

# **Lezione 1 – Concetti base**

## **Fondamenti teorici**

**Angelo Gargantini**

---

Università degli Studi di Bergamo – Informatica III

# Definizioni di base del testing

Un **programma**  $P$  è una funzione da un dominio  $D$  ad un codominio  $R$ :

$$P: D \rightarrow R$$

può essere parziale – non definita per qualche  $d$  in  $D$

**introduciamo un predicato **OK**:**

**OK**( $P,d$ ) con  $d$  in  $D$ , se  $P$  è corretto per l'input  $d$ , cioè se produce  $P(d)$  corretto

**OK**( $P$ ) se  $P$  è **corretto**, se per ogni  $d$  in  $D$  **OK**( $P,d$ )

**Rivediamo alcune definizioni:**

**P sia il programma corretto**

**FAILURE o malfunzionamento:**

- Eseguo P con d e non ottengo P(d): o ottengo un altro valore P'(d) oppure il programma si blocca, ...
- cioè  $\text{not OK}(P,d)$

**FAULT o difetto o bug:**

- P è stato implementato in P'
- Se  $P' \neq P$ , la diversità è il difetto

**ERRORE:**

- Il motivo del difetto

**Un caso di test (test case) o test è un elemento di D**

- **esempio:** se D è gli interi, un test è un intero, se D è sequenze di interi, un test sarà un sequenza (possibilmente finita) di interi, ecc ...

**Un test set T o test suite è un sotto insieme (finito) di D**

- **esempio:** D una coppia di interi ( $= \text{int} \times \text{int}$ ), T sarà un sottoinsieme di D, cioè un insieme di coppie di interi:  $T \subseteq \text{int} \times \text{int}$

**Applico il test set T al programma P:**

OK(P,T) se per ogni t in T, OK(P,t): il test set T non ha scoperto errori (negativo)

se t in T tale che not OK(P,t): test positivo

dato il metodo – programma P

```
static void foo(String s, int x)
```

**un caso di test è una coppia String e int**

- esempio ["pippo",3]

**un test set è un insieme di coppie String e int**

- esempi:
  - {"pippo",3} : insieme con un solo caso di test
  - con tre casi di test:  
{"pippo",3}, ["",54657], ["ccc",-10]}

# Test ideale e correttezza dei programmi

Inizialmente la ricerca cercò di trovare dei test che potessero **dimostrare** la correttezza dei programmi

- cioè testando il programma con un insieme finito  $T$  e non osservando difetti, si potesse essere sicuri che non ci fossero difetti
- tale test set (se esiste) è detto **ideale**

**$T$  è **ideale** se  $OK(P,T) \rightarrow OK(P)$**

**In questo caso l'esecuzione di  $T$  senza errori è prova di correttezza di  $P$**

**Come faccio a trovare un  $T$  ideale?**

# Testing esaustivo

**Un test ideale è quello esaustivo con  $T = D$**   
 **$OK(P,T) \rightarrow OK(P,D)$**

è fattibile? Considera questo esempio:

```
static int sum(int a, int b) return a + b;
```

- un int è un numero binario di 32 bit, quindi ci sono solo  $2^{32} \times 2^{32} = 2^{64}$  ca  $10^{21}$  test ad un nanosecondo ( $10^{-9}$ ) per test case circa 30000 anni.
- **se D è infinito? (ad esempio un sistema reattivo)**

**Il test esaustivo non è fattibile:  $T \subset D$**

**Devo selezionare un sottoinsieme di D**

# Test Criteria (formale)

**Un test **criteria** o di **adeguatezza** è una funzione che**

- [per un programma  $P$  e la sua specifica (quello che  $P$  deve fare)  $S$ ]

**dato un test set  $T$  restituisce vero o falso,**

- cioè è un predicato su l'insieme dei possibili  $T$

Dato un test suite  $T$ ,  $C_{PS}(T)$  è vero se e solo se  $T$  è adeguato a trovare ogni difetto in  $P$  rispetto  $S$  secondo il criterio  $C$



# Esempio di Test Criteria

sia  $P$  un programma con dominio  $D$  gli interi:

- un test set  $T$  è un insieme di interi
- un criterio  $C$  dovrà dire quando  $T$  è adeguato a testare  $P$
- $C$  sarà un predicato sugli insiemi di interi

**un esempio è il seguente  $C_{ex}$ :**

un test set è adeguato secondo  $C_{ex}$  se contiene un numero positivo e uno negativo, formalmente:

$C_{ex}$  se  $\exists t \in T t > 0$  and  $\exists t \in T t < 0$

– test set adeguati:  $\{-1, 1\}$ ,  $\{-1, -5, -10, -20, 5\}$

– test set non adeguati  $\{-1, -5\}$ ,  $\{0, 15\}$

# Program e Spec based

**Il criterio è quindi una funzione C che dato un programma P, la sua specifica S e un test set T restituisce vero o falso**

**C:  $P \times S \times T \rightarrow \{T, F\}$**

In **program-based** testing C non dipende da S

**C:  $P \times T \rightarrow \{T, F\}$**

In **specification-based** testing non dipende da P

**C:  $S \times T \rightarrow \{T, F\}$**

**Un criterio può essere anche inteso come generatore di test set dato P e S:**

**C:  $P \times S \rightarrow T$  (tale che  $C(P, S, T)$  è true)**

**dato P e S genera un test set T adeguato**

# Criterio affidabile (1)

**Un criterio è affidabile se per ogni coppia di test set T1 e T2 adeguati secondo il criterio C, se T1 individua un malfunzionamento, allora anche T2 e viceversa**

- $OK(P,T)$  non dipende dal T particolare selezionato
- tutti i test set o riescono a trovare i difetti, oppure non riescono a trovarne
- posso prendere il piu' piccolo T: se scopro errori con uno piu' grande, li scoprirei anche con uno piu' piccolo

## Criterio affidabile (2)

- Un criterio affidabile produce sempre risultati consistenti e ugualmente utili
- Non è detto però che un criterio affidabile riesca a scoprire malfunzionamenti:
  - considera raddoppia:

```
int raddoppia(int x) { return x*x ; }
```

- un criterio C1 che seleziona solo sottoinsiemi di  $\{0,2\}$  è affidabile
  - $T1 = \{0,2\}$  ,  $T2 = \{2\}$  ...
- Affidabilità è importante ma non è sufficiente

# Criterio valido

**Un criterio C è **valido** se, qualora il programma P non sia corretto, esiste almeno un test set T che soddisfa C che è in grado di individuare il difetto**

- formalmente  
 $\exists T$  tale che  $C_{PS}(T)$  and not  $OK(P,T)$
- un criterio valido riesce (ma non garantisce) a trovare il difetto
- prendi un input a caso è un criterio valido (non molto utile)

# Esempio di criterio valido

## considera raddoppia:

```
int raddoppia(int x) { return x*x ; }
```

- un criterio C2 che seleziona  
qualsiasi  $T \subseteq \{0,1,2,3,4\}$

C2 è valido, basta prendere  $\{1\}$  o  $\{0,1\}$  o  $\{1,3\}$   
ma non affidabile:  $\{0,2\}$  non scopre il difetto

- C3:  $\exists t \in T$  tale che  $t \geq 3$   
esempi di T:  $\{3\}$ ,  $\{0,2,3\}$ ,  $\{-4,3,6,10\}$  ,...

C3 è valido : **scopro** il difetto  
e affidabile : scopro **sempre** il difetto

# Teorema di Goodenough e Gerhart

Dato un criterio  $C$ , un programma  $P$

Se  $C$  è *affidabile* per  $P$  e  
 $C$  è *valido* per  $P$  e

}  $C$  è *ideale*

$T$  test suite selezionato con  $C$  e

$T$  non trova malfunzionamenti in  $P$ :  $ok(P, T)$

**$\Rightarrow OK(P), T$  è ideale**

Se un programma  $P$  passa un test set  $T$  che è selezionato con un criterio affidabile e valido (si dice in questo caso IDEALE), allora  $P$  è corretto

# Esempio di criterio ideale

**programma raddoppia:**

```
int raddoppia( int x ) {  
    return x*x ;  
}
```

**un criterio che selezioni casi di test con almeno un numero in T che sia  $\geq 3$  è valido e affidabile**



# In sintesi

- Abbiamo introdotto:
  - un caso di test è un elemento del dominio su cui è definito il programma P
  - un criterio di test è una funzione che dice se un insieme di test (test set) è adeguato per testare P
  - un criterio si può pensare come funzione generatrice che dato un programma costruisce un test set T
- Ricordati:
  - un test **ideale** è quello che **garantisce** la scoperta di ogni difetto
  - il test esaustivo sarebbe ideale ma non fattibile
  - un criterio di test è ideale se seleziona test ideali



# **L2.1 I limiti del testing**

**Angelo Gargantini**

---

Università degli Studi di Bergamo – Informatica III

**Nella scorsa lezione, abbiamo visto che:**

- **un criterio C dice se un test set T è adeguato a testare un programma P**
  - C può essere utilizzato per generare un test set T
- **C è ideale se selezionando T in accordo con C si è sicuri di trovare ogni difetto di P**

# Esempio di criterio valido e affidabile

```
int raddoppia( int x ) {  
    return x*x ;  
}
```

**Un criterio che selezioni casi di test con un numero  $\geq 3$  e' **valido e affidabile**.**

**Però:**

- di questo programma sappiamo già il difetto
- sappiamo già esattamente cosa deve fare

**Possiamo trovare criteri in generale **validi e affidabili**?**

- Un criterio che richiede  $T = D$  e' ideale ma impraticabile

# Test esaustivo

## Il test esaustivo (T=D) è

**impraticabile**: lo spazio D è normalmente troppo grande

**impossibile**: applicando un test, non riesco a distinguere se un programma P si è bloccato (halt) o devo aspettare ancora

## Esempio:

```
int foo(int x) {  
    y = very-complex-computation(x);  
    write(y);  
}
```

- `foo` potrebbe contenere un bug in `vcc(x)` e andare avanti all'infinito: dato un input `x` quanto devo aspettare per decidere se c'è un bug o no?
- **Vedi problema dell'halt**

## **NON POSSO IN MODO ALGORITMICO TROVARE CASI DI TEST IDEALI:**

### **Teorema di Howden**

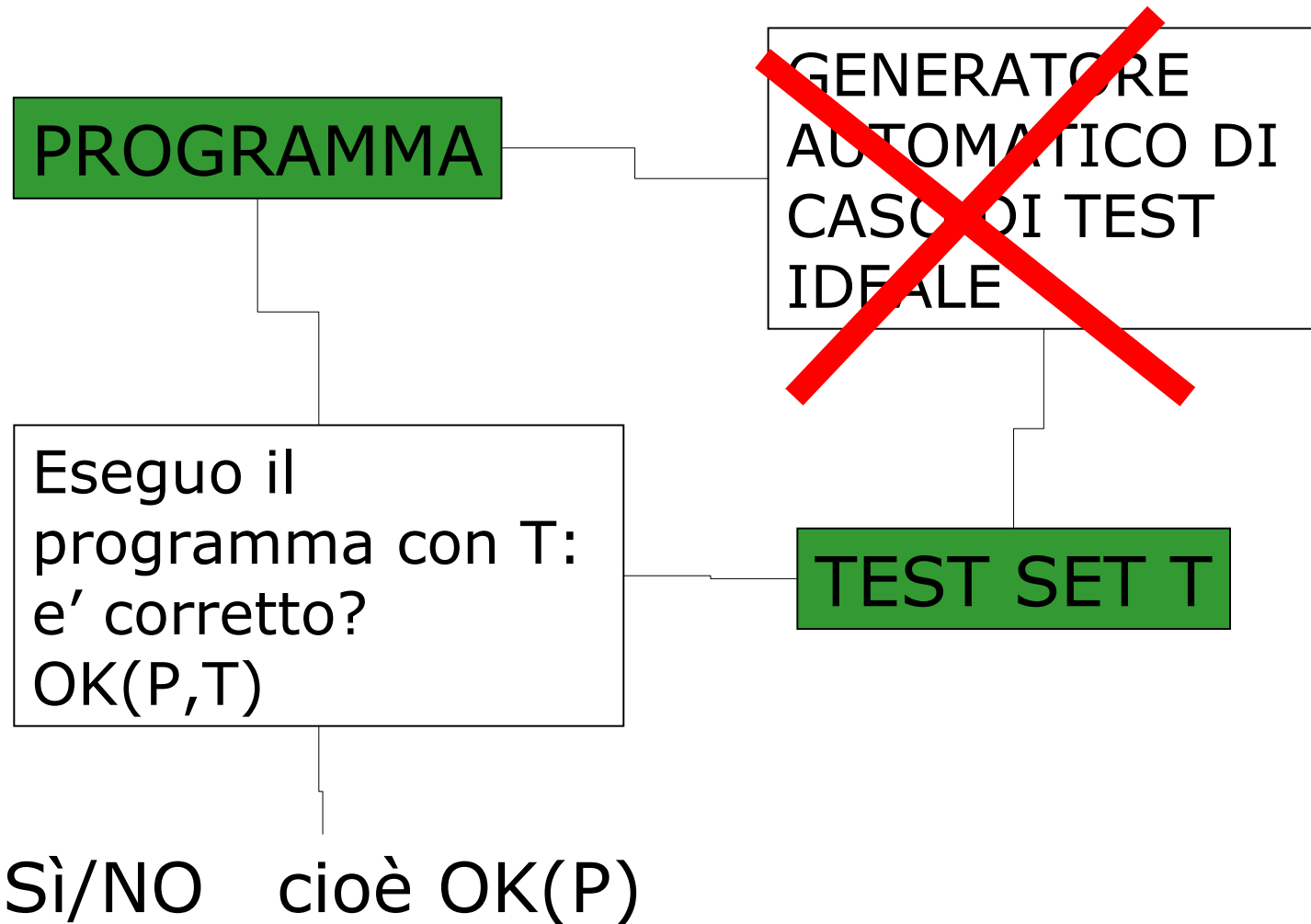
non esiste un algoritmo che dato un programma  $P$  generi un test ideale finito.

## IL TESTING NON PUÒ DIMOSTRARE CORRETTEZZA DEL SW

### **Dijskra:**

Il test di un programma può rilevare la presenza di malfunzionamenti ma mai dimostrarne l'assenza.

# Correttezza mediante testing?





# Limite del testing

## Il testing

- **non** consiste nel mostrare l'assenza di difetti in un programma

**ma nel mostrarne la presenza**

**Un test che ha **successo** è uno che trova un bug!**

# Test Criteria non ideali

**Se non posso definire test criteria ideali a cosa servono?**

**Definizione di “empirical” test criteria**

- criteri che non sono ne’ validi ne’ affidabili ma si sono dimostrati utili

**Test criteria come “stopping rule”:**

- ho testato abbastanza (posso essere confidente – ma non certo - che P sia corretto?)

**Test adequacy non solo true o false ma e’ una **misura** di copertura:**

**$P \times S \times T \rightarrow [0, 1]$**

# Teorema di Weyuker

**Anche i seguenti problemi risultano indecidibili (non risolvibili mediante algoritmo in tutti i casi):**

- esiste almeno un dato in ingresso che causa l'esecuzione di un particolare comando?
- esiste almeno un dato in ingresso che causa l'esecuzione di un particolare cammino?
- esiste almeno un dato in ingresso che causa l'esecuzione di tutte le istruzioni?
- ...

# Generazione in pratica

## **Problema non computabile:** trovare un insieme di input che esegua tutte le istruzioni

- **non esiste** algoritmo per risolvere questo problema per ogni programma
- esistono però algoritmi e tecniche che sono in grado di risolverlo per molti programmi
- esistono tool per la generazione di casi di test: non è detto che funzionino sempre, però negli usi pratici
  - es. **CUTE, e altri commerciali**

# In sintesi

- Abbiamo visto:
  - non esiste un algoritmo per generare per un qualsiasi programma casi di test ideali
  - il testing può provare la presenza di difetti ma non garantire l'assenza.
- Ricordate che:
  - molti problemi riguardo il testing non sono risolvibili per un qualsiasi programma mediante algoritmi
    - esempio come coprire una certa istruzione;
    - però posso scrivere algoritmi che risolvono il problema per alcuni (molti?) programmi
  - per non potendo trovare criteri di test ideali, alcuni criteri di test sono utili in pratica

