Debugging a Program using Symbolic Execution

Amanda Crowell

CMSC631 Fall 2010

Background

- Symbolic Execution
  - Supply symbolic inputs to a program
  - Can determine valid paths through program & values for program variables

Background

- Symbolic Executors (SEs)
  - Powerful tools
  - Many applications
    - Bug finding
    - Program testing
    - Understanding configurable software systems (Otter)
    - Debugging

Motivation

- Issues with SEs
  - Fully automated black boxes
  - Only as good as what you can **do with the output**
  - Output usefulness **decreases** as number of lines **increases**
- How can we
  - Bring the human into the loop
  - Allow more control over the execution

Idea

- Write a debugging interface for a SE that
  - Allows for normal debugger functions
    - Stepping
    - Breakpoints
    - Value inspection
  - But also incorporates SE functions
    - Inspect symbolic values & constraints
    - Inspect path condition
    - Pick paths to follow at branches

Otter – a SE for C

- Currently in development at UMD
- Written in OCAML
- CIL– intermediate representation
- STP – evaluate symbolic expressions
- Implements a lot of C functionality

THE DEBUGGER

Stepping

- Basic debugging idea
  - Allow user to pause execution after each individual step
  - Conceptually operates at high level, the line of source code

```
0    int x, y, z;
1    x = 2;
2    y = 3;
3    z = x * y
```

Stepping + Next

- Otter debugging idea
  - Conceptually, the same
  - Otter adds incremental steps (CIL)
  - Use “step”, add “next”

```
6    __SYMBOLIC(&n);
```

```
6    (BLOCK)
6    (INSTRS)
6    n
6    = Bytearray(\<4>\<3>\<2>\<1>)
```

Other Normal Functionality

- Run
  - Run until end of program
- Breakpoints
  - Run until specified line
- Print file contents
- Print current instruction

Control Flow

- Idea
  - Choice of path at branches
  - Possible due to nature of SE
- Details
  - Otter normally forks into **true** and **false** job at a branch (Ex. “if”)
  - Display output of both executions
  - Instead, ask user which to execute

Value Modification

- Allow user to **view** and **edit** a value
  - Concrete
  - Stored in state, must be updated
  - Account for local, global variables
- Create new symbolic values
  - For assigning on the fly

Path Condition Operations

- Print path condition
  - At user request
  - At end of execution
- Ask for concrete solution to
  - Specific symbolic values
  - Entire path condition
- Ask for constraints for a symbolic value
  - Determined by path condition
- Edit path condition

DEMO

C File

```
0: void main(){
1:   int n;
2:   __SYMBOLIC(&n);
3:
4:   if(n>6)
5:        n += 1;
6:   if(n<4)
7:        n -= 1;
8:
9:   __ASSERT(n!=6);
10: }
11:
```

Output

- Normal Otter Runs

invariant1.c.normal.out.txt

test-prop7.c.normal.out.txt

- DebugOtter Run

test-prop7.c.debug.out.txt