



Elementi teorici di base per il testing

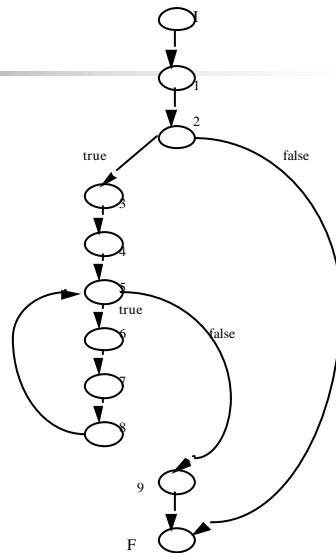


Control Flow Analysis

- Il flusso di controllo è esaminato per verificarne la correttezza.
- Il codice è rappresentato tramite un grafo, il grafo del flusso di controllo (Control flow Graph - CfG), i cui nodi rappresentano statement (istruzioni e/o predicati) del programma e gli archi il passaggio del flusso di controllo.
- Il grafo è esaminato per identificare ramificazioni del flusso di controllo e verificare l'esistenza di eventuali anomalie quali codice irraggiungibile e non strutturazione

Grafo di flusso di controllo di un programma

```
procedure Quadrato;  
  var x, y, n: integer;  
  begin  
    1. read(x);  
    2. if x > 0  
      then begin  
        3.   n := 1;  
        4.   y := 1;  
        5.   while x > 1 do  
            begin  
              6.   n := n + 2;  
              7.   y := y + n;  
              8.   x := x - 1;  
            end;  
        9.   write(y);  
      end;  
  end;
```



Elementi teorici per il testing

3

Costruzione di un CFG

Il grafo è costruito secondo la seguente notazione:

○ NODO rappresenta un'istruzione, identificato da un numero

→ ARCO rappresenta il passaggio del flusso di controllo, etichettato con {vero, falso, incond}

Un CfG può essere costruito combinando opportunamente insieme i grafi relativi alle strutture di controllo

Elementi teorici per il testing

4

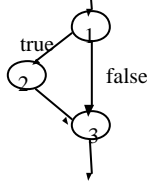


Costruzione di un CFG

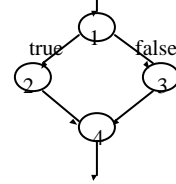
Sequenza



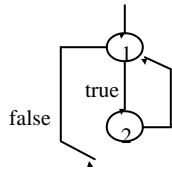
if then



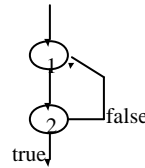
if then ... else



while do



repeat ... until



Elementi teorici per il testing

5



Definizione di CFG

Il grafo del flusso di controllo di un programma P è definito come:

$$\text{GFC}(P) = \langle N, AC, nI, nF \rangle$$

dove:

- $\langle N, AC \rangle$ è un grafo diretto con archi etichettati, $\{nI, nF\} \subseteq N$, $N - \{nI, nF\} = N_s \cup N_p$
- N_s e N_p sono insiemi disgiunti di nodi istruzione e nodi predicato
- nI ed nF sono detti rispettivamente nodo iniziale e nodo finale
- $AC \subseteq (N - \{nF\}) \times (N - \{nI\}) \times \{\text{vero}, \text{falso}, \text{incond}\}$ rappresenta la relazione flusso di controllo

Elementi teorici per il testing

6



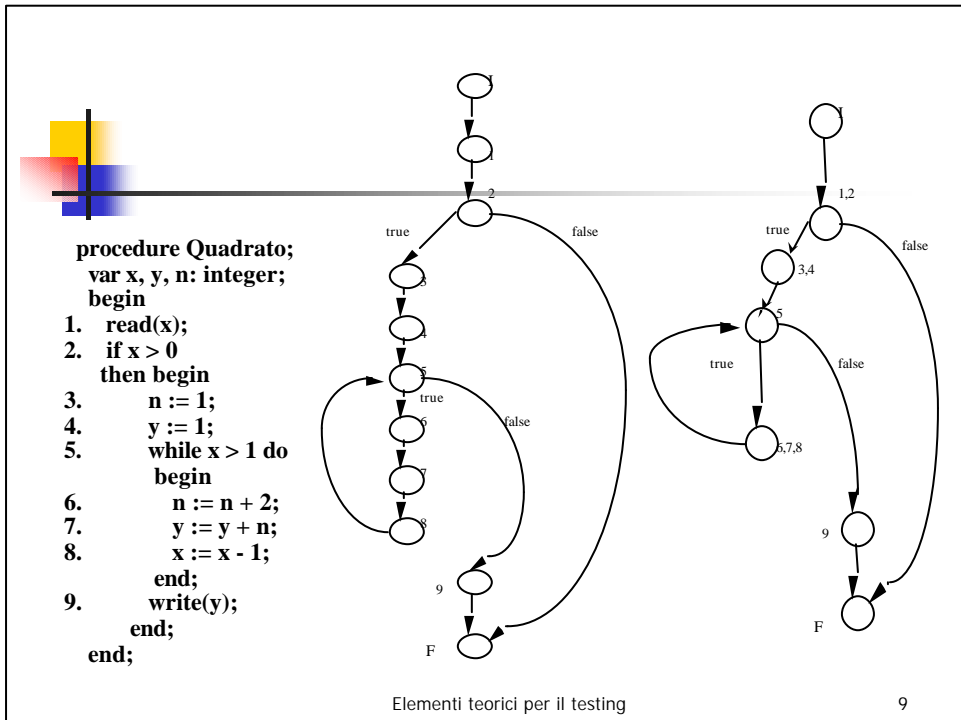
Definizione di CFG

- Un nodo $n \in N_s \cup \{nI\}$ ha un solo successore immediato e il suo arco uscente è etichettato con incond.
- Un nodo $n \in N_p$ ha due successori immediati e i suoi archi uscenti sono etichettati rispettivamente con vero e falso.
- Un GFC(P) é detto 'ben formato' se esiste un cammino dal nodo iniziale nI ad ogni nodo in $N - \{nI\}$ e da ogni nodo in $N - \{nF\}$ al nodo finale nF .
- Diremo semplicemente 'cammino' o 'cammino totale' un cammino da nI ad nF .
 - Un cammino può essere indicato tramite i nodi che sono attraversati percorrendolo.



Semplificazione di un CFG

- Un nodo ed il suo successore sono semplificabili se essi hanno un solo punto d'ingresso ed un solo punto di uscita
- I due nodi si riducono ad un solo nodo nella rappresentazione del grafo
- Analogamente sequenze di nodi verificanti tale condizione possono essere ridotte ad un unico nodo; tale nodo può essere etichettato con i numeri dei nodi in esso ridotti



Path condition

- Nel caso di esecuzioni simboliche con condizioni alcuni statement sono eseguiti solo se gli input soddisfano determinate condizioni.
- Una Path Condition (pc), per un determinato statement, indica le condizioni che gli input devono soddisfare affinché una esecuzione percorra un cammino lungo cui lo statement sia eseguito.**
- Una pc è un'espressione Booleana sugli input simbolici di un programma.
- All'inizio dell'esecuzione simbolica essa assume il valore vero ($pc := true$).
- Per ogni condizione che si incontrerà lungo l'esecuzione pc assumerà differenti valori a seconda dei differenti casi relativi ai diversi cammini dell'esecuzione.

Elementi teorici per il testing



Path condition: esempio

Es.
if C then S1 else S2;

$pc := \text{true}$
 $pc := pc \wedge C$ [pc per S1]
 $pc := pc \wedge (\text{not } C)$ [pc per S2]

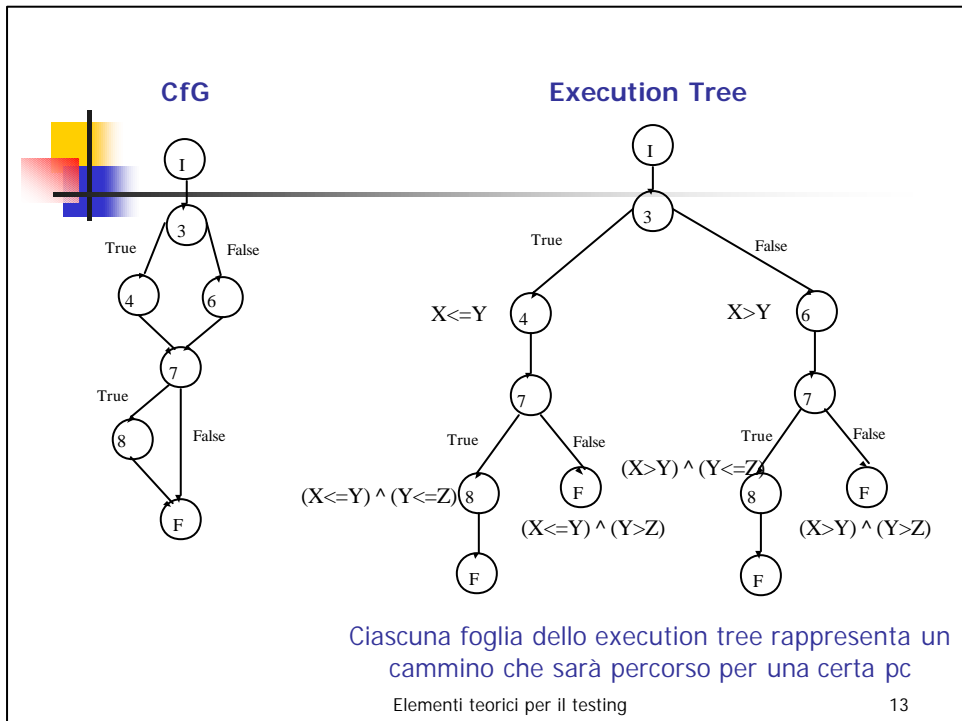


Path condition

```

1 function max (x,y,z: integer): integer;
2 begin
3   if x <= y then
4     max := y
5   else
6     max := x;
7   if max <= z then
8     max := z;
9 end;
```

Stmt	pc	max	Stmt	pc	max
1	true	?	1	true	?
<i>Case (x>y)</i>			<i>Case (x<=y)</i>		
6	X > Y	X	4	X <= Y	Y
<i>case (max < z)</i>			<i>case (max < z)</i>		
8	(X > Y) ^ (X <= Z)	Z	8	(X <= Y) ^ (Y <= Z)	Z
ritorna questo valore per max			ritorna questo valore per max		
<i>case (max > z)</i>			<i>case (max > z)</i>		
8	(X > Y) ^ (X > Z)	X	8	(X < Y) ^ (Y > Z)	Y
ritorna questo valore per max			ritorna questo valore per max		



Execution tree

- Ogni foglia dello execution tree rappresenta un cammino che sarà percorso per certi valori di input
- Le *pc* associate a due differenti foglie sono distinte; ciascuna foglia dello execution tree rappresenta un cammino che sarà percorso per la *pc* ad essa associata.
- Non esistono esecuzioni per cui sono vere contemporaneamente più *pc* (per linguaggi di programmazione sequenziali).
- Se l'output ad ogni foglia è corretto allora il programma è corretto.

Ma, quanti rami può avere un execution tree?

Elementi teorici per il testing 14



Eseguibilità dei cammini

- Diremo '*cammino eseguibile*' un cammino per il quale esiste un insieme di dati di ingresso che soddisfa la path condition.
- Diremo '*cammino non eseguibile*' ('*infeasible path*') un cammino per il quale non esiste un insieme di dati di ingresso che soddisfa la path condition.



Grafi utilizzati nel testing

- GFC(P)
- Call Graph
- Grafi di dominanza
- Grafi di Dipendenza
- Nesting tree
- Grafi causa-effetto
- Grafi delle transazioni
- Carte di struttura
- Data Flow Diagram



Call-Directed-Graph CDG(P)

é definito da:

$$\text{CDG}(P) = (\text{PP}, E, s)$$

dove

- PP è l'insieme dei sottoprogrammi di P,
- E è la relazione di chiamata $(s \cup \text{PP}) \times \text{PP}$,
- s è il main program.

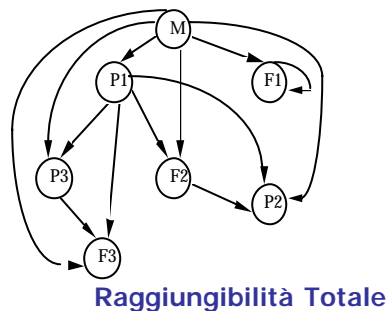
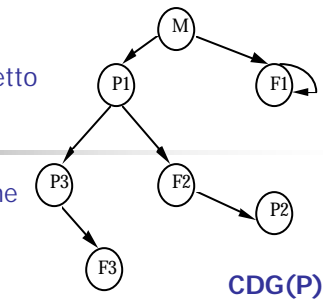


Come per GFC(P) rilevante il concetto di cammino

Circuiti e Cappi implicano Ricorsione

Raggiungibilità K-fold

Raggiungibilità Totale (Chiusura Transitiva)





Arricchimenti e varianti

...arricchimenti...

- Esempio: **Interprocedural data flow**
archi labellati con parametri formali,
effettivi, globali ...ed altre annotazioni

...varianti...

- Esempio: **Perform Call Graph in COBOL**

PP= {paragrafi o sections di P attivati mediante PERFORM}

E è la relazione di attivazione di un paragrafo o di una section
mediante l'istruzione PERFORM

s è il primo paragrafo (section) eseguito all'atto del lancio del
programma



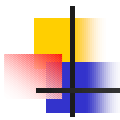
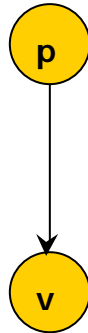
Relazione di dominanza

- Dato un GFC(P) o un CDG(P), posto $s=n_l$ ovvero $s=m$, reso aciclico tale grafo (collassando in un unico nodo le sue componenti connesse):
- Se p_x e p_y sono due nodi in un grafo aciclico, allora p_x **domina** p_y se e solo se:
 $\forall \mu_i(s, p_y), p_x \in \mu_i$
dove con $\mu_i(s, p_y)$ si indica un generico cammino da s
a p_y .
- Il nodo s domina tutti gli altri nodi;
- La relazione di dominanza è una relazione d'ordine parziale (riflessiva, antisimmetrica, transitiva).



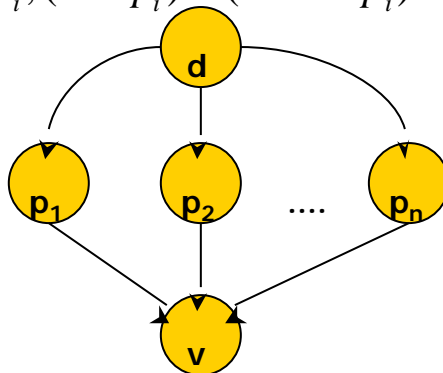
Dominatori

- **Caso 1:** Se esiste un predecessore p di v , allora p **dom** v



Dominatori

- **Caso 2:** Se i predecessori p_i di v sono più di uno $\forall p_i, (d \neq p_i) \wedge (d \text{ dom } p_i) \Rightarrow d \text{ dom } v$





Dominanza diretta

- px **direttamente domina** py se e solo se:
($px <> py$) and (px domina py) and ($\forall pz: pz <> px$
and $pz <> py$, se pz domina py allora pz domina px)
- Quindi, un nodo px domina direttamente py se è dominato da ogni altro dominatore di py diverso da py stesso
- E' possibile provare che ogni nodo n eccetto s ha un unico dominatore diretto (il dominatore più vicino a n nel cammino da s a n)



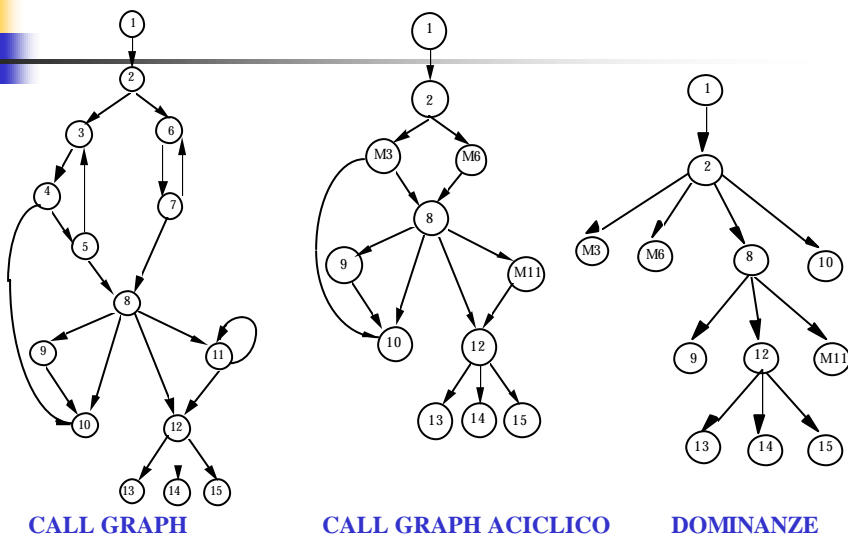
Dominatori: esempio

```
main()  
{  
1.  x=3;           1.  {1}  
2.  if(x)         2.  {1,2}  
3.    goto a;     3.  {1,2,3}  
4.  y=x+1;       4.  {1,2,4}  
5.  while(y)     5.  {1,2,4,5}  
6.    y--;       6.  {1,2,4,5,6}  
7.  a:           7.  {1,2,7}  
}
```



Albero delle dominanze

- Il grafo della riduzione riflessiva e transitiva della relazione di dominanza in CDAG e' un **albero**.
- Vi e' un arco (px,py) nell'albero se e solo se px direttamente domina py , e c' e' un cammino $\mu(pu,pv)$ nell'albero se e solo se pu domina pv .





Postdominanza

- Dato un grafo $GFC(P) = \langle N, AC, nI, nF \rangle$, siano n ed m due nodi in N . Si dice che n è **postdominato** da m se e solo se per ogni cammino $n=p_0, p_1, \dots, p_k=nF$ in $GFC(P)$, $m \in \{p_1, \dots, p_k\}$.
- La relazione di postdominanza non è riflessiva:

$$\neg (n \text{ pdom } n)$$

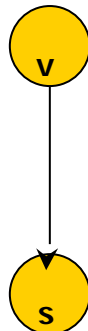
- La relazione di postdominanza è transitiva:

$$(w \text{ pdom } n) \wedge (n \text{ pdom } r) \Rightarrow (w \text{ pdom } r)$$



Postdominatori

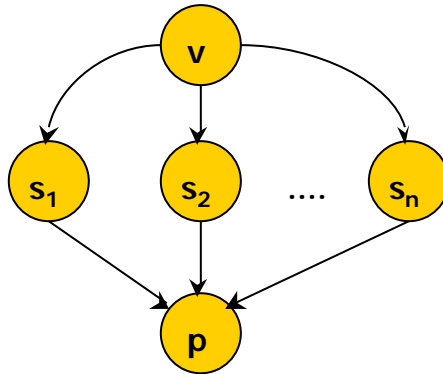
- **Caso 1:** Se esiste un successore s di v , allora s **pdom** v





Postdominatori

- **Caso 2:** Se i successori s_i di v sono più di uno:
$$\forall s_i, (p \neq s_i) \wedge (p \text{ pdom } s_i) \Rightarrow p \text{ pdom } v$$



Elementi teorici per il testing

29



Postdominatori - Teorema

- **Teorema:** se un nodo w non postdomina n allora w coincide con n oppure esiste un cammino da n al nodo finale v_M che non contiene w .

- **Dimostrazione:**

$$\neg (w \text{ pdom } n) =$$

$$\neg ((w \neq n) \wedge (\forall p = \langle n, \dots, v_M \rangle, w \in p)) =$$

$$\neg (w \neq n) \vee (\neg (\forall p = \langle n, \dots, v_M \rangle, w \in p)) =$$

$$(w = n) \vee (\exists p = \langle n, \dots, v_M \rangle, w \notin p)$$

Elementi teorici per il testing

30



Postdominatori diretti

- **Definizione:** un nodo m postdomina direttamente (o immediatamente) il nodo n se è postdominato da ogni altro postdominatore di n

$$m \text{ ipdom } n \Leftrightarrow \forall d, (d \text{ pdom } n) \wedge (d \neq m) \Rightarrow d \text{ pdom } m$$

- **Proprietà:** Ogni nodo n (diverso dal nodo finale v_M) possiede un unico postdominatore diretto, che è il postdominatore più vicino al nodo n nel cammino da n al nodo finale v_M



Albero di postdominanza

- **Definizione:** il PostDominator Tree (PDT) di un programma P è l'albero (N, E_{PDT}) in cui:
 - L'insieme dei nodi N è l'insieme degli statements P del programma;
 - L'insieme degli archi E_{PDT} è l'insieme delle coppie di statements $(n_i, n_j) \in N \times N$, tale che $n_i \text{ ipdom } n_j$
 - Il nodo radice è il nodo v_M



Postdominatori: esempio

```
main()  
{  
  1. x=3;           1. {2,7}  
  2. if(x)          2. {7}  
  3.   goto a;     3. {7}  
  4. y=x+1;        4. {5,7}  
  5. while(y)      5. {7}  
  6.   y--;        6. {5,7}  
  7. a:            7. {}  
}
```



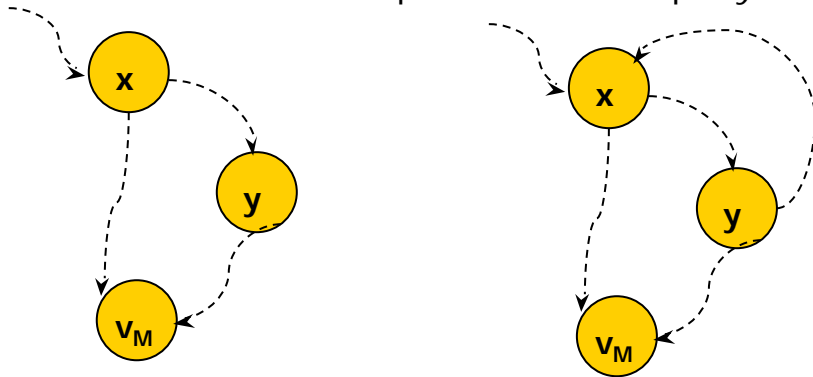
Dipendenze sul controllo

- Dato un grafo $GFC(P) = \langle N, AC, nI, nF \rangle$, siano n ed m due nodi in N . Si dice che m è **dipendente sul controllo** da n se e solo se:
 - esiste un cammino $n=p_0, p_1, \dots, p_k=m$ tale che $\forall i, 1 \leq i \leq k-1, p_i$ è postdominato da m ;
 - n non è postdominato da m .



Dipendenza sul controllo

- In pratica, x CD y se e solo se x decide se il flusso di controllo passerà o meno per y



Elementi teorici per il testing

35



Dipendenza unitaria sul controllo

- **Definizione:** un nodo x detiene una dipendenza unitaria sul controllo su y se x non è postdominato da y ed esiste un arco tra x e y nel CFG

$$xUCD y \Leftrightarrow (\langle x, y \rangle \in E) \wedge \neg(y \text{ pdom } x)$$

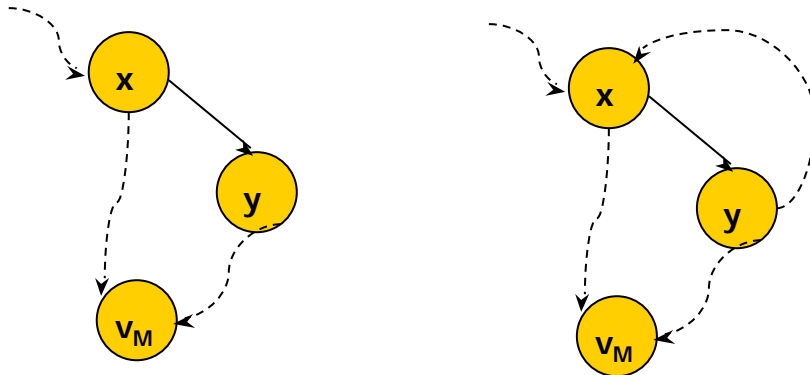
- **Proprietà:**
 - $xUCD y \Rightarrow xCD y$ (Perché non esiste un nodo intermedio z)
 - $xUCD y \Rightarrow \exists p = \langle x, \dots, v_M \rangle, y \notin p$ perché $\langle x, x \rangle \notin E$

Elementi teorici per il testing

36

Dipendenza unitaria sul controllo

- In pratica, x UCD y se e solo se x decide se y sarà o meno il prossimo statement ad essere eseguito



Elementi teorici per il testing

37

Dipendenza sul controllo: esempio

```
main()
{
1.  if (a==c) {                1.  {2,3,5}
2.    d--                      2.  {}
3.  while(a<d)                 3.  {3,4}
4.    a++; }                   4.  {}
5.  else while(a>d) {          5.  {5,6,7}
6.    a--;                     6.  {}
7.    c--; }                   7.  {}
8.  printf("%d",a);           8.  {}
}
```

Elementi teorici per il testing

38



Grafo delle dipendenze sul controllo

- Le dipendenze sul controllo possono essere espresse mediante apposito grafo.
- Ogni arco del grafo è etichettato con vero, falso o incond. Le etichette vero e falso indicano una dipendenza sul controllo sul ramo vero o falso, rispettivamente.
- Perché ogni nodo predicato abbia al più due archi uscenti (l'uno etichettato con vero e l'altro con falso), degli ulteriori nodi, chiamati nodi regione, vengono inseriti nel grafo.



Grafo delle dipendenze sul controllo

- I nodi regione riassumono l'insieme delle dipendenze sul controllo per ogni predicato. Gli archi uscenti di un nodo regione sono etichettati con incond.
- Per programmi strutturati, il grafo delle dipendenze sul controllo corrisponde all'albero dell'innesto delle strutture di controllo



Grafo delle dipendenze sul controllo

- Data un $GFC(P) = \langle N, AC, nI, nF \rangle$, il grafo delle dipendenze sul controllo di P è una struttura $GDC(P) = \langle N \cup R, DC, nI \rangle$ dove $N \cup R$ è l'insieme dei nodi, $nI \in N$ è il nodo iniziale, R è detto insieme dei nodi regione, $DC \subseteq (N \cup R \cup \{nI\}) \times N - \{nI\} \times \{\text{vero, falso, incond}\}$ è l'insieme degli archi e rappresenta l'insieme delle dipendenze sul controllo.



Grafo delle dipendenze sul controllo: proprietà

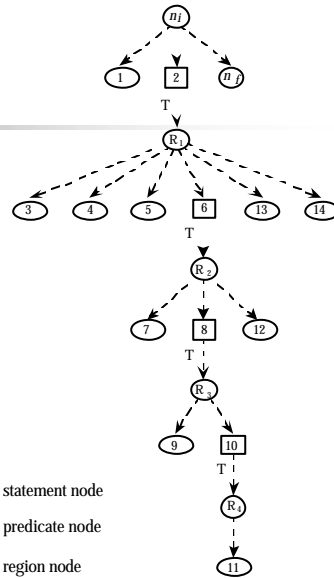
$\forall n \in N \cup R$ valgono le seguenti proprietà:

- Se $n = nI$, allora $\forall m \in N - \{nI\}$, tale che m non è dipendente sul controllo da nessun predicato in N_p , $(nI, m, \text{incond}) \in DC$;
- Se $n \in N - (N_p \cup \{nI\})$, allora $\forall m \in N \cup R$ e $\forall e \in \{\text{vero, falso, incond}\}$, $(n, m, e) \notin DC$;
- Se $n \in N_p$, allora siano $N_v(n)$ e $N_f(n)$ gli insiemi dei nodi in $N - \{nI\}$ che sono dipendenti sul controllo da n sul ramo vero e sul ramo falso, rispettivamente. Se $N_v(n) \neq \emptyset$ ($N_f(n) \neq \emptyset$, risp.) allora $\exists r \in R$ tale che $(n, r, \text{vero}) \in DC$ ($(n, r, \text{falso}) \in DC$, risp.). Inoltre, $\forall m \in N_v(n)$ ($m \in N_f(n)$), $(r, m, \text{incond}) \in DC$.

```

procedure PosAndEven;
  var n, i, k, npos, neven: integer;
  begin
1.  read(n);
2.  if n > 0
    then begin
3.      npos := 0;
4.      neven := 0;
5.      i := 1;
6.      while i <= n do
          begin
7.          read(k);
8.          if k > 0
              then begin
9.              npos := npos + 1;
10.             if (n mod 2) = 0
11.             then neven := neven + 1;
              end;
12.          i := i + 1;
          end;
13.      write(npos);
14.      write(neven);
    end;
  end;
end;

```



Data flow e dipendenze sui dati

- Data una componente di tipo procedura P, sia V l'insieme delle variabili di P (referenziate da istruzioni e predicati in P) e sia $GFC(P) = \langle N, AC, nI, nF \rangle$, il suo grafo del flusso di controllo. $\forall n \in N - \{nI, nF\}$, siano $DEF(n) \subseteq V$ e $USO(n) \subseteq V$ gli insiemi delle variabili rispettivamente definite e usate da n.
- La relazione $DEF_USO(P) \subseteq N - \{nI, nF\} \times N - \{nI, nF\} \times V$ è definita come segue:

$(n, m, v) \hat{\in} DEF_USO(P)$ se e solo se:

 - $v \in DEF(n) \cap USO(m)$;
 - \exists un cammino di lunghezza $k \geq 1$, $n = p_0, p_1, \dots, p_k = m$ in $GFC(P)$ tale che $\forall i, 1 \leq i \leq k-1, v \notin DEF(p_i)$.
- La relazione $DEF_USO(P)$ non è né riflessiva, né transitiva.



Dipendenza sui dati: esempio

```
main( )
{
1.  scanf("%d", &a);           1.  {a:1}
2.  if (a==3)                 2.  {a:1}
3.      x=a;                  3.  {a:1; x:3}
4.  else if (a==4)           4.  {a:1}
5.      a++;                  5.  {a:5}
6.  else while(x)            6.  {a:1; x:7}
7.      x--;                  7.  {a:1; x:7}
8.  printf("%d %d", a, x);   8.  {a:1,5; x:3,7}
}
```



c-use e p-use

- **Computational-use(c-use):** si ha un c-use(v) quando la variabile v è usata in output o è usata nella espressione di una assegnazione;
- **Predicate-use(p-use):** si ha un p-use(v) quando la variabile v è usata in un predicato che condiziona il control flow;



Def-use graph e def clear path

- **def-use graph:** un GFC(p) i cui ogni nodo n è etichettato anche con gli insiemi $def(n)$, $c-use(n)$, $p-use(n)$;

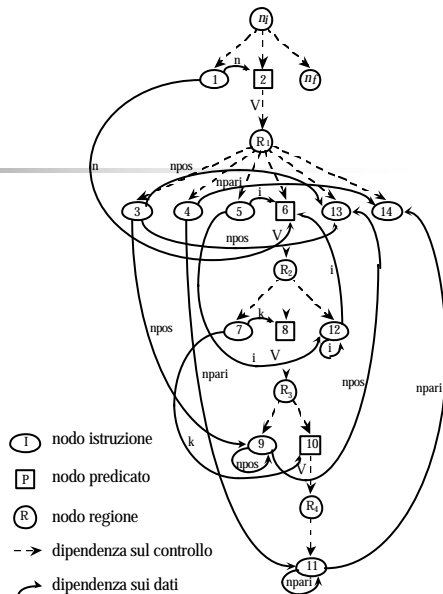
una particolare classe di cammini:

- **definition clear path (def-clear):** con riferimento ad una variabile v , un cammino $\langle i, n_1, \dots, n_K, J \rangle$ si dice def-clear(v) se per ogni nodo n_i del cammino $v \notin DEF(n_i)$.



Grafo delle dipendenze

- Data una componente di tipo procedura P , il suo grafo delle dipendenze è una struttura $GD(P) = \langle N \cup R, DC, V, DD, nI \rangle$, dove $\langle N \cup R, DC, nI \rangle$ è il grafo delle dipendenze sul controllo di P , V è l'insieme delle variabili in P e DD è un insieme di archi etichettati e rappresenta la relazione $DEF_USO(P)$ su $N - \{nI, nF\} \times N - \{nI, nF\} \times V$.
- L'insieme delle variabili V rappresenta l'insieme delle etichette degli archi in DD .

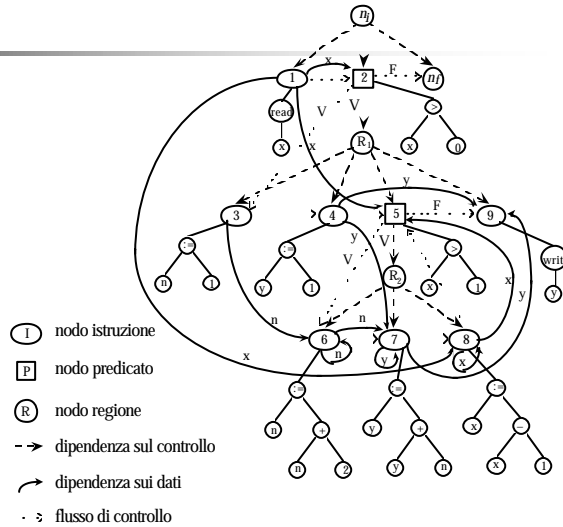


Grafo rappresentazioni unificate

```

procedure Quadrato;
var x, y, n: integer;
begin
1. read(x);
2. if x > 0
then begin
3.   n := 1;
4.   y := 1;
5.   while x > 1 do
begin
6.     n := n + 2;
7.     y := y + n;
8.     x := x - 1;
end;
9.   write(y);
end;
end;

```



Variabili Live

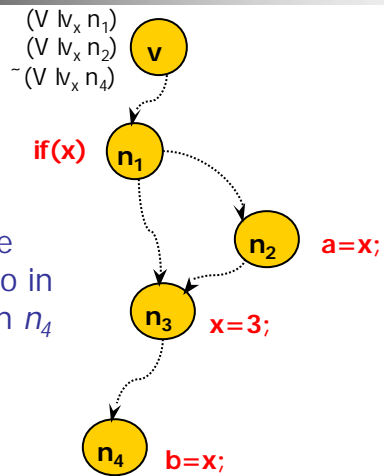
- Definizione:** la variabile x è live al nodo v se esiste un cammino nel CFG da v a n tale che x non è ridefinito in tale cammino, e n usa x .

$$v lv_x n \Leftrightarrow \exists p = \langle v, \dots, m, \dots, n \rangle, (n \text{ usa } x), \\ (m \neq m) \wedge (m \neq v) \wedge \neg(m \text{ def } x)$$

- Proprietà:** la relazione lv_x non è riflessiva né transitiva.



Live variables



La variabile x è live
Al nodo v per l'uso in
 n_1 e n_2 , ma non in n_4