EECS 219C: Computer-Aided Verification

Symbolic Model Checking Part I

Sanjit A. Seshia

EECS, UC Berkeley

Announcement

- Extra lecture on Friday, 11 am – 12:30 pm in 540 Cory

Today's Lecture

$\boxed{\text{Symbolic}}$ model checking $\boxed{\text{with BDDs}}$

Manipulate sets (of states and transitions) rather than individual elements and represent sets as Boolean formulas

Represent Boolean formulas as BDDs

Today's Lecture

- Symbolic model checking
  - Basics of symbolic representation
  - Quantified Boolean formulas (QBF)
  - Checking G p
  - Fixpoint theory
  - Checking CTL properties

Sets as Boolean functions

- Every finite set can be represented as a Boolean function
  - Suppose the set has N (> 0) elements
  - Each element is encoded as a string of at least $\lceil \log N \rceil$ bits
  - Characteristic Boolean function is the one whose ON-set (satisfying assignments) are those strings
  - Empty set is "False"

Set Operations as Boolean Operations

- $A \cup B = ?$
- $A \cap B = ?$
- $A \subset B = ?$
- Is A empty?

Sets of states and transitions

- Set of states → each state s is bit-string comprising values of state variables
- Set of transitions →
  - Transition is a state pair (s, s')
  - View the pair as a combined bit-string
- From now, we will view the set of states S and the transition relation R as Boolean formulas over vector of current state variables v and next state variables v'
  - S(v), R(v, v')

Quantified Boolean Formulas

- Let F denote a Boolean formula, and v denote one or more Boolean variables
- A quantified Boolean formula ϕ is obtained as:

$$\phi ::= F \mid \exists v\, \phi \mid \forall v\, \phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \neg \phi$$

- How do you express $\exists v_i\, \phi$ and $\forall v_i\, \phi$ in terms of ϕ's cofactors and standard Boolean operators?

Symbolic Model Checking G p

- Given: Set of initial states S_0, transition relation R
- Check property G p (or AG p)
- How symbolic model checking will do this:
  - Compute $S_0, S_1, S_2, \ldots$ where S_i is the set of states reachable from some initial state in at most i steps
    - What kind of search is this: DFS or BFS?
    - When do we stop?
  - After computing each S_i, check whether any element of S_i satisfies $\neg$ p [How?]
    - How do we generate a counterexample?

Reachability Analysis

- The process of computing the set of states reachable from some initial state in 0 or more steps
  - Often characterized as checking (AG true)
  - The resulting set is called "reachable set" or "set of reachable states"
    - This is the "strongest invariant" of the system → WHY? What is a "system invariant"?

Implementing Reachability Analysis

- How is S_i related to S_{i+1}?
  - In words
  - As a recurrence relation using QBF

Implementing Reachability Analysis

- How is S_i related to S_{i+1}?
- $v \in S_{i+1}$ iff $v \in S_i$ or there is a state $x \in S_i$ such that $R(x, v)$
- $S_{i+1}(v) = S_i(v) \vee \exists x \{ S_i(x) \wedge R(x,v) \}$

Implementing Reachability Analysis

- How is S_i related to S_{i+1}?
- $v \in S_{i+1}$ iff $v \in S_i$ or there is a state $x \in S_i$ such that $R(x, v)$
- $S_{i+1}(v) = S_i(v) \vee \exists x \{ S_i(x) \wedge R(x,v) \}$
- $S_{i+1}(v) = S_i(v) \vee (\exists v \{ S_i(v) \wedge R(v,v') \}) [v/v']$
  - $F[x/y]$ means that we substitute x for y in F

Implementing Reachability Analysis

i := 0;
do {
  i++;
  $S_i(v) = S_{i-1}(v) \vee (\exists v \{ S_{i-1}(v) \wedge R(v,v') \}) [v/v']$
} while $(S_i(v) \mathrel{!=} S_{i-1}(v))$
$S_i(v)$ is the set of reachable states

BDD Issues

- Remember that S_i and R are represented as BDDs
- How large they grow determines the space and time usage of the algorithm

Backwards Reachability

- Suppose we want to verify G p
- The formula $\neg$ p characterizes all error states
- We can search backwards for a path to an error state from some initial state
  - Compute E_0, E_1, E_2, … as states reachable from the error states in at most 0, 1, 2, … steps
  - $E_0 = \neg p$
  - How to express E_{i+1} in terms of E_i ?
- Why would we want to do backwards reachability analysis? Is it always better?

Verification of G p

- Corresponding CTL formula is AGp
- with Forward Reachability Analysis:
  - Check if some $S_i \wedge \neg p$ is true
- with Backward Reachability Analysis:
  - Set $E_0 = \neg p$
  - Check if $E_k \wedge S_0$ is true for any k

S. A. Seshia 17

Symbolic Model Checking, General Case

- We will consider properties in CTL
  - As implemented in the original SMV model checker
  - Later we will see how LTL properties can be verified using symbolic techniques

S. A. Seshia 18

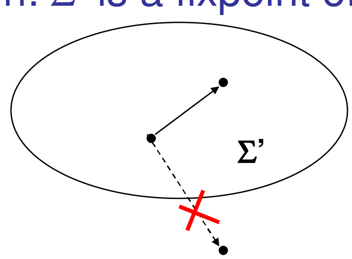
Model Checking Arbitrary CTL

- Need only consider the following types of CTL properties:
  - E X p
  - E G p
  - E (p U q)

- Why? $\leftarrow$ all others are expressible using above
  - A G p = ?
  - A G (p $\rightarrow$ (A F q)) = ?

Fixpoint (Fixed point)

- Let Σ be a set, and $\Sigma' \subseteq \Sigma$
  - In model checking, Σ = True
- Let $\tau : P(\Sigma) \rightarrow P(\Sigma)$
- Definition: Σ' is a fixpoint of τ if $\tau(\Sigma') = \Sigma'$



Example

- What's an example of a fixpoint we've seen already? What was τ?

S. A. Seshia 21

Example

- What's an example of a fixpoint we've seen already? What was τ?
  - A G true can be computed using a fixpoint formulation
  - τ computes the "next state"
- What we need: a way to generalize this for arbitrary CTL properties: EX, EG, EU
  - Fixpoint theory helps us do this

S. A. Seshia 22

More Definitions

- τ is *monotonic* if for $P \subseteq Q$, $\tau(P) \subseteq \tau(Q)$
- τ is $\cup$-*continuous* if: $P_1 \subseteq P_2 \subseteq P_3 \ldots \rightarrow \tau(\cup_i P_i) = \cup_i \tau(P_i)$
- τ is $\cap$-*continuous* if: $P_1 \supseteq P_2 \supseteq P_3 \ldots \rightarrow \tau(\cap_i P_i) = \cap_i \tau(P_i)$

Main Theorems (Tarski)

- τ is *monotonic* if for $P \subseteq Q$, $\tau(P) \subseteq \tau(Q)$
- τ is $\cup$-*continuous* if: $P_1 \subseteq P_2 \subseteq P_3 \ldots \rightarrow \tau(\cup_i P_i) = \cup_i \tau(P_i)$
- τ is $\cap$-*continuous* if: $P_1 \supseteq P_2 \supseteq P_3 \ldots \rightarrow \tau(\cap_i P_i) = \cap_i \tau(P_i)$

- A monotonic τ on $P(\Sigma)$ always has
  - a least fixpoint: written $\mu Z.\ \tau(Z)$
  - a greatest fixpoint: written $\nu Z.\ \tau(Z)$

Main Theorems (Tarski)

- τ is *monotonic* if for $P \subseteq Q$, $\tau(P) \subseteq \tau(Q)$
- τ is $\cup$-*continuous* if: $P_1 \subseteq P_2 \subseteq P_3 \ldots \Rightarrow \tau(\cup_i P_i) = \cup_i \tau(P_i)$
- τ is $\cap$-*continuous* if: $P_1 \supseteq P_2 \supseteq P_3 \ldots \Rightarrow \tau(\cap_i P_i) = \cap_i \tau(P_i)$

- A monotonic τ on $P(\Sigma)$ always has
  - a least fixpoint: written $\mu Z. \tau(Z)$
  - a greatest fixpoint: written $\nu Z. \tau(Z)$
  - $\mu Z. \tau(Z) = \cap \{ Z \mid \tau(Z) \subseteq Z \}$
  - $\nu Z. \tau(Z) = \cup \{ Z \mid \tau(Z) \supseteq Z \}$

S. A. Seshia

25

Main Theorems (Tarski)

- τ is *monotonic* if for $P \subseteq Q$, $\tau(P) \subseteq \tau(Q)$
- τ is $\cup$-*continuous* if: $P_1 \subseteq P_2 \subseteq P_3 \ldots \Rightarrow \tau(\cup_i P_i) = \cup_i \tau(P_i)$
- τ is $\cap$-*continuous* if: $P_1 \supseteq P_2 \supseteq P_3 \ldots \Rightarrow \tau(\cap_i P_i) = \cap_i \tau(P_i)$
- A monotonic τ on $P(\Sigma)$ always has
  - a least fixpoint: written $\mu Z. \tau(Z)$
  - a greatest fixpoint: written $\nu Z. \tau(Z)$
  - $\mu Z. \tau(Z) = \cap \{ Z \mid \tau(Z) \subseteq Z \}$
  - $\nu Z. \tau(Z) = \cup \{ Z \mid \tau(Z) \supseteq Z \}$
  - $\mu Z. \tau(Z) = \cup_i \tau^i(\phi)$ when τ is $\cup$-*continuous*
  - $\nu Z. \tau(Z) = \cap_i \tau^i(\Sigma)$ when τ is $\cap$-*continuous*

S. A. Seshia

26

Main Lemma for us

- If Σ is finite and τ is monotonic, then τ is also $\cup$-continuous and $\cap$-continuous
- Proof? (of $\cup$-continuous)

τ is $\cup$-*continuous* if: $P_1 \subseteq P_2 \subseteq P_3 \ldots \Rightarrow \tau(\cup_i P_i) = \cup_i \tau(P_i)$

What's Left?

- We have the needed fixpoint theory
- Now all we need to do is formulate the result of CTL operators as fixpoints
  - We will identify a CTL formula with the set of states that satisfy that formula
    - Remember that CTL formulas start with A or E which are interpreted over states, not runs

CTL Results as Fixpoints

- $A\,G\,p = \nu\, Z.\ p \wedge AX\,Z$
  - $\tau(Z) = p \wedge AX\,Z$
  - Given a point (state) in Z, τ maps it to another state that
    - Satisfies p
    - Can reach a state in Z along any execution path in one step
    - So what happens when we reach τ's fixpoint?
  - Remember: ν fixpoint computation starts with the universal set Σ and works 'downward'

Other Fixpoint Formulations

- $AF\,p = \mu\, Z.\ p \vee AX\,Z$
- $EG\,p = \nu\, Z.\ p \wedge EX\,Z$
- $E(p\ U\ q) = \mu\, Z.\ q \vee (p \wedge EX\,Z)$

- Intuitively:
  - Eventualities → least fixpoints
  - Always/Forever → greatest fixpoints

Model Checking CTL Properties

- We define a general recursive procedure called "Check" to do the fixpoint computations
- Definition of Check:
  - Input: A CTL property Π (and implicitly, R)
  - Output: A Boolean formula B representing the set of states satisfying Π
- If $S_0(v) \rightarrow B(v)$, then Π is true

The "Check" procedure

Cases:

- If Π is a Boolean formula, then Check(Π) = Π
- Else:
  - Π = EX p, then Check(Π) = CheckEX(Check(p))
  - Π = E(p U q), then
    Check(Π) = CheckEU(Check(p), Check(q))
  - Π = E G p, then Check(Π) = CheckEG(Check(p))
- Note: What are the arguments to CheckEX, CheckEU, CheckEG? CTL properties or Boolean formulas?

CheckEX

- CheckEX(p) returns a set of states such that p is true in their next states
- How to write this?

CheckEU

- CheckEU(p, q) returns a set of states, each of which is such that
  - Either q is true in that state
  - Or p is true in that state and you can get from it to a state in which p U q is true

CheckEU

- CheckEU(p, q) returns a set of states, each of which is such that
  - Either q is true in that state
  - Or p is true in that state and you can get from it to a state in which p U q is true
- Let Z_0 be our initial approximation to the answer to CheckEU(p, q)
- $Z_k(v) = \{ q(v) + [p(v) . \exists v' \{ R(v, v') . Z_{k-1}(v') \}] \}$
- What's Z_0? Why will this terminate?

Summary

- EGp computed similarly
- Definition of Check:
  - Input: A CTL property Π (and implicitly, R)
  - Output: A Boolean formula B representing the set of states satisfying Π
- All Boolean formulas represented "symbolically" as BDDs
  - "Symbolic Model Checking"

Next class

- More on symbolic model checking
- Start topics on “abstraction”