L3 Mention Informatique
Parcours Informatique et MIAGE

Génie Logiciel Avancé - Advanced Software Engineering

White-Box Tests (Rev)

Burkhart Wolff
wolff@lri.fr

Towards Static Program-based Unit Test

Towards Static Program-based Unit Test

- How can we test during development
  (at coding time, even at design-time ?)

Towards Static Program-based Unit Test

- How can we test during development (at coding time, even at design-time ?)
- How can we test "systematically"?

Towards Static Program-based Unit Test

- How can we test during development
  (at coding time, even at design-time ?)
- How can we test "systematically"?
  - What could be a test-generation method?

Towards Static Program-based Unit Test

- How can we test during development
  (at coding time, even at design-time ?)
- How can we test "systematically"?
  - What could be a test-generation method?
  - What could be an algorithm to generate tests?

Towards Static Program-based Unit Test

- How can we test during development
  (at coding time, even at design-time ?)
- How can we test "systematically"?
  - What could be a test-generation method?
  - What could be an algorithm to generate tests?
  - What could be a coverage criterion ?
    (or: adequacy criterion,
    telling that we "tested enough")

Towards Static Program-based Unit Test

Towards Static Program-based Unit Test

- Let's exploit the structure of the program !!!

  (and not, as before in specification based tests („black box"-tests), depend entirely on the spec).

Towards Static Program-based Unit Test

- Let's exploit the structure of the program !!!

  (and not, as before in specification based tests („black box"-tests), depend entirely on the spec).

- Assumption: Programmers make most likely errors in branching points of a program (Condition, While-Loop, ...), but get the program "in principle right". (Competent programmer assumption)

Towards Static Program-based Unit Test

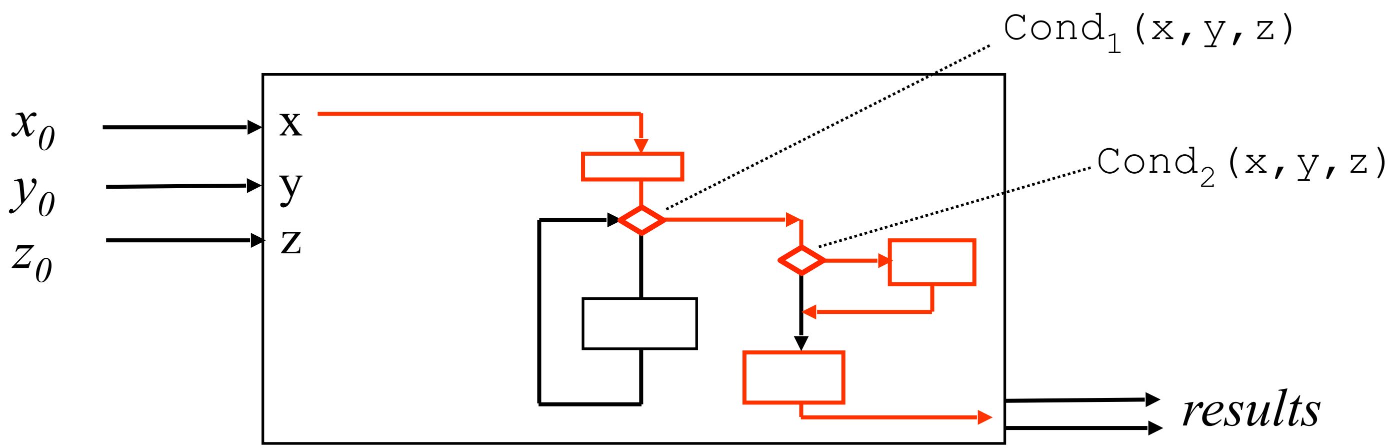
- Let's exploit the structure of the program !!!

  (and not, as before in specification based tests („black box"-tests), depend entirely on the spec).

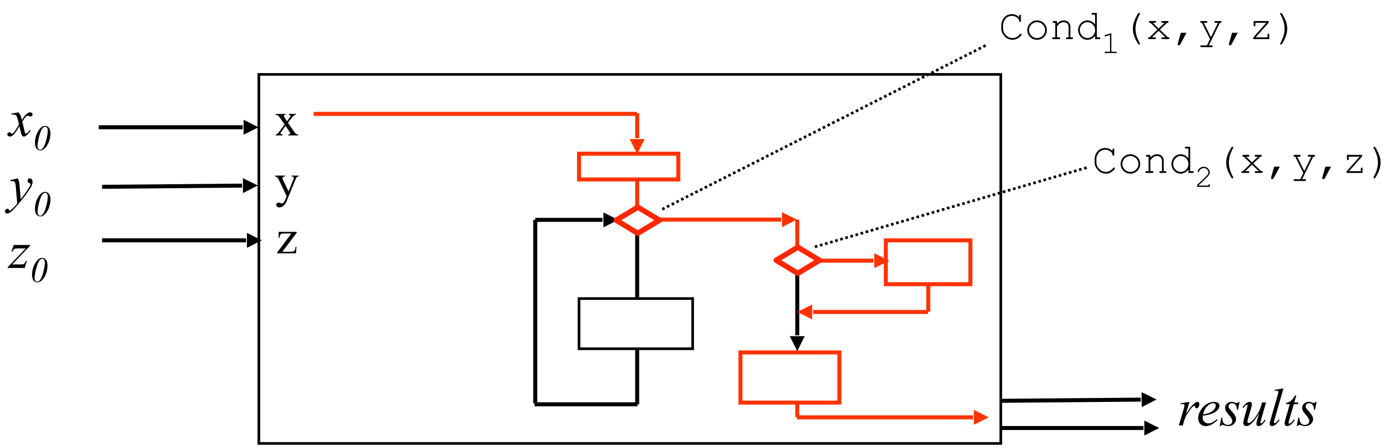
- Assumption: Programmers make most likely errors in branching points of a program (Condition, While-Loop, ...), but get the program "in principle right".
  (Competent programmer assumption)

- Lets develop a test method that exploits this !

Static Structural ("white-box") Tests

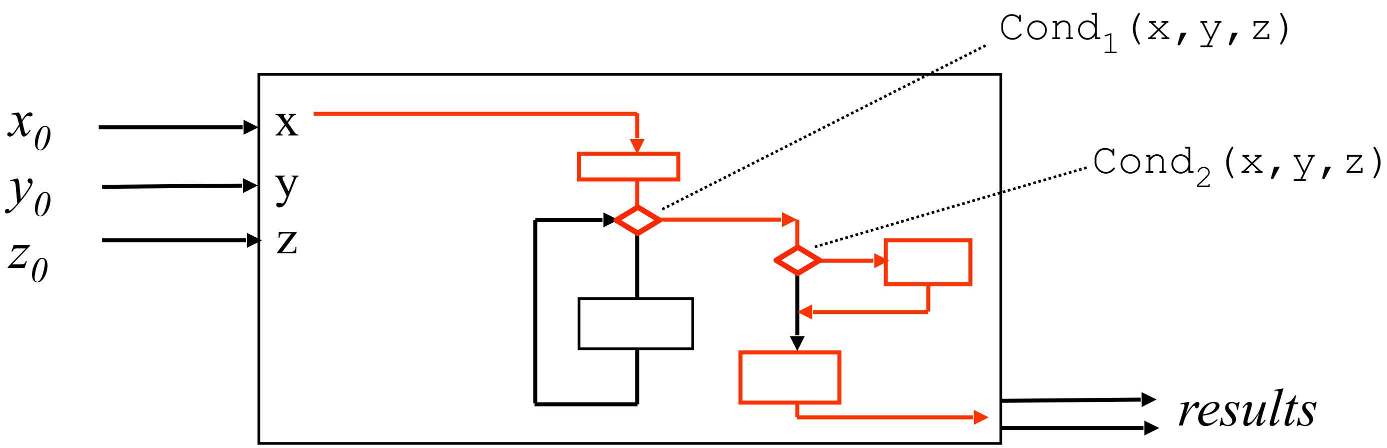


Static Structural ("white-box") Tests



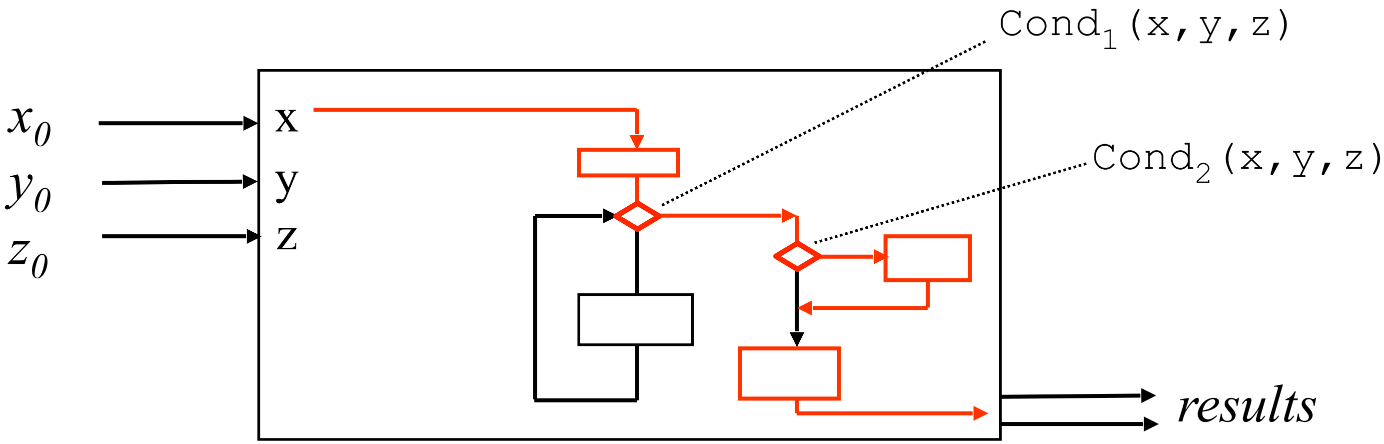
- we select "critical" paths

Static Structural ("white-box") Tests

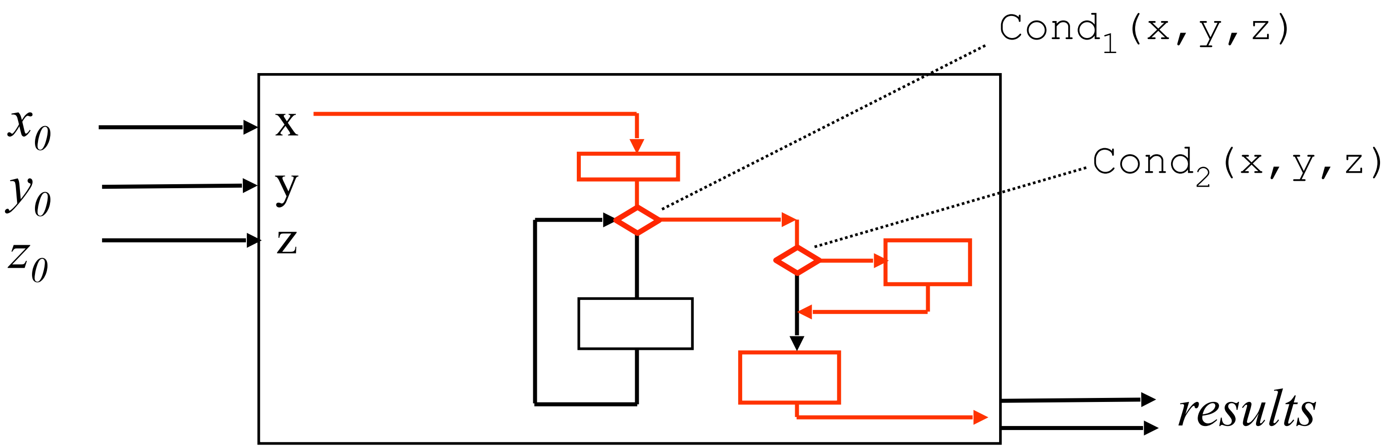


- we select "critical" paths
- specification used to verify the obtained resultants

Static Structural ("white-box") Tests



Static Structural ("white-box") Tests

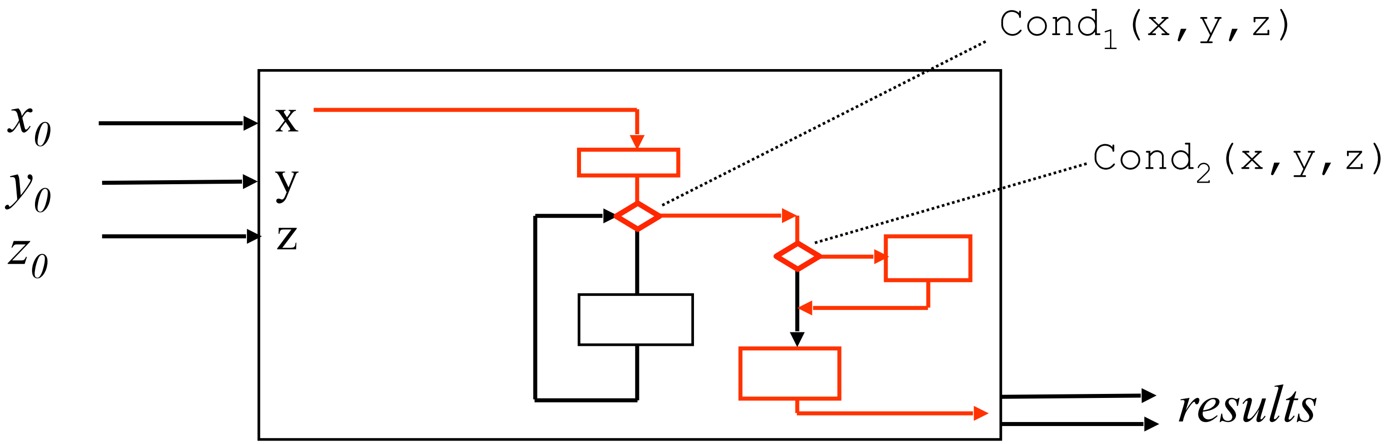


Idea:
a path corresponds to <u>one</u> logical expression over initial values x_0, y_0, z_0 .
corresponding to one test-case (comprising several test data ...)

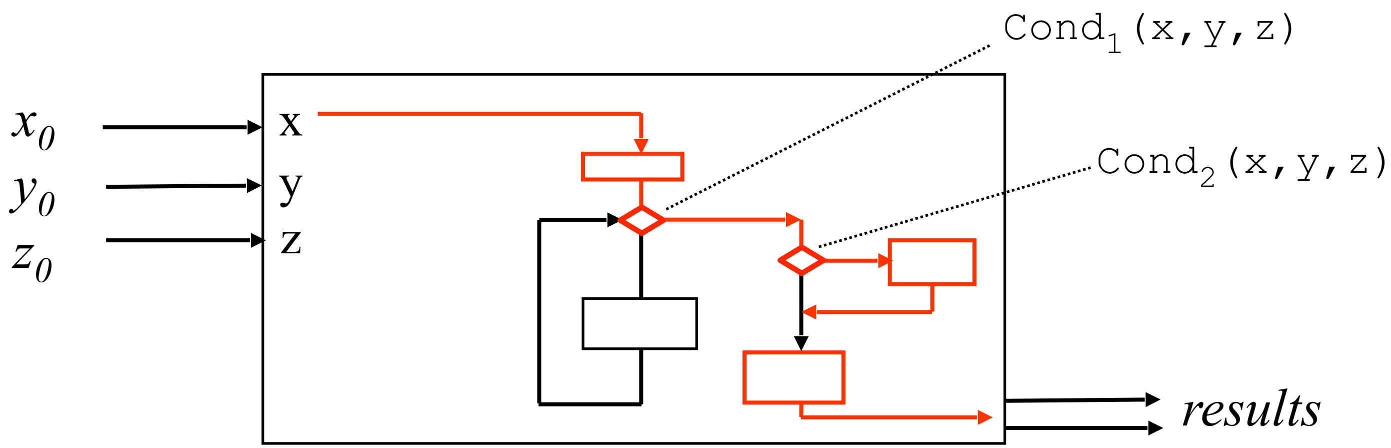
$$\neg\, Cond_1(x_0,\ y_0,\ z_0) \wedge \neg\, Cond_2(x_0,\ y_0,\ z_0)$$

We are interested either in edges (control flow), or in nodes (data flow)

Static Structural ("white-box") Tests



Static Structural ("white-box") Tests



Recall: This path-condition can be effectively constructed by a process called "symbolic execution

The notion of a "coverage criterion"

The notion of a "coverage criterion"

A coverage criterion is a function mapping a CFG
to a particular subset of its paths …

- the set of paths covering all basic blocks
- the set of paths covering all conditions
- the set with all loops are traversed
- a particular subset of calls/labels occurring in the CFG has been covered
- …

Well-known Coverage Criteria I

Criterion C = AllInstructions(CFG):

For all nodes N in CFG (basic instructions or decisions) exists a path in C that contains N

Well-known Coverage Criteria II

Criterion C = AllTransitions(CFG):

For all arcs A in the CFG exists a path in C that uses A

Well-known Coverage Criteria III

Criterion C = AllPaths(CFG):

All possible paths ...

Well-known Coverage Criteria III

Criterion C = AllPaths(CFG):

All possible paths ...

☹ Whenever there is a loop, C is infinite !

Well-known Coverage Criteria III

Criterion C = AllPaths(CFG):

All possible paths ...

☹ Whenever there is a loop, C is infinite !

Well-known Coverage Criteria III

Criterion C = AllPaths(CFG):

All possible paths ...

☹ Whenever there is a loop, C is infinite !

☞ weaker variant: AllPaths_k(CFG).

We limit the paths through a loop to maximally k times …

Well-known Coverage Criteria III

Criterion C = AllPaths(CFG):

All possible paths ...

☹ Whenever there is a loop, C is infinite !

☞ weaker variant: $\text{AllPaths}_k(\text{CFG})$.

We limit the paths through a loop to maximally k times …

☞ we have again a finite number of paths

Well-known Coverage Criteria III

Criterion C = AllPaths(CFG):

All possible paths ...

☹ Whenever there is a loop, C is infinite !

☞ weaker variant: AllPaths_k(CFG).

We limit the paths through a loop to maximally k times …

☞ we have again a finite number of paths

A Hierarchy of Coverage Criteria

- $\text{AllPaths(CFG)} \supseteq$
  $\text{AllPaths}_k\text{(CFG)} \supseteq$
  $\text{AllTransitions(CFG)} \supseteq$
  $\text{AllInstructions(CFG)}$

- Each of these implications reflects a proper containment; the other way round is never true.

Schmankerle

- Program:

```
int f (int a) {
        int i = 0;
        int tm = 1;
        int sum = 1;
        while(sum <= a) {
                i = i+1;
                tm = tm+2;
                sum = tm+sum;
        }
        return i;
}
```

Schmankerle

- Program:

```
int f (int a) {
        int i = 0;
        int tm = 1;
        int sum = 1;
        while(sum <= a) {
                i = i+1;
                tm = tm+2;
                sum = tm+sum;
        }
        return i;
}
```

Specification:

pre : $a \geq 0$

post: $a \leq result^2 \wedge a < (result+1)^2$

Schmankerle

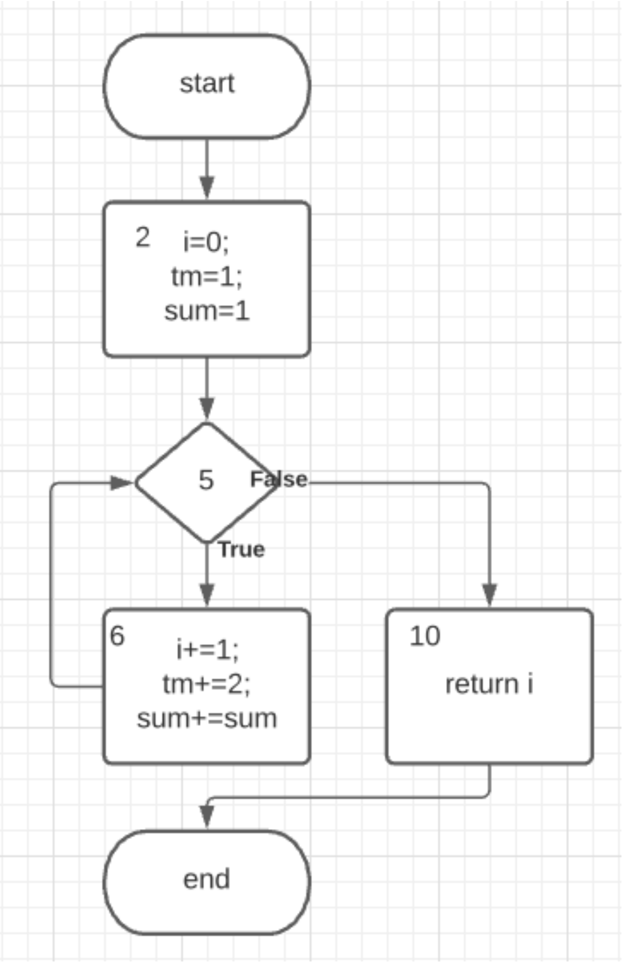
- Program:

```
int f (int a) {
    int i = 0;
    int tm = 1;
    int sum = 1;
    while(sum <= a) {
        i = i+1;
        tm = tm+2;
        sum = tm+sum;
    }
    return i;
}
```

Schmankerle

- Program:

```
int f (int a) {
    int i = 0;
    int tm = 1;
    int sum = 1;
    while(sum <= a) {
        i = i+1;
        tm = tm+2;
        sum = tm+sum;
    }
    return i;
}
```

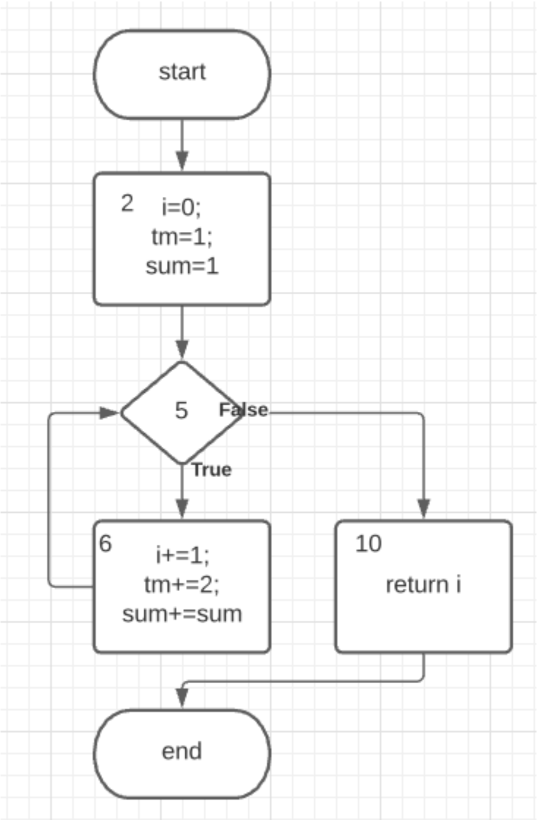


Schmankerle

- CFG de f:

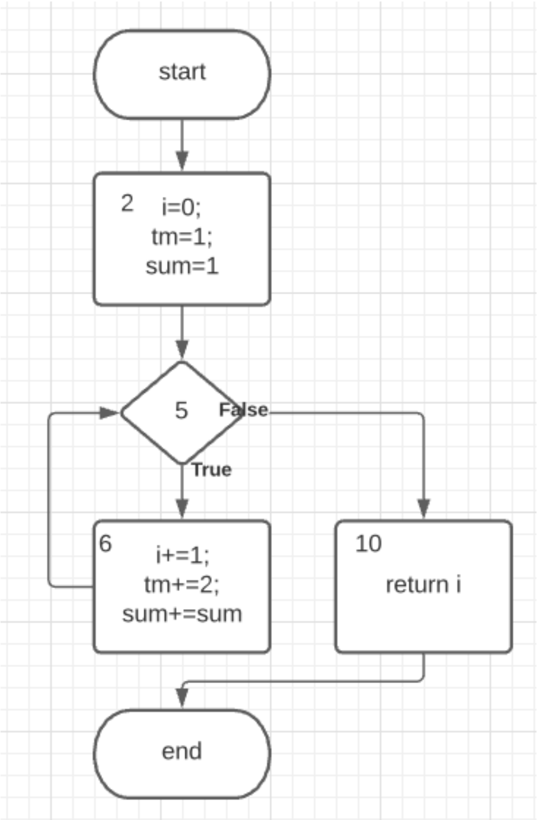
Schmankerle

- CFG de f:



Schmankerle

- CFG de f:



- For example:

AllInstructions(CFG)={[start,2,5,6,5,10,end]}

AllTransitions(CFG)={[start,2,5,6,5,10,end]}

AllPath_3(CFG)={[start,2,5,10,end],
[start,2,5,6,5,10,end],
[start,2,5,6,6,5,10,end],
[start,2,5,6,6,6,5,10,end]}

AllPath(CFG)={ $S \mid \exists\ k \in \mathbb{N}$.

S = [start,2,5,$(6,5)^k$,10,end]}

(infinite !)

Example: A Symbolic Path Execution

We want to execute the path from $AllPath_3$:

[S, 2, 5, 6, 5, 10, E]

Example: A Symbolic Path Execution

We want to execute the path from AllPath$_3$:

[S, 2, 5, 6, 5, 10, E]

$\Phi \mapsto a_0 \geq 0$	$a_0 \geq 0$					
$a \mapsto a_0$	a_0					
$i \mapsto i_0$	0					
$tm \mapsto tm_0$	1					
$sum \mapsto sum_0$	1					

Example: A Symbolic Path Execution

We want to execute the path from AllPath3:

[S, 2, 5, 6, 5, 10, E]

$\Phi \mapsto$ $a_0 \geq 0$	$a_0 \geq 0$	$(sum \leq a)\sigma_2$				
$a \mapsto a_0$	a_0					
$i \mapsto i_0$	0					
$tm \mapsto tm_0$	1					
$sum \mapsto sum_0$	1					

Example: A Symbolic Path Execution

We want to execute the path from $AllPath_3$:

[S, 2, 5, 6, 5, 10, E]

$\Phi \mapsto a_0 \geq 0$	$a_0 \geq 0$	$(sum \leq a)\sigma_2 \wedge a_0 \geq 0$				
$a \mapsto a_0$	a_0					
$i \mapsto i_0$	0					
$tm \mapsto tm_0$	1					
$sum \mapsto sum_0$	1					

Example: A Symbolic Path Execution

We want to execute the path from $AllPath_3$:

[S, 2, 5, 6, 5, 10, E]

$\Phi \mapsto a_0 \geq 0$	$a_0 \geq 0$	$(sum \leq a)\sigma_2 \wedge a_0 \geq 0$				
$a \mapsto a_0$	a_0	a_0				
$i \mapsto i_0$	0	0				
$tm \mapsto tm_0$	1	1				
$sum \mapsto sum_0$	1	1				

Example: A Symbolic Path Execution

We want to execute the path from AllPath$_3$:

[S, 2, 5, 6, 5, 10, E]

$\Phi \mapsto a_0 \geq 0$	$a_0 \geq 0$	$(sum \leq a)\sigma_2 \wedge a_0 \geq 0$	$1 \leq a_0 \wedge$			
$a \mapsto a_0$	a_0	a_0				
$i \mapsto i_0$	0	0				
$tm \mapsto tm_0$	1	1				
$sum \mapsto sum_0$	1	1				

Example: A Symbolic Path Execution

We want to execute the path from AllPath$_3$:

[S, 2, 5, 6, 5, 10, E]

$\Phi \mapsto a_0 \geq 0$	$a_0 \geq 0$	$(sum \leq a)\sigma_2 \wedge a_0 \geq 0$	$1 \leq a_0 \wedge a_0 \geq 0$			
$a \mapsto a_0$	a_0	a_0				
$i \mapsto i_0$	0	0				
$tm \mapsto tm_0$	1	1				
$sum \mapsto sum_0$	1	1				

Example: A Symbolic Path Execution

We want to execute the path from $AllPath_3$:

[S, 2, 5, 6, 5, 10, E]

$\Phi \mapsto a_0 \geq 0$	$a_0 \geq 0$	$(sum \leq a)\sigma_2 \wedge a_0 \geq 0$	$1 \leq a_0 \wedge a_0 \geq 0$			
$a \mapsto a_0$	a_0	a_0	a_0			
$i \mapsto i_0$	0	0	1			
$tm \mapsto tm_0$	1	1	3			
$sum \mapsto sum_0$	1	1	4			

Example: A Symbolic Path Execution

We want to execute the path from AllPath$_3$:

[S, 2, 5, 6, 5, 10, E]

$\Phi \mapsto a_0 \geq 0$	$a_0 \geq 0$	$(sum \leq a)\sigma_2 \wedge a_0 \geq 0$	$1 \leq a_0 \wedge a_0 \geq 0$	$1 \leq a_0 \wedge \neg (sum \leq a)\sigma_6$		
$a \mapsto a_0$	a_0	a_0	a_0			
$i \mapsto i_0$	0	0	1			
$tm \mapsto tm_0$	1	1	3			
$sum \mapsto sum_0$	1	1	4			

Example: A Symbolic Path Execution

We want to execute the path from $AllPath_3$:

[S, 2, 5, 6, 5, 10, E]

$\Phi \mapsto a_0 \geq 0$	$a_0 \geq 0$	$(sum \leq a)\sigma_2 \wedge a_0 \geq 0$	$1 \leq a_0 \wedge a_0 \geq 0$	$1 \leq a_0 \wedge \neg (sum \leq a)\sigma_6$		
$a \mapsto a_0$	a_0	a_0	a_0	a_0		
$i \mapsto i_0$	0	0	1	1		
$tm \mapsto tm_0$	1	1	3	3		
$sum \mapsto sum_0$	1	1	4	4		

Example: A Symbolic Path Execution

We want to execute the path from $AllPath_3$:

[S, 2, 5, 6, 5, 10, E]

$\Phi \mapsto a_0 \geq 0$	$a_0 \geq 0$	$(sum \leq a)\sigma_2 \wedge a_0 \geq 0$	$1 \leq a_0 \wedge a_0 \geq 0$	$1 \leq a_0 \wedge \neg (sum \leq a)\sigma_6$	$1 \leq a_0 \wedge 4 > a_0$	
$a \mapsto a_0$	a_0	a_0	a_0	a_0		
$i \mapsto i_0$	0	0	1	1		
$tm \mapsto tm_0$	1	1	3	3		
$sum \mapsto sum_0$	1	1	4	4		

Example: A Symbolic Path Execution

We want to execute the path from AllPath$_3$:

[S, 2, 5, 6, 5, 10, E]

$\Phi \mapsto a_0 \geq 0$	$a_0 \geq 0$	$(sum \leq a)\sigma_2 \wedge a_0 \geq 0$	$1 \leq a_0 \wedge a_0 \geq 0$	$1 \leq a_0 \wedge \neg(sum \leq a)\sigma_6$	$1 \leq a_0 \wedge 4 > a_0$	
$a \mapsto a_0$	a_0	a_0	a_0	a_0	a_0	
$i \mapsto i_0$	0	0	1	1	1	
$tm \mapsto tm_0$	1	1	3	3	3	
$sum \mapsto sum_0$	1	1	4	4	4	

Example: A Symbolic Path Execution

We want to execute the path from AllPath$_3$:

[S, 2, 5, 6, 5, 10, E]

$\Phi \mapsto a_0 \geq 0$	$a_0 \geq 0$	$(sum \leq a)\sigma_2 \wedge a_0 \geq 0$	$1 \leq a_0 \wedge a_0 \geq 0$	$1 \leq a_0 \wedge \neg(sum \leq a)\sigma_6$	$1 \leq a_0 \wedge 4 > a_0$	$1 \leq a_0 \wedge 4 > a_0 \wedge res = 1$
$a \mapsto a_0$	a_0	a_0	a_0	a_0	a_0	
$i \mapsto i_0$	0	0	1	1	1	
$tm \mapsto tm_0$	1	1	3	3	3	
$sum \mapsto sum_0$	1	1	4	4	4	

Example: A Symbolic Path Execution

We want to execute the path from AllPath3:

[S, 2, 5, 6, 5, 10, E]

$\Phi \mapsto a_0 \geq 0$	$a_0 \geq 0$	$(sum \leq a)\sigma_2 \wedge a_0 \geq 0$	$1 \leq a_0 \wedge a_0 \geq 0$	$1 \leq a_0 \wedge \neg(sum \leq a)\sigma_6$	$1 \leq a_0 \wedge 4 > a_0$	$1 \leq a_0 \wedge 4 > a_0 \wedge res = 1$
$a \mapsto a_0$	a_0	a_0	a_0	a_0	a_0	a_0
$i \mapsto i_0$	0	0	1	1	1	1
$tm \mapsto tm_0$	1	1	3	3	3	3
$sum \mapsto sum_0$	1	1	4	4	4	4

Example: A Symbolic Path Execution

We want to execute the path from AllPath$_3$:

[S, 2, 5, 6, 5, 10, E]

$\Phi \mapsto a_0 \geq 0$	$a_0 \geq 0$	$(sum \leq a)\sigma_2 \land a_0 \geq 0$	$1 \leq a_0 \land a_0 \geq 0$	$1 \leq a_0 \land \neg (sum \leq a)\sigma_6$	$1 \leq a_0 \land 4 > a_0$	$\boxed{1 \leq a_0 \land 4 > a_0 \land res = 1}$
$a \mapsto a_0$	a_0	a_0	a_0	a_0	a_0	a_0
$i \mapsto i_0$	0	0	1	1	1	1
$tm \mapsto tm_0$	1	1	3	3	3	3
$sum \mapsto sum_0$	1	1	4	4	4	4

Example: A Symbolic Path Execution

Result:

Test-Case:

For the path M=[start,2,5,6,5,10,end]

we have the path condition $\Phi \mapsto 1 \leq a_0 \wedge 4 > a_0$

Example: A Symbolic Path Execution

Result:

Test-Case:

For the path M=[start,2,5,6,5,10,end]

we have the path condition $\Phi \mapsto 1 \leq a_0 \wedge 4 > a_0$

A concrete Test, satisfying Φ:

Example: A Symbolic Path Execution

Result:

Test-Case:

For the path M=[start,2,5,6,5,10,end]

we have the path condition $\Phi \mapsto 1 \leq a_0 \wedge 4 > a_0$

A concrete Test, satisfying Φ: $\boxed{a_0 \mapsto 3}$

Example: A Symbolic Path Execution

Result:

Test-Case:

For the path M=[start,2,5,6,5,10,end]

we have the path condition $\Phi \mapsto 1 \leq a_0 \wedge 4 > a_0$

A concrete Test, satisfying Φ: $\boxed{a_0 \mapsto 3}$

Execution of program with this test vector 3:

Example: A Symbolic Path Execution

Result:

Test-Case:

For the path M=[start,2,5,6,5,10,end]

we have the path condition $\Phi \mapsto 1 \leq a_0 \;\wedge\; 4 > a_0$

A concrete Test, satisfying Φ: $\boxed{a_0 \mapsto 3}$

Execution of program with this test vector 3: f(3) = 1

Example: A Symbolic Path Execution

Result:

Test-Case:

For the path M=[start,2,5,6,5,10,end]

we have the path condition $\Phi \mapsto 1 \leq a_0 \wedge 4 > a_0$

A concrete Test, satisfying Φ: $\boxed{a_0 \mapsto 3}$

Execution of program with this test vector 3: f(3) = 1

Verification of the post-condition: post(3, 1)

Example: A Symbolic Path Execution

Result:

Test-Case:

For the path M=[start,2,5,6,5,10,end]

we have the path condition $\Phi \mapsto 1 \leq a_0 \wedge 4 > a_0$

A concrete Test, satisfying Φ: $\boxed{a_0 \mapsto 3}$

Execution of program with this test vector 3: f(3) = 1

Verification of the post-condition: post(3, 1) = true