Cycle Ingénieur – 2^{ème} année Département Informatique

Verification and Validation

Part IV: White-Box Testing

Burkhart Wolff Département Informatique Université Paris-Sud / Orsay



Cycle Ingénieur – 2^{ème} année Département Informatique

Verification and Validation

Part IV: White-Box Testing

Burkhart Wolff Département Informatique Université Paris-Sud / Orsay



Cycle Ingénieur – 2^{ème} année Département Informatique

Verification and Validation

Part IV: White-Box Testing

Burkhart Wolff Département Informatique Université Paris-Sud / Orsay



Cycle Ingénieur – 2^{ème} année Département Informatique

Verification and Validation

Part IV : White-Box Testing

Burkhart Wolff Département Informatique Université Paris-Sud / Orsay

Idea:

Lets 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 !

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

Idea:

Lets 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!

Idea:

Lets 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!

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test

Idea:

Lets exploit the structure of the program !!!

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

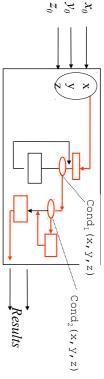
Assumption: Programmers make most likely errors in branching points of a program (Condition, While-Loop, ...), but get the program "in principle right".

(Competent programmer assumption)

Lets develop a test method that exploits this!

Static Structural ("white-box") Tests

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



what the program does and how ...

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

$$\neg Cond_1(x_0, Y_0, z_0) \land \neg Cond_2(x_0, Y_0, z_0)$$

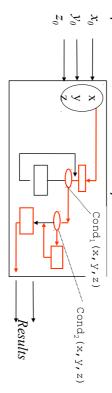
We are interested either in edges (control flow), or in nodes (data flow)

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

Static Structural ("white-box") Tests

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



what the program does and how . .

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

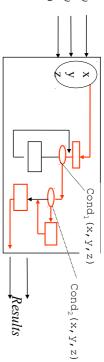
$$\neg Cond_1(x_0, y_0, z_0) \land \neg Cond_2(x_0, y_0, z_0)$$

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

B. Wolff - Inge 2 V&V - White-Box-Test

Static Structural ("white-box") Tests

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



what the program does and how ...

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

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

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

)5/03/18

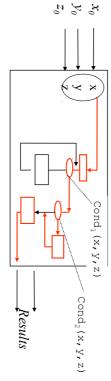
ω

B. Wolff - Inge 2 V&V - White-Box-Test

ω

Static Structural ("white-box") Tests

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



what the program does and how ...

 \underline{A} path corresponds to \underline{one} logical expression over x_0 , y_0 , z_0 . corresponding to one test-case (comprising several test data ...)

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

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

ω

ω

A Program for the triangle example

```
end triangle;
                                                                                                                                                       else if j = k then
                                                                                                                                                                                        if j + k <= 1 or k + 1
                                                                                                                                                                                                                                               procedure triangle(j,k,l : positive) is
                   end if;
                                                                                                                                                                                                                             eg: natural := 0;
                                                                                                                                                                         put("impossible");
                                     end if;
                                                                        elsif eg = 1 then put("isocele");
                                                                                                                if 1 = k then
                                                                                             if eg = 0 then put("arbitrary");
                                                                                                                                  j = 1 then
                                                       put("equilateral");
                                                                                                                                                                                            <u>^</u>
                                                                                                               eg := eg + 1; end if;
                                                                                                                                 eg := eg + 1;
                                                                                                                                                   eg := eg + 1;
                                                                                                                                                                                         ٠.
                                                                                                                                                                                          or _ +
                                                                                                                                                                                         ٠.
                                                                                                                                                                                            <u>^</u>
                                                                                                                                     end if;
                                                                                                                                                       end if;
                                                                                                                                                                                            k then
```

)5/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

4

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

4

A Program for the triangle example

```
end triangle;
                   end if;
                                                                                                                                                         else if j = k then
                                                                                                                                                                                            if j + k <= 1 or k + 1
                                                                                                                                                                                                                                                       procedure triangle(j,k,l : positive) is
                                                                                                                                                                                                                                   eg: natural := 0;
                                                                                                                                                                              put("impossible");
                                                                           elsif eg = 1 then put("isocele");
                                                                                                if eg = 0 then put("arbitrary");
                                                                                                                    if 1 = k then
                                                                                                                                      j = 1 then
                                                          put("equilateral");
                                                                                                                                                                                                  ^
                                                                                                                   eg := eg + 1; end if;
                                                                                                                                    eg := eg + 1;
                                                                                                                                                        eg := eg + 1;
                                                                                                                                                                                              ٠.
                                                                                                                                                                                               or 1 +
                                                                                                                                                                                             j \le k then
                                                                                                                                         end if;
                                                                                                                                                           end if;
```

05/03/1

B. Wolff - Inge 2 V&V - White-Box-Test

A Program for the triangle example

```
else if j = k then
                                                                                                                                                                                        if j + k <= 1 or k + 1
                                                                                                                                                                                                                                                procedure triangle(j,k,l : positive) is
end triangle;
                                                                                                                                                                                                                              eg: natural := 0;
                                                                                                                                                                         put("impossible");
                                    end if;
                                                                                                                                  if j = 1 then
                                                                                                                if 1 = k then
                                                                        elsif eg = 1 then put("isocele");
                                                                                            if eg = 0 then put("arbitrary");
                                                       put("equilateral");
                                                                                                                                                                                             Â
                                                                                                               eg := eg + 1;
                                                                                                                                 eg := eg + 1;
                                                                                                                                                                                          еg
                                                                                                                                     end if;
                                                                                                                                                       end if;
                                                                                                                                                                                           k then
```

A Program for the triangle example

```
else if j = k then
                                                                                                                                                                                                if j + k <= 1 or k + 1
                                                                                                                                                                                                                                                            procedure triangle(j,k,l : positive) is
end triangle;
                                                                                                                                                                                                                                        eg: natural := 0;
                                                                                                                                                                                 put("impossible");
                                       end if;
                                                                                                                                         if j = 1 then
                                                                                                                      if 1 = k then
                                                                             elsif eg = 1 then put("isocele");
                                                                                                 if eg = 0 then put("arbitrary");
                                                           put("equilateral");
                                                                                                                                                                                                      Â
                                                                                                                     eg := eg + 1;
                                                                                                                                        eg := eg + 1;
                                                                                                                                                            eg := eg
                                                                                                                                                                                                   or | +
                                                                                                                                           end if;
                                                                                                                                                              end if;
                                                                                                                                                                                                    k then
```

What are tests adapted to this program?

- try a certain number of execution "paths" (which ones? all of them?)
- find input values to stimulate these paths
- compare the results with expected values (i.e. the specification)

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

Ф

What are tests adapted to this program?

- try a certain number of execution "paths" (which ones ? all of them ?)
- find input values to stimulate these paths
- compare the results with expected values (i.e. the specification)

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

ъ

What are tests adapted to this program?

- try a certain number of execution "paths" (which ones ? all of them ?)
- find input values to stimulate these paths
- compare the results with expected values (i.e. the specification)

What are tests adapted to this program?

- try a certain number of execution "paths" (which ones? all of them?)
- find input values to stimulate these paths

 compare the results with expected values (i.e. the specification)

Ф

Functional-test vs. structural test?

Both are complementary and complete each other:

- Structural Tests have weaknesses in principle:
- if you forget a condition, the specification will most likely reveal this!
- a chance to find this! (Example: perm generator with 3 loops) if your algorithm is incomplete, a test on the spec has at least
- Structural Tests have weaknesses in principle: implementations (working more or less differently from the spec): for a given specification, there are several possible
- sorted arrays : linear search ? binary search ?
- $(x, n) \rightarrow x^n$: successive multiplication ? quadratic multiplication ?

Each implementation demands for different test sets!

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

Functional-test vs. structural test?

Both are complementary and complete each other:

- Structural Tests have weaknesses in principle:
- if you forget a condition, the specification will most likely reveal this!
- a chance to find this! (Example: perm generator with 3 loops) if your algorithm is incomplete, a test on the spec has at least
- Structural Tests have weaknesses in principle: implementations (working more or less differently from the spec): for a given specification, there are several possible
- sorted arrays : linear search ? binary search ?
- $(x, n) \rightarrow x^n$: successive multiplication ? quadratic multiplication ?

Each implementation demands for different test sets!

B. Wolff - Inge 2 V&V - White-Box-Test

Functional-test vs. structural test?

Both are complementary and complete each other:

- Structural Tests have weaknesses in principle:
- if you forget a condition, the specification will most likely reveal this!
- if your algorithm is incomplete, a test on the spec has at least a chance to find this! (Example: perm generator with 3 loops)
- Structural Tests have weaknesses in principle: implementations (working more or less differently from the spec): for a given specification, there are several possible
- sorted arrays : linear search ? binary search ?
- $(x, n) \rightarrow x^n$: successive multiplication ? quadratic multiplication ?

Each implementation demands for different test sets!

05/03/18

6

B. Wolff - Inge 2 V&V - White-Box-Test

6

Functional-test vs. structural test?

Both are complementary and complete each other:

Structural Tests have weaknesses in principle:

- if you forget a condition, the specification will most likely reveal this!
- if your algorithm is incomplete, a test on the spec has at least a chance to find this! (Example: perm generator with 3 loops)
- Structural Tests have weaknesses in principle: implementations (working more or less differently from the spec): for a given specification, there are several possible
- sorted arrays : linear search ? binary search ?
- $(x, n) \rightarrow x^n$: successive multiplication ? quadratic multiplication ?

Each implementation demands for different test sets!

B. Wolff - Inge 2 V&V - White-Box-Test

6

6

05/03/18

Equivalent programs ...

```
Program 1:
    S:=1; P:=N;
    while P >= 1 loop S:= S*X; P:= P-1; end loop;

Program 2:
    S:=1; P:= N;
    while P >= 1 loop
    if P mod 2 /= 0 then P := P -1; S := S*X; end if;
    S:= S*S; P := P div 2;
end loop;
```

Both programs satisfy the same spec but ...

- one is more efficient, but more difficult to test.
- test sets for one are not necessarily "good" for the other, too!

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test

Equivalent programs ...

```
Program 1:
    S:=1; P:=N;
    while P >= 1 loop S:= S*X; P:= P-1; end loop;

Program 2:
    S:=1; P:= N;
    while P >= 1 loop
    if P mod 2 /= 0 then P := P -1; S := S*X; end if;
    S:= S*S; P := P div 2;
end loop;
```

Both programs satisfy the same spec but ...

- one is more efficient, but more difficult to test.
- test sets for one are not necessarily "good" for the other, too!

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test

Equivalent programs ...

```
Program 1 :
    S:=1; P:=N;
while P >= 1 loop S:= S*X; P:= P-1; end loop;

Program 2 :
    S:=1; P:= N;
while P >= 1 loop
    if P mod 2 /= 0 then P := P -1; S := S*X; end if;
    S:= S*S; P := P div 2;
end loop;
```

Both programs satisfy the same spec but ...

- one is more efficient, but more difficult to test.
- test sets for one are not necessarily "good" for the other, too!

B. Wolff - Inge 2 V&V - White-Box-Test

05/03/18

Equivalent programs ...

```
Program 1:
    S:=1; P:=N;
    while P >= 1 loop S:= S*X; P:= P-1; end loop;

Program 2:
    S:=1; P:= N;
    while P >= 1 loop
    if P mod 2 /= 0 then P := P -1; S := S*X; end if;
    S:= S*S; P := P div 2;
end loop;
```

Both programs satisfy the same spec but ...

- one is more efficient, but more difficult to test.
- test sets for one are not necessarily "good" for the other, too!

Control Flow Graphs

A graph with oriented edges root E and an exit S,

- or "decision nodes" labelled by a predicate. the <u>nodes</u> be either "elementary instruction blocs'
- the arcs indicate the control flow between the elementary instruction blocs and decision nodes (control flow)
- all blocs of predicates are accessible from E and lead to S (otherwise, dead code is to be supressed!)

elementary instruction blocs: a sequence of

- assignments
- update operations (on arrays, ..., not discussed here)
- procedure calls (not discussed here !!!)
- conditions and expressions are assumed to be side-effect free

B. Wolff - Inge 2 V&V - White-Box-Test

∞

Control Flow Graphs

A graph with oriented edges root E and an exit S,

- the arcs indicate the control flow between the or "decision nodes" labelled by a predicate. the nodes be either "elementary instruction blocs'
- elementary instruction blocs and decision nodes (control flow)
- all blocs of predicates are accessible from E and lead to S (otherwise, dead code is to be supressed!)

elementary instruction blocs: a sequence of

- assignments
- update operations (on arrays, ..., not discussed here)
- procedure calls (not discussed here !!!)
- conditions and expressions are assumed

to be side-effect free

B. Wolff - Inge 2 V&V - White-Box-Test

Control Flow Graphs

A graph with oriented edges root E and an exit S,

- or "decision nodes" labelled by a predicate. the <u>nodes</u> be either "elementary instruction blocs'
- the arcs indicate the control flow between the elementary instruction blocs and decision nodes (control flow)
- all blocs of predicates are accessible from E and lead to S (otherwise, dead code is to be supressed!)

elementary instruction blocs: a sequence of

- assignments
- update operations (on arrays, ..., not discussed here)
- procedure calls (not discussed here !!!)
- conditions and expressions are assumed to be side-effect free

B. Wolff - Inge 2 V&V - White-Box-Test

∞

Control Flow Graphs

A graph with oriented edges root E and an exit S,

- the <u>nodes</u> be either "elementary instruction blocs" or "decision nodes" labelled by a predicate.
- elementary instruction blocs and decision nodes (control flow) the arcs indicate the control flow between the
- all blocs of predicates are accessible from E and lead to S (otherwise, dead code is to be supressed!)

elementary instruction blocs: a sequence of

- assignments
- update operations (on arrays, ..., not discussed here)
- procedure calls (not discussed here !!!)
- conditions and expressions are assumed to be side-effect free

∞

B. Wolff - Inge 2 V&V - White-Box-Test

Identify longest sequences of assignments

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

9

Computing Control Flow Graphs

Identify longest sequences of assignments

Computing Control Flow Graphs

Identify longest sequences of assignments

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

9

Computing Control Flow Graphs

Identify longest sequences of assignments

Identify longest sequences of assignments

Example:

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

10

Computing Control Flow Graphs

Identify longest sequences of assignments

Example:

B. Wolff - Inge 2 V&V - White-Box-Test

05/03/18

Computing Control Flow Graphs

Identify longest sequences of assignments

Example:

```
S:=1;
P:=N;
end loop;
                 loop S:= S*X;
                             while P >= 1
         P:= P-1;
```

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

10

Computing Control Flow Graphs

Identify longest sequences of assignments

Example:

S := 1;P := N;

```
end loop;
                     loop S:= S*X;
                                  while P >= 1
           P:= P-1;
```

Identify longest sequences of assignments

Example:

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

11

Computing Control Flow Graphs

Identify longest sequences of assignments

Example:

end loop;

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

11

05/03/18

Computing Control Flow Graphs

Identify longest sequences of assignments

Example:

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

11

Computing Control Flow Graphs

Identify longest sequences of assignments

Example:

P:=N;

S := 1;

- Identify longest sequences of assignments
- Erase if_then_elses by branching

0	
5	
\geq	
0	
ω	
-	
<u> </u>	
∞	

B. Wolff - Inge 2 V&V - White-Box-Test

12

Computing Control Flow Graphs

- Identify longest sequences of assignments
- Erase if_then_elses by branching

Computing Control Flow Graphs

- Identify longest sequences of assignments
- Erase if_then_elses by branching

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

12

Computing Control Flow Graphs

- Identify longest sequences of assignments
- Erase if_then_elses by branching

05/03/18

- Identify longest sequences of assignments
- Erase if_then_elses by branching
- Erase while_loops by loop-arc, entry-arc, exit-arc

03/18

B. Wolff - Inge 2 V&V - White-Box-Test

13

05/03/18

Computing Control Flow Graphs

- Identify longest sequences of assignments
- Erase if_then_elses by branching
- Erase while_loops by loop-arc, entry-arc, exit-arc

Computing Control Flow Graphs

- Identify longest sequences of assignments
- Erase if_then_elses by branching

Erase while_loops by loop-arc, entry-arc, exit-arc

B. Wolff - Inge 2 V&V - White-Box-Test

13

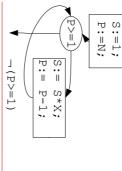
Computing Control Flow Graphs

- Identify longest sequences of assignments
- Erase if_then_elses by branching
- Erase while_loops by loop-arc, entry-arc, exit-arc

05/03/18

- Identify longest sequences of assignments
- Erase if_then_elses by branching
- Erase while_loops by loop-arc, entry-arc, exit-arc

Example:



B. Wolff - Inge 2 V&V - White-Box-Test

05/03/18

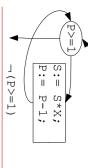
14

Computing Control Flow Graphs

- Identify longest sequences of assignments
- Erase if_then_elses by branching
- Erase while_loops by loop-arc, entry-arc, exit-arc

Example:

S:=1;P:=N;



B. Wolff - Inge 2 V&V - White-Box-Test

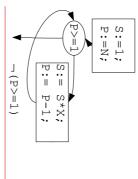
05/03/18

Computing Control Flow Graphs

- Identify longest sequences of assignments
- Erase if_then_elses by branching

Erase while_loops by loop-arc, entry-arc, exit-arc

Example:



05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

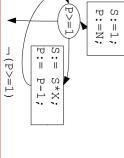
14

Computing Control Flow Graphs

- Identify longest sequences of assignments
- Erase if_then_elses by branching

Erase while_loops by loop-arc, entry-arc, exit-arc

Example:

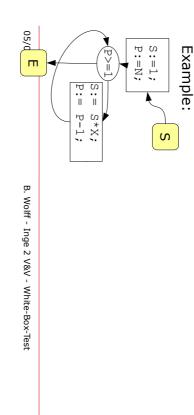


05/03/18

14

B. Wolff - Inge 2 V&V - White-Box-Test

- Identify longest sequences of assignments
- Erase if_then_elses by branching
- Erase while_loops by loop-arc, entry-arc, exit-arc



Computing Control Flow Graphs

15

15

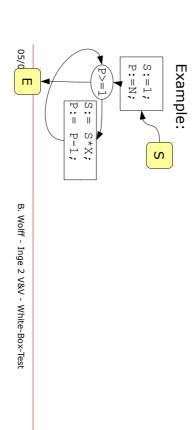
- Identify longest sequences of assignments
- Erase if_then_elses by branching
- Erase while_loops by loop-arc, entry-arc, exit-arc

05/d S:=1; P:=N; Example: S:= S*X; P:= P-1; B. Wolff - Inge 2 V&V - White-Box-Test

Computing Control Flow Graphs

- Identify longest sequences of assignments
- Erase if_then_elses by branching

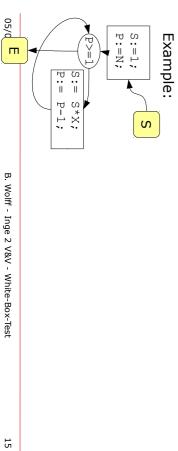
Erase while_loops by loop-arc, entry-arc, exit-arc



Computing Control Flow Graphs

- Identify longest sequences of assignments
- Erase if_then_elses by branching

Erase while_loops by loop-arc, entry-arc, exit-arc



- Identify longest sequences of assignments
- Erase if_then_elses by branching
- Erase while_loops by loops
- Add entry node and exit loop-arc, entry-arc, exit-arc

A Control-Flow-Graph (CFG) is usually a by-product of a compiler ...

)5/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

16

Computing Control Flow Graphs

- Identify longest sequences of assignments
- Erase if_then_elses by branching
- Erase while_loops by loops
- Add entry node and exit loop-arc, entry-arc, exit-arc

A Control-Flow-Graph (CFG) is usually a by-product of a compiler ...

Computing Control Flow Graphs

- Identify longest sequences of assignments
- Erase if_then_elses by branching
- Erase while_loops by loops
- Add entry node and exit loop-arc, entry-arc, exit-arc

A Control-Flow-Graph (CFG) is usually a by-product of a compiler ...

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

16

Computing Control Flow Graphs

- Identify longest sequences of assignments
- Erase if_then_elses by branching
- Erase while_loops by loops
- Add entry node and exit loop-arc, entry-arc, exit-arc

A Control-Flow-Graph (CFG) is usually a by-product of a compiler ...

05/03/18

16

B. Wolff - Inge 2 V&V - White-Box-Test

Q: What is the CFG of the body of triangle?

Q: What is the CFG of the body of triangle?

Q: What is the CFG of the body of triangle?

Q: What is the CFG of the body of triangle?

Revisiting our triangle example ...

```
end triangle;
                                                                                                                                                     else if j = k then
                                                                                                                                                                                     if j + k <= 1 or k + 1
                                                                                                                                                                                                         begin
                                                                                                                                                                                                                                             procedure triangle(j,k,l : positive) is
                   end if;
                                                                                                                                                                                                                          eg: natural := 0;
                                                                                                                                                                      put("impossible");
                                    end if;
                                                                       elsif eg = 1 then put("isocele");
                                                                                           if eg = 0 then put("quelconque");
                                                                                                                if 1 = k then
                                                                                                                                j = 1 then
                                                      put("equilateral");
                                                                                                                                                                                         <u>^</u>
                                                                                                             eg := eg + 1; end if;
                                                                                                                               eg := eg + 1;
                                                                                                                                                 eg := eg + 1;
                                                                                                                                                                                      U.
                                                                                                                                                                                       or | +
                                                                                                                                                                                      ٠.
                                                                                                                                                                                         <u>^</u>
                                                                                                                                   end if;
                                                                                                                                                     end if;
                                                                                                                                                                                         k then
```

B. Wolff - Inge 2 V&V - White-Box-Test

18

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

18

Revisiting our triangle example ...

```
end triangle;
                    end if;
                                                                                                                                                            else if j = k then
                                                                                                                                                                                                if j + k <= 1 or k + 1
                                                                                                                                                                                                                      begin
                                                                                                                                                                                                                                                            procedure triangle(j,k,l : positive) is
                                                                                                                                                                                                                                        eg: natural := 0;
                                                                                                                                                                                 put("impossible");
                                                                             elsif eg = 1 then put("isocele");
                                                                                                  if eg = 0 then put("quelconque");
                                                                                                                      if 1 = k then
                                                                                                                                         j = 1 then
                                                           put("equilateral");
                                                                                                                                                                                                      ^
                                                                                                                     eg := eg + 1; end if;
                                                                                                                                       eg := eg + 1; end if;
                                                                                                                                                           eg := eg + 1;
                                                                                                                                                                                                  ٠.
                                                                                                                                                                                                   or 1 +
                                                                                                                                                                                                 j \le k then
                                                                                                                                                              end if;
```

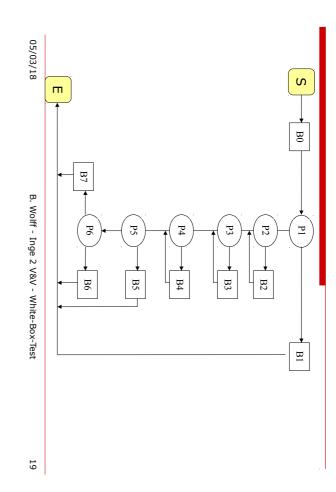
Revisiting our triangle example ...

```
else if j = k then
                                                                                                                                                                                if j + k <= 1 or k + 1
                                                                                                                                                                                                                                      procedure triangle(j,k,l : positive) is
end triangle;
                                                                                                                                                                                                                    eg: natural := 0;
                                                                                                                                                                  put("impossible");
                                                                                                                             if j = 1 then
                                  end if;
                                                                                                            if 1 = k then
                                                       else
                                                                     elsif eg = 1 then put("isocele");
                                                                                         if eg = 0 then put("quelconque");
                                                     put("equilateral");
                                                                                                                                                                                     Â
                                                                                                                                             eg :=
                                                                                                                           eg := eg + 1;
                                                                                                           eg := eg + 1; end if;
                                                                                                                                                                                   or | +
                                                                                                                                               eg
                                                                                                                                                 + 1;
                                                                                                                               end if;
                                                                                                                                                 end if;
                                                                                                                                                                                    k then
```

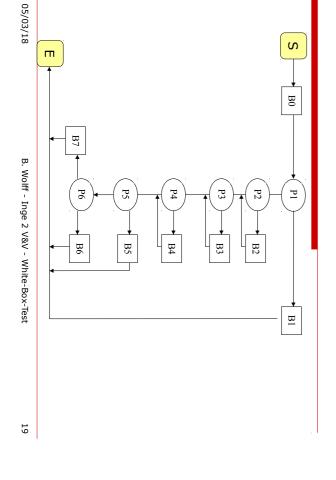
Revisiting our triangle example ...

```
else if j = k then
                                                                                                                                                                                   if j + k <= 1 or k + 1
                                                                                                                                                                                                                                           procedure triangle(j,k,l : positive) is
end triangle;
                                                                                                                                                                                                                        eg: natural := 0;
                                                                                                                                                                     put("impossible");
                                     end if;
                                                                                                                                if j = 1 then
                                                                                                              if 1 = k then
                                                        else
                                                                         elsif eg = 1 then put("isocele");
                                                                                           if eg = 0 then put("quelconque");
                                                       put("equilateral");
                                                                                                                                                                                        Â
                                                                                                            eg := eg + 1; end if;
                                                                                                                              eg := eg + 1;
                                                                                                                                                  eg :=
                                                                                                                                                                                      or | +
                                                                                                                                                  eg
                                                                                                                                                   + 1;
                                                                                                                                                    end if;
                                                                                                                                  end if;
                                                                                                                                                                                       k then
```

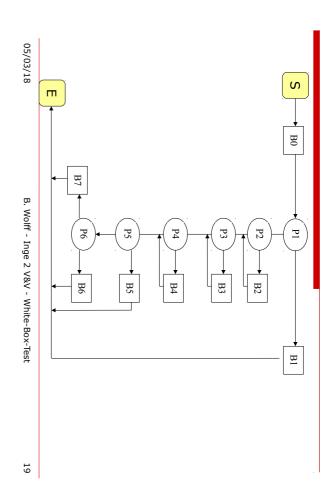
The non-structured control-flow graph of a program



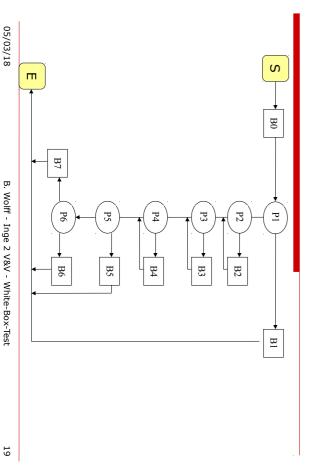
The non-structured control-flow graph of a program



The non-structured control-flow graph of a program



The non-structured control-flow graph of a program



A procedure with loop and return

```
end supprime;
                                                                                                                                                                                             procedure supprime
                                                                                                                                                             i: integer := 1;
                     end loop;
                                                                                                                       while
                                                                                                     j.
                                        end if;
                                                                                 elsif
                                                             else
                                                           T[i] := T[p-1]; p := p -1;
                                                                               i = p - 1
                                                                                                   T[i].val \Leftrightarrow x  then i := i
                                                                                                                        \Diamond
                                                                                                                       D
                                                                                                                                                                                              (T: in out Table; p: in out integer;
                                                                                                                       loop
                                                                                                                                                                                in integer) is
                                                                                 then
                                                                                d
≕. d
                                                                               - 1; return;
                                                                                                    + 1;
                                                             return;
```

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

20

A procedure with loop and return

```
end supprime;
                                                                                                                                                                                                                procedure supprime (T: in out Table; p: in out integer;
                                                                                                                                                                          i: integer := 1;
                      end loop;
                                                                                                                                 while
                                                                                                             j.
                                           end if;
                                                                                      elsif
                                                                 else
                                                               T[i] := T[p-1]; p := p -1;
                                                                                     i = p - 1
                                                                                                           T[i].val \leftrightarrow x then i := i
                                                                                                                                  \Diamond
                                                                                                                                 Q
                                                                                                                                 loop
                                                                                                                                                                                                in integer) is
                                                                                     then p := p
                                                                                                            + 1;
                                                                                      - 1; return;
                                                                 return;
```

A procedure with loop and return

```
end supprime;
                                                                                                                                                                                     procedure supprime
                                                                                                                                   begin
                                                                                                                                                     i: integer := 1;
                                                                                                               while
                   end loop;
                                    end if;
                                                                                               Ħ.
                                                                            elsif
                                                        else
                                                       T[i] := T[p-1]; p := p -1;
                                                                         - d
= т
                                                                                             T[i].val \leftrightarrow x then i := i
                                                                                                                 \Diamond
                                                                                                                ם
                                                                                                                                                                      (T: in out Table; p: in out integer;
x: in integer) is
                                                                                                                 loop
                                                                          then p :=
                                                                           Q
                                                                                               + 1;
                                                                              I
                                                        return;
                                                                           1; return;
```

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

20

A procedure with loop and return

```
end supprime;
                                                                                                  begin
                                                                                                              i: integer := 1;
                                                                                   while
              end loop;
                           end if;
                                                                       j.
E
                                                       elsif
                                          else
                                        T[i] := T[p-1]; p := p -1;
                                                      - d
т
                                                                     T[i].val \leftrightarrow x then
                                                                                    \Diamond
                                                                                   Q
                                                                                   loop
                                                      then p := p
                                                                       1.
= 1.
                                                                       +
                                                         1
                                          return;
                                                       1; return;
```

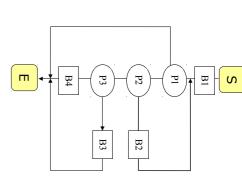
20

B. Wolff - Inge 2 V&V - White-Box-Test

... and its control flow graph

What are the feasible paths?

How to describe this?



05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

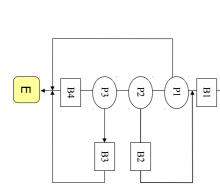
21

... and its control flow graph

S

What are the feasible paths?

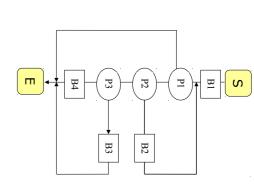
How to describe this?



... and its control flow graph

What are the feasible paths?

How to describe this?



05/03/18

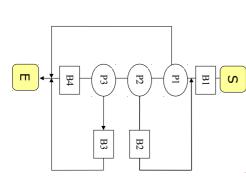
B. Wolff - Inge 2 V&V - White-Box-Test

21

... and its control flow graph

What are the feasible paths?

How to describe this?



Paths and Path Conditions

- Let M a procedure to test, and G its control-flow graph. Terminology:
- sub-path of M = path of G
- initial path of M = path of G starting at S
- path of M = path of G starting at S and leading to E

i.e. a complete execution of the procedure

- a given path is associated to <u>predicate</u> (over parameters and state): a condition over the **initial values initiales** of parameters (and global variables) to achieve exactly this execution path
- variables exists such that the path is executable. <u>faisable paths</u> = a path of M pour a set for all parameters and global

i.e. the path condition is satisfiable

B. Wolff - Inge 2 V&V - White-Box-Test

22

Paths and Path Conditions

- Let M a procedure to test, and G its control-flow graph. Terminology:
- sub-path of M = path of G
- initial path of M = path of G starting at S
- path of M = path of G starting at S and leading to E

i.e. a complete execution of the procedure

a given path is associated to <u>predicate</u> (over parameters and state): a condition over the **initial values initiales** of parameters (and global variables) to achieve exactly this execution path

variables exists such that the path is executable. <u>faisable paths</u> = a path of M pour a set for all parameters and global

i.e. the path condition is satisfiable

Paths and Path Conditions

- Terminology: Let M a procedure to test, and G its control-flow graph.
- sub-path of M = path of G
- initial path of M = path of G starting at S
- path of M = path of G starting at S and leading to E

i.e. a complete execution of the procedure

- a condition over the **initial values initiales** of parameters a given path is associated to <u>predicate</u> (over parameters and state): (and global variables) to achieve exactly this execution path
- variables exists such that the path is executable. faisable paths = a path of M pour a set for all parameters and global

i.e. the path condition is satisfiable

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

22

Paths and Path Conditions

- Terminology: Let M a procedure to test, and G its control-flow graph.
- sub-path of M = path of G
- initial path of M = path of G starting at S
- path of M = path of G starting at S and leading to E

i.e. a complete execution of the procedure

- a condition over the **initial values initiales** of parameters a given path is associated to <u>predicate</u> (over parameters and state):
- variables exists such that the path is executable. faisable paths = a path of M pour a set for all parameters and global

(and global variables) to achieve exactly this execution path

i.e. the path condition is satisfiable

Computing Path Conditions by Symbolic Execution

Let P be an initial path in M.

- \triangleright we give symbolic values for each variable $x_0, y_0, z_0, ...$
- we set the path condition Φ initially "true"
- We follow the path, block for block, along P: If the block is an instruction block B:

we execute symbolically B by memorizing the new values by expressions (symbolically) dependent on $\mathbf{x}_0, \mathbf{y}_0, \mathbf{z}_0, \dots$

If the block is a decision block P(x,...,z)

if we follow the « true » arc we set $\Phi := \Phi \land P(\underline{x},...,\underline{z})$, if we follow the «false» arc we set $\Phi := \Phi \land \neg P(\underline{x},...,\underline{z})$. (The $\underline{x},...,\underline{z}$ are the symbolic values for x,...,z. This effect is produced by a substitution to be discussed later.)

05/03/1

B. Wolff - Inge 2 V&V - White-Box-Test

23

Computing Path Conditions by Symbolic Execution

Let P be an initial path in M.

- ightharpoonup we give symbolic values for each variable x_0, y_0, z_0, \dots
- we set the path condition Φ initially "true"
- We follow the path, block for block, along P:

If the block is an instruction block B: we execute symbolically B by memorizing the new values by expressions (symbolically) dependent on x_0, y_0, z_0, \ldots

If the block is a decision block P(x,...,z)

if we follow the « true » arc we set $\Phi:=\Phi \land P(\underline{x},...,\underline{z})$, if we follow the «false» arc we set $\Phi:=\Phi \land \neg P(\underline{x},...,\underline{z})$. (The $\underline{x},...,\underline{z}$ are the symbolic values for x,...,z. This effect is produced by a substitution to be discussed later.)

01/0

B. Wolff - Inge 2 V&V - White-Box-Test

Computing Path Conditions by Symbolic Execution

Let P be an initial path in M.

- we give symbolic values for each variable $x_0, y_0, z_0, ...$
- we set the path condition Φ initially "true"
- We follow the path, block for block, along P:

If the block is an instruction block B:

we execute symbolically B by memorizing the new values by expressions (symbolically) dependent on $\mathbf{x}_{0},Y_{0},\mathbf{z}_{0},\dots$

If the block is a decision block P(x,...,z)

if we follow the « true » arc we set $\Phi := \Phi \land P(\underline{x},...,\underline{z})$, if we follow the «false» arc we set $\Phi := \Phi \land \neg P(\underline{x},...,\underline{z})$. (The $\underline{x},...,\underline{z}$ are the symbolic values for x,...,z. This effect is produced by a substitution to be discussed later.)

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

23

Computing Path Conditions by Symbolic Execution

Let P be an initial path in M.

- we give symbolic values for each variable x_0, y_0, z_0, \dots
- we set the path condition Φ initially "true"
- We follow the path, block for block, along P:

If the block is an instruction block B: we execute symbolically B by memorizing the new values by expressions (symbolically) dependent on $\mathbf{x}_0, \mathbf{y}_0, \mathbf{z}_0, \dots$

If the block is a decision block P(x,...,z)

if we follow the « true » arc we set $\Phi := \Phi \land P(\underline{x},...,\underline{z})$, if we follow the «false» arc we set $\Phi := \Phi \land \neg P(\underline{x},...,\underline{z})$. (The $\underline{x},...,\underline{z}$ are the symbolic values for x,...,z. This effect is produced by a substitution to be

B. Wolff - Inge 2 V&V - White-Box-Test

discussed later.)

Execution

 Execution (in imperative languages) is based on the notion of state.

A state is a table (or: function) that maps a variable V to some value of a domain D.

state =
$$V \rightarrow D$$

As usual, we denote (finite) functions as follows:

$$\{ x \mapsto 1, y \mapsto 5, x \mapsto 12 \}$$

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

24

Execution

 Execution (in imperative languages) is based on the notion of state.

A state is a table (or: function) that maps a variable V to some value of a domain D.

state =
$$V \rightarrow D$$

As usual, we denote (finite) functions as follows:

$$\{ \times \downarrow 1, y \downarrow 5, \times \downarrow 12 \}$$

Execution

Execution (in imperative languages) is based on the notion of *state*.

A state is a table (or: function) that maps a variable V to some value of a domain D

state =
$$V \rightarrow D$$

As usual, we denote (finite) functions as follows:

$$\{ x \mapsto 1, y \mapsto 5, x \mapsto 12 \}$$

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

24

Execution

 Execution (in imperative languages) is based on the notion of state.

A state is a table (or: function) that maps a variable V to some value of a domain D.

state =
$$V \rightarrow D$$

As usual, we denote (finite) functions as follows:

$$\{ \times \rightarrow 1, y \rightarrow 5, \times \rightarrow 12 \}$$

24

B. Wolff - Inge 2 V&V - White-Box-Test

Symbolic Execution

In static program analysis, it is in general not possible to infer concrete values of D.

values However, it can be inferred a set of possible

For example, if we know that

$$x \in \{1..10\}$$

and we have an assignment x := x+2, we know:

$$x \in \{3..12\}$$

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test afterwards

25

Symbolic Execution

In static program analysis, it is in general not possible to infer concrete values of D.

values However, it can be inferred a set of possible

For example, if we know that

$$x \in \{1..10\}$$

and we have an assignment x := x+2, we know:

$$x \in \{3..12\}$$

afterwards

Symbolic Execution

In static program analysis, it is in general not possible to infer concrete values of D.

values. However, it can be inferred a set of possible

For example, if we know that

$$x \in \{1..10\}$$

and we have an assignment x := x+2, we know:

$$x \in \{3..12\}$$
 afterwards

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

25

Symbolic Execution

In static program analysis, it is in general not possible to infer concrete values of D.

values However, it can be inferred a set of possible

For example, if we know that

$$x \in \{1..10\}$$

and we have an assignment x := x+2, we know:

afterwards

25

05/03/18

Symbolic Execution

This gives rise to the notion of a symbolic state.

$$state_{sym} = V \rightarrow Set(D)$$

As usual, we denote sets by

$$\{x \mid E\}$$

where E is a boolean expression. In our concrete technique, sets will always have the form $\{x_0 \mid x_0 = E \}$ where E is an arithmetic expression (possibly containing variables of V).

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

26

Symbolic Execution

This gives rise to the notion of a symbolic state.

$$state_{sym} = V \rightarrow Set(D)$$

As usual, we denote sets by

$$\{x \mid E\}$$

where E is a boolean expression. In our concrete technique, sets will always have the form $\{x_0 \mid x_0 = E \}$ where E is an arithmetic expression (possibly containing variables of V).

Symbolic Execution

This gives rise to the notion of a symbolic state.

$$state_{sym} = V \rightarrow Set(D)$$

As usual, we denote sets by

$$\{x \mid E\}$$

where E is a boolean expression. In our concrete technique, sets will always have the form $\{x_0 \mid x_0 = E \}$ where E is an arithmetic expression (possibly containing variables of V).

/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

2

Symbolic Execution

• This gives rise to the notion of a symbolic state.

$$\mathsf{state}_{\mathsf{sym}} = \ \mathsf{V} \to \mathsf{Set}(\mathsf{D})$$

As usual, we denote sets by

$$\{ \times \mid E \}$$

where E is a boolean expression. In our concrete technique, sets will always have the form $\{x_0 \mid x_0 = E \}$ where E is an arithmetic expression (possibly containing variables of V).

Symbolic States and Substitutions

• Since in our concrete technique, sets have have the form $\{x_0 \mid x_0 = E\}$, we can abbreviate:

$$\{x \mapsto \{x_0 | x_0 = E_1\}, y \mapsto \{y_0 | y_0 = E_2\}, z \mapsto \{z_0 | z_0 = E_3\}\}$$

to $\{x \mapsto E_1, y \mapsto E_2, z \mapsto E_3\}$

and treat them as substitutions - all variables in an expression were subsequently replaced by their substituands ...

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

27

Symbolic States and Substitutions

• Since in our concrete technique, sets have have the form $\{x_0 \mid x_0 = E\}$, we can abbreviate:

$$\{x \mapsto \{x_0 | x_0 = E_1\}, y \mapsto \{y_0 | y_0 = E_2\}, z \mapsto \{z_0 | z_0 = E_3\}\}$$

to $\{x \mapsto E_1, y \mapsto E_2, z \mapsto E_3\}$

and treat them as substitutions - all variables in an expression were subsequently replaced by their substituands ...

Symbolic States and Substitutions

• Since in our concrete technique, sets have have the form $\{x_0 \mid x_0 = E\}$, we can abbreviate:

$$\{x \mapsto \{x_0 | x_0 = E_1\}, y \mapsto \{y_0 | y_0 = E_2\}, z \mapsto \{z_0 | z_0 = E_3\}\}$$

 $\{x \mapsto E_1, y \mapsto E_2, z \mapsto E_3\}$

and treat them as substitutions - all variables in an expression were subsequently replaced by their substituands ...

)5/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

.

Symbolic States and Substitutions

• Since in our concrete technique, sets have have the form $\{x_0 \mid x_0 = E\}$, we can abbreviate:

$$\{x \mapsto \{x_0 | x_0 = E_1\}, y \mapsto \{y_0 | y_0 = E_2\}, z \mapsto \{z_0 | z_0 = E_3\}\}$$

 $\{x \mapsto E_1, y \mapsto E_2, z \mapsto E_3\}$

and treat them as substitutions - all variables in an expression were subsequently replaced by their substituands ...

27

B. Wolff - Inge 2 V&V - White-Box-Test

Symbolic States and Substitutions

$$(x + 2 * y) \{x \rightarrow 1, y \rightarrow x_0\}$$

$$= 1 + 2 * x_0$$

An initial symbolic state is a state of the form:

$$\{ \times \mapsto \times_0, \ y \mapsto y_0, \ z \mapsto z_0 \}$$

B. Wolff - Inge 2 V&V - White-Box-Test

28

Symbolic States and Substitutions

Example substitution:

$$(x + 2 * y) \{x \mapsto 1, y \mapsto x_0\}$$

$$= 1 + 2 * x_0$$

An initial symbolic state is a state of the form:

$$\{ x \mapsto x_0, y \mapsto y_0, z \mapsto z_0 \}$$

Example substitution:

Symbolic States and Substitutions

Example substitution:

$$(x + 2 * y) \{x \mapsto 1, y \mapsto x_0\}$$

$$= 1 + 2 * x_0$$

An initial symbolic state is a state of the form:

$$X \mapsto X_0, Y \mapsto Y_0, Z \mapsto Z_0$$

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

28

Symbolic States and Substitutions

Example substitution:

$$(x + 2 * y) \{x \rightarrow 1, y \rightarrow x_0\}$$

$$= 1 + 2 * x_0$$

An initial symbolic state is a state of the form:

$$x \mapsto x_0, y \mapsto y_0, z \mapsto z_0$$

Basic Blocks as Substitutions

Symbolic Pre-State

$$\begin{array}{c|cccc} x & \downarrow & \chi \\ \hline & \chi & \downarrow & \chi_0 \\ \hline & \chi & \chi & \chi_0 \\ \hline & \chi_0 \\ \hline & \chi & \chi_0 \\ \hline &$$

Block

$$i := x+y+1$$
$$z := z+i$$

$$x \mapsto x_0$$

 $y \mapsto y_0 + 3 * x_0$
 $z \mapsto z_0 + y_0 + 4 * x_0 + 1$
 $z \mapsto y_0 + 4 * x_0 + 1$

Ν

Ţ

Į

Symbolic Post-State

Symbolic Pre-State

Block

Basic Blocks as Substitutions

$$\begin{array}{c} \rightarrow \times_{0} \\ \rightarrow \times_{0} + 3 \times \times_{0} \\ \rightarrow \times_{0} + \times_{0} + 4 \times_{0} + 1 \end{array}$$

 $y \mapsto y_0 + 3 * x_0$ ¥ z := z+i

 $_{1}:=x+y+1$

Symbolic Post-State

¥ $y \mapsto y_0 + 3 * x_0$ \rightarrow y_0 + $4*x_0$ +1 $\rightarrow z_0 + y_0 + 4 \times x_0 + 1$

 $x_{\mbox{\tiny 0}},\,y_{\mbox{\tiny 0}}$ and $z_{\mbox{\tiny 0}}$ represent the initial values of $x,\,y$ et z.

is supposed to be a local variable (not initialized at the beginning).

B. Wolff - Inge 2 V&V - White-Box-Test

05/03/18

29

 x_0 , y_0 and z_0 represent the initial values of x, y et z.

i is supposed to be a local variable (not initialized at the beginning).

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

29

Basic Blocks as Substitutions

Symbolic Pre-State

$$\begin{array}{c|cccc} x & \downarrow & \chi_0 \\ y & \downarrow & \chi_0 \\ z & \downarrow & \chi_0 \\ \vdots & \downarrow & \chi_0 \\ \end{array}$$

Block

$$i := x+y+1$$

$$z := z+i$$

Symbolic Post-State

$$\begin{array}{c} x \mapsto x_0 \\ y \mapsto y_0 + 3 \times x_0 \\ z \mapsto z_0 + y_0 + 4 \times x_0 + 1 \\ \vdots \mapsto y_0 + 4 \times x_0 + 1 \end{array}$$

Symbolic Pre-State

Block

Basic Blocks as Substitutions

$$i := x+y+1$$
$$z := z+i$$

Symbolic Post-State

$$\begin{array}{c} x \mapsto x_0 \\ y \mapsto y_0 + 3 * x_0 \\ z \mapsto z_0 + y_0 + 4 * x_0 + 1 \\ \dot{1} \mapsto y_0 + 4 * x_0 + 1 \end{array}$$

 $\mathbf{x}_{\scriptscriptstyle{0}},\,\mathbf{y}_{\scriptscriptstyle{0}}$ and $\mathbf{z}_{\scriptscriptstyle{0}}$ represent the initial values of $\mathbf{x},\,\mathbf{y}$ et $\mathbf{z}.$

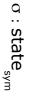
 x_0 , y_0 and z_0 represent the initial values of x, y et z.

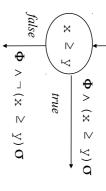
is supposed to be a local variable (not initialized at the beginning).

i is supposed to be a local variable (not initialized at the beginning).

05/03/18

Symbolic Execution





Thus, we execute symbolically and transform the symbolic state in order to obtain an expression depending on the initial values of the parameters, (accesses to undefined local variables are treated by exception)

Thus, we can construct for a given path the path-condition. For reasoning GLOBALLY over a loop, we would have to invent an « invariant » (corresponding to an induction scheme).

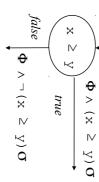
05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

30

Symbolic Execution

 $\sigma: \textbf{state}_{\text{sym}}$



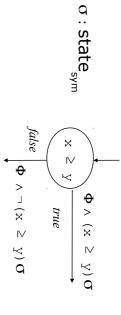
Thus, we execute symbolically and transform the symbolic state in order to obtain an expression depending on the **initial values of the parameters**, (accesses to undefined local variables are treated by exception)

Thus, we can construct for a given path the path-condition. For reasoning GLOBALLY over a loop, we would have to invent an « invariant » (corresponding to an induction scheme).

5/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

Symbolic Execution



Thus, we execute symbolically and transform the symbolic state in order to obtain an expression depending on the initial values of the parameters, (accesses to undefined local variables are treated by exception)

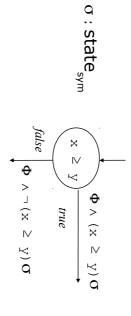
Thus, we can construct for a given path the path-condition. For reasoning GLOBALLY over a loop, we would have to invent an « invariant » (corresponding to an induction scheme).

03/03/10

B. Wolff - Inge 2 V&V - White-Box-Test

30

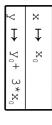
Symbolic Execution



Thus, we execute symbolically and transform the symbolic state in order to obtain an expression depending on the initial values of the parameters, (accesses to undefined local variables are treated by exception)

Thus, we can construct for a given path the path-condition. For reasoning GLOBALLY over a loop, we would have to invent an « invariant » (corresponding to an induction scheme).

Symbolic Execution



(accesses to undefined local variables are treated by exception) obtain an expression depending on the initial values of the parameters, Thus, we execute symbolically and transform the symbolic state in order to

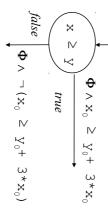
(corresponding to an induction scheme). GLOBALLY over a loop, we would have to invent an « invariant » Thus, we can construct for a given path the path-condition. For reasoning

B. Wolff - Inge 2 V&V - White-Box-Test

31

Symbolic Execution

Ţ Į $y_0 + 3 * x_0$ ×

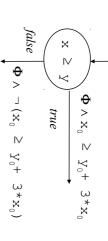


(accesses to undefined local variables are treated by exception) obtain an expression depending on the initial values of the parameters, Thus, we execute symbolically and transform the symbolic state in order to

GLOBALLY over a loop, we would have to invent an « invariant » (corresponding to an induction scheme). Thus, we can construct for a given path the path-condition. For reasoning

B. Wolff - Inge 2 V&V - White-Box-Test

Symbolic Execution



(accesses to undefined local variables are treated by exception) obtain an expression depending on the initial values of the parameters, Thus, we execute symbolically and transform the symbolic state in order to

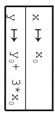
(corresponding to an induction scheme). GLOBALLY over a loop, we would have to invent an « invariant » Thus, we can construct for a given path the path-condition. For reasoning

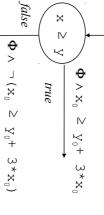
05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

31

Symbolic Execution





(accesses to undefined local variables are treated by exception) obtain an expression depending on the initial values of the parameters, Thus, we execute symbolically and transform the symbolic state in order to

(corresponding to an induction scheme). GLOBALLY over a loop, we would have to invent an « invariant » Thus, we can construct for a given path the path-condition. For reasoning

Recal

```
end supprime,
                                                                                                                                                        procedure supprime
                                                                                                                              i: integer := 1;
                end loop;
                                                                                               while
                                                                                Ħ.
                                                                elsif
                                end if;
                                                 else
                                                T[i] := T[p-1];
                                                               д
=
т
                                                                              \mathbb{T}[i] \iff x \text{ then}
                                                                                                \Diamond
                                                                                              D
                                                                                                                                                        (T: in out Table; p: in out integer;
                                                                                                                                             x: in integer) is
                                                                                              loop
                                                                1 then
                                               Д
                                                              Д
                                                 ..
                                                                 ..
                                                 D
                                                               b
                                                               - 1; return;
                                                1; return;
```

Example: A Symbolic Path Execution

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

32

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

32

Recall

```
end supprime,
                                                                                                                                                procedure supprime
                                                                                                                       i: integer := 1;
               end loop;
                                                                                           while
                                                                             j.
                                                            elsif
                              end if;
                                              else
                                             T[i] := T[p-1];
                                                            d
=
-t
                                                                           T[i] \Leftrightarrow x then
                                                                                           \Diamond
                                                                                          Д
                                                                                                                                                 (T: in out Table; p: in out integer;
                                                                                         loop
                                                             1 then
                                                                                                                                      in integer) is
                                             Д
                                                           Д
                                               ..
                                                              ..
                                              b
                                                            b
                                                              ı
                                                            1; return;
                                              1; return;
```

Example: A Symbolic Path Execution

Recal

```
procedure supprime
end supprime;
                                                                                                                       i: integer := 1;
                                                                                          while
                end loop;
                               end if;
                                                           elsif
                                                                            Ħ.
                                              else
                                                             р.
П
                                                                          T[i] \Leftrightarrow x then
                                             T[i] := T[p-1];
                                                                                           <u></u> →
                                                             ا
م
                                                                                          Д
                                                                                                                                    (T: in out Table; p: in out integer;
x: in integer) is
                                                                                         loop
                                                             1 then
                                             Д
                                                            Д
                                               ..
                                                              ..
||
                                              Q
                                                             ש
                                               1
                                                            - 1; return;
                                                                             +
                                              1; return;
```

Example: A Symbolic Path Execution

Recall

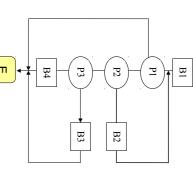
```
procedure supprime
end supprime
                                                                                                                       i: integer := 1;
                                                                                           while
              end loop;
                              end if;
                                                             elsif
                                                                             Ħ.
                                               else
                                                                                           \
\
⊢-
                                                             р-
П
                                                                           T[i] \Leftrightarrow x \text{ then}
                                             T[i] := T[p-1];
                                                             Д
                                                                                          Д
                                                                                                                                     (T: in out Table; p: in out integer;
x: in integer) is
                                                                                        loop
                                                             1 then
                                             D
                                                            Д
                                                              ..
||
                                               ·.
                                              Д
                                                             ט
                                                              ı
                                                                             +
                                                            1; return;
                                              1; return;
```

S

... and the corresponding control flow graph.

We want to execute the path:

[S,B1,P1,E]



B. Wolff - Inge 2 V&V - White-Box-Test

05/03/18

33

Ш

Example: A Symbolic Path Execution

... and the corresponding control flow graph.

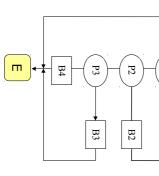
В1

S

P1

We want to execute the path:

[S,B1,P1,E]



Example: A Symbolic Path Execution

... and the corresponding control flow graph.

В1

S

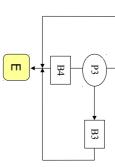
P1

P2

В2

We want to execute the path:

[S,B1,P1,E]



05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

33

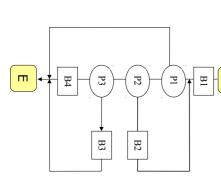
Example: A Symbolic Path Execution

S

... and the corresponding control flow graph.

We want to execute the path:

[S,B1,P1,E]



We want to execute the path:

[S, В1, Ρ1,

Ш

Example: A Symbolic Path Execution

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

34

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

34

We want to execute the path:

В1, Ρ1, Ш

[S,

Example: A Symbolic Path Execution

We want to execute the path:

В1, Ρ1,

[S,

Ш

Example: A Symbolic Path Execution

We want to execute the path:

В1, Ρ1, Ш

[S,

We want to execute the path:

[S,

Ţ

. H

Ф

:= True

D

Ţ Į

× o G

Ţ .

В1,

Ρ1,

Ш

05/03/18

35

B. Wolff - Inge 2 V&V - White-Box-Test

Example: A Symbolic Path Execution

We want to execute the path:

В1,

[S,

Ρ1,

Ш

Ф := True

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

Example: A Symbolic Path Execution

We want to execute the path:

[S

В1,

Ρ1,

Ш

 Φ := True

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

35

Example: A Symbolic Path Execution

We want to execute the path:

[S

В1,

Ρ1,

Ш

 $\Phi := \text{True}$ ×

Į Į

o O

T_o

Į Į P

We want to execute the path:

[S

В1,

Ρ1,

Ш

Ţ D Į T_o o D

Ţ

×

×

Ţ .

Į Ţ Ф

:= True

Ф

:= True

Н Į Į T_o o G

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

36

Example: A Symbolic Path Execution

We want to execute the path:

[S

В1,

Ρ1,

Ш

Ф := True

Ф := True

Ţ D Į Į Į ۲. To × od o

Į Į Į Į × To o D

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

Example: A Symbolic Path Execution

We want to execute the path:

[S

В1,

P1,

Ш

 $\Phi := \text{True}$

Ф

:= True

Н

H_o

Į Į

o G

T_o

Į Į. \times

> ⊢ Į Q Į Ţ Ţ × o O

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

36

Example: A Symbolic Path Execution

We want to execute the path:

[S

В1,

Ш

Ρ1,

 $\Phi := \text{True}$

Н

T_o

Į Į

> Ф := True

Į Į ۲. \times o O

Ţ D Į Į Į od o T_o \times

We want to execute the path:

[S

В1,

Ρ1,

Ш

Ф := True

Ф

:= True

Ţ

T_o

D Н Į Ţ Ţ Ţ ×

D

Į

Į

Į

× g

> H o G

> > Т Į Į

D Į Ţ Ţ $^{\circ}$ o O

05/03/18

37

05/03/18

۲. B. Wolff - Inge 2 V&V - White-Box-Test

Example: A Symbolic Path Execution

We want to execute the path:

[S

В1,

Ρ1,

Ш

Ф := True

Ф := True

 \vdash

Į

Τ $\Phi \!\!:= \neg \; (\, \text{i} \! < \!\! > \!\! p) \, \sigma_{_{Bl}}$ Į Į

Ţ D Į Į Į ۲. T × o G

Į Į ×

Ţ H g

Д Į Į Ţ × o O

Example: A Symbolic Path Execution

We want to execute the path:

[S

В1,

P1,

Ш

Ф := True

Ф

:= True

Н Į Į \times H_o o G

⊢ Į Q Į H_o o G

Н Д $\Phi\text{:=}\neg\text{ (i<>p)}\,\sigma_{_{BI}}$ Į Į Į °G

Ţ Ţ \approx

Į

Ţ

 $^{\circ}$

Į. Ţ

Example: A Symbolic Path Execution

B. Wolff - Inge 2 V&V - White-Box-Test

37

We want to execute the path:

[S

В1,

Ш

Ρ1,

Ţ D Į Į Ţ \times o G T_o

Н

Į

T_o

Ф

:= True

Ф

:= True

 $\Phi := \neg \; (\text{i} <> p) \; \sigma_{_{B1}}$

↓ F_o Į Į Į $^{\circ}$ od o

Į

۲.

Į Į

 \times o O

We want to execute the path:

[S

Ф

:= True

Ф

:= True

Ф

.. II

å II В1,

Ρ1,

Ш

Ţ D Į Ţ Ţ T_o ۳. × g

D Н Į Ţ Ţ Ţ × H g

Т D

Į Į To Ţ Ţ $^{\circ}$ o O

> Н D Į Ţ H

Ţ Ţ \times o G

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

38

05/03/18

Į

۲.

Į

Ţ

Example: A Symbolic Path Execution

We want to execute the path:

[S

В1,

Ρ1,

Ш

Ф := True

Ţ

To

 \vdash

Į

H

Τ

Į Į

Н

Į

Н

Į

T_o

o D

D

o D H

D

o G

Į

۲. ×

Į

Į Ţ

Ф

:= True

 $\Phi\text{:=}\neg\text{ (i<>p)}\,\sigma_{_{Bl}}$

Ф .. || °

II

Ф

:= True

× g Д Į Į Ţ

Į Ţ

 \times

o D d Ţ

Į Ţ \times

> Į Į Į

۲- \times

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

Example: A Symbolic Path Execution

We want to execute the path:

[[]S]

В1,

P1,

Ш

Ф := True

Ф

Ф

..

o D

|| |-

Н

Į Į \times H_o o G Q

⊢ Į Į Ţ \approx H_o o G

:= True Н Д

Į Į To °G

Ţ Ţ ×

Н D Ţ Į Ţ Ţ T_0 \times o G

B. Wolff - Inge 2 V&V - White-Box-Test

38

Example: A Symbolic Path Execution

We want to execute the path:

[[]S]

В1,

Ρ1,

Ш

Ф

:= True

Ф

..

o

|| |-

Ţ

Į Į Ţ \times o G T_o

H d Į Į Į Ţ × o, H

Н ם Į Į Į Ţ \times To o D

Result:

Test-Case:

Path : [S,B1,P1,E]

Path Condition: $\Phi := p_0 = 1$

A concrete Test, satisfying Φ

 \bowtie

Ţ

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

39

Example: A Symbolic Path Execution

Result:

Test-Case:

Path : [S,B1,P1,E]

Path Condition: $\Phi := p_0 = 1$

satisfying Φ A concrete Test,

$$\begin{array}{c|c} T & \rightarrow & \text{mtTab} \\ \hline p & \rightarrow & 1 \\ \times & \rightarrow & 17 \\ \end{array}$$

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

39

05/03/18

Example: A Symbolic Path Execution

Result:

Test-Case:

Path : [S,B1,P1,E]

Path Condition: $\Phi := p_0 = 1$

satisfying Φ A concrete Test,

× Q ⊢ Į Į mtTab 17

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

39

Example: A Symbolic Path Execution

Result:

Test-Case:

Path : [S,B1,P1,E]

Path Condition: $\Phi := p_0 = 1$

satisfying Φ A concrete Test,

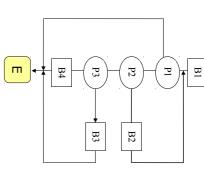


S

... and the corresponding control flow graph.

We want to execute the path:

[S,B1,P1,P2,B2,P1,E]



05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

40

Example: A Symbolic Path Execution

... and the corresponding control flow graph.

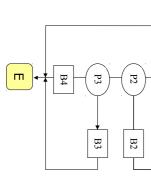
В1

S

P1

We want to execute the path:

[S,B1,P1,P2,B2,P1,E]



Example: A Symbolic Path Execution

... and the corresponding control flow graph.

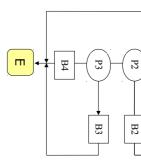
В1

S

P1

We want to execute the path:

[S,B1,P1,P2,B2,P1,E]



05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

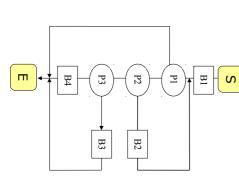
40

Example: A Symbolic Path Execution

... and the corresponding control flow graph.

We want to execute the path:

[S,B1,P1,P2,B2,P1,E]



05/03/18

We want to execute the path:

[S,

В1,

Ρ1,

P2,

В2,

Ρ1,

Ш

×.	ч	ы	
Ţ	Ţ	Ţ	Φ →
×	Po	T ₀	(D
X_0	$\mathtt{p}_{\scriptscriptstyle{0}}$	${\mathbb T}_0$	True
	$X \rightarrow X_0 \qquad X_0$	$\begin{array}{cccccccccccccccccccccccccccccccccccc$	$\begin{array}{cccccccccccccccccccccccccccccccccccc$

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

Example: A Symbolic Path Execution

41

Example: A Symbolic Path Execution

We want to execute the path:

[S,	↓	True	T → T ₀	p → p₀	×	Ţ. Ţ.
[S, B1, P1,		True	${ m T}_{ m 0}$	P_0	X ₀	⊢
P1,						
P2,						
B2,						
P1,						
皿						

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

41

Example: A Symbolic Path Execution

We want to execute the path:

	i → i₀	$x \mapsto x_0 \qquad x_0$	p → p ₀	$T \mapsto T_0$	True	Ф ↓	[S,
	1	X_0	\mathbf{p}_0	${ m T}_{ m 0}$	True		[S, B1, P1,
							P1,
							P2,
							B2,
•							P1,
							皿

Ţ g ⊢ Į

od o ×

× o D H

H

Ţ Į

ф Į True

True

[S

В1,

Ρ1,

P2,

В2,

Ρ1,

Ш

We want to execute the path:

We want to execute the path:

[S,

В1,

Ρ1,

P2,

В2,

Ρ1,

Ш

i → i₀	$x \rightarrow x_0 \qquad x_0$	p → p ₀	T ↓ T ₀	Φ ↔ True
1		P_0	${\mathbb T}_0$	True
1	X ₀	\mathfrak{P}_0	${ m T}_0$	$(i <> p) \sigma_{BI}$ $\equiv p_0 \neq 1$

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

42

Example: A Symbolic Path Execution

We want to execute the path:

[S, B1, P1, P2, B2, P1, E]
$$\Phi \mapsto \begin{bmatrix} (i <> p) \sigma_{Bl} \\ \exists p_0 \neq 1 \end{bmatrix}$$

ı. ↓ ı. 0	×	p ↓ p₀	T ↓ T _o	Φ →
₽	X ₀	p_0	T _o	True
1	X_0	\mathfrak{P}_0	${\mathtt T}_0$	$(i <> p) \sigma_{BI}$ $\equiv p_0 \neq 1$

Example: A Symbolic Path Execution

We want to execute the path:

[S В1, Ρ1, P2,

	ı. ↓ ı. 0	$X \longrightarrow X_0 \longrightarrow X_0$	p → p ₀	T → T ₀	Φ ↔ True	[S,
	Н	× ₀	₽ ₀	\mathbb{T}_0	True	B1,
	⊢	X ₀	P_0	T ₀	$\begin{array}{l} (i <> p) \sigma_{BI} \\ \equiv p_0 \neq 1 \end{array}$	[S, B1, P1,
						P2,
						B2,
						P1,
•						皿

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

42

Example: A Symbolic Path Execution

We want to execute the path:

i → i ₀	X X X X X	p ↓ p ₀	T → T ₀	Φ →	[-/
1	X ₀	P ₀	${\mathtt T}_0$	True	,
1	X_0	P_0	${f T}_0$	$(i <> p) \sigma_{BI}$ $\equiv p_0 \neq 1$	[-]]
					,
					1
					, -,
]

We want to execute the path:

[S,

В1,

Ρ1,

P2,

В2,

Ρ1,

Ш

[S

В1,

Ρ1,

P2,

В2,

Ρ1,

Ш

р ,	∓ ↓ Į	Φ ↓ True
P ₀	T	True
P_0	T	$(i <> p) \sigma_{BI}$ $\equiv p_0 \neq 1$
P ₀	T	p ₀ ≠1 Λ (T[i] <>x)σ _{B1}

05/03/18

Į Ţ

×

×

×

×

B. Wolff - Inge 2 V&V - White-Box-Test

43

Example: A Symbolic Path Execution

We want to execute the path:

[S,	[S, B1,	P1,	P2,	B2,	P1,	皿
Ţ		$(i <> p) \sigma_{Bl} p_0 \neq 1 \wedge $	ზ° ₀ ≠1 ∨			
True	True	≡ p ₀ ≠ 1	(+ [+] G _{B1}			
T ↓ T ₀	丁。	${ t T}_0$	T ₀			
p → p ₀	$\mathtt{p}_{\scriptscriptstyle{0}}$	P_0	₽ ₀			
X X O	X_0	X_0	X_0			
ı. ↓ ı. 0	1	1	₽			

Example: A Symbolic Path Execution

We want to execute the path:

Φ →	
True	
$\begin{array}{l} (i <> p) \sigma_{BI} \\ \equiv p_0 \neq 1 \end{array}$	
$\begin{array}{c} p_{_{0}} \neq 1 \text{ A} \\ (\text{T[i]} \\ <> x) \sigma_{_{B1}} \end{array}$	

Ţ

Н

Н g

H

Д

Ţ Ţ Į

 \times o G

 \times o G

×

× o G

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

43

Example: A Symbolic Path Execution

We want to execute the path:

[S,	B1,	[S, B1, P1,	P2,	B2,	P1,
Ţ Ф		$(i <> p) \sigma_{BI} p_0 \neq 1 \Lambda$	ზ°≢1 ∨		
True	True	≡ p ₀ ≠ 1	(Σ[Σ] <>x) σ _{B1}		
T → T ₀	${ m T}_{ m 0}$	$\mathtt{T}_{\scriptscriptstyle{0}}$	${\mathbb T}_0$		
р , р ₀	P_0	\mathbf{p}_0	\mathcal{P}_0		
¥ ¥ ×	X_0	X_0	X_0		
i ↓ i₀	1	1	1		

We want to execute the path:

Ф ф

В1, Ρ1,

[S,

P2,

В2,

Ρ1, Ш

₽ T Н True Į Ţ Į Į × o G H True Н × g $(i <> p) \sigma_{BI}$ $\equiv p_0 \neq 1$ × H d p₀≠1 ∧ (T[i] $\langle \rangle_X) \sigma_{_{B1}}$ × H g $T_0[1] \neq X_0$ $(i+1)\sigma_{\mathbf{I}}$ × H o G

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

44

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

44

Example: A Symbolic Path Execution

We want to execute the path:

Ф ф

[S]В1, Ρ1,

P2,

В2,

Ρ1,

Ш

i → i₀	°°	p → p ₀	$T \mapsto T_0$	Φ ↓
₽	× _o	P ₀	${ m T}_{ m 0}$	True
1	X_0	P_0	${ m T}_{ m 0}$	$\begin{array}{c} (\texttt{i} < > \texttt{p}) \sigma_{\texttt{B}} & \texttt{p}_{_0} \neq \texttt{1} \; \land \\ \equiv \; \texttt{p}_{_0} \neq \; \texttt{1} & (\texttt{T} [\texttt{i}] \\ < > \texttt{x}) \sigma_{\texttt{B}} & < > \texttt{x} \end{array}$
⊢	X_0	\mathcal{P}_0	${ m T}_{ m 0}$	
(i+1) $\sigma_{_{\rm B1}}$	X ₀	P ₀	\mathtt{T}_0	$\begin{array}{c} p_0 \neq 1 \mathbf{A} \\ T_0 \left[1 \right] \neq X_0 \end{array}$

Example: A Symbolic Path Execution

ф

We want to execute the path:

Ф

P2, В2,

₽ T

[2,

В1,

True

Ρ1, Ш

True H \times g $(i <> p) \sigma_{BI}$ $\equiv p_0 \neq 1$ Ρ1, × H ď p₀≠1 ∧
(T[i] <>X) $\sigma_{_{B1}}$ × H g p₀≠1 ∧ $T_0[1] \neq X_0$ (i+1)σ × H o D

Н

Į

Į

Example: A Symbolic Path Execution

We want to execute the path:

ФФ

P2, В2,

i	× ↓ × ₀	p → p ₀	$T \rightarrow T_0$	Φ →	[S,
1	X ₀	$\mathtt{p}_{\scriptscriptstyle{0}}$	${\mathbb T}_0$	True	[S, B1,
1	X_0	P_0	${f T}_0$	$(i <> p) \sigma_{BI} \qquad p_0 \neq 1 \Lambda$ $\equiv p_0 \neq 1 \qquad (T[i] <> x) c$	P1,
⊢	X_0	₽ ₀	T ₀	, B1	P2,
(i+1)σ _{B1}	X_0	Þ ₀	F ₀	$P_0 \neq 1$ Λ $T_0 [1] \neq X_0$	B2,
					P1,
					旦

Į

Į

× o D H

We want to execute the path:

+	
,	[S,
	В1,
(i<>>b) σ	P1,
p,≠1 ∧	P2,
p,≠1 ∧	B2,
p ≠1 ∧	P1,

	2	$(i+1)\sigma_{_{\mathbf{R}1}}$	1	₽	1	i. ↓ i.
	X_0	X ₀	X ₀	X ₀	X_0	×
	₽ ₀	Þ ₀	₽ ₀	P_0	\mathbf{p}_0	р ↓ р ₀
	T ₀	T ₀	T ₀	$\mathbb{T}_{\scriptscriptstyle{0}}$	${f T}_0$	T → T ₀
Ö	$\begin{array}{ccc} \mathbf{p}_0 \neq 1 & \mathbf{\Lambda} \\ \mathbf{T}_0 [1] & \neq & \mathbf{X}_0 \\ \mathbf{\Lambda}^{-} & (\text{i} <> \mathbf{p}) & \mathbf{\sigma}_{\mathbf{B}} \end{array}$	$\begin{array}{c} p_0 \neq 1 \mathbf{A} \\ T_0 [1] \neq X_0 \end{array}$	31	$(i <> p) \sigma_{BI} \qquad p_0 \neq 1 \Lambda$ $\equiv p_0 \neq 1 \qquad (T[i] \times x) \sigma_{BI} \qquad (x \neq y) \sigma_{BI} \qquad (x \neq y$	True	Φ → True

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

45

Example: A Symbolic Path Execution

_	_
~	2
a	D
2	٤
alle	ن
_	ŗ
_	+
_	+
5)
_	
ú	'n
>	<
Yachra	D
\sim)
_	=
	+
a	0
_	+
2	÷
7	ζ.
(I	•
τ	7
ň	ú
שנו	¥
-	÷
-	:

$\begin{array}{cccccccccccccccccccccccccccccccccccc$	[S,	[S, B1,	P1,			P1,	皿
True $ \begin{array}{ c c c c c c }\hline \text{True} & \begin{array}{ c c c c c c c c c c c c c c c c c c c$	Ф ↓		$(i <> p) \sigma_{BI}$			P₀≠1 ∧	
$egin{array}{cccccccccccccccccccccccccccccccccccc$	True		$\equiv p_0 \neq 1$	(T[i]	$T_0[1] \neq X_0$	$T_0[1] \neq X_0$	
$egin{array}{cccccccccccccccccccccccccccccccccccc$				<>x) σ _{B1}		Λ (i<>p) $\sigma_{\rm B}$	2
$egin{array}{cccccccccccccccccccccccccccccccccccc$	T → T ₀	${f T}_0$	$\mathtt{T}_{\scriptscriptstyle{0}}$	T ₀	\mathbb{T}_0	T_{o}	
$egin{array}{cccccccccccccccccccccccccccccccccccc$	p → p ₀	\mathbf{p}_0	\mathfrak{P}_0	₽ ₀	Þ ₀	P ₀	
1 1 1 $(\dot{1}+1)\sigma_{\text{Bl}}$	¥ ¥ ×	X_0	X_0	X_0	X_0	X ₀	
	i → i₀	1	1	1	(i+1) o _{B1}	2	

Example: A Symbolic Path Execution

We want to execute the path:

Φ → [S,	¶True	T ↓ T ₀	p → p ₀	¥ × ×	i ↓ i₀
[S, B1,	True	ОТ	0 d	×	1
P1,	$\equiv p_0 \neq 1 (T[i]$	${f T}_0$	P_0	× ₀	1
	5 B1	T ₀	₽ ₀	×	1
B2, p ₀ ≠1 ∧	$\vec{\mathrm{T_{0}}}[1] \neq X_{0}$	\mathbb{T}_0	\mathfrak{P}_0	× ₀	$(i+1)\sigma_{_{\mathrm{R}1}}$
P1 , p ₀ ≠1 ∧	$\tilde{T}_0[1] \neq X_0$ $\Lambda^-(i <> p) \sigma_B$	\mathbf{T}_0	ಿದ	X_0	2
	2				

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

45

Example: A Symbolic Path Execution

We want to execute the path:

	Λ ¬ (i<>p) σ_{B2}		<>x) σ _{B1}			
	$T_0[1] \neq X_0$	$T_0[1] \neq X_0$	(T[i]	$\equiv p_0 \neq 1 \mid (T[i])$	True	True
	$p_0 \neq 1$ A		p ₀ ≠1 ∧	$(i <> p) \sigma_{BI}$		↔ ф
Ш	P1,	B2,	P2,	P1,	[S, B1,	[S,

i → i ₀	X X 0 X 0	р Т р ₀	$T \mapsto T_0$	Φ →
1	X_0	Po	\mathbf{T}_0	True
1	X ₀	\mathbf{p}_0	\mathbf{T}_0	$\begin{array}{l} (i <> p) \sigma_{BI} \\ \equiv p_{_{0}} \neq 1 \end{array}$
1	X ₀	P ₀	T ₀	$\begin{array}{c} p_{_{0}} \neq 1 \\ (\text{T[i]} \\ <> x) \sigma_{_{BI}} \end{array}$
$(i+1)\sigma_{\rm Bl}$	X ₀	Ъo	T _o	$p_0 \neq 1 \wedge \\ T_0 [1] \neq X_0$
2	× ₀	Po	T ₀	$\begin{array}{c} p_0 \neq 1 \mathbf{A} \\ T_0 [1] \neq X_0 \\ \mathbf{A} \neg (\text{i} <> p) \mathbf{G}_B \end{array}$
				2

We want to execute the path:

	×	$p \mapsto p_0 p_0$	$T \mapsto T_0 \qquad T_0$	Φ → True True	[S, B1,
Н	× ₀	\mathcal{P}_0	T ₀	$(i <> p) \sigma_{BI}$ $\equiv p_0 \neq 1$	P1,
Р	X ₀	₽ ₀	円。	ν _{Β1}	P2,
(i+1)σ _{в1}	X ₀	₽ ₀	T _o	$p_0 \neq 1$ Λ $T_0 [1] \neq X_0$	B2,
2	X_0	P ₀	H ₀	$\begin{array}{cccccccccccccccccccccccccccccccccccc$	P1,
2	X ₀	P ₀	T _o	$ \begin{array}{ccc} p_0 \neq 1 & \mathbf{A} \\ T_0 & [1] \neq X_0 \\ \mathbf{A} & p_0 = 2 \end{array} $	匝

05/03/18

46

B. Wolff - Inge 2 V&V - White-Box-Test

Example: A Symbolic Path Execution

We want to execute the path:

[S,	[S, B1,	P1,	P2,	B2,	P1,	旦
Ţ		$(\dot{\text{1}}<>p)\sigma_{_{B1}}$	p ₀ ≠1 ∧		p ₀ ≠1 ∧	p ₀ ≠1 ∧
True	True	$\equiv p_0 \neq 1$		$T_0[1] \neq X_0$	$T_0[1] \neq X_0 T_0[1] \neq X_0$	$T_0[1] \neq X_0$
F S (} } (<>x) σ _{B1}		Λ^{-} (i<>p) $\sigma_{B2} \Lambda p_0=2$	$^{2} \ \mathbf{p}_{0} = 2$
T ↓ T ₀	丁。	${ m T}_{ m 0}$	${ m T}_{ m 0}$	${f T}_0$	T _o	T_0
$p \mapsto p_0$	\mathbf{p}_0	P_0	₽ ₀	p ₀	P ₀	P_0
×	X_0	X_0	X_0	X ₀	X ₀	X_0
↓. ↓ ↓. 0	1	1	₽	(i+1)σ _{n1}	2	2

Example: A Symbolic Path Execution

We want to execute the path:

[S,

В1,

Ρ1,

P2,

В2,

Ρ1,

Ш

ı. ↓ ı. 0	↓ × ×	p ↓ p ₀	T → T ₀	Φ → True
₽	X ₀	P ₀	${ m T}_{ m 0}$	True
1	X_0	P_0	${f T}_0$	$(i <> p) \sigma_{BI}$ $\equiv p_0 \neq 1$
₽	X ₀	Po	T ₀	p ₀ ≠1 Λ (T[i] <>x) σ _{B1}
(i+1)σ _B	× ₀	ರಂ	F _o	$p_0 \neq 1$ Λ $T_0 [1] \neq X_0$
2	X_0	P_0	T ₀	$\begin{array}{cccccccccccccccccccccccccccccccccccc$
2	X ₀	\mathcal{P}_0	T _o	$ \begin{array}{ccc} p_0 \neq 1 & \mathbf{A} \\ T_0 & [1] \neq X_0 \\ \mathbf{A} & p_0 = 2 \end{array} $

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

46

Example: A Symbolic Path Execution

We want to execute the path:

i i₀	× ↓ × ₀	р Т р ₀	$T \mapsto T_0$	Φ →	[S,
₽	X ₀	P ₀	\mathbf{T}_0	True	[S, B1,
1	X_0	P_0	T ₀	$(i <> p) \sigma_{\mathbf{B}_{\mathbf{I}}} p_{0} \neq \mathbf{I}$ $\equiv p_{0} \neq \mathbf{I} (T[i])$ $<> x) c$	P1,
1	X_0	P ₀	T ₀	ν _{В1}	P2,
(i+1)σ _{в1}	X ₀	₽ ₀	T _o	$p_0 \neq 1$ Λ $T_0 [1] \neq X_0$	B2,
2	× ₀	Po	H ₀	$p_0 \neq 1 \land p_0 \neq 1 \land p_0 \neq 1 \land p_0 = 1 \land p_0 = 2$ $p_0 \neq 1 \land p_0 \neq 1 \land p_0 = 1 \land p_0 \neq 1 \land p_0 = 2$	P1,
2	X ₀	\mathcal{P}_0	T _o	$ \begin{array}{ccc} p_0 \neq 1 & \mathbf{A} \\ T_0 & 1 & 1 \neq X_0 \\ \mathbf{A} & \mathbf{P}_0 = 2 \end{array} $	旦

We want to execute the path:

₽ ↓ ₽ ₀	$\mathtt{T} \;\; {\boldsymbol{\mapsto}} \;\; \mathtt{T}_0$	Φ →	[S,
\mathbf{p}_0	${f T}_0$	True	[S, B1,
${\tt p}_{\scriptscriptstyle 0}$	${f T}_0$	$(i <> p) \sigma_{Bi} \qquad p_0 \neq 1 \Lambda$ $\equiv p_0 \neq 1 \qquad (T[i] \qquad <> x) \sigma_i$	P1,
\mathcal{P}_0	T ₀	$p_0 \neq 1$ \wedge $(T[i]$ $>x) \sigma_{B1}$	P2,
P_0	T _o	$\begin{array}{c} p_0 \neq 1 \mathbf{A} \\ T_0 [1] \neq X_0 \end{array}$	B2,
P_0	T _o	$\begin{array}{ccccc} p_{_{0}} \! \! \! \! \! \! \! \! \! \! \! \! \! \! \! \! \! \! $	P1,
P_0	T_{o}	$ \begin{array}{ccc} p_0 \neq 1 & \mathbf{A} \\ T_0 & 1 & 1 \neq X_0 \\ \mathbf{A} & p_0 = 2 \end{array} $	E]

05/03/18

Į Ţ

×

×

×

×

 $(i+1)\sigma_{I}$

N

N

B. Wolff - Inge 2 V&V - White-Box-Test

≠X₀ =2

× × ×

47

Example: A Symbolic Path Execution

We want to execute the path:

i ↓ i₀	¥	р , р ₀	T → T _o	Φ →	S]
-0 1	X ₀	ρ ₀	T ₀	True	[S, B1,
1	× ₀	0 ರ	日。	$\begin{array}{l} (\texttt{i} < > \texttt{p}) \sigma_{\texttt{BI}} \\ \equiv p_{_{0}} \neq 1 \end{array}$	P1,
1	X ₀	P ₀	다 ₀	P ₀ ≠1 Λ (T[i] <>x) σ _{B1}	P2,
$(i+1)\sigma_{B1}$	× ₀	P ₀	T_0	p ₀ ≠1 ∧ T ₀ [1]≠X ₀	B2,
N	X_0	P ₀	H _o	$\begin{array}{cccccccccccccccccccccccccccccccccccc$	P1,
2	X ₀	₽°	T _o	$ \begin{array}{c} p_0 \neq 1 \mathbf{A} \\ T_0 [1] \neq X_0 \\ \mathbf{A} p_0 = 2 \end{array} $	皿

Example: A Symbolic Path Execution

We want to execute the path:

ı́ ↓ ¹₀	· · · · · · · · · · · · · · · · · · ·	Į × ×	p → p ₀	$T \rightarrow T_0$		True	0 ↓	[S,
1		× ₀	$\mathbf{p}_{\scriptscriptstyle{0}}$	${\mathbb T}_0$		True		[S, B1,
Р		X ₀	P_0	${ m T}_{ m o}$		$\equiv p_0 \neq 1$	$(i <> p) \sigma_{_{BI}}$	P1,
Н		× ₀	P ₀	T ₀	<>x) σ _{B1}		$p_0 \neq 1 \wedge$	P2,
$(i+1)\sigma_{\rm Bl}$		X ₀	₽ ₀	T ₀		$T_0[1] \neq X_0$		B2,
^)	X_0	\mathbf{p}_0	T_0	Λ (i<>p) $\sigma_{\rm p} \Lambda p_0=2$	$T_0[1] \neq X_0 T_0[1] \neq X$	p ₀ ≠1 ∧	P1,
)	X ₀	P ₀	T _o	$^{2} \wedge p_{0} = 2$	T ₀ [1]≠X	$p_{_0} \neq 1$ /	E

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

47

Example: A Symbolic Path Execution

We want to execute the path:

i → i₀	↓ × °	$p \mapsto p_0$	$T \mapsto T_0$	Φ ↔ True	[S,
1	X_0	p ₀	T ₀	True	[S, B1,
1	X ₀	P_0	T _o	$(i <> p) \sigma_{\mathbf{B}_{\mathbf{I}}} p_{0} \neq 1 $ $\equiv p_{0} \neq 1 (T[i] <> x) c$	P1,
₽	X_0	P ₀	T ₀	ν _{В1}	P2,
(i+1) σ _{B1}	X_0	P ₀	T _o	$p_0 \neq 1$ Λ $T_0 [1] \neq X_0$	B2,
N	X_0	p ₀	т _о	$\begin{array}{cccccccccccccccccccccccccccccccccccc$	P1,
2	X_0	P ₀	T _o	$\begin{array}{c} \mathbf{p}_{o} \neq 1 \mathbf{\Lambda} \\ \mathbf{T}_{o} [1] \neq \mathbf{X}_{o} \\ \mathbf{\Lambda} \mathbf{p}_{o} = 2 \end{array}$	皿

Result:

Test-Case:

Path : [S,B1,P1,P2,B2,P1,E]

Path Condition: $\Phi := T_0[1] \neq X_0 \land p_0=2$

satisfying Φ A concrete Test,

Į Į N $[\omega]$

Į 17

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

48

Example: A Symbolic Path Execution

Result:

Test-Case:

Path : [S,B1,P1,P2,B2,P1,E]

Path Condition: $\Phi := T_0[1] \neq X_0 \land p_0=2$

satisfying **Φ** A concrete Test,

Н Į Ţ Į 17 $[\omega]$

05/03/18

Example: A Symbolic Path Execution

Result:

Test-Case:

Path: [S,B1,P1,P2,B2,P1,E]

Path Condition: $\Phi := T_0[1] \neq X_0 \land p_0=2$

satisfying **Φ** A concrete Test,

Į Q Į ω

×

Į

17

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

48

Example: A Symbolic Path Execution

Result:

Test-Case:

Path: [S,B1,P1,P2,B2,P1,E]

Path Condition: $\Phi := T_0[1] \neq X_0 \land p_0=2$

satisfying **Φ** A concrete Test,



Paths and Test Sets

In (this version of) program-based testing a test case with a (feasable) path

- □ a test case ≈ an initial path in M
- a collection of values for variables (params and global) (+ the output values described by the spécification)
- a test case set ≈ a finite set of paths of M
- (by assuming a uniformity hypothesis)
 a finite set of input values and
 a set of expected outputs.

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

49

Paths and Test Sets

In (this version of) program-based testing a test case with a (feasable) path

- □ a test case ≈ an initial path in M
- a collection of values for variables (params and global)
 (+ the output values described by the spécification)
- a test case set ≈ a finite set of paths of M
- = (by assuming a uniformity hypothesis)
 a finite set of input values and
 a set of expected outputs.

Paths and Test Sets

In (this version of) program-based testing a test case with a (feasable) path

a test case ≈ an initial path in M

- a collection of values for variables (params and global) (+ the output values described by the spécification)
- $lue{}$ a test case set pprox a finite set of paths of M
- = (by assuming a uniformity hypothesis)
 a finite set of input values and
 a set of expected outputs.

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

49

Paths and Test Sets

In (this version of) program-based testing a test case with a (feasable) path

a test case ≈ an initial path in M

- a collection of values for variables (params and global)
 (+ the output values described by the spécification)
- a test case set \approx a finite set of paths of M

II

(by assuming a uniformity hypothesis) a finite set of input values and a set of expected outputs.

05/03/18

Unfeasible paths and decidability

- In general, it is undecidable of a path is feasible
- In general, it is undecidable if a program will terminate ...
- In general, equivalence on two programs is undecidable ...
- In general, a first-order formula over arithmetic is undecidable ...

"we know none"! that there is no algorithm; this is worse than **Indecidable** = it is known (mathematically proven)

BUT: for many relevant programs, practically good solutions exist (Z3, Simplify, CVC4, AltErgo ...)

B. Wolff - Inge 2 V&V - White-Box-Test

50

Unfeasible paths and decidability

- In general, it is undecidable of a path is feasible
- In general, it is undecidable if a program will terminate ...
- In general, equivalence on two programs is undecidable ...
- In general, a first-order formula over arithmetic is undecidable ...

"we know none"! that there is no algorithm; this is worse than **Indecidable** = it is known (mathematically proven)

BUT: for many relevant programs, practically good solutions exist (Z3, Simplify, CVC4, AltErgo)

B. Wolff - Inge 2 V&V - White-Box-Test

Unfeasible paths and decidability

- In general, it is undecidable of a path is feasible
- In general, it is undecidable if a program will terminate ...
- In general, equivalence on two programs is undecidable ...
- In general, a first-order formula over arithmetic is undecidable ...

"we know none"! that there is no algorithm; this is worse than **Indecidable** = it is known (mathematically proven)

BUT: for many relevant programs, practically good solutions exist (Z3, Simplify, CVC4, AltErgo ...

B. Wolff - Inge 2 V&V - White-Box-Test

50

Unfeasible paths and decidability

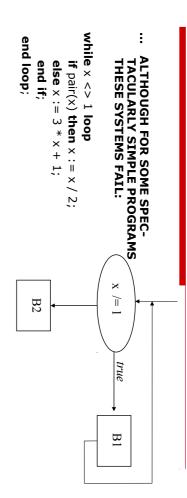
- In general, it is undecidable of a path is feasible
- In general, it is undecidable if a program will terminate ...
- In general, equivalence on two programs is undecidable ...
- In general, a first-order formula over arithmetic is undecidable ...

that there is no algorithm; this is worse than "we know none"! **Indecidable** = it is known (mathematically proven)

BUT: for many relevant programs, practically good solutions exist (Z3, Simplify, CVC4, AltErgo ...)

B. Wolff - Inge 2 V&V - White-Box-Test

A Challange-Example (Collatz-Function):

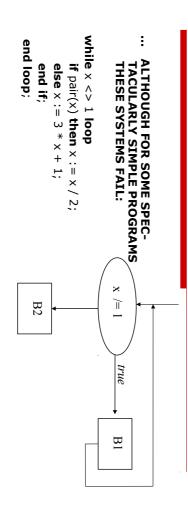


- does this function terminate for all x?
- or equivalently: is B2 reached for all x?

B. Wolff - Inge 2 V&V - White-Box-Test

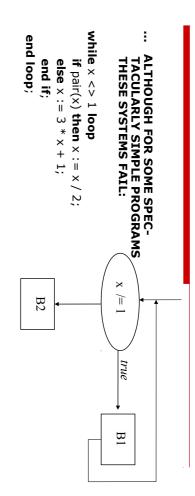
05/03/18

A Challange-Example (Collatz-Function):



- does this function terminate for all x ?
- or equivalently: is B2 reached for all \times ?

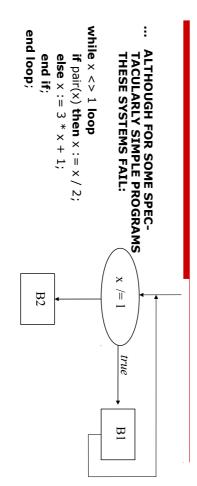
A Challange-Example (Collatz-Function):



- does this function terminate for all x ?
- or equivalently: is B2 reached for all x?

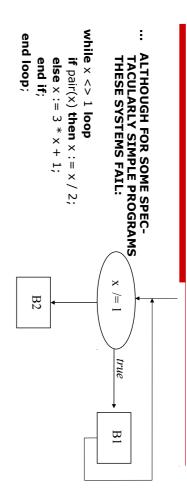
05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 51

A Challange-Example (Collatz-Function):



- does this function terminate for all x?
- or equivalently: is B2 reached for all x ?

A Challange-Example (Collatz-Function):

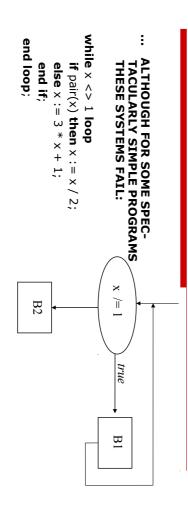


- does this function terminate for all x ?
- or equivalently: is B2 reached for all x? ANSWER:unknown

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test

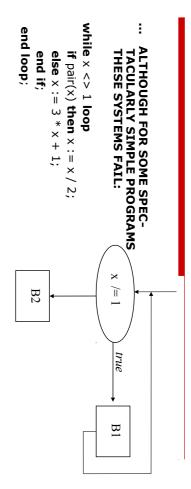
52

A Challange-Example (Collatz-Function):



- does this function terminate for all x?
- or equivalently: is B2 reached for all x? ANSWER:unknown

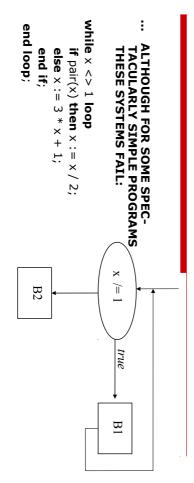
A Challange-Example (Collatz-Function):



- does this function terminate for all x ?
- or equivalently: is B2 reached for all x? ANSWER:unknown

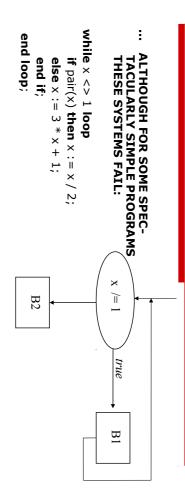
05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 52

A Challange-Example (Collatz-Function):



- does this function terminate for all x ?
- or equivalently: is B2 reached for all x? ANSWER:unknown

A Challange-Example (Collatz-Function):



- does this function terminate for all x ?
- or equivalently: is B2 reached for all x? ANSWER:unknown
- this implies that we can not know in advance that there exist infeasible paths!

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

53

A Challange-Example (Collatz-Function):

```
... ALTHOUGH FOR SOME SPEC-
TACULARLY SIMPLE PROGRAMS
THESE SYSTEMS FAIL:

while x <> 1 loop
if pair(x) then x := x / 2;
else x := 3 * x + 1;
end if;
end loop;

B1

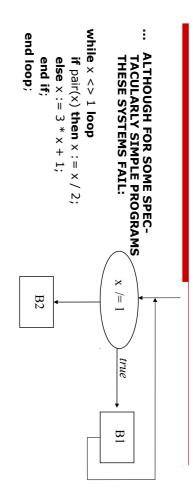
B2
```

- does this function terminate for all x?
- or equivalently: is B2 reached for all x? ANSWER:unknown
- this implies that we can not know in advance that there exist infeasible paths!

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

A Challange-Example (Collatz-Function):



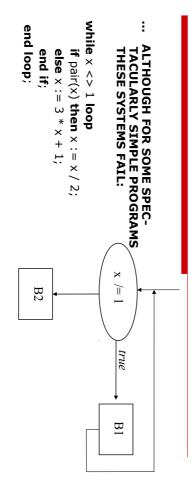
- does this function terminate for all x?
- or equivalently: is B2 reached for all x? ANSWER:unknown
- this implies that we can not know in advance that there exist infeasible paths!

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

53

A Challange-Example (Collatz-Function):



- does this function terminate for all x?
- or equivalently: is B2 reached for all x? ANSWER:unknown
- this implies that we can not know in advance that there exist infeasible paths!

The Triangle Prog without Unfeasible Paths

```
end if;
                                                                                                                                                                                                                                                                                                                          procedure triangle(j,k,l)
                                                      else
                                                                                                    elsif j = k or k =1 or j = 1 then put("isocele")
                                                                                                                                                          elsif j = k and k = l then put("equilateral");
                                                                                                                                                                                                                if j \ k \le 1 \ \text{or} \ k+1 \le j \ \text{or} \ 1+j \le k \ \text{then} \ \text{put("impossible")};
                                                put ("quelconque");
```

If we find a path for which we do not know that it is feasible is an error ... our prover is too week), however, it is likely in practice that there (maybe for deep mathematical reasons, maybe simply because

B. Wolff - Inge 2 V&V - White-Box-Test

54

The Triangle Prog without Unfeasible Paths

```
end if;
                                                                                                                                                                                                                                                                                                     procedure triangle(j,k,l)
                                                                                             elsif j = k or k =1 or j = 1 then put("isocele")
                                                                                                                                               elsif j = k and k = l then put("equilateral");
                                                                                                                                                                                                 if j k<=l or k+1<=j or l+j<=k then put("impossible");</pre>
                                               put ("quelconque");
```

If we find a path for which we do not know that it is feasible our prover is too week), however, it is likely in practice that there is an error ... (maybe for deep mathematical reasons, maybe simply because

B. Wolff - Inge 2 V&V - White-Box-Test

procedure triangle(j,k,1) if j k<=l or k+l<=j or l+j<=k then put("impossible");</pre> elsif j = k or k =1 or j = 1 then put("isocele") elsif j = k and k = l then put("equilateral"); The Triangle Prog without Unfeasible Paths put ("quelconque");

end if;

else

If we find a path for which we do not know that it is feasible is an error ... our prover is too week), however, it is likely in practice that there (maybe for deep mathematical reasons, maybe simply because

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

54

The Triangle Prog without Unfeasible Paths

```
end if;
                                                                                                                                                                                                                                                              procedure triangle(j,k,1)
                                                                                                                           elsif j = k and k = l then put("equilateral");
                                                                                                                                                                            if j k<=l or k+l<=j or l+j<=k then put("impossible");</pre>
                                         else
                                                                                  elsif j = k or k = l or j = l then put("isocele")
                                    put ("quelconque");
```

If we find a path for which we do not know that it is feasible is an error ... our prover is too week), however, it is likely in practice that there (maybe for deep mathematical reasons, maybe simply because

The notion of a "couverage criteria"

characterizing a particular subset of its paths ... A coverage criterion is a predicate on CFG

M = a procedure (with associated CFG G)

T = a test case set = a finite set of **feasable** paths in M

C = a coverage criterion (= a "set of paths")

C(M, T) is true iff T satisfies the criterion C

Examples

- all nodes appear at least once in T
- all arcs appear at least once in T

B. Wolff - Inge 2 V&V - White-Box-Test

55

The notion of a "couverage criteria"

characterizing a particular subset of its paths ... A coverage criterion is a predicate on CFG

M = a procedure (with associated CFG G)

T = a test case set = a finite set of **feasable** paths in M

C = a coverage criterion (= a "set of paths")

C(M, T) is true iff T satisfies the criterion C

Examples

- all nodes appear at least once in T
- all arcs appear at least once in T

B. Wolff - Inge 2 V&V - White-Box-Test

55

The notion of a "couverage criteria"

characterizing a particular subset of its paths ... A coverage criterion is a predicate on CFG

M = a procedure (with associated CFG G)

C = a coverage criterion (= a "set of paths") T = a test case set = a finite set of **feasable** paths in M

C(M, T) is true iff T satisfies the criterion C

Examples

- all nodes appear at least once in T
- all arcs appear at least once in T

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

55

The notion of a "couverage criteria"

characterizing a particular subset of its paths ... A coverage criterion is a predicate on CFG

M = a procedure (with associated CFG G)

C = a coverage criterion (= a "set of paths") T = a test case set = a finite set of **feasable** paths in M

C(M, T) is true iff T satisfies the criterion C

Examples

- all nodes appear at least once in T
- all arcs appear at least once in T

05/03/18

Well-known Coverage Criteria I

Criterion AllInstructions(M,T):

For all nodes N (basic instructions or decisions) in the CFG of M exists a path in T that contains N

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

56

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

56

Well-known Coverage Criteria I

Criterion AllInstructions(M,T):

For all nodes N (basic instructions or decisions) in the CFG of M exists a path in T that contains N

Well-known Coverage Criteria I

Criterion AllInstructions(M,T):

For all nodes N (basic instructions or decisions) in the CFG of M exists a path in T that contains N

Well-known Coverage Criteria I

Criterion AllInstructions(M,T):

For all nodes N (basic instructions or decisions) in the CFG of M exists a path in T that contains N

05/03/18

Well-known Coverage Criteria II

Criterion AllTransitions(M,T):

For all arcs A in the CFG of M exists a path in T that uses A $\,$

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

57

Well-known Coverage Criteria II

Criterion AllTransitions(M,T):

For all arcs A in the CFG of M exists a path in T that uses A $\,$

Well-known Coverage Criteria II

Criterion AllTransitions(M,T):

For all arcs A in the CFG of M exists a path in T that uses A $\,$

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

57

Well-known Coverage Criteria II

Criterion AllTransitions(M,T):

For all arcs A in the CFG of M exists a path in T that uses A

Well-known Coverage Criteria III

Criterion AllPaths(M,T):

All possible paths ...

© Whenever there is a loop, T is usually infinite!

Variant: AllPaths_k(M,T).

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

- we have again a finite number of paths

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

58

Well-known Coverage Criteria III

Criterion AllPaths(M,T):

All possible paths ...

© Whenever there is a loop, T is usually infinite!

Variant: AllPaths_k(M,T).

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

- we have again a finite number of paths
- ${}^{\textit{\tiny{CP}}}$ the criterion is less constraining than AllTransitions $_k(M,T)$

5/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

Well-known Coverage Criteria III

Criterion AllPaths(M,T):

All possible paths ...

© Whenever there is a loop, T is usually infinite!

Variant: AllPaths $_{k}(M,T)$.

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

- we have again a finite number of paths
- $\ \ \$ the criterion is less constraining than AllTransitions $_{k}(M,T)$

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

58

Well-known Coverage Criteria III

Criterion AllPaths(M,T):

All possible paths ...

© Whenever there is a loop, T is usually infinite!

Variant: AllPaths_k(M,T).

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

- we have again a finite number of paths
- The criterion is less constraining than AllTransitions_k(M,T)

B. Wolff

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

A Hierarchy of Coverage Criteria

□ AllPaths(M,T) ⇒ $AllPaths_k(M,T) \Rightarrow$ AllTransitions(M,T) ⇒ AllInstructions(M,T)

Each of these implications reflects a proper containement; the other way round is never

B. Wolff - Inge 2 V&V - White-Box-Test

59

05/03/18

A Hierarchy of Coverage Criteria

□ AllPaths(M,T) ⇒ $AllPaths_k(M,T) \Rightarrow$ $AllTransitions(M,T) \Rightarrow$ AllInstructions(M,T)

Each of these implications reflects a proper containement; the other way round is never

A Hierarchy of Coverage Criteria

□ AllPaths(M,T) \Rightarrow $AllPaths_k(M,T) \Rightarrow$ $AllTransitions(M,T) \Rightarrow$ AllInstructions(M,T)

Each of these implications reflects a proper containement; the other way round is never

59

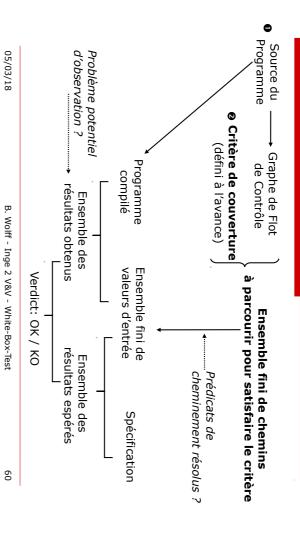
B. Wolff - Inge 2 V&V - White-Box-Test

A Hierarchy of Coverage Criteria

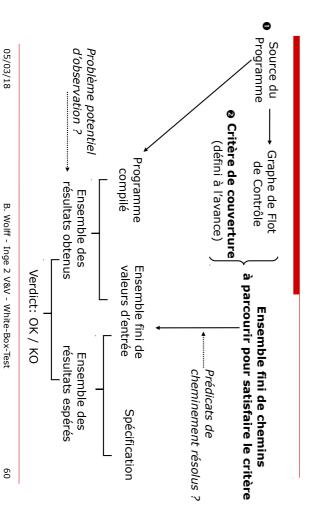
□ AllPaths(M,T) ⇒ $AllPaths_k(M,T) \Rightarrow$ $AllTransitions(M,T) \Rightarrow$ AllInstructions(M,T)

Each of these implications reflects a proper containement; the other way round is never

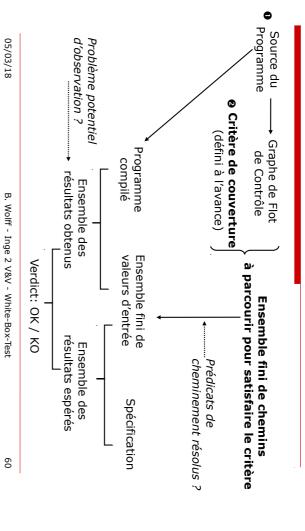
Using Coverage Criteria 1



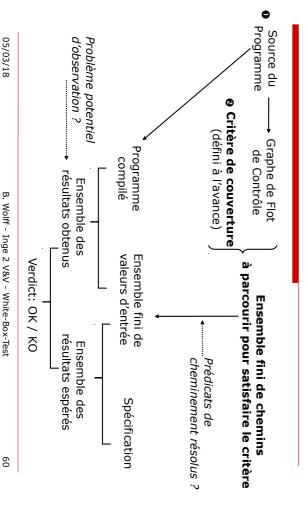
Using Coverage Criteria 1



Using Coverage Criteria 1



Using Coverage Criteria 1



Summary

- We have developed a technique for programbased tests
- ... based on symbolic execution
- ... used in tools like JavaPathFinder-SE or Pex
- Core-Concept:

Feasible Paths in a Control Flow Graph

- Although many theoretical negative results on key properties, good practical approximations are available
- CFG based Coverage Critieria give rise to a Hierarchy

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

61

Summary

- We have developed a technique for programbased tests
- ... based on symbolic execution
- ... used in tools like JavaPathFinder-SE or Pex
- Core-Concept:

Feasible Paths in a Control Flow Graph

- Although many theoretical negative results on key properties, good practical approximations are available
- CFG based Coverage Critieria give rise to a Hierarchy

Summary

- We have developed a technique for programbased tests
- ... based on symbolic execution
- ... used in tools like JavaPathFinder-SE or Pex
- Core-Concept:

Feasible Paths in a Control Flow Graph

- Although many theoretical negative results on key properties, good practical approximations are available
- CFG based Coverage Critieria give rise to a Hierarchy

05/03/18

B. Wolff - Inge 2 V&V - White-Box-Test

61

Summary

- We have developed a technique for programbased tests
- ... based on symbolic execution
- ... used in tools like JavaPathFinder-SE or Pex
- Core-Concept:

Feasible Paths in a Control Flow Graph

- Although many theoretical negative results on key properties, good practical approximations are available
- CFG based Coverage Critieria give rise to a Hierarchy

Summary

```
Int xxxx(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;
}</pre>
```

Summary

Program:

```
int xxxx(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;
}</pre>
```

Summary

Program:

```
int xxxx(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;
}</pre>
```

05/03/18

62

B. Wolff - Inge 2 V&V - White-Box-Test

62

Summary

Program:

```
int xxxx(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;
}</pre>
```

05/03/18

62

05/03/18