Criteri di copertura

Introduzione

Abbiamo visto in generale cosa si intende per criterio di test in generale, fino ad arrivare ai dei casi specifici a ad introdurre i **criteri di test strutturali**, che sono quelli che ora affronteremo.

Criteri di test strutturali

La definizione di criterio di test strutturale è la stessa di un criterio che fa testing white box:

$$C : P \times T \mapsto \{true, false\}$$

Inoltre può essere anche un criterio generatore di test:

$$C : P \mapsto T \quad \text{tale che } C(P, T) = true$$

Prima di vedere alcuni **criteri di copertura**, introduciamo il modo in cui questi vengono valutati.

Fault Detection Capability

> La **fault detection capability** indica quali errori un test set adeguato a un certo criterio *garantisce* di trovare

Copertura delle istruzioni (*statement coverage*)

> Un test set T è **adeguato** per testare un programma P secondo il criterio di ***copertura delle istruzioni***, se per ogni istruzione s di P esiste un caso di test in T che esegue s

L'idea è che ogni istruzione di P venga eseguita almeno una volta da almeno un caso di test.

```
if (x ≠ 0)
    x = x + 10;
else
    x = x - 10;
```

Per eseguire tutte le istruzioni, T dovrà contenere un caso di test che esegua il ramo `if` e uno che esegua il ramo `else`, quindi un caso con `x ≠ 0` e un caso con `x==0`, ad esempio $T = \{0, 3\}$

✔ La copertura delle istruzioni garantisce di trovare istruzioni errate

✘ ...ma non riesce a individuare errori nelle decisioni.

Prima di passare al prossimo criterio di copertura, vediamo cosa accade se un criterio viene soddisfatto parzialmente da un test set.

Misura di copertura

Abbiamo visto i criteri di test empirici, i quali hanno come codominio non più un booleano $\{true, false\}$, bensì un intervallo continuo tra $[0, 1]$, in
H4 quanto esprimono un *grado di copertura*. Quindi, specificando nel nostro caso in cui stiamo descrivendo criteri strutturali che non considerano le specifiche avremo in particolare:

$$C : P \times T \mapsto [0, 1]$$

il quale viene anche definito come **misura di copertura**. Nel momento in cui un test set T soddisfa un criterio, avremo $C(P, T) = 1$

Nel caso della **copertura delle istruzioni**, la misura di copertura è definita così:

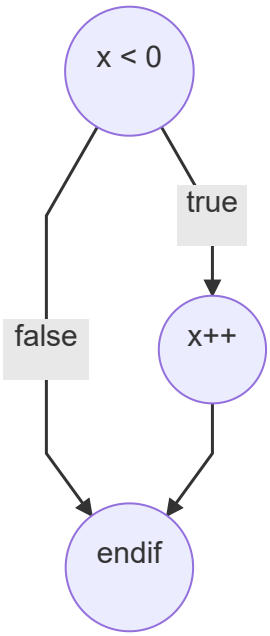
$$C_{\text{istr}} = \frac{\#\text{ istruzioni eseguite}}{\#\text{ istruzioni eseguibili}}$$

H3 Copertura degli archi (*branch coverage*)

> Un test set T soddisfa il criterio di ***copertura degli archi*** di un programma P se e solo se ogni **arco** (***branch***) del grafo di P viene percorso almeno una volta

```
if (x < 0)
    x++;
endif
```

Il grafo di questo codice è il seguente:



Se volessimo trovare un criterio di copertura delle istruzioni, ci basterebbe eseguire tutte le istruzioni, cioè ci basterebbe entrare nel ramo `if`, ad esempio con $T = -3$ eseguiremmo tutte e tre le istruzioni.
Però non stiamo soddisfacendo il criterio di copertura degli archi, in quanto l'arco etichettato con `false` non sarebbe percorso. Per farlo ad esempio $T = \{-3, 2\}$, dove con `x=-3` percorro gli archi di destra come prima, mentre con `x=2` percorro anche l'arco a sinistra.

Il **branch coverage** implica lo statement coverage ed è quindi un criterio più forte.

La **misura di copertura** del branch coverage è data da:

$$C_{\text{archi}} = \frac{\#\text{ archi percorsi}}{\#\text{ archi percorribili}}$$

H3 Copertura delle decisioni

> Un test set T è adeguato per testare un programma P secondo un criterio di ***copertura delle decisioni***, se per ogni **decisione** di P esiste:
>
> - un caso di test in T in cui la decisione *è presa*
> - un caso di test in T in cui la decisione *NON è presa*

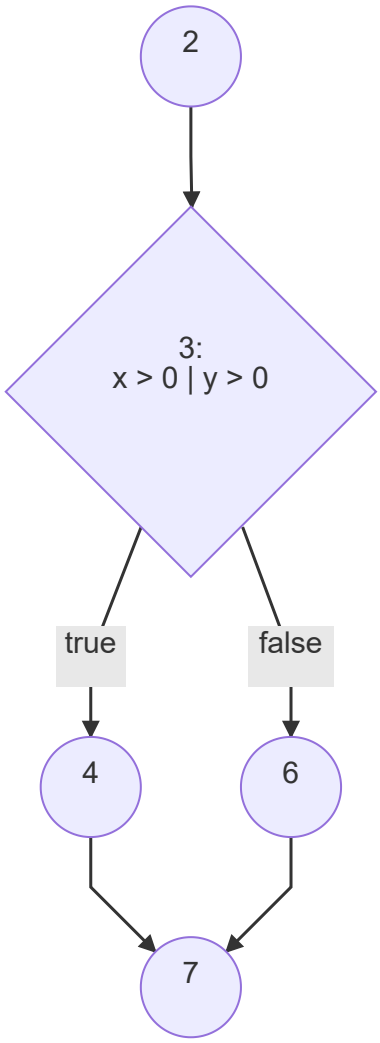
dove per **decisione** intendiamo:

> **Espressione booleana** a guardia di un'istruzione condizionale o di una iterativa

---

```
int main(int x,y){
    int result;
    if (x > 0 | y > 0)
        result = x;
    else
        result = y;
    printf("d", result);
}
```

il cui grafo è:



Per soddisfare il criterio di copertura delle decisioni, serve un test set che esegua entrambi i casi dell' `if` , come ad esempio
$T = \{\{x = 1, y = 1\}, \{x = -1, y = -1\}\}$:

- Il primo caso di test percorre il ramo `true`
- il secondo percorre il ramo `false`

Il criterio di copertura delle decisioni è **equivalente** alla copertura degli archi , in quanto in entrambi i casi ogni arco del grafo viene percorso.

H3 Copertura delle condizioni

> Un test set T è adeguato per testare un programma P secondo il criterio di ***copertura delle condizioni***, se per ogni **condizione** di P esiste:
>
> - Un caso di test in T in cui la condizione *è vera*

> - Un caso di test in T in cui la condizione *è falsa*

dove per **condizione** intendiamo:

> *Espressione booleana **atomica*** che appare in una *decisione*

Prendendo l'esempio di sopra, un criterio di copertura delle condizioni può essere il seguente: $T = \{\{x = 1, y = -1\}, \{x = -1, y = 1\}\}$, infatti data la decisione `x > 0 | y > 0`, avremo:

- $x = 1 \rightarrow$ x > 0 ; $y = -1 \rightarrow$ y > 0 $\implies$ `true`
- $x = -1 \rightarrow$ x > 0 ; $y = 1 \rightarrow$ y > 0 $\implies$ `true`

cioè, ogni condizione viene coperta.
Tale criterio non copre tuttavia le decisioni, in quanto viene sempre percorso il ramo `true`.

Come si vede dall'esempio, la copertura delle condizioni **non implica** la copertura delle decisioni né la copertura degli archi. Vale l'opposto? Vediamo:

Prendiamo il solito programma, consideriamo il criterio che seleziona il seguente test set: $T = \{\{x = 1, y = -1\}, \{x = -1, y = -1\}\}$ con la decisione
`x > 0 | y > 0` avremo:

- $x = 1 \rightarrow$ x > 0 ; $y = -1 \rightarrow$ y > 0 $\implies$ `true`
- $x = -1 \rightarrow$ x > 0 ; $y = -1 \rightarrow$ y > 0 $\implies$ `false`

cioè ogni decisione viene coperta, ma non ogni condizione! (manca y > 0).

Quindi il criterio di copertura delle decisioni **non implica** la copertura delle condizioni.

L'idea è quella di unire i due criteri.

H3 Copertura delle decisioni e delle condizioni

> Richiede che siano rispettate *sia la copertura delle decisioni sia la copertura delle condizioni*

Prendendo in considerazione il solito programma, consideriamo il criterio che seleziona il seguente test set:
$T = \{\{x = 1, y = 1\}, \{x = -1, y = -1\}\}$ con la decisione `x > 0 | y > 0` avremo:

- $x = 1 \rightarrow$ x > 0 ; $y = 1 \rightarrow$ y > 0 $\implies$ `true`
- $x = -1 \rightarrow$ x > 0 ; $y = -1 \rightarrow$ y > 0 $\implies$ `false`

cioè sono coperte tutte le decisioni e tutte le condizioni.

H3 Multiple Condition Coverage (*MCC*)

> Un test set soddisfa ***MCC*** se testa **ogni possibile combinazione** dei valori di verità delle condizioni atomiche in ogni decisione.

Quindi con n condizioni in una decisione avremo 2^n combinazioni da soddisfare. Il numero si può ridurre attraverso il meccanismo della **lazy evaluation**, ma spesso può diventare impraticabile.

Data la seguente decisione: `x > 0 || y > 0`, avremo:

Test case	x > 0	y > 0	binary
$x = -1, y = -1$	`false`	`false`	00
$x = -1, y = 1$	`false`	`true`	01
$x = 1, y = \text{qualsiasi}$	`true`	non valutato	10 & 11

In questo caso avremmo $2^2 = 4$ combinazioni, ma la **lazy evaluation** ci permette di fondere le combinazioni in cui `x > 0`, quindi 3 combinazioni.

Come detto, spesso l'**MCC** può essere impraticabile, in quanto esponenziale rispetto al numero di condizioni. Per questo si è introdotto un criterio che invece è lineare.

H3 Modified Condition/Decision Coverage (*MCDC*)

> Il test set deve essere scelto in modo che *ogni condizione all'interno di un decisione deve far variare in modo indipendente il valore finale della decisione*

Cioè, il criterio seleziona dei test set in cui i casi di test fanno variare una singola condizione, che a sua volta è determinante nel far variare la decisione. In questo modo si escludono le combinazioni ridondanti.