
Lazy Symbolic Evaluation and it's Path Constraints Solution

LIN Meng-xiang, CHEN Yin-li, YU Kai,
WU Guo-shi

State Key Laboratory of Software Development Environment
Beihang University

AST 2009

Contents

- **Lazy symbolic evaluation.**
- **A method to solve path constraints generated by lazy symbolic evaluation.**

Background

- **Symbolic evaluation**

Evaluate programs using symbolic values instead of concrete values.

- **Test data generation based on symbolic evaluation**

- > Select paths according to path exploration strategies;
- > Generate path constraints by symbolic evaluation;
- > Solve path constraints.

Problems

```
1 . int a[3],i,j,x;
2 . scanf(“%d&d”,&i,&j);
3 . a[j]=2;
4 . a[i]=0;
5 . a[j]= a[j]+1;
6 . if(a[j]!=3)
7 .     x=a[j];
   else
8 .     x= a[i];
```

```
int obscure(int x,int y)
{ int z;
  1. z= encrypt(y);
  2. if (x ==z)
  3. return -1;
   return 0;
}
```

Observation

- Some program structures in programming languages can be reasoned symbolically, whereas the others have to be evaluated concretely.
- Traditional symbolic evaluation can only cope with the former.

Lazy Evaluation

- **Eager evaluation**

Evaluate an expression as soon as it gets bound to a variable.

- **Lazy evaluation**

Delay an evaluation of an expression until such time as the outcome of the computation is known to be needed.

Lazy Symbolic Evaluation

- Combine a form of laziness with eager evaluation used in traditional symbolic evaluation.
- Handle the structures that need to be evaluated dynamically.

Examples

Path	Symbolic memory	Path Constraint
<pre>int *p,A[10]; 1. p=A; 2. if (j<3) 3. p=p+j; 4. if (*p==12345678)</pre>	<pre>A→A₀;j→j₀ p →(A₀,0) p →(A, j₀)</pre>	<pre>j₀<3 *p==12345678</pre>

$$PC = (j_0 < 3) \wedge (*p == 12345678)$$

Examples (cont.)

Path	Symbolic memory	Path Constraint
<pre>int z; 1. z= encrypt(y); 2. if (x ==z) 3. return -1;</pre>	<pre>x→x₀;y→y₀ z→ ⊥</pre>	<pre>x₀ == z</pre>

PC= (x₀ == z)

Semantics on SIL

- **A simple imperative language(SIL)**

The constructs of SIL consist of constants, variables, expressions and statements.

Statements is defined as follows :

$$c ::= \text{skip} \mid x \leftarrow a \mid c_0; c_1 \mid \text{if } b \text{ then } c_0 \text{ else } c_1 \mid \\ \text{while } b \text{ do } c \mid$$

Semantics on SIL(cont.)

- **Definition 1.**

Suppose that e and v are expressions, define $e[v/x]$ to be a symbolic substitution for e , if all occurrences of variable x in e are replaced by v .

Semantics on SIL(cont.)

Expressions:

$$\langle e, \sigma \rangle \rightarrow (e[val(x_1)/x_1]_l, \dots, e[val(x_i)/x_i]_l, \dots, e[val(x_n)/x_n]_l) \\ \forall x_i \in use(e), 1 \leq i \leq n$$

Assignment:

$$\langle x := a, \sigma \rangle \rightarrow \left\{ \begin{array}{l} val(x) = \perp \\ \quad \text{if } loc(x) \text{ is symbolic;} \\ val(x) = \perp \\ \quad \text{if } \exists x_i, val(x_i) \text{ is symbolic or undetermined,} \\ \quad \quad x_i \in use(a); \\ val(x) = (a[val(x_1)/x_1], \dots, a[val(x_i)/x_i], \\ \quad \quad \quad \dots, a[val(x_n)/x_n]), \\ \quad \quad \quad \forall x_i \in use(a), 1 \leq i \leq n \\ \text{otherwise.} \end{array} \right.$$

Semantics on SIL(cont.)

Sequence command:

$$\langle s_1; s_2, \sigma \rangle \rightarrow \langle s_2, \langle s_1, \sigma \rangle \rangle$$

Conditional command:

$$\langle \text{if } e \text{ then } s_1 \text{ else } s_2, \sigma \rangle \rightarrow \begin{cases} \langle s_1, \sigma \rangle & \text{if } \langle e, \sigma \rangle \text{ is true;} \\ \langle s_2, \sigma \rangle & \text{if } \langle e, \sigma \rangle \text{ is false.} \end{cases}$$

Function:

$$\langle x := \text{func}(a_1, \dots, a_n), \sigma \rangle \rightarrow \begin{cases} \text{val}(x) = \perp & \text{if there is no source code of } \text{func}(); \\ \text{val}(x) = \langle \text{func}(a_1, \dots, a_n), \sigma \rangle & \text{otherwise.} \end{cases}$$

Features of Path Constraints

- In tradition symbolic evaluation, path constraints are in terms of input variables.
- In lazy symbolic evaluation, path constraints are in terms of input variables and intermediate variables.

For example,

$$PC = (j_0 < 3) \wedge (*p == 12345678)$$

$$PC = (x_0 == z)$$

The basic idea

```
int *p;
```

```
1. p=A;
```

```
2. if (j<3)
```

```
3. p=p+j;
```

```
4. if (*p==12345678)
```

```
PC= (j0<3)^(*p==12345678)
```

- Divide PC into two parts $\{(j_0 < 3), (*p = 12345678)\}$.
- Solve the constrain $(j_0 < 3)$ using a constraint solver and get some solutions, e.g. $j_0 = 1$.
- Execute the path using $j_0 = 1$ and the resulting PC' is $(1 < 3) \wedge (A[1] = 12345678)$.
- Solve PC' and get the input $\{j = 1, A[1] = 12345678\}$ for the path.

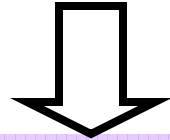
Path Constraint Solving

- Primitive constraints: each relational expression in path constraints is referred to as a primitive constraint.
- Input-constraints: primitive constraints contain only input variables
- Inter-constraints: primitive constraints contain input and intermediate variables .

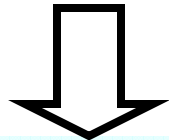
$$PC = (j_0 < 3) \wedge (*p = 12345678)$$

Path Constraint Solving

Transforming PC



Solving $\langle C, PC_b \rangle$



Solving a satisfied formula A

Transforming PC

e.g. $PC = (j_0 < 3) \wedge (*p = 12345678)$

the set of primitive constraints

$$C = \{((j_0 < 3)), ((*p = 12345678))\}$$

the Boolean path constraint

$$PC_b = b_1 \wedge b_2$$

where: $b_1 \leftrightarrow (j_0 < 3)$, $b_2 \leftrightarrow (*p = 12345678)$

$$T(PC) = \langle C, PC_b \rangle$$

Solving $\langle C, PC_b \rangle$

- **A solution to PC_b**

Suppose that PC_b is satisfiable and one of the solutions is $\langle d_{j_1}, \dots, d_{j_j}, \dots, d_{j_n} \rangle$, $1 \leq j \leq m$, $1 \leq i \leq n$, d_{ji} is the truth value of a primitive constraint c_i .

Solving $\langle C, PC_b \rangle$ (cont.)

- **A satisfied formula A**

For each solution to PC_b , a satisfied formula is:

$$A_j = a_{j_1} \wedge \dots \wedge a_{jj} \wedge \dots \wedge a_{j_n} \quad (1 \leq j \leq m)$$

$$a_{jj} = \begin{cases} c_{ij} & , d_{ji} = 1 \\ \neg c_{ij} & , d_{ji} = 0 \end{cases}$$

$$PC = \bigvee A_j$$

Solving a satisfied formula A

- For a satisfied formula

$$A = a_1 \wedge a_2 \wedge \dots \wedge a_n$$

Divide it into two parts: $A = A^I \wedge A^M$.

- An input-formula A^I is a constraint system in terms of input variables.
- An inter-formula A^M is a constraint system in terms of input and intermediate variables.

Discussion

- $A^M = \emptyset$
 - The path constraint is a constraint system on input variables.
 - Pure static method was designed to solve it.
- $A^I = \emptyset$
 - The path constraint is a constraint system in terms of intermediate variables.
 - Dynamic method works well.
- $A^I \neq \emptyset \wedge A^M \neq \emptyset$
 - The path constraint is a hybrid constraint system containing input and intermediate variables.
 - An approach combining static method and dynamic method is developed to address the issue.

Summary

- A light-weight symbolic evaluation that combines lazy evaluation with eager evaluation.
- Path constraints derived by lazy symbolic evaluation is different from the traditional.
- An approach that solves path constraints generated by lazy symbolic evaluation.

Future Work

- Improve the efficiency of path constraints solving by integrate more search strategies.
- More experiments will be investigated to evaluate the performance for larger programs with proper path exploration strategies.
- Eliminate infeasible paths as early as possible.

Thank You!