Today’s Lecture

- Abstraction in Model Checking
  - Interpolation-based model checking
- Automata-based Property Specification
  - Properties as (Buchi) automata
  - Notions of Trace Containment, Simulation, Bisimulation, Refinement
Abstract/Concrete Error Trace

1. Abstract trace OK
2. Abstract trace spurious

Counterexample Guided Abstraction-Refinement (CEGAR)

1. Create abstraction A
2. Perform (unbounded) model checking on A
3. Prove that this abstract counterexample of length k is a concrete counterexample using k-step BMC on M
4. Extract information for refinement from refutation
5. Property true
6. Counterexample of length k
7. Proof succeeds
8. Proof fails
9. OK
10. Counterexample
Proof-based Abstraction (PBA) [McMillan, Amla, 2003]

- BMC on M at depth k
  - Cex?
    - Counterexample
  - No Cex?
    - Use refutation to choose abstraction
      - Property true?
        - OK
      - False, counterexample of length k’?
    - Increase k to k’

Abstraction and Reachability

- An abstraction expands the set of states reachable from the initial state
  - OVER-APPROXIMATION
- Instead of starting by abstracting states, one can directly abstract the transition relation
  - Each time you compute the set of next states, you get an over-approximation of the actual set of next states
  - Gives a way of computing an over-approximation of the set of reachable states
Abstraction using Interpolation

- Abstraction is extracting sufficient/relevant information from a system to prove a given property.
- This notion is in some sense closely related to a notion of “interpolant” and a lemma called “Craig’s interpolation lemma”

Interpolation Lemma (Craig, 57)

- If $A \land B = \text{false}$, there exists an interpolant $A'$ for $(A,B)$ such that:
  
  $A \Rightarrow A'$
  
  $A' \land B = \text{false}$
  
  $A'$ refers only to common variables of $A, B$

- Example:
  
  $A = p \land q$, $B = \neg q \land r$, $A' = q$
Interpolants from Proofs (Pudlak, Krajicek, 97)

- Interpolant $A'$ for $A \land B$:
  
  $A \Rightarrow A'$
  
  $A' \land B = \text{false}$
  
  $A'$ refers only to common variables of $A, B$

- Interpolants can be obtained from proofs
  
  - given a resolution-based refutation (proof of unsatisfiability) of $A \land B$,
  
  $A'$ can be derived in time linear in the proof

Interpolation based Model Checking (McMillan, 2003)

- Main Idea: Pose the problem of over-approximating the set of next states as finding an interpolant

\[
S_0(v_0) \land R(v_0, v_1) \land R(v_1, v_2) \land \ldots \land R(v_{k-1}, v_k) \land E_k(v_k)
\]
Interpolation based Model Checking

\[ S_0(v_0) \land R(v_0, v_1) \land R(v_1, v_2) \land \ldots \land R(v_{k-1}, v_k) \land E_k(v_k) \]

\[ A = S_0(v_0) \land R(v_0, v_1) \]

\[ B = R(v_1, v_2) \land \ldots \land R(v_{k-1}, v_k) \land E_k(v_k) \]

\[ A' \text{ is a function of } v_i \text{ s.t.} \]
\[ 1. \ A \Rightarrow A' \]
\[ 2. \ A' \land B \text{ is unsat} \]

What set of states does \( A' \) represent?

Interpolation based MC

For a fixed \( k \):

1. Set \( Z \) initially to \( S_0 \)
2. Do BMC starting from \( Z \) for \( k \) steps
   - If SAT: have we found a counterexample?
   - If UNSAT, continue
3. Use interpolation to compute over-approximation of next states of \( Z \) and add them back into \( Z \)
   - Can newly added states lead to error states in \( k-1 \) steps? In \( k \) steps?
4. If \( Z \) does not increase
   - We’ve reached a fixed point. Is the property true?
5. Otherwise, back to step 2
Intuition

- A' tells us everything the prover deduced about the image of \( S_0 \) in proving it can't reach an error in \( k \) steps.
- Hence, A' is in some sense an abstraction of the image relative to the property and the bound \( k \).

Refinement

- Model checking may fail for a fixed \( k \)
  - May add a state that reaches error in \( k \) steps (getting SAT in step 2 with \( Z \neq S_0 \))
- Refinement is just increasing \( k \)
  - How big can \( k \) get?
Proof-based Abstract. vs Interpolation

Properties as Automata

- Often properties themselves are finite-state machines
  - E.g. two versions of the same system, an optimized “implementation”, and a simple-and-correct “specification”
- How do we formalize the notion of “implementation satisfies specification”? 
Properties as Automata

• Often your properties themselves are finite-state machines
  – E.g. two versions of the same system, an optimized “implementation”, and a simple-and-correct “specification”

• How do we formalize the notion of “implementation satisfies specification”?
  – All behaviors (traces) of the implementation are also traces of the specification

TRACE CONTAINMENT
(traces are projected over a common set of atomic propositions)

Abstraction A and Original System M

• All traces of M are also traces of A
• If A satisfies an LTL property, does M also satisfy that property?
• How about for CTL*?
Abstraction A and Original System M

- All traces of M are also traces of A
- So any LTL property that A satisfies will also be satisfied by M
- Holds good for any CTL* property that
  - Has all negations appearing only over atomic propositions
  - Has only the “A” quantifier, not the “E” quantifier
  - ACTL*

Simulation --- Intuition

- Two finite state machines M and M’
- M’ simulates M if
  - M’ can start in a similarly labeled state as M
  - For every step that M takes from s to t, M’ can mimic it by stepping to a state with similar label as t
Simulation

• M = (S, S₀, R, L) and M' = (S', S₀', R', L')
• A relation H ⊆ S x S' is a simulation relation between M and M' means that:
  For all (s, s'), if H(s, s') then:
  – L'(s') = L(s) ∩ AP'
  – For every state t s.t. R(s, t) there is a state t' such that R'(s', t') and H(t, t')
• M' simulates M if
  – there exists a simulation relation H between them, and
  – For each s₀ ∈ S₀, there exists s₀' ∈ S₀' s.t. H(s₀, s₀')

Simulation and Trace Containment

Are they the same? If not, which implies which?
Bisimulation

• M and M’ are bisimulation equivalent (bisimilar) if
  – M simulates M’ and vice-versa
  – Note: atomic proposition sets must be identical

• Are bisimulation and trace equivalence the same thing?

(Bi)Simulation and (A)CTL*

• If M’ simulates M, then any ACTL* property satisfied by M’ is satisfied by M

• If M’ and M are bisimilar, any CTL* property satisfied by one is also satisfied by the other
Verification

• How do we check for:
  – Trace containment?
  – Simulation?
  – Bisimulation?

• Assume that your machines are given as Kripke structures/Buchi automata
  – For the latter, all accepting paths correspond to runs

Verification

• How do we check for:
  – Trace containment?
    • Can be done using LTL model checking (see MC Sec. 9.6)
  – Simulation?
    • Iterative computation → next slide
  – Bisimulation?
    • Effectively same as simulation check (just done in two directions) [see Ch. 11 of MC]
Simulation Checking

- We attempt to compute the largest relation $H$ such that
  
  For all $(s, s')$, if $H(s, s')$ then:
  - $L'(s') = L(s) \cap AP'$
  - For every state $t$ s.t. $R(s, t)$ there is a state $t'$ such that $R'(s', t')$ and $H(t, t')$

- Then, check whether every initial state of $M$ is related by $H$ to an initial state of $M'$

Simulation Checking

- We attempt to compute the largest relation $H$ such that
  
  For all $(s, s')$, if $H(s, s')$ then:
  - $L'(s') = L(s) \cap AP'$
  - For every state $t$ s.t. $R(s, t)$ there is a state $t'$ such that $R'(s', t')$ and $H(t, t')$

- Compute sequence $H_0, H_1, \ldots, H_k$ where:
  - $H_0(s, s')$ iff $L'(s') = L(s) \cap AP$
  - $H_{n+1}(s, s')$ iff
    - $H_n(s, s')$, and
    - $\forall t \{ R(s, t) \Rightarrow \exists t' \ ( R(s', t') \wedge H_n(t, t') ) \}$
  (How to implement this? Why will it terminate?)
Simulation vs. Trace Containment

• Why would we want to use one over the other?

Next class

• Other optimizations in model checking:
  – Compositional reasoning
  – Symmetry reduction
• Mu-calculus