Symbolic Execution and Program Verification

- We present symbolic execution in SOS as abstract interpretation in the syntactic domain of arithmetic and Boolean expressions.

- A symbolic configuration $\langle S, ss, pc \rangle$ consists of:
  a statement $S \in Stm$
  a symbolic state $ss : Var \to AExp_{LVar}$ initially $x \mapsto x_0, \ldots$
  a path condition $pc \in BExp_{LVar}$ initially true

- We extend While with three statements:

  assume b : halts if b doesn't hold
  assert b : aborts if b doesn't hold
  havoc X : assigns random values to the variables in $X \subseteq Var$

- The intention is that

  $\models_{par} \{P\} S \{Q\}$ if assume P; S; assert Q has no aborting executions

- Complication: executions can be infinite. Possible ways of addressing while loops:
  - unfolding some fixed number k of times (unsound)
  - using user-provided loop invariants to convert into loop-free program

- A path is feasible if all involved path conditions are satisfiable

• Rules for symbolic execution

$[skip_{SE}]$ $\quad \langle skip, ss, pc\rangle \Rightarrow \langle ss, pc\rangle$

$[ass_{SE}]$ $\quad \langle x := a, ss, pc\rangle \Rightarrow \langle ss[x \mapsto SA[\![a]\!](ss)], pc\rangle$

$[asm_{SE}]$ $\quad \langle \text{assume } P, ss, pc\rangle \Rightarrow \langle ss, pc \wedge SB[\![P]\!](ss)\rangle$

$[asr_{SE}]$ $\quad \langle \text{assert } P, ss, pc\rangle \Rightarrow \langle ss, pc\rangle$ if $pc \Rightarrow SB[\![P]\!](ss)$

$[hav_{SE}]$ $\quad \langle \text{havoc } X, ss, pc\rangle \Rightarrow \langle ss[x' \mapsto x'_0, \ldots, x^n \mapsto x^n_0], pc\rangle$
where $\{x', \ldots, x^n\} = X$ and $x'_0, \ldots, x^n_0$ are fresh

$$[comp^1_{SE}] \quad \frac{\langle S_1, ss, pc\rangle \Rightarrow \langle ss', pc'\rangle}{\langle S_1; S_2, ss, pc\rangle \Rightarrow \langle S_2, ss', pc'\rangle}$$

$$[comp^2_{SE}] \quad \frac{\langle S_1, ss, pc\rangle \Rightarrow \langle S_1', ss', pc'\rangle}{\langle S_1; S_2, ss, pc\rangle \Rightarrow \langle S_1'; S_2, ss', pc'\rangle}$$

$[if^1_{SE}]$ $\quad \langle \underline{\text{if}}\ b\ \underline{\text{then}}\ S_1\ \underline{\text{else}}\ S_2, ss, pc\rangle \Rightarrow \langle S_1, ss, pc \wedge SB[\![b]\!](ss)\rangle$

$[if^2_{SE}]$ $\quad \langle \underline{\text{if}}\ b\ \underline{\text{then}}\ S_1\ \underline{\text{else}}\ S_2, ss, pc\rangle \Rightarrow \langle S_2, ss, pc \wedge SB[\![\neg b]\!](ss)\rangle$

$[while^1_{SE}]$ $\quad \langle \underline{\text{while}}\ b\ \underline{\text{do}}\ S, ss, pc\rangle \Rightarrow \langle S; \underline{\text{while}}\ b\ \underline{\text{do}}\ S, ss, pc \wedge SB[\![b]\!](ss)\rangle$

$[while^2_{SE}]$ $\quad \langle \underline{\text{while}}\ b\ \underline{\text{do}}\ S, ss, pc\rangle \Rightarrow \langle ss, pc \wedge SB[\![\neg b]\!](ss)\rangle$

where: $SA[\![a]\!](ss) \stackrel{def}{=} a[ss]$ $\quad SB[\![b]\!](ss) \stackrel{def}{=} b[ss]$

EXAMPLE 1

Program $\text{SWAP} \stackrel{\text{def}}{=} x := x+y ; y := x-y ; x := x-y$

Spec $\{\text{true}\}\ \text{SWAP}\ \{x = y_0 \wedge y = x_0\}$

Converted program, with labels:

L_0 : assume true;
L_1 : $x := x+y$;
L_2 : $y := x-y$;
L_3 : $x := x-y$;
L_4 : assert $x = y_0 \wedge y = x_0$;
L_F :

Symbolic execution:

$\langle L_0, [x \mapsto x_0, y \mapsto y_0], \text{true} \rangle$
$\Rightarrow \langle L_1, [x \mapsto x_0, y \mapsto y_0], \text{true} \rangle$
$\Rightarrow \langle L_2, [x \mapsto x_0+y_0, y \mapsto y_0], \text{true} \rangle$
$\Rightarrow \langle L_3, [x \mapsto x_0+y_0, y \mapsto (x_0+y_0)-y_0], \text{true} \rangle$
$\Rightarrow \langle L_4, [x \mapsto (x_0+y_0)-((x_0+y_0)-y_0), y \mapsto (x_0+y_0)-y_0], \text{true} \rangle$
$\overset{(1)}{\Rightarrow} \langle L_F, [\quad - \cdot - \quad], \text{true} \rangle$

(1) $\text{true} \Rightarrow (x_0+y_0)-((x_0+y_0)-y_0) = y_0 \wedge (x_0+y_0)-y_0 = x_0$ ✓

EXAMPLE 2

Verify $\{x \leq 0\}$ if $0 \leq x$ then $x := x-1$ else $x := 1$ $\{x = 1\}$

- Loop elimination

A while loop annotated with a candidate invariant

$$\{I\} \text{ while } b \text{ do } S$$

is translated to:

```
assert I;
havoc X_S;
assume I;
if b then
    S;
    assert I;
    assume false;
else
    skip;
```

where X_S is the set of all variables that are assigned to in S

EXAMPLE 3

Program $Pow \stackrel{def}{=} p := 1$; **while** $\neg(y=0)$ **do** $p := p * x$; $y := y-1$

Spec $\{true\}\ Pow\ \{p = x_0^{y_0}\}$

Invariant $p * x^y = x_0^{y_0}$

Converted program, with labels:

l_0: assume true;
l_1: $p := 1$;
l_2: assert $p * x^y = x_0^{y_0}$;
l_3: havoc p, y;
l_4: assume $p * x^y = x_0^{y_0}$;
l_5: **if** $\neg(y=0)$ **then**
l_6: $\quad p := p * x$;
l_7: $\quad y := y-1$;
l_8: $\quad$ assert $p * x^y = x_0^{y_0}$;
l_9: $\quad$ assume false;
else
l_{10}: $\quad$ skip
l_{11}: assert $p = x_0^{y_0}$
l_F:

Symbolic execution:

$$\langle l_0, [x \mapsto x_0, y \mapsto y_0, p \mapsto p_0], true \rangle$$
$$\Rightarrow \langle l_1, [x \mapsto x_0, y \mapsto y_0, p \mapsto p_0], true \rangle$$
$$\Rightarrow \langle l_2, [x \mapsto x_0, y \mapsto y_0, p \mapsto 1], true \rangle$$
$$\overset{(1)}{\Rightarrow} \langle l_3, [x \mapsto x_0, y \mapsto y_0, p \mapsto 1], true \rangle$$
$$\Rightarrow \langle l_4, [x \mapsto x_0, y \mapsto y_0', p \mapsto p_0'], true \rangle$$
$$\Rightarrow \langle l_5, [x \mapsto x_0, y \mapsto y_0', p \mapsto p_0'], p_0' * x_0^{y_0'} = x_0^{y_0} \rangle$$
$$\Rightarrow \langle l_6, [x \mapsto x_0, y \mapsto y_0', p \mapsto p_0'], p_0' * x_0^{y_0'} = x_0^{y_0} \wedge \neg(y_0' = 0) \rangle$$
$$\Rightarrow \langle l_7, [x \mapsto x_0, y \mapsto y_0', p \mapsto p_0' * x_0], \text{---} \rangle$$
$$\Rightarrow \langle l_8, [x \mapsto x_0, y \mapsto y_0' - 1, p \mapsto p_0' * x_0], \text{---} \rangle$$
$$\overset{(2)}{\Rightarrow} \langle l_9, [\text{---}], \text{---} \rangle$$
$$\Rightarrow \langle l_{11}, [\text{---}], \text{---} \wedge false \rangle$$

infeasible!

$$\Rightarrow \langle l_{10}, [x \mapsto x_0, y \mapsto y_0', p \mapsto p_0'], p_0' * x_0^{y_0'} = x_0^{y_0} \wedge \neg\neg(y_0' = 0) \rangle$$
$$\Rightarrow \langle l_{11}, [\text{---}], \text{---} \rangle$$
$$\overset{(3)}{\Rightarrow} \langle l_F, [\text{---}], \text{---} \rangle$$

(1) $true \Rightarrow 1 * x_0^{y_0} = x_0^{y_0}$ ✓

(2) $p_0' * x_0^{y_0'} = x_0^{y_0} \wedge \neg(y_0' = 0) \Rightarrow (p_0' * x_0) * x_0^{(y_0' - 1)} = x_0^{y_0}$ ✓

(3) $p_0' * x_0^{y_0'} = x_0^{y_0} \wedge \neg\neg(y_0' = 0) \Rightarrow p_0' = x_0^{y_0}$ ✓