Valutazione a corto circuito degli operatori logici binari

Molti linguaggi di programmazione tra cui il C ed il C++ offrono un interessante meccanismo di valutazione veloce delle espressioni in cui siano presenti gli operatori logici AND e OR.

Richiamo brevemente il funzionamento di questi operatori mediante le apposite tabelle della verità (truth tables).

AND

A	B	U
0	0	0
0	1	0
1	0	0
1	1	1

OR

A	B	U
0	0	0
0	1	1
1	0	1
1	1	1

A e B sono due ingressi booleani mentre U (a sua volta booleano) rappresenta il risultato dell'operazione AND/OR.

Ricordo che un numero booleano può assumere solo i valori 0 oppure 1.

Consideriamo ora il seguente frammento di programma scritto in pseudo-codice

Se condizione1 è vera e condizione2 è vera allora

Inizio

… esegui operazioni

Fine Se

In questo caso la 'e' rappresenta la nostra operazione AND, mentre condizione1 è vera corrisponde ad A ed infine condizione2 è vera corrisponde a B.

A = condizione1 è vera (1 se condizione1 è vera, 0 altrimenti)

B = condizione2 è vera (1 se condizione2 è vera, 0 altrimenti)

Possiamo quindi riscrivere il codice di cui sopra nel seguente modo:

Se A = 1 e B = 1 allora

Inizio

… esegui operazioni

Fine Se

Oppure:

Se A AND B allora

Inizio

… esegui operazioni

Fine Se

Normalmente dovrebbero essere valutati entrambi gli ingressi (A/B) per decidere se eseguire o meno le operazioni contenute nel blocco Inizio/Fine Se.

Con la valutazione a corto circuito invece la valutazione può fermarsi prima in funzione del valore del primo ingresso. Esempio, se condizione1 è falsa (è equivalente ad avere A = 0), il risultato dell'operazione AND vale per forza di cose 0 (vedi tabella di verità AND) e quindi, è inutile andare a valutare il valore di B, in quanto qualunque valore esso avesse, non potrebbe comunque inficiare il risultato della valutazione.

In questo caso il blocco di codice compreso tra Inizio e Fine Se non verrà eseguito, giacché la condizione non viene verificata.

Questo è ciò che fanno molti linguaggi di programmazione consentendo quindi di saltare passaggi superflui ed ottimizzando quindi le performance in termini di velocità. Ciò diviene tanto più vero al crescere del numero di ingressi coinvolti nell'espressione.

Lo stesso criterio viene applicato anche con l'operatore OR.

Se A OR B allora

Inizio

… esegui operazioni

Fine Se

Anche in questo il valore di A consentirà di fermare o meno la valutazione. Infatti, se A vale 1, il risultato dell'operazione OR varrà sempre 1, indipendentemente dal valore assunto da B. In questo caso il blocco di codice verrà eseguito perché la condizione è verificata.

Infine, un’applicazione del teorema di De Morgan

Spesso capita di vedere del codice del tipo:

Se condizione1 è falsa e condizione2 è falsa allora

Altrimenti

Inizio

… esegui operazioni

Fine Se

Ovvero se la condizione primaria è vera (entrambe condizione1 e condizione2 sono false), non viene eseguita nessuna operazione, mentre se è falsa (Altrimenti) verranno eseguite delle operazioni.

Sebbene tale approccio funzioni correttamente, essa è poco leggibile e finisce per creare confusione in chi deve manutenere il codice.

Possiamo riscriverla in modo diverso andando a guardare le tavole della verità delle operazioni AND/OR.

Prima di tutto riscriviamo il blocco di codice usando A e B.

A = condizione1 è falsa (1 se condizione1 è falsa, 0 altrimenti)

B = condizione2 è falsa (1 se condizione2 è falsa, 0 altrimenti)

Quindi possiamo riscrivere il codice di cui sopra nel seguente modo

Se A AND B allora

Altrimenti

Inizio

… esegui operazioni

Fine Se

Dalla tavola della verità dell’operazione AND vediamo che se entrambi A e B valgono 1, allora U vale 1 (U è la nostra condizione primaria). In tutti gli altri casi U vale 0.

Possiamo allora riscrivere il nostro codice in modo diverso, ma prima dobbiamo introdurre l’operatore NOT (negazione booleana).

L’operatore NOT è definito dalla seguente tabella:

A	NOT (A)
0	1
1	0

Il codice diviene così:

Se NOT (A) OR NOT (B) allora

Inizio

… esegui operazioni

Fine Se

Ovvero:

Se condizione1 è vera oppure condizione2 è vera allora

Inizio

… esegui operazioni

Fine Se

Si è così eliminato un blocco di codice vuoto a favore del blocco “Altrimenti”, ottenendo un codice più pulito.

Per arrivare a questo risultato ho applicato il primo teorema di De Morgan di seguito enunciato.

NOT (A AND B) = NOT (A) OR NOT (B)

Riscriviamo il codice originale nel seguente modo:

Se A AND B allora

Altrimenti Se NOT (A AND B)

Inizio

… esegui operazioni

Fine Se

Successivamente:

Se NOT (A AND B) allora

Inizio

… esegui operazioni

Fine Se

Applichiamo il teorema di De Morgan:

Se NOT (A) OR NOT (B) allora

Inizio

… esegui operazioni

Fine Se

Ovvero:

Se condizione1 è vera oppure condizione2 è vera allora

Inizio

… esegui operazioni

Fine Se