Symbolic Execution

CS252r Spring 2011
Contains content from slides by Jeff Foster
Static analysis

- Static analysis allows us to reason about all possible executions of a program
  - Gives assurance about any execution, prior to deployment
  - Lots of interesting static analysis ideas and tools
- But difficult for developers to use
  - Commercial tools spend a lot of effort dealing with developer confusion, false positives, etc.

See A Few Billion Lines of Code Later: Using Static Analysis to Find Bugs in the Real World in CACM 53(2), 2010
One issue is abstraction

- Abstraction lets us scale and model all possible runs
  - But must be conservative
  - Try to balance precision and scalability
    - Flow-sensitive, context-sensitive, path-sensitivity, ...

- And static analysis abstractions do not cleanly match developer abstractions
Testing

• Fits well with developer intuitions
• In practice, most common form of bug-detection
• But each test explores only one possible execution of the system
  • Hopefully, test cases generalize
Symbolic execution

• King, CACM 1976.

• Key idea: generalize testing by using unknown symbolic variables in evaluation

• Symbolic executor executes program, tracking *symbolic state*.

• If execution path depends on unknown, we fork symbolic executor
  • at least, conceptually
Symbolic execution example

1. int a = α, b = β, c = γ;
2. // symbolic
3. int x = 0, y = 0, z = 0;
4. if (a) {
5.    x = -2;
6. }
7. if (b < 5) {
8.    if (!a && c) { y = 1; }
9.    z = 2;
10.}
11. assert(x+y+z!=3)
1. int a = α, b = β, c = γ;
2. // symbolic
3. int x = 0, y = 0, z = 0;
4. if (a) {
5.   x = -2;
6. }
7. if (b < 5) {
8.   if (!a && c) { y = 1; }
9.   z = 2;
10. }
11. assert(x+y+z!=3)
1. int a = α, b = β, c = γ;
2. // symbolic
3. int x = 0, y = 0, z = 0;
4. if (a) {
5.   x = -2;
6. }
7. if (b < 5) {
8.   if (!a && c) { y = 1; }
9.   z = 2;
10. }
11. assert(x+y+z!=3)
What’s going on here?

• During symbolic execution, we are trying to determine if certain formulas are satisfiable

  • E.g., is a particular program point reachable?
    • Figure out if the path condition is satisfiable
  
  • E.g., is array access $a[i]$ out of bounds?
    • Figure out if conjunction of path condition and $i<0 \lor i > a.length$ is satisfiable

• E.g., generate concrete inputs that execute the same paths

• This is enabled by powerful SMT/SAT solvers

  • SAT = Satisfiability
  
  • SMT = Satisfiability modulo theory = SAT++
    • E.g. Z3, Yices, STP
SMT

• Satisfiability Modulo Theory

• SMT instance is a formula in first-order logic, where some function and predicate symbols have additional meaning

• The “additional meaning” depends on the theory being used
  • E.g., Linear inequalities
    • Symbols with extra meaning include the integers, +, -, ×, ≤
  • A richer modeling language than just Boolean SAT
  • Some commonly supported theories: Uninterpreted functions; Linear real and integer arithmetic; Extensional arrays; Fixed-size bit-vectors; Quantifiers; Scalar types; Recursive datatypes, tuples, records; Lambda expressions; Dependent types

• A lot of recent success using SMT solvers
  • In symbolic execution and otherwise...
Predicate transformer semantics

- **Predicate transformer semantics** give semantics to programs as relations from logical formulas to logical formulas
  - Strongest post-condition semantics: if formula $\varphi$ is true before program $c$ executes, then formula $\psi$ is true after $c$ executes
    - Like forward symbolic execution of program
  - Weakest pre-condition semantics: if formula $\varphi$ is true after program $c$ executes, then formula $\psi$ must be true before $c$ executes
    - Like backward symbolic execution of program
Predicate transformer semantics

• Predicate transformers operationalize Hoare Logic
• Hoare Logic is a deductive system
  • Axioms and inference rules for deriving proofs of Hoare triples (aka partial correctness assertion)
  • \{ \varphi \} \ c \ { \psi \} says that if \varphi \ holds before execution of program c and c terminates, then \psi \ holds after c terminates
• Predicate transformers provide a way of producing valid Hoare triples
Hoare logic

• First we need a language for the assertions
  • E.g., first order logic

assertions

\[ P, Q \in \text{Assn} \]

\[ P ::= \text{true} \mid \text{false} \mid a_1 < a_2 \]
\[ \mid P_1 \land P_2 \mid P_1 \lor P_2 \mid P_1 \Rightarrow P_2 \mid \neg P \]
\[ \mid \forall i. P \mid \exists i. P \]

arithmetic expressions

\[ a \in \text{Aexp} \]

\[ a ::= \ldots \]

logical variables

\[ i, j \in \text{LVar} \]

• We also need a semantics for assertions
  • For state \( \sigma: \text{Var} \rightarrow \text{Int} \) and interpretation \( I: \text{LVar} \rightarrow \text{Int} \) we write \( \sigma, I \models P \) if \( P \) is true when interpreted under \( \sigma, I \)
**Rules of Hoare Logic**

**SKIP**

\[ \{P\} \text{skip} \{P\} \]

**SEQ**

\[ \{P\} c_1 \{R\} \quad \{R\} c_2 \{Q\} \]

\[ \{P\} c_1; c_2 \{Q\} \]

**ASSIGN**

\[ \{P[a/x]\} x := a \{P\} \]

**IF**

\[ \{P \land b\} c_1 \{Q\} \quad \{P \land \neg b\} c_2 \{Q\} \]

\[ \{P\} \text{if } b \text{ then } c_1 \text{ else } c_2 \{Q\} \]

**CONSEQUENCE**

\[ \vdash (P \Rightarrow P') \quad \{P'\} c \{Q'\} \quad \vdash (Q' \Rightarrow Q) \]

\[ \{P\} c \{Q\} \]

**WHILE**

\[ \{P \land b\} c \{P\} \]

\[ \{P\} \text{while } b \text{ do } c \{P \land \neg b\} \]
Soundness and completeness of Hoare Logic

• Semantics of Hoare Triples
  • $\sigma, I \models \{P\} c \{Q\} \triangleq \text{if } \sigma, I \models P \text{ and } \llbracket c \rrbracket_\sigma = \sigma', \text{ then } \sigma', I \models P$
  • $\models \{P\} c \{Q\} \triangleq \text{for all } \sigma, I \text{ we have } \sigma, I \models \{P\} c \{Q\}$

• Soundness: If there is a proof of $\{P\} c \{Q\}$, then $\models \{P\} c \{Q\}$

• Relative completeness: If $\models \{P\} c \{Q\}$ then there is a proof of $\{P\} c \{Q\}$
  • (assuming you can prove the implications in the rule of consequence).
Weakest pre-condition semantics

- Function \( wp \) takes command \( c \) and assertion \( Q \) and returns assertion \( P \) such that \( \models \{P\}c\{Q\} \)
- \( wp(c, Q) \) is the **weakest** such condition
  - \( \models \{P\}c\{Q\} \) if and only if \( P \Rightarrow wp(c, Q) \)
- \( wp(\text{skip}, Q) = Q \)
- \( wp(x:=a, Q) = Q[a/x] \)
- \( wp(c_1;c_2, Q) = wp(c_1, wp(c_2, Q)) \)
- \( wp(\text{if } b \text{ then } c_1 \text{ else } c_2, Q) = (b \Rightarrow wp(c_1, Q) \land (\neg b \Rightarrow wp(c_2, Q)) \)
What about loops?

- Two possibilities: do we want the weakest precondition to guarantee termination of the loop?
- **Weakest liberal precondition**: does not guarantee termination
  - Corresponds to partial correctness of Hoare triples
  - \( \text{wp(while } b \text{ do } c, Q) = \forall i \in \text{Nat. } L_i(Q) \)
  - where \( L_0(Q) = \text{true} \)
    
    \[
    L_{i+1}(Q) = (\neg b \Rightarrow Q) \land (b \Rightarrow \text{wp}(c, L_i(Q)))
    \]
  - Ensures loop terminates in a state that satisfies \( Q \) or runs forever
What about loops?

- **Weakest precondition**: guarantees termination
  - Corresponds to total correctness of Hoare triples
  - $\text{wp(while } b \text{ do } c, Q) = \exists i \in \text{Nat. } L_i(Q)$
    - $L_0(Q) = \text{false}$
    - $L_{i+1}(Q) = (\neg b \Rightarrow Q) \land (b \Rightarrow \text{wp}(c, L_i(Q)))$
  - Ensures loop terminates in a state that satisfies $Q$
Strongest post condition

- Function sp takes command c and assertion P and returns assertion Q such that $\models \{P\}c\{Q\}$
- $sp(c, P)$ is the strongest such condition
  - $\models \{P\}c\{Q\}$ if and only if $sp(c, P) \Rightarrow Q$
Strongest post condition

- \( \text{sp}(\text{skip}, P) = P \)
- \( \text{sp}(x:=a, P) = \exists n. x=a[n/x] \land P[n/x] \)
- \( \text{sp}(c_1; c_2, P) = \text{sp}(c_2, \text{sp}(c_1, P)) \)
- \( \text{sp}(\text{if } b \text{ then } c_1 \text{ else } c_2, P) = \text{sp}(c_1, b \land P) \lor \text{sp}(c_2, \neg b \land P)) \)
- \( \text{sp}(\text{while } b \text{ do } c, P) = \neg b \land \exists i. L_i(P) \)
  where
  \[ L_0(P) = P \]
  \[ L_{i+1}(P) = \text{sp}(c, b \land L_i(P)) \]
- Weakest preconditions are typically easier to use than strongest postconditions
Symbolic execution

• Symbolic execution can be viewed as a predicate transformation semantics

• Symbolic state and path condition correspond to a formula that is true at a program point
  • e.g., Symbolic state \([x\mapsto\alpha, y\mapsto\beta+7]\) and path condition \(\alpha>0\) may correspond to \(\alpha>0 \land x=\alpha \land y=\beta+7\)

• Strongest post condition transformations gives us a forward symbolic execution of a program

• Weakest pre condition transformations gives us a backward symbolic execution of a program
Symbolic execution

• Recall
  • \( \text{sp}(x:=e, P) = \exists n. x=e[n/x] \land P[y/x] \)
  • \( \text{sp}(c_1;c_2, P) = \text{sp}(c_2, \text{sp}(c_1, P)) \)
  • \( \text{sp}(\text{if } b \text{ then } c_1 \text{ else } c_2, P) = \text{sp}(c_1, b \land P) \lor \text{sp}(c_2, \neg b \land P) \)
  • \( \text{sp}(\text{while } b \text{ do } c, P) = \neg b \land \exists i. \text{L}_i(P) \)
    where \( \text{L}_0(P) = \text{true} \)
    \( \text{L}_{i+1}(P) = \text{sp}(c, b \land \text{L}_i(P)) \)

• Disjunction encoded by multiple states
  • \( \langle \text{if } b \text{ then } c_1 \text{ else } c_2, P \rangle \Downarrow \langle \text{skip}, \{b \land P, \neg b \land P\} \rangle \)
  • or equivalently with non-deterministic semantics?
    • \( \langle \text{if } b \text{ then } c_1 \text{ else } c_2, P \rangle \leftrightarrow \langle c_1, b \land P \rangle \) and
    \( \langle \text{if } b \text{ then } c_1 \text{ else } c_2, P \rangle \leftrightarrow \langle c_2, \neg b \land P \rangle \)

• While loops simply unrolled (may fail to terminate)
Symbolic execution and abstract interpretation

• Can we use logical formulas as an abstract domain?
  • Yes! See Sumit Gulwani’s paper next week, which uses logical abstract interpretation
  • Also makes use of SMT solvers

• Can perhaps be seen as an abstract semantics for a concrete predicate transformer semantics?
Summary

• Symbolic execution
  • Predicate transformation semantics
  • Allows us to reason about multiple concrete executions
    • But may not allow us to reason about all possible executions
  • Enabled by recent advances in SMT solvers
• Next class: two symbolic execution papers
• Next week: logical abstract interpretation