Verification Using Temporal Logic

Sources:


Temporal Logic ... 

... a language for describing time-varying properties of systems.

... originally developed as a branch of philosophical logic.

... is a modal logic (logic of possibility and necessity).

... brought to attention of computer scientists by A. Pnueli in 1977.
How is the verification question $S \text{ sat } R$ formulated in the temporal-logic setting?

- Temporal logic used to define requirements $Spec$.
- $Sys? \text{ sat}$?
Kripke structures used as system models in temporal logic.
Defining Kripke Structures

Mathematically, a Kripke structure has form $\langle S, A, R, \ell, s_I \rangle$, where:

- $S$ is a set of states;
- $A$ is a set of atomic propositions;
- $R \subseteq S \times S$ is the transition relation;
- $\ell : S \rightarrow 2^A$ is the labeling; and
- $s_I \in S$ is the initial state.
Another Kripke Structure Example

ready1, ready2

in1, ready2

out1, ready2

out1, in2

out1, out2

ready1, in2

ready1, out2

in1, out2

©2015 Rance Cleaveland. All rights reserved.
Where Do Kripke Structures Come From?

Kripke structures are usually not specified directly by users; they are “compiled” from higher-level specifications using (small-step) operational semantics of programming / modeling notation.

- States: assignments of variables to values
- Labeling function given as relation on states / atomic propositions
- Transitions: execution steps defined by semantics
Example: UNITY

- A modeling notation for concurrent systems
- Emblematic of notations used by model-checking tools
Example UNITY Model

program mutex
declare
    in: array [0..1] of int
    lock: int
initially
    in[0] = in[1] = lock = 0
assign
    P0enter: lock = 0 → lock, in[0] = 1, 1
    P1enter: lock = 0 → lock, in[1] = 1, 1
    P0exit: in[0] = 1 → lock, in[0] = 0, 0
    P1exit: in[1] = 1 → lock, in[0] = 0, 0
end
UNITY in Detail

- Programs consist of declare, initially, assign sections (among others)

- declare section defines variables

- initially section initializes variables

- assign section contains guarded commands
  - Guarded command has form $G \rightarrow S$
  - $G$ is boolean condition, $S$ is Hoare program

Intended semantics: at each step, command with true guard is selected, executed atomically. Program terminates when all guards are false.
Defining Temporal Logic

Now that we know what systems (Sys) are, what about requirements (Spec)?

Idea: Formulas of temporal logic constitute Spec.

⇒ but there are a number of variants of temporal logic.

An important categorization: linear time vs. branching time. The difference: how time is modeled! (More on this later.)

We’ll start with linear-time.
Linear-Time Temporal Logic (LTL)

Used to specify properties of *sequences*.

Given: set \((a, b \in \mathcal{A})\) of atomic propositions. Formulas generated as follows.

\[
\phi \; ::= \; a \\
| \; \neg \phi \\
| \; \phi \lor \phi \\
| \; X \phi \quad \text{“next”}
| \; \phi \; U \; \phi \quad \text{“until”}
\]

\(\Phi_{\text{LTL}}\) is the set of all LTL formulas.
Formal semantics given as \( \models \subseteq (2^A)^\omega \times \Phi_{\text{LTL}} \).

- \( A \): set of atomic propositions
- \( 2^A \): set of all subsets of \( A \).
- \( (2^A)^\omega \): set of infinite sequences of \( 2^A \).

That is, formulas interpreted with respect to sequences of sets of atomic propositions.

\[ \pi \models \phi: \text{“sequence } \pi \text{ has property } \phi \text{”} \]
Notation

If $\pi = A_0 A_1 \ldots \in (2^A)^\omega$ then

- $\pi[i] = A_i \in 2^A$
- $\pi[i..] = A_i A_{i+1} \ldots \in (2^A)^\omega$
LTL Semantics (cont.)

\( \pi \models a \) if \( a \in \pi[0] \).

\( \pi \models \neg \phi \) if \( \pi \not\models \phi \).

\( \pi \models \phi_1 \lor \phi_2 \) if \( \pi \models \phi_1 \) or \( \pi \models \phi_2 \).

\( \pi \models X\phi \) if \( \pi[1..] \models \phi \).

\( \pi \models \phi_1 \mathbf{U} \phi_2 \) if

\[ \exists i \geq 0. \ \pi[i..] \models \phi_2 \text{ and } \forall j < i. \ \pi[j..] \models \phi_1 \]
Derived Operators

\[
\begin{align*}
    \text{tt} & \triangleq a \lor \neg a \\
    \text{ff} & \triangleq \neg \text{tt} \\
    \phi_1 \land \phi_2 & \triangleq \neg (\neg \phi_1 \lor (\neg \phi_2)) \quad \text{conjunction} \\
    \phi_1 \Rightarrow \phi_2 & \triangleq (\neg \phi_1) \lor \phi_2 \quad \text{implication}
\end{align*}
\]
More Derived Operators

Eventually  \( F\phi \)
\[ tt \ U \phi \]

Always  \( G\phi \)
\[ \neg F\neg \phi \]
Still More Derived Operators

Release \[ \phi_1 R \phi_2 \]
\[ \neg((\neg \phi_1) U (\neg \phi_2)) \]

Weak Until \[ \phi_1 W \phi_2 \]
\[ \phi_2 R (\phi_1 \lor \phi_2) \]

What does \[ \neg((\neg \phi_2) U (\neg \phi_1 \land \neg \phi_2)) \] mean?
Writing Requirements in LTL

LTL can be used to specify different properties of “execution sequences”.

- **Safety**: "Nothing bad ever happens."
  
  $G \neg \text{bad}$

- **Liveness**: "Something good eventually happens."
  
  $F \text{good}$
More Kinds of Properties

Responsiveness  “Every request is eventually serviced.”
\[ G \left( req \Rightarrow F \text{ serviced} \right) \]

Reactivity
\[ (GF \text{ enabled}) \Rightarrow (GF \text{ executed}) \]

Partition
\[ \phi_1 \cup (\phi_2 \cup \phi_3) \]
Example: Specifying Mutual Exclusion

\[ A = \{in_i, out_i, trying_i\}, i = 1, 2. \]

**Mutual Exclusion**

\[ G \neg (in_1 \land in_2) \]

**Starvation-Freedom for Process 1**

\[ G (trying_1 \Rightarrow F in_1) \]

**1-Bounded Overtaking for Process 1**

\[ G (trying_1 \Rightarrow (trying_2 \cup in_2 \cup in_1)) \]
But What About System Properties?

So far we have used LTL to define properties of “executions”.

How do we use it to define properties/requirements of systems?

**Idea** Define a system to satisfy a formula if “all its executions” do.

How to make this precise?
Formalizing sat

Let \( M = \langle S, A, R, \ell, s_I \rangle \) be a Kripke structure.

- \( \pi = s_0 s_1 \ldots \in S^\omega \) is an execution of \( M \) if:
  - \( s_0 = s_I \)
  - For all \( i \geq 0 \), \( \langle s_i, s_{i+1} \rangle \in R \) or for all \( s \in S \), \( \langle s_i, s \rangle \notin R \) and \( s_{i+1} = s_i \).

\( E(M) \): set of all executions of \( M \).

- \( \ell(s_0 s_1 \ldots) = \ell(s_0) \ell(s_1) \ldots \)

- \( M \) sat \( \phi \) if for all \( \pi \in E(M) \), \( \ell(\pi) \models \phi \).
What We Have...

... a verification framework based on LTL!

- Systems: Kripke structures
- Requirements: LTL formulas
- sat: sat

What's next?

1. Other temporal logics.
2. How to show $M \text{ sat } \phi$. 
Branching-Time Temporal Logic

Recall:

- Two kinds of temporal logic: linear-time and branching-time
- Linear-time: models are sequences
- Branching-time: models are “trees” (alternatively, states in Kripke structures)

How do we define a branching-time temporal logic?
The CTL* approach: add “path quantifiers” to LTL!

$E \phi$: satisfied by a state if there exists a path from the state and satisfying $\phi$.

$E$ is a *path quantifier*.
Syntax of CTL\(^*\)

Defines state formulas ...

\[
\sigma \ ::= \ a \\
| \neg \sigma \\
| \sigma \lor \sigma \\
| E\phi
\]

\(\Sigma_{CTL^*}\): set of all CTL\(^*\) (state) formulas.
... and path formulas.

\[
\phi ::= \sigma \\
| \neg \phi \\
| \phi \lor \phi \\
| X\phi \\
| \phi U \phi
\]

\(\Phi_{\text{CTL}^*}\): set of all CTL* path formulas.
Defining Semantics of CTL* ... given with respect to Kripke structure $M = \langle S, A, R, \ell, s_I \rangle$ as relation

$$\models_M \subseteq (S \times \Sigma_{\text{CTL}*}) \cup (S^\omega \times \Phi_{\text{CTL}*})$$

... i.e. states related to state formulas and paths to path formulas.

Fix $M$ in what follows.

**Notation** $E(M, s) \subseteq S^\omega$: execution paths emanating from $s$. $\pi \in E(M, s)$ if:

- $\pi[0] = s$
- For all $i \geq 0$, $\langle \pi[i], \pi[i + 1] \rangle \in R$ or for all $s' \in S$, $\langle \pi[i], s' \rangle \not\in R$ and $\pi[i + 1] = \pi[i]$. 
Semantics of State Formulas

\[ s \models_M a \quad \text{if} \quad a \in \ell(s). \]

\[ s \models_M \neg \sigma \quad \text{if} \quad s \not\models_M \sigma. \]

\[ s \models_M \sigma_1 \lor \sigma_2 \quad \text{if} \quad s \models_M \sigma_1 \text{ or } s \models_M \sigma_2. \]

\[ s \models_M E \phi \quad \text{if there exists } \pi \in E(M, s) \text{ such that } \pi \models_M \phi. \]
Semantics of Path Formulas

\[
\pi \models_M \sigma \text{ if } \pi[0] \models_M \sigma.
\]

\[
\pi \models_M \neg \phi \text{ if } \pi \not\models_M \phi.
\]

\[
\pi \models_M \phi_1 \lor \phi_2 \text{ if } \pi \models_M \phi_1 \text{ or } \pi \models_M \phi_2.
\]

\[
\pi \models_M X\phi \text{ if } \pi[1..] \models_M \phi.
\]

\[
\pi \models_M \phi_1 U \phi_2 \text{ if } \exists i \geq 0. \sigma[i..] \models_M \phi_2 \text{ and } \forall j < i. \sigma[j..] \models_M \phi_1
\]

Note: \(\neg, \lor, X, U\) as in LTL!

Derived operators: usual LTL derived operators, \(A \equiv \neg E \neg\).
### CTL* Verification Framework

**Sys:** Kripke structures $M = \langle S, A, R, \ell, s_I \rangle$

**Spec:** CTL* state formulas $\sigma \in \Sigma_{CTL*}$

**sat:** $M$ sat $\sigma$ if $s_I \models_M \sigma$. 
Expressing Properties in CTL* 

Possibility. “Any message can be lost.”

\[ \text{AG} (\text{sent} \Rightarrow \text{EF lost}) \]

LTL. Any LTL system specification \( \phi \) can be rendered as a CTL* state formula \( A\phi \).

So CTL* is at least as expressive a system specification notation as LTL. In fact, it is more so (LTL can’t express E).
CTL ...

... a sublogic of CTL*

... development preceded that of CTL*

... stands for “Computation Tree Logic”

... first temporal logic used in model checking

CTL formulas: state formulas in CTL* in which every path modality (U, X, G, etc.) is immediately preceded by a path quantifier.

\[
\text{CTL} \ AG(\text{sent } \Rightarrow \text{AF received})
\]

\[
\text{Not CTL} \ AG(\text{sent } \Rightarrow F \text{ received})
\]
Formal CTL Syntax

Note All CTL formulas are state formulas.

\[
\sigma ::= a \\
   \quad \mid \neg \sigma \\
   \quad \mid \sigma \lor \sigma \\
   \quad \mid \text{EX} \sigma \\
   \quad \mid E(\sigma \cup \sigma) \\
   \quad \mid E(\sigma \cap \sigma)
\]

\(\Sigma_{\text{CTL}}\): set of all CTL formulas. Recall that \(R\) is dual of \(U\).

Derived operators: \(AX, AU, AR, EG, EF, AG, AF,\) etc.
Expressiveness of CTL System Specifications

- No more expressive than CTL*, since $\Sigma_{\text{CTL}} \subseteq \Sigma_{\text{CTL}^*}$.
- Incomparable to LTL:
  - CTL can express possibility properties.
  - LTL can express “fairness” properties.
    - AFG $\alpha$ expressible in LTL, not in CTL.
- Consequently, strictly less expressive than CTL*! (Why?)