Formal Verification

Temporal Logic. Model Checking Basics

13 October 2008

- **Modeling** systems using finite-state machines
- Formal *specification* of sequencing properties: temporal logic
- **Model checking**: verification by traversing the state graph
Q: What kind of systems can we verify?

A: systems whose behavior is described precisely ⇒ mathematically
One of the simplest models: finite state machine
states and transitions (informally: “circles and arrows”)
Another view: system state: set of all quantities that determine the behavior of the system in time
Representation: every state has unique binary encoding (state variable)
Definition of state: depends on abstraction level
Example for a processor: instruction set level; internal organization (incl. pipeline); register transfer level; gate-level; transistor level
– discrete, continuous or hybrid systems
– finite (⇒ must be discrete) or infinite (continuous systems; programs with recursion or dynamic data structures)
Modeling finite state systems

Finite state machines (automata): defined by *states* and *transitions*
ex. program state = variables + prog. counter; transitions = statements
(finite state if finite types, no recursion, no dynamic data)

Our model: a set $V = \{v_1, v_2, \cdots, v_n\}$ of variables over a domain $D$
- a *state*: an *assignment* $s : V \rightarrow D$ of values for each variable in $D$
- A *state* (assignment) $\iff$ a *formula* true only for that assignment
  $\langle v_1 \leftarrow 7, v_2 \leftarrow 4, v_3 \leftarrow 2 \rangle \quad (v_1 = 7) \land (v_2 = 4) \land (v_3 = 2)$
- A *formula* $\iff$ the set of all assignments that make it true
  $\Rightarrow$ *sets of states*: representable by logic formulas, e.g., $v_1 \leq 5 \land v_2 > 3$
- A *transition* $s \rightarrow s'$ has two states $\Rightarrow$ a formula over $V \cup V'$
  where $V' = \text{copy of } V$ (next state variables)
e.g., $(\text{semaphore} = \text{red}) \land (\text{semaphore}' = \text{green})$
- *Transition relation*: set of all transitions $= \text{a formula } \mathcal{R}(V, V')$
Modeling with Kripke structures

Kripke structure $= \text{finite-state automaton with labeled states}$

$M = (S, S_0, R, L)$

(compare with automata: labels (input symbols) on \textit{transitions})

$S$: finite set of states

$S_0 \subseteq S$: set of initial states

$R \subseteq S \times S$: \textit{transition relation}

transition relation is \textit{total} if every state has at least one transition

$$\forall s \in S \ \exists s' \in S \ . \ (s, s') \in R$$

$L : S \rightarrow \mathcal{P}(AP)$: state \textit{labeling function} \quad $\mathcal{P}$: powerset (set of subsets)

where $AP = \text{set of atomic propositions}$ (observable boolean features that appear in formulas, properties, specifications). Examples:

- a state is \textit{stable} (or not)

  define the proposition: $bad ::= \text{number\_of\_errors} > 0$

\textit{Path} (trajectory) from a state $s_0$: \textit{infinite} sequence of states:

$$\pi = s_0s_1s_2\ldots, \text{ such that } R(s_i, s_{i+1}) \text{ for all } i \geq 0$$
Nondeterminism

Transitions are given as a \textit{relation}, not a \textit{function}.
\[ \Rightarrow \text{there can be several states } s' \text{ such that } s \rightarrow s', \text{ i.e., } (s, s') \in R \]
In this case the model (Kripke structure) is called \textit{nondeterministic} (the future behavior in a state is not uniquely determined).

This is different from the DFA / NFA distinction: finite state automata have \textit{transitions labeled} with \textit{input symbols}
\[ \Rightarrow \text{deterministic if unique next state for given state } \text{and } \text{input symbol} \]
(\text{even if different inputs can lead to different states})

For systems viewed as \textit{open} (interacting with an environment), this is called \textit{input nondeterminism}

Typically, we view Kripke models as \textit{closed}; we will discuss possible parallel composition with an environment
Expressing behavior

Input-output (functional) behavior is not enough for many systems: Reactive systems interact with environment: reaction to a stimulus
⇒ Often have infinite execution (operating systems, schedulers, servers)
⇒ A computation is an infinite sequence of states

Desired properties:
- A given (error) state is not reached (reachability problem)
- The system does not deadlock (deadlock freedom), etc.

More general properties can be described in temporal logic
= a modal logic, i.e., truth is qualified (possibly, always, etc.)
In this case: with temporal modalities: before, after, in the future, ...
– used already by ancient philosophers for reasoning about time
– formalized and applied by Pnueli (1977) to concurrent programs
Linear Temporal Logic (LTL)

Defined by Amir Pnueli in 1977 (ACM Turing Award 1996)
Describes event sequencing along an execution path ⇒ linear structure
   – an event happens in the future
   – a property is invariant (holds everywhere) starting at a given state
   – an event follows another event

Temporal operators (truth modalities along an execution trace)

- **X** (next): in the next state also written $\circ$
- **F** (future): sometime in the future $\lozenge$
- **G** (globally): in every future state (including now) $\square$
  unary operators, refer to one property
- **U** (until)
  binary operator, $property_1$ until $property_2$
Sometimes also: release operator $R$ (dual to until). Ignored here.
LTL Syntax

Express that a property is true for all paths
⇒ using the universal quantifier \( \mathbf{A} \)
⇒ LTL formulas are of the form \( \mathbf{A} f \), where \( f \) is a path formula

Syntax of path formulas:
\[
f ::= p \quad \text{base case: } p \in AP \text{ is an atomic proposition}
\]
\[
| \neg f_1 \mid f_1 \lor f_2 \mid f_1 \land f_2 \quad \text{usual boolean connectors}
| \mathbf{X} f_1 \mid \mathbf{F} f_1 \mid \mathbf{G} f_1 \mid f_1 \mathbf{U} f_2 \quad \text{temporal operators}
\]

Since the \( \mathbf{A} \) quantifier is mandatory, and appears only once, it is sometimes left implicit (some authors write path formulas only)
LTL Semantics

LTL formulas of the form $\mathbf{A} f$ have their meaning defined in a state ⇒ called state formulas: true if all paths from $s$ satisfy $f$

Path formulas have their meaning (truth value) defined over a path.

Notations:
$M, s \models f$ in the model (Kripke structure) $M$, state $s$ satisfies $f$
$M, \pi \models f$ in model $M$, path $\pi$ satisfies $f$

If $M$ is fixed (given), we simply write $s \models f$, $\pi \models f$

$\pi^i = \text{suffix of path } \pi = s_0 s_1 s_2 \ldots \text{ starting at } s_i : s_i s_{i+1} s_{i+2} \ldots$

Semantics of state formulas:
$s \models p \iff p \in L(s)$ (state $s$ has $p$ as a label)
$s \models \mathbf{A} f \iff \pi \models f$ for all paths $\pi$ from $s$

For path formulas, define semantics as usual by structural induction: the semantics of a formula is given in terms of its simpler subformulas
LTL semantics: path formulas

Semantics of path formulas:
\[ \pi \models p \iff s \models p \quad \text{for } p \in AP \text{ holds in path origin} \]
\[ \pi \models \neg f \iff \pi \not\models f \]
\[ \pi \models f_1 \lor f_2 \iff \pi \models f_1 \lor \pi \models f_2 \]
\[ \pi \models f_1 \land f_2 \iff \pi \models f_1 \land \pi \models f_2 \]
\[ \pi \models X f \iff \pi^1 \models f \]
  \text{for } f \text{ holds on the path suffix starting from state 1} \]
\[ \pi \models F f \iff \exists k \geq 0 . \pi^k \models f \]
  \text{there exists a suffix on which } f \text{ holds (} f \text{ holds in a state)} \]
\[ \pi \models G f \iff \forall k \geq 0 . \pi^k \models f \]
  \text{for } f \text{ holds on all path suffixes (} f \text{ holds in all states)} \]
\[ \pi \models f_1 U f_2 \iff \exists k \geq 0 . \pi^k \models f_2 \land \forall j < k . \pi^j \models f_1 \]
  \text{for some } k , f_1 \text{ holds everywhere prior to } f_2 \text{ on the path starting at } k \]
LTL is a \textit{linear} logic: paths are viewed independently; there may be many futures from origin, but can't express branching \textit{at each step} \Rightarrow not expressive enough (e.g., always \textit{possible} to reach a state) \Rightarrow another model: \textit{computation trees} (branching view) finite unfolding of a state-transition graph starting from an initial state
Additional path quantifier: $E$  there exists (a path)

Two classes of formulas:

*state formulas*, evaluated in a state

\[ f ::= p \quad \text{base case: } p \in AP \text{ atomic proposition} \]
\[ \quad | \neg f_1 | f_1 \lor f_2 | f_1 \land f_2 \quad f_1, f_2 \text{ state formulas} \]
\[ \quad | E g | A g \quad g \text{ path formula} \]

*path formula*, evaluated over a path

\[ g ::= f \quad \text{base case: } f \text{ is state formula} \]
\[ \quad | \neg g_1 | g_1 \lor g_2 | g_1 \land g_2 \]
\[ \quad | X g_1 | F g_1 | G g_1 | g_1 U g_2 \]
\[ \quad (\text{same rules as LTL, only base case more complex/expressive}) \]

Semantics: same rules as LTL, plus:

\[ s \models E g \iff \text{there exists a path } \pi \text{ from } s \text{ with } \pi \models g \]
Computation tree logic \(\text{CTL}\)

defined by Clarke \& Emerson (1981)

⇒ Turing Award 2007 with J. Sifakis for model checking

Tradeoff: expressiveness of specifications vs. efficiency of checking

⇒ CTL is subset of \(\text{CTL}^*\), efficient to check, enough in many cases

 CTL is a *branching-time* logic, like \(\text{CTL}^*\)

\(\text{CTL}\) *quantifies* over paths starting from a state

⇒ operators \(\text{X}, \text{F}, \text{G}, \text{U}\) are immediately preceded by \(\text{A sau E}\)

⇒ syntax of path formulas simplified, directly using state formulas:

\[
g ::= \text{X} f \mid \text{F} f \mid \text{G} f \mid f_1 \text{U} f_2 \mid f_1 \text{R} f_2
\]

Expressiveness: LTL and CTL incomparable (neither includes the other); both less expressive than \(\text{CTL}^*\)
Relations between operators

\[ f \land g \equiv \neg(\neg f \lor \neg g) \]
\[ \mathbf{F} \ f \equiv true \mathbf{U} \ f \]
\[ \mathbf{G} \ f \equiv \neg \mathbf{F} \neg f \]
\[ \mathbf{A} \ f \equiv \neg \mathbf{E} \neg f \]

\[ \Rightarrow \text{Operators } \neg, \lor, \mathbf{X}, \mathbf{U} \text{ and } \mathbf{E} \text{ suffice to express any } \mathbf{CTL}^* \text{ formula.} \]

\[ \mathbf{CTL} \text{ has } 2 \times 4 = 8 \text{ pairs of quantifier } \times \text{ temporal operator:} \]
\[ \mathbf{AX} \ f \equiv \neg \mathbf{EX} \neg f \]
\[ \mathbf{EF} \ f \equiv \mathbf{E}[true \mathbf{U} \ f] \]
\[ \mathbf{AF} \ f \equiv \neg \mathbf{EG} \neg f \]
\[ \mathbf{AG} \ f \equiv \neg \mathbf{EF} \neg f \]
\[ \mathbf{A}[f \mathbf{U} g] \equiv \neg \mathbf{EG} \neg g \land \neg \mathbf{E}[\neg g \mathbf{U}(\neg f \land \neg g)] \]

\[ \Rightarrow \text{ all of them expressible using } \mathbf{EX}, \mathbf{EU} \text{ and } \mathbf{EG} \]
Sample CTL formulas

- **EF** \( finish \)
  It is possible to get to a state in which \( finish = true \).

- **AG** \((send \rightarrow AF \, ack)\)
  Any \( send \) is eventually followed by an \( ack \).

- **AF AG** \( stable \)
  On any path, \( stable \) is invariant (always holds) after some point.

- **AG** \((req \rightarrow A \ [reg U grant])\)
  A \( req \) stays active until a \( grant \) is issued.

- **AG AF** \( ready \)
  On any path \( ready \) holds infinitely often.

- **AG EF** \( restart \)
  From any state, it is possible to reach a state labeled \( restart \).
Model checking. Problem setting

Given a Kripke structure $M = (S, S_0, R, L)$ and a temporal logic formula $f$, find which states from $S$ satisfy $f$: $\{s \in S \mid s \models f\}$

Def: A formula (spec.) $f$ holds in $M$ iff all initial states satisfy $f$:

$$M \models f \equiv \forall s_0 \in S_0 . s_0 \models f$$

History

– independently due to Clarke & Emerson; Queille & Sifakis (1981).
– initially: $10^4 – 10^5$ states. Now: to $10^{100}$ states (symbolic checking)

Model checking for CTL

By structural decomposition of formula $f$: compute truth of all sub-formulas of $f$ for each $s \in S$.

– initially, set $l(s) = L(s)$ (atomic propositions true in state $s$)
– trivial for logical connectors $\neg, \lor, \land$
– $\textbf{EX} f$: just label each state that has a successor labeled with $f$.
– to discuss: two algorithms for basic operators $\textbf{EU}$ and $\textbf{EG}$
CTL model checking. The operator EU

Idea: backwards traversal from states labeled $f_2$ as long as $f_1$ holds

procedure $CheckEU(f_1, f_2)$

$$T := \{ s \mid f_2 \in l(s) \}$$

forall $s \in T$ do $l(s) := l(s) \cup \{ E[f_1 U f_2] \}$; \hspace{1cm} \text{if $f_2$ holds in $s$}

while $T \neq \emptyset$ do

choose $s \in T$;

$T := T \setminus \{ s \}$;

forall $s_1 . R(s_1, s)$ do

if $E[f_1 U f_2] \not\in l(s_1) \land f_1 \in l(s_1)$ then

$$l(s_1) := l(s_1) \cup \{ E[f_1 U f_2] \};$$

$T := T \cup \{ s_1 \};$

\hspace{1cm} \text{if $E[f_1 U f_2]$ holds, label $s$}

\hspace{1cm} \text{still have candidates for search}

\hspace{1cm} \text{never consider $s$ twice}

\hspace{1cm} \text{for all predecessors of $s$}

\hspace{1cm} \text{$s_1$ not labeled but $f_1$ holds}$

\hspace{1cm} \text{$E[f_1 U f_2]$ also holds, label it}$

\hspace{1cm} \text{$s_1$ is candidate for continuing search}$

Terminates since $S$ finite and no labeled state reenters $T$
Consider only states satisfying \( f \). Traverse backwards starting from strongly connected components (on cycles where \( f \) perpetually holds).

**procedure** \( \text{CheckEG}(f) \)  

\[
S' := \{ s : f \in l(s) \}; \\
SCC := \{ C : C \text{ is nontrivial SCC of } S' \}; \\
T := \bigcup_{C \in SCC} \{ s : s \in C \}; \\
\text{forall } s \in T \text{ do } l(s) := l(s) \cup \{ \text{EG } f \}; \\
\text{while } T \neq \emptyset \text{ do} \\
\quad \text{choose } s \in T; \\
\quad T := T \setminus \{ s \}; \\
\quad \text{forall } s_1 : s_1 \in S' \land R(s_1, s) \text{ do} \\
\quad \quad \text{if } \text{EG } f \notin l(s_1) \text{ then} \\
\quad \quad \quad l(s_1) := l(s_1) \cup \{ \text{EG } f \}; \\
\quad \quad \quad T := T \cup \{ s_1 \}; \ s_1 \text{ is candidate for continuing search} \\
\text{Terminates; will reach at most every state in } S'
Fairness

In practice, we check systems subject to “reasonable” assumptions as:
– a request is not ignored forever (by a scheduler/arbiter)
– communication channels do not continually fail (thus, a message being retransmitted is eventually delivered)
These are properties expressible in CTL*, but not CTL.
⇒ need to extend CTL (semantics) with *fairness constraints*

Intuitively: decision fairness = if a decision (several transitions from a state) is repeated infinitely often, each branch is eventually taken
Reformulate: each destination state of the decision is eventually reached

Formally: A fairness constraint is a *formula* in temporal logic. A path is *fair* iff the constraint is infinitely often true along the path.
II LTL, we would write: \( \mathbf{F} \mathbf{G} \text{assumption} \Rightarrow \text{conclusion} \)
In particular: fairness constraint expressed as *set of states*
⇒ a *fair path* passes infinitely often through the set
Model checking CTL with fairness

Augment the Kripke structure \( M = (S, S_0, R, L, F) \), with \( F \subseteq \mathcal{P}(S) \) (\( F \) = set of subsets of states, \( \{P_1, \ldots, P_n\} \), \( P_i \subseteq S \))

\[
\inf(\pi) \overset{\text{def}}{=} \{ s \mid s = s_i \text{ for infinitely many } i \}
\]
(set of states appearing infinitely often on \( \pi \))

\( \pi \) is a fair path \( \iff \forall P \in F . \inf(\pi) \cap P \neq \emptyset \).
(\( \pi \) passes infinitely often through each set from \( F \))

For \( \models_F \), ("holds fairly") replace "path" with "fair path" in semantics

For model checking, define new atomic proposition \( \text{fair} \):

\[
\text{fair} \in L(s) \iff M, s \models_F \text{EG true}
\]

\( \Rightarrow \) fair-CTL model checking reduces to CTL for \( AP \cup \{\text{fair}\} \)
Complexity of model checking algorithms

- CTL model checking: \( O(|f| \cdot (|S| + |R|)) \)
  (linear in size of model and formula)
- CTL with fairness:
  \( O(|f| \cdot (|S| + |R|) \cdot |F|) \)
- LTL: PSPACE-complete
  (different type of algorithm, based on a tableau construction)
- CTL*: like LTL
  \( |M| \cdot 2^{O(|f|)} \)

CTL: usually preferred, because of polynomial (linear!) algorithm
Spin uses LTL: exponential only in size of formula (usually small)
Synchronous and asynchronous composition

Behavior of composed systems emerges from component behavior.

For concurrently executing components: \textit{parallel} composition:

- \textbf{synchronous}: conjunction (simultaneous transitions)
  \[
  R(V, V') = R_1(V_1, V'_1) \land R_2(V_2, V'_2) \quad V = V_1 \cup V_2
  \]

- \textbf{asynchronous}: disjunction (individual transitions)
  \[
  R(V, V') = R_1(V_1, V'_1) \land Eq(V \setminus V_1) \lor R_2(V_2, V'_2) \land Eq(V \setminus V_2)
  \]
  \[
  Eq(U) = \bigwedge_{v \in U} (v = v')
  \]

- arbitrary interleaving between transitions of components
- a transition modifies just the variables of \textit{one} component
- simultaneous transitions are deemed impossible