Symbolic Execution

Programs section, Lecture 12

Pavel Panchekha
CS 6110, U of Utah
13 February 2020
Syntax

\[ e, e_1, e_2 := e_1 + e_2 \mid e_1[e_2] \mid -e_1 \mid \ldots \]

\[ a[i+1] \]
def `eval`(env, expr):
    match expr:
        Add(expr1, expr2):
            `eval`(env, expr1) + `eval`(env, expr2)
...
def run(stmt):
    env = initial_env()
    while stmt != HALT:
        env, stmt = step(env, stmt)
Class Progress

Logical reasoning
Program logics
Static analysis

Expressions
Statements
Procedures
Loops
Expressions

What does an expression **compute**?

Symbolic values and environments

**Executing statements** without running expressions

Evaluating statements with path conditions

**Feasibility** of program executions

Test generation and symbolic/concrete execution
Symbolic Expressions

Running expressions without running them
Evaluating Expressions

```plaintext
x = (2 << (2 << 8))
return x - x
```

```plaintext
x = (2 << (2 << n))
return x - x
```

```plaintext
y = x
return x - y
```

0

0

0
Symbolic Evaluation

Convert program expressions to equivalent formulas
Program variables become logical variables

```python
def sym_eval(expr):  # Returns a z3 formula
    match expr:
        case Add(expr1, expr2):
            return z3.Add(sym_eval(expr1), ...)  
    ...
```
Environments

Run statements, convert expressions

\[ \text{env} : \text{Map<Variable, Formula>} \]

Step semantics don’t change

But expression evaluation now symbolic

\[
\{ x: 3n + 1, y: n - 1 \}, (x -= y; s) \downarrow
\]

\[
\{ x: 2n + 2, y: n - 1 \}, s
\]
Example

Execute; what is returned; what is the environment?

```python
def do_something(x: Int, y: Int):
    z = x + y
    w = z - x
    return [1, 4, 8][w % 3]
```

\[
[1,4,8][(x+y)-x] \mod 3
\]

\{
x: x, y: y, z: x+y, w: (x+y)-x
\}
Challenges

Logic **may not match** expressions in language

\[ s = \text{arr}[1:7] \quad \rightarrow \quad \text{slice}(a, 1, 7) \]

\[ \forall k, 0 \leq k < j - i \rightarrow \text{slice}(a, i, j)[k] = a[k + i] \]

Need **logical form** for each primitive operation

**Summaries** of library functions also helpful

\[ \forall a, \forall x, \text{indexOf}(a, x) \neq -1 \rightarrow a[\text{indexOf}(a, x)] = x \]
Path Conditions
Executing statements to track control flow
Conditionals

What does the following code return?

```python
def min(x: Int, y: Int):
    z = x
    if (x < y):
        z = x
    if (y < x):
        z = y
    return z
```

How do we handle conditionals, symbolically?
Path Conditions

Symbolic evaluation doesn’t produce true or false

Try both ways, record symbolic conditions used

\[
\begin{align*}
\Gamma \vdash e \Downarrow \top & \implies \Gamma, s_t \\
\Gamma \vdash e \Downarrow \bot & \implies \Gamma, s_f
\end{align*}
\]

\[
\begin{align*}
\Gamma, \text{if } (e) \{s_t\} \text{ else } \{s_f\} & \sim \Gamma, s_t \\
\Gamma, \text{if } (e) \{s_t\} \text{ else } \{s_f\} & \sim \Gamma, s_f
\end{align*}
\]

\[
\begin{align*}
\text{step}(\text{env}, \text{stmt}) & \rightarrow \text{List}<\text{(Env, Stmt, Formula)}>
\end{align*}
\]

\[
\begin{align*}
\Gamma, \text{if } (e) \{s_t\} \text{ else } \{s_f\}, p & \sim \left[ \Gamma, s_t, p \land e(\Gamma) \right. \\
& \left. \Gamma, s_f, p \land \neg e(\Gamma) \right]
\end{align*}
\]
Path Conditions

if (x < y):
    z = x

if (y < x):
    z = y

{x: x, y: y}, x < y → {x: x, y: y}, T

{x: x, y: y}, x < y → {x: x, y: y, z: x}, x < y

{x: x, y: y}, ¬(x < y)
Path Conditions

if \( x < y \):
  \( z = x \)

if \( y < x \):
  \( z = y \)

\( \neg(x < y) \land \neg(y < x) \)
Path Conditions

if (x < y):
    z = x

if (y < x):
    z = y

Impossible
Feasibility

Path condition may be **impossible** to meet

**No program execution** could take that program path

Important to **discard** those paths; avoid exponential growth

To symbolically execute “$\Gamma$, if $(e)$ $\{s_t\}$ else $\{s_f\}, p$”:

Check $p \land e(\Gamma)$; if so execute $\Gamma, s_t, p'$

Check $p \land \neg e(\Gamma)$; if so execute $\Gamma, s_f, p'$
Architecture

Program → Run Statements → Paths

Run Statements →
  - Check Feasibility
  - Convert to Formula
def which(x, y, z, target):
    i = 0
    a = [x, y, z]
    while i < 3:
        if a[i] == target:
            return i
        i = i + 1
    return -1

Draw the path tree. **How many paths** are there?

What is the **path condition** and **return value** for each
def has_x(s: String):
    i = 0
    out = False
    while i < len(s):
        if s[i] == "x":
            out = True
        i = i + 1
    return out
Merge vs Split

Length $k$ means $2^k$ paths

But how different are they?

$\text{out} = T \quad \text{out} = T \quad \text{out} = T$

Many paths, same environment

Merge paths; $k + 1$ left (here)

If $\forall s, \Gamma_1 = \Gamma_2$, keep one with path $p_1 \lor p_2$
Course Updates
Proposal Presentations
Presentations

Project proposal presentations on Monday

8 minutes for groups, 4 minutes for solo

Upload slides in PDF format for fast switching

For groups:
- Motivation
- Goal
- Approach

For solo:
- Syntax
- Design choices
- Example code
Presentation Tips

**Practice!** It takes 8/4 minutes. If stuck, change slides!

- Slides aren’t set in stone! **Practice**, then change slides.
- If presenting together, **practice together**

**Examples and demos > explanations or summaries**

Use Keynote, Powerpoint, or Google Docs

- Beamer, etc encourage bad presentation habits

Happy to give **personalized feedback** on request!
Assignment 3

Analyze GNU echo using symbolic execution tool KLEE

1. Find all valid flags
2. Find hidden bug
3. Convert path condition to regular expression

Recitation Friday 13:00 MEB 3485 (as usual)

Installation, how to use KLEE, example output
Concrete/Symbolic
Generating inputs that test program paths
∃x, p(x)

x is input that follows the path
Each path condition is **one input**

\[ \neg (0 < \text{len}(s)) \rightarrow "" \]

\[ 0 < \text{len}(s) \land s[0] = x \land \neg (1 < \text{len}(s)) \rightarrow "x" \]

\[ 0 < \text{len}(s) \land s[0] \neq x \land \neg (1 < \text{len}(s)) \rightarrow "y" \]

Each input is **distinct**

Path conditions **mutually exclusive**

Achieve **complete path coverage**
Concrete/symbolic

**Totally symbolic** inputs challenging in many cases

- Loops: *infinite path trees*, must be cut off
- Data structures: *infinite possibilities*, must be bounded
- System calls: *how do you print a symbolic string?*

Solution: *solve for inputs* while symbolically executing

Input is *representative* for current path condition
Generating Inputs

Query: $\exists x : \mathbb{Z}, \exists y : \mathbb{Z}, \top$

Solution: $x = 0, y = 0$

$0 < 0 \nRightarrow \bot$

$x < y$
Generating Inputs

Query: $\exists x : \mathbb{Z}, \exists y : \mathbb{Z}, T$

Solution: $x = 0, y = 0$

$0 < 0 \not\Rightarrow \bot$
Generating Inputs

Query: $\exists x : \mathbb{Z}, \exists y : \mathbb{Z}, T$

Solution: $x = 0, y = 0$

Path 1 done

Change path

$y < x$
Generating Inputs

Query: $\exists x \in \mathbb{Z}, \exists y \in \mathbb{Z}, \neg(x < y) \land y < x$

Solution: $x = 1, y = 0$
Generating Inputs

Query: $\exists x : \mathbb{Z}, \exists y : \mathbb{Z}, x < y$

Solution: $x = 0, y = 1$

Path 3 done

Change path
Generating Inputs

Query: $\exists x : \mathbb{Z}, \exists y : \mathbb{Z}, x < y \land y < x$

Solution: UNSAT

Path 4 impossible
Next class:

Hoare Logic

To do:
- Course feedback
- Proposal presentation
- Assignment 3 released
Expressions

What does an expression **compute**?
Symbolic values and environments

**Executing statements** without running expressions
Symbolic evaluation of expressions

**Feasibility** of program executions
Path conditions, concolic execution, and test generation
PRE/POST

ONE STEP

AT A TIME
ANNOTATIONS

SPECIFICATION

FOR PROGRAM
PROPAGATION

WEAKEST

PRECONDITION
Next class:

Hoare Logic

To do:

☐ Course feedback
☐ Proposal presentation
☐ Assignment 3 released