Verifica – parte IIA

Rif. Ghezzi et al.

6.3 - 6.3.3

Test (o analisi dinamica)

- Consiste nell'osservare il comportamento del sistema in un certo numero di condizioni significative
- ***Non*** può (in generale) essere ***esaustivo***
- Scelta di un ***insieme finito di dati di test***
- La scelta discende da considerazioni sul programma e/o sul problema

Mancanza di continuità

- Il test in un numero finito di casi non può portare a estrapolare conclusioni sulla correttezza del programma.
- Il software è purtroppo un manufatto con *mancanza di continuità*: un programma che funziona correttamente in un caso può presentare malfunzionamenti anche in un caso leggermente diverso

Esempio

```
procedure binary-search (key: in element;
            table: in elementTable; found: out Boolean) is
begin
bottom := table'first; top := table'last;
while bottom < top loop
    if (bottom + top) rem 2 ≠ 0 then
        middle := (bottom + top - 1) / 2;
    else
        middle := (bottom + top) / 2;
    end if;
    if key ≤ table (middle) then
        top := middle;
    else
        bottom := middle + 1;
    end if;
end loop;
found := key = table (top);
end binary-search
```

L'assenza della clausola else non ha conseguenze in tutti i casi in cui la dimensione della tabella è una potenza di 2.

Test in piccolo e in grande

- Test in piccolo: dei singoli moduli. Approcci:
  - White box (o strutturale o trasparente): test set scelti in base all'implementazione.
  - Black box (o funzionale): test set scelti in base alla specifica.
- Test in grande: organizzazione e suddivisione dell'attività di test in base alla struttura di sistemi complessi.

Terminologia (IEEE)

- Il funzionamento non corretto di un programma è detto ***malfunzionamento*** (*failure*)
- E' legato al comportamento di un programma, ma non alla sua struttura statica (il codice)
- Un malfunzionamento è causato dalla presenza di ***anomalie*** (*fault*) nel programma (***bug*** nell'uso comune)
- Esempio: programma che somma X e Y, anomalia se uso l'operatore * al posto dell'operatore +
- Il termine ***errore*** (*error*) indica, nello standard proposto dall'*IEEE*, la *causa* di un anomalia

Obiettivi dell'attività di test

- Dimostrare la ***presenza*** di errori, non la loro assenza (Dijkstra).
- Il successo del test non implica la correttezza del programma.
- Dovrebbe essere eseguita secondo ***metodi sistematici***, e integrata con altre tecniche (analisi statica).

Obiettivi dell'attività di test

- Dovrebbe aiutare a localizzare gli errori, non solo rilevarli, per facilitarne la correzione (*debugging*)
- Dovrebbe essere ***ripetibile***
  - Influenza dell'ambiente di esecuzione (ad esempio, variabili non inizializzate)
  - Non-determinismo (*concorrenza*)
- Dovrebbe essere accurata (utilità di specifiche formali)

Definizioni

- Se
  - P è il programma
  - D è il dominio di input
  - R è il dominio di output
- P si può vedere come una funzione

$$P: D \rightarrow R$$

- Dato un dato d del dominio D, $d \in D$, $P(d) \in R$ è il risultato dell'esecuzione di P con d come ingresso

Correttezza

- OR: descrizione (formale o informale) della relazione input-output del programma
- $OR \subseteq D \times R$
- Dato $d \in D$, P è corretto per d se

  $(d, P(d)) \in OR$
- P è corretto se è corretto ***per ogni*** $d \in D$

Non-correttezza

- Fallimento:

  P non è corretto per qualche d

- Errore (o difetto):

  Qualsiasi causa che può portare a un fallimento

- Esempi:
  - errore di battitura
  - variabile non inizializzata

Non-correttezza

- Malfunzionamento (*fault*): stato intermedio scorretto che può essere assunto dal programma
  - Es. variabile con valore sbagliato
- Fallimento $\Rightarrow$ Malfunzionamento $\Rightarrow$ Errore
- In generale non valgono le implicazioni inverse

Definizioni

- Caso di test t per P: D → R:

  Un elemento t di D

- Insieme di test (*test set*) T:

  Insieme di casi di test, sottoinsieme di D

- P è corretto per T (ha successo per T) se è corretto per ogni suo elemento

Test set ideale

- Dato P: D → R, un test set T è *ideale* per P se la presenza di un errore in P implica che P fallisca per qualche elemento d di T.
- Un test set ideale permetterebbe la dimostrazione di correttezza di un programma tramite test (*se T è ideale e ha successo per P, allora P è corretto*)
- Per un programma corretto, ogni *test set* è ideale.

Criteri di selezione di test

- Dato P: D $\rightarrow$ R, un criterio di selezione C $\subseteq 2^D$ è un insieme di *test set* per P
- Può essere definito come predicato sui test.
- Se T $\in$ C, T *soddisfa* C.
- Esempio:
  - se D=I (interi), C può essere l'insieme dei test set che contengono almeno un negativo, 0 e un positivo.
  - {-4, 0, 5} $\in$ C
  - {-4, 5, 7} $\notin$ C

Coerenza

- Un criterio C per un programma P è *coerente* se e solo se, per ogni T_1, T_2 $\in$C, P è corretto per T_1 se e solo se P è corretto per T_2.
- Se un criterio è coerente, ogni suo elemento fornisce la stessa informazione sulla correttezza di P (*stesso successo*, ma non necessariamente provoca gli stessi fallimenti).

Completezza

- Un criterio C per un programma P è *completo* se e solo se, in caso di scorrettezza di P, P non ha successo per almeno un insieme di test T selezionato da C.
- Tutti i test set che soddisfano un criterio coerente e completo sono ideali: ogni test set permette di decidere la correttezza di P.

Esempio

```
1 program raddoppia (input, output);
2 var x,y: integer;
3 begin
4    read(x);
5    y := x*x;
6    write(y)
7 end.
```

- C_1 che seleziona sottoinsiemi di {0,2} è coerente, ma non completo (sempre successo)
- C_2 che seleziona sottoinsiemi di {0,1,2,3,4} è completo, ma non coerente ({0,4} e {0,2} danno risultati opposti)
- C_3 che seleziona insiemi che hanno almeno un elemento maggiore o uguale a 3 è coerente e completo

Esempio

```
1 program raddoppia (input, output);
2 var x,y: integer;
3 begin
4     read(x);
5     y := x+5;
6     write(y)
7 end.
```

- Le proprietà sono relative a un programma: con la modifica,
  - C_1, C_2 diventano coerenti e completi
  - C_3 è completo, ma non coerente (il test set {5} non rileva malfunzionamenti).

Più fine di

- Un criterio C_1 è ***più fine di*** un criterio C_2 se, per ogni programma P,

$\forall T_1$ soddisfa C_1 $\exists T_2 \subseteq T_1 \mid T_2$ soddisfa C_2

- Esempio:
  - $C_2 = \{(x_1,x_2,x_3) \mid x_1<0, x_2=0, x_3>0\}$

$C_1 = \{(x_1,x_2,\ldots,x_n) \mid n \geq 3, (\exists\ i,j,k,m,p \mid x_i<0, x_j=0, x_k>0, \text{even}(x_m), \text{odd}(x_p))\}$

Definizioni non effettive

- Non esistono algoritmi per verificare se un elemento (programma, test set, criterio) soddisfa la definizione
- In particolare:
  - non esiste un algoritmo per decidere la correttezza di un programma
  - non esiste un algoritmo per costruire un criterio completo e coerente che non sia l'esaustività

Quindi?

- Si utilizzano approcci empirici, consapevoli che non si riesce a verificare per via sperimentale la correttezza

Principi empirici di test

- Nozione intuitiva di *test set significativo* (approssimazione del test set ideale): alta probabilità di rilevare errori.
- Dati due test set significativi T_1 e T_2, se $T_1 \subseteq T_2$
  - T_2 sarà più attendibile di T_1, ma
  - T_1 può essere preferibile per motivi di costo
- Compromesso fra attendibilità e costo

Significatività

- Non coincide con la cardinalità del test set.
- if x > y then

  max = x;

  else

  max = x;
- {(3,2), (2,3)} rileva l'errore; {(3,2), (4,3), (5,1)} no.

Principio di completa copertura

- Se è possibile dividere il dominio D di input come

  $D = D_1 \cup D_2 \cup \ldots \cup D_n$

  dove gli elementi di ogni D_i causano un comportamento simile, un test set significativo si ottiene prendendo un rappresentante per ogni D_i.

  (Principio di completa copertura)
- Il criterio corrispondente è l'appartenenza al test set di un elemento rappresentativo di ogni D_i.

Esempio

- Specifica del programma per il calcolo del fattoriale di un numero n:
  - Se n < 0, errore
  - Se $0 \leq n < 20$, restituire n!
  - Se $20 \leq n \leq 200$, approssimazione di n! a meno dello 0.1%, ottenuta con calcoli in virgola mobile
  - Se n > 200, l'input può essere rifiutato
- $D = \{n|n<0\} \cup \{ 0..19, 20,..200 \} \cup \{n|n>200\}$
- La correttezza su un rappresentante di ogni D_i non implica la correttezza del programma

Principio di completa copertura

- Se gli insiemi D_i non sono disgiunti, si possono scegliere rappresentanti nelle intersezioni per ridurre il numero di test
- Esempio:
  - D1 = {d | d mod 2 = 0}
  - D2 = {d | d $\leq$ 0}
  - D3 = {d | d mod 2 $\neq$ 0}
- Si ottiene completa copertura con {-37, 48}

Principio di completa copertura

- Interpretazioni degeneri:
  - Un solo D_i
  - D_i composti da un solo elemento *esaustivo!*
- Si ottiene un criterio coerente suddividendo D in modo che, se due test set T_1 e T_2 differiscono solo per gli elementi rappresentativi di ogni D_i che li compongono, i risultati del test sono uguali.

Principio di completa copertura

- Un criterio è completo se, per ogni errore possibile, esiste almeno un D_i tale che ogni suo elemento rileva l'errore.
- Scegliendo i criteri con questi obiettivi si approssimano coerenza e completezza.
- Se ci sono dubbi, si può rendere più affidabile il test scegliendo più di un rappresentante per ogni D_i.

Criteri di selezione del test

- Non è possibile identificare automaticamente una scomposizione in D_i che garantisca coerenza e completezza …
- Si lavora in base a buon senso ed analisi, per aumentare l'affidabilità dei test scelti → struttura del programma