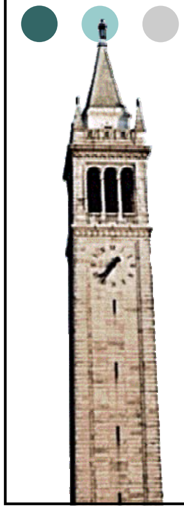
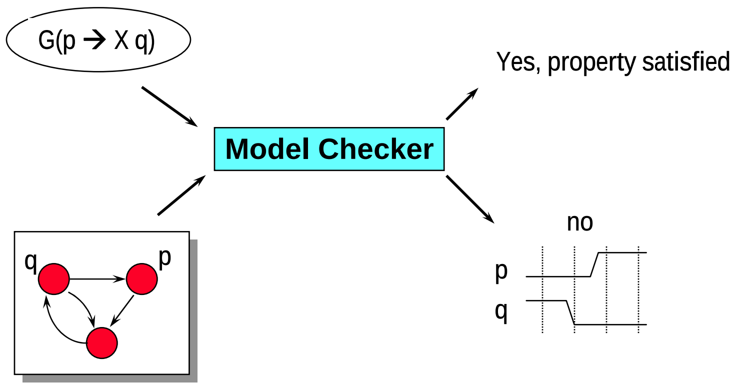


Fundamental Algorithms for System Modeling, Analysis, and Optimization

Edward A. Lee, Jaijeet Roychowdhury, Sanjit A. Seshia
UC Berkeley
EECS 144/244
Fall 2011

Model Checking

Model Checking

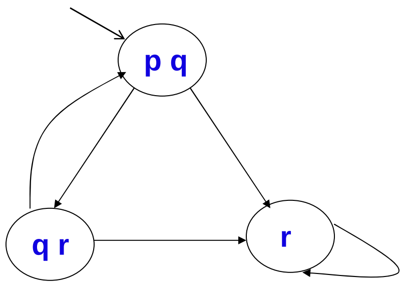


Outline

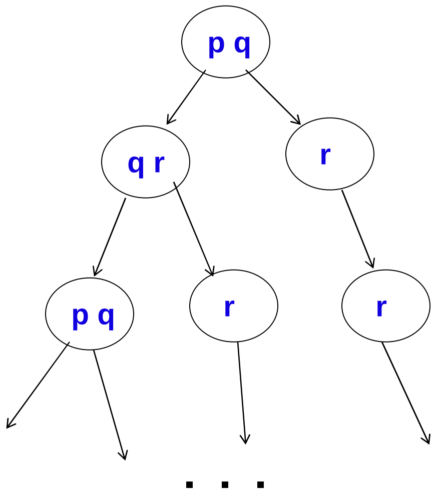
- Computation Tree Logic and why it is useful for model checking
- Model Checking with BDDs
- Bounded Model Checking with SAT

3

Labelled State Transition Graph



"Kripke structure"



Infinite Computation Tree

4

Temporal Logic

Linear Temporal Logic (LTL)

Properties expressed over a single time-line

Computation Tree Logic (CTL, CTL*)

Properties expressed over a tree of all possible executions

CTL* gives more expressiveness than LTL

CTL is a subset of CTL* that is easier to verify than arbitrary CTL*

Computation Tree Logic (CTL*)

Introduce two new operators called "Path quantifiers"

A p : Property p holds along all computation paths

E p : Property p holds along at least one path

Example:

"From any state, it is possible to get to the reset state "

A G (E F reset)

CTL: Every F, G, X, U must be preceded by either an A or a E

- E.g., Can't write A (FG p)

LTL is just like having an "A" on the outside

Why CTL?

Verifying LTL properties turns out to be computationally harder than CTL

Exponential in the size of the LTL expression

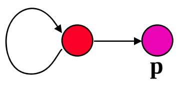
– linear for CTL

For both, verification is linear in the size of the state graph

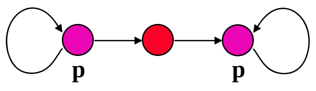
CTL as a way to approximate LTL

– AG EF p is <u>weaker</u> than G F p

Good for finding bugs...



– AF AG p is <u>stronger</u> than F G p



Good for verifying correctness...

CTL Model Checking

So, we've decided to do CTL model checking.

What are the algorithms?

Recap: Reachability Analysis

Given:

1. A Boolean formula corresponding to initial states R_0
2. δ

To find: All states reachable from R_0 in 1, 2, 3, … transitions (clock ticks)

Strategy: Denote set of states reachable from R_0 in k (or less) clock ticks as R_k

$$R_{k+1}(s^+) = R_k(s^+) + \exists s \{ R_k(s) . \delta(s, s^+) \}$$

Backwards Reachability Analysis

Given:

1. A Boolean formula corresponding to error states E_0
2. δ

To find: All states that can reach E_0 in 1, 2, 3, … transitions (clock ticks)

Strategy: Denote set of states reachable from E_0 in k (or less) clock ticks as E_k

$$E_{k+1}(s) = \quad E_k(s) \quad + \ \exists\, s^+ \ \{ \ E_k(s^+) \ . \ \delta(s, s^+) \ \}$$

Verification of G p

Corresponding CTL formula is AGp

– Remember that p is a function of s

Forward Reachability Analysis:

– Check if any $R_k(s)$. p'(s) is true for any s

Backward Reachability Analysis:

– Set E_0 = p'
– Check if $E_k(s)$. $R_0(s)$ is true for any s

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)) = ?

Model Checking CTL Properties

We define a general recursive procedure called "Check" to do this

– Performs fixpoint computation

Definition of Check:

– Input: A CTL property Π (and implicitly, δ)

– Output: A Boolean formula B representing the set of states satisfying Π

If B(s) . R_0(s) != 0, then Π is true (in the initial state)

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?

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

Seems like circular reasoning!

But it works out: using an recursive computation like in reachability analysis

– We compute a series of approximations leading to the right answer

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)

$Z_k(s) = \{ q(s) + [p(s) . \exists x,s^+ \{ \delta(s, x, s^+) . Z_{k-1}(s^+) \}] \}$

What’s a good choice for Z_0? Why will this terminate?

Summary

EGp computed similarly

Definition of Check:

- Input: A CTL property Π (and implicitly, δ)
- Output: A Boolean formula B representing the set of states satisfying Π

All Boolean formulas represented “symbolically” as BDDs

- “Symbolic Model Checking”

Bounded Model Checking [Biere, Clarke, Cimatti, Zhu99]

Given

- A finite state machine M (“transition system”)
- A property p

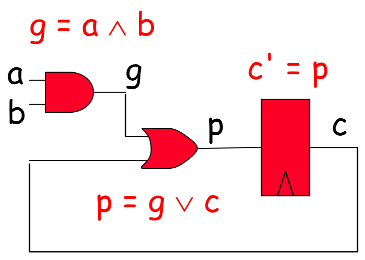
Determine

- Does M allow a counterexample to p of *k transitions or fewer*?

This problem can be translated to a SAT problem

Models

Transition system described by a set of constraints



Model:

$C = \{$
$g = a \wedge b,$
$p = g \vee c,$
$c' = p$
$\}$

Each circuit element is a constraint
note: $a = a_t$ and $a' = a_{t+1}$

Properties

We restrict our attention to safety properties.

Characterized by:

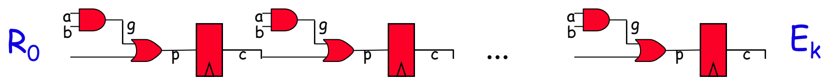
- Initial condition R_0
- Final condition E (representing “error" states)

A counterexample is a path from a state satisfying R_0 to state satisfying E, where every transition satisfies C.

Unfolding

Unfold the model k times:

$$U_k = C_0 \wedge C_1 \wedge \ldots \wedge C_{k-1}$$



- Use SAT solver to check satisfiability of

$$R_0 \wedge U_k \wedge E_k$$

- A satisfying assignment is a counterexample of k steps

BMC applications

Debugging:

- Can find counterexamples using a SAT solver

Proving properties:

- Only possible if a bound on the length of the shortest counterexample is known.
  - I.e., we need a *diameter* bound. The diameter is the maximum length of the shortest path between any two states.
- Worst case is exponential. Obtaining better bounds is sometimes possible, but generally intractable.

New Developments in SAT-based MC

- SAT-based bounded model checking has scaled to thousands of state bits and is very useful for debugging
  - Can verify LTL properties too
- Unbounded model checking is now also possible with SAT
  - interpolation-based model checking
- But on some problems, BDD-based model checking is still better