



Laboratory of Software Analysis *Testing*

Filippo Ricca

ITC-Irst

Istituto per la ricerca
Scientifica e Tecnologica

ricca@itc.it



Papers

1. An approach to program Testing. J.C Huang. 1975.
2. Branch coverage for arbitrary Languages made easy. I.D.Baxter 2001.

Testing

One of the practical methods commonly used to detect the presence of errors in a computer program is to **test** it for a set of **testcases**.

Considering the program as **black box** and test it for all possible input cases is not possible in practice.

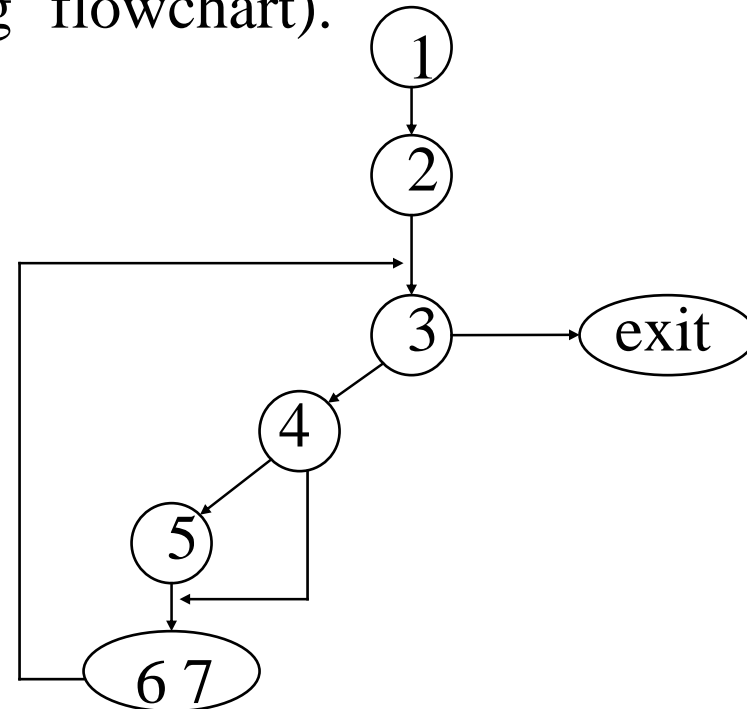
One widely accepted idea is this:

1. A randomly selected set of testcases is statistically insignificant.
2. A selection of testcases based on the **program structure** can be statistically significant.

Program Structure

The program structure used in structural testing is the **CFG** i.e. control flow graph (in 'Huang' flowchart).

```
1 read x,y;  
2 z:=1;  
3 while not y=0 do  
4   if y mod 2 =1  
5     z:=z*x  
6   y:=y/2  
7   x:=x*x  
end
```

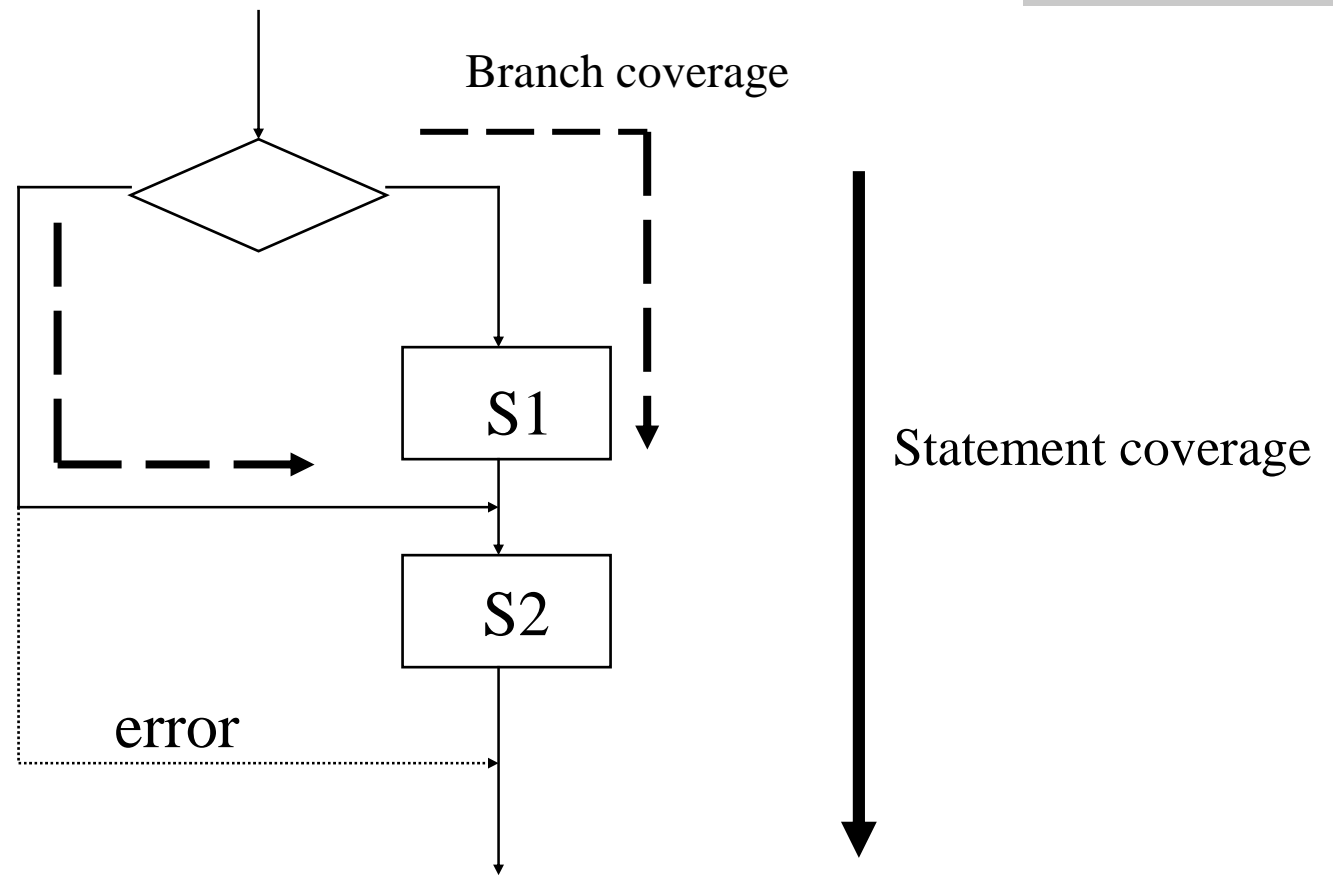


Test Criteria

1. **Statement coverage:** every **statement** in the program is executed at least once.
2. **Branch coverage:** every **branch** (or edge) in the CFG is traversed at least once.
3. **Path coverage:** every **path** in the CFG is traversed at least once.

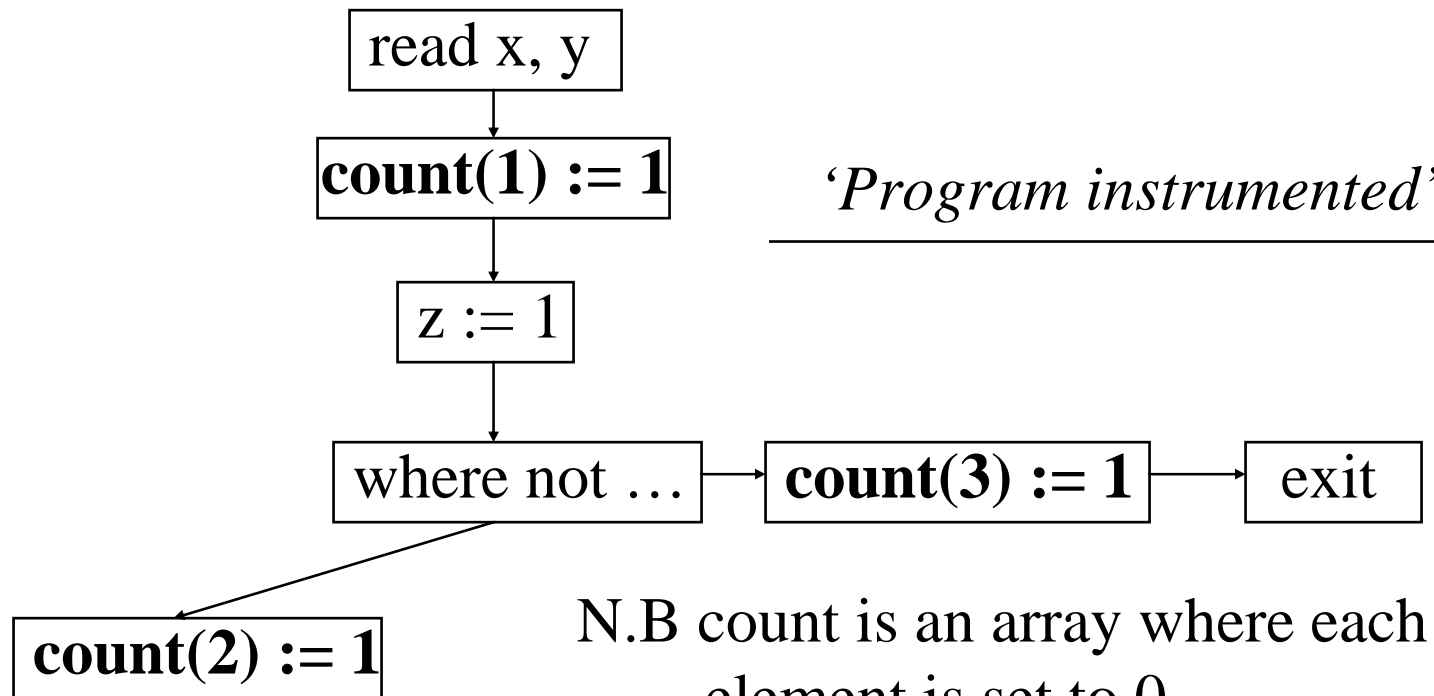
*Path coverage is impractical. Too long ...
Branch coverage is better than Statement coverage.*

Branch Vs. statement



How traversing each branch?

To determine whether or not each branch is traversed, we can place a 'counter' on each branch.



N.B count is an array where each element is set to 0.

Code instrumented

Now we are ready to test the program instrumented for a set of testcases. To reach complete coverage we **need a set of testcases** s.a:

$$\text{count}(1) \text{ or } \text{count}(2) \text{ or } \dots \text{count}(n) = \overline{1}$$

testcases

counter values

x

y

count(1)

count(2)

count(3)

count(4)

count(5)

10

0

1

0

1

0

0

20

1

1

1

1

1

0

- The branch '5' is not traversed. We have to find a new testcase that will force traversal of this branch. (es. x=5, y=2)

Testcase generation

In order to satisfy the branch coverage, we have to use the information provided by the counters to find additional testcases.

The process is **manual**:

1. executing the program instrumented using our set of testcases.
2. For each $\text{count}(i) = 0$ we have to find in the code the position of the branch i . Then we have to recover the condition C_i connected to the branch and finding a testcase that satisfy C_i .



Laboratory(1)

1. Writing TXL rules to put instrumentation in JAVA programs.

Such rules are simple to define in Java because each branch is marked with explicit syntax. The only tricky part is the introduction of a new ‘place number’ for each rule (see ‘counting function’ in TXL).

Rules (C-like)

rule **mark_function_entry**

replace f [function]

t [type] name [id] { decls [declarations] stmts [statements] }

by

t name { decls count(*new_number*)=1; stmts }

end rule

rule **mark_if_then_else_simple**

replace if_st [if_statement]

if (c [cond]) stmt1 [statement] else stmt2 [statement]

by

if (c) { count(*new_number*)=1; stmt1 }

else { count(*new_number*)=1; stmt2 }

end rule

switch,
while,
for,
do-while,
...

Laboratory(2)

2. Additional infrastructure needed

- a. Additional code to declare and initialize the *count* array.
- b. A script that executes *Jconsole_instrumented* using different inputs.
- c. A script that prints *count* on a file.
- d. An accumulating engine that computes binary-OR and put the result in *was_visited_by_some_test* file
- e. A program that display the *was_visited_by_some_test* file

Complete framework

