Model Checking

Sanjit Seshia
EECS
UC Berkeley

(with thanks to Kenneth McMillan)

Formal Verification as practiced today

Verification Engineer

RTL →

Properties →

(Spec)

← Correct / Error trace

MODEL GENERATER
MODEL CHECKER

Today's Lecture

What you know: How to formally specify properties using temporal logic

Today:

- Given a FSM description and a temporal logic property, how do we automatically check if that property holds?
  - Model checking
- Survey of some other formal verification topics
- What's next in verification?

S. Seshia

3

Recap: Terminology and Temporal Logic

Behavior / Execution / Trace / Run / Path

A property corresponds to a "set of behaviors"

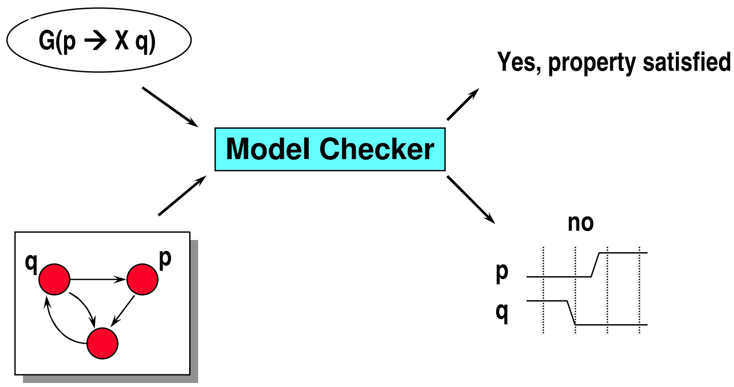
Operators to express properties over time:

G	"globally"
F	"eventually"/ "in the future"
X	"in the next state"
U	"until"

S. Seshia

4

Model Checking



S. Seshia

5

Brief History of Finite-State Model Checking

1977: Pnueli introduces use of (linear) temporal logic for program verification [1996 Turing Award]

1981: Model checking introduced by Clarke & Emerson and Quielle & Sifakis

– But capacity limited by "state explosion"

1986: Bryant publishes paper on BDDs

1987: McMillan comes up with idea for "Symbolic Model Checking" (using BDDs)

– First step towards tackling state explosion

1987-1999: Flurry of activity on model checking with BDDs, lots of progress using: abstraction, compositional reasoning, …

– More techniques to tackle state explosion

1999: Clarke et al. introduce "Bounded Model Checking" using SAT

– Exploits advantages of SAT over BDDs

1999-date: More advances based on both BDDs and SAT, industrial use increases especially for corner-case and control logic debugging

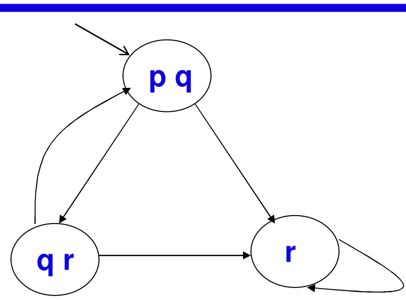
S. Seshia

6

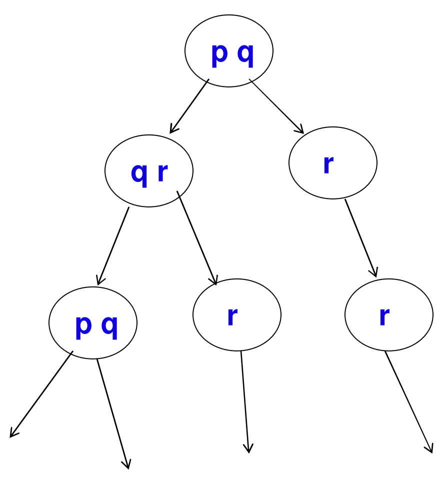
Outline

- Recap of Computation Tree Logic and why it is useful for designing verification algorithms
- Model Checking with BDDs
- Bounded Model Checking with SAT

Labelled State Transition Graph



"Kripke structure"



Infinite Computation Tree

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*

S. Seshia

9

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

S. Seshia

10

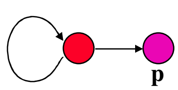
Why CTL?

- 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

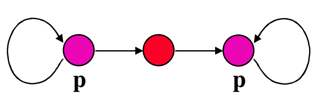
CTL as a way to approximate LTL

- 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?

S. Seshia

13

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

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

S. Seshia

14

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)) = ?

S. Seshia

17

Model Checking CTL Properties

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

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)

S. Seshia

18

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?

S. Seshia

19

CheckEX

CheckEX(p) returns a set of states such that p is true in their next states

How to write this?

S. Seshia

20

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)

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

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

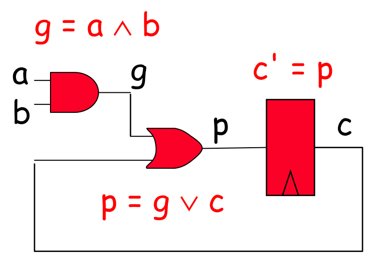
Determine

- 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:

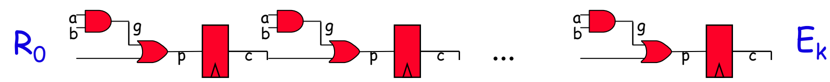
- **Initial condition R_0**
- **Final condition E (representing "error" states)**

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

But on some problems, BDD-based model checking is still better

Some Other Formal Verification Topics

Scaling up Model Checking

- Abstraction: Keep only the relevant state variables
- Compositional Reasoning: Break a system up into modules, prove the property for the modules, combine the proofs
- …

Model Generation

- Counterexample-guided Abstraction-Refinement
- Machine learning (especially for Environment model)

Theorem proving is also used, sometimes combined with Model Checking

Some References for Further Study

- Model Checking, E. Clarke, O. Grumberg, D. Peled, MIT Press, 2000.
- Verification Tools for Finite-State Concurrent Systems, Clarke, Grumberg, Long (in prelim reading list)
- A. Biere, A. Cimatti, E. Clarke, O. Strichman, Y. Zhu. Bounded Model Checking. In *Advances in Computers*, vol. 58, Academic Press 2003.

Formal Verification in Industry

Some commercial tools in EDA: Synopsys Magellan, 0-In FV, Jasper JasperGold, Real Intent Verix, IBM RuleBase, …

Theorem proving also used: e.g., Intel's Forte system, ACL2 prover at AMD

Software: Microsoft Static Driver Verifier (SDV), VeriSoft (Bell Labs), SPIN (Bell Labs, now NASA/JPL), …

Industry view: Useful, but not the only tool

What's next in Verification?

- Non-Boolean (infinite-state) Model Checking
  - Software (why aren't FSMs enough to express these?)
  - Real-time systems
  - Hybrid systems
  - Verifying data-dependent properties
- Computer Security
- Run-time Verification & Robustness

Computer Security

How is verifying security different from other forms of verification?

- What's different about the properties?
- What's different about the system model?

An Example of a Security Problem

Assume cryptography works perfectly, can't be broken

d → Encrypt, Key K → $E_K(d)$

It can still be possible to get unauthorized access to information!

- Encryption must be used carefully!

S. Seshia

35

Example: IBM 4758 Secure Co-processor

Used widely in the banking industry

Software Interaction

Common Cryptographic Architecture (CCA) Interface

Key	Control Vector
K	CV_K = Read, Write
…	

MK is in-built, secret, unique to each chip

Picture courtesy IBM

S. Seshia

36

The Problem [discovered by M. Bond, et al. at Cambridge,UK]

Using perfectly legal CCA commands, it is possible to generate a control vector to do operations one is not allowed to do

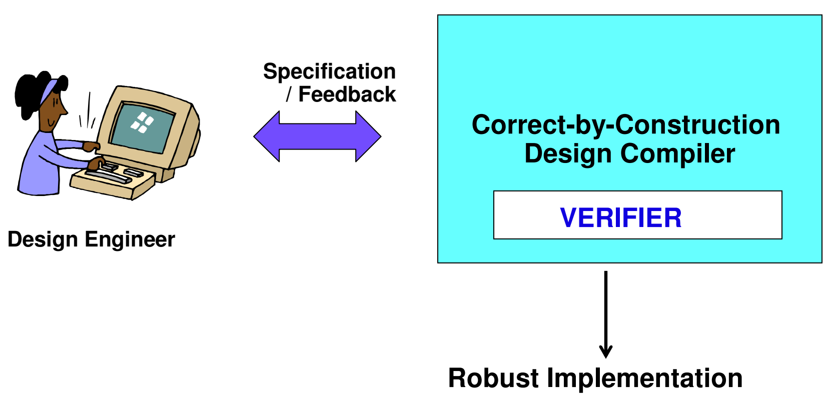
– E.g., read and write account information

Has to be an "inside job" at one of the bank branches

Can be discovered by a form of Bounded Model Checking [Ganapathy et al., ICSE'05]

S. Seshia 37

The Vision



S. Seshia 38