2017-18

POLYTECH PARIS-SUD

Cycle Ingénieur – 2ème année

Département Informatique

Verification and Validation

Part IV : White-Box Testing

Burkhart Wolff
Département Informatique
Université Paris-Sud / Orsay

2017-18

POLYTECH PARIS-SUD

Cycle Ingénieur – 2ème année

Département Informatique

Verification and Validation

Part IV : White-Box Testing

Burkhart Wolff
Département Informatique
Université Paris-Sud / Orsay

2017-18

POLYTECH PARIS-SUD

Cycle Ingénieur – 2ème année

Département Informatique

Verification and Validation

Part IV : White-Box Testing

Burkhart Wolff
Département Informatique
Université Paris-Sud / Orsay

2017-18

POLYTECH PARIS-SUD

Cycle Ingénieur – 2ème 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 !

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 2

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 2

Idea:

- Lets exploit the structure of the program !!!

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

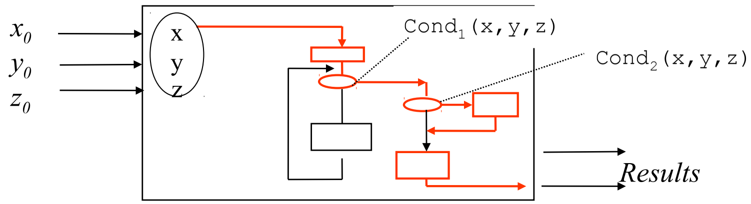
  Assumption: Programmers make most likely errors in branching points of a program (Condition, While-Loop, ...), but get the program "in principle right".
  (Competent programmer assumption)

  Lets develop a test method that exploits this !

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

Static Structural ("white-box") Tests

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



what the program does and how …

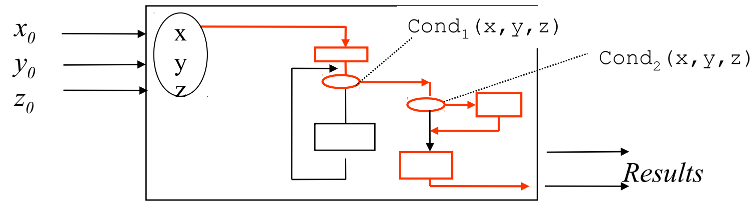
A path corresponds to one logical expression over x_0, y_0, z_0 .
corresponding to one test-case (comprising several test data ...)

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

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

Static Structural ("white-box") Tests

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



what the program does and how …

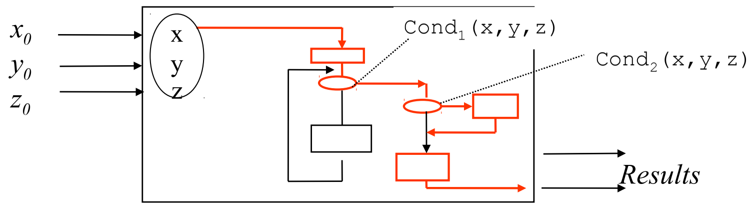
A path corresponds to one logical expression over x_0, y_0, z_0 .
corresponding to one test-case (comprising several test data ...)

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

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

Static Structural ("white-box") Tests

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



what the program does and how …

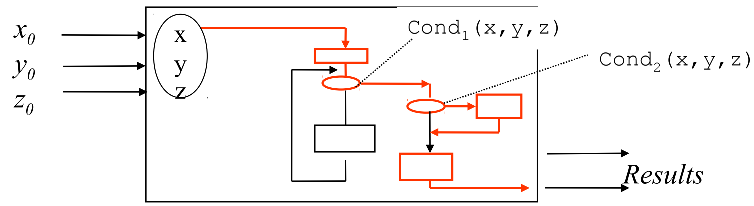
A path corresponds to one logical expression over x_0, y_0, z_0 .
corresponding to one test-case (comprising several test data ...)

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

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

Static Structural ("white-box") Tests

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



what the program does and how …

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

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

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

A Program for the triangle example

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


A Program for the triangle example

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

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 4

A Program for the triangle example

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

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 4

A Program for the triangle example

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

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 4

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)

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 5

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 5

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 5

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:
  for a given specification, there are several possible implementations (working more or less differently from the spec):
  - *sorted arrays : linear search ? binary search ?*
  - *$(x, n) \rightarrow x^n$: successive multiplication ? quadratic multiplication ?*

  Each implementation demands for different test sets !

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:
  for a given specification, there are several possible implementations (working more or less differently from the spec):
  - *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 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:
  for a given specification, there are several possible implementations (working more or less differently from the spec):
  - *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 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:
  for a given specification, there are several possible implementations (working more or less differently from the spec):
  - *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 6

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 !

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 7

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 7

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 7

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.
- the <u>arcs</u> 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

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.
- the <u>arcs</u> 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

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 8

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.
- the <u>arcs</u> 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

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 8

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.
- the <u>arcs</u> 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

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 8

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

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 9

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

Example:

```
S:=1;
P:=N;

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


Computing Control Flow Graphs

- Identify longest sequences of assignments

Example:

```
S:=1;
P:=N;

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

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;

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

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;

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

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;

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


Computing Control Flow Graphs

- Identify longest sequences of assignments

Example:

```
S:=1;
P:=N;

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

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 11

Computing Control Flow Graphs

- Identify longest sequences of assignments

Example:

```
S:=1;
P:=N;

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

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 11

Computing Control Flow Graphs

- Identify longest sequences of assignments

Example:

```
S:=1;
P:=N;

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

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 11

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 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 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
- 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

05/03/18 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 B. Wolff - Inge 2 V&V - White-Box-Test 13

Computing Control Flow Graphs

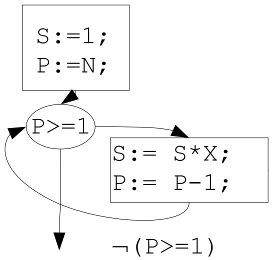
- Identify longest sequences of assignments
- Erase if_then_elses by branching
- Erase while_loops by loop-arc, entry-arc, exit-arc

05/03/18 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

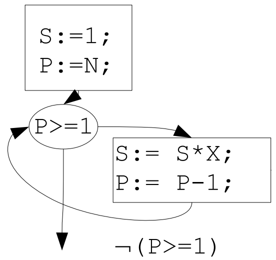
Example:



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

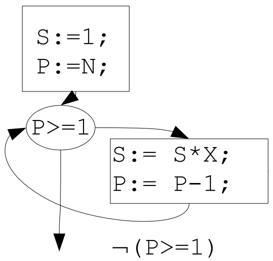
Example:



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

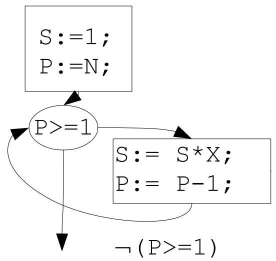
Example:



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

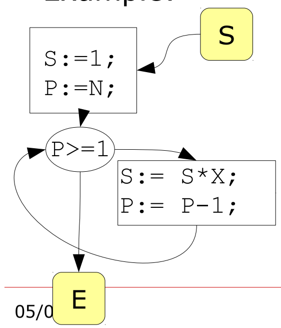
Example:



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

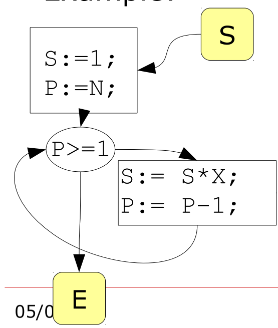
Example:



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

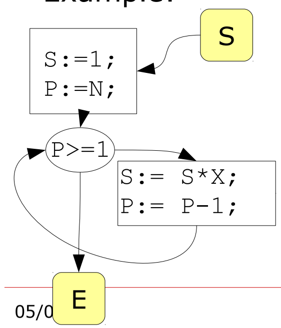
Example:



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

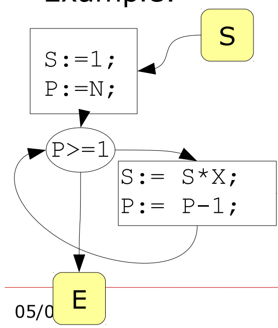
Example:



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:



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 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 B. Wolff - Inge 2 V&V - White-Box-Test 16

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 ...

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


Revisiting our triangle example ...

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

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

Revisiting our triangle example ...

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

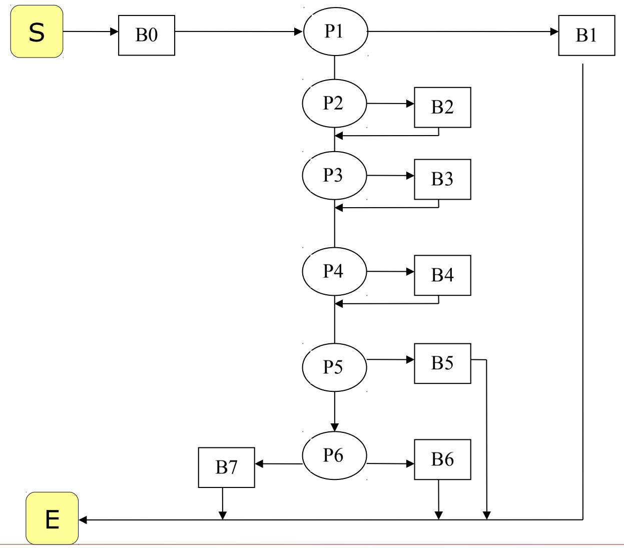
05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 18

Revisiting our triangle example ...

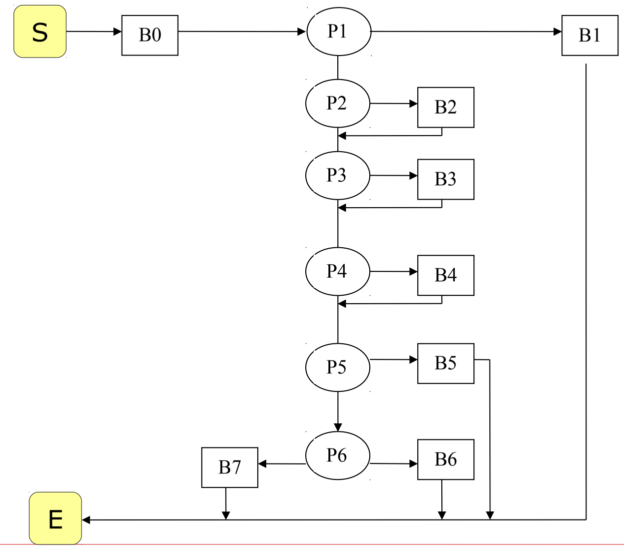
procedure triangle(j,k,l : positive) is
 eg: natural := 0;
begin
if j + k <= l or k + l <= j or l + j <= k then
   put("impossible");
else if j = k then   eg := eg + 1; end if;
     if j = l then   eg := eg + 1; end if;
     if l = k then   eg := eg + 1; end if;
      if eg = 0 then put("quelconque");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

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

The non-structured control-flow graph of a program

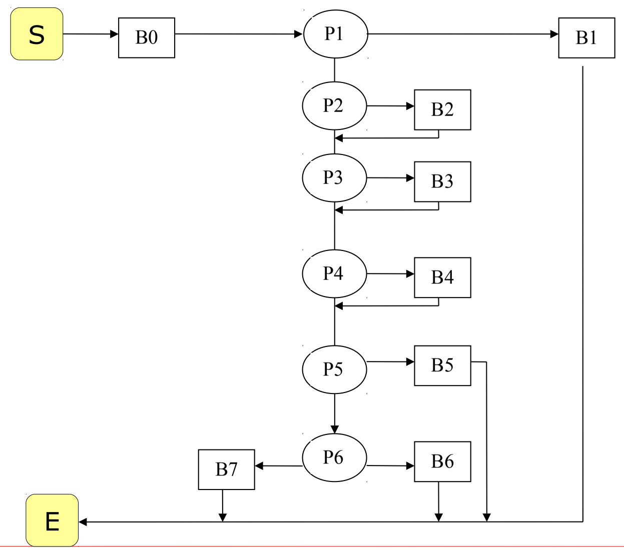


The non-structured control-flow graph of a program



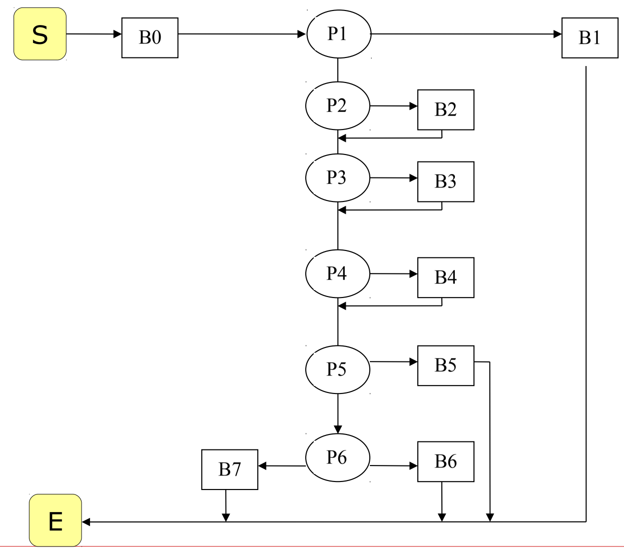
05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 19

The non-structured control-flow graph of a program



05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 19

The non-structured control-flow graph of a program



05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 19

A procedure with loop and return

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


A procedure with loop and return

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

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 20

A procedure with loop and return

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

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 20

A procedure with loop and return

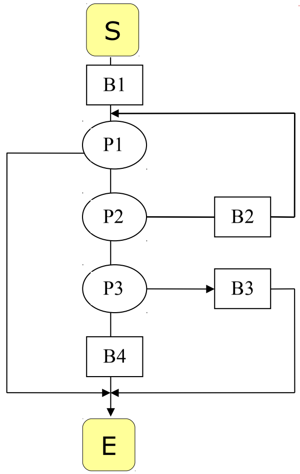
```
procedure supprime (T: in out Table; p: in out integer;
                    x: in integer) is
 i: integer := 1;
begin
   while     i <> p  loop
      if     T[i].val <> x then  i := i + 1;
      elsif  i = p - 1     then  p := p - 1; return;
       else  T[i] := T[p-1]; p := p -1;  return;
     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

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 20

… and its control flow graph

What are the feasible paths ?

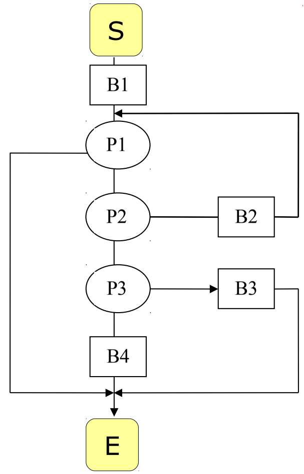
How to describe this ?



… and its control flow graph

What are the feasible paths ?

How to describe this ?

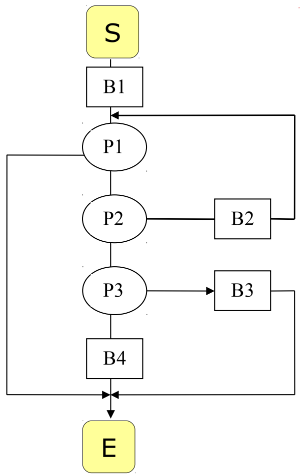


05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 21

… and its control flow graph

What are the feasible paths ?

How to describe this ?

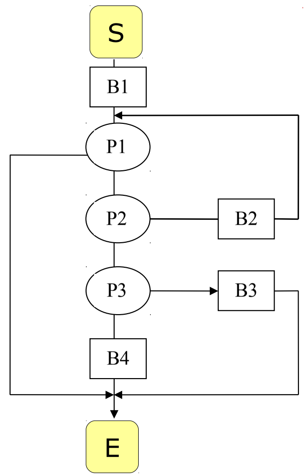


05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 21

… and its control flow graph

What are the feasible paths ?

How to describe this ?



05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 21

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

    i.e. the path condition is satisfiable

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 predicate (over parameters and state): a condition over the initial values initiales of parameters (and global variables) to achieve exactly this execution path
faisable paths = a path of M pour a set for all parameters and global variables exists such that the path is executable.
i.e. the path condition is satisfiable

05/03/18 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 predicate (over parameters and state): a condition over the initial values initiales of parameters (and global variables) to achieve exactly this execution path
faisable paths = a path of M pour a set for all parameters and global variables exists such that the path is executable.
i.e. the path condition is satisfiable

05/03/18 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 predicate (over parameters and state): a condition over the initial values initiales of parameters (and global variables) to achieve exactly this execution path
faisable paths = a path of M pour a set for all parameters and global variables exists such that the path is executable.
i.e. the path condition is satisfiable

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 22

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 x_0, y_0, z_0, ...

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

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

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 x_0, y_0, z_0, ...

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

  if we follow the « true » arc we set $\Phi := \Phi \wedge P(\underline{x},\ldots,\underline{z})$,
  if we follow the «false» arc we set $\Phi := \Phi \wedge \neg P(\underline{x},\ldots,\underline{z})$.
  (The $\underline{x},\ldots,\underline{z}$ 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, ...
- 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, ...

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

  if we follow the « true » arc we set $\Phi := \Phi \wedge P(\underline{x},\ldots,\underline{z})$,
  if we follow the «false» arc we set $\Phi := \Phi \wedge \neg P(\underline{x},\ldots,\underline{z})$.
  (The $\underline{x},\ldots,\underline{z}$ 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, ...
- 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, ...

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

  if we follow the « true » arc we set $\Phi := \Phi \wedge P(\underline{x},\ldots,\underline{z})$,
  if we follow the «false» arc we set $\Phi := \Phi \wedge \neg P(\underline{x},\ldots,\underline{z})$.
  (The $\underline{x},\ldots,\underline{z}$ 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

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.

  $$state = V \to D$$

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

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

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.

  $$state = V \to D$$

  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.

  $$state = V \to D$$

  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.

  $$state = V \to D$$

  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

Symbolic Execution

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

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

  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.

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

  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.

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

  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.

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

  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

- 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).

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).

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).

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 26

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 subse-quently 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:

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

  to

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

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

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 27

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:

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

Symbolic States and Substitutions
Example substitution:
(x + 2 * y) {x ↦ 1, y ↦ x0}
= 1 + 2 * x0
An initial symbolic state is a state of the form:
{ x ↦ x0, y ↦ y0, z ↦ z0 }
05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 28
Symbolic States and Substitutions
Example substitution:
(x + 2 * y) {x ↦ 1, y ↦ x0}
= 1 + 2 * x0
An initial symbolic state is a state of the form:
{ x ↦ x0, y ↦ y0, z ↦ z0 }
05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 28
Symbolic States and Substitutions
Example substitution:
(x + 2 * y) {x ↦ 1, y ↦ x0}
= 1 + 2 * x0
An initial symbolic state is a state of the form:
{ x ↦ x0, y ↦ y0, z ↦ z0 }
05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 28

Basic Blocks *as* Substitutions

Symbolic Pre-State

$x \mapsto x_0$
$y \mapsto y_0+3*x_0$
$z \mapsto z_0$
$i \mapsto i_0$

Block

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

Symbolic Post-State

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

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).

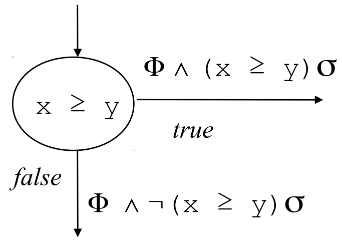
Basic Blocks as Substitutions
Symbolic Pre-State: x ↦ x0; y ↦ y0+3*x0; z ↦ z0; i ↦ i0
Block: i := x+y+1; z := z+i
Symbolic Post-State: x ↦ x0; y ↦ y0+3*x0; z ↦ z0+y0+4*x0+1; i ↦ y0+ 4*x0+1
x0, y0 and z0 represent the initial values of x, y et z.
i is supposed to be a local variable (not initialized at the beginning).
05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 29

Basic Blocks as Substitutions
Symbolic Pre-State: x ↦ x0; y ↦ y0+3*x0; z ↦ z0; i ↦ i0
Block: i := x+y+1; z := z+i
Symbolic Post-State: x ↦ x0; y ↦ y0+3*x0; z ↦ z0+y0+4*x0+1; i ↦ y0+ 4*x0+1
x0, y0 and z0 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: x ↦ x0; y ↦ y0+3*x0; z ↦ z0; i ↦ i0
Block: i := x+y+1; z := z+i
Symbolic Post-State: x ↦ x0; y ↦ y0+3*x0; z ↦ z0+y0+4*x0+1; i ↦ y0+ 4*x0+1
x0, y0 and z0 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

Symbolic Execution

σ : $state_{sym}$

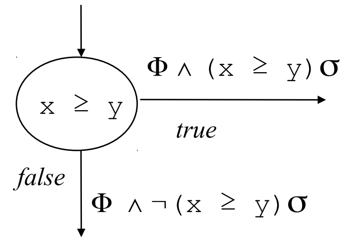


Thus, we execute symbolically and transform the symbolic state in order to obtain an expression depending on the **initial values of the parameters, (accesses to undefined local variables are treated by exception)**

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).

Symbolic Execution

σ : $state_{sym}$



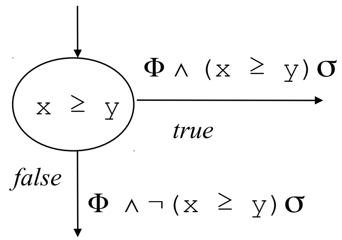
Thus, we execute symbolically and transform the symbolic state in order to obtain an expression depending on the **initial values of the parameters, (accesses to undefined local variables are treated by exception)**

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).

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 30

Symbolic Execution

σ : $state_{sym}$



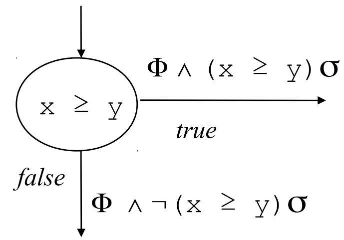
Thus, we execute symbolically and transform the symbolic state in order to obtain an expression depending on the **initial values of the parameters, (accesses to undefined local variables are treated by exception)**

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).

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 30

Symbolic Execution

σ : $state_{sym}$



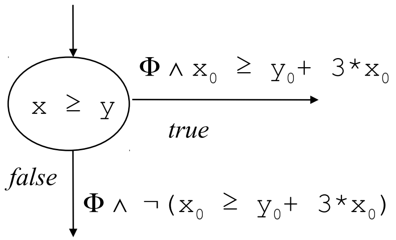
Thus, we execute symbolically and transform the symbolic state in order to obtain an expression depending on the **initial values of the parameters, (accesses to undefined local variables are treated by exception)**

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).

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 30

Symbolic Execution

x	$\mapsto x_0$
y	$\mapsto y_0 + 3*x_0$

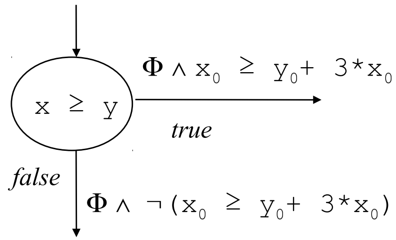


Thus, we execute symbolically and transform the symbolic state in order to obtain an expression depending on the **initial values of the parameters, (accesses to undefined local variables are treated by exception)**

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

x	$\mapsto x_0$
y	$\mapsto y_0 + 3*x_0$



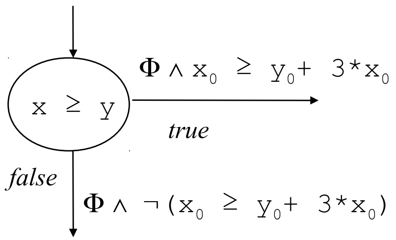
Thus, we execute symbolically and transform the symbolic state in order to obtain an expression depending on the **initial values of the parameters, (accesses to undefined local variables are treated by exception)**

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 31

Symbolic Execution

x	$\mapsto x_0$
y	$\mapsto y_0 + 3*x_0$



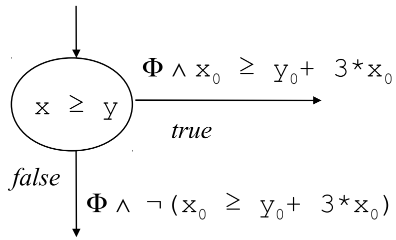
Thus, we execute symbolically and transform the symbolic state in order to obtain an expression depending on the **initial values of the parameters, (accesses to undefined local variables are treated by exception)**

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 31

Symbolic Execution

x	$\mapsto x_0$
y	$\mapsto y_0 + 3*x_0$



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 31

Example: A Symbolic Path Execution

Recall

```
procedure supprime (T: in out Table; p: in out integer;
                    x: in integer) is
 i: integer := 1;
begin
   while      i <> p  loop
      if      T[i] <> x then       i := i + 1;
      elsif   i = p - 1 then       p := p - 1; return;
       else   T[i] := T[p-1];      p := p - 1; return;
      end if;
   end loop;
end supprime;
```


Example: A Symbolic Path Execution

Recall

procedure supprime (T: in out Table; p: in out integer;
                    x: in integer) is
 i: integer := 1;
begin
   while   i <> p loop
      if   T[i] <> x then   i := i + 1;
      elsif   i = p - 1 then   p := p - 1; return;
       else   T[i] := T[p-1];   p := p - 1; return;
      end if;
   end loop;
end supprime;

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 32

Example: A Symbolic Path Execution

Recall

procedure supprime (T: in out Table; p: in out integer;
                    x: in integer) is
 i: integer := 1;
begin
   while   i <> p loop
      if   T[i] <> x then   i := i + 1;
      elsif   i = p - 1 then   p := p - 1; return;
       else   T[i] := T[p-1];   p := p - 1; return;
      end if;
   end loop;
end supprime;

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 32

Example: A Symbolic Path Execution

Recall

procedure supprime (T: in out Table; p: in out integer;
                    x: in integer) is
 i: integer := 1;
begin
   while   i <> p loop
      if   T[i] <> x then   i := i + 1;
      elsif   i = p - 1 then   p := p - 1; return;
       else   T[i] := T[p-1];   p := p - 1; return;
      end if;
   end loop;
end supprime;

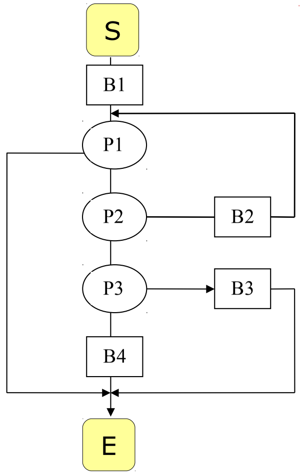
05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 32

Example: A Symbolic Path Execution

... and the corresponding control flow graph.

We want to execute the path:

[S,B1,P1,E]

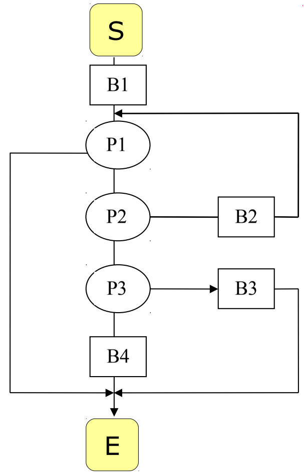


Example: A Symbolic Path Execution

... and the corresponding control flow graph.

We want to execute the path:

[S,B1,P1,E]

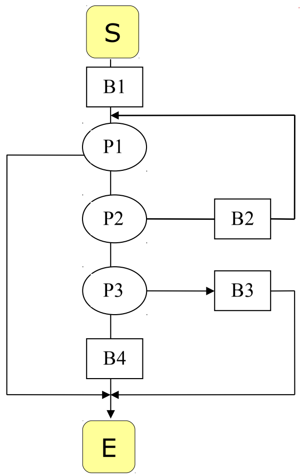


Example: A Symbolic Path Execution

... and the corresponding control flow graph.

We want to execute the path:

[S,B1,P1,E]

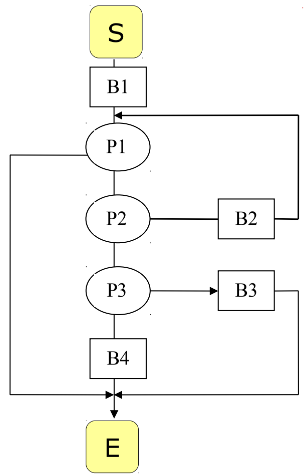


Example: A Symbolic Path Execution

... and the corresponding control flow graph.

We want to execute the path:

[S,B1,P1,E]



Example: A Symbolic Path Execution

We want to execute the path:

[S, B1, P1, E]

Example: A Symbolic Path Execution

We want to execute the path:

[S, B1, P1, E]

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 34

Example: A Symbolic Path Execution

We want to execute the path:

[S, B1, P1, E]

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 34

Example: A Symbolic Path Execution

We want to execute the path:

[S, B1, P1, E]

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 34

Example: A Symbolic Path Execution

We want to execute the path:

[S, B1, P1, E]

Φ := True

$T \mapsto T_0$
$p \mapsto p_0$
$x \mapsto X_0$
$i \mapsto i_0$

Example: A Symbolic Path Execution
We want to execute the path:
[S, B1, P1, E]
Φ := True
T ↦ T0
p ↦ p0
x ↦ X0
i ↦ i0
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, B1, P1, E]
Φ := True
T ↦ T0
p ↦ p0
x ↦ X0
i ↦ i0
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, B1, P1, E]
Φ := True
T ↦ T0
p ↦ p0
x ↦ X0
i ↦ i0
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, B1, P1, E]

Φ := True

$T \mapsto T_0$
$p \mapsto p_0$
$x \mapsto X_0$
$i \mapsto i_0$

Φ := True

$T \mapsto T_0$
$p \mapsto p_0$
$x \mapsto X_0$
$i \mapsto 1$

Example: A Symbolic Path Execution

We want to execute the path:

[S, B1, P1, E]

Φ := True

$T \mapsto T_0$
$p \mapsto p_0$
$x \mapsto X_0$
$i \mapsto i_0$

Φ := True

$T \mapsto T_0$
$p \mapsto p_0$
$x \mapsto X_0$
$i \mapsto 1$

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, B1, P1, E]

Φ := True

$T \mapsto T_0$
$p \mapsto p_0$
$x \mapsto X_0$
$i \mapsto i_0$

Φ := True

$T \mapsto T_0$
$p \mapsto p_0$
$x \mapsto X_0$
$i \mapsto 1$

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, B1, P1, E]

Φ := True

$T \mapsto T_0$
$p \mapsto p_0$
$x \mapsto X_0$
$i \mapsto i_0$

Φ := True

$T \mapsto T_0$
$p \mapsto p_0$
$x \mapsto X_0$
$i \mapsto 1$

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, B1, P1, E]

$\Phi := True$	$\Phi := True$	$\Phi := \neg(i<>p)\,\sigma_{B1}$
$T \mapsto T_0$	$T \mapsto T_0$	$T \mapsto T_0$
$p \mapsto p_0$	$p \mapsto p_0$	$p \mapsto p_0$
$x \mapsto X_0$	$x \mapsto X_0$	$x \mapsto X_0$
$i \mapsto i_0$	$i \mapsto 1$	$i \mapsto 1$

Example: A Symbolic Path Execution

We want to execute the path:

[S, B1, P1, E]

$\Phi := True$	$\Phi := True$	$\Phi := \neg(i<>p)\,\sigma_{B1}$
$T \mapsto T_0$	$T \mapsto T_0$	$T \mapsto T_0$
$p \mapsto p_0$	$p \mapsto p_0$	$p \mapsto p_0$
$x \mapsto X_0$	$x \mapsto X_0$	$x \mapsto X_0$
$i \mapsto i_0$	$i \mapsto 1$	$i \mapsto 1$

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 37

Example: A Symbolic Path Execution

We want to execute the path:

[S, B1, P1, E]

$\Phi := True$	$\Phi := True$	$\Phi := \neg(i<>p)\,\sigma_{B1}$
$T \mapsto T_0$	$T \mapsto T_0$	$T \mapsto T_0$
$p \mapsto p_0$	$p \mapsto p_0$	$p \mapsto p_0$
$x \mapsto X_0$	$x \mapsto X_0$	$x \mapsto X_0$
$i \mapsto i_0$	$i \mapsto 1$	$i \mapsto 1$

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 37

Example: A Symbolic Path Execution

We want to execute the path:

[S, B1, P1, E]

$\Phi := True$	$\Phi := True$	$\Phi := \neg(i<>p)\,\sigma_{B1}$
$T \mapsto T_0$	$T \mapsto T_0$	$T \mapsto T_0$
$p \mapsto p_0$	$p \mapsto p_0$	$p \mapsto p_0$
$x \mapsto X_0$	$x \mapsto X_0$	$x \mapsto X_0$
$i \mapsto i_0$	$i \mapsto 1$	$i \mapsto 1$

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 37

Example: A Symbolic Path Execution

We want to execute the path:

[S, B1, P1, E]

$\Phi := True$	$\Phi := True$	$\Phi := \neg(i<>p)\sigma_{B1}$	$\Phi := p_0 = 1$
$T \mapsto T_0$	$T \mapsto T_0$	$T \mapsto T_0$	$T \mapsto T_0$
$p \mapsto p_0$	$p \mapsto p_0$	$p \mapsto p_0$	$p \mapsto p_0$
$x \mapsto X_0$	$x \mapsto X_0$	$x \mapsto X_0$	$x \mapsto X_0$
$i \mapsto i_0$	$i \mapsto 1$	$i \mapsto 1$	$i \mapsto 1$

Example: A Symbolic Path Execution

We want to execute the path:

[S, B1, P1, E]

$\Phi := True$	$\Phi := True$	$\Phi := \neg(i<>p)\sigma_{B1}$	$\Phi := p_0 = 1$
$T \mapsto T_0$	$T \mapsto T_0$	$T \mapsto T_0$	$T \mapsto T_0$
$p \mapsto p_0$	$p \mapsto p_0$	$p \mapsto p_0$	$p \mapsto p_0$
$x \mapsto X_0$	$x \mapsto X_0$	$x \mapsto X_0$	$x \mapsto X_0$
$i \mapsto i_0$	$i \mapsto 1$	$i \mapsto 1$	$i \mapsto 1$

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 38

Example: A Symbolic Path Execution

We want to execute the path:

[S, B1, P1, E]

$\Phi := True$	$\Phi := True$	$\Phi := \neg(i<>p)\sigma_{B1}$	$\Phi := p_0 = 1$
$T \mapsto T_0$	$T \mapsto T_0$	$T \mapsto T_0$	$T \mapsto T_0$
$p \mapsto p_0$	$p \mapsto p_0$	$p \mapsto p_0$	$p \mapsto p_0$
$x \mapsto X_0$	$x \mapsto X_0$	$x \mapsto X_0$	$x \mapsto X_0$
$i \mapsto i_0$	$i \mapsto 1$	$i \mapsto 1$	$i \mapsto 1$

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 38

Example: A Symbolic Path Execution

We want to execute the path:

[S, B1, P1, E]

$\Phi := True$	$\Phi := True$	$\Phi := \neg(i<>p)\sigma_{B1}$	$\Phi := p_0 = 1$
$T \mapsto T_0$	$T \mapsto T_0$	$T \mapsto T_0$	$T \mapsto T_0$
$p \mapsto p_0$	$p \mapsto p_0$	$p \mapsto p_0$	$p \mapsto p_0$
$x \mapsto X_0$	$x \mapsto X_0$	$x \mapsto X_0$	$x \mapsto X_0$
$i \mapsto i_0$	$i \mapsto 1$	$i \mapsto 1$	$i \mapsto 1$

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 38

Example: A Symbolic Path Execution

Result:

Test-Case:

Path : [S,B1,P1,E]

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

A concrete Test, satisfying Φ

T	↦	mtTab
p	↦	1
x	↦	17

Example: A Symbolic Path Execution

Result:

Test-Case:

Path : [S,B1,P1,E]

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

A concrete Test, satisfying Φ

T	↦	mtTab
p	↦	1
x	↦	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: $\Phi := p_0 = 1$

A concrete Test, satisfying Φ

T	↦	mtTab
p	↦	1
x	↦	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: $\Phi := p_0 = 1$

A concrete Test, satisfying Φ

T	↦	mtTab
p	↦	1
x	↦	17

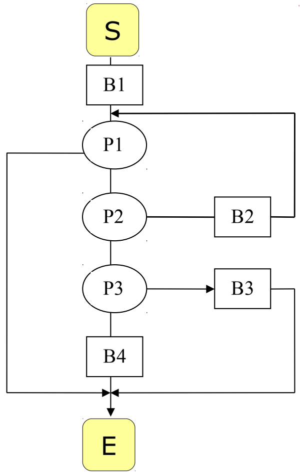
05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 39

Example: A Symbolic Path Execution

... and the corresponding control flow graph.

We want to execute the path:

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

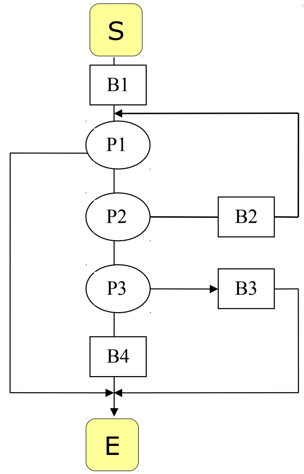


Example: A Symbolic Path Execution

... and the corresponding control flow graph.

We want to execute the path:

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



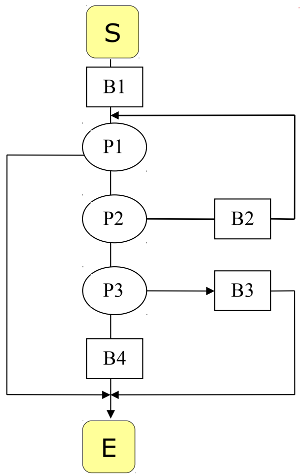
05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 40

Example: A Symbolic Path Execution

... and the corresponding control flow graph.

We want to execute the path:

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



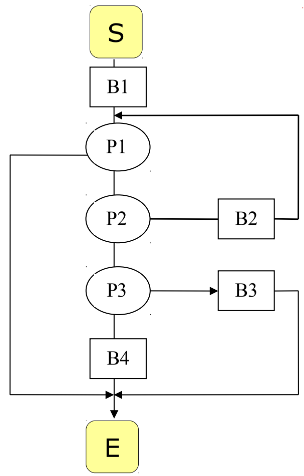
05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 40

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 B. Wolff - Inge 2 V&V - White-Box-Test 40

Example: A Symbolic Path Execution

We want to execute the path:

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

$\Phi \mapsto$ True	True					
$T \mapsto T_0$	T_0					
$p \mapsto p_0$	p_0					
$x \mapsto X_0$	X_0					
$i \mapsto i_0$	1					

Example: A Symbolic Path Execution

We want to execute the path:

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

$\Phi \mapsto$ True	True					
$T \mapsto T_0$	T_0					
$p \mapsto p_0$	p_0					
$x \mapsto X_0$	X_0					
$i \mapsto i_0$	1					

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 41

Example: A Symbolic Path Execution

We want to execute the path:

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

$\Phi \mapsto$ True	True					
$T \mapsto T_0$	T_0					
$p \mapsto p_0$	p_0					
$x \mapsto X_0$	X_0					
$i \mapsto i_0$	1					

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 41

Example: A Symbolic Path Execution

We want to execute the path:

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

$\Phi \mapsto$ True	True					
$T \mapsto T_0$	T_0					
$p \mapsto p_0$	p_0					
$x \mapsto X_0$	X_0					
$i \mapsto i_0$	1					

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 41

Example: A Symbolic Path Execution

We want to execute the path:

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

$\Phi \mapsto$ True	True	$(i<>p)\,\sigma_{B1} \equiv p_0 \neq 1$				
$T \mapsto T_0$	T_0	T_0				
$p \mapsto p_0$	p_0	p_0				
$x \mapsto X_0$	X_0	X_0				
$i \mapsto i_0$	1	1				

Example: A Symbolic Path Execution

We want to execute the path:

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

$\Phi \mapsto$ True	True	$(i<>p)\,\sigma_{B1} \equiv p_0 \neq 1$				
$T \mapsto T_0$	T_0	T_0				
$p \mapsto p_0$	p_0	p_0				
$x \mapsto X_0$	X_0	X_0				
$i \mapsto i_0$	1	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]

$\Phi \mapsto$ True	True	$(i<>p)\,\sigma_{B1} \equiv p_0 \neq 1$				
$T \mapsto T_0$	T_0	T_0				
$p \mapsto p_0$	p_0	p_0				
$x \mapsto X_0$	X_0	X_0				
$i \mapsto i_0$	1	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]

$\Phi \mapsto$ True	True	$(i<>p)\,\sigma_{B1} \equiv p_0 \neq 1$				
$T \mapsto T_0$	T_0	T_0				
$p \mapsto p_0$	p_0	p_0				
$x \mapsto X_0$	X_0	X_0				
$i \mapsto i_0$	1	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]

$\Phi \mapsto$ True	True	$(i<>p)\sigma_{B1} \equiv p_0 \neq 1$	$p_0 \neq 1 \wedge (T[i] <> x)\sigma_{B1}$			
$T \mapsto T_0$	T_0	T_0	T_0			
$p \mapsto p_0$	p_0	p_0	p_0			
$x \mapsto X_0$	X_0	X_0	X_0			
$i \mapsto i_0$	1	1	1			

Example: A Symbolic Path Execution

We want to execute the path:

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

$\Phi \mapsto$ True	True	$(i<>p)\sigma_{B1} \equiv p_0 \neq 1$	$p_0 \neq 1 \wedge (T[i] <> x)\sigma_{B1}$			
$T \mapsto T_0$	T_0	T_0	T_0			
$p \mapsto p_0$	p_0	p_0	p_0			
$x \mapsto X_0$	X_0	X_0	X_0			
$i \mapsto i_0$	1	1	1			

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, P1, P2, B2, P1, E]

$\Phi \mapsto$ True	True	$(i<>p)\sigma_{B1} \equiv p_0 \neq 1$	$p_0 \neq 1 \wedge (T[i] <> x)\sigma_{B1}$			
$T \mapsto T_0$	T_0	T_0	T_0			
$p \mapsto p_0$	p_0	p_0	p_0			
$x \mapsto X_0$	X_0	X_0	X_0			
$i \mapsto i_0$	1	1	1			

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, P1, P2, B2, P1, E]

$\Phi \mapsto$ True	True	$(i<>p)\sigma_{B1} \equiv p_0 \neq 1$	$p_0 \neq 1 \wedge (T[i] <> x)\sigma_{B1}$			
$T \mapsto T_0$	T_0	T_0	T_0			
$p \mapsto p_0$	p_0	p_0	p_0			
$x \mapsto X_0$	X_0	X_0	X_0			
$i \mapsto i_0$	1	1	1			

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 43

Example: A Symbolic Path Execution

We want to execute the path: Φ Φ

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

S	B1	P1	P2	B2	P1	E
$\Phi \mapsto$ True	True	$(i<>p)\sigma_{B1} \equiv p_0 \neq 1$	$p_0 \neq 1 \wedge (T[i] <> x)\sigma_{B1}$	$p_0 \neq 1 \wedge T_0[1] \neq X_0$		
$T \mapsto T_0$	T_0	T_0	T_0	T_0		
$p \mapsto p_0$	p_0	p_0	p_0	p_0		
$x \mapsto X_0$	X_0	X_0	X_0	X_0		
$i \mapsto i_0$	1	1	1	$(i+1)\sigma_{B1}$		

Example: A Symbolic Path Execution

We want to execute the path: Φ Φ

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

S	B1	P1	P2	B2	P1	E
$\Phi \mapsto$ True	True	$(i<>p)\sigma_{B1} \equiv p_0 \neq 1$	$p_0 \neq 1 \wedge (T[i] <> x)\sigma_{B1}$	$p_0 \neq 1 \wedge T_0[1] \neq X_0$		
$T \mapsto T_0$	T_0	T_0	T_0	T_0		
$p \mapsto p_0$	p_0	p_0	p_0	p_0		
$x \mapsto X_0$	X_0	X_0	X_0	X_0		
$i \mapsto i_0$	1	1	1	$(i+1)\sigma_{B1}$		

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 44

Example: A Symbolic Path Execution

We want to execute the path: Φ Φ

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

S	B1	P1	P2	B2	P1	E
$\Phi \mapsto$ True	True	$(i<>p)\sigma_{B1} \equiv p_0 \neq 1$	$p_0 \neq 1 \wedge (T[i] <> x)\sigma_{B1}$	$p_0 \neq 1 \wedge T_0[1] \neq X_0$		
$T \mapsto T_0$	T_0	T_0	T_0	T_0		
$p \mapsto p_0$	p_0	p_0	p_0	p_0		
$x \mapsto X_0$	X_0	X_0	X_0	X_0		
$i \mapsto i_0$	1	1	1	$(i+1)\sigma_{B1}$		

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 44

Example: A Symbolic Path Execution

We want to execute the path: Φ Φ

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

S	B1	P1	P2	B2	P1	E
$\Phi \mapsto$ True	True	$(i<>p)\sigma_{B1} \equiv p_0 \neq 1$	$p_0 \neq 1 \wedge (T[i] <> x)\sigma_{B1}$	$p_0 \neq 1 \wedge T_0[1] \neq X_0$		
$T \mapsto T_0$	T_0	T_0	T_0	T_0		
$p \mapsto p_0$	p_0	p_0	p_0	p_0		
$x \mapsto X_0$	X_0	X_0	X_0	X_0		
$i \mapsto i_0$	1	1	1	$(i+1)\sigma_{B1}$		

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 44

Example: A Symbolic Path Execution

We want to execute the path:

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

$\Phi \mapsto$ True	True	$(i<>p)\sigma_{B1} \equiv p_0 \neq 1$	$p_0\neq 1 \wedge (T[i] <> x)\sigma_{B1}$	$p_0\neq 1 \wedge T_0[1]\neq X_0$	$p_0\neq 1 \wedge T_0[1] \neq X_0 \wedge\neg(i<>p)\sigma_{B2}$	
$T \mapsto T_0$	T_0	T_0	T_0	T_0	T_0	
$p \mapsto p_0$	p_0	p_0	p_0	p_0	p_0	
$x \mapsto X_0$	X_0	X_0	X_0	X_0	X_0	
$i \mapsto i_0$	1	1	1	$(i+1)\sigma_{B1}$	2	

Example: A Symbolic Path Execution

We want to execute the path:

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

$\Phi \mapsto$ True	True	$(i<>p)\sigma_{B1} \equiv p_0 \neq 1$	$p_0\neq 1 \wedge (T[i] <> x)\sigma_{B1}$	$p_0\neq 1 \wedge T_0[1]\neq X_0$	$p_0\neq 1 \wedge T_0[1] \neq X_0 \wedge\neg(i<>p)\sigma_{B2}$	
$T \mapsto T_0$	T_0	T_0	T_0	T_0	T_0	
$p \mapsto p_0$	p_0	p_0	p_0	p_0	p_0	
$x \mapsto X_0$	X_0	X_0	X_0	X_0	X_0	
$i \mapsto i_0$	1	1	1	$(i+1)\sigma_{B1}$	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:

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

$\Phi \mapsto$ True	True	$(i<>p)\sigma_{B1} \equiv p_0 \neq 1$	$p_0\neq 1 \wedge (T[i] <> x)\sigma_{B1}$	$p_0\neq 1 \wedge T_0[1]\neq X_0$	$p_0\neq 1 \wedge T_0[1] \neq X_0 \wedge\neg(i<>p)\sigma_{B2}$	
$T \mapsto T_0$	T_0	T_0	T_0	T_0	T_0	
$p \mapsto p_0$	p_0	p_0	p_0	p_0	p_0	
$x \mapsto X_0$	X_0	X_0	X_0	X_0	X_0	
$i \mapsto i_0$	1	1	1	$(i+1)\sigma_{B1}$	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:

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

$\Phi \mapsto$ True	True	$(i<>p)\sigma_{B1} \equiv p_0 \neq 1$	$p_0\neq 1 \wedge (T[i] <> x)\sigma_{B1}$	$p_0\neq 1 \wedge T_0[1]\neq X_0$	$p_0\neq 1 \wedge T_0[1] \neq X_0 \wedge\neg(i<>p)\sigma_{B2}$	
$T \mapsto T_0$	T_0	T_0	T_0	T_0	T_0	
$p \mapsto p_0$	p_0	p_0	p_0	p_0	p_0	
$x \mapsto X_0$	X_0	X_0	X_0	X_0	X_0	
$i \mapsto i_0$	1	1	1	$(i+1)\sigma_{B1}$	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:

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

$\Phi \mapsto$ True	True	$(i<>p)\sigma_{B1} \equiv p_0 \neq 1$	$p_0 \neq 1 \wedge (T[i] <> x)\sigma_{B1}$	$p_0 \neq 1 \wedge T_0[1] \neq X_0$	$p_0 \neq 1 \wedge T_0[1] \neq X_0 \wedge \neg (i<>p)\sigma_{B2}$	$p_0 \neq 1 \wedge T_0[1] \neq X_0 \wedge p_0 = 2$
$T \mapsto T_0$	T_0	T_0	T_0	T_0	T_0	T_0
$p \mapsto p_0$	p_0	p_0	p_0	p_0	p_0	p_0
$x \mapsto X_0$	X_0	X_0	X_0	X_0	X_0	X_0
$i \mapsto i_0$	1	1	1	$(i+1)\sigma_{B1}$	2	2

Example: A Symbolic Path Execution

We want to execute the path:

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

$\Phi \mapsto$ True	True	$(i<>p)\sigma_{B1} \equiv p_0 \neq 1$	$p_0 \neq 1 \wedge (T[i] <> x)\sigma_{B1}$	$p_0 \neq 1 \wedge T_0[1] \neq X_0$	$p_0 \neq 1 \wedge T_0[1] \neq X_0 \wedge \neg (i<>p)\sigma_{B2}$	$p_0 \neq 1 \wedge T_0[1] \neq X_0 \wedge p_0 = 2$
$T \mapsto T_0$	T_0	T_0	T_0	T_0	T_0	T_0
$p \mapsto p_0$	p_0	p_0	p_0	p_0	p_0	p_0
$x \mapsto X_0$	X_0	X_0	X_0	X_0	X_0	X_0
$i \mapsto i_0$	1	1	1	$(i+1)\sigma_{B1}$	2	2

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 46

Example: A Symbolic Path Execution

We want to execute the path:

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

$\Phi \mapsto$ True	True	$(i<>p)\sigma_{B1} \equiv p_0 \neq 1$	$p_0 \neq 1 \wedge (T[i] <> x)\sigma_{B1}$	$p_0 \neq 1 \wedge T_0[1] \neq X_0$	$p_0 \neq 1 \wedge T_0[1] \neq X_0 \wedge \neg (i<>p)\sigma_{B2}$	$p_0 \neq 1 \wedge T_0[1] \neq X_0 \wedge p_0 = 2$
$T \mapsto T_0$	T_0	T_0	T_0	T_0	T_0	T_0
$p \mapsto p_0$	p_0	p_0	p_0	p_0	p_0	p_0
$x \mapsto X_0$	X_0	X_0	X_0	X_0	X_0	X_0
$i \mapsto i_0$	1	1	1	$(i+1)\sigma_{B1}$	2	2

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 46

Example: A Symbolic Path Execution

We want to execute the path:

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

$\Phi \mapsto$ True	True	$(i<>p)\sigma_{B1} \equiv p_0 \neq 1$	$p_0 \neq 1 \wedge (T[i] <> x)\sigma_{B1}$	$p_0 \neq 1 \wedge T_0[1] \neq X_0$	$p_0 \neq 1 \wedge T_0[1] \neq X_0 \wedge \neg (i<>p)\sigma_{B2}$	$p_0 \neq 1 \wedge T_0[1] \neq X_0 \wedge p_0 = 2$
$T \mapsto T_0$	T_0	T_0	T_0	T_0	T_0	T_0
$p \mapsto p_0$	p_0	p_0	p_0	p_0	p_0	p_0
$x \mapsto X_0$	X_0	X_0	X_0	X_0	X_0	X_0
$i \mapsto i_0$	1	1	1	$(i+1)\sigma_{B1}$	2	2

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 46

Example: A Symbolic Path Execution

We want to execute the path:

[S,	B1,	P1,	P2,	B2,	P1,	E]
$\Phi \mapsto$ True	True	$(i<>p)\sigma_{B1} \equiv p_0 \neq 1$	$p_0 \neq 1 \wedge (T[i] <> x)\sigma_{B1}$	$p_0 \neq 1 \wedge T_0[1] \neq X_0$	$p_0 \neq 1 \wedge T_0[1] \neq X_0 \wedge \neg (i<>p)\sigma_{B2}$	$p_0 \neq 1 \wedge T_0[1] \neq X_0 \wedge p_0 = 2$
$T \mapsto T_0$	T_0	T_0	T_0	T_0	T_0	T_0
$p \mapsto p_0$	p_0	p_0	p_0	p_0	p_0	p_0
$x \mapsto X_0$	X_0	X_0	X_0	X_0	X_0	X_0
$i \mapsto i_0$	1	1	1	$(i+1)\sigma_{B1}$	2	2

Example: A Symbolic Path Execution

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 47

Example: A Symbolic Path Execution

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 47

Example: A Symbolic Path Execution

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 47

Example: A Symbolic Path Execution

Result:

Test-Case:

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

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

A concrete Test, satisfying Φ

T	↦	[3]
p	↦	2
x	↦	17

Example: A Symbolic Path Execution

Result:

Test-Case:

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

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

A concrete Test, satisfying Φ

T	↦	[3]
p	↦	2
x	↦	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: $\Phi := T_0[1] \neq X_0 \wedge p_0 = 2$

A concrete Test, satisfying Φ

T	↦	[3]
p	↦	2
x	↦	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: $\Phi := T_0[1] \neq X_0 \wedge p_0 = 2$

A concrete Test, satisfying Φ

T	↦	[3]
p	↦	2
x	↦	17

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 48

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)

- 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.

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.

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 49

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 ...
- **...**

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

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

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 ...
- **...**

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

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

05/03/18 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 ...
- **...**

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

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

05/03/18 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 ...
- **...**

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

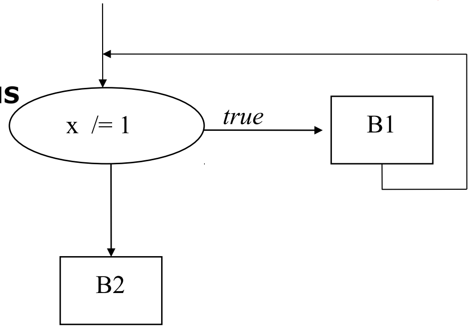
BUT: for many relevant programs, practically good solutions exist (Z3, Simplify, CVC4, AltErgo ...)

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 50

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;

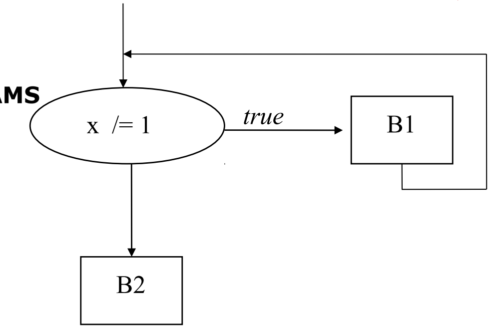


- does this function terminate for all x ?
- or equivalently: is B2 reached for all x ?

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;

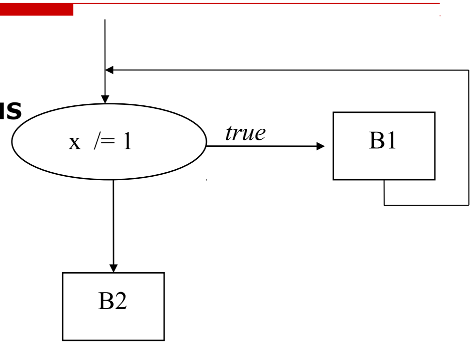


- does this function terminate for all x ?
- or equivalently: is B2 reached for all x ?

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;

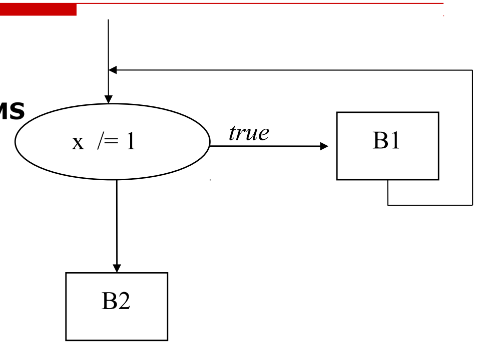


- does this function terminate for all x ?
- or equivalently: is B2 reached for all x ?

A Challange-Example (Collatz-Function):

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

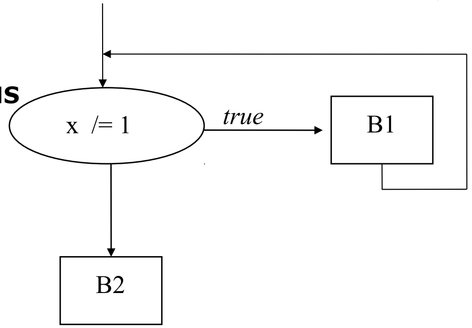
while x <> 1 loop
if pair(x) then x := x / 2;
else x := 3 * x + 1;
end if;
end loop;



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

A Challange-Example (Collatz-Function):

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

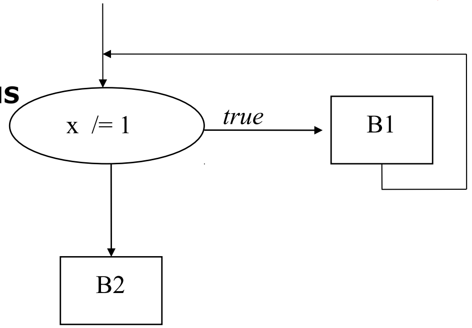



```
while x <> 1 loop
      if pair(x) then x := x / 2;
      else x := 3 * x + 1;
      end if;
 end loop;
```

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

A Challange-Example (Collatz-Function):

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



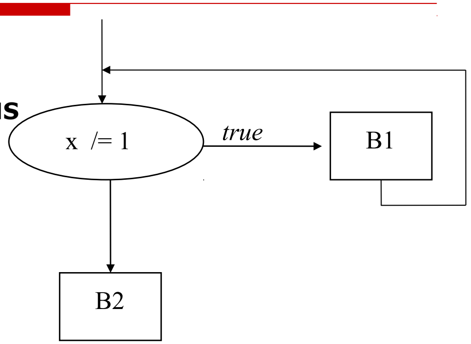
```
while x <> 1 loop
      if pair(x) then x := x / 2;
      else x := 3 * x + 1;
      end if;
 end loop;
```

- 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):

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



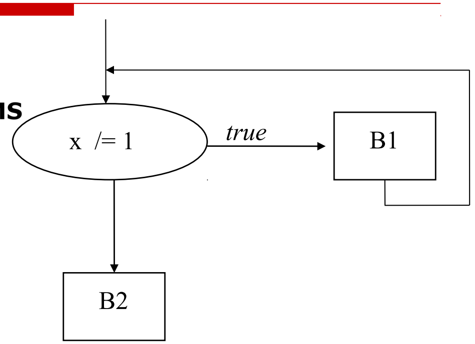
```
while x <> 1 loop
      if pair(x) then x := x / 2;
      else x := 3 * x + 1;
      end if;
 end loop;
```

- 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):

... 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;
```

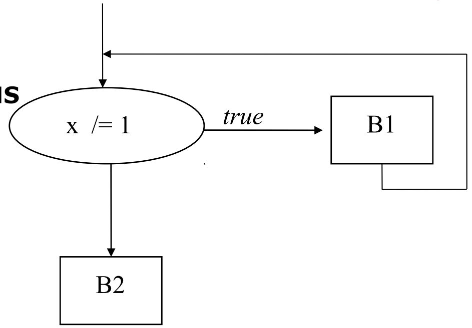
- does this function terminate for all x ?
- or equivalently: is B2 reached for all x ? ANSWER:unknown

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

A Challange-Example (Collatz-Function):

... **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;

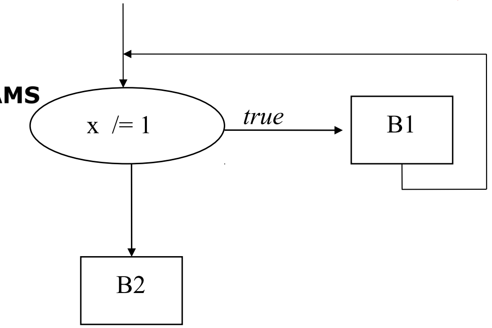


- does this function terminate for all x ?
- or equivalently: is B2 reached for all x ? ANSWER:unknown
- this implies that we can not know in advance that there exist infeasible paths !

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;



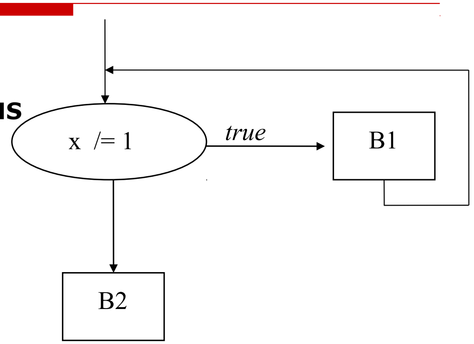
- does this function terminate for all x ?
- or equivalently: is B2 reached for all x ? ANSWER:unknown
- this implies that we can not know in advance that there exist infeasible paths !

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;



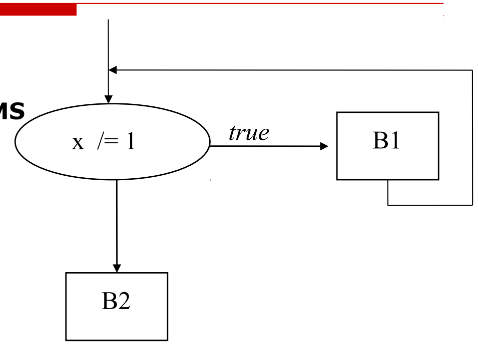
- does this function terminate for all x ?
- or equivalently: is B2 reached for all x ? ANSWER:unknown
- this implies that we can not know in advance that there exist infeasible paths !

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;



- 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

The Triangle Prog without Unfeasible Paths

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

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

The Triangle Prog without Unfeasible Paths

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

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

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 54

The Triangle Prog without Unfeasible Paths

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

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

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 54

The Triangle Prog without Unfeasible Paths

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

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

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 54

The notion of a "couverage criteria"

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

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
- …

The notion of a "couverage criteria"

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

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
- …

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 55

The notion of a "couverage criteria"

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

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
- …

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 55

The notion of a "couverage criteria"

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

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
- …

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 55

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 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

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

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 56

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

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

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 57

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)

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)

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)

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 58

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 true.

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 true.

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 59

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 true.

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 59

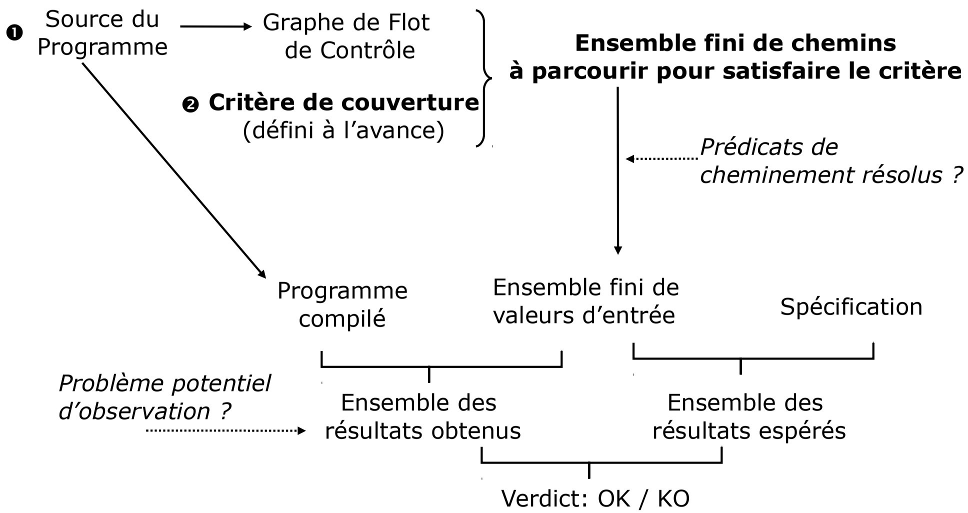
A Hierarchy of Coverage Criteria

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

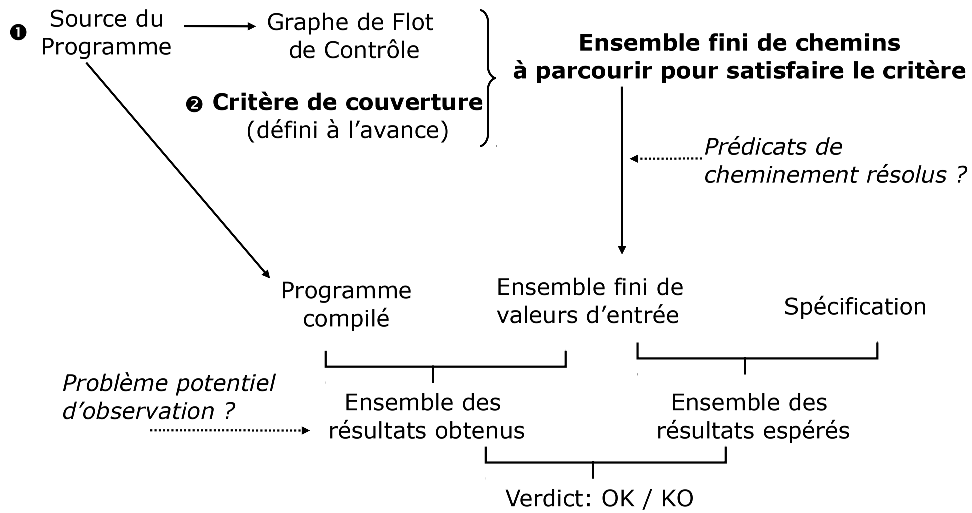
- Each of these implications reflects a proper containement; the other way round is never true.

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 59

Using Coverage Criteria 1

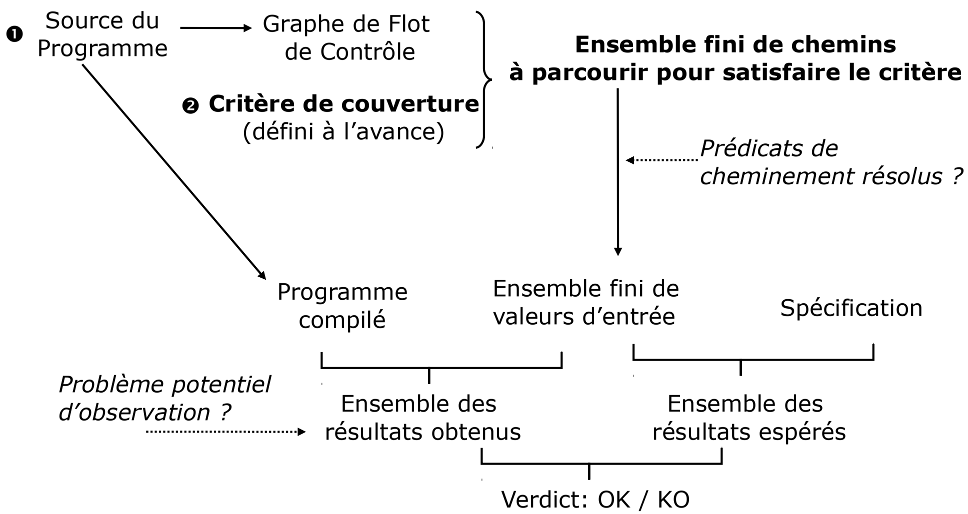


Using Coverage Criteria 1



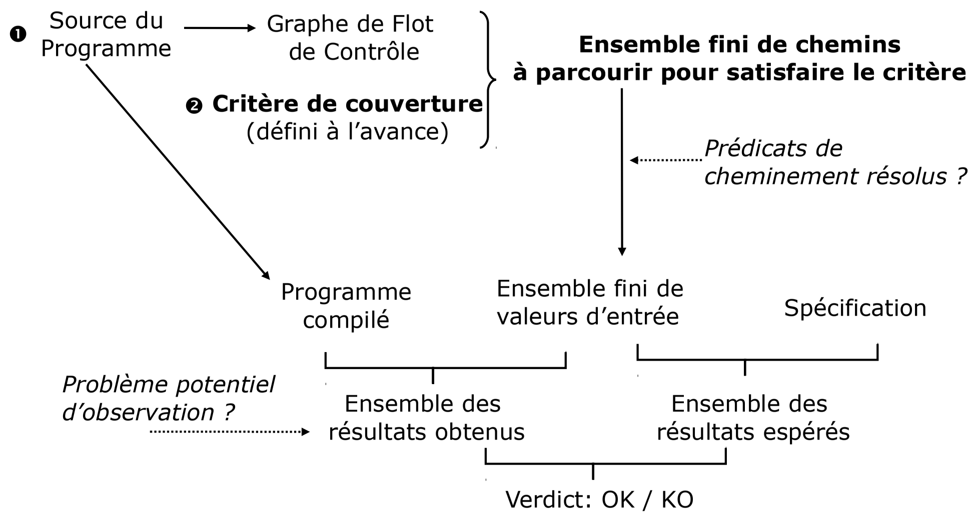
05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 60

Using Coverage Criteria 1



05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 60

Using Coverage Criteria 1



05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 60

Summary

- We have developed a technique for program-based 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 program-based 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 program-based 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 program-based 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

- 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;
}
```


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;
}
```

05/03/18 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;
}
```

05/03/18 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;
}
```

05/03/18 B. Wolff - Inge 2 V&V - White-Box-Test 62