Compilatori - Assignment 2: Dataflow Analysis

Manuel Vincenzo Lonetti, Davide Camassi

15 aprile 2025

Indice

1 Introduzione al secondo Assigment

In questo secondo assignment del corso di Compilatori, affronteremo l'analisi del Dataflow di tre problemi noti:

- ***Very Busy Expressions***
- ***Dominator Analysis***
- ***Constant Propagation***

Per ognuno di questi problemi, costruiremo inizialmente il proprio ***Framework***, così da essere in grado in secondo luogo di risolvere gli esercizi forniti con l'assignment.

In questi esercizi, quello che andremo a fare sarà appunto sfruttare i framework dei singoli problemi precedentemente definiti, per eseguire correttamente un'analisi del control flow graph degli esempi forniti.

I risultati delle analisi che otterremo, saranno la base in futuro per andare a creare passi di ottimizzazione concreti su rappresentazioni intermedie LLVM.

2 Dataflow Analysis

Osserviamo nel dettaglio le analisi appena introdotte:

2.1 Very Busy Expressions

Un'espressione è ***very busy*** se, a partire da un punto P del nostro CFG, l'espressione viene valutata (utilizzata) in tutti i possibili percorsi futuri a partire dal punto P fino ad exit, senza che nessun operando venga ridefinito.

È una definzione molto vicina all'analisi di ***Avaiable Expression*** affrontata a lezione. La differenza però in una Dataflow analysis per espressioni very busy è che ci interessa appunto capire gli usi futuri dell'espressione, per eventualmente pre-calcolarne il valore.

Questa analisi di fatti è alla base della code hoisting: un tipo particolare di ottimizzazione in cui, se un'espressione si trova all'interno di un ciclo, e a partire da esso viene utilizzata in tutti i suoi possibili percorsi futuri, senza che gli operandi siano ridefiniti, possiamo calcolare il valore dell'espressione una sola volta fuori dal ciclo ed utilizzarlo quando ci serve.

2.1.1 Framework Very Busy Expressions

	Very Busy Expressions
Dominio	Espressioni
Direzione	Backward
Funzione di trasferimento	IN[B] = gen[B] $\cup$ (OUT[B] - kill[B])
Meet Operation ($\wedge$)	$\cap$
Boundary Condition	IN[exit] = $\emptyset$
Condizioni iniziali	IN[B] = υ

Tabella 1: Very Busy Expressions Dataflow Framework

Commentiamo le scelte applicate per la costruzione del framework:

1. Il dominio è **l'insieme delle espressioni**, infatti stiamo andando a studiarne il loro utilizzo futuro.

2. L'analisi è **backward**: partendo da un punto P, diciamo che un'espressione è very busy se utilizzata in tutti i percorsi che partono da P.

   Dovendo osservare il "futuro", faremo un'analisi di tipo backward, risalendo fino al punto P.

3. La **funzione di trasferimento** è:

$$\text{OUT}[B] = \bigcap_{s \in \text{succ}(B)} IN[s]$$

$$\text{IN}[B] = \text{gen}[B] \cup (\text{OUT}[B] - \text{kill}[B])$$

4. **Meet Operation**: si usa l'intersezione perché un'espressione è very busy solo se è usata in **tutti i cammini** futuri.

5. **IN[exit] = $\emptyset$** indica che quando iniziamo l'analisi, non abbiamo ancora alcuna espressione very busy

6. **Condizioni iniziali**: non si utilizza l'insieme vuoto, ma l'insieme υ, così che qualora si facesse un'intersezione con un BB per cui ancora non abbiamo informazioni, utilizzando questa condizione di partenza eviteremo risultati annullati erroneamente.

2.1.2 Esempio di Dataflow analysis per very busy expressions

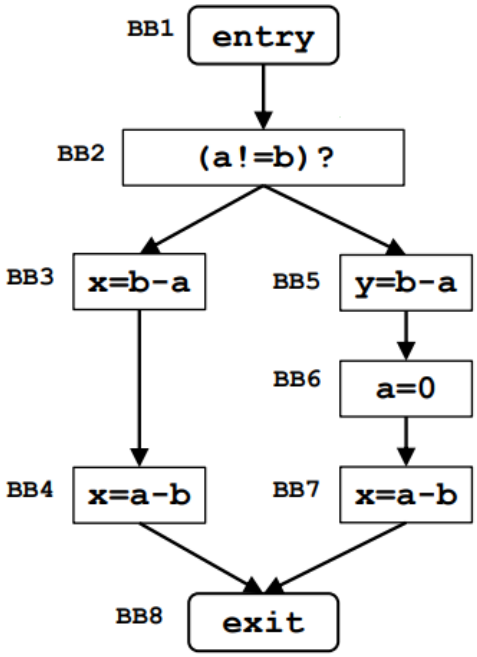


Figura 1: Very Busy Expressions Dataflow analysis

Facciamo prima una tabella per riassumere il comportamento di ogni Basic Block nel control flow graph:

Basic Block	**Gen[B]**	**Kill[B]**
B8 (Exit)	$\emptyset$	$\emptyset$
B7	$\{a-b\}$	$\emptyset$
B6	$\emptyset$	$\{a-b, b-a\}$
B5	$\{b-a\}$	$\emptyset$
B4	$\{a-b\}$	$\emptyset$
B3	$\{b-a\}$	$\emptyset$
B2	$\{a!=b\}$	$\emptyset$
B1 (Entry)	$\emptyset$	$\emptyset$

Possiamo ora iniziare a svolgere l'esercizio, applicando le funzioni di trasferimento e le informazioni presenti all'interno del framework della very busy expressions, mediante i dati ricavati nella precedente tabella.

Il risultato finale è:

Basic Block	**IN[B]**	**OUT[B]**
B8 (Exit)	$\emptyset$	–
B7	$\{a-b\}$	$\emptyset$
B6	$\emptyset$	$\{a-b\}$
B5	$\{b-a\}$	$\emptyset$
B4	$\{a-b\}$	$\emptyset$
B3	$\{b-a, a-b\}$	$\{a-b\}$
B2	$\{a!=b, b-a\}$	$\{b-a\}$
B1 (Entry)	–	$\{a!=b, b-a\}$

Tabella 2: Esercizio Very Busy Expressions

2.2 Dominator Analysis

In un control flow graph, un nodo $\boldsymbol{x}$ domina un nodo $\boldsymbol{y}$ se, a partire dal blocco entry, tutti i percorsi per raggiungere il nodo $\boldsymbol{y}$ passano attraverso il nodo $\boldsymbol{x}$.

Questo tipo di Dataflow analysis è alla base dell'individuazione dei loop all'interno di un control flow graph.

2.2.1 Framework Dominator Analysis

	Dominator Analysis
Dominio	Insieme dei basic blocks nel CFG
Direzione	Forward
Funzione di trasferimento	$\text{OUT}[B] = \{B\} \cup \bigcap_{p \in \text{pred}(B)} \text{OUT}[p]$
Meet Operation ($\wedge$)	$\cap$
Boundary Condition	$\text{OUT}[entry] = \{entry\}$
Condizioni iniziali	$\text{OUT}[B] = \{B\}$

Tabella 3: Dominator Analysis Dataflow Framework

Commentiamo le scelte applicate per la costruzione del framework:

1. Il **dominio** è l'insieme dei basic block: per ogni blocco individuiamo l'insieme dei blocchi che lo dominano.

2. L'analisi è **forward**. Per capire un blocco da chi è dominato, bisogna guardare i suoi predecessori.

   L'analisi assume quindi una forma Forward, dove si propagano informazioni verso il basso, ed ogni blocco va a guardare i propri predecessori.

3. La **funzione di trasferimento** è data da

   $$\text{OUT}[B] = \text{IN}[B] \cup \text{B}$$

   Dove OUT[B] è l'insieme dei basic block che domina B.

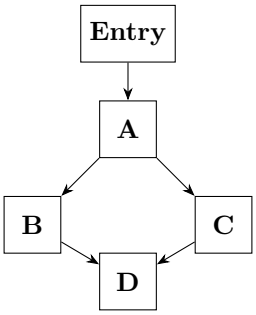
   $$\text{IN}[B] = \bigcap_{p \in \text{pred}(B)} \text{OUT}[p]$$

   Per calcolare il dato IN[B] dobbiamo fare l'intersezione di OUT dei predecessori di B.

   Dobbiamo dunque mettere in intersezione gli insiemi che dominano i predecessori di B.

   Il risultato saranno i Basic block che domineranno anche il blocco B (oltre che il blocco B stesso, ma non stiamo calcolando OUT[B] (DOM[B])).

4. La **meet operation** è l'**intersezione**: per spiegare il perché, osserviamo un semplice esempio, collegandoci anche a ciò che abbiamo spiegato in precedenza:



Supponiamo:

- OUT[B] = {Entry, A, B}
- OUT[C] = {Entry, A, C}

Allora:

$$\text{IN[D]} = \text{OUT[B]} \cap \text{OUT[C]} = \{Entry, A\}$$

D è dominato da Entry e da A, perché sono gli unici blocchi presenti in tutti i cammini verso D.

Quando un blocco ha più predecessori, allora usiamo l'intersezione sull'insieme dei dominatori di ciascun predecessore per determinare chi domina il blocco corrente.

5. La **boundary condition** indica che il blocco *entry* dalla quale iniziamo l'analisi, è dominato da sé stesso.

6. Le **condizioni iniziali** impongono che ogni blocco, all'inizio dell'analisi, è dominato da sé stesso.

2.2.2 Esempio di Dataflow analysis per Dominator Analysis

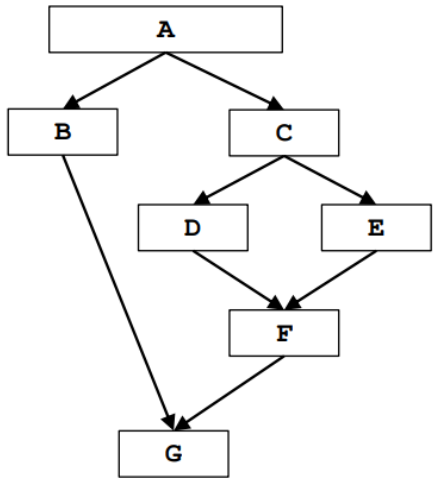


Figura 2: Dominator Analysis

Basic Block	**IN[B]**	**OUT[B]**
A	–	$\{A\}$
B	$\{A\}$	$\{A, B\}$
C	$\{A\}$	$\{A, C\}$
D	$\{A, C\}$	$\{A, C, D\}$
E	$\{A, C\}$	$\{A, C, E\}$
F	$\{A, C\}$	$\{A, C, F\}$
G	$\{A\}$	$\{A, G\}$

Tabella 4: Esercizio - Dominator Analysis

2.3 Constant Propagation

La Dataflow analysis per la **Constant Propagation** determina per ogni basic block del programma che analizziamo, quali variabili sono **costanti**: ossia assumono un valore costante identico lungo tutti i cammini che portano a quel basic block.

Definiamo quindi costante una variabile in un basic block B se, per ogni percorso (Basic Block) che arriva a B, la variabile ha lo stesso valore costante in ognuno di essi. Se nei percorsi diversi che arrivano al punto B, la variabile assume valori diversi, allora non sarà costante in quel punto B.

Questo tipo di analisi è alla base di molte ottimizzazioni: individuate le variabili costanti, l'obiettivo sarà poi di rimpiazzarne direttamente il valore.

2.3.1 Framework Constant Propagation

	Constant Propagation Framework
Dominio	Coppie del tipo (variabile, valore costante)
Direzione	Forward
Funzione di trasferimento	`OUT[B] = GEN[B]` $\cup$ `(IN[B]` $\setminus$ `KILL[B])`
Meet Operation ($\wedge$)	$\bigcap$: Intersezione delle coppie costanti dai predecessori
Boundary Condition	`OUT[Entry]` $= \emptyset$
Condizioni iniziali	`OUT[B]` $= v$

Tabella 5: Constant Propagation Dataflow Framework

Commentiamo le scelte applicate per la costruzione del framework:

1. **Dominio:** L'informazione che analizziamo riguarda coppie del tipo `(variabile, valore costante)`.

   Una variabile è considerata costante in un punto del programma se ha lo stesso valore in tutti i percorsi che portano a quel punto, senza essere ridefinita.

2. **Direzione:** L'analisi è *forward* perché per ogni blocco, determiniamo l'insieme delle variabili costanti (con relativo valore v) sulla base delle informazioni all'interno del basic block stesso, ma anche delle informazioni propagate dai basic block predecessori

3. **Funzione di trasferimento:**

$$\texttt{OUT[B]} = \texttt{GEN[B]} \cup (\texttt{IN[B]} \setminus \texttt{KILL[B]})$$

dove:

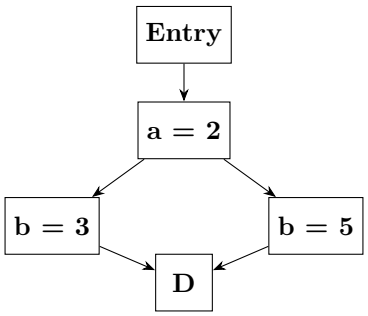
- `GEN[B]` è l'insieme delle assegnazioni costanti generate nel blocco B.
- `IN[B]` è l'insieme delle variabili costanti propagate dai blocchi predecessori.
- `KILL[B]` contiene le variabili ridefinite in B (quindi non più costanti).

  Se una o più variabili costanti propagate dai blocchi predecessori, vengono ridefinite nel blocco B attuale, queste automaticamente non saranno più costanti in questo punto del control flow graph.

Vediamo dunque come le informazioni vengono propagate dai basic block predecessori:

$$\text{IN}[B] = \bigcap_{p \in \text{pred}(B)} \text{OUT}[p]$$

Facciamo un esempio con la seguente figura:



Quello che succede nel Blocco D è che viene fatta l'intersezione tra i due blocchi predecessori.

Ciò che che otterremo sarà che: avendo valori costanti diversi della variabile ***b***, quest'ultima non sarà considerata costante, e non si troverà nell'insieme di variabili costanti del blocco D.

la variabile "***a***" invece è costante, avremo quindi nel blocco D la coppia {(a, 2)}.

4. **Meet Operator:** Si usa l'intersezione tra gli `OUT` dei predecessori. Questo perché una variabile può essere considerata costante solo se ha lo stesso valore in tutti i percorsi che arrivano a un blocco (come la variabile "***a***" nel blocco D nel precedente esempio).

5. **Boundary Condition:** All'inizio della analisi, nessuna variabile è ancora nota come costante, quindi `OUT[Entry]` $= \emptyset$.

6. **Condizioni iniziali:** Assumiamo la condizione iniziale `OUT[B]` $= \upsilon$ per fare in modo che l'intersezione con blocchi del CFG non ancora esplorati vada a buon fine.

2.3.2 Esempio di Dataflow analysis per Constant Propagation

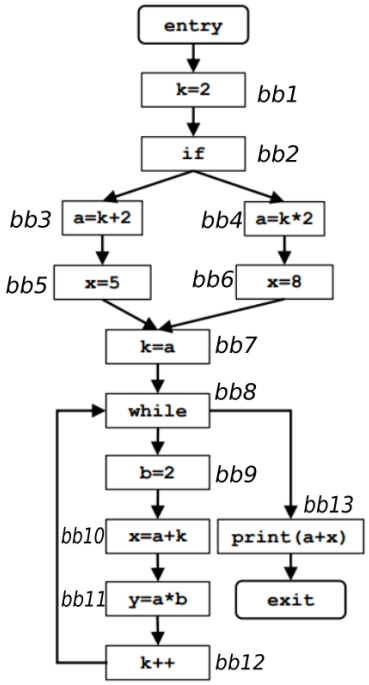


Figura 3: Constant Propagation Dataflow analysis

Partiamo prima da una tabella preliminare per riassumere il comportamento di ogni Basic Block nel control flow graph:

Basic Block	**Gen[B]**	**Kill[B]**
entry	$\emptyset$	$\emptyset$
B1	$\{(k, 2)\}$	$\emptyset$
B2	$\emptyset$	$\emptyset$
B3	$\{(a, 4)\}$	$\emptyset$
B4	$\{(a, 4)\}$	$\emptyset$
B5	$\{(x, 5)\}$	$\{(x, 8)\}$
B6	$\{(x, 8)\}$	$\{(x, 5)\}$
B7	$\{(k, 4)\}$	$\{(k, 2)\}$
B8	$\emptyset$	$\emptyset$
B9	$\{(b, 2)\}$	$\emptyset$
B10	$\{(x, 8)\}$	$\{(x, 5)\}$
B11	$\{(y, 8)\}$	$\emptyset$
B12	$\{(k, 5)\}$	$\{(k, 4)\}$
B13	$\emptyset$	$\emptyset$
exit	$\emptyset$	$\emptyset$

Ora siamo pronti a svolgere l'esercizio applicando ad ogni Basic Block del Control Flow Graph le funzioni di trasferimento per calcolare l'insieme delle variabili definite costanti.

Vediamo la tabella finale dopo l'esecuzione di due iterazioni dell'algoritmo, sufficienti ad arrivare ad una situazione di convergenza finale.

Basic Block	**Iterazione 1**		**Iterazione 2**	
	IN[B]	**OUT[B]**	**IN[B]**	**OUT[B]**
entry	-	$\emptyset$	-	$\emptyset$
B1	$\emptyset$	$\{(k, 2)\}$	$\emptyset$	$\{(k, 2)\}$
B2	$\{(k, 2)\}$	$\{(k, 2)\}$	$\{(k, 2)\}$	$\{(k, 2)\}$
B3	$\{(k, 2)\}$	$\{(a, 4), (k, 2)\}$	$\{(k, 2)\}$	$\{(a, 4), (k, 2)\}$
B4	$\{(k, 2)\}$	$\{(a, 4), (k, 2)\}$	$\{(k, 2)\}$	$\{(a, 4), (k, 2)\}$
B5	$\{(a, 4), (k, 2)\}$	$\{(x, 5), (a, 4), (k, 2)\}$	$\{(a, 4), (k, 2)\}$	$\{(x, 5), (a, 4), (k, 2)\}$
B6	$\{(a, 4), (k, 2)\}$	$\{(x, 8), (a, 4), (k, 2)\}$	$\{(a, 4), (k, 2)\}$	$\{(x, 8), (a, 4), (k, 2)\}$
B7	$\{(a, 4), (k, 2)\}$	$\{(a, 4), (k, 4)\}$	$\{(a, 4), (k, 2)\}$	$\{(a, 4), (k, 4)\}$
B8	$\{(a, 4), (k, 4)\}$	$\{(a, 4), (k, 4)\}$	$\{(a, 4)\}$	$\{(a, 4)\}$
B9	$\{(a, 4), (k, 4)\}$	$\{(b, 2), (a, 4), (k, 4)\}$	$\{(a, 4)\}$	$\{(b, 2), (a, 4)\}$
B10	$\{(b, 2), (a, 4), (k, 4)\}$	$\{(x, 8), (b, 2), (a, 4), (k, 4)\}$	$\{(b, 2), (a, 4)\}$	$\{(b, 2), (a, 4)\}$
B11	$\{(x, 8), (b, 2), (a, 4), (k, 4)\}$	$\{(y, 8), (x, 8), (b, 2), (a, 4), (k, 4)\}$	$\{(b, 2), (a, 4)\}$	$\{(y, 8), (b, 2), (a, 4)\}$
B12	$\{(y, 8), (x, 8), (b, 2), (a, 4), (k, 4)\}$	$\{(y, 8), (x, 8), (b, 2), (a, 4), (k, 5)\}$	$\{(y, 8), (b, 2), (a, 4)\}$	$\{(y, 8), (b, 2), (a, 4)\}$
B13	$\{(a, 4), (k, 4)\}$	$\{(a, 4), (k, 4)\}$	$\{(a, 4)\}$	$\{(a, 4)\}$
exit	$\{(a, 4), (k, 4)\}$	-	$\{(a, 4)\}$	-

Tabella 6: Esercizio - Constant Propoagation

2.3.3 Osservazioni sull'esercizio:

Nella seconda iterazione, quando calcoliamo $IN[B8] = OUT[B7] \cap OUT[B12]$ possiamo osservare come la variabile k, assuma un valore diverso nei due Basic Block.

Questo ci porterà a non valutarla più come costante, e così non lo sarà più nemmeno la variabile x, che usa k.

Dunque, eventuali altre iterazioni, non cambiano il flusso che osserviamo già in quest'ultimo passaggio dell'algoritmo.

3 Conclusioni

Abbiamo quindi concluso la Dataflow Analysis per questi 3 tipi di problemi noti, analizzando la struttura dei diversi Framework e risolvendo un esempio pratico per ognuno.