

1

Slicing: un po' di storia ...

- Inventato da Mark Weiser alla fine degli anni 70'
- Tecnica per la decomposizione dei programmi basata sull'analisi del flusso di controllo e del flusso dati: estrazione di slice
- Applicazioni iniziali: debugging e parallelizzazione
- Fornisce una risposta alla domanda: "Quali istruzioni influenzano il calcolo della variabile v all'istruzione p?"
- Caratteristiche di una slice secondo la definizione di Weiser: executable, backward, e static

© Andrea De Lucia Program Slicing 2

2

Slicing: un po' di storia ...

- In circa 40 anni migliaia di articoli pubblicati sul tema (anche un numero speciale doppio della rivista "Information and Software Technology")
- Diverse definizioni e varianti di slicing: dynamic, quasi-static, conditioned, decomposition, transform slicing, ...
- Applicazioni in diversi settori del Software Engineering: testing, integration, maintenance, comprehension, metrics, reverse engineering, reuse, and reengineering

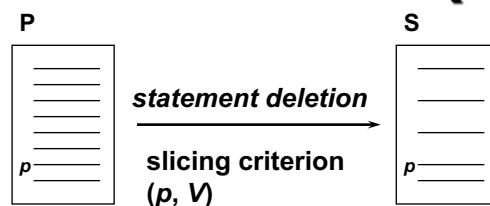
© Andrea De Lucia

Program Slicing

3

3

Definizione di Slice (Weiser)



- S è un sottoinsieme eseguibile di P
- S preserva il comportamento di P rispetto al criterio di slicing (p, V) :
 lo stato prima dell'esecuzione dell'istruzione p ristretto alle variabili in V è lo stesso per le esecuzioni sia di P che di S .
- Esiste una definizione formale di slice ...

© Andrea De Lucia

Program Slicing

4

4

Un esempio di slice

```
void main() {
    int n, i, k, npos, neven;
1.  scanf("%d", &n);
2.  if (n > 0) {
3.      npos = 0;
4.      neven = 0;
5.      i = 1;
6.      while (i <= n) {
7.          scanf("%d", &k);
8.          if (k > 0) {
9.              npos = npos + 1;
10.             if (n % 2 == 0)
11.                 neven = neven + 1;
12.             }
13.         }
14.     }
}
```

Slicing criterion(13, npos)

```
void main() {
    int n, i, k, npos;
1.  scanf("%d", &n);
2.  if (n > 0) {
3.      npos = 0;
4.      i = 1;
5.      while (i <= n) {
6.          scanf("%d", &k);
7.          if (k > 0)
8.              npos = npos + 1;
9.          i = i + 1;
10.     }
11. }
}
```

© Andrea De Lucia

Program Slicing

5

5

Un'altra slice

```
void main() {
    int n, i, k, npos, neven;
1.  scanf("%d", &n);
2.  if (n > 0) {
3.      npos = 0;
4.      neven = 0;
5.      i = 1;
6.      while (i <= n) {
7.          scanf("%d", &k);
8.          if (k > 0) {
9.              npos = npos + 1;
10.             if (n % 2 == 0)
11.                 neven = neven + 1;
12.             }
13.         }
14.     }
}
```

Slicing criterion (13, npos)

```
void main() {
    int n, i, k, npos;
1.  scanf("%d", &n);
2.  if (n > 0) {
3.      npos = 0;
4.      neven = 0;
5.      i = 1;
6.      while (i <= n) {
7.          scanf("%d", &k);
8.          if (k > 0)
9.              npos = npos + 1;
10.         }
11.     }
12. }
13. printf("%d \n", npos);
14. }
```

© Andrea De Lucia

Program Slicing

6

6

Il control flow graph (CFG)

- Un control flow graph di un programma P è una quadrupla $CFG = (N, E, n_i, n_f)$, dove:
 - $N \cup \{n_i, n_f\}$ è l'insieme dei nodi;
 - $E \subseteq N \cup \{n_i\} \times N \cup \{n_f\}$ è l'insieme degli archi;
 - per ogni nodo $n \in N \cup \{n_f\}$ esiste un cammino da n_i ad n (n_i è il nodo iniziale);
 - per ogni nodo $n \in N \cup \{n_i\}$ esiste un cammino da n ad n_f (n_f è il nodo finale);
 - l'insieme dei nodi N è partizionabile in due sottoinsiemi disgiunti N_s e N_p che rappresentano rispettivamente le istruzioni e i predicati di P ;
 - ogni arco in E rappresenta un possibile trasferimento del controllo; in particolare i nodi $n \in N_s \cup \{n_i\}$ hanno un solo arco uscente, mentre i nodi $n \in N_p$ hanno due archi uscenti etichettati con *true* e *false* rispettivamente

© Andrea De Lucia

Program Slicing

7

7

Un esempio di CFG

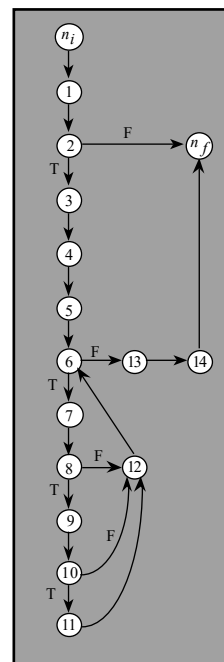
```

void main() {
    int n, i, k, npos, neven;
1.  scanf("%d", &n);
2.  if (n > 0) {
3.      npos = 0;
4.      neven = 0;
5.      i = 1;
6.      while (i <= n) {
7.          scanf("%d", &k);
8.          if (k > 0) {
9.              npos = npos + 1;
10.             if (n % 2 == 0)
11.                 neven = neven + 1;
12.             i = i + 1;
13.         }
14.         printf("%d \n", npos);
15.         printf("%d \n", neven);
16.     }
17. }
    
```

© Andrea De Lucia

Program Slicing

8



8

Traiettorie di stato

- Una traiettoria di stato di un programma P è una sequenza di coppie $T = (p_1, s_1), (p_2, s_2), \dots, (p_k, s_k)$, dove per ogni i :
 - p_i è un nodo del CFG di P ;
 - p_1, p_2, \dots, p_k è un cammino sul CFG di P ;
 - s_i è una funzione che mappa le variabili di P sui valori che assumono immediatamente prima dell'esecuzione di p_i
- In altre parole una traiettoria di stato di un programma è una traccia della sua esecuzione che evidenzia i valori delle variabili prima dell'esecuzione di ogni istruzione

© Andrea De Lucia

Program Slicing

9

9

Criterio di Slicing

- Un criterio di slicing di un programma P è una coppia $C = (p, V)$ dove p è un istruzione (nodo del CFG) di P e V è un sottoinsieme delle variabili di P .
 - Un criterio di slicing determina una funzione di proiezione Proj_C che elimina da una traiettoria di stato tutte le coppie che non iniziano con p e restringe la funzione di mapping s alle sole variabili in V
- Sia $T = (p_1, s_1), (p_2, s_2), \dots, (p_k, s_k)$ una traiettoria di stato di un programma P , allora:
 - $\text{Proj}_{(p, V)}(p_i, s_i) = \begin{cases} \lambda & \text{se } p_i \neq p \text{ (empty sequence)} \\ (p_i, s_i|V) & \text{se } p_i = p \end{cases}$
 - $\text{Proj}_{(p, V)}(T) = \text{Proj}_{(p, V)}(p_1, s_1) \text{ Proj}_{(p, V)}(p_2, s_2) \dots \text{Proj}_{(p, V)}(p_k, s_k)$

© Andrea De Lucia

Program Slicing

10

10

Slice

- Una slice di un programma P su un criterio di slicing $C = (p, V)$ è ogni programma eseguibile S con le due seguenti proprietà:
 - S è ottenuto da P mediante l'eliminazione di zero o più statement;
 - Se P si arresta su un input I con traiettoria di stato T , allora anche S si arresta su input I con traiettoria di stato T' e $\text{Proj}_C(T) = \text{Proj}_C(T')$, dove $C' = (\text{succ}(p), V)$ e $\text{succ}(p)$ è il più vicino successore di p che è in S , o p stesso se p è in S .
- Possono esistere più slice per un programma P e un criterio di slicing C ;
- Esiste almeno una slice per un programma P e un criterio di slicing C (il programma P stesso).

© Andrea De Lucia

Program Slicing

11

11

Calcolo della slice minima

- ◆ Problemi di indecidibilità: *“non è possibile calcolare la slice minima dati un programma P e un criterio di slicing $C = (p, V)$ ”* (deriva dall'indecidibilità del problema della terminazione)
- ◆ Approssimazione (Weiser's algorithm): *computing the least solution to a set of data flow equations relating a Control Flow Graph (CFG) node to the variables which are relevant at that node with respect to the slicing criterion*
- ◆ In pratica: a partire dall'istruzione p , calcolare tutte le istruzioni che influenzano direttamente o indirettamente i valori che assumono le variabili in V immediatamente prima dell'esecuzione di p .

© Andrea De Lucia

Program Slicing

12

12

Slicing e Program Dependence Graph

- Diversi algoritmi proposti, ma il più affascinante resta quello basato sul program dependence graph
- Il concetto di influenza di Weiser può infatti essere espresso in termini di due tipi di dipendenza:
 - dipendenza dal controllo
 - dipendenza sui dati
- Il calcolo di una slice si riduce ad un problema di raggiungibilità all'indietro (*backward*) sugli archi di un *program dependence graph* (un semplice algoritmo per il calcolo della chiusura transitiva)
- In questo caso il criterio di slicing è del tipo (p, v) dove p è un nodo e v è una variabile referenziata in p

© Andrea De Lucia

Program Slicing

13

13

Dipendenze sul controllo

- Dati un control flow graph $CFG = (N, E, n_i, n_f)$ e due nodi n ed m in N ,
 - m *postdomina* n se ogni cammino da n ad n_f contiene m ;
 - m è dipendente sul controllo da n se:
 - esiste un cammino (p_1, p_2, \dots, p_k) con $n = p_1$ e $m = p_k$, tale che m postdomina p_i per $1 < i < k$;
 - m non postdomina n .
- Informalmente questo significa che il nodo n rappresenta un predicato ossia ha due archi uscenti:
 - seguendo un arco m viene sicuramente eseguito;
 - seguendo l'altro arco m non viene necessariamente eseguito

© Andrea De Lucia

Program Slicing

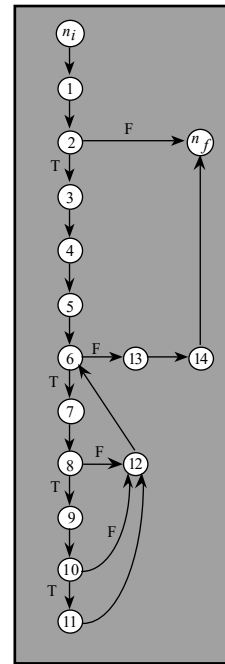
14

14

Dipendenze sul controllo: esempi

Il nodo 11 dipende sul controllo dal nodo 10

Il nodo 7 dipende sul controllo dal nodo 6



© Andrea De Lucia

Program Slicing

15

15

Il Control Dependence Graph

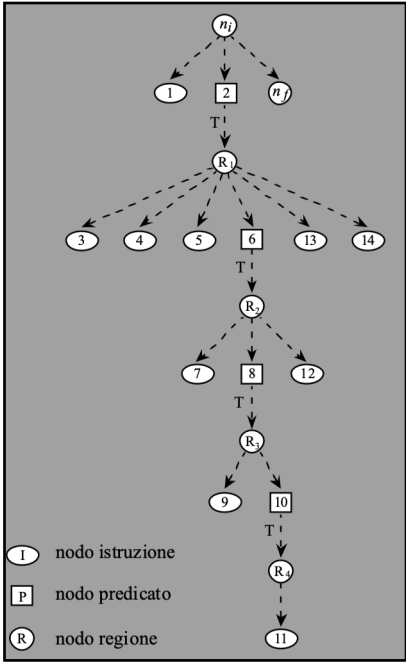
- Un control dependence graph di un programma P è una quadrupla $CDG = (N, E, n_i, n_f)$, dove:
 - $N \cup \{n_i, n_f\}$ è l'insieme dei nodi;
 - $N = N_s \cup N_p \cup R$ dove N_s e N_p sono definiti come in CFG e R è un insieme di nodi regione;
 - $E \subseteq N_p \cup R \cup \{n_i\} \times N \cup \{n_f\}$ è l'insieme degli archi e rappresenta la relazione dipendenza sul controllo;
 - i nodi regione riassumono le dipendenze sul controllo, ossia raggruppano i nodi con dipendenze comuni;
 - di conseguenza i nodi $n \in N_p$ hanno al più due archi uscenti labellati *true* o *false* e di solito diretti verso nodi regione.

© Andrea De Lucia

Program Slicing

16

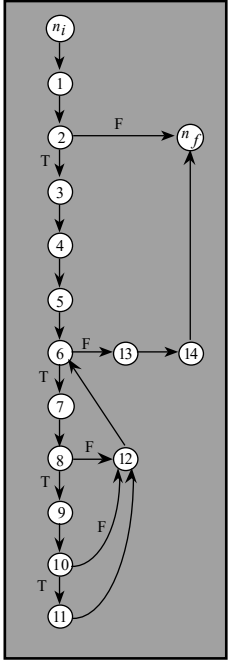
16



I nodo istruzione
 P nodo predicato
 R nodo regione

Un esempio di CDG

Calcolo di un CDG: un algoritmo proposto da Ferrante et al. con complessità di tempo $O(n^2)$



© Andrea De Lucia
Program Slicing
17

17

Dipendenze sui Dati

- Esistono diversi tipi di dipendenze sui dati definite in letteratura; in particolare la dipendenza sul flusso dati è quella che è utilizzata per il calcolo di una slice
- Dati un control flow graph $CFG = (N, E, n_i, n_f)$ di un programma P , due nodi n ed m in N e una variabile v di P ,
 - $DEF(n)$ è l'insieme delle variabili *definite* al nodo n ;
 - $USE(m)$ è l'insieme delle variabili *usate* al nodo m ;
 - $DEF_USE(P)$ è l'insieme delle triple definizione-uso (n, m, v) , dove v è definita in n ed usata in m ed esiste un cammino da n ad m (esclusi n ed m) su cui v non è ridefinita (*def-clear path*).

© Andrea De Lucia

Program Slicing

18

18

Esempio di triple def-use

```
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
                  then neven := neven + 1;
                11. end;
            12. i := i + 1;
            end;
          13. write(npos);
          14. write(neven);
        end;
      end;
  end;
```

(1, 2, n)
(1, 6, n)
(3, 9, npos)
(3, 13, npos)
(4, 11, neven)
(4, 14, neven)
(5, 6, i)
(5, 12, i)
(7, 8, k)
(7, 10, k)
(9, 9, npos)
(9, 13, npos)
(11, 11, neven)
(11, 14, neven)
(12, 6, i)
(12, 12, i)

© Andrea De Lucia

Program Slicing

19

19

Il Program Dependence Graph

- Ad un control dependence graph viene sovrapposto un altro grafo che rappresenta le dipendenze sui dati

Legend:

- I: nodo istruzione
- P: nodo predicato
- R: nodo regione
- >: dipendenza sul controllo
- ->: dipendenza sui dati

© Andrea De Lucia

Program Slicing

20

20

Un algoritmo di slicing basato su PDG

```

procedure ComputeSlice(in  $G$ : a program dependence graph,
                         $CS$ : a slicing criterion ( $p, v$ );
                        out  $Slicelist$ : a set of nodes)

  begin
     $Slicelist := \emptyset$ ;
     $Worklist := \{p\}$ ;
    while  $Worklist \neq \emptyset$  do
      Select and remove a node  $n$  from  $Worklist$ ;
       $Slicelist := Slicelist \cup \{n\}$ ;
      for all nodes  $m$  such that  $m \notin Slicelist$  and  $n$  is control
        dependent or data dependent on  $m$  do
         $Worklist := Worklist \cup \{m\}$ ;
      endfor
    endwhile
  end
  
```

© Andrea De Lucia

Program Slicing

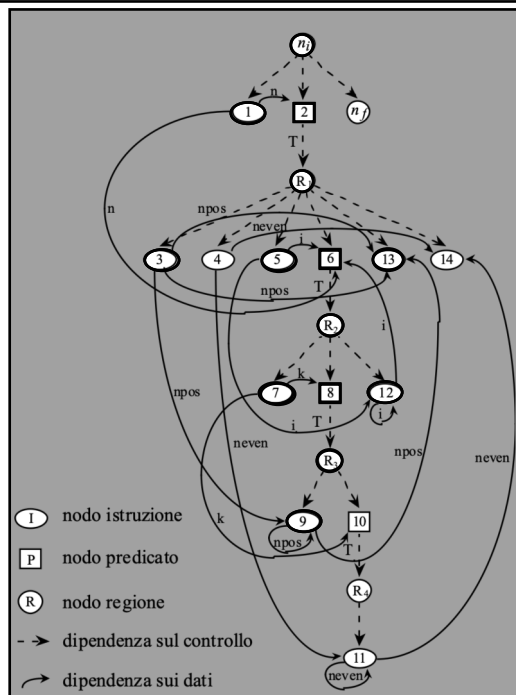
21

21

Esempio

- Criterio di slicing: (13, npos)
- L'evoluzione della *Worklist* è in rosso
- L'evoluzione della *Slicelist* è in blu
- I nodi calcolati corrispondono al primo esempio di slice visto + il nodo del criterio di slicing: 1, 2, 3, 5, 6, 7, 8, 9, 12, 13

© Andrea De Lucia



22

Altri algoritmi

- ◆ Un algoritmo parallelo proposto da Mark Harman et al. (1995)
 - Il control flow graph di un programma è convertito in un network di processi o concorrenti
 - L'esecuzione del network produce la slice
- ◆ Algoritmi per programmi non strutturati e programmi con i puntatori

© Andrea De Lucia

Program Slicing

23

23

Backward vs forward slicing

- Il program dependence graph si presta oltre che ad un'analisi di tipo *backward*, anche ad un'analisi di tipo *forward*
- Una *forward slice* risponde ad una domanda del tipo: "quali istruzioni sono influenzate dall'istruzione *p* del criterio di slicing ?"
- Questo tipo di analisi è particolarmente importante in *impact analysis*: quali istruzioni vengono impattate da una modifica all'istruzione *p* ?

© Andrea De Lucia

Program Slicing

24

24

Decomposition Slicing

- Inventato da Keith Gallagher e J. Lyle, ha come criterio di slicing solo una variabile (nessuna istruzione)
- Una slice di decomposizione si ottiene facendo l'unione delle slice rispetto a tutte le istruzioni del programma che referenziano la variabile v
- Gallagher e Lyle hanno dimostrato diverse proprietà sulle slice di decomposizione, come ad esempio
 - una slice di decomposizione ha una slice complemento rispetto al programma
 - esiste un ordinamento parziale tra le slice di decomposizione di un programma
- Utilizzato in software maintenance

© Andrea De Lucia

Program Slicing

25

25

Slicing Interprocedurale

- Considera le chiamate tra procedure;
- Una slice contiene statement di diverse procedure che influenzano i valori assunti dalle variabili del criterio di slicing (p, V) prima dell'esecuzione di p ;
- Un primo algoritmo proposto da Weiser produce slice imprecise (sovradimensionate)
 - *The algorithm does not take into account for calling contexts when descending in called procedures*

© Andrea De Lucia

Program Slicing

26

26

Slicing Interprocedurale e System Dependence Graph (SDG)

- Horwitz et al. hanno proposto un algoritmo più preciso basato sul *system dependence graph*, l'estensione a livello interprocedurale del program dependence graph
- Phase 1: does not descend into called procedures, uses dependences across actual parameters
- Phase 2: does not ascend to call sites

© Andrea De Lucia

Program Slicing

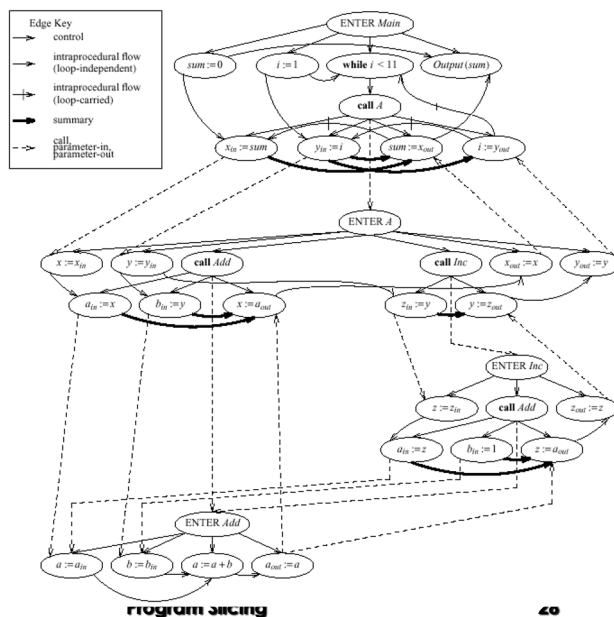
27

27

Classico esempio di SDG

```

procedure Main
  sum := 0
  i := 1
  while i < 11 do
    call A(sum, i)
  od
  print(sum)
end
procedure A(x, y)
  call Add(x, y)
  call Increment(y)
return
procedure Add(a, b)
  a := a + b
return
procedure Increment(z)
  call Add(z, 1)
return
    
```



© Andrea De Lucia

Program Slicing

28

28

Applicazioni: Function Extraction

- Una slice S su un criterio di slicing (p, V) di un programma P rappresenta un insieme eseguibile di istruzioni che sono necessarie per calcolare i valori delle variabili in V al punto p
- Se V rappresenta l'insieme delle variabili di output di una funzione, allora S è una sotto-funzione di P
 - Quindi lo slicing è particolarmente indicato per la decomposizione di un programma nelle sue sotto-funzioni ai fini sia della comprensione che del riuso o della reingegnerizzazione
 - Il problema in questo caso è la individuazione di un significativo criterio di slicing a partire dalla specifica della sotto-funzione

© Andrea De Lucia

Program Slicing

29

29

Individuazione del criterio di slicing

- ◆ **External functionality:** criterio di slicing costituito variabili e istruzioni di output, fermare l'algoritmo quando si raggiungono istruzioni e variabili di input
 - Using dynamic analysis, use case analysis, ...
- ◆ **Internal functions:** più difficile ...
 - *Identifying the set of variables is human intensive*
 - *How to identify the program point ?*
 - *When the computation of the slice should stop ?*
- ◆ Un approccio: fermare il calcolo della slice appena si incontra la definizione di variabili di input per la funzione cercata (Lanubile and Visaggio, 1997)
 - *Identifying the set of input variables and adding them to the slicing criterion*

© Andrea De Lucia

Program Slicing

30

30

Specification driven slicing

- ◆ Una funzione interna dovrebbe trovarsi su un sottografo one-in / one-out del CFG
- ◆ Il criterio di slicing dovrebbe contenere anche l'istruzione di input, oltre che quella di output della componente one-in / one-out
 - *The latter is the statement of a usual slicing criterion ...*
- ◆ Usare la specifica (formale) della funzione (pre- and post-conditions) e l'esecuzione simbolica per individuare le due istruzioni
 - ◆ Starting statement: path-condition → pre-condition
 - Ending statement: path-condition → post-condition
- ◆ Calcolare la slice usando varianti di algoritmi classici

© Andrea De Lucia

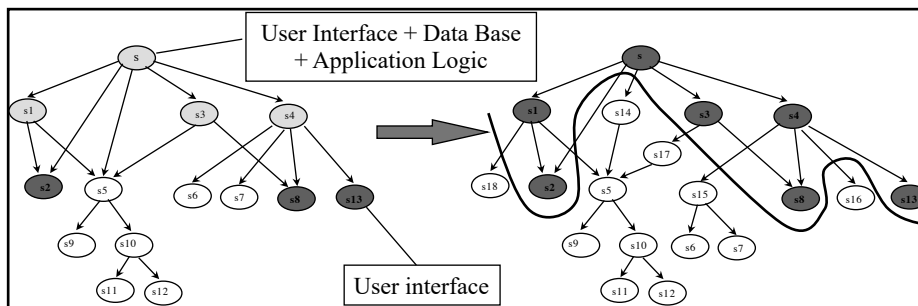
Program Slicing

31

31

Applicazioni: Client-Server Restructuring

- ◆ Client component:
 - Implementa l'interfaccia utente e controlla l'esecuzione
- ◆ Server component
 - Implementa funzioni di business e di gestione del DB



© Andrea De Lucia

Program Slicing

32

32

Applicazioni: Client-Server Restructuring

- ♦ **Primo passo (automatico): Individuazione della componente client**
 - Slicing intraprocedural and interprocedural control dependences from I/O statements
 - ... is this slicing ? ...
- ♦ **Secondo passo (semi-automatico): ristrutturazione client/server**
 - Extracting business functions and database components from subroutines of the client component
 - Encapsulating business and database components into called subroutines

© Andrea De Lucia

Program Slicing

33

33

Dynamic Slicing

- Inventato da Korel e Lasky
- Una slice statica preserva il comportamento di un programma rispetto al criterio di slicing e rispetto ad ogni possibile input (per questo statica)
- Una slice dinamica preserva il comportamento del programma rispetto ad un particolare input e quindi rispetto ad un'unica traiettoria di stato
- Permette di calcolare slice più piccole
- Molto utile per il debugging e per il program comprehension
- Algoritmi: data flow equations e PDG based

© Andrea De Lucia

Program Slicing

34

34

Quasi Static Slicing

- Venkatesh ha proposto una via di mezzo tra static e dynamic slicing: il quasi-static slicing
- Un criterio di slicing quasi statico definisce i valori solo per alcune variabili di ingresso
- Una slice quasi statica è quindi definita rispetto ad un possibile insieme di input (definiti dal variare delle variabili di input *libere*) e quindi ad un insieme di traiettorie di stato corrispondenti
- Il quasi static slicing combina insieme le tecniche di *partial* (o *mixed*) *computation* e di program slicing

© Andrea De Lucia

Program Slicing

35

35

Generalizzazione: conditioned slicing

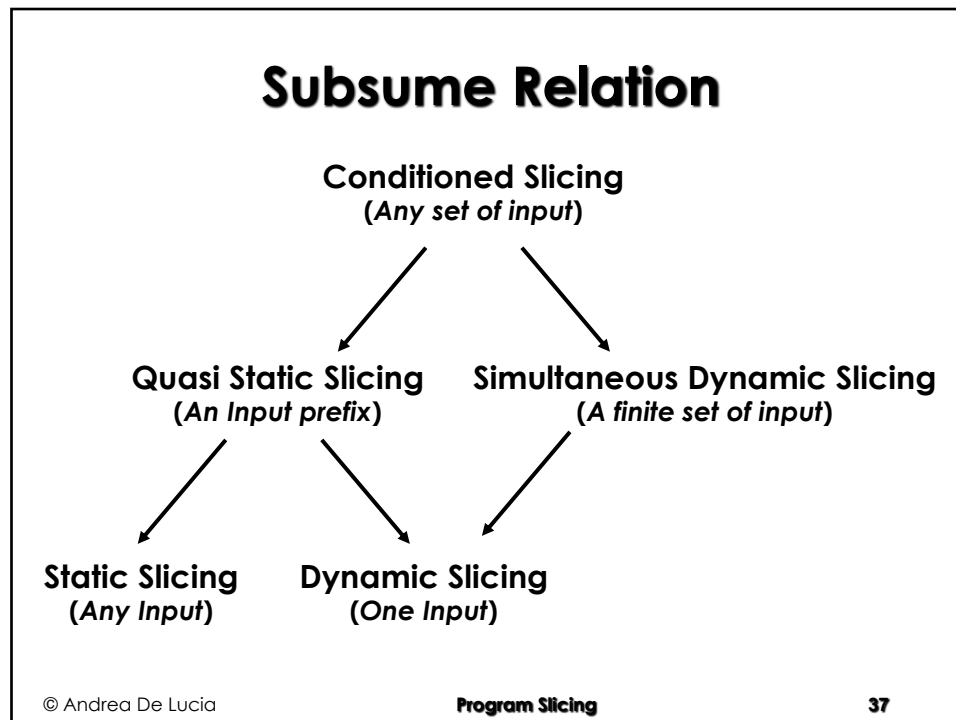
- Il conditioned slicing è una generalizzazione delle tecniche di slicing basate sulla eliminazione di statement
- L'insieme dei possibili input al programma è definito da una formula logica sulle variabili di ingresso
- Calcolo di una conditioned slice
 - ♦ Esecuzione simbolica: Produce un *Conditioned Program*
 - ♦ Slicing basato su PDG: Produce la *Conditioned Slice*
- Applicazioni a program comprehension, reengineering, testing

© Andrea De Lucia

Program Slicing

36

36



37

Altre forme di slicing

- ◆ **Slicing non procedural programs**
 - Proposti metodi di slicing per Object Oriented and Concurrent programs
 - ◆ Algoritmi basati su estensioni del PDG per gestire nuove features
 - Un metodo di slicing per applicazioni web (Ricca e Tonella, 2001)
- ◆ **Generalizzazioni: Amorphous program slicing**
 - Non solo statement deletion (Harman et al., 1997)
 - ◆ *Simplifying transformations which preserves the semantic projection*
 - ◆ Varianti di amorphous slicing per static slicing e conditioned slicing
 - ◆ Applicazioni a program comprehension e testing

© Andrea De Lucia Program Slicing 38

38

Conclusioni

- ◆ **Lo slicing dopo circa 40 anni è ancora prevalentemente un argomento di ricerca**
 - Molti articoli solo speculazioni teoriche ...
- ◆ **Pochi tool di slicing robusti, principalmente per slicing statico e dinamico**
 - Wisconsin project
 - Unravel project
 - In generale, slicing features non sono incluse in tool di analisi commerciali
- ◆ **Principali problemi**
 - Problemi di indecidibilità della data-flow analysis, dovuti ad esempio a puntatori e aliasing
 - Problemi specifici di alcuni linguaggi (ad esempio embedded side effects in C)

© Andrea De Lucia

Program Slicing

39

39

Conclusioni

- ◆ **Poche le applicazioni ed esperienze significative in ambiente industriale**
 - Scarsa maturità dei processi produttivi delle aziende software, con particolare riferimento a problemi di testing e maintenance
- ◆ **Quale futuro per la ricerca sullo slicing ?**
 - Nuove applicazioni coinvolgono tecnologie eterogenee
 - Necessità di tool per gestire tale complessità
 - Diversi livelli di astrazione
- ◆ **Quale futuro per applicazioni industriali ?**
 - Spesso problemi con formulazioni semplici
 - Problemi di ricerca complessi molto lontani dall'industria
 - Necessità di tool commerciali affidabili e usabili per convincere manager e maintenance programmer

© Andrea De Lucia

Program Slicing

40

40