



UNIVERSITÀ DEGLI STUDI
DI MILANO

Ingegneria del software

Progettazione

Copertura delle decisioni

Un test T soddisfa il criterio di **copertura delle decisioni** se e solo se ogni decisione effettiva viene resa sia vera che falsa in corrispondenza di almeno un caso di test t contenuto in T

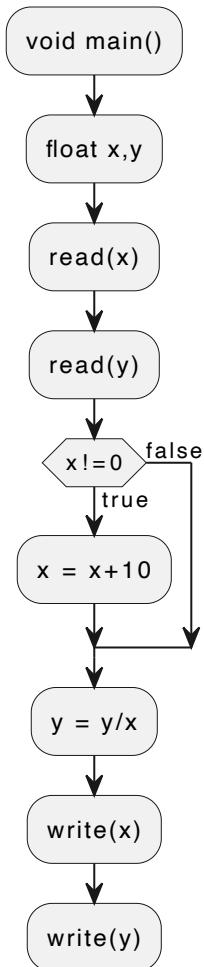
La metrica è la frazione delle decisioni che sono state rese sia vere che false su quelle per cui è possibile farlo

Implica copertura dei comandi

Se un test soddisfa la copertura delle decisioni, allora soddisfa anche la copertura dei comandi (ma non è garantito l'inverso)

Esempio

```
1 void main(){
2     float x,y;
3     read(x);
4     read(y);
5     if (x!=0)
6         x = x+10;
7     y = y/x;
8     write(x);
9     write(y);
10 }
```



Graficamente corrisponde a percorrere tutti gli archi (percorribili)

Ho bisogno di almeno due casi di test:
ad esempio $< 3, 7 >$ e $< 0, 5 >$

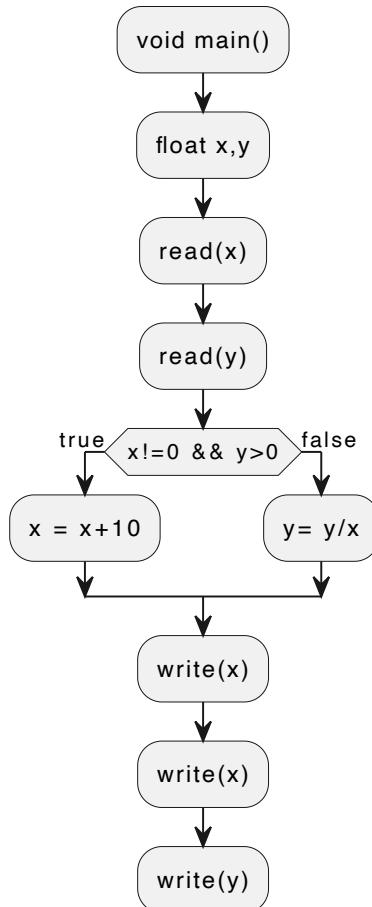
$T = \{< 3, 7 >, < 0, 5 >\}$ è quindi un test che soddisfa il criterio di copertura delle decisioni al 100%

Trova tutti i malfunzionamenti?

NO : ad esempio un possibile overflow alla somma di x alla riga 6

Esempio 2

```
1 void main(){  
2     float x,y;  
3     read(x);  
4     read(y);  
5     if (x!=0 && y>0)  
6         x = x+10;  
7     else  
8         y= y/x;  
9     write(x);  
10    write(y);  
11 }
```



Ho aggiunto anche il ramo `else` e ho messo due **condizioni** a comporre una **decisione**

Ho ancora bisogno di almeno due casi di test: ad esempio $< 3, 7 >$ e $< 3, -2 >$ vanno bene

$T = \{< 3, 7 >, < 3, -2 >\}$ è quindi un test che soddisfa il criterio di copertura delle decisioni al 100%

Trova tutti i malfunzionamenti?

NO: ad esempio il caso $< 0, 5 >$ percorrerebbe ramo *false* eseguendo divisione per zero

Copertura delle condizioni

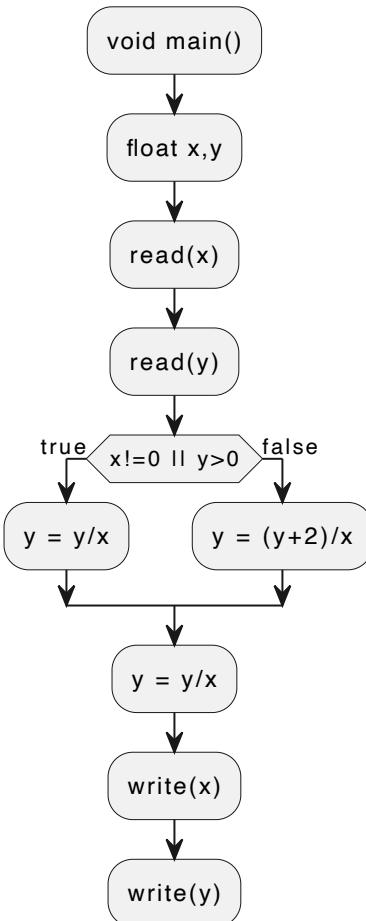
Un test T soddisfa il criterio di **copertura delle condizioni** se e solo se ogni singola condizione (effettiva) viene resa sia vera che falsa in corrispondenza di almeno un caso di test t contenuto in T

La metrica è la frazione delle condizioni che sono state rese sia vere che false su quelle per cui è possibile farlo

NON implica i criteri precedenti

Esempio

```
1 void main(){
2     float x,y;
3     read(x);
4     read(y);
5     if (x!=0 || y>0)
6         y = y/x;
7     else
8         y = (y+2)/x;
9     y = y/x;
10    write(x);
11    write(y);
12 }
```



$T = \{<0, 5>, <5, -5>\}$ è un test che soddisfa il criterio di copertura delle condizioni al 100%, ma la decisione è sempre vera

X	Y	decisione
0 (F)	5 (T)	T
5 (T)	-5 (F)	T

Ci sono anomalie sia alla riga 6 che 8, ma la seconda non verrebbe trovata non essendo coperta

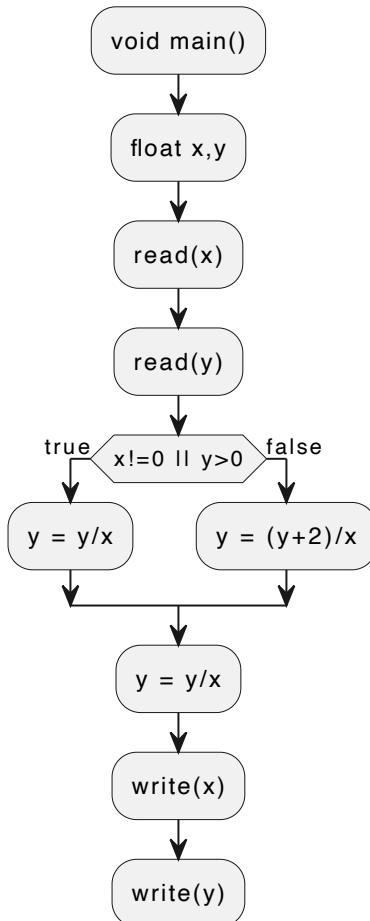
Copertura decisioni e condizioni

Un test T soddisfa il criterio di copertura delle decisioni e delle condizioni se e solo se ogni decisione vale sia vero che falso e ogni singola condizione che compare nelle decisioni del programma vale sia vero che falso per diversi casi di test in T

Questo criterio contiene i due precedenti

Esempio

```
1 void main(){
2     float x,y;
3     read(x);
4     read(y);
5     if (x!=0 || y>0)
6         y = y/x;
7     else
8         y = (y+2)/x;
9     y = y/x;
10    write(x);
11    write(y);
12 }
```



$T = \{<0, -5>, <5, 5>\}$ è un test che soddisfa il criterio di copertura delle decisioni e condizioni al 100%

X	Y	decisione
0 (F)	-5 (F)	F
5 (T)	5 (T)	T

scopre anomalia in 8 ma non quella in 6

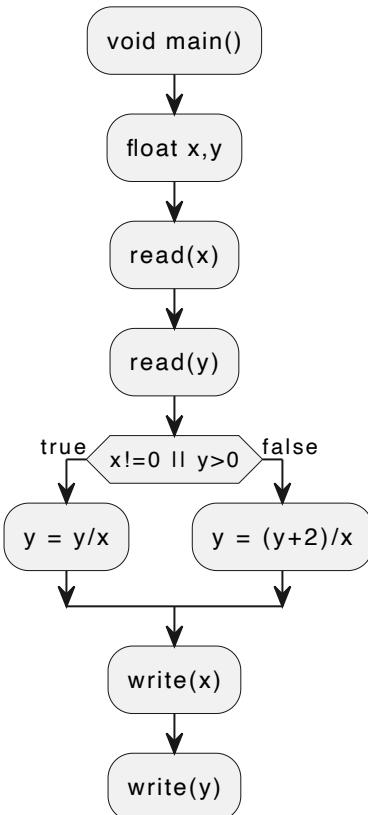
Copertura condizioni composte

Un test T soddisfa il criterio di copertura delle condizioni composte se e solo se ogni possibile composizione delle condizioni base vale sia vero che falso per diversi casi di test in T

Questo criterio contiene il precedente

Esempio

```
1 void main(){
2     float x,y;
3     read(x);
4     read(y);
5     if (x!=0 || y>0)
6         y = y/x;
7     else
8         y = (y+2)/x;
9     y = y/x;
10    write(x);
11    write(y);
12 }
```



X	Y	decisione
0 (F)	-5 (F)	F
0 (F)	5 (T)	T
5 (T)	-5 (F)	T
5 (T)	5 (T)	T

Crescita molto veloce del numero di casi al crescere del numero di condizioni base

Potrebbero esistere combinazioni non fattibili (condizioni base non indipendenti)

Copertura decisioni/condizioni modificate

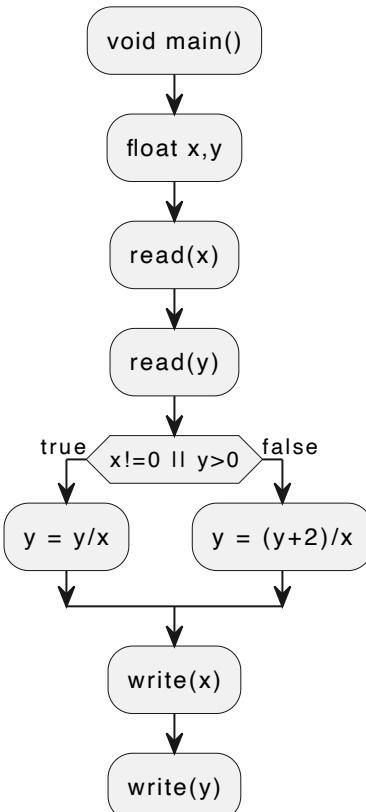
Si dà importanza, nella selezione delle combinazioni, al fatto che la modifica di una singola condizione base porti a modificare la decisione

Cioè devono esistere per ogni condizione base due casi di test che modificano il valore di una sola condizione base e che modificano il valore della decisione

Si può dimostrare che se ho N condizioni base, mi servono "solo" $N + 1$ casi di test.

Esempio

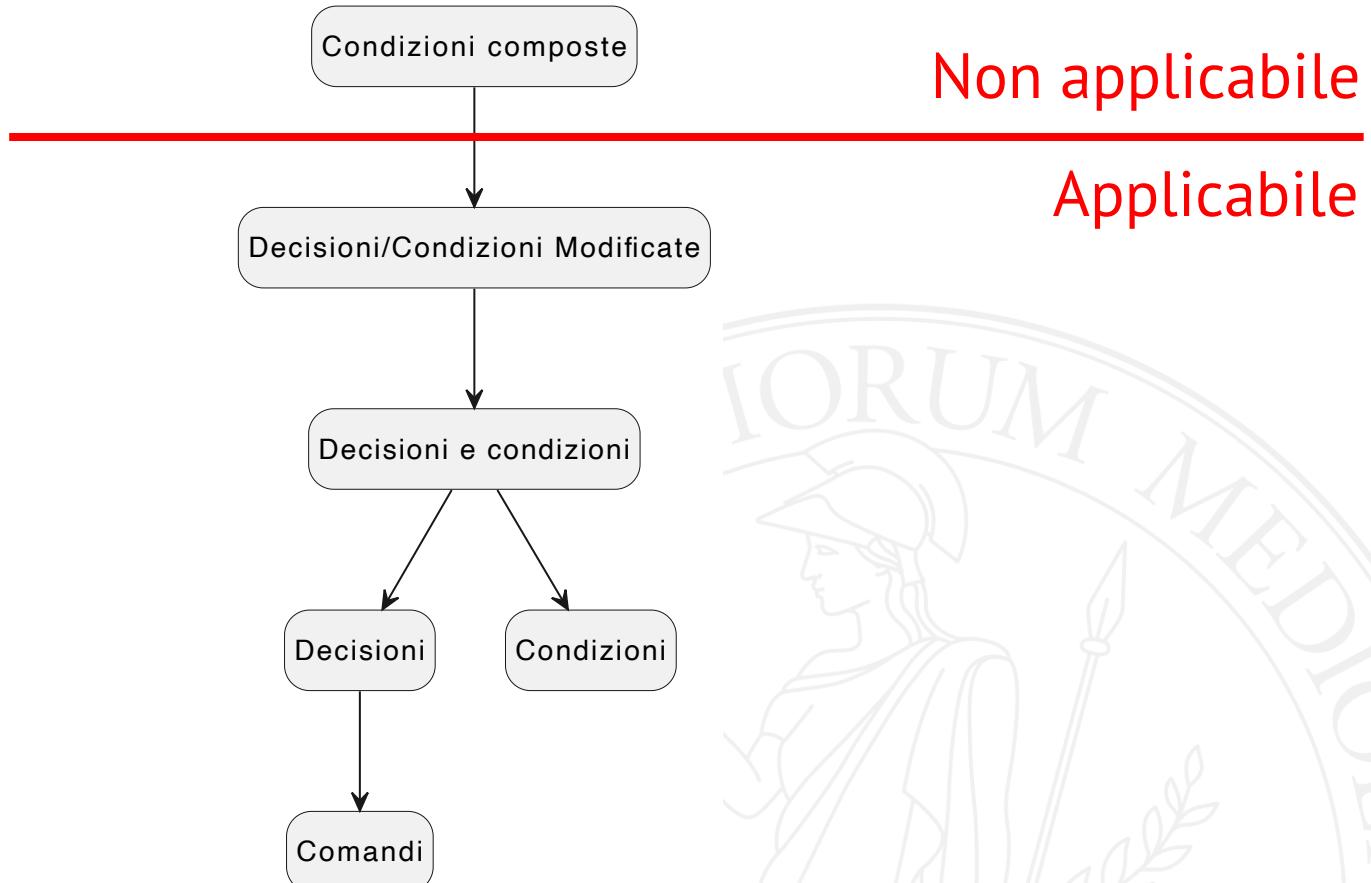
```
1 void main(){
2     float x,y;
3     read(x);
4     read(y);
5     if (x!=0 || y>0)
6         y = y/x;
7     else
8         y = (y+2)/x;
9     y = y/x;
10    write(x);
11    write(y);
12 }
```



X	Y	decision
0 (F)	-5 (F)	F
0 (F)	5 (T)	T
5 (T)	-5 (F)	T

Tolto caso vero - vero perché con un singolo cambio di condizione non era possibile cambiare decisione

Implicazioni tra criteri di copertura



Altri criteri

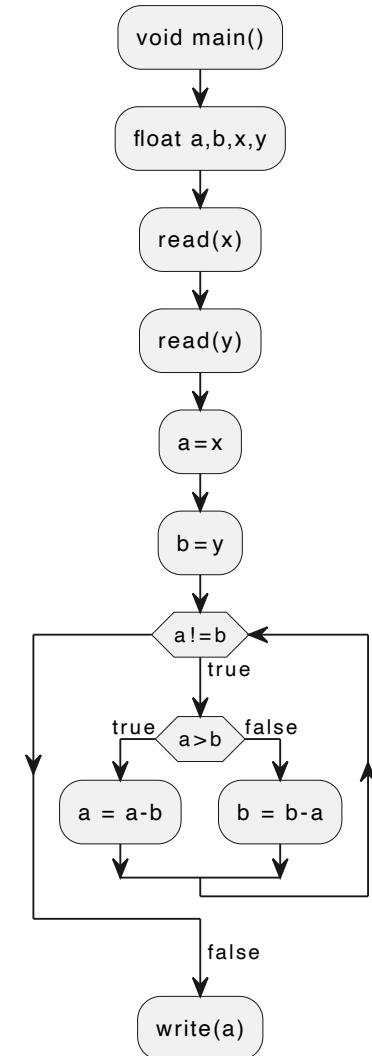
I criteri visti finora non considerano i cicli e possono essere soddisfatti da test che percorrono ogni ciclo al più una volta

Esperienza: molti errori si verificano durante iterazioni successive alla prima

- superamento limiti di un array

Occorre un criterio che tenga conto delle iterazioni

```
1 void main(){
2     float a,b,x,y;
3     read(x);
4     read(y);
5     a=x;
6     b=y;
7     while (a!=b)
8         if (a>b)
9             a = a-b;
10        else
11            b = b-a;
12    write(a);
13 }
```



Copertura dei cammini

Un test T soddisfa il criterio di copertura dei cammini se e solo se ogni cammino del grafo di controllo del programma viene percorso per almeno un caso di test in T

La metrica è la frazione dei cammini percorsi su quelli effettivamente percorribili

Molto generale, ma spesso impraticabile (anche per programmi semplici)

N-Copertura dei cicli

Un test soddisfa il criterio di n-copertura se e solo se ogni cammino del grafo contenente al massimo un numero di iterazioni di ogni ciclo non superiore a n viene percorso per almeno un caso di test

Si limita il numero massimo di percorrenze dei cicli

Quale è il valore ottimale di n?

- crescita molto rapida dei casi di test necessari al crescere di n

caso N = 2

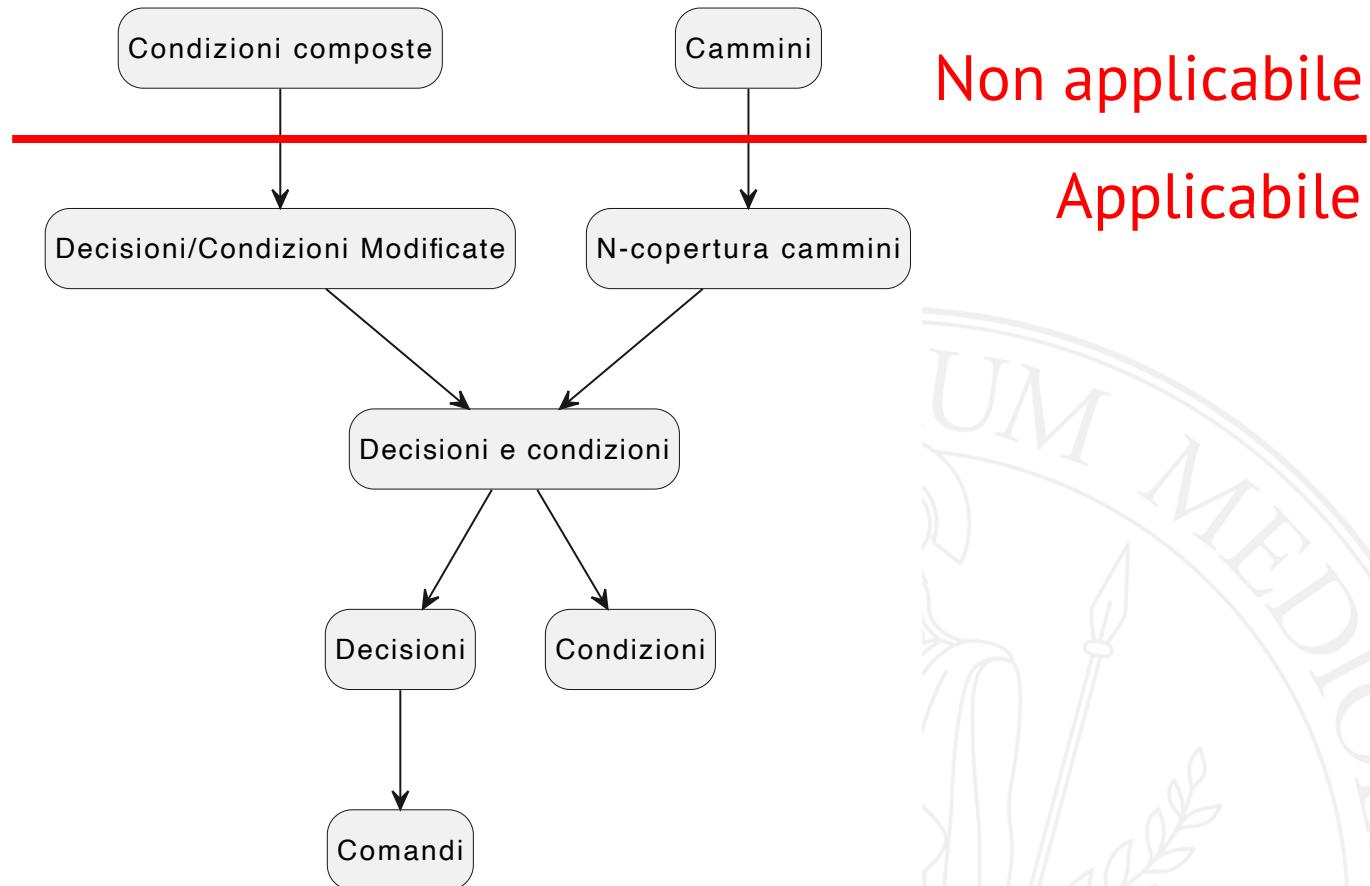
È il minimo per considerare cicli

se $n = 1$ un ciclo (`while`) sarebbe stato indistinguibile da una semplice selezione (`if`)

con $n = 2$ controllo

- casi in cui non devo entrare
- casi in cui entro una volta
- casi in cui rimango più di una volta

Implicazioni tra criteri di copertura aggiornato



Cosa si può ancora fare?

- Rimanendo nel testing strutturale possiamo fare una analisi più approfondita del codice
- Abbiamo considerato per adesso solo il flusso di controllo
- Possiamo fare considerazioni sul flusso dei dati?
 - necessita di una fase a priori di analisi statica

Analisi statica

- Si basa sull'esame di un insieme finito di elementi (le istruzioni del programma) contrariamente all'analisi dinamica (insieme degli stati delle esecuzioni)
- meno "costosa" del testing
 - non soffre del problema della "esplosione dello spazio degli stati"
- non può rilevare anomalie dipendenti da uno specifico valore assunto a run-time dalle variabili

Compilatori

- **analisi lessicale:** identificazione token del linguaggio, permette di identificare la presenza di simboli non appartenenti al linguaggio
- **analisi sintattica:** identifica relazione tra token; controlla conformità del codice alla grammatica del linguaggio
- **controllo dei tipi:** violazioni regole uso tipi
- **analisi flusso dei dati:** rileva problemi relativi a evoluzione del valore associato alle variabili

Analisi Data Flow

- I primi utilizzi sono stati fatti nel campo dell'ottimizzazione dei compilatori
- Il flusso dei dati sarebbe una analisi prettamente dinamica ma il sottoinsieme dei controlli statici è significativo
- Staticamente è possibile identificare il *tipo* di operazione che un comando esegue su una variabile
 - **definizione** se il comando assegna un valore alla variabile
 - **uso** se il comando richiede il valore di una variabile
 - **annullamento** se al termine dell'esecuzione dell'istruzione il valore della variabile non è significativo/affidabile

Analisi DF

Dal punto di vista di ciascuna variabile è possibile ridurre una sequenza di istruzioni (un cammino) ad una sequenza di

- **d**efinizioni
- **u** si
- **a**nnullamenti

Es. DF

```
1 void swap (int &x1, int &x2) {  
2     int x;  
3     x2 = x;  
4     x2 = x1;  
5     x1 = x;  
6 }
```

Anomalie rilevabili:

- x viene usata (2 volte) senza essere stata prima definita
- x1 OK?
- x2 viene definita più volte senza essere usata nel frattempo

Costruiamo le sequenze per ogni variabile:

x

auua

x1

...dud...

x2

...ddd...

Regole DF

- L'uso di una variabile deve sempre essere preceduto in ogni sequenza da una definizione senza annullamenti intermedi
- La *definizione* di una variabile deve sempre essere seguita da un uso prima di un suo annullamento o definizione
- L'*annullamento* di una variabile deve essere sempre seguito da una definizione prima di un uso o di altro annullamento

a	d	u
a	ERR	ERR
d	ERR	ERR
u		

Non sono regole assolute: dipende dal linguaggio classificarle come *warning* o *error*

Esempio

```
1 void main(){
2     float a,b,x,y;
3     read(x);
4     read(y);
5     a=x;
6     b=y;
7     while (a!=b)
8         if (a>b)
9             a = a-b;
10    else
11        b = b-a;
12    write(a);
13 }
```

a	d	u
x y a b	x	
	y	
	a	x
	b	y
	a'	a b
	b'	a b
x y a b	a b	a b
	a b	a b
	a b	a b
	a	

Sequenze DF

```
1 void main(){
2     float a,b,x,y;
3     read(x);
4     read(y);
5     a=x;
6     b=y;
7     while (a!=b)
8         if (a>b)
9             a = a-b;
10        else
11            b = b-a;
12     write(a);
13 }
```

$P(p, a)$ indica la sequenza ottenuta per la variabile a eseguendo il cammino p .

Es.

$$P([1, 2, 3, 4, 5, 6, 7, 8, 9, 7, 12, 13], a) =$$

$$a_2 d_5 u_7 u_8 u_9 d_9 u_7 u_{12} a_{13}$$

Per rappresentare P in caso di cammini contenenti cicli e decisioni si possono usare *espressioni regolari*

Esempio espressione regolare

```
1 void main(){
2     float a,b,x,y;
3     read(x);
4     read(y);
5     a=x;
6     b=y;
7     while (a!=b)
8         if (a>b)
9             a = a-b;
10    else
11        b = b-a;
12    write(a);
13 }
```

$$P([1 \rightarrow], a)$$

- Ci sono coppie sbagliate?
- Era l'unica possibile?

$a_2 \ d_5$

$a_2 \ d_5 \ u_7(\text{while-body})^* \ u_{12} \ a_{13}$

$a_2 \ d_5 \ u_7(\text{if-body})^* \ u_{12} \ a_{13}$

$a_2 \ d_5 \ u_7(\text{ (} u_8 \ (u_9d_9 \mid u_{11}) \text{) }^* \ u_{12} \ a_{13}$

$a_2 \ d_5 \ u_7(\text{ (} u_8 \ (u_9d_9 \mid u_{11}) \text{ } \color{red}{u_7} \text{) }^* \ u_{12} \ a_{13}$

Torniamo al testing e ai criteri di copertura

Osservazioni

- perché si presenti un malfunzionamento dovuto a una anomalia in una *definizione*, deve essere *usato* il valore che è stato assegnato
- un ciclo dovrebbe essere ripetuto (di nuovo) se verrà usato un valore definito alla iterazione precedente

Idea

- basare la selezione dei casi di test sulle sequenze definizione-uso delle variabili

Definizioni

- $\text{def}(i)$ è l'insieme delle variabili che sono definite in i
- $\text{du}(x, i)$ è l'insieme dei nodi j tali che:
 - $x \in \text{def}(i)$
 - x usato in j
 - esiste un cammino da i a j , libero da definizioni di x

Criterio copertura definizioni

Un test T soddisfa il criterio di copertura delle definizioni se e solo se per ogni nodo i e ogni variabile x appartenente a $\text{def}(i)$ T include un caso di test che esegue un cammino libero da definizioni da i ad almeno uno degli elementi di $\text{du}(i, x)$

$T \in C$ sse

$\forall i \in P \ \forall x \in \text{def}(i) \ \exists j \in \text{du}(i, x) \ \exists t \in T$ che esegue un cammino da i a j senza ulteriori definizioni di x

Esempio copertura definizioni

```
1 void main(){
2     float a,b,x,y;
3     read(x);
4     read(y);
5     a=x;
6     b=y;
7     while (a!=b)
8         if (a>b)
9             a = a-b;
10        else
11            b = b-a;
12    write(a);
13 }
```

Ad esempio considerando la variabile a

$$du(5, a) = \{7, 8, 9, 11, 12\}$$

$$du(9, a) = \{7, 8, 9, 11, 12\}$$

$d_5 u_7$ viene "gratis"

$d_9 u_7$ basta entrare una volta nel ciclo

$$T = \{<8, 4>\}$$

Criterio copertura degli usi

Un test T soddisfa il criterio di copertura degli usi se e solo se per ogni nodo i e ogni variabile x appartenente a $\mathrm{def}(i)$ per ogni elemento j di $\mathrm{du}(i, x)$ T include un caso di test che esegue un cammino libero da definizioni da i a j

$T \in C$ sse

$\forall i \in P \ \forall x \in \mathrm{def}(i) \ \forall j \in \mathrm{du}(i, x) \ \exists t \in T$ che esegue un cammino da i a j senza ulteriori definizioni di x

Include il precedente?

Esempio copertura usi

```
1 void main(){
2     float a,b,x,y;
3     read(x);
4     read(y);
5     a=x;
6     b=y;
7     while (a!=b)
8         if (a>b)
9             a = a-b;
10        else
11            b = b-a;
12        write(a);
13 }
```

Considerando ancora la variabile a

$du(5, a) = \{7, 8, 9, 11, 12\}$ e $du(9, a) = \{7, 8, 9, 11, 12\}$

$d_5 u_7 u_8 u_{11} u_7 u_{12}$

$\dots d_9 u_7 u_8 u_9 \dots$

$\dots d_5 u_7 u_8 u_9 \dots$

$\dots d_9 u_7 u_8 u_{12} \dots$

$\dots d_9 u_7 u_8 u_{11} \dots$

almeno due iterazioni

ad esempio $T = \{< 4, 8 >, < 12, 8 >, < 12, 4 >\}$

Criterio copertura cammini DU

Esistono diversi cammini che soddisfano il criterio precedente.

Questo criterio richiede che siano selezionati tutti.

$T \in C$ sse

$\forall i \in P \ \forall x \in \text{def}(i) \ \forall j \in \text{du}(i, x)$ **per ogni** cammino da i a j senza ulteriori definizioni di x $\exists t \in T$ che lo esegue

Oltre le variabili

Analisi del flusso si può fare anche con altri "oggetti"

- FILE

- **a** pertura (specializzata in *per lettura* o *per scrittura*)
- **c** chiusura
- **l** lettura
- **s** scrittura

Quali regole?

- l (e s e c) deve essere preceduta da a senza c intermedie
- a deve essere seguita da c prima di altra a
- legame tipo apertura ed operazioni...

Altri criteri

Copertura del budget

- Ci si ferma quando
 - sono finite le risorse (tempo, soldi, ...)
- Nessuna informazione sull'efficacia del test
- Spesso unico criterio applicato!!

