Analisi Statica in Compilazione

Analisi statica in compilazione

- I compilatori effettuano una analisi statica del codice per verificare che un programma soddisfi particolari caratteristiche di correttezza statica, per poter generare il codice oggetto
  - ✓ ***analisi lessicale***: consiste nell'identificazione dei singoli elementi (token) componenti il programma (keywords, identificatori, simboli del linguaggio)
  - ✓ ***analisi sintattica***: consiste nell'esaminare le relazioni tra gli elementi identificati durante l'analisi lessicale, che devono obbedire alle regole della grammatica del linguaggio
  - ✓ ***analisi semantica***: rileva altri errori, come l'utilizzo di variabili non dichiarate, effettua il *controllo dei tipi* nelle espressioni

Analisi statica in compilazione

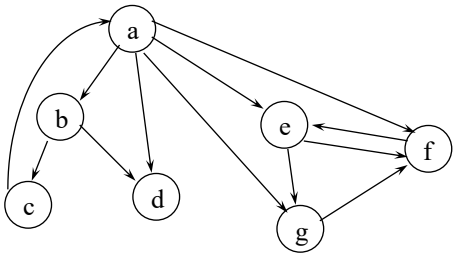
- Le informazioni e le anomalie che può rilevare un compilatore dipendono dalle caratteristiche del linguaggio e dalle facility di cui esso dispone.
  - ✓ Es.: linguaggi con regole di visibilità statica dei nomi permettono la rilevazione di un maggiore numero di anomalie di quelli con regole di visibilità dinamiche.
- La generazione di Cross Reference List risulta molto utile in successive analisi del codice per l'individuazione di anomalie non rilevabili dal compilatore.
- Tipici anomalie identificabili
  - ✓ nomi di identificatori non dichiarati, incoerenza tra tipi di dati coinvolti in una istruzione, incoerenza tra parametri formali ed effettivi in chiamate a subroutine, codice non raggiungibile dal flusso di controllo

3

3

Grafi e modelli di rappresentazione dei programmi

Un grafo diretto (o orientato) è una coppia (N,E) dove N è un insieme di nodi e $E \subseteq N \times N$ è un insieme di archi



N = {a, b, c, d, e, f, g}

E = {(a, b), (a, d), (a, e), (a, f),
(a, g), (b, c), (b, d), (c, a),
(e, f), (e, g), (f, e), (g, f)}

NB: gli archi del grafo possono essere etichettati ...

4

4

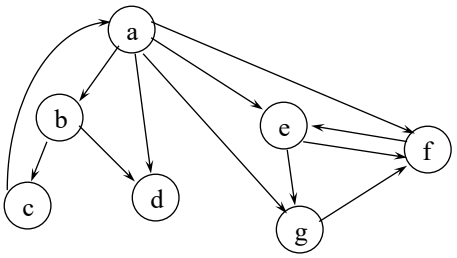
Cammini

Per ogni arco (n, m) in E, n è detto un predecessore di m e n è detto un successore di m. Un nodo può avere 0 o più successori e 0 più predecessori

Un cammino di lunghezza k-1 è una sequenza di nodi <n1, n2, ..., nk> tali che $\forall i,\ 1 \le i \le k-1,\ (n_i, n_{i+1}) \in E$

Esempi: <a, e, g, f> e <b, c, a> sono cammini

<e, f, g> non è un cammino



5

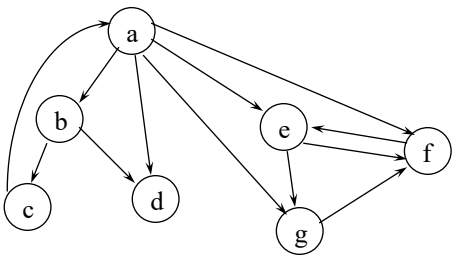
5

Circuiti (o cicli)

Un circuito è un cammino <n1, n2, ..., nk> tale che nk = n1

Esempi:

Circuiti: <a, b, c, a> e <e, g, f, e>
<e, f, g, e> non è un circuito



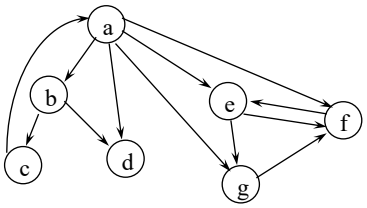
6

6

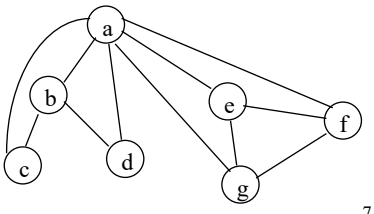
Raggiungibilità e grafi non orientati

Un nodo m si dice raggiungibile da un nodo n se esiste un cammino di lunghezza k-1 <n1, n2, ..., nk>, tale che n1 = n e nk = m

Esempi: f è raggiungibile da a,
g è raggiungibile da a,
f non è raggiungibile da d.



Un grafo si dice non orientato se i suoi archi non hanno un verso

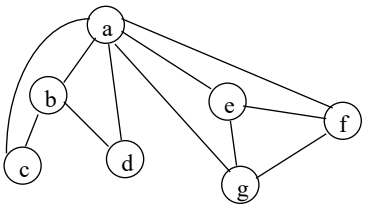


7

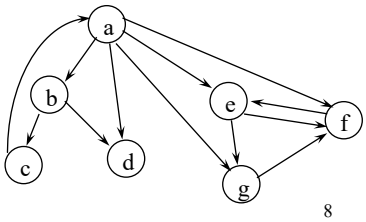
7

Raggiungibilità e grafi non orientati

Un grafo orientato si dice connesso se ogni suo nodo è raggiungibile da ogni altro nodo sul grafo che si ottiene eliminando l'orientamento degli archi.
Il grafo dei nostri esempi è connesso ...



Un grafo orientato si dice fortemente connesso se ogni nodo è raggiungibile da ogni altro nodo (ossia se esiste un ciclo che coinvolge tutti i nodi del grafo).
Il nostro grafo non è fortemente connesso ...



8

8

Un modello di rappresentazione dei programmi (1/2)

Il ***Grafo del Flusso di Controllo*** di un programma P è una quadrupla GFC(P) = (N, E, ni,n f) dove:

(N,E) è un grafo diretto con archi etichettati

$ni \in N, nf \in N, N - \{ni, nf\} = Ns \cup Np$

Ns e Np sono insiemi disgiunti di nodi, ossia $Ns \cap Np = \varnothing$ che rappresentano rispettivamente istruzioni e predicati

$E \subseteq (N - nf) \times (N - ni) \times \{true, false, uncond\}$

rappresenta la relazione di flusso di controllo

9

9

Un modello di rappresentazione dei programmi (2/2)

ni ed nf sono detti nodo iniziale e nodo finale

Un nodo in $Ns \cup \{ni\}$ ha un solo successore immediato ed il suo arco è etichettato con uncond.

Un nodo in Np ha due successori immediati e i suoi archi uscenti sono etichettati rispettivamente con true e false

Un GFC(P) e' 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

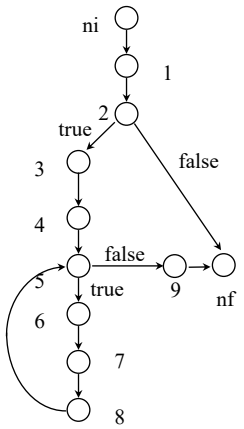
Diremo semplicemente cammino o cammino totale un cammino da ni a nf

10

10

Esempio

```
void Quadrato() {
int x, y, n;
1 cin >> x;
2 if (x>0) {
3         n = 1;
4         y = 1;
5         while (x>1) {
6                   n = n + 2;
7                   y = y + n;
8                   x = x - 1;
                    }
9         cout << y;
          }
}
```



11

11

Costruzione del control flow graph per programmi strutturati (1)

Il grafo è costruito secondo la seguente notazione:

NODO rappresenta un' istruzione, identificato da un numero

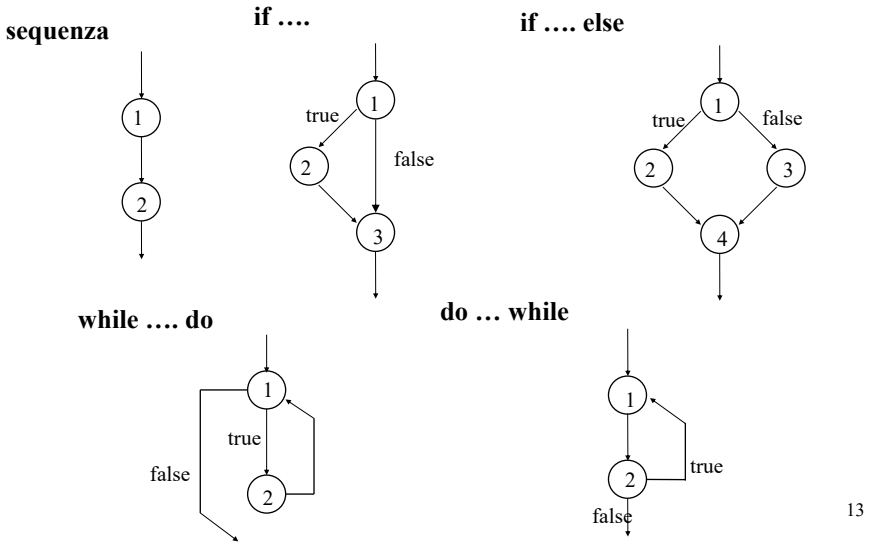
ARCO rappresenta il passaggio del flusso di controllo, etichettato con {vero, falso, incond}

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

12

12

Costruzione del control flow graph per programmi strutturati (2)



13

13

Semplificazione di un CFG

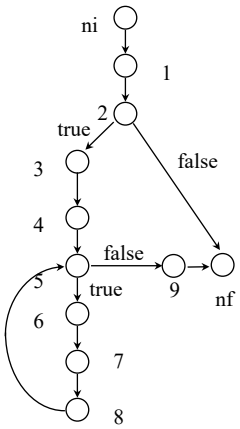
- Sequenza di nodi possono essere collassate in un solo nodo, purché nel grafo semplificato vengano mantenuti tutti i branch (punti di decisione e biforcazione del flusso di controllo)
- tale nodo può essere etichettato con i numeri dei nodi in esso ridotti

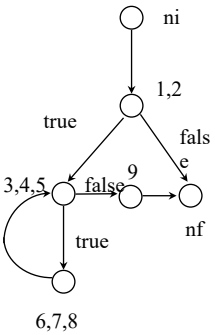
14

14

Esempio

```
void Quadrato() {
int x, y, n;
1 cin >> x;
2 if (x>0) {
3         n = 1;
4         y = 1;
5         while (x>1) {
6                 n = n + 2;
7                 y = y + n;
8                 x = x - 1;
                  }
9         cout << y;
          }
}
```





15

15

Analisi del flusso di controllo

- Il flusso di controllo è esaminato per verificarne la correttezza.
- Il codice è rappresentato tramite il Control flow Graph (CfG), i cui nodi rappresentano statement (istruzioni eo 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

16

16

Analisi del flusso dati (1)

- Consente di rilevare anomalie sull'utilizzo di variabili sui diversi cammini di esecuzione.
- Operazioni eseguite dalle istruzioni su una variabile ***x***:
  - ✓ ***definizione (d)***: assegna un valore alla variabile ***x*** (istruzioni di assegnamento e istruzioni di input)
  - ✓ ***uso (u)***: usa il valore della variabile ***x*** in un' istruzione di output, in una espressione per il calcolo di un' altra variabile, o in un predicato
  - ✓ ***annullamento (a)***: al termine dell' esecuzione dell' istruzione il valore assegnato alla variabile non è più significativo
  - ✓ Es.: nell'espressione

    a:=b+c;

    la variabile *a* è definita mentre *b* e *c* sono usate

17

17

Analisi del flusso dati (2)

- Analisi delle sequenze di definizioni, usi e annullamenti sui cammini del GFC
- La definizione di una variabile, così come un annullamento, cancella l'effetto di una precedente definizione della stessa variabile, ovvero ad essa è associato il nuovo valore derivante dalla nuova definizione (o il valore nullo)

18

18

Esempio

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

Operazioni:

variabile ***x***: ***auu***

variabile ***x1***: ***dud***

variabile ***x2***: ***ddd***

... anomalie sulle variabili x e x2 ...

19

19

Esempio corretto

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

Operazioni:

variabile ***x***: ***adu***

variabile ***x1***: ***dud***

variabile ***x2***: ***dud***

20

20

Regole

- R1: L'uso di una variabile *x* deve essere sempre preceduto in ogni sequenza da una definizione della stessa variabile *x*, senza annullamenti intermedi.
  - ✓ un uso non preceduto da una definizione può corrispondere al potenziale uso di un valore non determinato
- R2: Una definizione di una variabile *x* deve essere seguita da un uso della variabile *x*, prima di un'altra definizione o di un annullamento di *x*.
  - ✓ una definizione non seguita da un uso corrisponde all'assegnamento di un valore non utilizzato e quindi potenzialmente inutile

21

21

In alcuni casi però …

```
x = ...;
if(...)   x = ...;
... = ... x ...;
```

sui due cammini si ha:

ddu (ramo true)

du (ramo false)

... cattiva strutturazione ...

22

22

Espressioni cammino/variabile

L'espressione relativa ad un cammino *p* di un programma P per la variabile *x* è indicata con P(*p*;*x*)

```
1   ……
2   x = …..
3   if ….  x = ….
4   …. =   …. x ….
5   ….
```

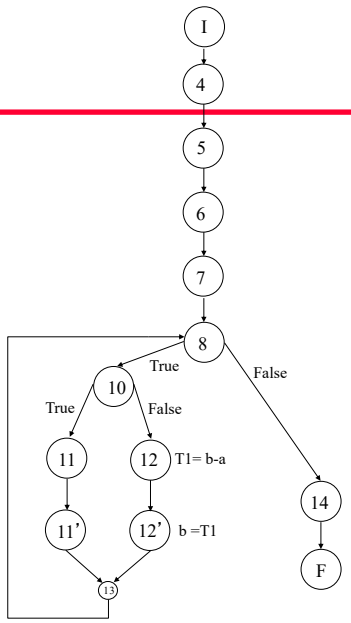
P([…, 2, 4, ….]; *x*) = (…du…)

P([…, 2, 3, 4, ….]; *x*) = (…ddu…)

23

Esempio (1)

```
1 void gcd ()
2{
3int x,y,a,b;
4 cin >> x;
5 cin >> y;
6 a =x;
7 a =y;
8 while (a !=b)
9     {
10      if (a>b)
11            a = a - b;
12      else b = b - a;
13  }
14 cout << "il massimo comune divisore è " << a;
15 }
```



24

Esempio (2)

A ciascun nodo del CfG è possibile associare l'insieme delle variabili definite in esso, quello delle variabili usate e quello delle variabile annullate.

Nodo	Var. definite	Var usate	Var. annullate
3			x, y, a, b
4, 5	x, y		
6	a	x	
7	a	y	
8		a, b	
10		a, b	
11		a, b	
11'	a		
12		a, b	
12'	b		
14		a	

E' quindi possibile scrivere l'espressione P(*p*;*x*) facendo riferimento a tali insiemi

P([I,3,4,5,6,7,8,10,11,11',8,14,F];*a*) = (-a--dduuuduu-)

P([I,3,4,5,6,7,8,10,11,11',8,14,F];*b*) = (-a----uuu-u--)

Anomalia ...dd... per *a*
anomalia ...a---u... per *b*
dovuta ad errore a linea 7[25]
7 b:= y;

25

Espressioni regolari

- Usate per rappresentare espressioni relative ad una variabile corrispondenti a più cammini
- Un'espressione regolare è definita a partire da un alfabeto finito A (nel nostro caso A = {a, d, u, -} e dalle seguenti regole ricorsive:
  - ✓ ε , stringa nulla, è un'espressione regolare
  - ✓ ogni simbolo di A è un'espressione regolare
  - ✓ se e1 ed e2 sono espressioni regolari, allora lo sono anche le espressioni che si formano da queste con l'uso degli operatori sequenza (.), alternativa (+) e ciclo (*); quindi e1.e2, e1+e2, e1* sono espressioni regolari
  - ✓ niente altro è un'espressione regolare

26

26

Esempio

P([I →];*a*) = a--ddu(u(ud+u)u)*u

P([→ 8];*a*) = -a-dd

[n →] indica tutti i cammini uscenti dal nodo n, nodo n escluso;

[→ n] indica tutti i cammini entranti nel nodo n, nodo n escluso

27

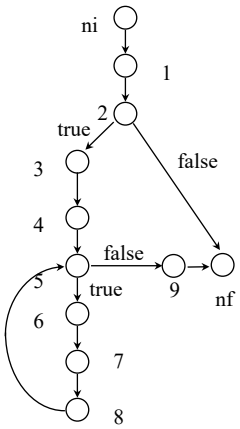
Ancora sull'analisi del flusso dati ...

- Espressioni regolari per modellare cammini multipli
  - ✓ Costoso costruire un'espressione regolare per ogni variabile
- Algoritmi per l'analisi del flusso dati efficienti
  - ✓ ***reaching definitions***: quali istruzioni (nodi del GFC) una definizione di una variabile ***x*** raggiunge senza essere ***"uccisa"*** da una nuova definizione di ***x*** ?
  - ✓ ***reacheable uses***: da quali definizioni della variabile *x* è raggiungibile un uso della stessa variabile ?
  - ✓ *situazioni anomale*: se una definizione di ***x*** non raggiunge nessun uso di ***x***, o se un uso di ***x*** non è raggiunto da nessuna definizione di ***x***.

28

Esempio

```
void Quadrato() {
int x, y, n;
1 cin >> x;
2 if (x>0) {
3         n = 1;
4         y = 1;
5         while (x>1) {
6                   n = n + 2;
7                   y = y + n;
8                   x = x - 1;
                    }
9         cout << y;
          }
}
```



La definizione di x al nodo 1 raggiunge gli usi di x ai nodi 2, 5 e 8;

L'uso di x al nodo 8 è raggiungibile dalle definizioni di x ai nodi 1 e 8