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* $s : V \to D$ of values for each variable in D

– A *state* (assignment) ⇔ a *formula* true only for that assignment
$\langle v_1 \leftarrow 7, v_2 \leftarrow 4, v_3 \leftarrow 2\rangle$ $(v_1 = 7) \wedge (v_2 = 4) \wedge (v_3 = 2)$
– A *formula* ⇔ the set of all assignments that make it true
⇒ *sets of states*: representable by logic formulas, e.g., $v_1 \le 5 \wedge v_2 > 3$
– A *transition* $s \to s'$ has *two* states ⇒ a formula over $V \cup V'$
where V' = copy of V (next state variables)
e.g., $(semaphore = red) \wedge (semaphore' = green)$
– *Transition relation*: set of all transitions = a formula $\mathcal{R}(V, V')$

Modeling with Kripke structures

Kripke structure = finite-state automaton with labeled *states*

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

(compare with automata: labels (input symbols) on *transitions*)
S: finite set of states
$S_0 \subseteq S$: set of initial states
$R \subseteq S \times S$: transition relation
transition relation is *total* if every state has at least one transition

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

$L : S \to \mathcal{P}(AP)$: state labeling function $\mathcal{P}$: powerset (set of subsets)
where AP = set of atomic propositions (observable boolean features that appear in formulas, properties, specifications). Examples:
a state is *stable* (or not)
define the proposition: $bad ::= number\_of\_errors > 0$
Path (trajectory) from a state s_0: *infinite* sequence of states:
$\pi = s_0 s_1 s_2 \ldots$, such that $R(s_i, s_{i+1})$ for all $i \ge 0$

Nondeterminism

Transitions are given as a *relation*, not a *function*.
⇒ there can be several states s' such that $s \to s'$, i.e., $(s, s') \in R$
In this case the model (Kripke structure) is called *nondeterministic*
(the future behavior in a state is not uniquely determined).

This is different from the DFA / NFA distinction: finite state automata have *transitions labeled* with *input symbols*
⇒ deterministic if unique next state for given state *and* input symbol
(even if different inputs can lead to different states)

For systems viewed as *open* (interacting with an environment), this is called *input nondeterminism*
Typically, we view Kripke models as *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 $\bigcirc$
- **F** (*future*): sometime in the future — $\Diamond$
- **G** (*globally*): in every future state (including now) — $\Box$
  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* **A**
⇒ LTL formulas are of of the form **A** f, where f is a *path formula*

Syntax of path formulas:

f ::= p — base case: $p \in AP$ is an atomic proposition
| $\neg f_1$ | $f_1 \vee f_2$ | $f_1 \wedge f_2$ — usual boolean connectors
| **X** f_1 | **F** f_1 | **G** f_1 | f_1 **U** f_2 — temporal operators

Since the **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 **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 π satisfies f
If M is fixed (given), we simply write $s \models f$, $\pi \models f$
π^i = suffix of path $\pi = s_0 s_1 s_2 \ldots$ starting at s_i : $s_i s_{i+1} s_{i+2} \ldots$

Semantics of state formulas:
$s \models p$ ⇔ $p \in L(s)$ (state s has p as a label)
$s \models$ **A** f ⇔ $\pi \models f$ for all paths π 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

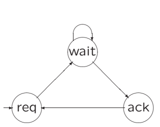
Semantics of path formulas:
$\pi \models p$ ⇔ $s \models p$ — $p \in AP$ holds in path origin

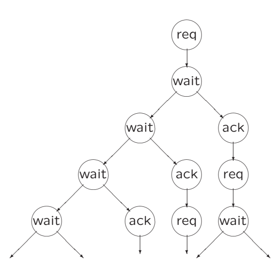
$\pi \models \neg f$ ⇔ $\pi \not\models f$
$\pi \models f_1 \vee f_2$ ⇔ $\pi \models f_1 \vee \pi \models f_2$
$\pi \models f_1 \wedge f_2$ ⇔ $\pi \models f_1 \wedge \pi \models f_2$

$\pi \models$ **X** f ⇔ $\pi^1 \models f$
f holds on the path suffix starting from state 1
$\pi \models$ **F** f ⇔ $\exists k \geq 0 \,.\, \pi^k \models f$
there exists a suffix on which f holds (f holds in a state)
$\pi \models$ **G** f ⇔ $\forall k \geq 0 \,.\, \pi^k \models f$
f holds on all path suffixes (f holds in all states)
$\pi \models f_1$ **U** f_2 ⇔ $\exists k \geq 0 \,.\, \pi^k \models f_2 \wedge \forall j < k \,.\, \pi^j \models f_1$
f_2 holds on path starting at k (for some k), f_1 holds everywhere prior

The temporal logic CTL*

LTL is a *linear* logic: paths are viewed independently; there may be many futures from origin, but can't express branching *at each step*
⇒ not expressive enough (e.g., always *possible* to reach a state)
⇒ another model: *computation trees* (branching view)
finite unfolding of a state-transition graph starting from an initial state





CTL* syntax and semantics

Additional path quantifier: **E** there exists (a path) — $\exists$

Two classes of formulas:
state formulas, evaluated in a state
f ::= p — base case: $p \in AP$ atomic proposition
| $\neg f_1$ | $f_1 \vee f_2$ | $f_1 \wedge f_2$ — f_1, f_2 state formulas
| **E** g | **A** g — g path formula

path formula, evaluated over a path
g ::= f — base case: f is state formula
| $\neg g_1$ | $g_1 \vee g_2$ | $g_1 \wedge g_2$
| **X** g_1 | **F** g_1 | **G** g_1 | g_1 **U** g_2
(same rules as LTL, only base case more complex/expressive)

Semantics: same rules as LTL, plus:
$s \models$ **E** g ⇔ there exists a path π from s with $\pi \models g$

Computation tree logic 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 CTL*, efficient to check, enough in many cases

CTL is a *branching-time* logic, like CTL*
CTL *quantifies* over paths starting from a state
⇒ operators **X**, **F**, **G**, **U** are immediately preceded by **A** sau **E**
⇒ syntax of path formulas simplified, directly using state formulas:

$$g ::= \mathbf{X}\, f \mid \mathbf{F}\, f \mid \mathbf{G}\, f \mid f_1 \,\mathbf{U}\, f_2 \mid f_1 \,\mathbf{R}\, f_2$$

Expressiveness: LTL and CTL incomparable (neither includes the other); both less expressive than CTL*

Relations between operators

$f \wedge g \equiv \neg(\neg f \vee \neg g)$
$\mathbf{F}\, f \equiv$ *true* $\mathbf{U}\, f$
$\mathbf{G}\, f \equiv \neg \mathbf{F}\, \neg f$
$\mathbf{A}\, f \equiv \neg \mathbf{E}\, \neg f$

⇒ Operators ¬, ∨, **X**, **U** and **E** suffice to express any *CTL** formula.

CTL has 2 x 4 = 8 pairs of quantifier × temporal operator:

$\mathbf{AX}\, f \equiv \neg \mathbf{EX}\, \neg f$
$\mathbf{EF}\, f \equiv \mathbf{E}\, [\mathit{true}\, \mathbf{U}\, f]$
$\mathbf{AF}\, f \equiv \neg \mathbf{EG}\, \neg f$
$\mathbf{AG}\, f \equiv \neg \mathbf{EF}\, \neg f$
$\mathbf{A}\, [f\, \mathbf{U}\, g] \equiv \neg \mathbf{EG}\, \neg g \wedge \neg \mathbf{E}\, [\neg g\, \mathbf{U}\, (\neg f \wedge \neg g)]$

⇒ all of them expressible using **EX**, **EU** and **EG**

Sample CTL formulas

- **EF** *finish*
  It is possible to get to a state in which *finish* = *true*.
- **AG** (*send* → **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* → **A** [*req* **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:

$$M \models f \overset{\text{def}}{=} \forall s_0 \in S_0 \,.\, s_0 \models f$$

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* subformulas of f for each $s \in S$.
– initially, set $l(s) = L(s)$ (atomic propositions true in state s)
– trivial for logical connectors $\neg, \vee, \wedge$
– **EX** f: just label each state that has a successor labeled with f.
– to discuss: two algoritms for basic operators **EU** and **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)
 $T := \{s \mid f_2 \in l(s)\}$ — if f_2 holds in s
 forall $s \in T$ **do** $l(s) := l(s) \cup \{\mathbf{E}\,[f_1\, \mathbf{U}\, f_2]\}$; — then $E[f_1\, \mathbf{U}\, f_2]$ holds, label s
 while $T \neq \emptyset$ **do** — still have candidates for search
  choose $s \in T$;
  $T := T \setminus \{s\}$; — never consider s twice
  forall $s_1 \,.\, R(s_1, s)$ **do** — for all predecessors of s
   if $\mathbf{E}\,[f_1\, \mathbf{U}\, f_2] \notin l(s_1) \wedge f_1 \in l(s_1)$ **then** — s_1 not labeled but f_1 holds
    $l(s_1) := l(s_1) \cup \{\mathbf{E}\,[f_1\, \mathbf{U}\, f_2]\}$; — $E[f_1\, \mathbf{U}\, f_2]$ also holds, label it
    $T := T \cup \{s_1\}$; — s_1 is candidate for continuing search

Terminates since S finite and no labeled state reenters T

CTL model checking. The operator EG

Consider only states satisfying f. Traverse backwards starting from strongly connected components (on cycles where f perpetually holds).

procedure *CheckEG*(f) — restrict to states where f holds
 $S' := \{s \mid f \in l(s)\}$;
 $SCC := \{C \mid C$ is nontrivial SCC of $S'\}$; — at least one edge
 $T := \cup_{C \in SCC}\{s \mid s \in C\}$; — all states in SCCs are on cycles
 forall $s \in T$ **do** $l(s) := l(s) \cup \{\mathbf{EG}\, f\}$; — thus get labeled
 while $T \neq \emptyset$ **do** — still have candidates for backwards search
  choose $s \in T$;
  $T := T \setminus \{s\}$; — continue from s only once
  forall $s_1 \,.\, s_1 \in S' \wedge R(s_1, s)$ **do** — for all predecessors of s
   if $\mathbf{EG}\, f \notin l(s_1)$ **then** — if s_1 not yet labeled
    $l(s_1) := l(s_1) \cup \{\mathbf{EG}\, f\}$; — label s_1
    $T := T \cup \{s_1\}$; — s_1 is candidate for continuing search

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.
$\Rightarrow$ 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: $\mathbf{F}\,\mathbf{G}\, assumption \Rightarrow \; conclusion$
In particular: fairness constraint expressed as *set of states*
$\Rightarrow$ 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, $\{P_1, \cdots, P_n\}, P_i \subseteq S$)

$$\text{inf}(\pi) \stackrel{\text{def}}{=} \{s \mid s = s_i \text{ for infinitely many } i\}$$

(set of states appearing infinitely often on π)

π is a *fair* path $\Leftrightarrow \forall P \in F \,.\, \text{inf}(\pi) \cap P \neq \emptyset$.
(π 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 *fair*:

$$\mathit{fair} \in L(s) \;\Leftrightarrow\; M, s \models_F \mathbf{EG}\, \mathit{true}$$

$\Rightarrow$ fair-CTL model checking reduces to CTL for $AP \cup \{fair\}$

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 $|M| \cdot 2^{O(|f|)}$
(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: *parallel* composition:

- synchronous: conjunction (simultaneous transitions)

$$R(V, V') = R_1(V_1, V_1') \wedge R_2(V_2, V_2') \qquad V = V_1 \cup V_2$$

- asynchronous: disjunction (individual transitions)

$$R(V, V') = R_1(V_1, V_1') \wedge \mathit{Eq}(V \setminus V_1) \vee R_2(V_2, V_2') \wedge \mathit{Eq}(V \setminus V_2)$$
$$\mathit{Eq}(U) = \bigwedge_{v \in U} (v = v')$$

– arbitrary interleaving between transitions of components
– a transition modifies just the variables of *one* component
– simultaneous transitions are deemed impossible