Verifica – parte IIC

Rif. Ghezzi et al.
6.3.4.2 - 6.3.4.4

Sommario (recap)

- **Test**
  - In piccolo
    - White box
    - **Black box**
    - Condizioni di confine
    - Problema dell' Oracolo
  - In grande
    - Test di modulo
    - Test di integrazione
    - Test di Sistema
    - Test di Accettazione

Test funzionali (black box)

- Il test white-box non rileva la mancata implementazione di funzionalità.
- Test ***black-box***, basati sulla specifica di un programma, anziché sulla loro implementazione.
- I ***test set*** si individuano ***in base a casistiche*** ricavate ***dalla specifica*** (formale o informale).

Esempio di specifica

- Il programma riceve in ingresso una registrazione che descrive una fattura (viene fornita una descrizione dettagliata del formato della registrazione). La fattura deve essere inserita in un file di fatture *ordinato per data*. La fattura deve essere inserita nella posizione corretta del file. Se esistono altre fatture che riportano la *stessa data*, la nuova fattura deve essere inserita *dopo* l'ultima già presente. Inoltre devono essere eseguite alcune verifiche di coerenza: il programma dovrebbe verificare se il *cliente è già presente* nel corrispondente archivio dei clienti, se i dati del cliente nei due file corrispondono, etc.

Approccio intuitivo al test

1. Si fornisce una fattura la cui data corrisponde alla data corrente
2. Si fornisce una fattura la cui data precede la data corrente (illegale?). Sottocasi:
    1. Fattura con stessa data di una fattura già presente
    2. Fattura con data diversa da tutte quelle già presenti
3. Si forniscono diverse fatture scorrette

Tecniche sistematiche black-box

- Test guidato da specifiche logiche
- Test guidato da sintassi
- Test basato su tavole di decisione
- Test basato su grafi causa-effetto

Specifica logica

- x, z: variabili di tipo invoice (strutture con campi customer, amount, date, etc.)
- current_date: il giorno in cui un'operazione viene eseguita
- f: variabile di tipo invoice_file. f(i) i-esimo elemento di f.
- file di tipo customer_file, etc.

Specifica logica

- dato un file f di record (senza ripetizioni) e un record x, pos(x,f) restituisce la posizione i di x in f
- predicato sorted_by_date con argomento di tipo invoice_file:

  sorted_by_date(y) $\equiv \forall$ i,j in Integers (i<j $\rightarrow$ f(i).date $\leq$ f(j).date)
- result e warning: variabili booleane che indicano rispettivamente successo e anomalia di un'operazione di inserimento

Specifica logica

```
for all x in Invoices, f in Invoice_Files
{sorted_by_date(f) and not exist j, k (j ≠ k and
   f(j) =f(k)}
insert(x, f)
{sorted_by_date(f) and
for all k (old_f(k) = z implies exists j (f(j) = z))
   and
for all k (f(k) = z and z ≠ x) implies exists j
   (old_f(j) = z) and
exists j (f(j).date = x.date and f(j) ≠ x)  implies
   j < pos(x, f) and
result ≡ x.customer belongs_to customer_file and
warning ≡ (x belongs_to old_f or x.date <
   current_date or ....)
}
```

Riscrittura della postcondizione

```
TRUE implies
   sorted_by_date(f) and for all k old_f(k) = z
   implies exists j (f(j) = z) and
   for all k (f(k) = z and z ≠ x) implies exists j
   (old_f(j) = z)
and
(x.customer belongs_to customer_file) implies result
and
not (x.customer belongs_to customer_file  and ...)
   implies not result
and
x belongs_to old_f implies warning
and
x.date < current_date implies warning
and
....
```

Applicazione del criterio di copertura delle condizioni

- Qualunque caso di test: verifica che il file contenga solo le fatture precedenti e quella nuova, e che sia ordinato
- Almeno un caso di test in cui il campo customer di una fattura è presente in customer_file, e uno in cui è assente
- Almeno un caso di test in cui il campo date della fattura è lo stesso di una fattura esistente, e uno in cui non lo è

Test syntax-driven

- Utilizzato nella verifica di compilatori (e di altri software basati su linguaggi formali).
- Grammatica utilizzata per definire insiemi di test.
- Esempio: test set in cui ogni produzione sintattica BNF è utilizzata almeno una volta.

Test syntax-driven

- Interprete per il seguente linguaggio:

```
<expression> ::= <expression> + <term>|
     <expression> - <term> | <term>
<term> ::= <term> * <factor> | <term> /
     <factor> | <factor>
<factor> ::= ident | ( <expression>)
```

Test syntax-driven

- Applicazione del principio di completa copertura alle regole della grammatica
- Generazione di un test per ogni regola della grammatica
  - Alcuni test possono coprire più di una regola
- La specifica è formale, quindi la generazione del test può essere automatizzata

Esempio

- Regola da testare

  `<term> ::= <term> * <factor>`

- Produzione di un caso di test:
  - <expression> ⇒ Partenza dallo scopo della grammatica
  - <expression> + <term> ⇒ Utilizzo della regola
  - <expression> + <term> * <factor> ⇒
  - <term> + <term> * <factor> ⇒
  - <factor> + <factor> * <factor> ⇒
  - ident + ident * ident

Test basato su tabelle di decisione

- Le tabelle di decisione sono un formalismo semplice per descrivere come combinazioni diverse di ingressi possono generare uscite diverse

Test basato su tabelle di decisione: specifica

- Il word processor può presentare le porzioni di testo in tre formati diversi: testo normale (plain, p), testo in grassetto (boldface, b) e testo in corsivo (italics, i). I seguenti comandi possono essere applicati a ciascuna porzione del testo: rendere il testo normale (P), rendere il testo in grassetto (B), rendere il testo in corsivo (I), enfatizzare (E) e super-enfatizzare (SE). Esistono, inoltre, comandi per specificare dinamicamente il fatto che E possa significare o B o I (questi comandi vengono denotati rispettivamente come E = B e E = I). Analogamente si può specificare dinamicamente che SE indichi B (comando SE = B), I (comando SE = I) o B e I (comando SE = B + I).

Tabella di decisione

P	*								
B		*							*
I			*						*
E				*	*				
SE						*	*	*	
E = B				*					
E = I					*				
SE = B						*			
SE = I							*		
SE = B + I								*	
action	**p**	**b**	**i**	**b**	**i**	**b**	**i**	**b,i**	**b,i**

- Righe rappresentano condizioni
- Colonne rappresentano regole: azioni come risultato delle condizioni
- Esplosione di casi di tesi

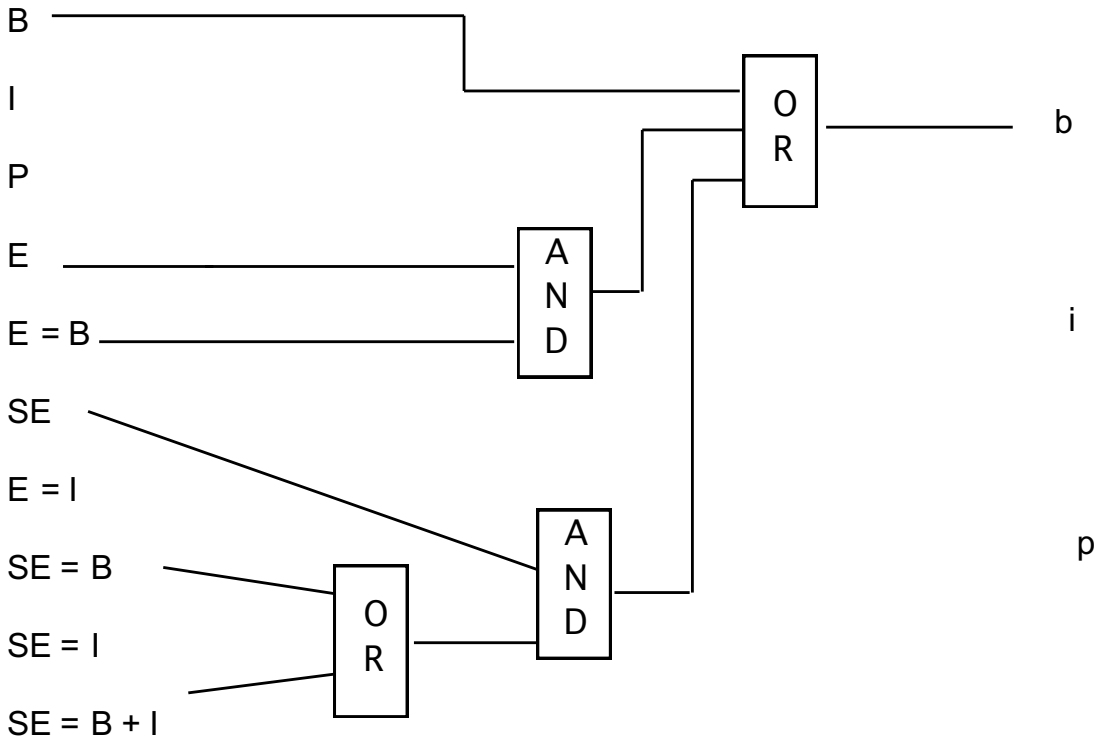
Test

- Applicazione del principio di completa copertura: generazione dei test in modo da esercitare ogni colonna.
- Applicazione esaustiva può essere molto costosa: in generale, a n condizioni possono corrispondere 2^n colonne

Grafi causa-effetto

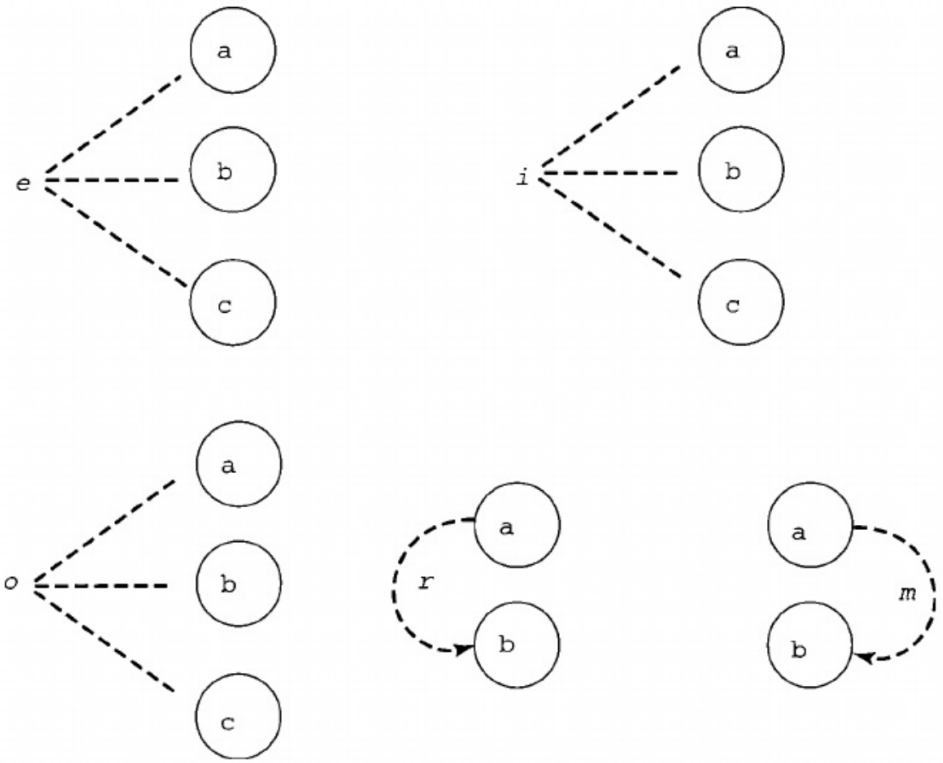
- n condizioni, m effetti
- Fissato un ordine per le condizioni, (es.: B, I, P, E, SE, E=B, E=I, SE=B, SE=I, SE=B+I) un comando di ingresso (combinazione di condizioni) si può rappresentare come n-pla di variabili booleane (es.: E=B ed E come <f,f,f,t,f,t,f,f,f,f>)
- Fissato un ordine per gli effetti (es.: b, p, i), un'uscita (combinazione di effetti) come m-pla di variabili booleane (es. p come <f,f,t>)

Grafi causa-effetto

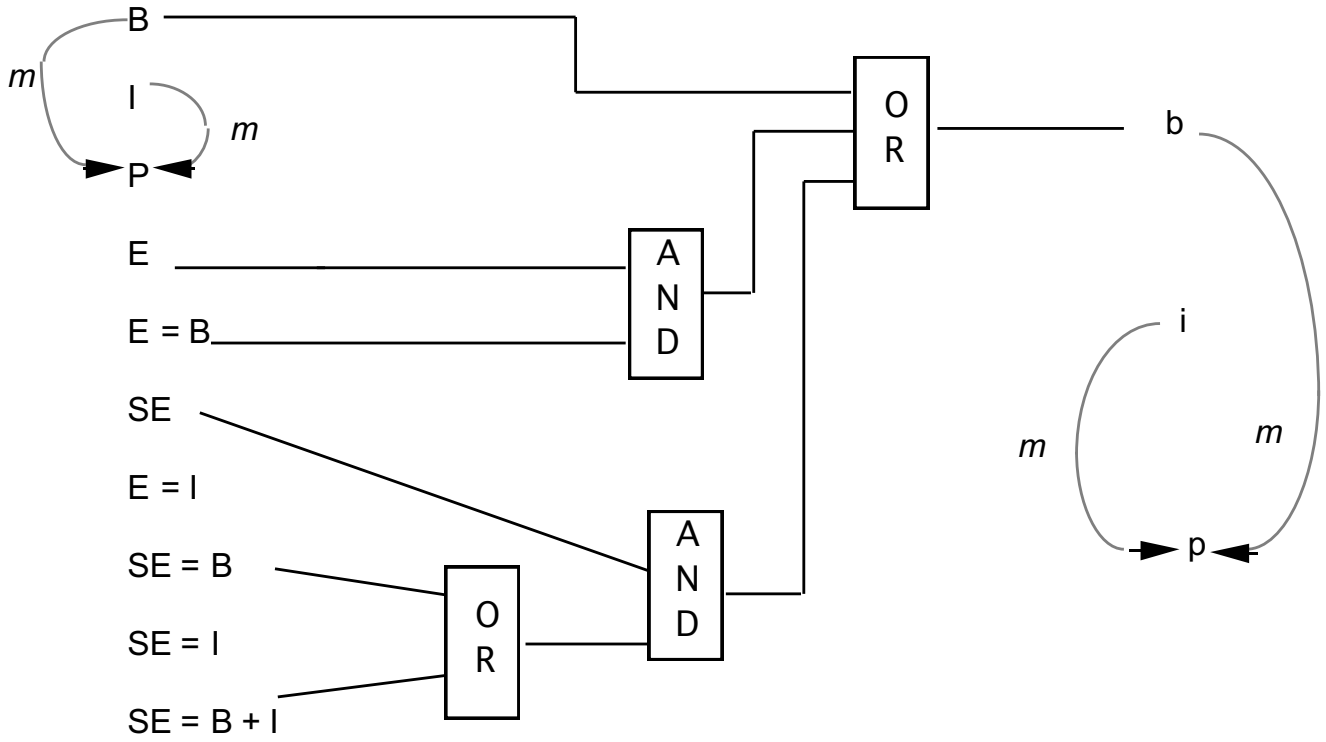


Vincoli fra variabili logiche

- *e*, al più uno; *i*, almeno uno; *o*, uno e uno solo;
- a richiede (implies) b; a maschera (implies not) b



Grafo con vincoli (causa-effetto)



Criterio di copertura

- Generare tutte le possibili combinazioni di input e controllare l'output rispetto alla specifica.
- Esempio: il comando E ed E=B dovrebbe dare l'uscita b.
- Il grafo può mostrare ***incoerenza*** (uscita corrispondente a ingressi inammissibili) e ***incompletezza*** (nessun ingresso porta a un'uscita ammissibile)

Riduzione del numero di test

- Per ogni combinazione dei valori di uscita, generazione di alcune combinazioni di input che li generano (scelte euristicamente) e propagazione all'indietro.
- Nodo OR con output t:
  - combinazioni con un solo input vero
- Nodo AND con output f:
  - combinazioni con un solo input falso

Test di condizioni di confine

- Partizione dei domini di input in classi
- Sia per il testing white-box che black-box, abbiamo supposto che il comportamento dei programmi per tutti gli elementi di una classe sia simile
- Molti errori di programmazione (come l'uso di < al posto di $\leq$) hanno effetti al confine fra le classi.
- Criterio: ***utilizzo di casi di test al confine fra le classi.***

Esempio

```
if x > y then
        S1
else
        S2
```

- Suddividendo il dominio nei due casi x>y e $x \leq y$, spesso si scelgono valori tali che x > y o x < y, trascurando x = y, che sarebbe significativo nel caso dell'uso di > al posto di $\geq$.

Il problema dell'oracolo

- *Oracolo*: entità in grado di dire se un output è corretto per un dato input.
- Necessario per l'attività di test.
- L'unica implementazione dell'oracolo potrebbe essere il programma stesso!
- Spesso la correttezza è definita in base a proprietà facili da verificare, che possono essere utilizzate come oracolo.