

EC4219: Software Engineering

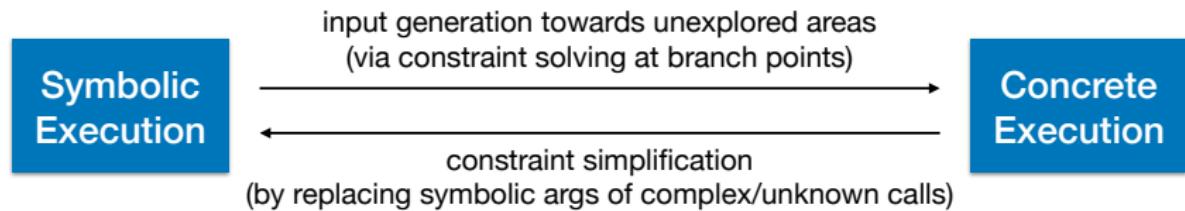
Lecture 15 — Symbolic Execution (2) *More Complex Examples*

Sunbeam So
2024 Spring

Symbolic execution is a testing approach for systematically finding SW bugs.

- Key idea: execute programs “symbolically” using “symbolic” inputs
- Challenge: path explosion + constraint solving

Concolic testing is a hybrid approach to mitigate the second challenge.



- Downside: false negative due to overly-constrained simplification
 - ▶ E.g., most hash functions are not collision-free thus F can be satisfiable. However, if $\alpha = 22$, $\beta = 7$, and $\text{hash}(22) \neq \text{hash}(7)$ from concrete execution, we incorrectly decide F is unsatisfiable.

$$F : \alpha \neq \beta \wedge \text{hash}(\alpha) = \text{hash}(\beta)$$

This Lecture

We will study how concolic testing works in more complex cases.

- Concolic testing of programs with loops
- Concolic testing of programs with data structures

Example: Concolic Testing of Loops

```
void testme(int x) {  
    int A[] = { 5, 7, 9 };  
  
    int i = 0;  
  
    while (i < 3) {  
        if (A[i] == x) break;  
        i++;  
    }  
  
    return i;  
}
```

Concrete
State

x=1

Symbolic
State

x=a
true

1st iteration

Example: Concolic Testing of Loops (Cont'd)

Concrete State	Symbolic State
x=1 A = {5,7,9}	x=a true

```
void testme(int x) {
    int A[] = { 5, 7, 9 };
    int i = 0;
    while (i < 3) {
        if (A[i] == x) break;
        i++;
    }
    return i;
}
```

1st iteration

Example: Concolic Testing of Loops (Cont'd)

```
void testme(int x) {  
  
    int A[] = { 5, 7, 9 };  
  
    int i = 0;  
    ←—————  
    while (i < 3) {  
        if (A[i] == x) break;  
        i++;  
    }  
  
    return i;  
}
```

Concrete
State

x=1, i=0,
A = {5,7,9}

Symbolic
State

x=a
true

1st iteration

Example: Concolic Testing of Loops (Cont'd)

```
void testme(int x) {  
  
    int A[] = { 5, 7, 9 };  
  
    int i = 0;  
    ←—————  
    while (i < 3) {  
        if (A[i] == x) break;  
        i++;  
    }  
  
    return i;  
}
```

Concrete
State

x=1, i=0,
A = {5,7,9}

Symbolic
State

x=a
true

1st iteration

Example: Concolic Testing of Loops (Cont'd)

Concrete State	Symbolic State
<pre>void testme(int x) { int A[] = { 5, 7, 9 }; int i = 0; while (i < 3) { if (A[i] == x) break; i++; } return i; }</pre>	<p>x=1, i=0, A = {5,7,9}</p> <p>x=a true</p>

1st iteration

Example: Concolic Testing of Loops (Cont'd)

Concrete State	Symbolic State
<pre>void testme(int x) { int A[] = { 5, 7, 9 }; int i = 0; while (i < 3) { if (A[i] == x) break; i++; } return i; }</pre> <p>1st iteration</p>	<p>x=1, i=0, A = {5,7,9}</p> <p>x=a 5≠a</p>

Example: Concolic Testing of Loops (Cont'd)

Concrete State	Symbolic State
<pre>void testme(int x) { int A[] = { 5, 7, 9 }; int i = 0; while (i < 3) { if (A[i] == x) break; i++; } return i; }</pre> <p>1st iteration</p>	<p>x=1, i=1, A = {5,7,9}</p> <p>x=a 5≠a</p>

Example: Concolic Testing of Loops (Cont'd)

Concrete State	Symbolic State
<pre>void testme(int x) { int A[] = { 5, 7, 9 }; int i = 0; while (i < 3) { if (A[i] == x) break; i++; } return i; }</pre>	<p>1st iteration</p> <p>x=1, i=1, A = {5,7,9}</p> <p>x=a 5≠a</p>

Example: Concolic Testing of Loops (Cont'd)

Concrete State	Symbolic State
<pre>void testme(int x) { int A[] = { 5, 7, 9 }; int i = 0; while (i < 3) { if (A[i] == x) break; i++; } return i; }</pre> <p>1st iteration</p>	<p>x=1, i=1, A = {5,7,9}</p> <p>x=a 5≠a ∧ 7≠a</p>

Example: Concolic Testing of Loops (Cont'd)

Concrete State	Symbolic State
<pre>void testme(int x) { int A[] = { 5, 7, 9 }; int i = 0; while (i < 3) { if (A[i] == x) break; i++; } return i; }</pre> <p>1st iteration</p>	<p>x=1, i=2, A = {5,7,9}</p> <p>x=a</p> <p>$5 \neq a \wedge 7 \neq a$</p>

Example: Concolic Testing of Loops (Cont'd)

Concrete State	Symbolic State
<pre>void testme(int x) { int A[] = { 5, 7, 9 }; int i = 0; while (i < 3) { if (A[i] == x) break; i++; } return i; }</pre>	<p>1st iteration</p> <p>x=1, i=2, A = {5,7,9}</p> <p>x=a 5≠a ∧ 7≠a</p>

Example: Concolic Testing of Loops (Cont'd)

Concrete State	Symbolic State
<pre>void testme(int x) { int A[] = { 5, 7, 9 }; int i = 0; while (i < 3) { if (A[i] == x) break; i++; } return i; }</pre> <p>1st iteration</p>	<p>x=a $5 \neq a \wedge 7 \neq a \wedge 9 \neq a$</p>

Example: Concolic Testing of Loops (Cont'd)

Concrete State	Symbolic State
<pre>void testme(int x) { int A[] = { 5, 7, 9 }; int i = 0; while (i < 3) { if (A[i] == x) break; i++; } return i; }</pre> <p>1st iteration</p>	<p>x=1, i=3, A = {5,7,9}</p> <p>x=a 5≠a ∧ 7≠a ∧ 9≠a</p>

Example: Concolic Testing of Loops (Cont'd)

Concrete State	Symbolic State
<pre>void testme(int x) { int A[] = { 5, 7, 9 }; int i = 0; while (i < 3) { if (A[i] == x) break; i++; } return i;</pre> <p>1st iteration</p>	$x=1, i=3,$ $A = \{5, 7, 9\}$ $x=a$ $5 \neq a \wedge 7 \neq a \wedge$ $9 \neq a$

Example: Concolic Testing of Loops (Cont'd)

```
void testme(int x) {  
  
    int A[] = { 5, 7, 9 };  
  
    int i = 0;  
  
    while (i < 3) {  
        if (A[i] == x) break;  
        i++;  
    }  
  
    return i;  
}
```

1st iteration

Concrete
State

Symbolic
State

- Constraint: $5 \neq a \wedge 7 \neq a \wedge 9 = a$
- Solution: $a=9$

$x=1, i=3,$
 $A = \{5, 7, 9\}$

$x=a$

$5 \neq a \wedge 7 \neq a \wedge$
 $9 \neq a$

Example: Concolic Testing of Loops (Cont'd)

	Concrete State	Symbolic State
void testme(int x) { int A[] = { 5, 7, 9 }; int i = 0; while (i < 3) { if (A[i] == x) break; i++; } return i; }	x=9	x=a true

2nd iteration

Example: Concolic Testing of Loops (Cont'd)

Concrete State	Symbolic State
x=9, A = {5,7,9}	x=a true

```
void testme(int x) {
    int A[] = { 5, 7, 9 };
    int i = 0;
    while (i < 3) {
        if (A[i] == x) break;
        i++;
    }
    return i;
}
```

2nd iteration

Example: Concolic Testing of Loops (Cont'd)

```
void testme(int x) {  
  
    int A[] = { 5, 7, 9 };  
  
    int i = 0;  
    ←—————  
    while (i < 3) {  
        if (A[i] == x) break;  
        i++;  
    }  
  
    return i;  
}
```

Concrete
State

x=9, i=0,
A = {5,7,9}

Symbolic
State

x=a
true

2nd iteration

Example: Concolic Testing of Loops (Cont'd)

Concrete State	Symbolic State
<pre>void testme(int x) { int A[] = { 5, 7, 9 }; int i = 0; while (i < 3) { if (A[i] == x) break; i++; } return i; }</pre>	<p>x=9, i=0, A = {5,7,9}</p> <p>x=a true</p>

2nd iteration

Example: Concolic Testing of Loops (Cont'd)

Concrete State	Symbolic State
<pre>void testme(int x) { int A[] = { 5, 7, 9 }; int i = 0; while (i < 3) { if (A[i] == x) break; i++; } return i; }</pre>	<p>2nd iteration</p> <p>x=9, i=0, A = {5,7,9}</p> <p>x=a 5≠a</p>

Example: Concolic Testing of Loops (Cont'd)

Concrete State	Symbolic State
<pre>void testme(int x) { int A[] = { 5, 7, 9 }; int i = 0; while (i < 3) { if (A[i] == x) break; i++; } return i; }</pre> <p>2nd iteration</p>	<p>x=9, i=1, A = {5,7,9}</p> <p>x=a 5≠a</p>

Example: Concolic Testing of Loops (Cont'd)

Concrete State	Symbolic State
<pre>void testme(int x) { int A[] = { 5, 7, 9 }; int i = 0; while (i < 3) { if (A[i] == x) break; i++; } return i; }</pre>	<p>x=9, i=1, A = {5,7,9}</p> <p>x=a 5≠a</p>

2nd iteration

Example: Concolic Testing of Loops (Cont'd)

Concrete State	Symbolic State
<pre>void testme(int x) { int A[] = { 5, 7, 9 }; int i = 0; while (i < 3) { if (A[i] == x) break; i++; } return i; }</pre>	<p>2nd iteration</p> <p>x=9, i=1, A = {5,7,9}</p> <p>x=a 5≠a ∧ 7≠a</p>

Example: Concolic Testing of Loops (Cont'd)

Concrete State	Symbolic State
<pre>void testme(int x) { int A[] = { 5, 7, 9 }; int i = 0; while (i < 3) { if (A[i] == x) break; i++; } return i; }</pre> <p>2nd iteration</p>	<p>x=9, i=2, A = {5,7,9}</p> <p>x=a</p> <p>$5 \neq a \wedge 7 \neq a$</p>

Example: Concolic Testing of Loops (Cont'd)

Concrete State	Symbolic State
void testme(int x) { int A[] = { 5, 7, 9 }; int i = 0; while (i < 3) { if (A[i] == x) break; i++; } return i; }	x=9, i=2, A = {5,7,9} x=a 5≠a ∧ 7≠a

2nd iteration

Example: Concolic Testing of Loops (Cont'd)

	Concrete State	Symbolic State
<pre>void testme(int x) { int A[] = { 5, 7, 9 }; int i = 0; while (i < 3) { if (A[i] == x) break; i++; } return i; }</pre>	x=9, i=2, A = {5,7,9}	x=a 5≠a ∧ 7≠a ∧ 9=a

2nd iteration

Example: Concolic Testing of Loops (Cont'd)

```
void testme(int x) {  
  
    int A[] = { 5, 7, 9 };  
  
    int i = 0;  
  
    while (i < 3) {  
        if (A[i] == x) break;  
        i++;  
    }  
  
    return i;  
}
```

2nd iteration

Concrete
State

Symbolic
State

- Constraint: $5 \neq a \wedge 7 = a$
- Solution: $a=7$

$x=9, i=2,$
 $A = \{5, 7, 9\}$

$x=a$
 $5 \neq a \wedge 7 \neq a \wedge$
 $9 = a$

Example: Concolic Testing of Loops (Cont'd)

	Concrete State	Symbolic State
void testme(int x) { int A[] = { 5, 7, 9 }; int i = 0; while (i < 3) { if (A[i] == x) break; i++; } return i; }	x=7	x=a true

3rd iteration

Example: Concolic Testing of Loops (Cont'd)

Concrete State	Symbolic State
x=7, A = {5,7,9}	x=a true

```
void testme(int x) {
    int A[] = { 5, 7, 9 };
    int i = 0;
    while (i < 3) {
        if (A[i] == x) break;
        i++;
    }
    return i;
}
```

3rd iteration

Example: Concolic Testing of Loops (Cont'd)

```
void testme(int x) {  
  
    int A[] = { 5, 7, 9 };  
  
    int i = 0;  
    ←—————  
    while (i < 3) {  
        if (A[i] == x) break;  
        i++;  
    }  
  
    return i;  
}
```

Concrete
State

x=7, i=0,
A = {5,7,9}

Symbolic
State

x=a
true

3rd iteration

Example: Concolic Testing of Loops (Cont'd)

Concrete State	Symbolic State
<pre>void testme(int x) { int A[] = { 5, 7, 9 }; int i = 0; while (i < 3) { if (A[i] == x) break; i++; } return i; }</pre>	<p>x=7, i=0, A = {5,7,9}</p> <p>x=a true</p>

3rd iteration

Example: Concolic Testing of Loops (Cont'd)

Concrete State	Symbolic State
<pre>void testme(int x) { int A[] = { 5, 7, 9 }; int i = 0; while (i < 3) { if (A[i] == x) break; i++; } return i; }</pre>	<p>3rd iteration</p> <p>x=7, i=0, A = {5,7,9}</p> <p>x=a 5≠a</p>

Example: Concolic Testing of Loops (Cont'd)

Concrete State	Symbolic State
<pre>void testme(int x) { int A[] = { 5, 7, 9 }; int i = 0; while (i < 3) { if (A[i] == x) break; i++; } return i; }</pre>	<p>3rd iteration</p> <p>$x=7, i=1,$ $A = \{5,7,9\}$</p> <p>$x=a$</p> <p>$5 \neq a$</p>

Example: Concolic Testing of Loops (Cont'd)

Concrete State	Symbolic State
<pre>void testme(int x) { int A[] = { 5, 7, 9 }; int i = 0; while (i < 3) { if (A[i] == x) break; i++; } return i; }</pre>	<p>x=7, i=1, A = {5,7,9}</p> <p>x=a 5≠a</p>

3rd iteration

Example: Concolic Testing of Loops (Cont'd)

```
void testme(int x) {  
  
    int A[] = { 5, 7, 9 };  
  
    int i = 0;  
  
    while (i < 3) {  
        if (A[i] == x) break;  
        i++;  
    }  
    return i;  
}
```

Concrete
State

x=7, i=2,
A = {5,7,9}

Symbolic
State

x=a
5≠a ∧ 7=a

3rd iteration

Example: Concolic Testing of Loops (Cont'd)

```
void testme(int x) {  
  
    int A[] = { 5, 7, 9 };  
  
    int i = 0;  
  
    while (i < 3) {  
        if (A[i] == x) break;  
        i++;  
    }  
  
    return i;  
}
```

Concrete
State

Symbolic
State

- Constraint: $5 \neq a$
- Solution: $a=5$

$x=7, i=2,$
 $A = \{5, 7, 9\}$

$x=a$

$5 \neq a \wedge 7 = a$

3rd iteration

Example: Concolic Testing of Loops (Cont'd)

	Concrete State	Symbolic State
void testme(int x) { int A[] = { 5, 7, 9 }; int i = 0; while (i < 3) { if (A[i] == x) break; i++; } return i; }	x=5	x=a true

4th iteration

Example: Concolic Testing of Loops (Cont'd)

Concrete State	Symbolic State
x=5, A = {5,7,9}	x=a true

```
void testme(int x) {
    int A[] = { 5, 7, 9 };
    int i = 0;
    while (i < 3) {
        if (A[i] == x) break;
        i++;
    }
    return i;
}
```

4th iteration

Example: Concolic Testing of Loops (Cont'd)

```
void testme(int x) {  
  
    int A[] = { 5, 7, 9 };  
  
    int i = 0;  
    ←—————  
    while (i < 3) {  
        if (A[i] == x) break;  
        i++;  
    }  
  
    return i;  
}
```

Concrete
State

x=5, i=0,
A = {5,7,9}

Symbolic
State

x=a
true

4th iteration

Example: Concolic Testing of Loops (Cont'd)

Concrete State	Symbolic State
<pre>void testme(int x) { int A[] = { 5, 7, 9 }; int i = 0; while (i < 3) { if (A[i] == x) break; i++; } return i; }</pre>	<p>x=5, i=0, A = {5,7,9}</p> <p>x=a true</p>

4th iteration

Example: Concolic Testing of Loops (Cont'd)

```
void testme(int x) {  
  
    int A[] = { 5, 7, 9 };  
  
    int i = 0;  
  
    while (i < 3) {  
        if (A[i] == x) break;  
        i++;  
    }  
    return i;  
}
```

Concrete
State

x=7, i=2,
A = {5,7,9}

Symbolic
State

x=a
5=a

4th iteration

Takeaways

- In symbolic states, we do not need to keep track of constraints that do not have data-dependency on inputs.
 - ▶ Why? To decide the next branch points to explore, it is enough to solve path constraints that are data-dependant on inputs.
- Negating path conditions does not always increase code coverage.
 - ▶ In the previous example, concolic testing terminates after four iterations but we executed all branches in the second iteration.

Example: Concolic Testing of Data Structures

	Concrete State	Symbolic State
typedef struct cell { int data; struct cell *next; } cell;		
int foo(int v) { return 2*v + 1; }		
void testme(int x, cell *p) { if (x > 0) if (p != NULL) if (foo(x) == p->data) if (p->next == p) Crash return 0; }	x=236 p=NULL	x=a, p=β true
	1st iteration	

Example: Concolic Testing of Data Structures (Cont'd)

	Concrete State	Symbolic State
typedef struct cell { int data; struct cell *next; } cell;		
int foo(int v) { return 2*v + 1; }		
void testme(int x, cell *p) { if (x > 0) if (p != NULL) if (foo(x) == p->data) if (p->next == p) Crash return 0; }	x=236 p=NULL	x=a, p=β a > 0
	1st iteration	

Example: Concolic Testing of Data Structures (Cont'd)

```
typedef struct cell {  
    int data;  
    struct cell *next;  
} cell;
```

```
int foo(int v) { return 2*v + 1; }
```

```
void testme(int x, cell *p) {  
    if (x > 0)  
        if (p != NULL)  
            if (foo(x) == p->data)  
                if (p->next == p)  
                    Crash  
    return 0; ←—————  
}
```

Concrete
State

x=236
p=NULL

Symbolic
State

$x=a$, $p=\beta$
 $a > 0 \wedge$
 $\beta = \text{NULL}$

1st iteration

Example: Concolic Testing of Data Structures (Cont'd)

```
typedef struct cell {  
    int data;  
    struct cell *next;  
} cell;
```

```
int foo(int v) { return 2*v }  
  
void testme(int x, cell *p) {  
    if (x > 0)  
        if (p != NULL)  
            if (foo(x) == p->data)  
                if (p->next == p)  
                    Crash  
    return 0; }
```

Concrete
State

Symbolic
State

- Constraint: $a > 0 \wedge \beta \neq \text{NULL}$
- Solution: $a = 236, \beta =$

634 NULL

x=236
p=NULL

x=a, p=β
 $a > 0 \wedge$
 $\beta = \text{NULL}$

1st iteration

Example: Concolic Testing of Data Structures (Cont'd)

```
typedef struct cell {  
    int data;  
    struct cell *next;  
} cell;
```

```
int foo(int v) { return 2*v + 1; }
```

```
void testme(int x, cell *p) {  
    if (x > 0)  
        if (p != NULL)  
            if (foo(x) == p->data)  
                if (p->next == p)  
                    Crash  
    return 0;  
}
```

Concrete
State

x=236
p=

634	NULL
-----	------

Symbolic
State

$x = \alpha, p = \beta$
 $p->\text{data} = \gamma$
 $p->\text{next} = \delta$
true

2nd iteration

Example: Concolic Testing of Data Structures (Cont'd)

```
typedef struct cell {  
    int data;  
    struct cell *next;  
} cell;
```

```
int foo(int v) { return 2*v + 1; }
```

```
void testme(int x, cell *p) {  
    if (x > 0)  
        if (p != NULL)  
            if (foo(x) == p->data)  
                if (p->next == p)  
                    Crash  
    return 0;  
}
```

Concrete
State

x=236
p =

634	NULL
-----	------

Symbolic
State

$x=a$, $p=\beta$
 $p->\text{data} = \gamma$
 $p->\text{next} = \delta$
 $a > 0$

2nd iteration

Example: Concolic Testing of Data Structures (Cont'd)

	Concrete State	Symbolic State
typedef struct cell { int data; struct cell *next; } cell;		
int foo(int v) { return 2*v + 1; }		
void testme(int x, cell *p) { if (x > 0) if (p != NULL) if (foo(x) == p->data) if (p->next == p) Crash return 0; }	x=236 p=NULL	x=a, p=β a > 0
	1st iteration	

Example: Concolic Testing of Data Structures (Cont'd)

```
typedef struct cell {  
    int data;  
    struct cell *next;  
} cell;
```

```
int foo(int v) { return 2*v + 1; }
```

```
void testme(int x, cell *p) {  
    if (x > 0)  
        if (p != NULL)  
            if (foo(x) == p->data)  
                if (p->next == p)  
                    Crash  
    return 0;  
}
```

Concrete
State

x=236
p=[634 | NULL]

Symbolic
State

$x = \alpha, p = \beta$
 $p->data = \gamma$
 $p->next = \delta$
 $\alpha > 0 \wedge$
 $\beta \neq \text{NULL}$

2nd iteration

Example: Concolic Testing of Data Structures (Cont'd)

```
typedef struct cell {  
    int data;  
    struct cell *next;  
} cell;
```

```
int foo(int v) { return 2*v + 1; }
```

```
void testme(int x, cell *p) {  
    if (x > 0)  
        if (p != NULL)  
            if (foo(x) == p->data)  
                if (p->next == p)  
                    Crash  
    return 0;  
}
```

Concrete
State

$x=236$
 $p=\boxed{634} \boxed{\text{NULL}}$

2nd iteration

Symbolic
State

$x=a, p=\beta$
 $p->\text{data} = \gamma$
 $p->\text{next} = \delta$
 $a > 0 \wedge$
 $\beta \neq \text{NULL} \wedge$
 $2*a+1 \neq \gamma$

Example: Concolic Testing of Data Structures (Cont'd)

```
typedef struct cell {  
    int data;  
    struct cell *next;  
} cell;  
  
int foo(int v) { return  
  
void testme(int x, cell *p) {  
    if (x > 0)  
        if (p != NULL)  
            if (foo(x) == p->data)  
                if (p->next == p)  
                    Crash  
    return 0;  
}
```

Concrete State	Symbolic State
<ul style="list-style-type: none">- Constraint: $a > 0 \wedge \beta \neq \text{NULL} \wedge 2*a+1 = \gamma$- Solution: $a = 1, \beta = \boxed{3} \boxed{\text{NULL}}$	<p>$x=a, p=\beta$ $p->\text{data} = \gamma$ $p->\text{next} = \delta$</p> <p>$a > 0 \wedge$ $\beta \neq \text{NULL} \wedge$ $2*a+1 \neq \gamma$</p>

Example: Concolic Testing of Data Structures (Cont'd)

```
typedef struct cell {  
    int data;  
    struct cell *next;  
} cell;
```

```
int foo(int v) { return 2*v + 1; }
```

```
void testme(int x, cell *p) {  
    if (x > 0) ←
```

```
        if (p != NULL)  
            if (foo(x) == p->data)  
                if (p->next == p)  
                    Crash
```

```
    return 0;  
}
```

Concrete
State

x=1
p =

3	NULL
---	------

Symbolic
State

$x=\alpha, p=\beta$
 $p->\text{data} = \gamma$
 $p->\text{next} = \delta$
true

3rd iteration

Example: Concolic Testing of Data Structures (Cont'd)

```
typedef struct cell {  
    int data;  
    struct cell *next;  
} cell;
```

```
int foo(int v) { return 2*v + 1; }
```

```
void testme(int x, cell *p) {  
    if (x > 0)  
        if (p != NULL)  
            if (foo(x) == p->data)  
                if (p->next == p)  
                    Crash  
    return 0;  
}
```

3rd iteration

Concrete
State

$x=1$

$p = \boxed{3 | \text{NULL}}$

Symbolic
State

$x=a, p=\beta$
 $p->\text{data} = \gamma$
 $p->\text{next} = \delta$
 $a > 0 \wedge$
 $\beta \neq \text{NULL} \wedge$
 $2*a+1 = \gamma$

Example: Concolic Testing of Data Structures (Cont'd)

```
typedef struct cell {  
    int data;  
    struct cell *next;  
} cell;
```

```
int foo(int v) { return 2*v + 1; }
```

```
void testme(int x, cell *p) {  
    if (x > 0)  
        if (p != NULL)  
            if (foo(x) == p->data)  
                if (p->next == p)  
                    Crash  
    return 0; ←  
}
```

Concrete
State

x=1
p =

3	NULL
---	------

3rd iteration

Symbolic
State

$x=a$, $p=\beta$
 $p->\text{data} = \gamma$
 $p->\text{next} = \delta$
 $a > 0 \wedge$
 $\beta \neq \text{NULL} \wedge$
 $2*a+1 = \gamma \wedge$
 $\delta \neq \beta$

Example: Concolic Testing of Data Structures (Cont'd)

```
typedef struct cell {  
    int data;  
    struct cell *next;  
} cell;
```

```
int foo(int v) { re  
void testme(int x, cell *p) {  
    if (x > 0)  
        if (p != NULL)  
            if (foo(x) == p->data)  
                if (p->next == p)  
                    Crash  
    return 0;  
}
```

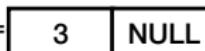
Concrete
State

Symbolic
State

- Constraint: $a > 0 \wedge \beta \neq \text{NULL} \wedge 2*a + 1 = \gamma \wedge \delta = \beta$

- Solution: $a = 1, \beta =$ 

$x=a, p=\beta$
 $p->\text{data} = \gamma$
 $p->\text{next} = \delta$
 $a > 0 \wedge$
 $\beta \neq \text{NULL} \wedge$
 $2*a + 1 = \gamma \wedge$
 $\delta \neq \beta$

$x=1$
 $p =$ 

3rd iteration

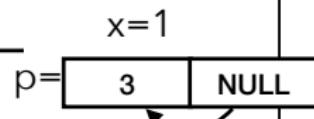
Example: Concolic Testing of Data Structures (Cont'd)

```
typedef struct cell {  
    int data;  
    struct cell *next;  
} cell;
```

```
int foo(int v) { return 2*v + 1; }
```

```
void testme(int x, cell *p) {  
    if (x > 0)  
        if (p != NULL)  
            if (foo(x) == p->data)  
                if (p->next == p)  
                    Crash  
    return 0;  
}
```

Concrete
State



Symbolic
State

$x=a$, $p=\beta$
 $p->\text{data} = \gamma$
 $p->\text{next} = \delta$
true

4th iteration

Example: Concolic Testing of Data Structures (Cont'd)

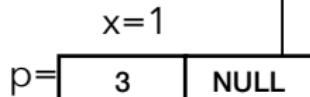
```
typedef struct cell {  
    int data;  
    struct cell *next;  
} cell;
```

```
int foo(int v) { return 2*v + 1; }
```

```
void testme(int x, cell *p) {  
    if (x > 0)  
        if (p != NULL)  
            if (foo(x) == p->data)  
                if (p->next == p)  
                    Crash  
    return 0;  
}
```

4th iteration

Concrete
State



Symbolic
State

$$\begin{aligned}x &= a, p = \beta \\p->\text{data} &= \gamma \\p->\text{next} &= \delta \\a > 0 \wedge \\ \beta \neq \text{NULL} \wedge \\ 2*a+1 &= \gamma \wedge \\ \delta &= \beta\end{aligned}$$

Summary

We explored how concolic testing works in more complex scenarios.

- Concolic testing of loops – there were two notable points.
 - ▶ It is enough to record path constraints that are data-dependant on inputs, because solutions of those constraints decide the next branches to explore.
 - ▶ Negating path conditions does not always increase code coverage.
- Concolic testing of data structures