软件分析

符号执行

熊英飞
北京大学

提醒

- 代码提交（11月12日）
  - Readme.pdf：A4两页以内，描述算法的主要设计思想，小组成员姓名、学号和分工
  - Code目录：项目源代码
  - analyzer.jar: 编译好的jar文件
- 现场报告（11月15日）
  - 各组交流所采用的算法，预计每组5分钟左右

符号执行

- 抽象解释：每次分析完整程序，但在一个抽象域上进行
- 符号执行：每次分析一条路径，按某种顺序遍历路径

- 首先我们假设程序中
  - 没有数组
  - 没有指针
  - 没有函数调用
  - 没有系统调用

符号执行

```
int main(x,y) {
  y+=10;
  if (x>0) {
    x+=10;
    z=x/5;
  }
  else {
    z=x/5+2;
    x+=10;
  }
  z+=y;
  return z;
}
```

程序是否满足如下规约:

前条件： y>0

后条件： main(x,y) > 0

即y>0 -> main(x,y) > 0

符号执行

```
1.   int main(x,y) {
2.     y+=10;
3.     if (x>0) {
4.       x+=10;
5.       z=x/5;
6.     }
7.     else {
8.       z=x/5+2;
9.       x+=10;
10.    }
11.    z+=y;
12.    return z;
13.  }
```

待探状态

x=a
y=b
z=0
Next: 2
Cond: b>0

符号执行

```
1.   int main(x,y) {
2.     y+=10;
3.     if (x>0) {
4.       x+=10;
5.       z=x/5;
6.     }
7.     else {
8.       z=x/5+2;
9.       x+=10;
10.    }
11.    z+=y;
12.    return z;
13.  }
```

当前状态

x=a
y=b
z=0
Next: 2
Cond: b>0

待探状态

x=a
y=b+10
z=0
Next: 3
Cond: b>0

符号执行

```
1.    int main(x,y) {
2.      y+=10;
3.      if (x>0) {
4.        x+=10;
5.        z=x/5;
6.      }
7.      else {
8.        z=x/5+2;
9.        x+=10;
10.     }
11.     z+=y;
12.     return z;
13.   }
```

当前状态

x=a
y=b+10
z=0
Next: 3
Cond: b>0

待探状态

x=a
y=b+10
z=0
Next: 4
Cond: $a>0 \wedge b>0$

x=a
y=b+10
z=0
Next: 8
Cond: $a \leq 0 \wedge b>0$

符号执行

```
1.    int main(x,y) {
2.      y+=10;
3.      if (x>0) {
4.        x+=10;
5.        z=x/5;
6.      }
7.      else {
8.        z=x/5+2;
9.        x+=10;
10.     }
11.     z+=y;
12.     return z;
13.   }
```

当前状态

x=a
y=b+10
z=0
Next: 4
Cond: $a > 0 \wedge b > 0$

待探状态

x=a+10
y=b+10
z=0
Next: 5
Cond: $a > 0 \wedge b > 0$

x=a
y=b+10
z=0
Next: 8
Cond: $a \leq 0 \wedge b > 0$

符号执行

```
1.  int main(x,y) {
2.    y+=10;
3.    if (x>0) {
4.      x+=10;
5.      z=x/5;
6.    }
7.    else {
8.      z=x/5+2;
9.      x+=10;
10.   }
11.   z+=y;
12.   return z;
13. }
```

当前状态

x=a+10
y=b+10
z=0
Next: 5
Cond: $a>0 \wedge b>0$

待探状态

x=a+10
y=b+10
z=(a+10)/5
Next: 11
Cond: $a>0 \wedge b>0$

x=a
y=b+10
z=0
Next: 8
Cond: $a \leq 0 \wedge b>0$

符号执行

```
1.  int main(x,y) {
2.    y+=10;
3.    if (x>0) {
4.      x+=10;
5.      z=x/5;
6.    }
7.    else {
8.      z=x/5+2;
9.      x+=10;
10.   }
11.   z+=y;
12.   return z;
13. }
```

当前状态

x=a+10
y=b+10
z=(a+10)/5
Next: 11
Cond: $a > 0 \wedge b > 0$

待探状态

x=a+10
y=b+10
z=(a+10)/5+(b+10)
Next: 12
Cond: $a > 0 \wedge b > 0$

x=a
y=b+10
z=0
Next: 8
Cond: $a \leq 0 \wedge b > 0$

符号执行

```
1.    int main(x,y) {
2.      y+=10;
3.      if (x>0) {
4.        x+=10;
5.        z=x/5;
6.      }
7.      else {
8.        z=x/5+2;
9.        x+=10;
10.     }
11.     z+=y;
12.     return z;
13.   }
```

当前状态

x=a+10
y=b+10
z=(a+10)/5+(b+10)
Next: 12
Cond: $a>0 \land b>0$

- 求$a>0 \land b>0 \rightarrow$ (a+10)/5+(b+10) > 0是否恒成立
- 取反后判可满足性
- 如果满足，则程序不满足规约

待探状态

x=a
y=b+10
z=0
Next: 8
Cond: $a \leq 0 \land b>0$

符号执行

```
1.   int main(x,y) {
2.     y+=10;
3.     if (x>0) {
4.       x+=10;
5.       z=x/5;
6.     }
7.     else {
8.       z=x/5+2;
9.       x+=10;
10.    }
11.    z+=y;
12.    return z;
13.  }
```

当前状态

x=a
y=b+10
z=0
Next: 8
Cond: $a \le 0 \wedge b > 0$

待探状态

x=a+10
y=b+10
z=a/5+2+(b+10)
Next: 12
Cond: $a \le 0 \wedge b > 0$

符号执行

```
1.    int main(x,y) {
2.      y+=10;
3.      if (x>0) {
4.        x+=10;
5.        z=x/5;
6.      }
7.      else {
8.        z=x/5+2;
9.        x+=10;
10.     }
11.     z+=y;
12.     return z;
13.   }
```

当前状态

x=a+10

y=b+10

z=a/5+2+(b+10)

Next: 12

Cond: $a \leq 0 \wedge b > 0$

- 求$a \leq 0 \wedge b > 0 \rightarrow$ a/5+2+(b+10)>0是否恒成立
- 取反后判可满足性
- 如果满足，则程序不满足规约

待探状态

符号执行小结

- 状态:
  - 当前变量的符号值
  - 下一条待执行的语句
  - 当前状态的路径约束
- 不断遍历状态，遇上结束状态时用SMT求解器判断规约是否被满足
- 程序不满足规约=任意路径不满足规约
- 程序满足规约=所有路径满足规约
  - 路径无穷多时做不到
  - 通常遍历有限次循环来模拟

符号执行检查程序内部错误

- 用符号执行发现缓冲区溢出
  - ……
  - a[i] = 4;
  - 判断后条件0<=i&&i<a.length是否总是成立
- 用符号执行发现除0错误
  - ……
  - x = 3 / i;
  - 判断后条件i!=0是否总是成立
- 用符号执行发现路径可行性
  - 判断给定路径上的路径约束是否可满足

调用约束求解的时机

```
1.  if (x>0) {
2.      y = ...
3.    }
4.  if (x<=0) {
5.      y = ...
6.  }
```

- 无需对不可达路径继续探索或验证
- Eager evaluation：在分支的时候就判断路径的可达性
- Lazy evaluation：只对完整路径判断
  - 和之前的方法等价
- 在不同程序中两种方法各有优劣
  - eager evaluation对同一条路径可能调用更多次，但探索的路径总数会减少
  - 如果路径不可达，lazy evaluation的约束不会更简单，但有可能更容易产生冲突，导致解得更快

从冲突中学习

```
1.  if (x>0) {
2.      y = …
3.    }
4.  if (z > 0) {
5.    …
6.  } else {
7.    …
8.  }
9.  if (x<=0) {
10.     y = …
11. }
```

- 第1行的条件和第9行的条件互斥，但有多条路径都包括这两个条件
- 假设先选择路径1-2-4-5-9-10，则生成路径约束
  - 1: $x > 0$
  - 4: $z > 0$
  - 9: $x \leq 0$
- SMT判断不能满足，返回矛盾集(1, 9)
- 假设又选择路径1-2-4-7-9-10，则路径约束包含(1,9)，不用调用SMT约束求解直接可知不可满足

处理数组

```
1.  int main(int[] array, int index) {
2.     array[index]=1;
3.     assert(array[0]==1);
4.  }
```

通过SMT的Array Theory处理

array=a
index=b
Next: 2

array=write(a, b, 1)
index=b
Next: 3

判断True->read(write(a, b, 1), 0)=1是否恒成立

处理函数调用

```
1.  int power(int x, int n) {
2.    if (n > 0) {
3.      int ret=power(x, n-1)*x;
4.      return ret;  }
5.    else return 1; }
```

x=a
n=b
Next: 2

x=a
n=b
Next: 3
Cond: b>0

x=a
n=b
Next: 5
Cond: ¬b>0

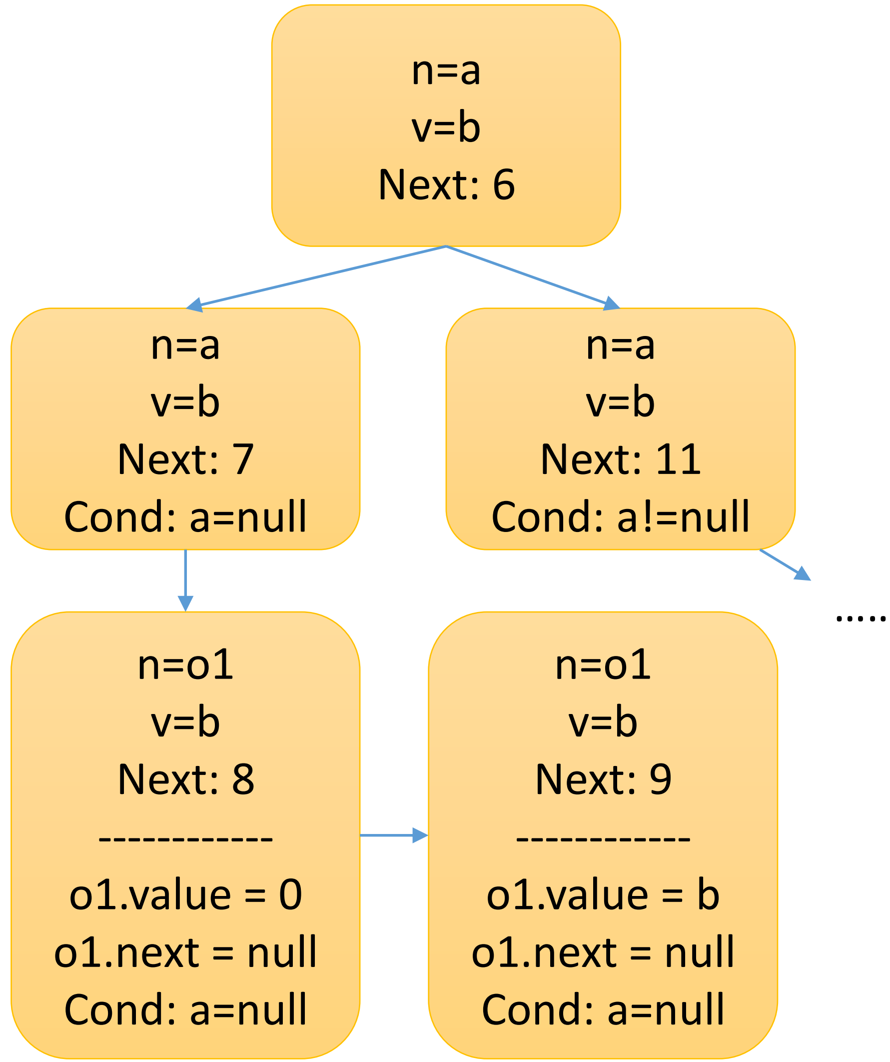
x=a
n=b-1
Next: 2
----
x=a
n=b
Next: 3
Cond: b>0

x=a
n=b-1
Next: 5
----
x=a
n=b
Next: 3
Cond: b>0 ∧ ¬b-1>0

x=a
n=b
ret=1*a
Next: 4
Cond: b>0 ∧ ¬b-1>0

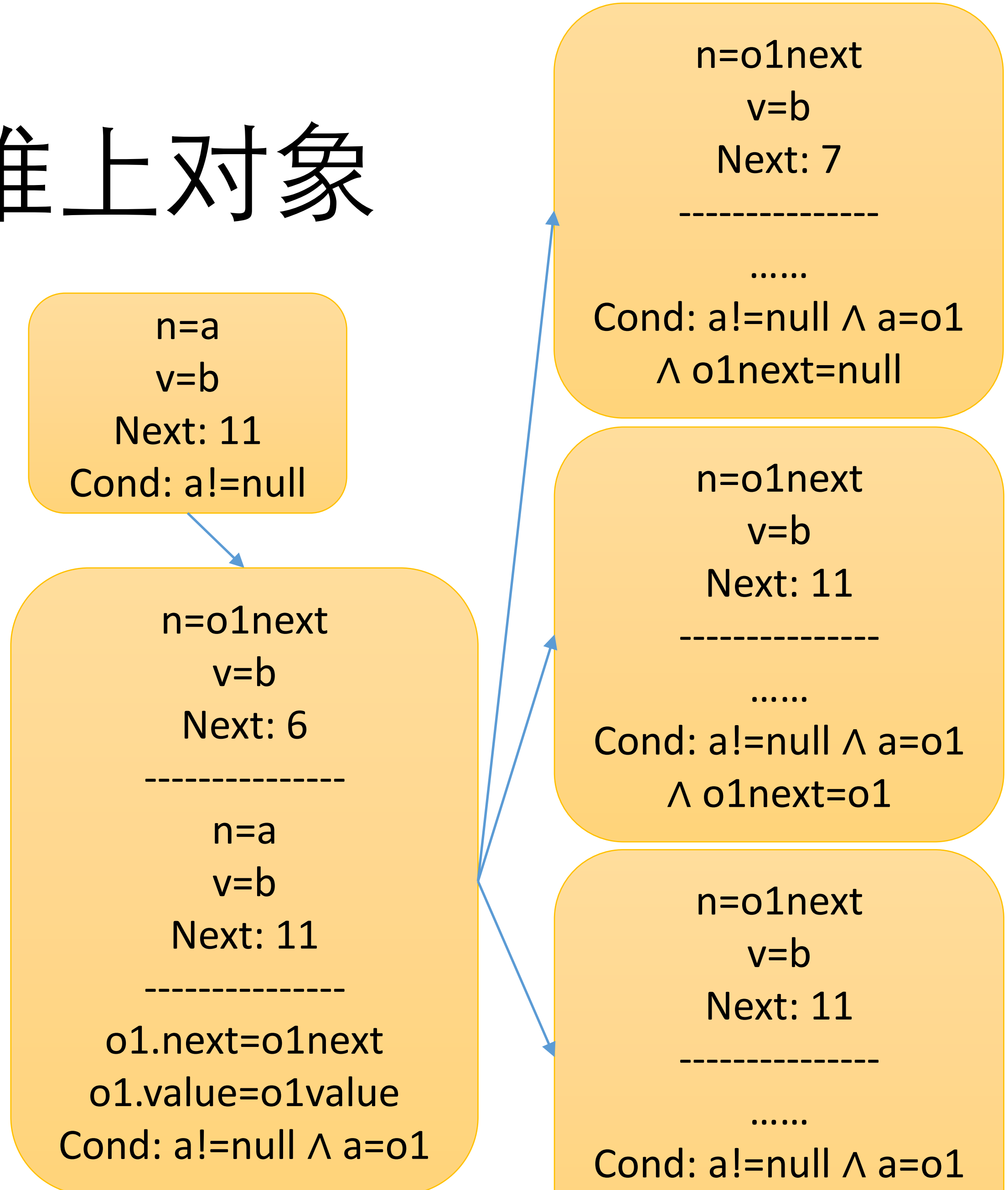
处理指针和堆上对象

```
1.   Class Node {
2.     int value;
3.     Node next;
4.   }
5.   int append(Node n, int v) {
6.     if(n == null) {
7.       n = new Node();
8.       n.value = v;
9.       return n;
10.    }
11.    n.next = append(n.next, v);
12.    return n;
13.  }
```



处理指针和堆上对象

```
1.   Class Node {
2.     int value;
3.     Node next;
4.   }
5.   int append(Node n, int v) {
6.     if(n == null) {
7.       n = new Node();
8.       n.value = v;
9.       return n;
10.    }
11.    n.next = append(n.next, v);
12.    return n;
13.  }
```



引入新符号o1next和o1value

约束求解失败的情况

- 形成了复杂条件
  - $x^5 + 3x^3 == y$
  - p->next->value == x
- 调用了系统调用
  - If (file.read()==x)

- 动态符号执行
  - 混合程序的真实执行和符号执行
  - 在约束求解无法进行的时候，用真实值代替符号值
    - 如果真实值x=10，则$x^5 + 3x^3 == y$变为103000==y，可满足

动态符号执行-例子

```
1.  int foo (int v) {
2.    return (v*v) % 50; }
3.  void testme (int x, int y) {
4.    z = foo (y);
5.    if (x > y+10) {
6.      if (z == x)
7.        assert(false);
8.    }}
```

x=a
y=b
z=b*b % 50
Next: 7
Cond: $a > b + 10 \land b*b \% 50 == a$

求解约束:
$a > b + 10 \land b*b \% 50 = a \rightarrow$ false
会返回Unknown

动态符号执行-例子

```
1.  int foo (int v) {
2.    return (v*v) % 50; }
3.  void testme (int x, int y) {
4.    z = foo (y);
5.    if (x > y+10) {
6.      if (z == x)
7.        assert(false);
8.    }}
```

x=1, a
y=5, b
z=0, 0
Next: 4
Cond:

x=1, a
y=5, b
z=25, (b*b) % 50
Next: 5
Cond:

x=1, a
y=5, b
z=25, (b*b) % 50
Next: --
Cond: $\neg(a>b+10)$

保存条件的列表

动态符号执行-例子

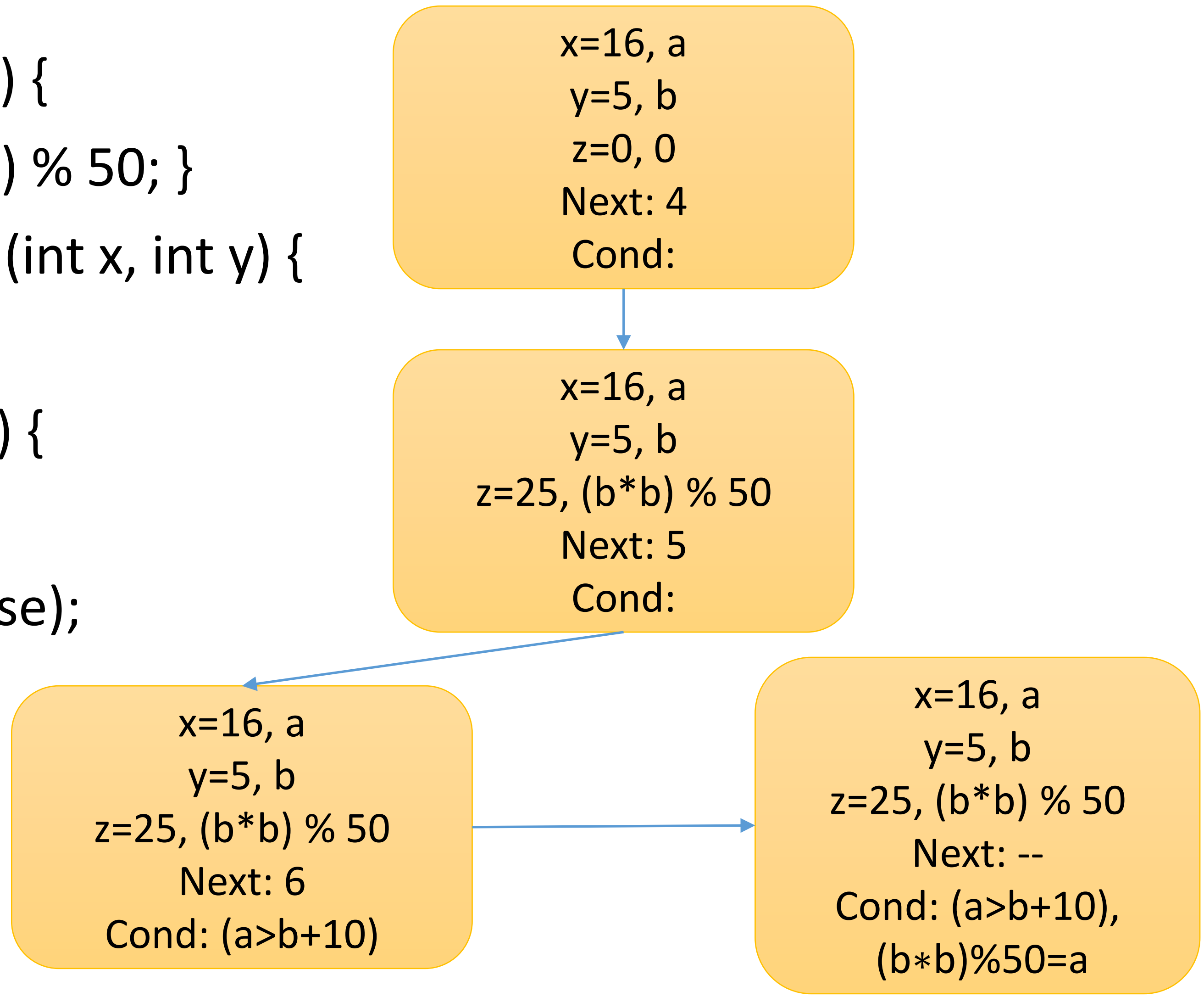
```
1.  int foo (int v) {
2.    return (v*v) % 50; }
3.  void testme (int x, int y) {
4.    z = foo (y);
5.    if (x > y+10) {
6.      if (z == x)
7.        assert(false);
8.    }}
```

x=1, a
y=5, b
z=25, (b*b) % 50
Next: --
Cond: ¬(a>b+10)

- 从列表中取最后一个没有被取反过的条件取反
  - a>b+10
- 求解，得到a=16, b=5

动态符号执行-例子

```
1.  int foo (int v) {
2.    return (v*v) % 50; }
3.  void testme (int x, int y) {
4.    z = foo (y);
5.    if (x > y+10) {
6.      if (z == x)
7.        assert(false);
8.    }}
```



动态符号执行-例子

```
int foo (int v) {
  return (v*v) % 50; }
void testme (int x, int y) {
  z = foo (y);
  if (x > y+10) {
    if (z == x)
      assert(false);
  }}
```

x=16, a
y=5, b
z=25, (b*b) % 50
Next: --
Cond: (a>b+10),
(b*b)%50=a

- 从列表中取最后一个没有被取反过的条件取反
  - $(a>b+10) \wedge (b*b)\%50=a$
- 求解，发现%不被SMT支持
- 将%运算涉及的变量替换成具体值
  - $(a>5+10) \wedge (5*5)\%50=a$
- 得到a=25

动态符号执行-例子

```
1. int foo (int v) {
2.   return (v*v) % 50; }
3. void testme (int x, int y) {
4.   z = foo (y);
5.   if (x > y+10) {
6.     if (z == x)
7.       assert(false);
8.   }}
```

x=25, a
y=5, b
z=0, 0
Next: 4
Cond:

↓

x=25, a
y=5, b
z=25, (b*b) % 50
Next: 7
Cond: (a>b+10)
∧ (b∗b)%50=a

具体执行直接触发AssertionError

动态符号执行小结

- 替换变量为具体值的方法不保证完整性
  - 可满足的约束可能变得不可满足
  - $(a>b+10) \wedge (b*b)\%50=a$中，如果b=0，则不可满足
- 但效果一定优于静态符号执行
  - 替换只在原约束无法求解的情况下进行
- 为什么将具体值代入执行而不是在约束无法求解的时候替换为随机值?
  - 因为有可能拿不到完整的路径约束
  - 比如路径约束有可能是foo(x)==0，但foo的实现代码拿不到，foo也不能脱离系统状态单独调用

常见符号执行工具

- C语言： KLEE
- Java语言： SymbolicPathFinder, JBSE
- C#语言： Pex
  - 北京大学谢涛老师和微软合作开发

基于霍尔逻辑的符号执行

霍尔逻辑

- 霍尔三元组
  - {前条件}语句{后条件}
- 霍尔逻辑表示三者之间的推导关系
- 又称为公理语义

While语言

Statement ::=

| skip

| while (Expr) Statement

| if (Expr) Statement else Statement

| Statement; Statement

| Var = Expr

霍尔逻辑规则

$$\textsc{Skip}\ \frac{}{\{P\}\ \textbf{skip}\ \{P\}}$$

$$\textsc{Assign}\ \frac{}{\{P[a/x]\}\ x := a\ \{P\}}$$

$$\textsc{Seq}\ \frac{\{P\}\ c_1\ \{R\} \qquad \{R\}\ c_2\ \{Q\}}{\{P\}\ c_1; c_2\ \{Q\}}$$

$$\textsc{If}\ \frac{\{P \wedge b\}\ c_1\ \{Q\} \qquad \{P \wedge \neg b\}\ c_2\ \{Q\}}{\{P\}\ \textbf{if}\ b\ \textbf{then}\ c_1\ \textbf{else}\ c_2\ \{Q\}}$$

$$\textsc{Consequence}\ \frac{\vDash (P \Rightarrow P') \qquad \{P'\}\ c\ \{Q'\} \qquad \vDash (Q' \Rightarrow Q)}{\{P\}\ c\ \{Q\}}$$

$$\textsc{While}\ \frac{\{P \wedge b\}\ c\ \{P\}}{\{P\}\ \textbf{while}\ b\ \textbf{do}\ c\ \{P \wedge \neg b\}}$$

用霍尔逻辑证明举例

- if (x > 0) x += 10; else x = 20;
  - 该程序执行结束后，x是否一定大于0?
- 根据Assign，可得
  - {x+10>0} x+=10 {x > 0}
  - {True} x=20 {x >0}
- 因为x>0 => x+10 > 0且¬x>0 => True，根据Consequence，可得
  - {x>0} x+=10 {x > 0}
  - {¬x>0} x=20 {x > 0}
- 根据If，可得
  - {True} if (x > 0) x += 10; else x = 20;{x>0}

用霍尔逻辑证明练习

- while (x < 10) x += 1;
  - 该程序执行结束后，x是否一定大于0?
- 根据Assign，可得
  - {True} x+=1 {True}
- 根据Consequence，可得
  - {x<10/\True} x+=10 {True}
- 根据While，可得
  - {True} while (x < 10) x += 1; {x>=10}
- 根据Consequence，可得
  - {True} while (x < 10) x += 1; {x>0}

谓词转换计算

- 最弱前条件计算：给定后条件和语句，求能形成霍尔三元组的最弱前条件
- 最强后条件计算：给定前条件和语句，求能形成霍尔三元组的最强后条件

- 基于谓词转换的符号执行
  - 给定输入需要满足的条件P，代码c，输出需要满足的条件Q
  - 前向符号执行：基于P和c计算最强后条件Q’，验证Q’->Q是否恒成立
  - 后向符号执行：基于Q和c计算最弱前条件P’，验证P->P’是否恒成立

最弱前条件计算

- $wp(skip, Q) = Q$

$$\text{SKIP}\ \frac{}{\{P\}\ \textbf{skip}\ \{P\}}$$

- $wp(x \coloneqq a, Q) = Q[a/x]$

$$\text{ASSIGN}\ \frac{}{\{P[a/x]\}\ x := a\ \{P\}}$$

- $wp(c_1; c_2, Q) = wp\big(c_1, wp(c_2, Q)\big)$

$$\text{SEQ}\ \frac{\{P\}\ c_1\ \{R\} \qquad \{R\}\ c_2\ \{Q\}}{\{P\}\ c_1; c_2\ \{Q\}}$$

- $wp(if\ b\ then\ c_1\ else\ c_2, Q) = \big(b \to wp(c_1, Q)\big) \land \big(\neg b \to wp(c_2, Q)\big)$

$$\text{IF}\ \frac{\{P \land b\}\ c_1\ \{Q\} \qquad \{P \land \neg b\}\ c_2\ \{Q\}}{\{P\}\ \textbf{if}\ b\ \textbf{then}\ c_1\ \textbf{else}\ c_2\ \{Q\}}$$

最弱前条件：举例

- wp(if (x > 0) x += 10; else x = 20, x>0)
  - =(x>0->wp(x+=10, x>0)) /\ (x<=0 -> wp(x=20, x>0))
  - =(x>0->x+10>0) /\ (x<=0 -> 20>0)
  - =True

最弱前条件计算：循环

- $wp(while\ b\ do\ c, Q) = \exists i \in Nat. L_i(Q)$
  - where
    - $L_0(Q) = false$
    - $L_{i+1}(Q) = (\neg b \Rightarrow Q) \wedge \left(b \Rightarrow wp\left(c, L_i(Q)\right)\right)$
- i代表循环最多执行了$i-1$次
- 注意这个最弱前条件蕴含了循环必然终止

$$\text{WHILE}\ \frac{\{P \wedge b\}\ c\ \{P\}}{\{P\}\ \textbf{while}\ b\ \textbf{do}\ c\ \{P \wedge \neg b\}}$$

最强后条件计算

- $sp(P, skip) = P$
- $sp(P, x \coloneqq a) = \exists n.\ x = a[n/x] \wedge P[n/x]$
- $sp(P, c_1; c_2) = sp(sp(P, c_1), c_2)$
- $sp(P, if\ b\ then\ c_1\ else\ c_2) = sp(b \wedge P, c_1) \vee sp(\neg b \wedge P, c_2)$
- $sp(P, while\ b\ do\ c) = \neg b \wedge \exists i. L_i(P)$
  - where
    - $L_0(P) = P$
    - $L_{i+1}(P) = sp(b \wedge L_i(P), c)$

因为约束更复杂，实际使用较少

符号执行和谓词转换

- 在没有循环的情况下，最弱前条件和符号执行等价
- 例：If (y > 0) x+=1; else x+=2; assert(x<3)
- 符号执行
  - 令x=a, y=b, 计算得到
    - $(b > 0 \land a + 1 < 3) \lor (\neg b > 0 \land a + 2 < 3)$
- 最弱前条件
  - $wp(x \mathrel{+}= 1, x < 3) = x + 1 < 3$
  - $wp(x \mathrel{+}= 2, x < 3) = x + 2 < 3$
  - $wp(\text{原程序}, x < 3) = (y > 0 \rightarrow x + 1 < 3) \land (\neg y > 0 \rightarrow x + 2 < 3) = (y > 0 \land x + 1 < 3) \lor (\neg y > 0 \land x + 2 < 3)$

参考资料

- 《程序设计语言的形式语义》Glynn Winskel著
- Baldoni R , Coppa E , D’elia, Daniele Cono, et al. A Survey of Symbolic Execution Techniques[J]. Acm Computing Surveys, 2016.