

Lecture 15: Symbolic Execution

17-355/17-665/17-819: Program Analysis

Rohan Padhye

March 22, 2022

* Course materials developed with Jonathan Aldrich and Claire Le Goues

Recall: VCGen from Axiomatic Semantics

 $\{P\}$ $x := e_1$ $x := e_2$ $\{Q\}$

What is the Proof Obligation for backwards reasoning?

What if we just went forwards?

$\{P\}$

$x := e_1$

$x := e_2$

$\{Q\}$

Generate “fresh” math variables
for every mutable program
variable

Proof Obligation:

$$\forall x_n : ([x_0/x]P \wedge x_1 = [x_0/x]e_1 \wedge x_2 = ([x_1/x]e_2)) \Rightarrow [x_2/x]Q$$

What if we just went forwards?

 $\{P\}$ $x := e_1$ $x := e_2$ $\{Q\}$ $\{x > 0\}$ $x := x * 2$ $x := x + 1$ $\{x > 1\}$

Proof Obligation:

$$\forall x_n : ([x_0/x]P \wedge x_1 = [x_0/x]e_1 \wedge x_2 = ([x_1/x]e_2)) \Rightarrow [x_2/x]Q$$
$$\forall x_0, x_1, x_2 \in \mathbb{Z} : (x_0 > 0 \wedge x_1 = x_0 * 2 \wedge x_2 = x_1 + 1) \Rightarrow x_2 > 1$$

Dealing with conditional paths

$\{true\}$

if ($x < 0$) :

$y := -x$

else :

$y := x$

$\{y \geq 0\}$

Dynamic Symbolic Execution:

$$\forall x_0, y_0 \in \mathbb{Z} : (x_0 < 0 \wedge y_0 = -x_0) \Rightarrow y_0 \geq 0$$

$$\forall x_0, y_0 \in \mathbb{Z} : (x_0 \geq 0 \wedge y_0 = x_0) \Rightarrow y_0 \geq 0$$

Static Symbolic Execution:

$$\forall x_0, y_0 \in \mathbb{Z} : ((x_0 < 0 \Rightarrow y_0 = -x_0) \vee (x_0 \geq 0 \Rightarrow y_0 = x_0)) \Rightarrow y_0 \geq 0$$

Dealing with conditional paths

$\{x > 0\}$

if ($x < 0$) :

$y := x$

else :

$y := x$

$\{y \geq 0\}$

Exercise: Generate the VC for this program. Is it true?

Formalizing DSE with Guards and Symbolic Formulas

$\Sigma \in Var \rightarrow a_s$

$g ::=$

- $true$
- $false$
- $not\ g$
- $g_1\ op_b\ g_2$
- $a_{s1}\ op_r\ a_{s2}$

$a_s ::=$

- α
- n
- $a_{s1}\ op_a\ a_{s2}$

Symbolic Evaluation of Expressions

$$\frac{}{\langle \Sigma, n \rangle \Downarrow n} \textit{big-int}$$

$$\frac{}{\langle \Sigma, x \rangle \Downarrow \Sigma(x)} \textit{big-var}$$

$$\frac{\langle \Sigma, a_1 \rangle \Downarrow a_{s1} \quad \langle \Sigma, a_2 \rangle \Downarrow a_{s2}}{\langle \Sigma, a_1 + a_2 \rangle \Downarrow a_{s1} + a_{s2}} \textit{big-add}$$

Symbolic Execution of Statements (DSE)

$$\frac{}{\langle g, \Sigma, \text{skip} \rangle \Downarrow \langle g, \Sigma \rangle} \textit{big-skip}$$

$$\frac{\langle g, \Sigma, s_1 \rangle \Downarrow \langle g', \Sigma' \rangle \quad \langle g', \Sigma', s_2 \rangle \Downarrow \langle g'', \Sigma'' \rangle}{\langle g, \Sigma, s_1; s_2 \rangle \Downarrow \langle g'', \Sigma'' \rangle} \textit{big-seq}$$

$$\frac{\langle \Sigma, a \rangle \Downarrow a_s}{\langle g, \Sigma, x := a \rangle \Downarrow \langle g, \Sigma[x \mapsto a_s] \rangle} \textit{big-assign}$$

Symbolic Execution with Branching (DSE)

$$\frac{\langle \Sigma, b \rangle \Downarrow g' \quad g \wedge g' \text{ SAT} \quad \langle g \wedge g', \Sigma, s_1 \rangle \Downarrow \langle g'', \Sigma' \rangle}{\langle g, \Sigma, \text{if } b \text{ then } s_1 \text{ else } s_2, \rangle \Downarrow \langle g'', \Sigma' \rangle} \textit{big-iftrue}$$

$$\frac{\langle \Sigma, b \rangle \Downarrow g' \quad g \wedge \neg g' \text{ SAT} \quad \langle g \wedge \neg g', \Sigma, s_2 \rangle \Downarrow \langle g'', \Sigma' \rangle}{\langle g, \Sigma, \text{if } b \text{ then } s_1 \text{ else } s_2, \rangle \Downarrow \langle g'', \Sigma' \rangle} \textit{big-iffalse}$$

Symbolic Execution of Loops

Q. What's wrong here?

$$\frac{\langle \Sigma, b \rangle \Downarrow g' \quad g \wedge g' \text{ SAT} \quad \langle g \wedge g', \Sigma, s; \text{while } b \text{ do } s \rangle \Downarrow \langle g'', \Sigma' \rangle}{\langle g, \Sigma, \text{while } b \text{ do } s, \rangle \Downarrow \langle g'', \Sigma' \rangle} \textit{big-whiletrue}$$

$$\frac{\langle \Sigma, b \rangle \Downarrow g' \quad g \wedge \neg g' \text{ SAT}}{\langle g, \Sigma, \text{while } b \text{ do } s, \rangle \Downarrow \langle g \wedge \neg g', \Sigma \rangle} \textit{big-whilefalse}$$

Symbolic Execution of Loops

Bounded exploration (k -limited)

$$\frac{k > 0 \quad \langle \Sigma, b \rangle \Downarrow g' \quad g \wedge g' \text{ SAT} \quad \langle g \wedge g', \Sigma, s; \text{while}_{k-1} b \text{ do } s \rangle \Downarrow \langle g'', \Sigma' \rangle}{\langle g, \Sigma, \text{while}_k b \text{ do } s, \rangle \Downarrow \langle g'', \Sigma' \rangle} \textit{big-whiletrue}$$

$$\frac{\langle \Sigma, b \rangle \Downarrow g' \quad g \wedge \neg g' \text{ SAT}}{\langle g, \Sigma, \text{while}_k b \text{ do } s, \rangle \Downarrow \langle g \wedge \neg g', \Sigma \rangle} \textit{big-whilefalse}$$

Q. What are the implications?

Symbolic Execution with Loops

- Loop invariants can be used if given
 - Often works better with SSE
- But we can choose to explore only partial set of paths
 - K-bounded loops (often: $k < 3$)
 - “Unsound” for verification
 - Sound but “Incomplete” for bug finding when used with DSE
 - DSE formulas for a given path can be solved to find a witness = test input

Digression: Soundness and Completeness

- Soundness = “Doesn’t lie” or “all claims are true”
- Completeness = “All truths are claimed”
- For Verification (claim is “*program is correct*”)
 - Soundness: Reasoning along all possible paths (over-approximation)
- For Bug-Finding (claim is “*a bug exists*”)
 - Soundness: Reasoning along feasible paths only (under-approximation)
- Soundness & Completeness is impossible in general (Rice’s theorem)
 - Most systems are sound but incomplete (e.g. can’t prove all programs, or can’t find all bugs)

Symbolic Execution: A Generalization of Testing

```
1  int x=0, y=0, z=0;
2  if (a) {
3      x = -2;
4  }
5  if (b < 5) {
6      if (!a && c) { y = 1; }
7      z = 2;
8  }
9  assert (x + y + z != 3);
```

What input values of a,b,c will cause the assert to fail?

```
1  int x=0, y=0, z=0;
2  if(a) {
3      x = -2;
4  }
5  if (b < 5) {
6      if (!a && c) { y = 1; }
7      z = 2;
8  }
9  assert(x + y + z != 3);
```

line	g	E


```

1  int x=0, y=0, z=0;
2  if(a) {
3      x = -2;
4  }
5  if (b < 5) {
6      if (!a && c) { y = 1; }
7      z = 2;
8  }
9  assert(x + y + z != 3);

```

line	g	E
0	true	$a \mapsto \alpha, b \mapsto \beta, c \mapsto \gamma$

```

1  int x=0, y=0, z=0;
2  if(a) {
3      x = -2;
4  }
5  if (b < 5) {
6      if (!a && c) { y = 1; }
7      z = 2;
8  }
9  assert(x + y + z != 3);

```

line	g	E
0	true	$a \mapsto \alpha, b \mapsto \beta, c \mapsto \gamma$
1	true	$\dots, x \mapsto 0, y \mapsto 0, z \mapsto 0$

```

1  int x=0, y=0, z=0;
2  if(a) {
3      x = -2;
4  }
5  if (b < 5) {
6      if (!a && c) { y = 1; }
7      z = 2;
8  }
9  assert(x + y + z != 3);

```

line	g	E
0	true	$a \mapsto \alpha, b \mapsto \beta, c \mapsto \gamma$
1	true	$\dots, x \mapsto 0, y \mapsto 0, z \mapsto 0$
2	$\neg\alpha$	$\dots, x \mapsto 0, y \mapsto 0, z \mapsto 0$

```

1  int x=0, y=0, z=0;
2  if(a) {
3      x = -2;
4  }
5  if (b < 5) {
6      if (!a && c) { y = 1; }
7      z = 2;
8  }
9  assert(x + y + z != 3);

```

line	g	E
0	true	$a \mapsto \alpha, b \mapsto \beta, c \mapsto \gamma$
1	true	$\dots, x \mapsto 0, y \mapsto 0, z \mapsto 0$
2	$\neg \alpha$	$\dots, x \mapsto 0, y \mapsto 0, z \mapsto 0$
5	$\neg \alpha \wedge \beta \geq 5$	$\dots, x \mapsto 0, y \mapsto 0, z \mapsto 0$

```

1  int x=0, y=0, z=0;
2  if(a) {
3      x = -2;
4  }
5  if (b < 5) {
6      if (!a && c) { y = 1; }
7      z = 2;
8  }
9  assert(x + y + z != 3);

```

line	g	E
0	true	$a \mapsto \alpha, b \mapsto \beta, c \mapsto \gamma$
1	true	$\dots, x \mapsto 0, y \mapsto 0, z \mapsto 0$
2	$\neg\alpha$	$\dots, x \mapsto 0, y \mapsto 0, z \mapsto 0$
5	$\neg\alpha \wedge \beta \geq 5$	$\dots, x \mapsto 0, y \mapsto 0, z \mapsto 0$
9	$\neg\alpha \wedge \beta \geq 5 \wedge 0 + 0 + 0 \neq 3$	$\dots, x \mapsto 0, y \mapsto 0, z \mapsto 0$

```

1  int x=0, y=0, z=0;
2  if(a) {
3      x = -2;
4  }
5  if (b < 5) {
6      if (!a && c) { y = 1; }
7      z = 2;
8  }
9  assert(x + y + z != 3);

```

Exercise: Generate path constraints for another path.

line	g	E
0	true	$a \mapsto \alpha, b \mapsto \beta, c \mapsto \gamma$
1	true	$\dots, x \mapsto 0, y \mapsto 0, z \mapsto 0$
2	$\neg \alpha$	$\dots, x \mapsto 0, y \mapsto 0, z \mapsto 0$
5	$\neg \alpha \wedge \beta \geq 5$	$\dots, x \mapsto 0, y \mapsto 0, z \mapsto 0$
9	$\neg \alpha \wedge \beta \geq 5 \wedge 0 + 0 + 0 \neq 3$	$\dots, x \mapsto 0, y \mapsto 0, z \mapsto 0$

```

1  int x=0, y=0, z=0;
2  if(a) {
3      x = -2;
4  }
5  if (b < 5) {
6      if (!a && c) { y = 1; }
7      z = 2;
8  }
9  assert(x + y + z != 3);

```

Exercise: Generate path constraints for another path (e.g. one that executes line 6).

line	g	E

Symbolic Execution Tree

Exercise: How many feasible paths are in the program?

```
1  int x=0, y=0, z=0;
2  if (a) {
3      x = -2;
4  }
5  if (b < 5) {
6      if (!a && c) { y = 1; }
7      z = 2;
8  }
9  assert (x + y + z != 3);
```