Assignment 2

Algert Mucaj, Simone Garzarelli

April 2024

1 Very Busy Expressions

Very Busy Expressions	
Domain	Insiemi di espressioni
Direction	Backward IN[b] = $f_b(OUT[b])$ OUT[b] = ^ IN[successori(b)]
Transfer function	$f_b(x) = Gen_b \cup (x - Kills_b)$
Meet Operation	$\cap$
Boundary Condition	IN[exit] = $\emptyset$
Initial interior points	In[b] = μ

1.1 Scelte implementative

Un'espressione è **very busy** in un punto p se, indipendentemente dal percorso preso da p, l'espressione viene usata prima che uno dei suoi operandi venga definito.

Trattandosi di insiemi di espressioni, possiamo intuire che la direzione di scorrimento è come quella di quando trattiamo la liveness analysis di una variabile.

Utilizzando una direzione "Backward", l'analisi delle "Very Busy Expressions" determina l'insieme di espressioni che saranno "molto usate" prima di ogni assegnazione o modifica di un operando. Questo tipo di analisi aiuta a precalcolare l'uso delle espressioni all'indietro nel flusso di esecuzione del programma, partendo dai punti di uscita (exit) fino al punto di ingresso (entry).

Il bit vector, di conseguenza, sarà costituito dalle espressioni $b - a$ e $a - b$.

Come leggere la tabella delle iterazioni: Ovunque si trovi nella tabella la coppia (1,1), significa che in quella determinata configurazione, i bit corrispondenti alle espressioni $b - a$ e $a - b$, sono alti.

BB	Gen	Kill
1	∅	∅
2	∅	∅
3	b-a	∅
4	a-b	∅
5	b-a	∅
6	∅	b-a,a-b
7	a-b	∅
8	∅	∅

Table 2: Tabella Gen e Kill

1.2 Iterazioni dell'algoritmo

BB	Iterazione 1		iterazione 2		iterazione 3	
	in	out	in	out	in	out
BB1	1,1	1,1	1,0	1,0	1,0	1,0
BB2	1,1	1,1	1,0	1,0	1,0	1,0
BB3	1,1	1,1	1,1	0,1	1,1	0,1
BB4	1,1	∅	0,1	∅	0,1	∅
BB5	1,1	1,1	1,0	0,0	1,0	0,0
BB6	1,1	1,1	0,0	0,1	0,0	0,1
BB7	1,1	∅	0,1	∅	0,1	∅
BB8	∅	0,0	∅	∅	∅	∅

2 Dominator Analysis

In un CFG diciamo che un nodo X **domina** un altro nodo Y se il nodo X *appare in ogni percorso* del grafo che porta dal blocco ENTRY al blocco Y

- Annotiamo ogni basic block Bi con un insieme DOM[Bi]

  $B_i \in DOM[B_j]$ se e solo se Bi domina Bj

- Per definizione, un nodo *domina* **se stesso**

Dominator Analysis	
Domain	Insiemi di Basic Block
Direction	Forward OUT[b] = $f_b(IN[b])$ IN[b] = ^ OUT[predecessori(b)]
Transfer function	$f_b(x) = B \cup x$
Meet Operation	$\cap$
Boundary Condition	OUT[entry] = entry
Initial interior points	OUT[b$_i$] = μ

Non troviamo funzioni Gen e Kill poiché stiamo lavorando direttamente con dei basic block. L'intersezione è l'operazione che rispecchia meglio questa proprietà di inclusione. Quando calcoliamo l'OUT di un basic block b, consideriamo l'intersezione degli IN dei suoi predecessori. Questo significa che l'OUT[b] deve contenere solo quei blocchi che sono dominatori comuni di tutti i predecessori di b.

L'intersezione riduce l'insieme di possibili dominatori, mantenendo solo quei blocchi che sono dominatori comuni a tutti i percorsi di ingresso nel blocco.

BB	**Iterazione 1**		**iterazione 2**		**iterazione 3**	
	in	out	in	out	in	out
A	A	A	A	A	A	A
B	A	B	A	AB	A	AB
C	AC	C	A	AC	A	AC
D	AC	D	AC	ACD	AC	ACD
E	AC	E	AC	ACE	AC	ACE
F	$\emptyset$	F	AC	ACF	AC	ACF
G	$\emptyset$	G	A	AG	A	AG

3 Constant Propagation

L'obiettivo della constant propagation è quello di determinare in quali punti del programma le variabili hanno un valore costante.
L'informazione da calcolare per ogni nodo n del CFG è un insieme di coppie del tipo *(variabile, valore costante)*. Se abbiamo la coppia (x, c) al nodo n, significa che x è garantito avere il valore c ogni volta che n viene raggiunto durante l'esecuzione del programma.

Constant Propagation	
Domain	Insiemi di coppie *(variabile, costante)*
Direction	Forward OUT[b] = $f_b(IN[b])$ IN[b] = ^ OUT[predecessori(b)]
Transfer function	$f_b(x) = Gen_b \cup (x - Kills_b)$
Meet Operation	$\cap$
Boundary Condition	OUT[entry] = $\emptyset$
Initial interior points	OUT[b_i] = μ

BB	**Gen**	**Kill**
entry	$\emptyset$	$\emptyset$
1	(k,2)	$\emptyset$
2	$\emptyset$	$\emptyset$
3	(a,k+2)	$\emptyset$
4	(x,5)	$\emptyset$
5	(a,4)	(a,k+2)
6	(x,8)	(x,5)
7	(k,a)	(k,2)
8	$\emptyset$	$\emptyset$
9	(b,2)	$\emptyset$
10	(x,a+k)	(x,8)(x,5)
11	(y,a*b)	$\emptyset$
12	(k, k+1)	(k,a)
13	$\emptyset$	$\emptyset$
exit	$\emptyset$	$\emptyset$

Table 7: Tabella Gen e Kill

Nella propagation delle costanti, l'obiettivo principale è determinare quali valori costanti possono essere conosciuti prima di eseguire ogni blocco di base del programma.L'analisi forward si allinea naturalmente con il flusso di esecuzione del programma, dove le variabili e i valori vengono definiti prima di essere utilizzati.

Nell'analisi forward, i risultati per ciascun blocco di base dipendono dagli ingressi provenienti dai blocchi predecessori.

3.1 Iterazioni dell'algoritmo

BB	**Iterazione 1**		**iterazione 2**		**iterazione 3**		**iterazione 4**	
	in	out	in	out	in	out	in	out
Entry	$\emptyset$	μ	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
1	μ	μ	$\emptyset$	(k,2)	$\emptyset$	(k,2)	$\emptyset$	(k,2)
2	μ	μ	(k,2)	(k,2)	(k,2)	(k,2)	(k,2)	(k,2)
3	μ	μ	(k,2)	(k,2) (a,4)	(k,2)	(k,2) (a,4)	(k,2)	(k,2) (a,4)
4	μ	μ	(k,2) (a,4)	(k,2) (a,4) (x,5)	(k,2) (a,4)	(k,2) (a,4) (x,5)	(k,2) (a,4)	(k,2) (a,4) (x,5)
5	μ	μ	(k,2)	(k,2) (a,4)	(k,2)	(k,2) (a,4)	(k,2)	(k,2) (a,4)
6	μ	μ	(k,2) (a,4)	(k,2) (a,4) (x,8)	(k,2) (a,4)	(k,2) (a,4) (x,8)	(k,2) (a,4)	(k,2) (a,4) (x,8)
7	μ	μ	(k,2) (a,4)	(k,2) (a,4)	(k,2) (a,4)	(k,2) (a,4)	(k,2) (a,4)	(k,2) (a,4)
8	μ	μ	(k,2) (a,4)	(k,2) (a,4)	(a,4)	(a,4)	(a,4)	(a,4)
9	μ	μ	(k,2) (a,4)	(b,2) (k,2) (a,4)	(a,4)	(a,4) (b,2)	(a,4)	(a,4) (b,2)
10	μ	μ	(b,2) (k,4) (a,4)	(b,2) (k,4) (a,4) (x,8)	(a,4) (b,2)	(a,4) (b,2)	(a,4) (b,2)	(a,4) (b,2)
11	μ	μ	(b,2) (k,4) (a,4) (x,8)	(b,2) (k,4) (a,4) (x,8) (y,8)	(a,4) (b,2)	(a,4) (b,2) (y,8)	(a,4) (b,2)	(a,4) (b,2) (y,8)
12	μ	μ	(b,2) (k,4) (a,4) (x,8) (y,8)	(b,2) (a,4) (x,8) (y,8) (k,5)	(a,4) (b,2) (y,8)	(a,4) (b,2) (y,8)	(a,4) (b,2) (y,8)	(a,4) (b,2) (y,8)
13	μ	μ	(k,4) (a,4)	(k,4) (a,4)	(a,4)	(a,4)	(a,4)	(a,4)
Exit	μ	μ	(k,4) (a,4)	(k,4) (a,4)	(a,4)	(a,4)	(a,4)	(a,4)