Symbolic Execution Semantics for WHILE Language

Arie Gurfinkel

April 2, 2017

1 Symbolic Execution Semantics

Symbolic execution semantics of the WHILE language are shown in Fig. 1. The judgement in symbolic execution has a form $\langle s, q, pc\rangle \Downarrow q', pc'$, where s is a statement, q and q' are the input and output symbolic environments, respectively, and pc and pc' are input and output path conditions, respectively. A symbolic environment maps program variables to symbolic expressions. A path condition is a formula over symbolic expressions. Note that since WHILE language does not have inputs, symbolic expressions are introduced into the state by the **havoc** statement.

Unlike the concrete operational semantics, a statement might have several executions. For example, both branches of an if-statement can be evaluated if both the condition b and its negation $\neg b$ are consistent with the current path condition pc.

$$\frac{}{\langle \textbf{skip}, q, pc\rangle \Downarrow q, pc}$$

$$\frac{}{\langle \textbf{print\_state}, q, pc\rangle \Downarrow q, pc}$$

$$\frac{\langle s_1, q, pc\rangle \Downarrow q'', pc'' \qquad \langle s_2, q'', pc''\rangle \Downarrow q', pc'}{\langle s_1 \,;\, s_2, q, pc\rangle \Downarrow q', pc'}$$

$$\frac{\langle e, q\rangle \Downarrow n}{\langle x := e, q, pc\rangle \Downarrow q[x := n], pc}$$

$$\frac{\langle b, q\rangle \Downarrow v \qquad pc \wedge v \text{ is SAT} \qquad \langle s_1, q, pc \wedge v\rangle \Downarrow q', pc'}{\langle \textbf{if } b \textbf{ then } s_1 \textbf{ else } s_2, q, pc\rangle \Downarrow q', pc'}$$

$$\frac{\langle b, q\rangle \Downarrow v \qquad pc \wedge \neg v \text{ is SAT} \qquad \langle s_2, q, pc \wedge \neg v\rangle \Downarrow q', pc'}{\langle \textbf{if } b \textbf{ then } s_1 \textbf{ else } s_2, q, pc\rangle \Downarrow q', pc'}$$

$$\frac{\langle b, q\rangle \Downarrow v \qquad pc \wedge \neg v \text{ is SAT}}{\langle \textbf{while } b \textbf{ do } s, q, pc\rangle \Downarrow q, pc \wedge \neg v}$$

$$\frac{\langle b, q\rangle \Downarrow v \qquad pc \wedge v \text{ is SAT} \qquad \langle s \,;\, \textbf{while } b \textbf{ do } s, q, pc \wedge v\rangle \Downarrow q', pc'}{\langle \textbf{while } b \textbf{ do } s, q, pc\rangle \Downarrow q', pc'}$$

$$\frac{V \text{ is a fresh constant}}{\langle \textbf{havoc } x, q, pc\rangle \Downarrow q[x := V]}$$

Figure 1: Symbolic Execution Semantics of the WHILE language.

2 Operational Semantics

Operational semantics for the WHILE language is shown in Fig. 2. The judgement in symbolic execution has a form $\langle s, q\rangle \Downarrow q'$, where s is a statement, q and q' are the input and output states, respectively. A (concrete) state (or an environment) maps program variables to integers. Except for **havoc** statement, the rules are deterministic – each input state has only one legal output state.

3 Operational Semantics of Concolic Execution

Let L be the set of program variables. For concolic execution, they are partitioned into symbolic, $Sym(L)$, and concrete, $Con(L)$, variables. It is possible that all variables are symbolic, i.e., $Con(L) = \emptyset$. For a variable a, we write

$$\frac{}{\langle \textbf{skip}, q\rangle \Downarrow q} \qquad \frac{}{\langle \textbf{print\_state}, q\rangle \Downarrow q} \qquad \frac{\langle s_1, q\rangle \Downarrow q'' \qquad \langle s_2, q''\rangle \Downarrow q'}{\langle s_1 \,;\, s_2, q\rangle \Downarrow q'}$$

$$\frac{\langle e, q\rangle \Downarrow n}{\langle x := e, q\rangle \Downarrow q[x := n]} \qquad \frac{\langle b, q\rangle \Downarrow \text{true} \qquad \langle s_1, q\rangle \Downarrow q'}{\langle \textbf{if } b \textbf{ then } s_1 \textbf{ else } s_2, q\rangle \Downarrow q'} \qquad \frac{\langle b, q\rangle \Downarrow \text{false} \qquad \langle s_2, q\rangle \Downarrow q'}{\langle \textbf{if } b \textbf{ then } s_1 \textbf{ else } s_2, q\rangle \Downarrow q'}$$

$$\frac{\langle b, q\rangle \Downarrow \text{false}}{\langle \textbf{while } b \textbf{ do } s, q\rangle \Downarrow q} \qquad \frac{\langle b, q\rangle \Downarrow \text{true} \qquad \langle s \,;\, \textbf{while } b \textbf{ do } s, q\rangle \Downarrow q'}{\langle \textbf{while } b \textbf{ do } s, q\rangle \Downarrow q'} \qquad \frac{}{\langle \textbf{havoc } x, q\rangle \Downarrow q[x := n]}$$

Figure 2: Operational semantics of the WHILE language.

$Sym(a)$ and $Con(a)$ to indicate that it is symbolic or concrete, respectively.

A state of concolic execution is a triple $q = \langle c, s, pc\rangle$, where c is a concrete state, s a symbolic environment, and pc is a formula called the *path condition*. Given a state $q = \langle c, s, pc\rangle$, we write $con(q)$ for c, $sym(q)$ for s, and $pc(q)$ for pc. Symbolic environment, path condition, and concrete state are as in symbolic and concrete execution, respectively. However, concrete state also has a value for every symbolic variable. We call those *concrete shadows*. That is, if b is a symbolic variable and q a concolic state, then $sym(q)(b)$ is the symbolic value of b, and $con(q)(b)$ is the value of the concrete shadow of b. Given two concrete states c_1 and c_2, we write $c_1 \equiv_{con} c_2$ to indicate that that they are identical on the concrete variables:

$$c_1 \equiv_{con} c_2 \iff \forall a \in Con(L) \cdot c_1(a) = c_2(a)$$

Given a concrete state c and a symbolic state $\langle s, pc\rangle$, we write $c \models \langle s, pc\rangle$ to indicate that the concrete state c is contained in the symbolic state.

The semantics of expressions is as usual with variables evaluated based on their kind: concrete variables are evaluated over $con(q)$ and symbolic over $sym(q)$:

$$\frac{con(a) \qquad \langle a, con(q)\rangle \Downarrow v}{\langle a, q\rangle \Downarrow v} \qquad \frac{sym(a) \qquad \langle a, sym(q)\rangle \Downarrow v}{\langle a, q\rangle \Downarrow v}$$

For most statements, the semantics is extended by applying both symbolic and concrete operational semantics in parallel. The last pre-condition ensures that the concrete and symbolic states are chosen consistently.

$$\frac{\langle s, con(q)\rangle \Downarrow c \qquad \langle s, sym(q), pc(q)\rangle \Downarrow s', pc' \qquad c \models \langle s', pc'\rangle}{\langle s, q\rangle \Downarrow \langle c', s', pc'\rangle}$$

Assignment of values to concrete variables is limited to concrete values only. Thus, it is not possible to assign symbolic variables (or symbolic expressions) to concrete variables. This can be done by either treating all variables as symbolic (i.e., $Con(L) = \emptyset$), or concretizing symbolic state before assignment (which we

describe later on).

$$\frac{\langle e, q\rangle \Downarrow n \qquad con(x) \qquad n \in \mathbb{Z}}{\langle x := e, q\rangle \Downarrow q[x := n]}$$

Assignment to symbolic variables also assigns to their concrete shadows.

At if-statement, concolic execution can chose to switch to the branch that is not consistent with current concrete state, as long as the concrete state can be adjusted. We only show one of the cases:

$$\frac{\begin{array}{c}\langle b, con(q)\rangle \Downarrow \text{true} \\ \langle b, q\rangle \Downarrow v \qquad pc(q) \wedge \neg v \text{ is SAT} \qquad c \models \langle sym(q), pc(q) \wedge \neg v\rangle \\ c \equiv_{con} con(q) \qquad \langle s_2, \langle c, sym(q), pc(q) \wedge \neg v\rangle\rangle \Downarrow q'\end{array}}{\langle \textbf{if } b \textbf{ then } s_1 \textbf{ else } s_2, q\rangle \Downarrow q'}$$

In this case, according to the concrete state the branch condition b is true. At the same time, according to symbolic state, the negation of the branch condition $\neg b$ and the current path condition are satisfiable. The concolic execution can proceed according to the else-branch, but the concolic state needs to be first updated such that:

1. the concrete portion is updated to be consistent with the negation of the branch condition
2. the path condition is extended with the negation of the path condition.

Note that it is only possible to take the else-branch if the condition value can be controlled by the symbolic part of the state. Branch conditions that do not depend on the input (such as iterations of the loops) can only be resolved one way (i.e., either true or false).

Finally, concolic execution semantics provide *concretization* step that allows to turn symbolic variables (or values) to their concrete values in the current concrete state. The effect of concretization is captured by the so called *concretization constraints* in the path condition:

$$\frac{\begin{array}{cc} sym(x) & \langle x, con(q)\rangle \Downarrow n \\ \langle x, sym(q)\rangle \Downarrow v & \langle s, \langle con(q), sym(q)[x := n], pc(q) \wedge v = n\rangle\rangle \Downarrow q'\end{array}}{\langle s, q\rangle \Downarrow q'}$$

The rule says that if x is a symbolic variable with symbolic value v and it is currently shadowed concretely by a concrete value n, then we can update the symbolic value of x to n as long as we also update the path condition with $v = n$ to reflect the concretization step.