Model Checking Basics

October 13, 2005

- Finite state systems
- Temporal logics: CTL*, CTL, LTL
- Explicit-state model checking

What kind of systems can we verify ?

– systems whose behavior can be described mathematically
– we analyze: the interaction of the system with its environment
– system *state* = all quantities that determine its future behavior in time
– the definition of state depends on the *abstraction* level in
Example for a processor: instruction set level; internal organization (incl. pipeline, etc.); register transfer level; gate-level; transistor level

System classification:
– *discrete*, *continuous* or *hybrid* systems
– *finite* (necessarily discrete) or *infinite* (continuous systems, recursive programs, programs with dynamic data structures)

Modeling of finite-state systems

– Finite state machines (automata): states + transitions
– Programs (finite): variables + program counter
There is no conceptual difference !

Let $V = \{v_1, v_2, \cdots, v_n\}$ be a set of variables.
A *state*: an *assignment* $s : V \to D$ of values from a given *domain* D for each variable $v \in V$.
– 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
e.g. $v_1 \leq 5 \wedge v_2 > 3$
⇒ *sets* of states can be represented by logic formulas

– A *transition* $s \to s'$: a formula over $V \cup V'$
V' = copy of V (next state formulas)
ex. $(semaphore = red) \wedge (semaphore' = green)$
– set of all transitions: *transition relation* = a formula $\mathcal{R}(V, V')$

Modeling with Kripke structures

Kripke structure = labeled finite-state automaton

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

– S: finite state set
– $S_0 \subseteq S$: set of initial states
– $R \subseteq S \times S$: *total* transition relation $\forall s \in S\ \exists s' \in S\ .\ (s, s') \in R$
(from every state there is at least one transition)
– $L : S \to 2^{AP}$: state labeling function

AP =set of atomic propositions (observations that appear in formulas/properties/specifications). Examples:
– a state is *stable* or not
– define the proposition $bad ::= red\_recvd > 1$ (Spin project)

Path (trajectory): *infinte* set of states starting from s_0:
$\pi = s_0 s_1 s_2 \ldots$, with $R(s_i, s_{i+1})$ for all $i \geq 0$

Modeling: circuits and programs

- *sequential* circuits: a variable for each state element (register) and for primary inputs
  instantaneous combinational propagation assumed
- *asynchronous* circuits: one variable for each signal
  (in more complex/accurate models: explicit physical time)
- programs: declared variables + program counter
  (for procedures, need to keep track of local variables on stack during time of procedure activation; potentially infinite-state)

Synchrony and asynchrony

Types of composition
(deriving system behavior from behavior of components)

- synchronous: conjunction (simultaneous transitions)
  $R(V, V') = R_1(V_1, V_1') \wedge R_2(V_2, V_2')$ $V = V_1 \cup V_2$
- asynchronous: disjunction (individual transitions)
  $R(V, V') = R_1(V_1, V_1') \wedge Eq(V \setminus V_1) \vee R_2(V_2, V_2') \wedge Eq(V \setminus V_2)$
  where $Eq(U) = \bigwedge_{v \in U}(v = v')$

– arbitrary interleaving between component transitions
– a transition changes just the variables of *one* component
– simultaneous transitions considered impossible

Programs are usually modeled asynchronously (there is no physical synchronization between instructions of concurrent programs)

Modeling behavior

Reactive systems
– interact with the environment (*reaction* to a given *stimulus*)
– often have infinite execution
⇒ a *computation* = infinite set of states
⇒ it is not enough to represent input-output behavior

– Examples:
a given (error) state is not reached
the system does not deadlock

More generally: properties described in temporal logic
– *modal* logic (truth with temporal modalities)
– used starting in anntiquity for reasoning about time
– formalized and applied by Pnueli (1977) to concurrent programs

Linear Temporal Logic (LTL)

– defined by Pnueli in 1977 (Turing Award 1996)
– describes events along an execution trace ⇒ *linear* structure
e.g. an event happens in the future; a property is invariant starting from a given timepoint; an event follows another event

Temporal operators (truth modalities along an execution trace):
- **X** : in the *next state* ○
- **F** : sometime in the *future* (incl. now) ◇
- **G** : *globally* (in every future state, starting now) □
- **U** : *until*; $prop_1$ must hold until $prop_2$ appears
  sometimes we also define
- **R** (*release*): appearance of $prop_1$ releases the need for $prop_2$

Syntax of LTL Formulas

– we wish a property to hold for *all* trajectories
⇒ we use the *universal quantifier* **A**
– formulas are of the form **A** f, where f is a *path formula*
– Syntax of path formulas
f ::= p (for $p \in AP$)
| $\neg f_1$ | $f_1 \vee f_2$ | $f_1 \wedge f_2$
| $\mathbf{X}\, f_1$ | $\mathbf{F}\, f_1$ | $\mathbf{G}\, f_1$ | $f_1\, \mathbf{U}\, f_2$ | $f_1\, \mathbf{R}\, f_2$

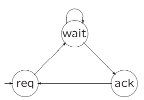
Semantics of LTL

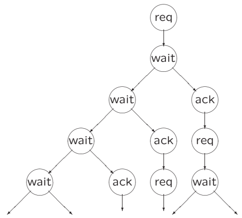
Denote $M, s \models f$: in the model M, state s satisfies f
π^i = suffix of the path $\pi = s_0 s_1 s_2 \ldots$ starting at s_i

$M, s \models p$	⇔	$p \in L(s)$
$M, s \models \mathbf{A}\, f$	⇔	∀ path π from s, $M, \pi \models f$
$M, \pi \models p$	⇔	$M, s \models p$, for $p \in AP$ and s the first state of π
$M, \pi \models \neg f$	⇔	$M, \pi \not\models f$
$M, \pi \models f_1 \vee f_2$	⇔	$M, \pi \models f_1 \vee M, \pi \models f_2$
$M, \pi \models f_1 \wedge f_2$	⇔	$M, \pi \models f_1 \wedge M, \pi \models f_2$
$M, \pi \models \mathbf{X}\, f$	⇔	$M, \pi^1 \models f$
$M, \pi \models \mathbf{F}\, f$	⇔	$\exists k \geq 0 \,.\, M, \pi^k \models f$
$M, \pi \models \mathbf{G}\, f$	⇔	$\forall k \geq 0 \,.\, M, \pi^k \models f$
$M, \pi \models f_1\, \mathbf{U}\, f_2$	⇔	$\exists k \geq 0 \,.\, M, \pi^k \models f_2 \wedge \forall j < k \,.\, M, \pi^j \models f_1$
$M, \pi \models f_1\, \mathbf{R}\, f_2$	⇔	$\forall k \geq 0 \,.\, (\forall j < k \,.\, M, \pi^j \not\models f_1) \rightarrow M, \pi^k \models f_2$

The temporal logic CTL*

Some properties cannot be expressed in the linear time model:
e.g. *it is possible* to reach a state
⇒ alternative model: *computation trees*:
infinite unfolding of state-transition system starting from initial state





Structure of CTL* Formulas

In addition to LTL operators:
existential quantifier **E** (there exists a path) ∃

Two types of formulas:
– *state formulas*, evaluated in a state
f ::= p (unde $p \in AP$)
| $\neg f_1$ | $f_1 \vee f_2$ | $f_1 \wedge f_2$
| $\mathbf{E}\, g$ | $\mathbf{A}\, g$ (where g = path formula)

– *path formulas*, evaluated along a path
g ::= f (where f = state formula)
| $\neg g_1$ | $g_1 \vee g_2$ | $g_1 \wedge g_2$
| $\mathbf{X}\, g_1$ | $\mathbf{F}\, g_1$ | $\mathbf{G}\, g_1$ | $g_1\, \mathbf{U}\, g_2$ | $g_1\, \mathbf{R}\, g_2$

Semantics: similar to LTL, plus:
$M, s \models \mathbf{E}\, g \Leftrightarrow \exists$ a path π from s such that $M, \pi \models g$

Relations among temporal operators

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

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

A sublogic: CTL

CTL (Computation Tree Logic) [Clarke, Emerson 1981]
– sufficient in many cases, but simpler ⇒ more efficient algorithms
– *branching* structure, like CTL*
– quantifies over all possible execution paths from a state
– operators **X**, **F**, **G**, **U**, **R** must be immediately preceded by **A** or **E**
– syntax of path formulas:

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

CTL: fundamental and derived operators

10 combinations, all expressible using **EX**, **EG** și **EU**:

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

Sample CTL formulas

- **EF** *finish*
  It is possible to reach a state in which *finish* = *true*.
- **AG** (*send* → **AF** *ack*)
  Any *send* is eventually followed by an *ack*.
- **AF AG** *stable*
  In any execution, from a given moment on, *stable* holds overall.
- **AG** (*req* → **A** [*req* **U** *grant*])
  A *req* stays always active until receiving a *grant*.
- **AG AF** *ready*
  On any path, *ready* holds an infinite number of times.
- **AG EF** *restart*
  From any state it is possible to get to the *restart* state.

Relations among various logics

CTL and LTL are incomparable:
– **A F G** p is in LTL, has no CTL equivalent
– **AG EF** p is in CTL, has no LTL equivalent
– their disjunction is in CTL*, but not in CTL, nor LTL

Some techniques (compositionality, abstraction) need restrictions:
typically, only the universal quantifier **A** is allowed
– ACTL (included in CTL, incomparable to LTL)
– ACTL* (included in CTL*, more expressive than LTL)

The notion of fairness

in practice: reasonable assumptions of the sort:
– an arbiter does not continuously ignore a particular request
– a continuously retransmitted message reaches destination
= properties which can be expressed in CTL* but not CTL
⇒ define a new semantics for CTL with *fairness*

A fairness constraint is a formula in temporal logic.

A path is *fair* is each constraint is true infinitely often along the path.
In particular: constraint expressed as set of states:
a fair path passes through that state infinitely often

CTL with fairness

Augment Kripke structure, $M = (S, S_0, R, L, F)$, by $F \subseteq 2^S$
(F = set of state sets, $\{P_1, \cdots, P_n\}, P_i \subseteq S$)

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

(set of states apearing infinitely often on π)

π is fair $\Leftrightarrow \forall P \in F \,.\, \text{inf}(\pi) \cap P \neq \emptyset$.
(π passes infinitely often through any set in F)

Denote $\models_F$ the satifaction relationship with fairness
Modified clauses in CTL semantics:

$M, s \models_F p \quad \Leftrightarrow$ there is a fair path from s and $p \in L(s)$
$M, s \models_F \mathbf{E}\, g \quad \Leftrightarrow \exists$ fair path π from s cu $M, \pi \models_F g$
$M, s \models_F \mathbf{A}\, g \quad \Leftrightarrow \forall$ fair paths π from s, $M, \pi \models_F g$

Model checking. Problem statement

Given a Kripke structure $M = (S, S_0, R, L)$ and a formula f in temporal logic, find the set of states S that satisfy f:

$$\{s \in S \mid M, s \models f\}$$

The specification is satisfied if all initial states satisfy f:

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

History

– independently, Clarke & Emerson, resp. Queille & Sifakis (1981).
– iniyially: $10^4 - 10^5$ states. currently, symbolic techniques: ca. 10^{100} states

Model checking for CTL

– Decompose according to the structure of formula f. For any $s \in S$, compute $l(s)$ = set of subformulas of f true in s.
– initially $l(s) = L(s)$. Trivial for logic connectors $\neg, \vee, \wedge$
– **EX** f: label any state with a successoor labeled by cu f.
– Other basic operators: **EU** and **EG**

Model checking for CTL. The EU Operator

E $[f_1$ **U** $f_2]$: backwards traversal from 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** $l(s) := l(s) \cup \{\mathbf{E}\,[f_1\, \mathbf{U}\, f_2]\}$;
  while $T \neq \emptyset$ **do**
    choose $s \in T$;
    $T := T \setminus \{s\}$;
    forall $s_1 \,.\, R(s_1, s)$ **do**
      if $\mathbf{E}\,[f_1\, \mathbf{U}\, f_2] \notin l(s_1) \wedge f_1 \in l(s_1)$ **then**
        $l(s_1) := l(s_1) \cup \{\mathbf{E}\,[f_1\, \mathbf{U}\, f_2]\}$;
        $T := T \cup \{s_1\}$;

Model checking for CTL. The EG Operator

EG f: consider only states that satisfy f. Traverse backwards starting from strongly connected components (SCC)

procedure *CheckEG*(f)
  $S' := \{s \mid f \in l(s)\}$;
  $SCC := \{C \mid C$ is a nontrivial SCC in $S'\}$;
  $T := \cup_{C \in SCC} \{s \mid s \in C\}$;
  forall $s \in T$ **do** $l(s) := l(s) \cup \{\mathbf{EG}\, f\}$;
  while $T \neq \emptyset$ **do**
    choose $s \in T$;
    $T := T \setminus \{s\}$;
    forall $s_1 \,.\, s_1 \in S' \wedge R(s_1, s)$ **do**
      if $\mathbf{EG}\, f \notin l(s_1)$ **then**
        $l(s_1) := l(s_1) \cup \{\mathbf{EG}\, f\}$;
        $T := T \cup \{s_1\}$;

Model checking with fairness

Consider the fairness constraint $F = \{P_1, \cdots, P_k\}$, with $P_i \subseteq S$

Let *fair* be a new atomic proposition, true in s iff there is a fair path starting from s.
Thus $\mathit{fair} \in L(s) \;\Leftrightarrow\; M, s \models_F \mathbf{EG}\, \mathit{true}$.
For the other operators, the problem is reduced to ordinary model checking
$M, s \models_F p \Leftrightarrow M, s \models p \wedge \mathit{fair}$
$M, s \models_F \mathbf{EX}\, f \Leftrightarrow M, s \models \mathbf{EX}\, (f \wedge \mathit{fair})$
$M, s \models_F \mathbf{E}\,[f_1\, \mathbf{U}\, f_2] \Leftrightarrow M, s \models \mathbf{E}\,[f_1\, \mathbf{U}\, (f_2 \wedge \mathit{fair})]$

For $M, s \models_F \mathbf{EG}\, f$ we modify the previous algorithm, considering only SCCs with $\forall i \,.\, C \cap P_i \neq \emptyset$ (that contain at least a state from each component of the fairness constraint)

Complexity of model checking algorithms

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

CTL: often preferred due to the polynomial algorithm
but also in LTL, the exponential is in the size of the formula (small)