# Computer-Aided Program Design

Spring 2015, Rice University

Unit 4

Swarat Chaudhuri

February 17, 2015

# Symbolic model checking

- So far, we have seen:
  1. Bounded verification
  2. Explicit-state (or enumerative) verification.
- Better for finding bugs...
- Today: symbolic verification.

# Symbolic model checking

- Based on a *fixpoint computation*.
- Compute a sequence of sets of states $S_0, S_1, S_2, \ldots$, where $S_i$ consists of the set of states reached in $i$ steps from the initial states.
  1. Suppose you want to find the set of all states that have some path to a buggy state.
  2. Consider a function $Pre(S)$ that gives you the set of states that from which you can reach $S$ in 1 step.
  3. Then $S_0 = Bad$, and $S_{i+1} = S_i \cup Pre(S_i)$
  4. The set you want is a *fixpoint* of this transformation.

# Representing sets

- Question: how do you represent sets?

# Representing sets

- Question: how do you represent sets?
- Idea: as boolean formulas.
- Let us assume a CNF representation...
- Can you give an implementation of *Pre*?

# Representing sets

- Question: how do you represent sets?
- Idea: as boolean formulas.
- Let us assume a CNF representation...
- Can you give an implementation of *Pre*?
  - You have to use quantifiers...

$$Pre(\varphi) = \exists x' : T(x, x') \wedge \varphi(x').$$

  - But in the boolean world, you can rewrite the existential quantifier using disjunction:

$$\exists x' : \varphi = \varphi[x' \mapsto \top] \vee \varphi[x' \mapsto \bot]$$

# *Pre* and *Post*

- Backward vs. forward fixpoint computations
- What you saw above: backward fixpoint computation
- Forward computation would use *Post* rather than *Pre*.

# Representing sets of states

- CNF formulas have some merits, but...
- *Binary decision diagrams* are a classic data structure especially useful for this application.
- Less in vogue since the rise of SAT-solvers, but potentially useful for future applications.
- See `http://www.itu.dk/courses/AVA/E2005/bdd-eap.pdf` for more details.

# Binary decision diagrams (BDDs)

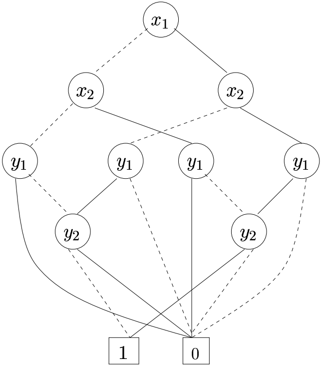
- Any boolean function $\varphi$ can be rewritten as a decision tree, where:
  - nodes are labeled by variables
  - if the root is labeled $x$, then the left and right subtrees off the root respectively represent the functions $\varphi[x \mapsto \bot]$ and $\varphi[x \mapsto \top]$
  - In other words, $\varphi$ is rewritten as $Ite(x, \varphi_1, \varphi_2)$, where *Ite* represents the if-then-else function.

# Binary decision diagrams (BDDs)

- Any boolean function $\varphi$ can be rewritten as a decision tree, where:
  - nodes are labeled by variables
  - if the root is labeled $x$, then the left and right subtrees off the root respectively represent the functions $\varphi[x \mapsto \bot]$ and $\varphi[x \mapsto \top]$
  - In other words, $\varphi$ is rewritten as $\mathit{Ite}(x, \varphi_1, \varphi_2)$, where *Ite* represents the if-then-else function.
- The BDD (technically, ROBDD) for the function is obtained by:
  - Setting an *order* on the variables
  - Removing nodes from which both edges lead to the same node
  - Coalescing isomorphic subtrees.

# Example

- Draw the BDD for $(x_1 \leftrightarrow x_2) \wedge (y_1 \leftrightarrow y_2)$
- Try the orders $(x_1 < x_2 < y_1 < y_2)$ and $(x_1 < y_1 < x_2 < y_2)$.




*BDD for order $(x_1 < x_2 < y_1 < y_2)$. The one for $(x_1 < y_1 < x_2 < y_2)$ is far larger.*

# BDDs

- For a fixed variable ordering, a function has a unique BDD.
- Finding the optimal ordering is a computationally hard problem (PSPACE-complete).
- Some functions, for example multiplication, have no polynomial-sized BDD.

# Operations on BDDs

How do we:

- Negate a BDD?

# Operations on BDDs

How do we:

- Negate a BDD?
- Check the satisfiability of a formula represented as a BDD?

# Operations on BDDs

How do we:

- Negate a BDD?
- Check the satisfiability of a formula represented as a BDD?
- Check the validity of a formula represented as a BDD?

# Operations on BDDs

How do we:

- Negate a BDD?
- Check the satisfiability of a formula represented as a BDD?
- Check the validity of a formula represented as a BDD?
- Check if two functions represented by BDDs are equivalent?

# Operations on BDDs

Suppose we are given BDDs for $\varphi_1$ and $\varphi_2$. Give the BDDs for:

- $\varphi_1 \vee \varphi_2$
- $\varphi_1 \wedge \varphi_2$.

# Operations on BDDs

Suppose we are given BDDs for $\varphi_1$ and $\varphi_2$. Give the BDDs for:

- $\varphi_1 \vee \varphi_2$
- $\varphi_1 \wedge \varphi_2$.

<u>Hint</u>: Use the following recurrence

$$Ite(x, f_1, f_2) \oplus Ite(x, g_1, g_2) = Ite(x, f_1 \oplus f_2, g_1 \oplus g_2)$$

if $\oplus$ is either $\vee$ or $\wedge$.

# Disjunction of BDDs using dynamic programming

```
function Apply_OR(f, g) =
   if T[f, g] ≠ empty:
        return T[f, g]
   else if f ∈ 0, 1 and  g ∈ 0, 1
        T[f, g] := f ∨ g
   else if var(f) = var(g):
         T[f,g] := BDD(var(f), Apply_OR (false(f), false(g)),
                                    Apply_OR(true(f), true(g)))
   else if var(f) < var(g):
         T[f,g] := BDD(var(f), Apply_OR (false(f), g),
                                    Apply_OR(true(f), g))
   else:
         T[f,g] := BDD(var(f), Apply_OR (f, false(g)),
                                    Apply_OR(f, true(g)))
   return T[f,g]
```

# Fixpoints and BDD

- How do we do fixpoint computation using BDDs?
- Core operations: *Pre* and *Post*.

# Fixpoints and BDD

- How do we do fixpoint computation using BDDs?
- Core operations: *Pre* and *Post*.
- Homework question: Give the most efficient algorithm you can think of to convert a boolean function given as CNF into a BDD.

# Simple example: two-bit counter

- Variables: $x, y$
- Initial condition: $\neg x \wedge \neg y$
- Transition relation:

$$((x' \leftrightarrow \neg x) \wedge (y' \leftrightarrow y)) \vee ((x' \leftrightarrow x) \wedge (y' \leftrightarrow \neg y))$$

- Goal: Find the set of states that from which a state that satisfies $\varphi \equiv \neg x \wedge \neg y$ is reachable.

# Exercise

Give a fixpoint-based algorithm to find all states from which all paths satisfy $\mathbf{F}\, q$.

# Exercise

Give a fixpoint-based algorithm to find all states from which all paths satisfy $\mathbf{F}\, q$.

Note that the above properties are *state properties* as opposed to *path properties*.

# CTL: a logic to express state properties

Let *Prop* be a set of *atomic propositions*. A formula $\varphi$ in Computation Tree Logic (CTL) has the form

$$\begin{array}{rcl} \varphi & ::= & p \mid \top \mid \bot \mid \neg\varphi_1 \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \\ & & \mid \mathbf{EX}\ \varphi_1 \mid \mathbf{E}(\varphi_1\, \mathbf{U}\, \varphi_2) \mid \mathbf{AX}\ \varphi_1 \mid \mathbf{A}(\varphi_1\, \mathbf{U}\, \varphi_2) \end{array}$$

where $p \in Prop$.

# CTL: a logic to express state properties

Let *Prop* be a set of *atomic propositions*. A formula $\varphi$ in Computation Tree Logic (CTL) has the form

$$\begin{array}{rcl} \varphi & ::= & p \mid \top \mid \bot \mid \neg\varphi_1 \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \\ & & \mid \mathbf{EX}\ \varphi_1 \mid \mathbf{E}(\varphi_1\, \mathbf{U}\, \varphi_2) \mid \mathbf{AX}\ \varphi_1 \mid \mathbf{A}(\varphi_1\, \mathbf{U}\, \varphi_2) \end{array}$$

where $p \in Prop$.

Semantics:

- $\mathbf{EX}\ \varphi$ means that there is a next state where $\varphi$ holds. (Same as $Pre(\varphi)$.)
- $AX\ \varphi$ means that for all next states, $\varphi$ holds.
- $\mathbf{E}(\varphi_1\, \mathbf{U}\, \varphi_2)$ means that there is a path from the current state that satisfies $(\varphi_1\, \mathbf{U}\, \varphi_2)$
- $\mathbf{A}(\varphi_1\, \mathbf{U}\, \varphi_2)$ means that on all paths from the current state, we have $(\varphi_1\, \mathbf{U}\, \varphi_2)$.

# CTL: a logic to express state properties

Let *Prop* be a set of *atomic propositions*. A formula $\varphi$ in Computation Tree Logic (CTL) has the form

$$\begin{array}{rcl} \varphi & ::= & p \mid \top \mid \bot \mid \neg\varphi_1 \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \\ & & \mid \mathbf{EX}\ \varphi_1 \mid \mathbf{E}(\varphi_1\, \mathbf{U}\, \varphi_2) \mid \mathbf{AX}\ \varphi_1 \mid \mathbf{A}(\varphi_1\, \mathbf{U}\, \varphi_2) \end{array}$$

where $p \in Prop$.

Semantics:

- **EX** $\varphi$ means that there is a next state where $\varphi$ holds. (Same as $Pre(\varphi)$.)
- $AX$ $\varphi$ means that for all next states, $\varphi$ holds.
- $\mathbf{E}(\varphi_1\, \mathbf{U}\, \varphi_2)$ means that there is a path from the current state that satisfies $(\varphi_1\, \mathbf{U}\, \varphi_2)$
- $\mathbf{A}(\varphi_1\, \mathbf{U}\, \varphi_2)$ means that on all paths from the current state, we have $(\varphi_1\, \mathbf{U}\, \varphi_2)$.
- Derived operators: **AF**, **AG**, **EG**, **EF**...

# CTL Verification

- Given: Symbolic transition system $\mathcal{M}$, CTL property $\varphi$.
- Question: Find the set of states $[\![\varphi]\!]$ in $\mathcal{M}$ that satisfies $\varphi$.

# Algorithm

- Induction on the formula.
- For instance,
$$[\![\varphi_1 \vee \varphi_2]\!] = [\![\varphi_1]\!] \cup [\![\varphi_2]\!].$$
- Challenge: Computing $[\![\varphi]\!]$ when $\varphi$ has temporal operators. How do we compute $[\![\mathbf{E}(\varphi_1 \,\mathbf{U}\, \varphi_2)]\!]$ when we have $[\![\varphi_1]\!]$ and $[\![\varphi_2]\!]$?

# Algorithm for $\mathbf{E}(\varphi_1 \, \mathbf{U} \, \varphi_2)$

1. Let $S := [\![\varphi_2]\!]$
2. Repeat until fixpoint:

$$S := S \cup ([\![\varphi_1]\!] \cap (\mathbf{EX}\ S))$$

# Algorithm for $\mathbf{E}(\varphi_1 \, \mathbf{U} \, \varphi_2)$

1. Let $S := [\![\varphi_2]\!]$
2. Repeat until fixpoint:

$$S := S \cup ([\![\varphi_1]\!] \cap (\mathbf{EX}\ S))$$

How do you do this for $\mathbf{A}(\varphi_1 \, \mathbf{U} \, \varphi_2)$?

# Algorithm for **EG** $\varphi$

- One possibility: use the identity **EG** $\varphi \equiv \neg$**AF** $(\neg\varphi)$.
- But can you give a direct algorithm for this property?

# Least and greatest fixpoints

- Consider transformations of the form $f : S_1 \mapsto S_2$, where $S_1$ and $S_2$ are sets of states
- Fixpoint of $f$: a solution to the equation $f(X) = X$.
- But an equation can have multiple solutions... think of $X = Pre(X)$.

# Least and greatest fixpoints

- Consider transformations of the form $f : S_1 \mapsto S_2$, where $S_1$ and $S_2$ are sets of states
- Fixpoint of $f$: a solution to the equation $f(X) = X$.
- But an equation can have multiple solutions... think of $X = Pre(X)$.
- In particular, we are interested in the *least* and *greatest* fixpoints.

# Knaster-Tarski theorem

- Let $U$ be the universe of states
- Assume that $f(X) : 2^U \to 2^U$ is a *monotone* function.
- In that case, $f(X)$ has a unique least fixpoint and a unique greatest fixpoint

# Knaster-Tarski theorem

- Let $U$ be the universe of states
- Assume that $f(X) : 2^U \to 2^U$ is a *monotone* function.
- In that case, $f(X)$ has a unique least fixpoint and a unique greatest fixpoint
- Moreover:
  - **lfp**.$f(X)$ appears in the sequence
  $$\emptyset, f(\emptyset), f(f(\emptyset)), \ldots$$
  - **gfp**.$f(X)$ appears in the sequence
  $$U, f(U), f(f(U)), \ldots$$
  - **lfp**.$f(X)$ is usually written as $\mu X.f$, and **gfp**.$f(X)$ is usually written as $\nu X.f$, when $f$ is represented by a formula.

# CTL properties as least and greatest fixpoints

Express the following properties in terms of least and greatest fixpoint operators:

- $\mathbf{EF}\ \varphi$
- $\mathbf{EG}\ \varphi$
- $\mathbf{E}(\varphi_1\ \mathbf{U}\ \varphi_2)$

# The modal $\mu$-calculus: a logic of fixpoints

Let *Prop* be a set of *propositions*, and *Var* a set of *variables*. A formula $\varphi$ in the modal $\mu$-calculus has the form

$$\begin{aligned} \varphi \quad ::= \quad & p \mid \neg p \mid Z \mid \top \mid \bot \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \\ & \mid \mathbf{EX}\ \varphi_1 \mid \mathbf{AX}\ \varphi_1 \mid \mu Z.\varphi_1 \mid \nu Z.\varphi_1 . \end{aligned}$$

where $p \in Prop$ and $Z \in Var$.
[See
http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.33.916]

# The modal $\mu$-calculus: Semantics

- (Set-valued) variables appearing in formulas $\varphi$ can be either *free* or *bound*.
- In $(Z \vee p)$, $Z$ is a free variable.
- A formula $\varphi(Z)$ with a free variable $Z$ can be viewed as a transformer from sets to sets.
- $\varphi[Z := T]$ = set of states that satisfy $\varphi$ if $Z$ is replaced by $T$.

# The modal $\mu$-calculus: Semantics

The semantics of a formula $\varphi$ is the set of states $S_\varphi$ where it is satisfied. These sets are defined inductively. For example:

- $[\![\varphi_1 \wedge \varphi_2]\!] = [\![\varphi_1]\!] \cap [\![\varphi_2]\!]$
- $[\![\varphi_1 \vee \varphi_2]\!] = [\![\varphi_1]\!] \cup [\![\varphi_2]\!]$
- $[\![\mu Z.\varphi(Z)]\!] =$ intersection of all sets $T$ such that $\varphi[Z := T] \subseteq T$
- $[\![\nu Z.\varphi(X)]\!] =$ union of all sets $T$ such that $T \subseteq \varphi[Z := T]$.

# Questions

- Why is it necessary that the fixpoints exist?

# Questions

- Why is it necessary that the fixpoints exist?
- How do you negate a formula?

# Questions

- Why is it necessary that the fixpoints exist?
- How do you negate a formula?
- How do you compile CTL formulas into the $\mu$-calculus?

# Benefits of the $\mu$-calculus: Games

- Suppose you have a symbolic transition system that is interacting with a hostile environment.
- The states of the system are divided into two partitions: the *system states* and the *environment states*. System states are controlled by the system; environment states by the environment.
- System's goal: to reach a set $G$ of good states no matter what the environment does.
- Objective in the verification problem: find the set of states from which the system has a strategy to reach $G$.

# Expressiveness of the $\mu$-calculus

- The $\mu$-calculus allows nesting of fixpoint quantifiers. This allows the expression of really complex properties. For example:

$$\nu Z.\mu Y.((p \wedge Z) \vee \mathbf{EX} Y).$$

  This formula has *alternation depth* 2.
- We make a distinction between alternation and nesting depth. All CTL formulas have alternation depth 1.
- The fragment of the $\mu$-calculus with alternation depth 1 is said to be *alternation-free*).

# Expressiveness of the $\mu$-calculus

- The $\mu$-calculus allows nesting of fixpoint quantifiers. This allows the expression of really complex properties. For example:

  $$\nu Z.\mu Y.((p \wedge Z) \vee \mathbf{EX}Y).$$

  This formula has *alternation depth* 2.
- We make a distinction between alternation and nesting depth. All CTL formulas have alternation depth 1.
- The fragment of the $\mu$-calculus with alternation depth 1 is said to be *alternation-free*).
- The $\mu$-calculus includes LTL as well, but alternation depth 2 is required.

# Model checking the $\mu$-calculus

Model checking requires the use of an *environment Env* that maps free variables in formulas to sets.
Fragment of the algorithm:

```
eval(φ, Env):
...
if φ = μZ.ψ(Z) then
    S := ∅
    repeat
        T := S
        S := eval(ψ,Env[Z := S])
    until S = T
...
```

# Complexity

- Each loop executes at most $O(n)$ times ($n$ is the total number of states)
- Each iteration does a recursive call to `eval` with a different value of the fixpoint variable.
- Complexity: $O(|\varphi| \times n^k)$, where $k$ is the maximum nesting depth of fixpoint operators.

# Complexity

- Each loop executes at most $O(n)$ times ($n$ is the total number of states)
- Each iteration does a recursive call to `eval` with a different value of the fixpoint variable.
- Complexity: $O(|\varphi| \times n^k)$, where $k$ is the maximum nesting depth of fixpoint operators.
- There is a better algorithm (by Emerson and Lei) that reduces the complexity to $O(|\varphi| \times n^d)$, where $d$ is the alternation depth of $\varphi$.

# Complexity

- Each loop executes at most $O(n)$ times ($n$ is the total number of states)
- Each iteration does a recursive call to `eval` with a different value of the fixpoint variable.
- Complexity: $O(|\varphi| \times n^k)$, where $k$ is the maximum nesting depth of fixpoint operators.
- There is a better algorithm (by Emerson and Lei) that reduces the complexity to $O(|\varphi| \times n^d)$, where $d$ is the alternation depth of $\varphi$.
- Does there exist a PTIME algorithm? An open question.
  - Known to be in NP $\cap$ co-NP.

# The $\mu$-calculus as a language for expressing data flow analyses

- Very busy expressions: Set of expressions that are used without modification along all possible program paths from the present point.
- Live variables: Set of variables that are used in some future point in an execution.

[See "Data flow analysis is model checking of abstract interpretations," By David Schmidt, http://dl.acm.org/citation.cfm?id=268950.]