Symbolic Analysis

Xiangyu Zhang

What is Symbolic Analysis

- Static analysis considers all paths are feasible
- Dynamic considers one path or a number of paths
- Symbolic analysis reasons about path feasibility
  - Much more precise
  - Scalability is an issue
- A lot of applications
  - Input generation
  - Vulnerability detection/Fuzzing
  - Verification
  - Many many others

An Example

```
1:   x=input()
2:   if (x>0)
3:      y=...;
4:   else
5:      y=...;
6:   if (x>10)
10:     z=y
```

Basic Idea

- Explore individual paths in the program; models the conditions and the symbolic values along a path to a symbolic constraint; a path is feasible if the corresponding constraint is satisfiable (SAT)
- Similar to our per-path static analysis, a worklist is used to maintain the paths being explored
- Upon a function invocation, the current worklist is pushed to a stack and a new worklist is initialized for path exploration within the callee
- Upon a return, the symbolic value of the return variable is passed back to the caller

Another Example

```
x=input()
if (x>0)
   y=…;
else
   y=…;
t= f (x)
if (t>0)
  z=y

f (k) {
  if (k<=-10)
        return  k+10;
  else
        return k;
```

Language

Program	$p ::= m^*$
Function	$m ::= f(x)\{s\}$
Constant	$c ::= \ldots \mid -1 \mid 0 \mid 1 \mid \ldots \mid \mathbf{true} \mid \mathbf{false}$
Expression	$e ::= x \mid \mathrm{c} \mid e_1 \,\mathbf{op}\, e_2$
Statement	$s ::= x = e \mid x = f(e) \mid x = \mathbf{unknown}() \mid s_1 ; s_2 \mid$ $\mathbf{if}\ (e)\ \mathbf{then}\ s_1\ \mathbf{else}\ s_2 \mid \mathbf{ret}^{\mathrm{f}}\, x$
Operators	$\mathbf{op} ::= + \mid - \mid \ldots \mid > \mid \geq \mid = \mid \neq \mid \wedge \mid \vee \mid \ldots$

Definitions

$$v \in SymValue \qquad\qquad \rho \in Constraint$$

$$\sigma \in SymStore \quad ::= \quad Variable \mapsto SymValue$$

$$\omega \in Worklist \quad ::= \quad \mathcal{P}(Stmt \times Constraint \times SymStore)$$

$$\Omega \in WLStack \quad ::= \quad \overline{Worklist}$$

$$\gamma \in RetConstraint \quad ::= \quad \mathcal{P}(Constraint)$$

$$\Gamma \in RCStack \quad ::= \quad \overline{RetConstraint}$$

$\sigma\downarrow \; ::= \bigwedge_{x \in \sigma} x = \sigma(x)$, the operator turns a store to a logical conjunction.

$\langle s, \rho, \sigma \rangle \bowtie_x^f \gamma \; ::= \bigcup_{\rho' \in \gamma} \{\langle s, \rho \wedge \rho', \sigma[x \mapsto rt_f] \rangle\}$, the operator propagates the constraints collected in the callee to the caller for an invocation $x = f(e)$; s is the statement right after the invocation. rt_f represents the return value of f.

select(ω) chooses the next symbolic state $\langle s, \rho, \sigma \rangle$ to explore.

follow(ρ) determines the feasibility of a path condition ρ and realizes some user specified path pruning heuristics.

Symbolic Execution Semantics

EXPRESSION RULES $\boxed{e \twoheadrightarrow v}$

$$\frac{}{c \twoheadrightarrow c} \quad \text{(E-CONST)} \qquad \frac{}{x \twoheadrightarrow x} \quad \text{(E-VAR)}$$

$$\frac{e_1 \twoheadrightarrow v_1 \quad e_2 \twoheadrightarrow v_2}{e_1 \,\mathbf{op}\, e_2 \twoheadrightarrow v_1 \,\boldsymbol{op}\, v_2} \quad \text{(E-OP)}$$

Symbolic Execution Semantics

STATEMENT RULES $\boxed{\langle s, \rho, \sigma \rangle \to \mathcal{P}(\langle s, \rho, \sigma \rangle)}$

$$\frac{e \twoheadrightarrow \rho_0}{\langle x = e; s,\ \rho, \sigma \rangle \to \{\langle s, \rho, \sigma[x \mapsto \rho_0] \rangle\}} \quad \text{(S-ASSIGN)}$$

$$\frac{e \twoheadrightarrow v \qquad \rho_1 = \rho \land v \qquad \rho_2 = \rho \land \neg v \\ follow(\rho_1 \land \sigma \downarrow) = \mathbf{true} \quad follow(\rho_2 \land \sigma \downarrow) = \mathbf{false}}{\langle \mathbf{if}\ e\ \mathbf{then}\ s_1\ \mathbf{else}\ s_2; s,\ \rho, \sigma \rangle \to \{\langle s_1; s, \rho_1, \sigma \rangle\}} \quad \text{(S-IF-T)}$$

$$\frac{e \twoheadrightarrow v \qquad \rho_1 = \rho \land v \qquad \rho_2 = \rho \land \neg v \\ follow(\rho_1 \land \sigma \downarrow) = \boldsymbol{true} \quad follow(\rho_2 \land \sigma \downarrow) = \boldsymbol{true}}{\langle \mathbf{if}\ e\ \mathbf{then}\ s_1\ \mathbf{else}\ s_2; s,\ \rho, \pi, \sigma \rangle \to \\ \{\langle s_1; s, \rho_1, \sigma \rangle, \langle s_2; s, \rho_2, \sigma \rangle\ \}} \quad \text{(S-IF-BOTH)}$$

$$\frac{r \text{ a fresh symbolic variable}}{\langle x = \mathbf{unknown}(); s,\ \rho, \sigma \rangle \to \{\langle s, \rho, \sigma[x \mapsto r] \rangle\}} \quad \text{(S-UNINTPRT)}$$

Symbolic Execution Semantics

GLOBAL RULES $\boxed{\langle \Omega, \Gamma \rangle \Rightarrow \langle \Omega', \Gamma' \rangle}$

$$\frac{\langle s, \rho, \sigma \rangle = select(\omega) \quad \langle s, \rho, \sigma \rangle \to t}{\langle \Omega \circ \omega, \Gamma \rangle \Rightarrow \langle \Omega \circ \omega[t / \langle s, \rho, \sigma \rangle], \Gamma \rangle} \quad \text{(G-STMT)}$$

$$\frac{\langle \mathrm{y} = f(e); s_0, \rho, \sigma \rangle = select(\omega) \quad f(x)\{s\} \text{ is a method} \qquad e \twoheadrightarrow v \quad \omega' = \{\langle s, \rho, \sigma[x \mapsto v \ \ ] \rangle\}}{\langle \Omega \circ \omega, \Gamma \rangle \Rightarrow \langle \Omega \circ \omega \circ \omega', \Gamma \circ \{\} \rangle} \quad \text{(G-CALL)}$$

$$\frac{\langle \mathbf{ret}_f\ x, \rho, \sigma \rangle = select(\omega) \quad f(y)\{s\} \text{ is a method} \qquad \omega' = \omega - \langle \mathbf{ret}_f\ x, \rho, \sigma \rangle \qquad \gamma' = \gamma \cup \quad \{\rho \wedge rt_f = \sigma(x)\}}{\langle \Omega \circ \omega, \Gamma \circ \gamma \rangle \Rightarrow \langle \Omega \circ \omega', \Gamma \circ \gamma' \rangle} \quad \text{(G-RET)}$$

$$\frac{\langle x = f(e); s, \rho, \sigma \rangle = select(\omega) \qquad \omega' = \omega[\langle s, \rho, \sigma \rangle \bowtie_x^f \gamma \,/\, \langle x = f(e); s, \rho, \sigma \rangle]}{\langle \Omega \circ \omega \circ \{\}, \Gamma \circ \gamma \rangle \Rightarrow \langle \Omega \circ \omega', \Gamma \rangle} \quad \text{(G-POST-CALL)}$$

A More Realistic Example

```
int readData(char type){
  int sum=0;
  if('F'==type){
     Scanner cin=new Scanner(FileReader("input"));
     while(cin.hasNext()){     // cin.hasNext()*
       sum += cin.nextInt();   // cin.nextInt() *
     }
     cin.close();
  }else{
     Socket s=new Socket("1.1.1.1");
     while(s.hasNext()){       // hasNext() *
        sum += s.nextInt();    // nextInt() *
     }
     s.close();
  }
  return sum;
}
int readAndNoti(){
     int type=readUserInput(); //readUserInput() *
     int s= readData(type);
     if( s>=0 )
            print("Zero or Positive");
      else
             print("negative");
     return s;
}
void main(){
  int  rawInput=readAndNoti();
  assert(rawInput>=0);
  …
}
```

Constraints

$$
\begin{aligned}
C1&: \ 'F' = type \wedge \neg x_1 \wedge RET = 0,\\
C2&: \ 'F' = type \wedge x_1 \wedge \neg x_2 \wedge RET = y_1,\\
C3&: \ 'F' = type \wedge x_1 \wedge x_2 \wedge \neg x_3 \wedge RET = y_1 + y_2,\\
C4&: \ 'F' \neq type \wedge \neg w_1 \wedge RET = 0,\\
C5&: \ 'F' \neq type \wedge w_1 \wedge \neg w_2 \wedge RET = z_1,\\
C6&: \ 'F' \neq type \wedge w_1 \wedge w_2 \wedge \neg w_3 \wedge RET = z_1 + z_2.
\end{aligned}
$$

$$
\begin{aligned}
C7&: \ 'F' = type \wedge \neg x_1 \wedge RET = 0 \wedge RET = RET2 \wedge RET2 < 0,\\
C8&: \ 'F' = type \wedge x_1 \wedge \neg x_2 \wedge RET = y_1 \wedge RET = RET2 \wedge RET2 < 0,\\
C9&: \ 'F' = type \wedge \neg x_1 \wedge RET = 0 \wedge RET = RET2 \wedge RET2 \geq 0,\\
C10&: \ 'F' = type \wedge x_1 \wedge \neg x_2 \wedge RET = y_1 \wedge RET = RET2 \wedge RET2 \geq 0,\\
C11&: \ 'F' \neq type \wedge \neg w_1 \wedge RET = 0 \wedge RET = RET2 \wedge RET2 < 0,\\
C12&: \ 'F' \neq type \wedge w_1 \wedge \neg w_2 \wedge RET = z_1 \wedge RET = RET2 \wedge RET2 < 0,\\
C13&: \ 'F' \neq type \wedge \neg w_1 \wedge RET = 0 \wedge RET = RET2 \wedge RET2 \geq 0,\\
C14&: \ 'F' \neq type \wedge w_1 \wedge \neg w_2 \wedge RET = z_1 \wedge RET = RET2 \wedge RET2 \geq 0.
\end{aligned}
$$

Technical Challenges

- How to encode a program to constraints
  - Arrays, loops, heap, strings
- How to solve constraints
  - Propositional logic and SAT/SMT solving