \leq j < i : \sigma[j..] \models \phi_1 \\
 & & \phi_2 \text{ holds for some suffix of } \sigma \text{ and} \\
 & & \phi_1 \text{ holds for all previous suffixes}
\end{array}$$

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.*
   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);
```

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

SAFETY and LIVENESS

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, or something else?

- $\mathbf{G}p$: safety.
- $\mathbf{F}p$: liveness.
- $\mathbf{X}p$: safety.
- $p\,\mathbf{U}\,q$: a "mix" of both!
- $\mathbf{GF}p$: liveness.
- $\mathbf{G}(p \rightarrow \mathbf{F}q)$: liveness.
- $\mathbf{G}(p \rightarrow \mathbf{X}q)$: safety.

Safety and Liveness – Formally

Let AP be a set of atomic propositions.

- 2^{AP} is the powerset (set of all subsets) of AP.
- $(2^{\text{AP}})^*$ is the set of all finite sequences over AP.
- $(2^{\text{AP}})^\omega$ is the set of all infinite sequences ("traces") over AP.

What is a *property*, formally?

A **property** L is a set of traces: $L \subseteq (2^{\text{AP}})^\omega$.

Examples:

- $L = (2^{\text{AP}})^\omega$: L holds on all traces (every trace is in L, i.e., every trace satisfies property L).
- $L = \emptyset$: no trace satisfies L.
- $L =$ the set of all traces satisfying $\mathbf{GF}p$.
- $L =$ the set of all traces such that p holds at every odd step in the trace.

Safety and Liveness – Formally

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

For a trace $\sigma = \alpha_1\alpha_2\alpha_3\cdots$, and length $k \in \mathbb{N}$, we denote by $\sigma[1..k]$ the finite prefix $\alpha_1 \cdots \alpha_k$ of σ. When $k = 0$ we get the **empty** prefix.

- L is a safety property if

$$\forall \sigma \notin L : \exists k \in \mathbb{N} : \forall \rho \in (2^{\text{AP}})^\omega : \sigma[1..k] \cdot \rho \notin L$$

  i.e., *for any σ 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.*

- L is a liveness property if

$$\forall \sigma \in (2^{\text{AP}})^* : \exists \rho \in (2^{\text{AP}})^\omega : \sigma \cdot \rho \in L$$

  i.e., *every finite trace can be extended, by appending a* **good suffix**, *into an infinite trace which satisfies the liveness property.*

Safety and Liveness – Formally

Theorem ([Alpern and Schneider, 1985])

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

THE MODEL-CHECKING PROBLEM

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

Implementation: state machine or transition system

Specification: LTL formula

The **LTL model checking problem**: does a given system M satisfy a given LTL formula ϕ?

Every execution trace of M must satisfy ϕ.

We write this as:

$$M \models \phi$$

(read "M satisfies ϕ").

Transition Systems

An even more basic model than automata and state machines:

transition system = states + transitions (+ labels)

Possibly infinite sets of states/transitions.

Transitions typically non-deterministic.

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

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

Labeled Transition Systems

An LTS is a tuple:

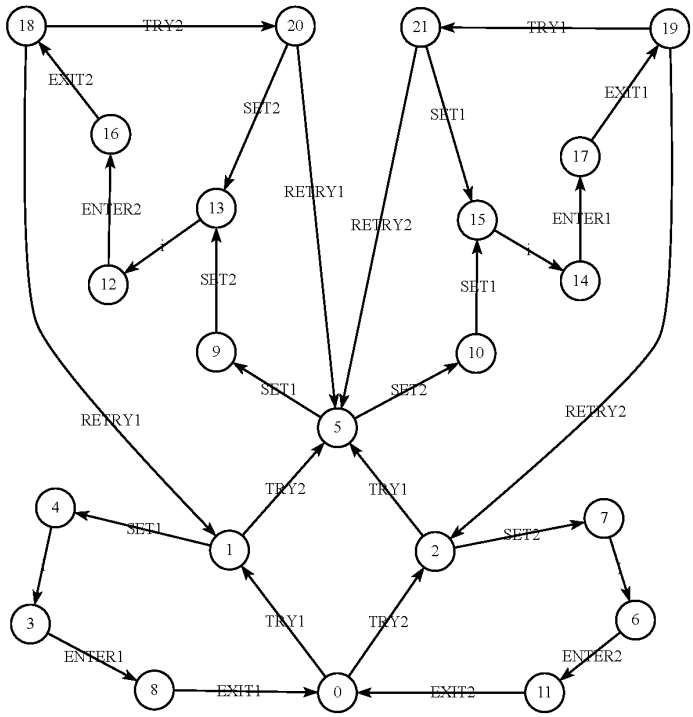
$$(\Sigma, S, S_0, R)$$

- Σ: set of labels (modeling events, actions, ...)
- S: set of states (perhaps infinite)
- $S_0 \subseteq S$: set of initial states
- R: transition relation

$$R \subseteq S \times (\Sigma \cup \{\epsilon\}) \times S$$

ϵ (sometimes τ): internal, unobservable action (used in composition, simulation/bisimulation equivalences, ...).

Example: LTS



In a LTS the labels are on the transitions.

Kripke Structures

A Kripke structure is a tuple:

$$(\mathsf{AP}, S, S_0, L, R)$$

- AP: set of atomic propositions (modeling state properties)
- S: set of states (perhaps infinite)
- $S_0 \subseteq S$: set of initial states
- L: labeling function on states

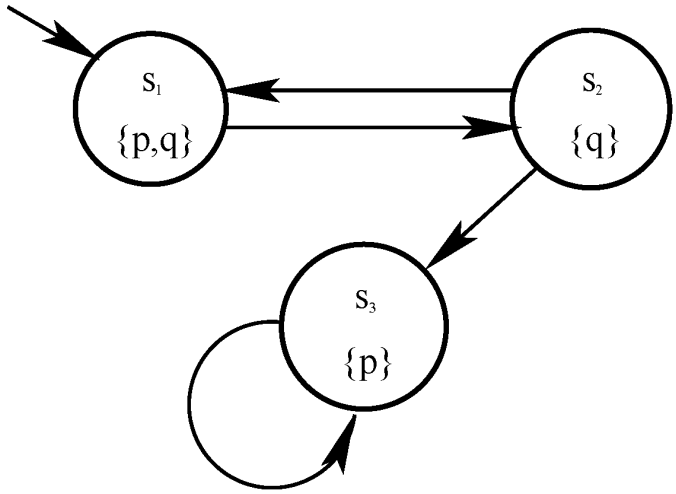
$$L : S \to 2^{\mathsf{AP}}$$

2^{AP}: the **powerset** (set of all subsets) of AP.
For $p \in \mathsf{AP}$ and $s \in S$: "s has property p" iff $p \in L(s)$.

- R: transition relation

$$R \subseteq S \times S$$

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).

LTS vs. Kripke structures

In LTS, the labels are on the transitions.

In Kripke structures, the labels are on the states.

Homework

1. Can we translate a Moore machine to an "equivalent" Mealy machine? (and what does equivalent mean?) And vice-versa?
2. Can we translate a KS to an "equivalent" LTS? (and what does equivalent mean?) And vice-versa?

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:

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

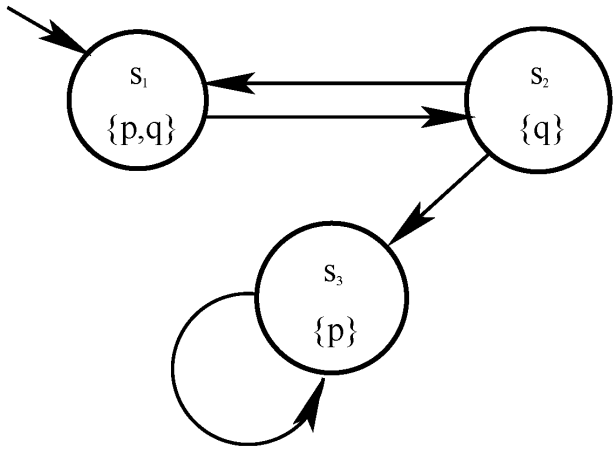
such that $s_0 \in S_0$ and $\forall i : (s_i, s_{i+1}) \in R$.

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

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

Example

List some of the traces of the following transition system:



How many traces are there in total?

Execution traces of a state machine

Recall: an infinite run of a Mealy machine $(I, O, S, s_0, \delta, \lambda)$ is an infinite sequence of states / transitions:

$$s_0 \overset{x_0/y_0}{\longrightarrow} s_1 \overset{x_1/y_1}{\longrightarrow} s_2 \overset{x_2/y_2}{\longrightarrow} s_3 \cdots$$

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 proposition "the value of the input is x_i" and y_i similarly.

(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.)

Back to LTL: examples

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

$$\mathbf{G}p$$

$$\mathbf{F}p$$

$$\mathbf{GF}p$$

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

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

BRANCHING-TIME PROPERTIES

Linear-Time vs. Branching-Time Properties

So far we have been talking about properties of **linear** behaviors (sequences, traces).

But some properties are not linear, e.g.:

> *"it is possible to recover from any fault"*

or

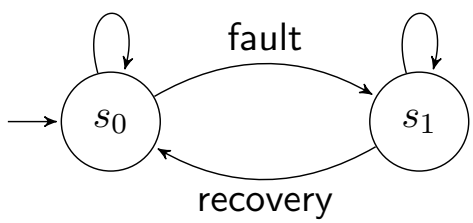
> *"there exists a way to get back to the initial state from any reachable state"*

Linear-Time vs. Branching-Time Properties

> *"it is possible to recover from any fault"*

Based on *one* (linear) behavior alone,[2] we cannot conclude whether our system satisfies the property.

E.g., the following system satisfies the property, although it contains a behavior that stays forever in state s_1:



[2] if we had *all* linear behaviors of a system, we could in principle reconstruct its branching behavior as well

Linear-Time vs. Branching-Time Temporal Logics

Linear-time: the "solutions" (models) of a temporal logic formula are infinite **sequences** (traces).

Branching-time: the "solutions" (models) of a temporal logic formula are infinite **trees**.

- Hence the name "Computation Tree Logic" for CTL.

Branching-Time Temporal Logic: CTL

We will simplify and define the semantics of CTL directly on states of a transition system (Kripke structure).

CTL (Computation Tree Logic) – Syntax

CTL formulas are defined by the following grammar:

$$\begin{aligned}\phi \quad ::= \quad & p \mid q \mid ..., \text{ where } p, q, ... \in \text{AP} \\ & \mid \phi_1 \wedge \phi_2 \mid \neg\phi_1 \\ & \mid \mathbf{EG}\phi_1 \mid \mathbf{AG}\phi_1 \\ & \mid \mathbf{EF}\phi_1 \mid \mathbf{AF}\phi_1 \\ & \mid \mathbf{EX}\phi_1 \mid \mathbf{AX}\phi_1 \\ & \mid \mathbf{E}(\phi_1 \,\mathbf{U}\, \phi_2) \mid \mathbf{A}(\phi_1 \,\mathbf{U}\, \phi_2)\end{aligned}$$

$\mathbf{E}$ ("there exists a path") and $\mathbf{A}$ ("for all paths") are called **path quantifiers**.

CTL (Computation Tree Logic) – Syntax

Examples of CTL formulas:

$$\mathbf{AG}p$$

$$\mathbf{EF}q$$

$$\mathbf{AGEF}(p \rightarrow q)$$

Syntactically incorrect CTL formulas:

$$\mathbf{G}p, \quad \mathbf{AGF}p, \quad (\mathbf{AG}p) \wedge \mathbf{F}q, \quad \mathbf{AEG}p, \quad \mathbf{A}p$$

Alternative notation: $\forall\Box p$, $\exists\Diamond q$, $\forall(p \,\mathbf{U}\, q)$, etc.

CTL – Semantics: Intuition

Let s be a state of the Kripke structure.

Then s satisfies the CTL formula $\mathbf{EG}\phi$, written

$$s \models \mathbf{EG}\phi$$

iff **there exists** a trace σ starting from s and satisfying $\mathbf{G}\phi$.

CTL – Semantics: Intuition

$$s \models \mathbf{AG}\phi$$

iff **every** trace σ starting from s satisfies $\mathbf{G}\phi$.

Examples

Let's construct transition systems (Kripke structures) satisfying or violating the following CTL formulas:

$$\mathbf{AG}p$$

$$\mathbf{AF}p$$

$$\mathbf{EG}p$$

$$\mathbf{EF}p$$

Facts about CTL

Quiz: do we need $\mathbf{EF}\phi$? Can we express it in terms of other CTL modalities?

CTL – Formal Semantics

Let $(\text{AP}, S, S_0, L, R)$ be a Kripke structure and let $s \in S$.

A *trace starting from* s is an infinite sequence $\sigma = \sigma_0, \sigma_1, \cdots$, such that there is an infinite path $s = s_0, s_1, \cdots$ starting from s, and $\sigma_i = L(s_i)$ for all i.

Satisfaction relation for CTL:

$$
\begin{array}{lll}
s \models p & \text{iff} & p \in L(s) \\
s \models \phi_1 \wedge \phi_2 & \text{iff} & s \models \phi_1 \text{ and } s \models \phi_2 \\
s \models \neg\phi & \text{iff} & s \not\models \phi \\
s \models \mathbf{EG}\phi & \text{iff} & \exists \text{trace } \sigma \text{ starting from } s : \sigma \models_{LTL} \mathbf{G}\phi \\
s \models \mathbf{AG}\phi & \text{iff} & \forall \text{traces } \sigma \text{ starting from } s : \sigma \models_{LTL} \mathbf{G}\phi \\
s \models \mathbf{EX}\phi & \text{iff} & \exists \text{trace } \sigma \text{ starting from } s : \sigma \models_{LTL} \mathbf{X}\phi \\
s \models \mathbf{E}(\phi_1 \,\mathbf{U}\, \phi_2) & \text{iff} & \exists \text{trace } \sigma \text{ starting from } s : \sigma \models_{LTL} \phi_1 \,\mathbf{U}\, \phi_2 \\
\ldots &&
\end{array}
$$

(Here $\sigma \models_{LTL} \mathbf{G}\phi$ means that the trace σ satisfies $\mathbf{G}\phi$ in the LTL sense. However, strictly speaking $\models_{LTL}$ is not the LTL satisfaction relation, because ϕ is not an LTL formula.)

The verification problem for CTL: CTL model checking

The **CTL model checking problem**: does a given transition system (Kripke structure) M satisfy a given CTL formula ϕ?

Let $M = (\text{AP}, S, S_0, L, R)$.
S_0 is a set, so M generally has many initial states.

We want **every** initial state of M to satisfy ϕ:

$$\forall s \in S_0 : s \models \phi$$

We write this as:

$$M \models \phi$$

(same notation as in LTL model-checking, but here ϕ is a CTL formula).

CTL – Examples

How to express these properties in CTL?

"p holds at all reachable states" $\mathbf{AG}p$

"there exists a way to get back to the initial state from any reachable state" $\mathbf{AG}\,\mathbf{EF}\,\mathit{init}$

"p is inevitable" $\mathbf{AF}\,p$

"p is possible" $\mathbf{EF}\,p$

How would you express the last two in LTL?

Bibliography

Alpern, B. and Schneider, F. B. (1985).
Defining liveness.
Information Processing Letters, 21(4):181 – 185.

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.

Emerson, E. A. (1990).
Handbook of theoretical computer science (vol. b).
chapter Temporal and modal logic, pages 995–1072. MIT Press.

Huth, M. and Ryan, M. (2004).
Logic in Computer Science: Modelling and Reasoning about Systems.
Cambridge University Press.

Manna, Z. and Pnueli, A. (1991).
The Temporal Logic of Reactive and Concurrent Systems: Specification.
Springer-Verlag, New York.

Manna, Z. and Pnueli, A. (1995).
Temporal Verification of Reactive Systems: Safety.
Springer-Verlag, New York.

Pnueli, A. (1981).
A temporal logic of concurrent programs.
Theoretical Computer Science, 13:45–60.