Software Systems Verification and Validation

Lecture 05 - Symbolic execution

Lect. dr. Andreea Vescan

Babeş-Bolyai University
Cluj-Napoca

2014-2015

1. Symbolic execution
   - Old research area but still active...
   - What is symbolic Execution?
2. Conventional vs Symbolic execution
   - Conventional Execution
   - Symbolic Execution
   - Commutative property
3. Conditional branching
   - If statement
   - If example execution
   - While statement
   - While example execution
4. Symbolic Execution Tree
   - Symbolic Execution Tree
   - Symbolic Execution Tree for Sum

Testing

- 1975 - First introduced
- King [Comm. ACM 1976], Clarke [IEEE TSE 1976]

Testing

- 1975 - First introduced
- King [Comm. ACM 1976], Clarke [IEEE TSE 1976]
- 2005 - DART (Bell Labs), EGT (Stanford), CUTE (UIUC)

Testing

- 1975 - First introduced
- King [Comm. ACM 1976], Clarke [IEEE TSE 1976]
- 2005 - DART (Bell Labs), EGT (Stanford), CUTE (UIUC)
- 2006 - EXE (Stanford)

Testing

- 1975 - First introduced
- King [Comm. ACM 1976], Clarke [IEEE TSE 1976]
- 2005 - DART (Bell Labs), EGT (Stanford), CUTE (UIUC)
- 2006 - EXE (Stanford)
- 2007- CatchConv (Berkeley)

Testing

- 1975 - First introduced
- King [Comm. ACM 1976], Clarke [IEEE TSE 1976]
- 2005 - DART (Bell Labs), EGT (Stanford), CUTE (UIUC)
- 2006 - EXE (Stanford)
- 2007- CatchConv (Berkeley)
- 2008 - KLEE (Stanford)

Testing

- 1975 - First introduced
- King [Comm. ACM 1976], Clarke [IEEE TSE 1976]
- 2005 - DART (Bell Labs), EGT (Stanford), CUTE (UIUC)
- 2006 - EXE (Stanford)
- 2007- CatchConv (Berkeley)
- 2008 - KLEE (Stanford)
- 1999 - 2013 - JPF (NASA Ames)

What is symbolic execution ?

- Symbolic execution = Execution of program with symbols as argument. Instead of supplying the normal inputs to a program, symbolic execution supplies symbols representing arbitrary values.

What is symbolic execution ?

- Symbolic execution = Execution of program with symbols as argument. Instead of supplying the normal inputs to a program, symbolic execution supplies symbols representing arbitrary values.
  - int FunctionName(1, 2) ⇒ int FunctionName(a1 , a2)

What is symbolic execution ?

- Symbolic execution = Execution of program with symbols as argument. Instead of supplying the normal inputs to a program, symbolic execution supplies symbols representing arbitrary values.
  - int FunctionName(1, 2) ⇒ int FunctionName(a1 , a2)
- The execution proceeds as in a normal execution except that values may be symbolic formulae over the input symbols.

What is symbolic execution ?

- Symbolic execution = Execution of program with symbols as argument. Instead of supplying the normal inputs to a program, symbolic execution supplies symbols representing arbitrary values.
  - int FunctionName(1, 2) ⇒ int FunctionName(a1 , a2)
- The execution proceeds as in a normal execution except that values may be symbolic formulae over the input symbols.
- During symbolic execution, program state consists of:

What is symbolic execution ?

- Symbolic execution = Execution of program with symbols as argument. Instead of supplying the normal inputs to a program, symbolic execution supplies symbols representing arbitrary values.
    - int FunctionName(1, 2) ⇒ int FunctionName(a1 , a2)
- The execution proceeds as in a normal execution except that values may be symbolic formulae over the input symbols.
- During symbolic execution, program state consists of:
    - Symbolic values/expressions for variables;

What is symbolic execution ?

- Symbolic execution = Execution of program with symbols as argument. Instead of supplying the normal inputs to a program, symbolic execution supplies symbols representing arbitrary values.
  - int FunctionName(1, 2) ⇒ int FunctionName(a1 , a2)
- The execution proceeds as in a normal execution except that values may be symbolic formulae over the input symbols.
- During symbolic execution, program state consists of:
  - Symbolic values/expressions for variables;
  - Path condition - a conjunct of constraints on the symbolic input values;

What is symbolic execution ?

- Symbolic execution = Execution of program with symbols as argument. Instead of supplying the normal inputs to a program, symbolic execution supplies symbols representing arbitrary values.
  - int FunctionName(1, 2) ⇒ int FunctionName(a1 , a2)
- The execution proceeds as in a normal execution except that values may be symbolic formulae over the input symbols.
- During symbolic execution, program state consists of:
  - Symbolic values/expressions for variables;
  - Path condition - a conjunct of constraints on the symbolic input values;
  - Program counter.

What is symbolic execution ?

- Symbolic execution = Execution of program with symbols as argument. Instead of supplying the normal inputs to a program, symbolic execution supplies symbols representing arbitrary values.
    - int FunctionName(1, 2) ⇒ int FunctionName(a1 , a2)
- The execution proceeds as in a normal execution except that values may be symbolic formulae over the input symbols.
- During symbolic execution, program state consists of:
    - Symbolic values/expressions for variables;
    - Path condition - a conjunct of constraints on the symbolic input values;
    - Program counter.
- Symbolic states represent sets of concrete states.

Conventional execution

- Function Sum

Conventional execution

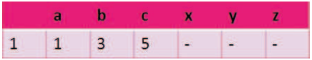
- Function Sum
- 1: int Sum(int a, int b, int c)
- 2: int x := a + b;
- 3: int y := b + c;
- 4: int z := *x* + *y* - *b*;
- 5: return z;
- 6:

Conventional execution

- Function Sum
- 1: int Sum(int a, int b, int c)
- 2: int x := a + b;
- 3: int y := b + c;
- 4: int z := x + y - b;
- 5: return z;
- 6:
- Normal execution result of Sum(1,3,5)

Conventional execution (cont.)

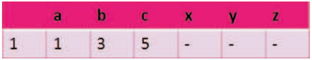
- Function Sum
- Normal execution result of Sum(1,3,5)



	a	b	c	x	y	z
1	1	3	5	-	-	-

Conventional execution (cont.)

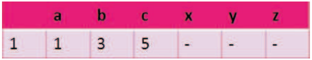
- Function Sum
- Normal execution result of Sum(1,3,5)
- 1 : int Sum(int a, int b, int c)



	a	b	c	x	y	z
1	1	3	5	-	-	-

Conventional execution (cont.)

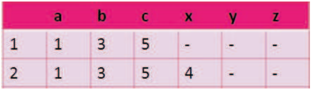
- Function Sum
- Normal execution result of Sum(1,3,5)
- 1 : int Sum(int a, int b, int c)
- 2 : int x := a + b;



	a	b	c	x	y	z
1	1	3	5	-	-	-

Conventional execution (cont.)

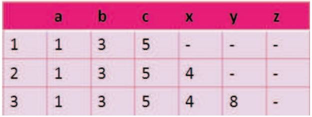
- Function Sum
- Normal execution result of Sum(1,3,5)
- 1 : int Sum(int a, int b, int c)
- 2 : int x := a + b;
- 3: int y := b + c;



	a	b	c	x	y	z
1	1	3	5	-	-	-
2	1	3	5	4	-	-

Conventional execution (cont.)

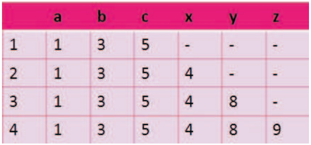
- Function Sum
- Normal execution result of Sum(1,3,5)
- 1 : int Sum(int a, int b, int c)
- 2 : int x := a + b;
- 3: int y := b + c;
- 4: int z := x + y - b;



	a	b	c	x	y	z
1	1	3	5	-	-	-
2	1	3	5	4	-	-
3	1	3	5	4	8	-

Conventional execution (cont.)

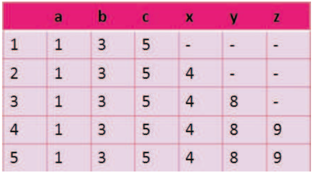
- Function Sum
- Normal execution result of Sum(1,3,5)
- 1 : int Sum(int a, int b, int c)
- 2 : int x := a + b;
- 3: int y := b + c;
- 4: int z := $x + y - b$;
- 5: return z;



	a	b	c	x	y	z
1	1	3	5	-	-	-
2	1	3	5	4	-	-
3	1	3	5	4	8	-
4	1	3	5	4	8	9

Conventional execution (cont.)

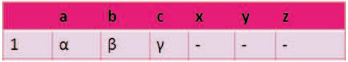
- Function Sum
- Normal execution result of Sum(1,3,5)
- 1 : int Sum(int a, int b, int c)
- 2 : int x := a + b;
- 3: int y := b + c;
- 4: int z := *x* + *y* - *b*;
- 5: return z;
- 6:



	a	b	c	x	y	z
1	1	3	5	-	-	-
2	1	3	5	4	-	-
3	1	3	5	4	8	-
4	1	3	5	4	8	9
5	1	3	5	4	8	9

Symbolic execution

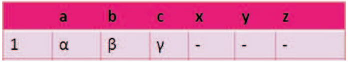
- Function Sum



	a	b	c	x	y	z
1	α	β	γ	-	-	-

Symbolic execution

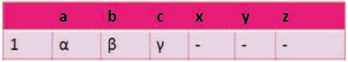
- Function Sum
- Symbolic execution result of Sum(α,β,γ)



	a	b	c	x	y	z
1	α	β	γ	-	-	-

Symbolic execution

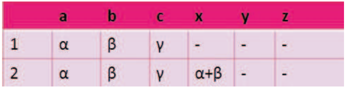
- Function Sum
- Symbolic execution result of Sum(α,β,γ)
- 1 : int Sum(int a, int b, int c)



	a	b	c	x	y	z
1	α	β	γ	-	-	-

Symbolic execution

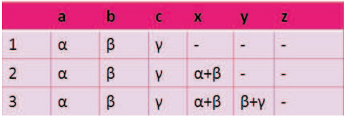
- Function Sum
- Symbolic execution result of Sum(α,β,γ)
- 1 : int Sum(int a, int b, int c)
- 2 : int x := a + b;



	a	b	c	x	y	z
1	α	β	γ	-	-	-
2	α	β	γ	α+β	-	-

Symbolic execution

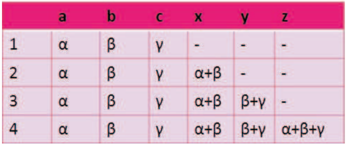
- Function Sum
- Symbolic execution result of Sum(α,β,γ)
- 1 : int Sum(int a, int b, int c)
- 2 : int x := a + b;
- 3: int y := b + c;



	a	b	c	x	y	z
1	α	β	γ	-	-	-
2	α	β	γ	α+β	-	-
3	α	β	γ	α+β	β+γ	-

Symbolic execution

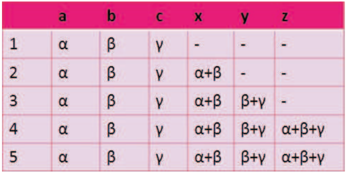
- Function Sum
- Symbolic execution result of Sum(α,β,γ)
- 1 : int Sum(int a, int b, int c)
- 2 : int x := a + b;
- 3: int y := b + c;
- 4: int z := x + y - b;



	a	b	c	x	y	z
1	α	β	γ	-	-	-
2	α	β	γ	α+β	-	-
3	α	β	γ	α+β	β+γ	-
4	α	β	γ	α+β	β+γ	α+β+γ

Symbolic execution

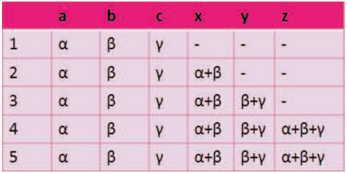
- Function Sum
- Symbolic execution result of Sum(α,β,γ)
- 1 : int Sum(int a, int b, int c)
- 2 : int x := a + b;
- 3: int y := b + c;
- 4: int z := *x* + *y* - *b*;
- 5: return z;



	a	b	c	x	y	z
1	α	β	γ	-	-	-
2	α	β	γ	α+β	-	-
3	α	β	γ	α+β	β+γ	-
4	α	β	γ	α+β	β+γ	α+β+γ
5	α	β	γ	α+β	β+γ	α+β+γ

Symbolic execution

- Function Sum
- Symbolic execution result of Sum(α,β,γ)
- 1 : int Sum(int a, int b, int c)
- 2 : int x := a + b;
- 3: int y := b + c;
- 4: int z := x + y - b;
- 5: return z;
- 6:



	a	b	c	x	y	z
1	α	β	γ	-	-	-
2	α	β	γ	α+β	-	-
3	α	β	γ	α+β	β+γ	-
4	α	β	γ	α+β	β+γ	α+β+γ
5	α	β	γ	α+β	β+γ	α+β+γ

Commutativity

- The same result is obtained using normal execution or using symbolic execution.

$$\begin{tikzcd}
(P(X),K) \arrow[r, Rightarrow, "\text{Symbolic execution}"] \arrow[d, Rightarrow, "\text{Set parameters to integer values}"'] & (E(P(X)),K) \arrow[d, Rightarrow, "\text{Substitute into result of symbolic execution}"] \\
P(K) \arrow[r, Rightarrow, "\text{Conventional execution}"'] & E(P(K))
\end{tikzcd}$$

Commutativity

- The same result is obtained using normal execution or using symbolic execution.
- Normal execution

$$\begin{tikzcd}
(P(X),K) \arrow[r, Rightarrow, "\text{Symbolic execution}"] \arrow[d, Rightarrow, "\text{Set parameters to integer values}"'] & (E(P(X)),K) \arrow[d, Rightarrow, "\text{Substitute into result of symbolic execution}"] \\
P(K) \arrow[r, Rightarrow, "\text{Conventional execution}"'] & E(P(K))
\end{tikzcd}$$

Commutativity

- The same result is obtained using normal execution or using symbolic execution.
- Normal execution
  - Sum(a, b, c) ⇒ Sum(1, 3, 5)

$$\begin{tikzcd}
\mathbf{(P(X),K)} \arrow[r, Rightarrow, "\text{Symbolic execution}"] \arrow[d, Rightarrow, "\textbf{Set parameters to integer values}"'] & (E(P(X)),K) \arrow[d, Rightarrow, "\text{Substitute into result of symbolic execution}"] \\
P(K) \arrow[r, Rightarrow, "\text{Conventional execution}"'] & E(P(K))
\end{tikzcd}$$

Commutativity

- The same result is obtained using normal execution or using symbolic execution.
- Normal execution
  - Sum(a, b, c) $\Rightarrow$ Sum(1, 3, 5)
  - Sum(1, 3, 5) $= 9$

$$\begin{tikzcd}
(P(X),K) \arrow[r, Rightarrow, "\text{Symbolic execution}"] \arrow[d, Rightarrow, "\text{Set parameters to integer values}"'] & (E(P(X)),K) \arrow[d, Rightarrow, "\text{Substitute into result of symbolic execution}"] \\
\mathbf{P(K)} \arrow[r, Rightarrow, "\textbf{Conventional execution}"'] & \mathbf{E(P(K))}
\end{tikzcd}$$

Commutativity

- The same result is obtained using normal execution or using symbolic execution.
- Normal execution
  - Sum(a, b, c) $\Rightarrow$ Sum(1, 3, 5)
  - Sum(1, 3, 5) $=$ 9
- Symbolic execution

$$\begin{tikzcd}
(P(X),K) \arrow[r, Rightarrow, "\text{Symbolic execution}"] \arrow[d, Rightarrow, "\text{Set parameters to integer values}"'] & (E(P(X)),K) \arrow[d, Rightarrow, "\text{Substitute into result of symbolic execution}"] \\
P(K) \arrow[r, Rightarrow, "\text{Conventional execution}"'] & E(P(K))
\end{tikzcd}$$

Commutativity

- The same result is obtained using normal execution or using symbolic execution.
- Normal execution
  - Sum(a, b, c) $\Rightarrow$ Sum(1, 3, 5)
  - Sum(1, 3, 5) $= 9$
- Symbolic execution
  - Sum(a, b, c) $= \alpha + \beta + \gamma$

$$\begin{tikzcd}
\mathbf{(P(X),K)} \arrow[r, Rightarrow, "\textbf{Symbolic}"] \arrow[d, Rightarrow, "\text{Set parameters to integer values}"'] & \mathbf{(E(P(X)),K)} \arrow[d, Rightarrow, "\text{Substitute into result of symbolic execution}"] \\
\text{P(K)} \arrow[r, Rightarrow, "\text{Conventional execution}"'] & \text{E(P(K))}
\end{tikzcd}$$

Commutativity

- The same result is obtained using normal execution or using symbolic execution.
- Normal execution
  - Sum(a, b, c) ⇒ Sum(1, 3, 5)
  - Sum(1, 3, 5) $=$ 9
- Symbolic execution
  - Sum(a, b, c) $= \alpha + \beta + \gamma$
  - Instantiate the symbolic result ⇒ α $=$ 1, β= 3 and γ=5 ⇒ 1+3+5=9.

$$\begin{tikzcd}
(P(X),K) \arrow[r, Rightarrow, "\text{Symbolic execution}"] \arrow[d, Rightarrow, "\text{Set parameters to integer values}"'] & \mathbf{(E(P(X)),K)} \arrow[d, Rightarrow, "\textbf{Substitute into result of symbolic execution}"] \\
P(K) \arrow[r, Rightarrow, "\text{Conventional execution}"'] & \mathbf{E(P(K))}
\end{tikzcd}$$

If statement

- Symbolic execution of an IF statement

If statement

- Symbolic execution of an IF statement
  - if η then
  - A
  - else
  - B.

If statement

- Symbolic execution of an IF statement
    - if η then
    - A
    - else
    - B.
- During symbolic execution ⇒ **value(η)** could be **true**, **false**, or some symbolic formula over the input symbols.

If statement

- Symbolic execution of an IF statement
  - if η then
  - A
  - else
  - B.
- During symbolic execution ⇒ **value(η)** could be **true**, **false**, or some symbolic formula over the input symbols.
  ⇒ "unresolved" execution of a conditional statement

If statement

- Symbolic execution of an IF statement
    - if η then
    - A
    - else
    - B.
- During symbolic execution ⇒ **value(η)** could be **true**, **false**, or some symbolic formula over the input symbols.
  ⇒ "unresolved" execution of a conditional statement
- If value(η) and reaching a statement with condition τ

or

If statement

- Symbolic execution of an IF statement
  - if η then
  - A
  - else
  - B.
- During symbolic execution ⇒ **value(η)** could be **true**, **false**, or some symbolic formula over the input symbols.
  ⇒ "unresolved" execution of a conditional statement
- If value(η) and reaching a statement with condition τ
  ⇒ value(η) $\supset$ value(τ) or

If statement

- Symbolic execution of an IF statement
  - if η then
  - A
  - else
  - B.
- During symbolic execution ⇒ **value(η)** could be **true**, **false**, or some symbolic formula over the input symbols.
  ⇒ "unresolved" execution of a conditional statement
- If value(η) and reaching a statement with condition τ
  ⇒ value(η) $\supset$ value(τ) or value(η) $\supset \neg$ value(τ)

If statement

- Symbolic execution of an IF statement
  - if η then
  - A
  - else
  - B.
- During symbolic execution ⇒ **value(**η**)** could be **true**, **false**, or some symbolic formula over the input symbols.
  ⇒ "unresolved" execution of a conditional statement
- If value(η) and reaching a statement with condition τ
  ⇒ value(η) $\supset$ value(τ) or value(η) $\supset \neg$ value(τ)
- Path Condition (Initial value of *pc* is true)

If statement

- Symbolic execution of an IF statement
  - if η then
  - A
  - else
  - B.
- During symbolic execution ⇒ **value(η)** could be **true**, **false**, or some symbolic formula over the input symbols.
  ⇒ "unresolved" execution of a conditional statement
- If value(η) and reaching a statement with condition τ
  ⇒ value(η) $\supset$ value(τ) or value(η) $\supset \neg$ value(τ)
- Path Condition (Initial value of *pc* is true)
  - Using the current path condition(pc), we have two following expressions:

If statement

- Symbolic execution of an IF statement
  - if η then
  - A
  - else
  - B.
- During symbolic execution $\Rightarrow$ **value(η)** could be **true**, **false**, or some symbolic formula over the input symbols.
  $\Rightarrow$ "unresolved" execution of a conditional statement
- If value(η) and reaching a statement with condition τ
  $\Rightarrow$ value(η) $\supset$ value(τ) or value(η) $\supset$ $\neg$ value(τ)
- Path Condition (Initial value of *pc* is true)
  - Using the current path condition(pc), we have two following expressions:
    - pc $\rightarrow$ η

If statement

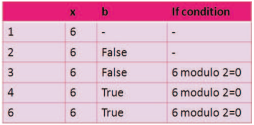
- Symbolic execution of an IF statement
  - if η then
  - A
  - else
  - B.
- During symbolic execution $\Rightarrow$ **value(η)** could be **true**, **false**, or some symbolic formula over the input symbols.
  $\Rightarrow$ "unresolved" execution of a conditional statement
- If value(η) and reaching a statement with condition τ
  $\Rightarrow$ value(η) $\supset$ value(τ) or value(η) $\supset \neg$ value(τ)
- Path Condition (Initial value of *pc* is true)
  - Using the current path condition(pc), we have two following expressions:
    - pc $\rightarrow \eta$

Conventional execution

- Function IsEven

Conventional execution

- Function IsEven
- 1 : boolean IsEven(int a)
- 2 : boolean b := False;
- 3: If (x modulo 2 =0) then
- 4: b:=true;
- else
- 5: b:=false;
- 6: IsEven:=b;
- 7:



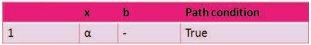
	x	b	If condition
1	6	-	-
2	6	False	-
3	6	False	6 modulo 2=0
4	6	True	6 modulo 2=0
6	6	True	6 modulo 2=0

Symbolic execution

- Function IsEven

Symbolic execution

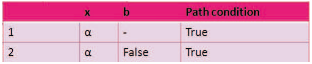
- Function IsEven
- 1 : boolean IsEven(int a)
- 2 : boolean b := False;
- 3: If (x modulo 2 = 0) then
- 4: b:=true;
- else
- 5: b:=false;
- 6: IsEven:=b;
- 7:



	x	b	Path condition
1	α	-	True

Symbolic execution

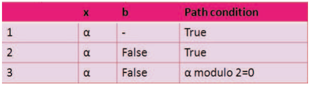
- Function IsEven
- 1 : boolean IsEven(int a)
- 2 : boolean b := False;
- 3: If (x modulo 2 = 0) then
- 4: b:=true;
- else
- 5: b:=false;
- 6: IsEven:=b;
- 7:



	x	b	Path condition
1	α	-	True
2	α	False	True

Symbolic execution

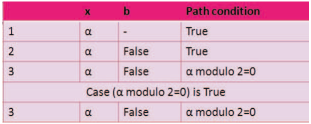
- Function IsEven
- 1 : boolean IsEven(int a)
- 2 : boolean b := False;
- 3: If (x modulo 2 = 0) then
- 4: b:=true;
- else
- 5: b:=false;
- 6: IsEven:=b;
- 7:



	x	b	Path condition
1	α	-	True
2	α	False	True
3	α	False	α modulo 2=0

Symbolic execution

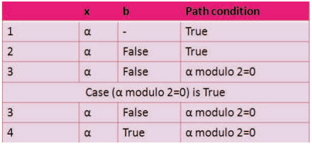
- Function IsEven
- 1 : boolean IsEven(int a)
- 2 : boolean b := False;
- 3: If (x modulo 2 = 0) then
- 4: b:=true;
- else
- 5: b:=false;
- 6: IsEven:=b;
- 7:



	x	b	Path condition
1	α	-	True
2	α	False	True
3	α	False	α modulo 2=0
Case (α modulo 2=0) is True			
3	α	False	α modulo 2=0

Symbolic execution

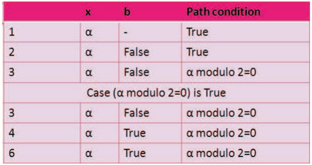
- Function IsEven
- 1 : boolean IsEven(int a)
- 2 : boolean b := False;
- 3: If (x modulo 2 = 0) then
- 4: b:=true;
- else
- 5: b:=false;
- 6: IsEven:=b;
- 7:



	x	b	Path condition
1	α	-	True
2	α	False	True
3	α	False	α modulo 2=0
Case (α modulo 2=0) is True			
3	α	False	α modulo 2=0
4	α	True	α modulo 2=0

Symbolic execution

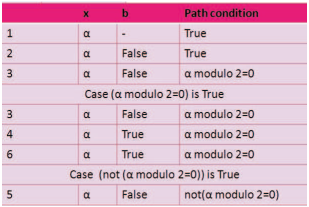
- Function IsEven
- 1 : boolean IsEven(int a)
- 2 : boolean b := False;
- 3: If (x modulo 2 = 0) then
- 4: b:=true;
- else
- 5: b:=false;
- 6: IsEven:=b;
- 7:



	x	b	Path condition
1	α	-	True
2	α	False	True
3	α	False	α modulo 2=0
Case (α modulo 2=0) is True			
3	α	False	α modulo 2=0
4	α	True	α modulo 2=0
6	α	True	α modulo 2=0

Symbolic execution

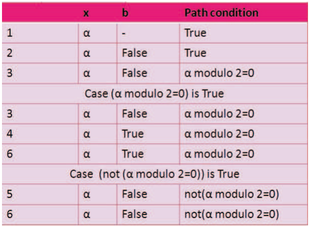
- Function IsEven
- 1 : boolean IsEven(int a)
- 2 : boolean b := False;
- 3: If (x modulo 2 = 0) then
- 4: b:=true;
- else
- 5: b:=false;
- 6: IsEven:=b;
- 7:



	x	b	Path condition
1	α	-	True
2	α	False	True
3	α	False	α modulo 2=0
Case (α modulo 2=0) is True			
3	α	False	α modulo 2=0
4	α	True	α modulo 2=0
6	α	True	α modulo 2=0
Case (not (α modulo 2=0)) is True			
5	α	False	not(α modulo 2=0)

Symbolic execution

- Function IsEven
- 1 : boolean IsEven(int a)
- 2 : boolean b := False;
- 3: If (x modulo 2 = 0) then
- 4: b:=true;
- else
- 5: b:=false;
- 6: IsEven:=b;
- 7:



	x	b	Path condition
1	α	-	True
2	α	False	True
3	α	False	α modulo 2=0
Case (α modulo 2=0) is True			
3	α	False	α modulo 2=0
4	α	True	α modulo 2=0
6	α	True	α modulo 2=0
Case (not (α modulo 2=0)) is True			
5	α	False	not(α modulo 2=0)
6	α	False	not(α modulo 2=0)

While statement

- Symbolic execution of an WHILE statement

While statement

- Symbolic execution of an WHILE statement
  - while η then
  -       A
  - endWh;
  - B.

While statement

- Symbolic execution of an WHILE statement
  - while η then
  - A
  - endWh;
  - B.
- During symbolic execution $\Rightarrow$ **value(η)** could be **true**, **false**, or some symbolic formula over the input symbols.

While statement

- Symbolic execution of an WHILE statement
  - while η then
  - A
  - endWh;
  - B.
- During symbolic execution ⇒ **value(η)** could be **true**, **false**, or some symbolic formula over the input symbols.
  ⇒ "unresolved" execution of a conditional statement

While statement

- Symbolic execution of an WHILE statement
    - while η then
    - A
    - endWh;
    - B.
- During symbolic execution ⇒ **value(η)** could be **true**, **false**, or some symbolic formula over the input symbols.
  ⇒ "unresolved" execution of a conditional statement
- Condition to execute A: pc for executing "while" and η.

While statement

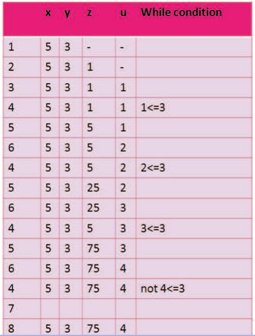
- Symbolic execution of an WHILE statement
  - while η then
  - A
  - endWh;
  - B.
- During symbolic execution ⇒ **value(η)** could be **true**, **false**, or some symbolic formula over the input symbols.
  ⇒ "unresolved" execution of a conditional statement
- Condition to execute A: pc for executing "while" and η.
- Condition to execute B: pc for executing "while" and $\neg\ \eta$.

Conventional execution

- Subalg. Power

Conventional execution

- Subalg. Power
- 1 : Power(int x, int y, int z)
- 2 : z := 1;
- 3: u:=1
- 4: while($u \leq y$)
- 5: z:=z*x;
- 6: u:=u+1
- 7: endwh;
- 8:



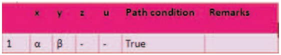
	x	y	z	u	While condition
1	5	3	-	-	
2	5	3	1	-	
3	5	3	1	1	
4	5	3	1	1	1<=3
5	5	3	5	1	
6	5	3	5	2	
4	5	3	5	2	2<=3
5	5	3	25	2	
6	5	3	25	3	
4	5	3	5	3	3<=3
5	5	3	75	3	
6	5	3	75	4	
4	5	3	75	4	not 4<=3
7					
8	5	3	75	4	

Symbolic execution

- Subalg. Power

Symbolic execution

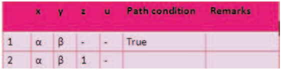
- Subalg. Power
- 1 : Power(int x, int y, int z)
- 2 : z := 1;
- 3: u:=1
- 4: while($u \leq y$)
- 5: z:=z*x;
- 6: u:=u+1
- 7: endwh;
- 8:



	x	y	z	u	Path condition	Remarks
1	α	β	-	-	True	

Symbolic execution

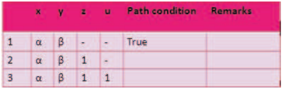
- Subalg. Power
- 1 : Power(int x, int y, int z)
- 2 : z := 1;
- 3: u:=1
- 4: while($u \leq y$)
- 5: z:=z*x;
- 6: u:=u+1
- 7: endwh;
- 8:



	x	y	z	u	Path condition	Remarks
1	α	β	-	-	True	
2	α	β	1	-		

Symbolic execution

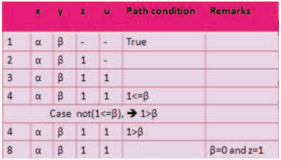
- Subalg. Power
- 1 : Power(int x, int y, int z)
- 2 : z := 1;
- 3: u:=1
- 4: while($u \leq y$)
- 5: z:=z*x;
- 6: u:=u+1
- 7: endwh;
- 8:



	x	y	z	u	Path condition	Remarks
1	α	β	-	-	True	
2	α	β	1	-		
3	α	β	1	1		

Symbolic execution

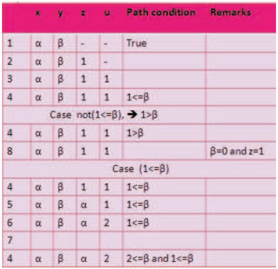
- Subalg. Power
- 1 : Power(int x, int y, int z)
- 2 : z := 1;
- 3: u:=1
- 4: while($u \leq y$)
- 5: z:=z*x;
- 6: u:=u+1
- 7: endwh;
- 8:



	x	y	z	u	Path condition	Remarks
1	α	β	-	-	True	
2	α	β	1	-		
3	α	β	1	1		
4	α	β	1	1	1<=β	
Case not(1<=β), ➔ 1>β						
4	α	β	1	1	1>β	
8	α	β	1	1		β=0 and z=1

Symbolic execution

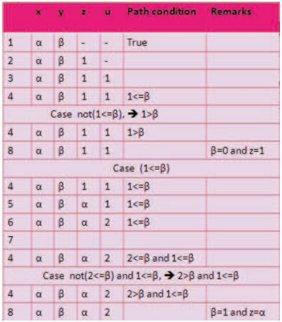
- Subalg. Power
- 1 : Power(int x, int y, int z)
- 2 : z := 1;
- 3: u:=1
- 4: while($u \leq y$)
- 5: z:=z*x;
- 6: u:=u+1
- 7: endwh;
- 8:



	x	y	z	u	Path condition	Remarks
1	α	β	-	-	True	
2	α	β	1	-		
3	α	β	1	1		
4	α	β	1	1	1<=β	
Case not(1<=β), ➔ 1>β						
4	α	β	1	1	1>β	
8	α	β	1	1		β=0 and z=1
Case (1<=β)						
4	α	β	1	1	1<=β	
5	α	β	α	1	1<=β	
6	α	β	α	2	1<=β	
7						
4	α	β	α	2	2<=β and 1<=β	

Symbolic execution

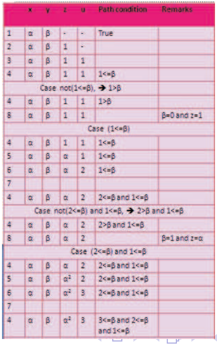
- Subalg. Power
- 1 : Power(int x, int y, int z)
- 2 : z := 1;
- 3: u:=1
- 4: while($u \leq y$)
- 5: z:=z*x;
- 6: u:=u+1
- 7: endwh;
- 8:



	x	y	z	u	Path condition	Remarks
1	α	β	-	-	True	
2	α	β	1	-		
3	α	β	1	1		
4	α	β	1	1	1<=β	
Case not(1<=β), ➔ 1>β						
4	α	β	1	1	1>β	
8	α	β	1	1		β=0 and z=1
Case (1<=β)						
4	α	β	1	1	1<=β	
5	α	β	α	1	1<=β	
6	α	β	α	2	1<=β	
7						
4	α	β	α	2	2<=β and 1<=β	
Case not(2<=β) and 1<=β, ➔ 2>β and 1<=β						
4	α	β	α	2	2>β and 1<=β	
8	α	β	α	2		β=1 and z=α

Symbolic execution

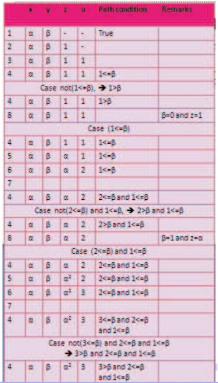
- Subalg. Power
- 1 : Power(int x, int y, int z)
- 2 : z := 1;
- 3: u:=1
- 4: while($u \leq y$)
- 5: z:=z*x;
- 6: u:=u+1
- 7: endwh;
- 8:



	x	y	z	u	Path condition	Remarks
1	α	β	-	-	True	
2	α	β	1	-		
3	α	β	1	1		
4	α	β	1	1	1<=β	
Case not(1<=β), ➔ 1>β						
4	α	β	1	1	1>β	
8	α	β	1	1		β=0 and z=1
Case (1<=β)						
4	α	β	1	1	1<=β	
5	α	β	α	1	1<=β	
6	α	β	α	2	1<=β	
7						
4	α	β	α	2	2<=β and 1<=β	
Case not(2<=β) and 1<=β, ➔ 2>β and 1<=β						
4	α	β	α	2	2>β and 1<=β	
8	α	β	α	2		β=1 and z=α
Case (2<=β) and 1<=β						
4	α	β	α	2	2<=β and 1<=β	
5	α	β	$α^2$	2	2<=β and 1<=β	
6	α	β	$α^2$	3	2<=β and 1<=β	
7						
4	α	β	$α^2$	3	3<=β and 2<=β and 1<=β	

Symbolic execution

- Subalg. Power
- 1 : Power(int x, int y, int z)
- 2 : z := 1;
- 3: u:=1
- 4: while($u \leq y$)
- 5: z:=z*x;
- 6: u:=u+1
- 7: endwh;
- 8:



	x	y	z	u	Path condition	Remarks
1	α	β	-	-	True	
2	α	β	1	-		
3	α	β	1	1		
4	α	β	1	1	1<=β	
Case not(1<=β), ➔ 1>β						
4	α	β	1	1	1>β	
8	α	β	1	1		β=0 and z=1
Case (1<=β)						
4	α	β	1	1	1<=β	
5	α	β	α	1	1<=β	
6	α	β	α	2	1<=β	
7						
4	α	β	α	2	2<=β and 1<=β	
Case not(2<=β) and 1<=β, ➔ 2>β and 1<=β						
4	α	β	α	2	2>β and 1<=β	
8	α	β	α	2		β=1 and z=α
Case (2<=β) and 1<=β						
4	α	β	α	2	2<=β and 1<=β	
5	α	β	$α^2$	2	2<=β and 1<=β	
6	α	β	$α^2$	3	2<=β and 1<=β	
7						
4	α	β	$α^2$	3	3<=β and 2<=β and 1<=β	
Case not(3<=β) and 2<=β and 1<=β ➔ 3>β and 2<=β and 1<=β						
4	α	β	$α^2$	3	3>β and 2<=β and 1<=β	

Symbolic Execution Tree

- We can generate symbolic execution tree characterizing the execution paths followed during the symbolic execution.

Symbolic Execution Tree

- We can generate symbolic execution tree characterizing the execution paths followed during the symbolic execution.
  - Associate a node with each statement executed.

Symbolic Execution Tree

- We can generate symbolic execution tree characterizing the execution paths followed during the symbolic execution.
    - Associate a node with each statement executed.
    - Associate a directed arc connecting the associated nodes with each transition between statements.

Symbolic Execution Tree

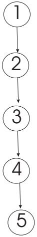
- We can generate symbolic execution tree characterizing the execution paths followed during the symbolic execution.
    - Associate a node with each statement executed.
    - Associate a directed arc connecting the associated nodes with each transition between statements.
    - For IF statement execution, the associated node has two arcs leaving the node which are labeled “T” and “F” for the true and false part, respectively.

Symbolic Execution Tree

- We can generate symbolic execution tree characterizing the execution paths followed during the symbolic execution.
    - Associate a node with each statement executed.
    - Associate a directed arc connecting the associated nodes with each transition between statements.
    - For IF statement execution, the associated node has two arcs leaving the node which are labeled “T” and “F” for the true and false part, respectively.
    - Associate the complete current execution state, i.e. variable values, statement counter, and pc with each node.

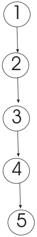
Symbolic Execution Tree

- Function Sum



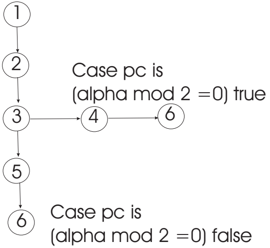
Symbolic Execution Tree

- Function Sum
- 1: int Sum(int a, int b, int c)
- 2: int x := a + b;
- 3: int y := b + c;
- 4: int z := *x* + *y* - *b*;
- 5: return z;
- 6:



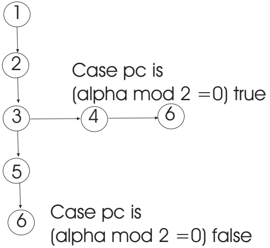
Symbolic Execution Tree

- Function IsEven



Symbolic Execution Tree

- Function IsEven
- 1 : boolean IsEven(int a)
- 2 : boolean b := False;
- 3: If (x modulo 2 =0) then
- 4: b:=true;
- else
- 5: b:=false;
- 6: IsEven:=b;
- 7:

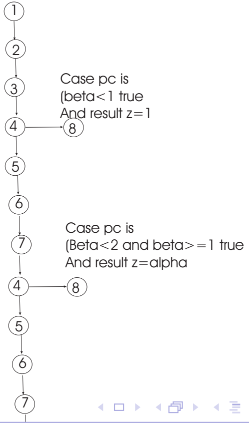


Case pc is
(alpha mod 2 =0) true

Case pc is
(alpha mod 2 =0) false

Symbolic Execution Tree

- Subalg. Power

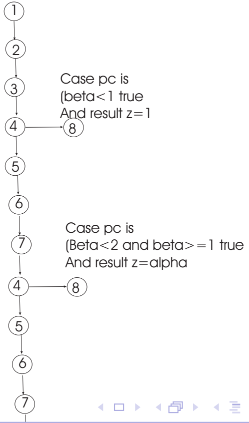


Case pc is
(beta<1 true
And result z=1

Case pc is
(Beta<2 and beta>=1 true
And result z=alpha

Symbolic Execution Tree

- Subalg. Power
- 1 : Power(int x, int y, int z)
- 2 : z := 1;
- 3: u:=1
- 4: while($u \leq y$)
- 5: z:=z*x;
- 6: u:=u+1
- 7: endwh;
- 8:



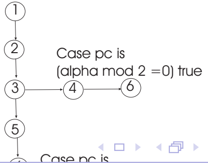
Properties of the Symbolic Execution Tree

- For each terminal leaf exists a particular nonsymbolic input.

Properties of the Symbolic Execution Tree

- For each terminal leaf exists a particular nonsymbolic input.
- The pc associated with any two terminal leaves are distinct.

- Function IsEven
- 1 : boolean IsEven(int a)
- 2 : boolean b := False;
- 3: If (x modulo 2 =0) then
- 4: b:=true;
- else
- 5: b:=false;
- 6: IsEven:=b;



Test case generation

- Test cases - to execute every statement at least once.

Test case generation

- Test cases - to execute every statement at least once.
- Test cases - to include execution of each branch both ways.

Test case generation

- Test cases - to execute every statement at least once.
- Test cases - to include execution of each branch both ways.
- Test case - finding input values to reach a particular point in a program ?

Test case generation

- Test cases - to execute every statement at least once.
- Test cases - to include execution of each branch both ways.
- Test case - finding input values to reach a particular point in a program ?

  Remaining problem - to instantiate the pc with particular values.

Test case generation

- Test cases - to execute every statement at least once.
- Test cases - to include execution of each branch both ways.
- Test case - finding input values to reach a particular point in a program ?
  Remaining problem - to instantiate the pc with particular values.
- The pc specifies a class of equivalent tests, and any feasible solution to the constraints (represented by the pc) would be a representative member.

Test case generation

- Test cases - to execute every statement at least once.
- Test cases - to include execution of each branch both ways.
- Test case - finding input values to reach a particular point in a program ?
  Remaining problem - to instantiate the pc with particular values.
- The pc specifies a class of equivalent tests, and any feasible solution to the constraints (represented by the pc) would be a representative member.
- The symbolic execution also provides expressions describing the program outputs for all inputs in this set.

Correctness by Symbolic testing

- Informal induction

Correctness by Symbolic testing

- Informal induction
- Program verification

Correctness by Symbolic testing

- Informal induction
- Program verification
    - A proof to be performed in terms of symbolic execution, based on standard inductive assertion method.

Next lecture

- Lecture 06
  - EVOZON: Testing Automation with Java and Selenium WebDriver.
  - Monday, March 30, 2015; hours 10:00-12:00; N. Iorga Hall (2/I), Main Building.
  - Compulsory Attendance
- Lecture 07
  - Midterm Exam: Lecture 02 + Lecture 03
  - Monday, April 6, 2015; hours 8:00-10:00; 6/II, Main Building.
  - Compulsory Attendance

Questions

- Thank You For Your Attention!

Questions

- Thank You For Your Attention!