Universiteit Antwerpen
Faculteit Wetenschappen

White-box testing

Kasper Engelen

Overview

White-box testing

- **Basis path testing**
  - Every node covered
- **Condition testing:**
  - Multiple condition coverage
  - “a > b && b > c*2”
  - Make “a > b” true and false
  - Make “b > c*2” true and false
- **Loop testing**
  - 1 iteration
  - 2 iterations
  - …
  - n iterations

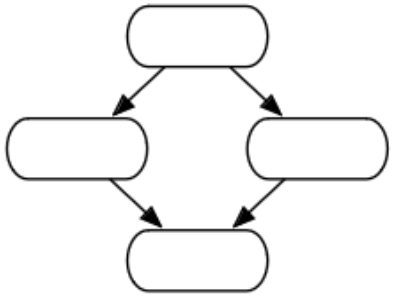
Using the control flow graph

Using the control flow graph

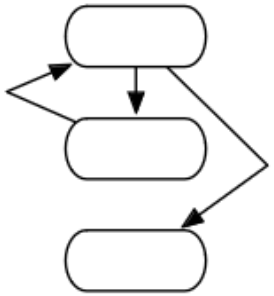
1. Simplify code
2. Translate code into a graph
3. Find paths through the graph
4. Trigger paths with test cases

Simplifying the code

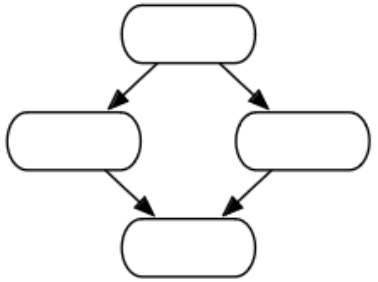
Allowed constructs



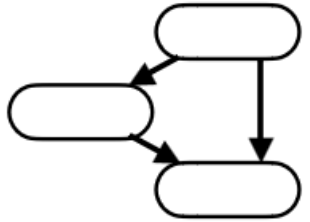
if-then-else
[cc = 2]



while
[cc = 2]



and/or
= if-then-else
[cc = 2]



If (with no else)

Replace for-loops

```python
def replace_for_loops_before():

    some_list = [1, 2, 3]

    for element in some_list:
        some_function(element)
```

```python
def replace_for_loops_after():

    some_list = [1, 2, 3]

    i = 0
    while i < len(some_list):
        element = some_list[i]
        some_function(element)
```

Replace ‘elif’

```python
def replace_elif_before(a: bool, b: bool, c: bool):
    if a:
        function_a()
    elif b:
        function_b()
    else:
        function_c()
```

```python
def replace_elif_after(a: bool, b: bool, c: bool):
    if a:
        function_a()
    else:
        if b:
            function_b()
        else:
            function_c()
```

Replace assignments with bool expressions

```python
def replace_bool_expressions_before(a: bool, b: bool, c: bool):
    result = (a or b) and ((b and c) or a)

    return result
```

```python
def replace_bool_expressions_after(a: bool, b: bool, c: bool):
    if (a or b) and ((b and c) or a):
        result = True
    else:
        result = False

    return result
```

Replace conditionals in if-statements (and)

```python
def replace_compound_if_and_before(a: bool, b: bool):
    if a and b:
        function_a()
    else:
        function_b()


def replace_compound_if_and_after(a: bool, b: bool):
    if a:
        if b:
            function_a()
        else:
            function_b()
    else:
        function_b()
```

Replace conditionals in if-statements (or)

```python
def replace_compound_if_or_before(a: bool, b: bool):
    if a or b:
        function_a()
    else:
        function_b()
```

```python
def replace_compound_if_or_after(a: bool, b: bool):
    if a:
        function_a()
    else:
        if b:
            function_a()
        else:
            function_b()
```

Replace conditionals in if-statements (3)

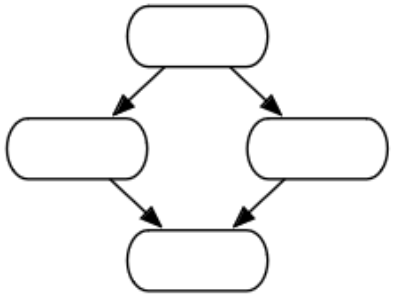
```python
def replace_compound_if_complicated_before(a: bool, b: bool, c: bool):
    if (a or b) and c:
        function_a()
    else:
        function_b()
```

```python
def replace_compound_if_complicated_step_1(a: bool, b: bool, c: bool):
    if a or b:
        if c:
            function_a()
        else:
            function_b()
    else:
        function_b()
```

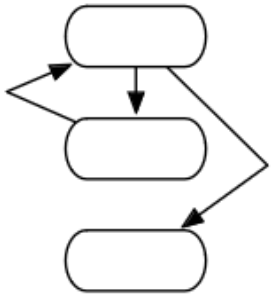
```python
def replace_compound_if_complicated_step_2(a: bool, b: bool, c: bool):
    if a:
        if c:
            function_a()
        else:
            function_b()
    else:
        if b:
            if c:
                function_a()
            else:
                function_b()
        else:
            function_b()
```

Constructing the graph

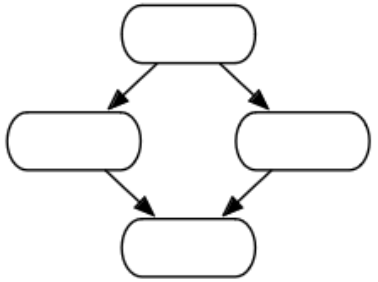
Constructing the CFG



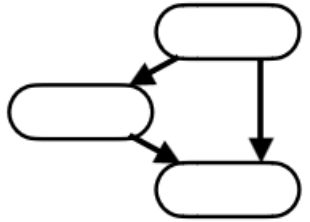
if-then-else
[cc = 2]



while
[cc = 2]



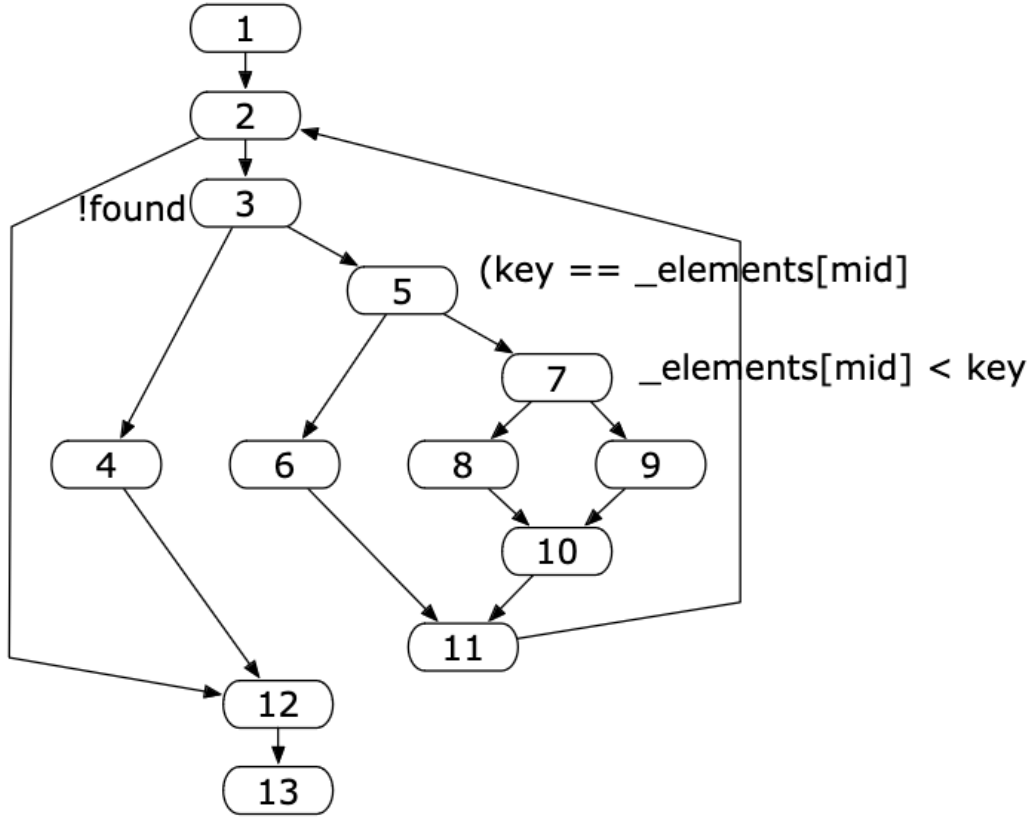
and/or
= if-then-else
[cc = 2]



If (with no else)

Constructing the CFG

```
 public boolean find(int key) {                          //Binary Search
   int bottom = 0;                                       // (1)
   int top = _elements.length-1;
   int lastIndex = (bottom+top)/2;
   int mid;
   boolean found = key == _elements[lastIndex];
   while ((bottom <= top) && !found) {                   // (2) (3)
     mid = (bottom + top) / 2;
     found = key == _elements[mid];
     if (found) {                                        // (5)
       lastIndex = mid;                                  // (6)
     } else {
       if (_elements[mid] < key) {                       // (7)
         bottom = mid + 1;                               // (8)
       } else {
         top = mid - 1; }                                // (9)
     }                                                   // (10) (11)
   }                                                     // (4) (12)
   return found;                                         // (13)
}
```



Paths and test-cases

Control flow graph

- **Find paths**
- **Come up with test-cases (= inputs/function args)**
  - Impossible path? => construct different paths
- **Differences:**
  - Basis path testing
  - Condition testing
  - Loop testing