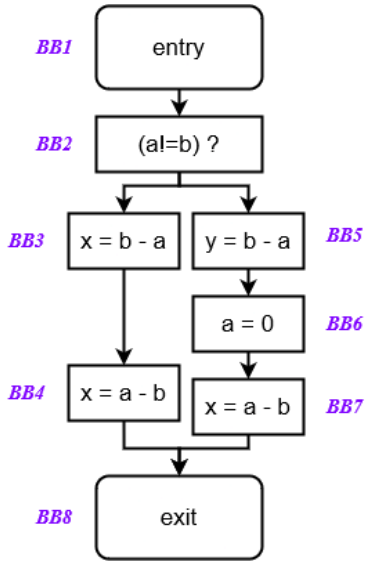
Secondo assignment

1. Very Busy Expressions

- 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.



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

> Utilizziamo un approccio **backward**, proprio come quanto visto nella **liveness analysis**, perché se un espressione è very busy *dopo* un istruzione e non viene aggiornata da quella stessa istruzione, allora l'espressione sarà very busy anche **prima** dell'istruzione.
>
> Da questo capiamo che l'informazione si propaga **all'indietro**.
>
> Partiamo quindi dai punti di uscita (exit) fino ai punti di ingresso (entry).
>
> > Il bit vector utilizzato include le espressioni b-a e a-b.
> >
> > Per comprendere la tabella delle iterazioni è importante notare che la il primo bit è associato all'espressione `b-a` mentre il secondo all'espressione `a-b`. Ad esempio, la coppia (1,1) indica che in quella configurazione specifica i bit corrispondenti alle espressioni `b-a` ed `a-b` sono attivi.

Tabella Gen e Kill

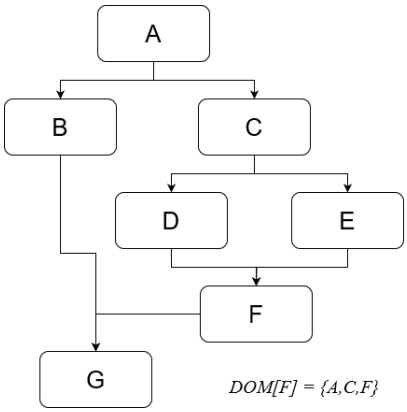
BB	Gen	Kill
1	∅	∅
2	∅	∅
3	*b-a*	∅
4	*a-b*	∅
5	*b-a*	∅
6	∅	*b-a*, *a-b*
7	*a-b*	∅
8	∅	∅

Iterazioni

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].
- Bi appartiene a DOM[Bj] se e solo se Bi domina Bj.
- Per definizione, un nodo domina se stesso.
- Bi appartiene a DOM[Bi].



Domain	Insiemi di espressioni
Direction	Forward
	$OUT[b] = f_b\ (IN[b])$
	$IN[b] = \hat{}OUT[predecessori(b)]$
Transfer function	$f_b(x) = B \cup x$
Meet Operation	$\cap$
Boundary Condition	$IN[exit] = entry$
Initial interior points	$In[b] = \mu$

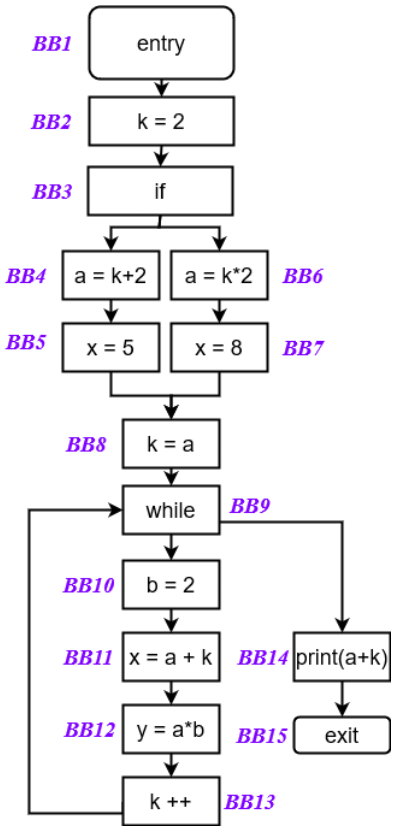
> Scegliamo come meet operator l'intersezione in questo caso perché ci rappresenta l'and logico delle informazioni provenienti da diversi percorsi. Per calcolare l'OUT di un Basic Block consideriamo l'intersezione degli IN dei blocchi precedenti. Per includere i dominatori comuni di tutti i predecessori di B usiamo quindi l'intersezione.
>
> La transfer function comprende B poichè ogni blocco è anche dominatore di se stesso.
>
> In questo esercizio è stata omessa la tabella delle Gen e delle Kill in quanto stiamo lavorando direttamente con dei basic block.

Iterazioni

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	∅	F	AC	ACF	AC	ACF
G	∅	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 .
- Se abbiamo la coppia 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	∩
Boundary Condition	OUT[entry] = ∅
Initial interior points	OUT[b_i] = μ

> Nella constant propagation ci interessa determinare i valori costanti che possono essere assegnati alle variabili all'interno del programma. Questa operazione la facciamo mentre attraversiamo il flusso dall'ingresso (entry) all'uscita (exit).
>
> Per questo motivo parliamo di direzione **forward**.

Iterazioni

BB	Iterazione 1		Iterazione 2		Iterazione 3		Iterazione 4	
	in	out	in	out	in	out	in	out
Entry	∅	μ	∅	∅	∅	∅	∅	∅
1	μ	μ	∅	(k,2)	∅	(k,2)	∅	(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)