EECS 219C: Computer-Aided Verification

Symbolic Model Checking
Part I

Sanjit A. Seshia

EECS, UC Berkeley

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 $\lceil \log M \rceil$ bits, where M is the number of elements in the universe
  - Characteristic Boolean function is the one whose ON-set (satisfying assignments) are those strings
  - Empty set is “False”

Set Operations as Boolean Operations

- A ^ B = ?

- A _ B = ?

- A ¯ 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:

$$\phi ::= F \mid < v\ \phi \mid ;\ v\ \phi \mid \phi\ a\ \phi \mid \phi\ b\ \phi \mid =\ \phi$$

- How do you express $< v_i\ \phi$ and $;\ v_i\ \phi$ 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, … 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 = 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 $R(x, v)$
- $S_{i+1}(v) = S_i(v) \lor \exists x \{ S_i(x) \land R(x,v) \}$

Implementing Reachability Analysis

- How is S_i related to S_{i+1}?
- v 5 S_{i+1} iff v 5 S_i or there is a state x 5 S_i such that R(x, v)
- S_{i+1}(v) = S_i(v) b < x { S_i(x) a R(x,v) }
- S_{i+1}(v) = S_i(v) b (< v { S_i(v) a R(v,v') }) [v/v']
  - F[x/y] means that we substitute x for y in F

Implementing Reachability Analysis

$i := 0;$

do $\{$

  $i{+}{+};$

  $S_i(v) = S_{i-1}(v)$ b $(< v \{ S_{i-1}(v)$ a $R(v,v') \})\ [v/v']$

$\}$ while $(S_i(v)\ != S_{i-1}(v))$

$S_i(v)$ 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 = 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 = = 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 a = p is true
- with Backward Reachability Analysis:
  - Set E_0 = = p
  - Check if E_k a S_0 is true for any k

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

Model Checking Arbitrary CTL

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

- Why? ← all others are expressible using above
  - A G p = ?
  - A G (p → (A F q)) = ?

Fixpoint Theory

- Theory about elements/points that are unchanged by application of a function (hence “fixed point”)
- A concept from mathematics and denotational semantics of programming languages
- For this class: Theoretical concepts and results that will help us design algorithms for CTL model checking

Fixpoint (Fixed point)

- Let Σ be a set (the "universe"), and Σ' » Σ
  - In model checking, Σ = True
- Let $\tau : P(\Sigma) \rightarrow P(\Sigma)$
  - $P(\Sigma)$ is the power set of Σ

- Definition: Σ' is a fixpoint of τ if $\tau(\Sigma') = \Sigma'$

Example of Fixpoint

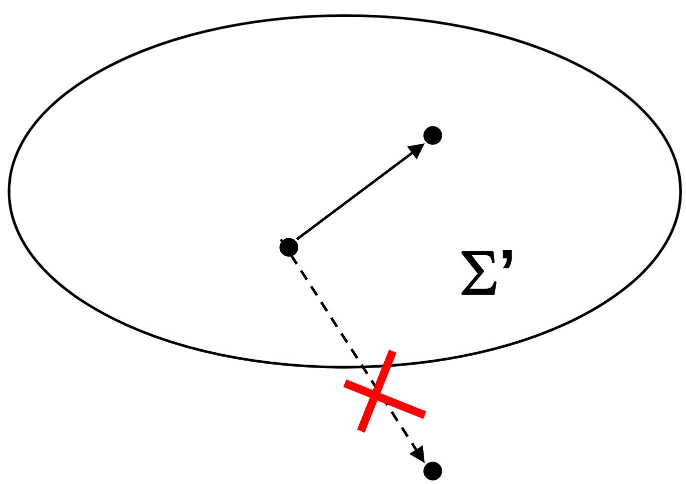
- Let
  - $\Sigma = \{s_0, s_1\}$
  - $\tau(Z) = Z$ ^ $\{s_0\}$, Z » Σ

- What is a fixpoint of τ? Is there only one?

Model Checking Example

In the context of Reachability Analysis:

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



Model Checking 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

More Definitions

- τ is *monotonic* if for P » Q, τ(P) » τ(Q)
- τ is *^-continuous* if: P_1 » P_2 » P_3 … ➔ $\tau(\wedge_i P_i) = \wedge_i \tau(P_i)$
- τ is *_-continuous* if: P_1 … P_2 … P_3 … ➔ $\tau(\__i P_i) = \__i \tau(P_i)$

Main Theorems (Tarski)

- τ is *monotonic* if for P » Q, τ(P) » τ(Q)
- τ is *^-continuous* if: P_1 » P_2 » P_3 … ➔ τ(^$_i$ P_i) = ^$_i$ τ(P_i)
- τ is *_-continuous* if: P_1 … P_2 … P_3 … ➔ τ(_$_i$ P_i) = _$_i$ τ(P_i)

- A monotonic τ on $P(\Sigma)$ always has
  - a least fixpoint: written μ Z. τ(Z)
  - a greatest fixpoint: written ν Z. τ(Z)
  - least and greatest refer to the size of the fixpoint Z.

Least and Greatest Fixpoints

- Let
  - $\Sigma = \{s_0, s_1\}$
  - $\tau(Z) = Z$ ^ $\{s_0\}$, Z » Σ
- What is the least fixpoint of τ? The greatest fixpoint? Are they the same?

Main Theorems (Tarski)

- τ is *monotonic* if for P » Q, τ(P) » τ(Q)
- τ is ^-*continuous* if: P_1 » P_2 » P_3 … ➔ τ(^$_i$ P_i) = ^$_i$ τ(P_i)
- τ is _-*continuous* if: P_1 … P_2 … P_3 … ➔ τ(_$_i$ P_i) = _$_i$ τ(P_i)

- A *monotonic* τ on *P*(Σ) always has
  - a least fixpoint: written μ Z. τ(Z)
  - a greatest fixpoint: written ν Z. τ(Z)
  - μ Z. τ(Z) = _#{ Z | τ(Z) » Z }
  - ν Z. τ(Z) = ^#{ Z | τ(Z) … Z }

Main Theorems (Tarski)

- τ is *monotonic* if for P » Q, τ(P) » τ(Q)
- τ is ^-*continuous* if: P_1 » P_2 » P_3 … ➔ $\tau(\hat{}_i\, P_i) = \hat{}_i\, \tau(P_i)$
- τ is _-*continuous* if: P_1 … P_2 … P_3 … ➔ $\tau(\__i\, P_i) = \__i\, \tau(P_i)$
- A *monotonic* τ on $P(\Sigma)$ always has
  - a least fixpoint: written μ Z. τ(Z)
  - a greatest fixpoint: written ν Z. τ(Z)
  - μ Z. τ(Z) = _#{ Z | τ(Z) » Z }
  - ν Z. τ(Z) = ^#{ Z | τ(Z) … Z }
  - μ Z. τ(Z) = $\hat{}_i\, \tau^i(>)$ when τ is ^-*continuous*
  - ν Z. τ(Z) = $\__i\, \tau^i(\Sigma)$ when τ is _-*continuous*

Main Lemma for us

- If Σ is finite and τ is monotonic, then τ is also ^-continuous and _-continuous
- Proof? (of ^-continuous)

τ is *^-continuous* if: P_1 » P_2 » P_3 … ➔ τ(^$_i$ P_i) = ^$_i$ $\tau(P_i)$

Next Steps

- 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 = ν Z. p a AX Z
  - τ(Z) = p a 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

- EF p = μ Z. p b EX Z
- EG p = ν Z. p a EX Z
- E(p U q) = μ Z. q b (p a 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?

$$< x \; [\; p(x) \;.\; R\,(s, x) \;]$$

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)

- $Z_k(v) = \{ q(v) \ + [p(v) \ . \ < v' \{ R(v, v') \ . \ Z_{k-1}(v') \} \ ] \}$

- 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"

Counterexample/Witness Generation for CTL

- Counterexample = run showing how the property is violated
  - – Formulas with universal path quantifier A
- Witness = run showing how the property is satisfied
  - – Formulas with existential path quantifier E
  - – Can also view as counterexample for the negated property
    - E.g. E G p and A F = p

Witness Generation for EG p

- Fixpoint formulation for E G p:
  - $\nu Z \,.\, p$ a EX Z
  - $\tau(Z) = p$ a EX Z
- Fixpoint computation yields sequence $Z_0, Z_1, \ldots, Z_k$
  - Z_0 = True (universal set)
  - $Z_1 = \tau(\text{True}) = ?$
  - each Z_i is a BDD representing a set of states
  - How would you describe an element of Z_i ?
- We need to generate the counterexample from $S_0, R, Z_0, Z_1, \ldots, Z_k$

Witness Generation for EG p

- Fixpoint computation yields sequence $Z_0, Z_1, \ldots, Z_k$
  - A state in Z_i ($i > 0$) satisfies p and there is a path of length i-1 from that state comprising states satisfying p
  - How would you describe an element of Z_k ?
    - Remember: it's the fixpoint

Witness Generation for EG p

- Fixpoint computation yields sequence $Z_0, Z_1, \ldots, Z_k$
  - A state in Z_i satisfies p and there is a path of length i-1 from that state comprising states satisfying p
  - How would you describe an element of Z_k ?
    - State in Z_k has path from it of length k-1 or more (including a cycle) with all states satisfying p
    - If S_0 is contained in Z_k, any initial state has such a path

Witness Generation for EG p

- Let s_0 be an initial state with a desired witness path
  - We need to reproduce one such witness
  - How can we do this?

Witness Generation for EG p

- Let s_0 be an initial state with a desired witness path
  - We need to reproduce one such witness
  - How can we do this?
    - Main insight: desired successor of s_0 also satisfies EG p, and so on
    - Look for a cycle in such a computed chain
      - Why should there be a cycle?

Fairness

- A computation path is defined as fair if a fairness constraint p is true infinitely often along that path
  - Fairness constraint is a state predicate
  - Generalized to set of fairness constraints $\{p_1, p_2, \ldots, p_k\}$ by requiring each element of the subset to be true infinitely often
- Example: Every process in an asynchronous composition must be scheduled infinitely often