Mini-SymEx – Rules

Alexander Weigl

March 19, 2021

Definition 1. *A configuration of the symbolic execution is a tuple of a function $\sigma\colon Var \to Var$, a set of assumptions $A \subseteq Fml_\Sigma$, and the remaining program R. A configuration is written as $\langle \sigma,\ A,\ R\rangle$.*

Our rules use construct the single static assignment form (SSA) dynamical. Using SSA reduces the size of verification, but makes the handling of branching a bit more difficult.

Some small notations in the rules below:

- ε denotes the empty program.
- $e[\sigma]$ denotes the application of the substitution σ on the expression e. This is equivalent to the symbolical evaluation of the expression.
- $\sigma[x \to y]$ is the update of the function at position x to value y.
- The function $fresh$ always returns a completely new unused variable.

$$\frac{\langle \sigma,\ A,\ S_1\rangle \leadsto \langle \sigma',\ A',\ \varepsilon\rangle}{\langle \sigma,\ A,\ S_1\,;S_2\rangle \leadsto \langle \sigma',\ A',\ S_2\rangle}\texttt{comp} \tag{1}$$

$$\frac{}{\langle \sigma,\ A,\ \texttt{assume}\, e\rangle \leadsto \langle \sigma',\ A \cup e[\sigma],\ \varepsilon\rangle}\texttt{assume} \tag{2}$$

$$\frac{A \implies e[\sigma]}{\langle \sigma,\ A,\ \texttt{assert}\, e\rangle \leadsto \langle \sigma',\ A \cup e[\sigma],\ \varepsilon\rangle}\texttt{assert} \tag{3}$$

Rule `assert` is the only rule which charges a verification condition, in particular, $A \implies e[\sigma]$ must be valid. For SMT-solving, we apply the deduction theorem and check $A \land \neg e[\sigma]$ for unsat.

$$\frac{a' = fresh() \quad A' = A \cup \{a' = e[\sigma]\} \quad \sigma' = \sigma[a \to a']}{\langle \sigma,\ A,\ a := e\rangle \leadsto \langle \sigma',\ A',\ \varepsilon\rangle}\texttt{assign} \tag{4}$$

$$\frac{a' = fresh() \quad \sigma' = \sigma[a \to a']}{\langle \sigma,\ A,\ \texttt{havoc}\, a\rangle \leadsto \langle \sigma',\ A,\ \varepsilon\rangle}\texttt{havoc} \tag{5}$$

$$\frac{\begin{array}{cc} (T) & \langle \sigma,\ A,\ \texttt{assume}\ c\,;S_1\rangle \leadsto \langle \sigma_T,\ A_T,\ \varepsilon\rangle \\ (E) & \langle \sigma,\ A,\ \texttt{assume}\ \neg c\,;S_2\rangle \leadsto \langle \sigma_E,\ A_E,\ \varepsilon\rangle \\ (M\sigma) & \sigma' = \sigma \cup \{v \mapsto fresh() \mid \sigma_T[v] \neq \sigma_E[v]\} \\ (MA) & A' = A \cup \{\sigma'(v) = \texttt{ite}(c[\sigma], \sigma_T[v], \sigma_E[v]) \mid \sigma_T[v] \neq \sigma_E[v]\} \end{array} \quad A'}{\langle \sigma,\ A,\ \texttt{if}\ (c)\ S_1\ \texttt{else}\ S_2\rangle \leadsto \langle \sigma',\ A',\ \varepsilon\rangle}\texttt{if} \tag{6}$$

The cases (T) and (E) should be clear: (T) is the execution of then-branch of the if-statement, and (E) for the else-branch, respectively. After both branches are executed, we need to merge their state and assumption for conflicting (re-assigned) variables. Every conflicting variable is assigned to a fresh variable (Mσ). In the assumption, we make a case distinction (`ite`), whether to use value and assumptions of the then- or else-branch.

$$\frac{\begin{array}{ll}(I) & \langle \sigma,\ A,\ \texttt{assert}\ \mathit{Inv}\rangle \\ (P) & \langle \sigma,\ A,\ \texttt{havoc}\ E\,\texttt{;}\,\texttt{assume}\ \mathit{Inv} \wedge c\,\texttt{;}\,S\,\texttt{;}\,\texttt{assert}\ \mathit{Inv}\rangle \\ (T) & \langle \sigma,\ A,\ \texttt{havoc}\ E\,\texttt{;}\,\texttt{assume}\ \mathit{Inv} \wedge \neg c\rangle \rightsquigarrow \langle \sigma',\ A',\ \texttt{skip}\rangle\end{array}}{\langle \sigma,\ A,\ \texttt{while}(c)\ S\rangle \rightsquigarrow \langle \sigma',\ A',\ \varepsilon\rangle}\,\texttt{while} \tag{7}$$

Note that *Inv* is the invariant of the for the given loop, and E is a set of variables. This sets contains all variables which are written during in the loop body S, and are therefore erased (havoc'd) when proving the preservation and termination of the loop.

$$\frac{\begin{array}{ll}(C) & \langle id,\ \{pre_f\},\ body_f\,\texttt{;}\,\texttt{assert}\ post\rangle \\ (U) & A \implies pre_f[\{x_i \mapsto ... \mid 1 \leq i \leq n\}] \\ (T) & \sigma' = \sigma[v \mapsto fresh()] \\ (T) & A' = A \cup \{post_f[\texttt{result}/\sigma(v)\}\end{array}}{\langle \sigma,\ A,\ v{:=}f(x_1, \ldots, x_n)\rangle \rightsquigarrow \langle \sigma',\ A',\ \varepsilon\rangle}\,\texttt{while} \tag{8}$$