Finite Automata, CTL, LTL and Model Checking

Lecture 2

Model-Checking Finite-State Systems
(untimed systems)

Finite Automata, CTL, LTL and Model Checking

Finite state automata

- Finite graphs with labels on edges/nodes
  - a set of nodes (states)
  - a set of edges (transitions)
  - a set of labels (alphabet)

Complete Systems and Kripke Structure

- From now on, we shall consider only Complete systems, that is, automata with labels on nodes.
  - There is no essential difference between models with labels on nodes or transitions
- This is the so called Kripke Structure, that is, automata with propositions labeled on states

CTL Models = Kripke Structures

A CTL model is a triple $\mathcal{M} = (S, R, \text{Label})$ where

- $S$ is a non-empty set of states,
- $R \subseteq S \times S$ is a total relation on $S$, which relates to $s \in S$ its possible successor states,
- $\text{Label} : S \rightarrow 2^{AP}$, assigns to each state $s \in S$ the atomic propositions $\text{Label}(s)$ that are valid in $s$.

Example

CTL: Computation Tree Logics
defined on Computation Trees of Kripke structures
Computation Tree Logic, CTL
Clarke & Emerson 1980

Syntax
\[ \phi ::= p | \neg \phi | \phi \lor \phi | EX\phi | E[\phi U \phi] | A[\phi U \phi]. \]

- EX (pronounced “for some path next”)
- E (pronounced “for some path”)
- A (pronounced “for all paths”) and
- U (pronounced “until”).

Path

Definition 20. (Path)
A path is an infinite sequence of states \( s_0, s_1, s_2, \ldots \) such that \( (s_i, s_{i+1}) \in R \) for all \( i \geq 0 \).

The set of path starting in \( s \)

\[ P_{\text{Ma}}(s) \]

Formal Semantics

(satisfaction relation \( \models \))

- \( s \models p \iff p \in \text{Label}(s) \)
- \( s \models \neg \phi \iff \neg (s \models \phi) \)
- \( s \models \phi \lor \psi \iff (s \models \phi) \lor (s \models \psi) \)
- \( s \models EX\phi \iff \exists \sigma \in P_{\text{Ma}}(s, \sigma)[\models \phi] \)
- \( s \models E[\phi U \psi] \iff \exists \sigma \in P_{\text{Ma}}(s, \sigma)[\exists j \geq 0. \sigma[j] \models \psi \land (\forall 0 \leq k < j. \sigma[k] \models \phi)] \)
- \( s \models A[\phi U \psi] \iff \forall \sigma \in P_{\text{Ma}}(s, \sigma)[\exists j \geq 0. \sigma[j] \models \psi \land (\forall 0 \leq k < j. \sigma[k] \models \phi)] \).

CTL, Derived Operators

- \( EF \phi \equiv E[\text{true U } \phi] \) (possible)
- \( AF \phi \equiv A[\text{true U } \phi] \) (inevitable)

There are too many operators! But

We need to remember only the following:

- \( X \) (next time)
- \( F \) (future, some time)
- \( G \) (global)
- \( U \) (until)

The most useful are \( EF, AG, EG \) and \( AF \):
Theorem

All operators are derivable from

- \( \text{EX } f \)
- \( \text{EG } f \)
- \( \text{E} [ f \cup g ] \)

and boolean connectives

\[ A[f \cup g] = \neg E[\neg g(U(\neg f \land \neg g)) \land \neg EG \neg g] \]

Example

\[ \text{EX } p \]

Example

\[ \text{EX } p \]

Example

\[ \text{AX } p \]

Example

\[ \text{AX } p \]

Note: state 1 doesn’t satisfy \( \text{AX } p \)
Properties of MUTEX example?

\[ \text{AG} \neg (C_1 \land C_2) \]
\[ \text{AG} (T_1 \Rightarrow AF(C_1)) \]
\[ \text{EG} \neg C_1 \]
\[ \text{AG} [C_1 \Rightarrow A C_1 \cup \neg C_1 \land A [\neg C_1 \cup C_2]] \]

HOW TO DECIDE IN GENERAL

CTL Model Checking Algorithms

Labeling Methods [Clarke et al 81]
- Check all sub-formulas of \( F \)
- For each sub-formula \( f \) of \( F \), label all nodes where \( f \) is true
- Check the composed formulas

Algorithm ideas for checking \( E(f \cup g) \)
- Mark all nodes where \( f \) is true and all nodes where \( g \) is true
- Start from all nodes where \( g \) is true and
- Perform backwards reachability analysis
- Each step backwards, store all nodes in \( Q \) where \( f \) is true
- Repeat the above step, until it converges
- \( Q \) contains all nodes satisfying \( E(f \cup g) \)

Function \( \text{Sat}(\phi : \text{Formula}): \text{set of State}; \)
(* precondition: \( \phi \) is true *)
begin
if \( \phi \) is true then return \( S \)
else return \( \emptyset \)
fi
\( \phi \in \text{AP} \rightarrow \text{return} \{ s | \phi \in \text{Label}(s) \} \)
\( \phi = \neg \phi_1 \rightarrow \text{return} \text{Sat}(\phi_1) \)
\( \phi = \phi_1 \lor \phi_2 \rightarrow \text{return} (\text{Sat}(\phi_1) \cup \text{Sat}(\phi_2)) \)
\( \phi = \text{EX} \phi_1 \rightarrow \text{return} \{ s \in S | (s, s') \in R \land s' \in \text{Sat}(\phi_1) \} \)
\( \phi = \text{E} [\phi_1 \cup \phi_2] \rightarrow \text{return} \text{Sat}_{E}(\phi_1, \phi_2) \)
\( \phi = A [\phi_1 \cup \phi_2] \rightarrow \text{return} \text{Sat}_{A}(\phi_1, \phi_2) \)
fi
(* postcondition: \( \text{Sat}(\phi) = \{ s | \mathcal{M}, s \models \phi \} \) *)
end

Function \( \text{Sat}_{E}(\phi, \psi : \text{Formula}): \text{set of State}; \)
(* precondition: \( \phi \) is true *)
begin
\( Q, Q' : \text{set of State}; \)
\( Q, Q' := \text{Sat}(\phi), \psi; \)
do
\( Q' := Q \)
\( Q := Q \cup \{ s | \exists s' \in Q, (s, s') \in R \land \text{Sat}(\phi) \} \)
do;
return \( Q \)
(* postcondition: \( \text{Sat}_{E}(\phi, \psi) = \{ s | \mathcal{M}, s \models E [\phi \cup \psi] \} \) *)
end

Table 3.4: Labelling procedure for \( E [\phi \cup \psi] \)
Algorithm ideas for checking $A(f \cup g)$

- Similar to the case for $A(f \cup g)$
- But each step backwards, store all nodes in $Q$ where $(f \lor g)$ is true, and the stored nodes do not lead to a node where $(f \lor g)$ is false
- Repeat the above step, until it converges
- $Q$ contains all nodes satisfying $A(f \cup g)$

### Fixpoint Characterizations

**$EF p \equiv p \lor EX EF p$**

or let $A$ be the set of states satisfying $EF p$ then

$$A \equiv p \lor EX A$$

In fact $A$ is the smallest one of sets satisfying the equations (the least fixpoint)

### Fixed points of monotonic functions

- Let $\tau$ be a function $S \to S$
- Say $\tau$ is monotonic when
  $$x \subseteq y \implies \tau(x) \subseteq \tau(y)$$
- Fixed point of $\tau$ is $y$ such that
  $$\tau(y) = y$$

If $\tau$ is monotonic, then it has
- least fixed point $\mu \tau(y)$
- greatest fixed point $\nu \tau(y)$

### Example: $EF p$

- $EF p$ is characterized by
  $$EF p = \mu \tau(p \lor EX y)$$
- Thus, it is the limit of the increasing series...

Note, since $S$ is finite, convergence is finite.
Example: $EG\ p$

- $EG\ p$ is characterized by
  \[ EG\ p = \forall y. \ (p \land EX\ y) \]
- Thus, it is the limit of the decreasing series...

Example, continued

$EF\ q = \mu y. (q \lor EX\ y)$

Remaining operators

- $AF\ p = \mu y. (p \lor AX\ y)$
- $AG\ p = \nu y. (p \land AX\ y)$
- $E(p \lor q) = \mu y. (q \lor (p \land EX\ y))$
- $A(p \lor q) = \mu y. (q \lor (p \land AX\ y))$

Complexity

The worst-case time complexity of checking whether system-model $sys$ satisfies the CTL-formula $\phi$ is $O(|S_{sys}|^2 \times |\phi|)$

However $|S_{sys}|$ may be EXPONENTIAL in number of parallel components!

- FIXPOINT COMPUTATIONS may be carried out using ROBDD’s (Reduced Ordered Binary Decision Diagrams)
  Bryant, 86

Branching time semantics

- Computation tree of an automaton is the unfolding of the automaton

Something more about Finite State Automata and Temporal Logics

(Continuation of Lecture 2)
Example (Branching Time)

Linear Time Semantics
- Sequences of transitions (or states)
  - set of possible executions of a system
- Suite best for closed systems

Equivalent and Preorders
- A equivalent to B if the tree of A is identical to the tree of B (Too strong!)
- A is simulated by B if every transition of A is simulated by a transition of B (simulation [Milner78])
- A and B are bisimilar if there is a symmetrical simulation between A’s and B’s states (bisimulation [Milner80])
- A and B are testing equivalent if they can pass the same set of tests (may and must testing [Nicola and Hennessy 84])
- A and B trace-equivalent if they provide the same set of sequences of transitions (trace equivalence [Hoare76])

LTL: Linear Time Logics
defined on infinite traces of Kripke structures with accepting conditions

Models: Infinite Sequences
(ω-language accepted by automata)
- Automata with accepting conditions
  - Buchi, Muller automata
- Infinite accepted sequences of transitions as semantics of automata
LTL: Syntax

- P
- not F
- F1 and F2
- O F (next time)
- F1 U F2 (Until)

LTL: semantics

- assume an automaton M
  - a sequence of M: t = s(0) → s(1) → s(2) → ... → s(i) ...
  - The set of sequences of M is Comp(M)
- s(i) sat p if p is a label of s(i)
- s(i) sat not F if not (s(i) sat F)
- s(i) sat F1 and F2 if s(i) sat F1 and S(i) sat F2
- s(i) sat O F if s(i+1) sat F
- s(i) sat F1 U F2 if s(k) sat F2 for some k=>i and s(j) sat F1 for all j such that i<=j<k

LTL: semantics (cont’d.)

- assume an automaton M
  - a sequence of M: t = s(0) → s(1) → s(2) → ... → s(i) ...
  - The set of sequences of M is Comp(M)
- t sat F iff s(0) sat F
- M sat F iff t sat F for all sequences t of Comp(M)

Derived Operators

- <>F denotes (true U F)
- [ ]F denotes not(<>not F)
- F1 W F2 denotes (F1 U F2) or [ ]F1 (weak Until-operator)

Model Checking LTL [Wolper et al 1986]

- Given an automata M and a formula F, to check M sat F
  - Construct the formula automaton: A~F
  - Construct the product automaton M || A~F (on-the-fly)
  - If M || A~F is empty then M sat F otherwise NO
  - Time-Complexity = |M|*2^|F|

Comparing CTL and LTL

- <> P (LTL) similar AF p (CTL)
- [] p (LTL) similar AG p (CTL)

However,

- LTL cannot express possibilities properties: EF P
- CTL cannot express <>[ ] p

- CTL* = LTL + CTL

The same idea can be used for CTL model checking using Tree-automata
Comparing CTL and LTL (contn.)

Satisfies $\langle\rangle[] p$
but it does not satisfy $AF AG p$

Why?

No subtree where $p$ is true everywhere

END
(Finite State Untimed Systems)