Software Model Checking

Scirbe

Xin. Qin[1] Mingyang. Zhang[1]

[1]Department of Computer Science
University of Southern California

CSCI 699, 2018

Instructor: Jyotirmoy Deshmukh

Outline I

1. Section 1
   - Precondition and Postcondition
   - Hoare Logic
   - Sound and Complete
   - Undecidability
2. Section 2
   - Quotient Graph
3. Section 3
   - Loop and Symbolic
   - Lattice
   - Knaster-Tarski Theorem
   - Simulation Relations/ Bisimulation Relations
   - Predicate Abstraction

Outline II

4. Section 6
   - Procedural Abstraction
   - Push Down Automata
5. Section 7
   - Heap Data Structures
6. Section 8
   - Ranking Function
7. Section 10
   - Discussion

Outline

1 Section 1

- Precondition and Postcondition
- Hoare Logic
- Sound and Complete
- Undecidability

Precondition and Postcondition

$$
\begin{aligned}
x := x + 5; \\
x > 10
\end{aligned} \tag{1}
$$

- **Weakest Precondition**:$wp(x := E, R) = R[x \leftarrow E]$.
  largest set that will hold: $\{x > 5\}$, requires loop to terminate.
- **Strongest Postcondition**: smallest set that will hold: $\{x > 15\}$
- **Weakest Liberal Precondition**: No loop terminate requirement.

Outline

1. Section 1
   - Precondition and Postcondition
   - Hoare Logic
   - Sound and Complete
   - Undecidability

Hoare logic: Program is nothing but a prove

$$\{A\}P\{B\} \tag{2}$$

- A,B is assertions; A precondition, B postcondition.
- Partial correctness: a pass reach B.
- Termination needs to prove separately.

$$\langle A\rangle P\langle B\rangle \tag{3}$$

Use $\langle\rangle$: indicates program terminates.

Outline

1. Section 1
   - Precondition and Postcondition
   - Hoare Logic
   - Sound and Complete
   - Undecidability

Sound and Complete

Sound

- Verification
  If the tool said the program is safe, then the program is safe.
- Bug finding
  If the tool said there is a bug, then the program really has a bug.
  Testing tool: Sound with respect to bug finding.

Complete

- If program safe, the tool also say safe
- If program has bug, you will find it (no testing tool is complete; function calls screw things up)

Trade off between soundness and completeness

Example

- Binary variable language: easy to achieve soundness but not expressive enough
- Complex language: hard to achieve soundness

Abstraction related

- Abstract too much: always unsafe (throw all)
- Throw away some of the program to see the answer: safe/unsafe;
- Abstract too small: too many detail.

Outline

1. Section 1
   - Precondition and Postcondition
   - Hoare Logic
   - Sound and Complete
   - Undecidability

Undecidable problem

Example

"Checking program termination is undecidable".

To verify if problem halt, should let program halt.
Reduce halting problem of Turing machine(NP-hard) to this problem.

Outline

2 Section 2
- Quotient Graph

Quotient graph

For reduction-based method mentioned the term: **quotient graph**.

- Related content in paper:
  "Behavioral equivalences such as similarity and bisimilarity construct a quotient graph that preserves reachability (i.e., there is a path from an initial state to ε in the original graph iff there is a path to ε in the quotient), and then performs reachability analysis on the quotient."

Outline

3 Section 3

- Loop and Symbolic
- Lattice
- Knaster-Tarski Theorem
- Simulation Relations/ Bisimulation Relations
- Predicate Abstraction

Discussed based on section 3.3

Loop can not be reasoned/verified symbolically

- Unrolling forever, coming closer and closer to fix point, but never able to find it.

Outline

3 Section 3
- Loop and Symbolic
- Lattice
- Knaster-Tarski Theorem
- Simulation Relations/ Bisimulation Relations
- Predicate Abstraction

Lattice

Introduction: Lattice theory is the study of sets of objects known as lattices. It is an outgrowth of the study of Boolean algebras, and provides a framework for unifying the study of classes or ordered sets in mathematics.
Partially ordered set (poset): A partially ordered set (poset) is a reflexive, antisymmetric and transitive binary relation.

- antisymmetric: $a \neq b : a \leq b \Longrightarrow not(b \leq a)$
- reflexive: for all elements: $\leq a$
- transitive: $a \leq b, b \leq c \Longrightarrow a \leq c$

Note: The binary relation of a partially ordered set is written as $\leq$.

Examples of Posets

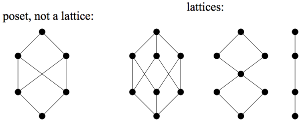
- Scheduling problems: PERT charts, flow charts
- Dependency graphs: software installers, compilers, variable dependencies
- C++ class hierarchy (a hierarchy where every node can have multiple parents)
- Part-whole relationships (e.g. food and ingredients)

Reference: http://www.upriss.org.uk/maths/mlec7.pdf

Lattice

Lattice A lattice is a poset where each two nodes have a greatest common child node and a least common parent node.

- lattice is meet, join closed
- **lub**: least upper bound, glb: greatest lower bound
- complete lattice: unique top, bottom element; (finite lattice mostly complete)



Reference: http://www.upriss.org.uk/maths/mlec7.pdf

Chain & Antichain

- chain: A sequence of ordered element.
- Antichain: non of them are ordered — $\{a, b\}\{b, c\}$ truly concurrent.

Example

Event in a distributed program: dimension, decomposition of antichain

Outline

3. Section 3
   - Loop and Symbolic
   - Lattice
   - Knaster-Tarski Theorem
   - Simulation Relations/ Bisimulation Relations
   - Predicate Abstraction

Fixed point on lattice

The theorem professor mentioned worth look at:

Theorem (Knaster-Tarski theorem)

Let L be a complete lattice and let $f : L \to L$ be an order-preserving function. Then the set of fixed points of f in L is also a complete lattice.

Reference: https://en.wikipedia.org/wiki/KnasterTarski_theorem

Usage

Example (1)

show program has a lattice structure,
states are lattice,
then can find fixed point.

Example (2)

Determine the bad state,
Then get all state that can reach the bad state,
If initial state in it, in trouble.

Outline

3 Section 3

- Loop and Symbolic
- Lattice
- Knaster-Tarski Theorem
- Simulation Relations/ Bisimulation Relations
- Predicate Abstraction

Simulation Relations/ Bisimulation Relations

- Labelled transition system $(S, Act, \rightarrow)$, where $\rightarrow \subseteq S \times Act \times S$ (write $s \xrightarrow{a} s'$ to denote $(s, a, s') \in \rightarrow$).
- Relation $R \subseteq S \times S$ is a simulation relation if R satisfies the following condition:
  $(s_1, s_2) \in R$ implies that, for each $s_1 \xrightarrow{a} s_1'$, there exists $s_2 \xrightarrow{a} s_2'$ such that $(s_1', s_2') \in R$.
- s_2 simulates s_1 if there exists a simulation relation R such that $(s_1, s_2) \in R$.
- Relation $R \subseteq S \times S$ is a bisimulation relation if both R and R^{-1} are simulation relations.
- s_1 and s_1 are bisimilar if there exists a bisimulation relation R such that $(s_1, s_2) \in R$.

Reference: https://pdfs.semanticscholar.org/presentation/c718/62eaaa72ee88baf35a66f39bf6375c47b29b.pdf

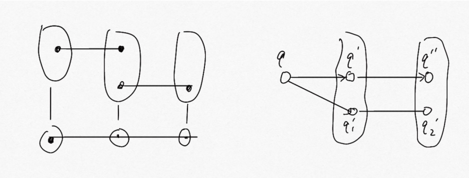
Simulation Relations/ Bisimulation Relations

OR:

$$
\begin{aligned}
& M_1 \prec M2 \\
& M_1 = (S_1, R_1, L) \\
& M_2 = (S_2, R_2, L) \\
& (q, r) \subseteq H, q \in S_1, r \in S_2 \\
\text{if (1) } & L(q) = L(r) \\
(2) & \forall q' \ (q, q') \subseteq R_1 \\
& \exists r' s.t. \\
& (1) \ (q', r') \subseteq R_2 \\
& (2) \ (r, r') \subseteq H
\end{aligned} \tag{4}
$$

Discussion

- Bisimulation can prove more
- Quationt: compute equivalence class
- Simulation relations example:



Outline

3 Section 3

- Loop and Symbolic
- Lattice
- Knaster-Tarski Theorem
- Simulation Relations/ Bisimulation Relations
- Predicate Abstraction

Predicate Abstraction

Transform program to operation on predicates

Rule

x := choose(m,n)
if (m is true) then x is true
if (n is true) then x is false
else *

Predicate Abstraction Example 1

```
#b1: {x > 0} b2: {x > 10}
x = x + 5  #b1 = choose(b1, false) b2 = choose(b2, false)
if (x > 0):
    x = x - 1 #assume (b1): x = x-1
if (x > 10):
    error  #assume(b2): error
```

Predicate Abstraction Example 2

```
# P = {x > 5, x < 5, y = 5}
#       b1      b2     b3
x = y
if (x == 5):
    error
```

- Assume $(\neg b_1 \wedge \neg b_2 \wedge \neg b_3)$

$$\langle b_1, b_2, b3\rangle := \langle \text{choose}(\mathit{false}, b_3), \text{choose}(\mathit{false}, b_3), \text{choose}(b_3, \neg b_3)\rangle \tag{5}$$

Thus from rule above, b_1, b_2 is undetermined.

Predicate Abstraction Example 3

```
# P = {x = 5, y > x, y = 5}
#       b1      b2     b3
x = y
if (x == 5):
    error
```

- Assume $(b_1 \wedge b_2 \wedge \neg b_3)$

$$\langle b_1, b_2, b_3 \rangle := \langle \text{choose}(b_3, \neg b_3), \text{choose}(\mathit{false}, \mathit{true}), \text{choose}(b_3, \neg b_3) \rangle \tag{6}$$

  Note that here implies b_2 statement cannot return true.

- Assume(b_1), ERROR is not reachable. As assume(b_1) means the situation when b_1 is true.

Outline

4 Section 6
- Procedural Abstraction
- Push Down Automata

procedural abstraction

- Behaviors reconstituted from the input-output behaviors common in static analysis.
  1. take all function call
  2. replace function call with doing an assignment
- Call graph
  - summary for leaf node can used for higher
- Recursive call graph has a loop
- Store VS Compute only when been called (lazy way)

Summary

Replace function call with effect of function.

Outline

4. Section 6
   - Procedural Abstraction
   - Push Down Automata

Push Down Automata(PDA)

Two push down machine can simulate a truing machine. Therefore checking concurrent program is like checking turing machine

Reachability very difficult to check in concurrent program.

Outline

5. Section 7
   - Heap Data Structures

Predicate abstraction

- **Points**:
  – Alias Analysis + data structure of it
  – Alias Analysis: care if two pointer can point to same location
  – Linked list
  – Shape invariant: care not number of elements, but for example, list is acyclic.

Outline

6 Section 8
- Ranking Function

Ranking function

Example

$$\begin{aligned} &\text{while}(x > 0)\{ \\ &x := x - 1; \} \end{aligned} \qquad (7)$$

1. Decrease in every program loop
2. In the end the statement $x > 0$ is false

No general determine if there exist a ranking function.

Outline

7. Section 10
   - Discussion

Discussion

- SMT solvers evolved a lot.
- Type system becomes super popular. Type based programming. Type checking instead of verification.
- State of art has evolved, but many of the question are still open.
- Black box verification is open.
- Question of environment: e.g. on the phone.

Other discussion

- Well typed program
- Recurrent set: $\forall s \in R, (s, s') \in T, s' \in R$
- Why use math to describe: for corner case

References

Ranjit Jhala and Rupak Majumdar. 2009.
Software model checking
ACM Comput. Surv. Surv. 41, 4, Article 21 (October 2009), 54 pages. DOI=http://dx.doi.org/10.1145/1592434.1592438.