

A Recursive Strategy for Symbolic Execution Expressed in Coq

Alyssa Byrnes

July 6, 2018

1 Data Types

- Object of type n to represent nodes on the symbolic execution tree.
 - $\text{get_phi}(n)$ returns φ , the symbolic state.
 - $\text{get_pc}(n)$ returns π , the path constraint.
- Object of type \mathcal{E} to represent a symbolic execution tree made of objects of type n .
 - $\text{is_leaf}(\mathcal{E}, n)$ returns *true* if n is a leaf in \mathcal{E} .
 - $\text{is_root}(\mathcal{E}, n)$ returns *true* if n is the root in \mathcal{E} .
 - $\text{get_root}(\mathcal{E})$ returns object of type n that is the root of the tree.
- Object of type φ to represent symbolic state.
 - $B = \text{sym_ex}(A)$, or $A \Rightarrow_{\mathcal{S}}^* B$, represents symbolic execution of state A for one clock cycle, where A and B are of type φ .
 - $\text{unif}(A)$, or $\llbracket A \rrbracket$ represents the set of concrete states represented by symbolic state A .
- Object of type γ to represent concrete state.
 - $B = \text{conc_ex}(A)$, or $A \Rightarrow_{\mathcal{S}}^* B$, represents concrete execution of state A for one clock cycle, where A and B are of type γ .
- Object E to represent set of concrete states.
 - Contains set of initial configuration states $\mathcal{T}_{Cf\bar{g}}$.

Shorthand:

- $\bar{s}_{r,m} = \text{get_phi}(\text{get_root}(\mathcal{E}_m))$
- $\pi_{r,m} = \text{get_pc}(\text{get_root}(\mathcal{E}_m))$
- $\bar{s}_{l,m} = \text{get_phi}(n_{l,m})$, 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 $\gamma \in \llbracket \varphi \rrbracket$, then there exists φ' such that $\gamma' \in \llbracket \varphi' \rrbracket$ and $[\varphi]_{\sim} \Rightarrow_{\mathcal{S}}^S [\varphi']_{\sim}$. [ALR13]*

Corollary 1 *For every concrete execution $\gamma_0 \Rightarrow_{\mathcal{S}} \gamma_1 \Rightarrow_{\mathcal{S}} \dots \gamma_n \Rightarrow_{\mathcal{S}} \dots$, and pattern φ_0 such that $\gamma_0 \in \llbracket \varphi_0 \rrbracket$, there exists a symbolic execution $[\varphi_0]_{\sim} \Rightarrow_{\mathcal{S}}^S [\varphi_1]_{\sim} \Rightarrow_{\mathcal{S}}^S \dots [\varphi_n]_{\sim} \Rightarrow_{\mathcal{S}}^S \dots$ such that $\gamma_i \in \llbracket \varphi_i \rrbracket$ for all $i = 0, 1, \dots$ [ALR13]*

Lemma 2 If $\gamma' \in \llbracket \varphi' \rrbracket$ and $[\varphi]_{\sim} \Rightarrow_S^S [\varphi']_{\sim}$ then there exists $\gamma \in \mathcal{T}_{Cf_g}$ such that $\gamma \Rightarrow_S \gamma'$ and $\gamma \in \llbracket \varphi \rrbracket$. [ALR13]

Corollary 2 For every feasible symbolic execution $[\varphi_0]_{\sim} \Rightarrow_S^S [\varphi_1]_{\sim} \Rightarrow_S^S \dots [\varphi_n]_{\sim} \Rightarrow_S^S \dots$ there is a concrete execution $\gamma_0 \Rightarrow_S \gamma_1 \Rightarrow_S \dots \gamma_n \Rightarrow_S \dots$ such that $\gamma_i \in \llbracket \varphi_i \rrbracket$ for all $i = 0, 1, \dots$ [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.

Corollaries 1 and 2 lift this definition to a consecutive sequence of executions. From here on, a sequence symbolic executions of an unspecified, finite length, $[\varphi_0]_{\sim} \Rightarrow_S^S [\varphi_1]_{\sim} \Rightarrow_S^S \dots [\varphi_n]_{\sim}$ will be denoted by $[\varphi_0]_{\sim} \Rightarrow_S^* [\varphi_n]_{\sim}$. A sequence of concrete executions of an unspecified, finite length, $\gamma_0 \Rightarrow_S \gamma_1 \Rightarrow_S \dots \gamma_n \Rightarrow_S$ will be denoted by $\gamma_0 \Rightarrow_S^* \gamma_n$.

3 Circle Operations

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

Definition 1 $circle_op_1 =$ the set $\gamma \in \llbracket \varphi \rrbracket \forall \gamma' \in \llbracket \varphi' \rrbracket$ of a given φ' where $[\varphi]_{\sim} \Rightarrow_S^* [\varphi']_{\sim}$ such that $\gamma \Rightarrow_S^* \gamma'$ and $is_leaf(\varphi') = true$.

Definition 2 $circle_op_2 =$ the set $\gamma' \in \llbracket \varphi' \rrbracket \forall \gamma \in \llbracket \varphi \rrbracket$ of a given φ where $[\varphi]_{\sim} \Rightarrow_S^S [\varphi']_{\sim}$ such that $\gamma \Rightarrow_S \gamma'$ and $is_leaf(\varphi') = true$.

$circle_op_1$ 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 $\mathcal{E}_0, \dots, \mathcal{E}_m$ such that $\forall \mathcal{E}_x, \exists n_{l,x}$ such that the conjunction of the following is true:

1. $s_0 \in circle_op_1(\bar{s}_{r,0}, \pi_{l,0})$
2. $E \cap circle_op_2(\bar{s}_{l,m}, \pi_{l,m}) \neq \{\}$
3. for $j = \{0, \dots, m-1\}$, $circle_op_2(\bar{s}_{l,j}, \pi_{l,j}) \subseteq circle_op_1(\bar{s}_{r,j+1}, \pi_{l,j+1})$
4. $is_leaf(\mathcal{E}_x, n_{l,x}) = true$.

5 To Do

- Reason about equivalence class, and how the notation transfers to our notation.
- Representing unification in Coq.
- Representing sequences in Coq.
- Representing sufficiency requirement in Coq.
- Representing “equivalence” of concrete states.

6 Notes for Consideration

- We need to consider uniqueness. This might come naturally from the overall tree structure.
- Our work assumes SAT-solver correctness.
- Our $circle_op$ returns concrete states, so we just need to be able to compare concrete states.

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.