A Recursive Strategy for Symbolic Execution Expressed in Coq

Alyssa Byrnes

July 2, 2018

1 Data Types

- Object E to represent set of concrete states.
- Object of type n to represent nodes on the symbolic execution tree.
- Contains initial configuration state $\mathcal{T}_{Cfg}$.
  - get_s(n) returns s, the symbolic state.
  - get_pc(n) returns π, the path constraint.
- Object of type $\mathcal{E}$ to represent a symbolic execution tree made of objects of type n.
  - is_leaf($\mathcal{E}$, n) returns *true* if n is a leaf in $\mathcal{E}$.
  - is_root($\mathcal{E}$, n) returns *true* if n is the root in $\mathcal{E}$.
  - get_root($\mathcal{E}$) returns object of type n that is the root of the tree.
- Object of type φ to represent symbolic state.
  - $B =$ sym_ex(A), or $A \Rightarrow^S_{\mathcal{S}} B$, represents symbolic execution of state A.
- Object of type γ to represent concrete state.
  - $B =$ conc_ex(A), or $A \Rightarrow_{\mathcal{S}} B$, represents concrete execution of state A.

Shorthand:

- $\bar{s}_{r,m} = \text{get\_s}(\text{get\_root}(\mathcal{E}_m))$
- $\pi_{r,m} = \text{get\_pc}(\text{get\_root}(\mathcal{E}_m))$
- $\bar{s}_{l,m} = \text{get\_s}(n_{l,m})$, where $n_{l,m} \in \mathcal{E}_m$.
- $\pi_{l,m} = \text{get\_pc}(n_{l,m})$, where $n_{l,m} \in \mathcal{E}_m$.

2 Accepted Knowledge

These are the properties outlined in Arusoaie et al.'s paper [ALR13].

Lemma 1 *If $\gamma \Rightarrow_{\mathcal{S}} \gamma'$, and $\gamma \in [\![\varphi]\!]$, then there exists φ' such that $\gamma' \in [\![\varphi']\!]$ and $[\varphi]_\sim \Rightarrow^S_{\mathcal{S}} [\varphi']_\sim$. [ALR13]*

Lemma 2 *If $\gamma' \in [\![\varphi']\!]$ and $[\varphi]_\sim \Rightarrow^S_{\mathcal{S}} [\varphi']_\sim$ then there exists $\gamma' \in \mathcal{T}_{Cfg}$ such that $\gamma \Rightarrow_{\mathcal{S}} \gamma'$ and $\gamma \in [\![\varphi]\!]$. [ALR13]*

Lemma 1 states that all concrete states have corresponding symbolic representations, and Lemma 2 states that if a program symbolically executes to a set of possible concrete assignments, then initial concrete assignments exist so the program can concretely execute to that state.

3 Circle Operations

These use the definitions defined in Arusoaie et al.'s paper [ALR13].

Definition 1 *circle_op_1 = the set* $\gamma \in [\![\varphi]\!] \; \forall \; \gamma' \in [\![\varphi']\!]$ *of a given* φ' *where* $[\varphi]_\sim \Rightarrow^S_{\mathcal{S}} [\varphi']_\sim$ *such that* $\gamma \Rightarrow_{\mathcal{S}} \gamma'$.

Definition 2 *circle_op_2 = the set* $\gamma' \in [\![\varphi']\!] \; \forall \; \gamma \in [\![\varphi]\!]$ *of a given* φ *where* $[\varphi]_\sim \Rightarrow^S_{\mathcal{S}} [\varphi']_\sim$ *such that* $\gamma \Rightarrow_{\mathcal{S}} \gamma'$.

circle_op_1 represents all concrete states that will take us down exactly one path in the symbolic execution tree. circle_op_2 represents all concrete output states of a given path in the symbolic execution tree.

4 Properties

For a given $E, X =$ a sequence $\mathcal{E}_0, ..., \mathcal{E}_m$ such that $\forall \mathcal{E}_x$, $\exists n_{l,x}$ such that is_leaf($\mathcal{E}_x$, $n_{l,x}$) = *true* $\rightarrow$ the conjunction of the following:

1. $s_0 \in$ circle_op_1($\bar{s}_{r,0}$, $\pi_{l,0}$)
2. $E \bigcap$ circle_op_2($\bar{s}_{l,m}$, $\pi_{l,m}$) $\neq \{\}$
3. for $j = \{0, ..., m-1\}$, circle_op_2($\bar{s}_{l,j}$, $\pi_{l,j}$) $\subseteq$ circle_op_1($\bar{s}_{r,j+1}$, $\pi_{l,j+1}$)

References

[ALR13] Andrei Arusoaie, Dorel Lucanu, and Vlad Rusu. A generic framework for symbolic execution. In *International Conference on Software Language Engineering*, pages 281–301. Springer, 2013.