Announcements

- HW7 due in about ten days
- HW8 will be optional
- No CLASS or office hours on Tuesday
  - I will catch up on grading next week!
- Presentations
  - Some of you haven’t sent me their selections
  - Please do!
- Quiz 6

Outlines

- SMT-LIB (for HW7)
- Symbolic Execution
  - Overview and applications
  - Challenges
  - Tools and techniques

SMT-LIB

- SMT-LIB is a language for specifying input to SMT solvers (e.g., Z3)
  - (declare-const x Int) declare an integer constant x
  - (assert (> x 0)) add x>0 to known facts
  - (check-sat) checks if there exist an assignment that makes all known facts true; returns (sat) or (unsat)
  - (get-model) print this assignment
- https://rise4fun.com/z3/tutorial

SMT-LIB

- Your homework is to write a Tiny Dafny
  - Given an IMP program \{ P \} c \{ Q \} generate verification conditions in SMT-LIB
  - Verify conditions with Z3
- Yet another programming language, OCaml!
- Some pitfalls
  - Function calls: (f arg1 arg2) NOT f(arg1, arg2)!
  - == is reference equality. Use (String.equal s1 s2)
Suppose we need to verify \( \{ P \} c \{ Q \} \).

Generate \( wp(c,Q) \).

Program verifies when \( P \Rightarrow wp(c,Q) \) is valid.

A logical formula is valid when true for all inputs.

Encoding

Duality of satisfiability and validity:

\( F \) is valid iff \( \neg F \) is unsatisfiable

Ask: is \( \{ P \Rightarrow wp(c,Q) \} \) satisfiable

If (unsat) program is correct

If (sat) our program is incorrect, we'll get model

Example

\begin{verbatim}
requires: x == 1 || x == -2
ensures: y == 0
{ y = x + 4;
  if (x > 0) {
    y = x*x - 1;
  } else {
    y = y + x;
  }
}
\end{verbatim}

Another Example

Is this formula valid?

\( (x>0 \text{ and } x+5 > 5) \) or

\( (x<=0 \text{ and } (x=0 => x + x + 5 = 5)) \)

SMT Solvers

SAT Solvers are at the heard of SMT Solvers.

In practice, optimizations on SMT expressions is crucial.

Simple identities (\( x+0=x \), \( x*0=0 \))

E.g., \((\text{simplify } (\geq (x 5) 5)) \) yields \((\text{not } (\leq x 0)) \)

Theory of arrays:

E.g., \((\text{simplify } (\text{select } (\text{store } a 42 x) 42) \)

Cache solver queries

Remove useless variables

Outline

SMT-LIB (for HW7)

Symbolic Execution

Overview and applications

Challenges

Tools and techniques

Classical References


Resurgence and Applications

- More powerful computers lead to much more powerful reasoning tools (e.g., Z3)
- Systems that started a resurgence
  - DART by Godefroid and Sen, PLDI 2005
  - EXE by Cadar, Ganesh, Pawlowski, Dill and Engler, CCS 2006

Symbolic Execution Example 1

```c
void foobar (int a, int b) {
    int x = 1, y = 0;
    if (a != 0) {
        y = 3+x;
        if (b == 0)
            x = 2*(a+b);
    }
    { x-y != 0 }
}
```

Motivation for Symbolic Execution

- Why?
  - One symbolic execution path covers many actual inputs
    - Exactly the set of inputs that satisfy the path condition
    - Thus, we cover a lot more of the program input space than testing

VC Generation Works Too

```c
void foobar (int a, int b) {
    int x = 1, y = 0;
    if (a != 0) {
        y = 3+x;
        if (b == 0)
            x = 2*(a+b);
    }
    { x-y != 0 }
}
VC generation vs. Symbolic Execution?
- VC generation = Backward reasoning
  - HW7
- Symbolic execution = Forward reasoning
  - HW8 (one option): Add a symbolic execution engine as another “interpreter” of IMP programs

Challenges to Symbolic Execution?
- State space explosion (Path explosion)
  - \( n \) conditionals generate \( 2^n \) paths
- Memory: how to handle pointers and arrays?
- Constraint solving: are SMT solvers good enough to solve complex constraints?
- “Edge” of program, i.e., libraries and binary code: how do we handle them, with no benefit of high-level static analysis?

Search Strategies
- We can think of program execution as a DAG
  - Nodes \( n_i \) represent states
  - Edges \((n_i, n_j)\) represent state transitions
  - We need strategies/heuristics for graph exploration
    - At each step, how do we chose which paths to explore and which paths to drop
- There are many strategies and heuristics!

Search Strategies
- DFS
- BFS
- Advantages
  - Simplicity
- Drawbacks
  - Generally, unguided by other “knowledge”
  - DFS can get stuck in one part of program
  - BFS considered the better one

Run Different Searches at the Same Time
- Advantages?
  - May achieves better coverage as it explores different strategies
  - Strategies target certain kinds of bugs better than others
- Drawbacks?
  - As good as “best” search strategy but wastes time running other search strategies too
Libraries and Binary Code

- "Edges" of the program
- Libraries, binary code

One way
- Pull in library code (libc, glibc)
  - Hard. Symbolic execution easily gets stuck

Another way
- Summaries (stubs) for library code
  - Also hard. A lot of work and often unsound

Concolic execution gets around these...

Concolic Execution

- Another big idea, due to Sen et al., FSE 2005
- Mixes concrete and symbolic execution

One variation: dynamic symbolic execution
- Instrument program to do symbolic execution
- Select some inputs
- Run path from start to finish, maintaining concrete state and symbolic state
- When finished, generate a new path condition by negating last path condition
- Solve path condition and if satisfiable, generate input and run

Concolic Execution, Example

Suppose we chose inputs \( a=1, b=1 \).

Recent Success

- SAGE
  - Microsoft, concolic execution
  - Finds bugs in file parsers
  - Microsoft continuously runs SAGE!

- Mayhem
  - Combines BFS and advanced search techniques
  - Runs on binary code
  - Automatically generates exploits when bug found

- KLEE
  - Symbolically executes LLVM bitcode