DATAFLOW ANALYSIS ASSIGNMENT

-

Giuseppe Catillo [165722]
Luca Lodesani [165507]
Riccardo Neri [165343]

1. Very Busy Expressions

1.2 Formalizzazione

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.

Un'espressione *a+b* è ***very busy*** in un punto *p* se *a+b* è valutata in tutti i percorsi da *p* a EXIT e non c'è una definizione di a o b lungo tali percorsi:
- Ci interessa l'insieme di espressioni disponibili (available) all'inizio del blocco B
- L'insieme dipende dai percorsi che cominciano al punto p prima di B

Very Busy Expressions	
Domain	Set of expressions
Direction[1]	Backward: in[b] = f_b(out[b]) out[b] = $\wedge$ in[succ(b)]
Transfer function[2]	$f_b(x) = Gen_b \cup (x - Kill_b)$
Meet operation ($\wedge$)	$\cap$
Boundary Condition[3]	in[EXIT] = Ø
Initial interior points[4]	in[b] = *u*

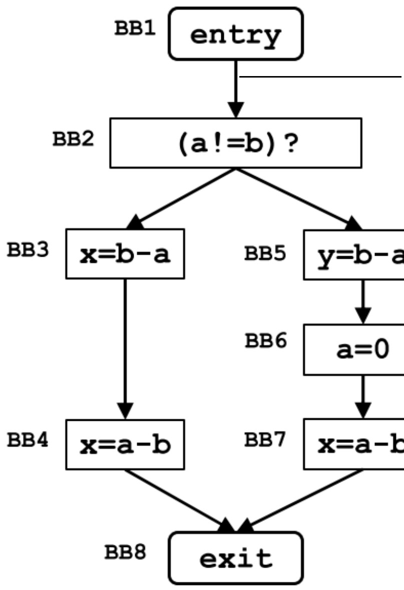
[1]
Considerando il percorso che porta "da *p* ad EXIT", l'analisi prosegue calcolando l'input di ogni basic block sulla base dell'output dei suoi predecessori.
Inoltre, nel momento in cui valuto un'espressione, verifico che gli operandi non siano stati definiti in precedenza.

[2]
Gen_b: espressioni binarie all'interno di *b* (i cui operandi non sono stati definiti nel basic block stesso)
$Kill_b$: espressioni che utilizzano un operando definito in *b*.
Seguendo un'analisi conservativa, consideriamo l'intero CFG, senza un ordine prestabilito.

[3]
Inizialmente, l'insieme delle espressioni very busy è vuoto.

[4]
Siccome il meet operator è ∩, un'analisi conservativa ci obbliga ad inizializzare l'input tramite universal set.
Se utilizzassimo l'insieme vuoto, l'intersezione con le informazioni precedentemente calcolate causerebbe la perdita di queste ultime.

1.2 Esempio



in questo punto, l'espressione *b-a* è very busy perché, qualunque percorso io prenda, i suoi operandi non vengono definiti prima che la valuti

	Iterazione 1		Iterazione 2	
	IN[B]	**OUT[B]**	**IN[B]**	**OUT[B]**
BB2	(b-a)	(b-a)	(b-a)	(b-a)
BB3	(a-b), (b-a)	(a-b)	(a-b), (b-a)	(a-b)
BB4	(a-b)	Ø	(a-b)	Ø
BB5	(b-a)	Ø	(b-a)	Ø
BB6	Ø	(a-b)	Ø	(a-b)
BB7	(a-b)	Ø	(a-b)	Ø

Svolgimento:

	Gen	**Kill**
BB2	/	/
BB3	(b-a)	/
BB4	(a-b)	/
BB5	(b-a)	/
BB6	/	{(a-b), (b-a)}
BB7	(a-b)	/

Iterazione 1)	**Iterazione 2)**
out[b7] = in[EXIT] = Ø oi[b7] = *u* in[b7] = (a-b) U (Ø - Ø) = (a-b)	out[b7] = in[EXIT] = Ø oi[b7] = (a-b) in[b7] = (a-b) U (Ø - Ø) = (a-b) **NC**
out[b6] = in[b7] = (a-b) oi[b6] = *u* in[b6] = Ø U [(a-b) - {(a-b), (b-a)}] = Ø	out[b6] = in[b7] = (a-b) oi[b6] = Ø in[b6] =Ø U [(a-b) - {(a-b), (b-a)}] = Ø **NC**
out[b5] = in[b6] = Ø oi[b5] = *u* in[b5] = (b-a) U (Ø - Ø) = (b-a)	out[b5] = in[b6] = Ø oi[b5] = (b-a) in[b5] = (b-a) U (Ø - Ø) = (b-a) **NC**
out[b4] = in[EXIT] = Ø oi[b4] = *u* in[b4] = (a-b) U (Ø - Ø) = (a-b)	out[b4] = in[EXIT] = Ø oi[b4] = (a-b) in[b4] = (a-b) U (Ø - Ø) = (a-b) **NC**
out[b3] = in[b4] = (a-b) oi[b3] = *u* in[b3] = (b-a) U {(a-b) - Ø} = (a-b), (b-a)	out[b3] = in[b4] = (a-b) oi[b3] = (a-b), (b-a) in[b3] = (b-a) U {(a-b) - Ø} = (a-b), (b-a) **NC**
out[b2] = in[b3] ∩ in[b5] = (b-a) oi[b2] = *u* in[b2] = Ø U (b-a) = (b-a)	out[b2] = in[b3] ∩ in[b5] = (b-a) oi[b2] = (b-a) in[b2] = Ø U (b-a) = (b-a) **NC**

2. Dominator Analysis

2.1 Formalizzazione

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]:

$$\text{Bi} \in \text{DOM[Bj] se e solo se Bi domina Bj}$$

Per definizione un nodo domina sé stesso:

$$\text{Bi} \in \text{DOM[Bi]}$$

Dominator Analysis	
Domain	Set of Basic Blocks
Direction[1]	Forward: $out[b] = f_b(in[b])$ $in[b] = \wedge\ out[pred(b)]$
Transfer function[2]	$f_b(x) = Gen_b \cup x$
Meet operation ($\wedge$)	$\cap$
Boundary Condition[3]	out[ENTRY] = ENTRY
Initial interior points[4]	out[b] = *u*

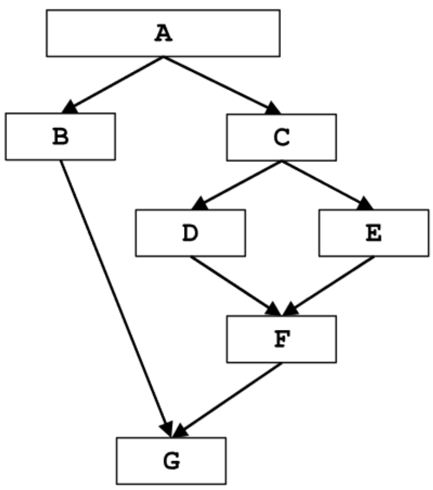
[1]
Considerando il percorso che porta "dal blocco ENTRY al blocco *Y*", l'analisi prosegue calcolando l'output di ogni basic block sulla base dell'input dei suoi predecessori.

[2]
Gen_b: basic block considerato (*b*)

[3]
Inizialmente, l'ENTRY del CFG (A) domina sé stesso.

[4]
Siccome il meet operator è $\cap$, un'analisi conservativa ci obbliga ad inizializzare l'output tramite universal set.
Se utilizzassimo l'insieme vuoto, l'intersezione con le informazioni precedentemente calcolate causerebbe la perdita di queste ultime.

2.2 Esempio



DOM[F] = {A,C,F}

	Iterazione 1		Iterazione 2	
	IN[B]	**OUT[B]**	**IN[B]**	**OUT[B]**
A	/	A	/	A
B	A	{A, B}	A	{A, B}
C	A	{A, C}	A	{A, C}
D	{A, C}	{A, C, D}	{A, C}	{A, C, D}
E	{A, C}	{A, C, E}	{A, C}	{A, C, E}
F	{A, C}	{A, C, F}	{A, C}	{A, C, F}
G	A	{A, G}	A	{A, G}

Svolgimento:

NB: Si considera A come l'ENTRY del CFG.

Iterazione 1)

oo[A] = *u*
out[A] = out[ENTRY] = A

in[B] = out[A] = A
oo[B] = *u*
out[B] = B U A = {A, B}

in[C] = out[A] = A
oo[C] = *u*
out[C] = C U A = {A, C}

in[D] = out[C] = {A, C}
oo[D] = *u*
out[D] = D U {A, C} = {A, C, D}

in[E] = out[C] = {A, C}
oo[E] = *u*
out[E] = E U {A, C} = {A, C, E}

in[F] = out[D] ∩ out[E] = {A, C}
oo[F] = *u*
out[F] = F U {A, C} = {A, C, F}

in[G] = out[B] ∩ out[F] = A
oo[G] = *u*
out[G] = G U A = {A, G}

Iterazione 2)

oo[A] = A
out[A] = out[ENTRY] = A **NC**

in[B] = out[A] = A
oo[B] = {A, B}
out[B] = B U A = {A, B} **NC**

in[C] = out[A] = A
oo[C] = {A, C}
out[C] = C U A = {A, C} **NC**

in[D] = out[C] = {A, C}
oo[D] = {A, C, D}
out[D] = D U {A, C} = {A, C, D} **NC**

in[E] = out[C] = {A, C}
oo[E] = {A, C, E}
out[E] = E U {A, C} = {A, C, E} **NC**

in[F] = out[D] ∩ out[E] = {A, C}
oo[F] = {A, C, F}
out[F] = F U {A, C} = {A, C, F} **NC**

in[G] = out[B] ∩ out[F] = A
oo[G] = {A, G}
out[G] = G U A = {A, G} **NC**

3. Constant Propagation

3.1 Formalizzazione

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	Set of couples <*variable, constant value*>
Direction[1]	Forward: $out[b] = f_b(in[b])$ $in[b] = \wedge\ out[pred(b)]$
Transfer function[2]	$f_b(x) = Gen_b \cup (x - Kill_b)$
Meet operation ($\wedge$)	$\cap$
Boundary Condition[3]	out[ENTRY] = Ø
Initial interior points[4]	out[b] = *u*

[1]
Considerando il normale flusso di esecuzione di un programma (dall'ENTRY), al fine di garantire la consistenza delle informazioni calcolate è necessario calcolare l'output di ogni basic block sulla base dell'input dei suoi predecessori.

[2]
Gen_b = coppia <*variabile, valore costante*>
$Kill_b$: coppie che utilizzano un operando definito in *b*.
Seguendo un'analisi conservativa, consideriamo l'intero CFG, senza un ordine prestabilito.

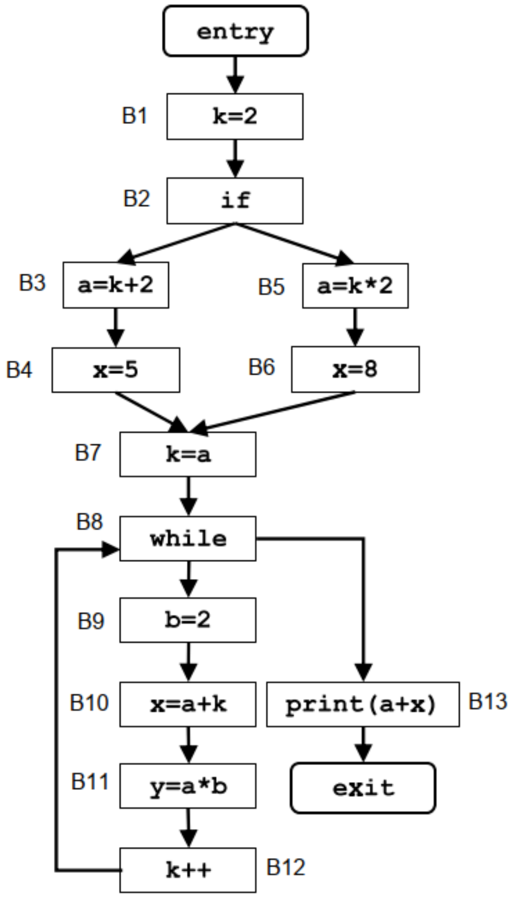
[3]
Inizialmente, l'insieme delle coppie <*variabile, valore costante*> è vuoto.

[4]
Siccome il meet operator è $\cap$, un'analisi conservativa ci obbliga ad inizializzare l'output tramite universal set.
Se utilizzassimo l'insieme vuoto, l'intersezione con le informazioni precedentemente calcolate causerebbe la perdita di queste ultime.

NB: Lo studio non tiene conto dell'ottimizzazione Constant Folding, perciò per *valore costante* si intende un letterale:

$$k = 2 \rightarrow \langle k, 2\rangle \ ✓$$
$$a = k + 2 = 4 \rightarrow \langle a, 4\rangle \ \text{X}$$

3.2 Esempio



	Iterazione 1		Iterazione 2	
	IN[B]	OUT[B]	IN[B]	OUT[B]
B1	Ø	<k, 2>	Ø	<k, 2>
B2	<k, 2>	<k, 2>	<k, 2>	<k, 2>
B3	<k, 2>	<k, 2>	<k, 2>	<k, 2>
B4	<k, 2>	{<k, 2>, <x, 5>}	<k, 2>	{<k, 2>, <x, 5>}
B5	<k, 2>	<k, 2>	<k, 2>	<k, 2>
B6	<k, 2>	{<k, 2>, <x, 8>}	<k, 2>	{<k, 2>, <x, 8>}
B7	<k, 2>	Ø	<k, 2>	Ø
B8	Ø	Ø	Ø	Ø
B9	Ø	<b, 2>	Ø	<b, 2>
B10	<b, 2>	<b, 2>	<b, 2>	<b, 2>
B11	<b, 2>	<b, 2>	<b, 2>	<b, 2>
B12	<b, 2>	<b, 2>	<b, 2>	<b, 2>
B13	Ø	Ø	Ø	Ø

Svolgimento:

	Iterazione 1	
	Gen	**Kill**
B1	<k, 2>	/
B2	/	/
B3	/	/
B4	<x, 5>	<x,8>
B5	/	/
B6	<x, 8>	<x,5>
B7	/	<k, 2>
B8	/	/
B9	<b, 2>	/
B10	/	<x,5>, <x,8>
B11	/	/
B12	/	<k, 2>
B13	/	/

Iterazione 1)	**Iterazione 2)**
in[B1] = out[ENTRY] = Ø oo[B1] = *u* out[B1] = <k, 2> U (Ø - Ø) = <k, 2>	in[B1] = out[ENTRY] = Ø oo[B1] = <k, 2> out[B1] = <k, 2> U (Ø - Ø) = <k, 2> **NC**
in[B2] = out[B1] = <k, 2> oo[B2] = *u* out[B2] = Ø U (<k, 2> - Ø) = <k, 2>	in[B2] = out[B1] = <k, 2> oo[B2] = *u* out[B2] = Ø U (<k, 2> - Ø) = <k, 2> **NC**
in[B3] = out[B2] = <k, 2> oo[B3] = *u* out[B3] = Ø U (<k, 2> - Ø) = <k, 2>	in[B3] = out[B2] = <k, 2> oo[B3] = <k, 2> out[B3] = Ø U (<k, 2> - Ø) = <k, 2> **NC**
in[B4] = out[B3] = <k, 2> oo[B4] = *u* out[B4] = <x, 5> U (<k, 2> - <x, 8>) = {<x, 5>, <k,2>}	in[B4] = out[B3] = <k, 2> oo[B4] = {<x, 5>, <k,2>} out[B4] = <x, 5> U (<k, 2> - <x, 8>) = {<x, 5>, <k,2>} **NC**
in[B5] = out[B2] = <k, 2> oo[B5] = *u* out[B5] = Ø U (<k, 2> - Ø) = <k, 2>	in[B5] = out[B2] = <k, 2> oo[B5] = <k, 2> out[B5] = Ø U (<k, 2> - Ø) = <k, 2> **NC**
in[B6] = out[B5] = <k, 2> oo[B6] = *u* out[B6] = <x, 8> U (<k, 2> - <x, 5>) = {<x, 8>, <k,2>}	in[B6] = out[B5] = <k, 2> oo[B6] = {<x, 8>, <k,2>} out[B6] = <x, 8> U (<k, 2> - <x, 5>) = {<x, 8>, <k,2>} **NC**
…	…

in[B7] = out[B4] ∩ out[B6] = <k, 2> oo[B7] = *u* out[B7] = Ø U (<k, 2> - <k, 2>) = Ø	in[B7] = out[B4] ∩ out[B6] = <k, 2> oo[B7] = Ø out[B7] = Ø U (<k, 2> - <k, 2>) = Ø **NC**
in[B8] = out[B7] ∩ out[B12] = Ø oo[B8] = *u* out[B8] = Ø U (Ø - Ø) = Ø	in[B8] = out[B7] ∩ out[B12] = Ø oo[B8] = Ø out[B8] = Ø U (Ø - Ø) = Ø **NC**
in[B9] = out[B8] = Ø oo[B9] = *u* out[B9] = <b, 2> U (Ø - Ø) = <b, 2>	in[B9] = out[B8] = Ø oo[B9] = <b, 2> out[B9] = <b, 2> U (Ø - Ø) = <b, 2> **NC**
in[B10] = out[B9] = <b, 2> oo[B10] = *u* out[B10] = Ø U [<b, 2> - (<x, 5>, <x, 8>)] = <b, 2>	in[B10] = out[B9] = <b, 2> oo[B10] = <b, 2> out[B10] = Ø U [<b, 2> - (<x, 5>, <x, 8>)] = <b, 2> **NC**
in[B11] = out[B10] = <b, 2> oo[B11] = *u* out[B11] = Ø U (<b, 2> - Ø) = <b, 2>	in[B11] = out[B10] = <b, 2> oo[B11] = <b, 2> out[B11] = Ø U (<b, 2> - Ø) = <b, 2> **NC**
in[B12] = out[B11] = <b, 2> oo[B12] = *u* out[B12] = Ø U (<b, 2> - <k, 2>) = <b, 2>	in[B12] = out[B11] = <b, 2> oo[B12] = <b, 2> out[B12] = Ø U (<b, 2> - <k, 2>) = <b, 2> **NC**
in[B13] = out[B8] = Ø oo[B13] = *u* out[B13] = Ø U (Ø - Ø) = Ø	in[B13] = out[B8] = Ø oo[B13] = Ø out[B13] = Ø U (Ø - Ø) = Ø **NC**