Abbiamo visto la rappresentazione di funzioni mediante tavole di verità, mediante formule algebriche, la rappresentazione grafica e quella mediante mappe di Karnaugh. Un’ultima rappresentazione, che è adeguata per l’implementazione di algoritmi su un sistema di calcolo (le altre vengono bene per la manipolazione manuale), è BDD (Binary Decision Diagrams).

L’idea di funzionamento di questa rappresentazione è abbastanza semplice: Innanzitutto se si vuole rappresentare una funzione di per esempio 3 variabili (a, b, c) si rappresentano in elenco (in colonna) le variabili in ingresso e quella di uscita. L’idea è quella di percorrere un grafo partendo dall’alto e ad ogni passo avere la possibilità di differenziare il valore 0 e il valore 1. Dai due valori possibili si diramano altri due rami per ciascuna variabile successiva alla prima fino a raggiungere il valore di output.

root: x

A 0 1

B: 0 1 0 1

C: 0 1 0 1 0 1 0 1

U: 0 1 1 0 1 1 0 0

È possibile ovviamente scrivere la mappa di Karnaugh per rendere più facile la lettura agli esseri umani. È possibile inoltre, per risparmiare, vedere quali valori sono equivalenti e ridirigere tutti i collegamenti che arrivano a uno stesso valore di Uscita a una singola foglia. Alla fine, si dovrebbe ottenere un’unica foglia di uscita che contiene il valore 0 e un’unica foglia che contiene il valore 1, alle quali sono collegati tutti i possibili collegamenti che scendono da c.

Se poi si nota che in alcuni casi il valore assunto da una delle variabili diventa ininfluente allora si può eliminare un nodo (le cui uscite vanno nella stessa direzione ovviamente) e sostituirlo con un unico arco.

Ecco quindi, che, partendo da una rappresentazione ridondante, applicando alcune regolette si può arrivare a una riduzione del grafo: ciò può portare anche a una semplificazione nella realizzazione FISICA della nostra funzione. Per esempio, nella parte a destra dell’albero il valore di C diventa inutile (indipendentemente da esso la funzione ritorna 1 o 0), quindi lì la funzione è semplificabile eliminando il contributo di C: ciò permette di risparmiare anche quando si costruisce fisicamente il circuito che dovrà eseguire la funzione.

root: x

A 0 1

B: 0 1 0 1

C: 0 1 0 1

U: 0 1 1 0 1 0

root: x

A 0 1

B: 0 1 0 1

C: 0/ \1 0\ /1 \ /

U: 0 1

Altro esempio:

root: x

A 0 1

B: 0 1 0 1

C: 0 1 0 1 0 1 0 1

U: 0 1 0 1 1 1 0 0

In questo caso i due nodi di C a sinistra dell’albero danno la stessa informazione (entrambi vanno a 0 se sono 0 e vanno ad 1 se sono 1), quindi si possono sostituire i due nodi con un nodo solo. Ma poiché adesso entrami i valori di b (a sinistra) portano allo stesso nodo in C si può tranquillamente eliminare il nodo inutile e sostituirlo con un unico arco che va da A all’unico nodo di C.

root: x

A 0 1

B: 0 1

C: 0 1 0 1 0 1

U: 0 1 1 1 0 0

Poiché la parte destra dell’albero è uguale all’esempio di prima, la si può compattare analogamente.

root: x

A 0 1

B: 0 1

C: 0 1

U: 0 1 1 0

|  |  |  |
| --- | --- | --- |
| b c \ a | 0 | 1 |
| 00 | 0 | 1 |
| 01 | 1 | 1 |
| 11 | 1 | 0 |
| 10 | 0 | 0 |

Se si costruisce la mappa di Karnaugh:

La formula semplificata che ne uscirebbe sarebbe u = -a \* c + a\*-b; Ciò si può riscontrare nella rappresentazione grafica ad albero (BDD).

C’è in realtà una piccola differenza nelle rappresentazioni BDD: essa è che per poterla avere è necessario stabilire l’ordine in cui si elencano le variabili e purtroppo al variare dell’ordine delle variabili può cambiare il grado di semplificazione/ottimizzazione che si può raggiungere applicando le regolette presentate. Nella pratica ciò che si può fare è provare più ordinamenti e vedere quale porta a una semplificazione migliore del grafo.

Altro esempio:

root: x

A: 1 0

B: 1 0 1 0

C: 1 0 1 0 1 0 1 0

U: 1 0 1 0 1 1 0 0

Mappa di Karnaugh.

|  |  |  |
| --- | --- | --- |
| ab \ c | 0 | 1 |
| 00 | 0 | 0 |
| 01 | 1 | 1 |
| 11 | 0 | 1 |
| 10 | 0 | 1 |

Si possono aggregare i due 1 adiacenti nell’angolo in basso a destra e i due 1 nella seconda riga e trovare la formula u = -A\*B + A\*C. Oppure ne si cerca una cancellando le parti inutili dell’albero:

root: x

A: 1 0

B: 1 0

C: 1 0

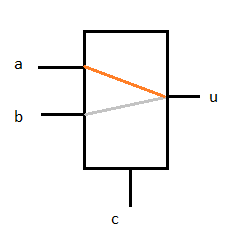
U: 1 0 1 0

Secondo l’albero la formula minimale con cui rappresentare la funzione è u = A\*C + -A\*B -> la stessa che si è trovato prima.

Riassumendo, abbiamo visto 5 tipe di rappresentazioni (4 da usare a mano e una quando si usano sistemi di sviluppo dell’hardware sulla base di applicazioni informatiche): Algebrica, a tavola di verità, Grafica, Mappa di Karnaugh e BDD.

Si può adesso procedere a ideare la realizzazione di dispositivi per la realizzazione di un sistema di calcolo. Il primo dispositivo che andiamo a studiare è il multiplexer. Un multiplexer può essere visto come un commutatore

Questo è un generico multiplexer.



C ha una funzione di controllo: a seconda che valga 0 o 1 l’uscita cambia. Funge quindi da deviatore: quando c = 0 l’uscita è = all’ingresso b, quando c = 1 l’uscita è = all’ingresso a.

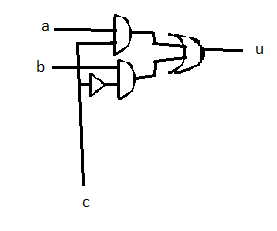
Si può rappresentare il funzionamento del multiplexer mediante una tavola di verità:

|  |  |  |  |
| --- | --- | --- | --- |
| c | a | b | u |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

Si potrebbe scrivere algebricamente mediante una formula in forma normale congiuntiva. Tuttavia, da qui si può passare alla rappresentazione sotto forma di mappa di Karnaugh

|  |  |  |
| --- | --- | --- |
| ab\c | 0 | 1 |
| 00 | 0 | 0 |
| 01 | 1 | 0 |
| 11 | 1 | 1 |
| 10 | 0 | 1 |

A questo punto si possono unire le due coppie di caselle adiacenti per semplificare il tutto e ottenere u = b\*-c + a\*c; Notare che c è l’unica variabile che è presente due volte e che viene negata in un caso e nell’altro no. Questa è una rappresentazione grafica di come si dovrebbe costruire quindi il circuito secondo la formula che abbiamo trovato.



(Si ricorda che il triangolino, che in realtà dovrebbe avere anche un pallino, indica negazione, il semicerchio la funzione AND e la doppia semicirconferenza/spicchio di luna rappresenta l’OR).

Ovviamente per poterlo veramente realizzare sarebbero necessarie delle nozioni di elettronica (che permettono per esempio di utilizzare dei transistor per costruire delle funzioni AND e OR). Noi non affronteremo questi argomenti perché a noi interessa unicamente di minimizzare il costo della realizzazione di questi circuiti/dispositivi (della realizzazione si occupano ben poche persone nel mondo).

Volendo si può espandere il multiplexer: per fare ciò è necessario aumentare il numero di input (da 2 a per esempio 3) tuttavia a questo punto per permettere a c di determinare tra 3 o più ingressi possibili è necessaria una rappresentazione di c a più di un bit. Se si rappresenta c su 2 bit ciò permette di scegliere tra 4 valori di ingresso (I0, I1, I2, I3). Proseguendo su questa strada i multiplexer si espandono per potenze di 2 ( 1,2,3,4, ecc. variabili di controllo permettono di scegliere tra 2,4,8,16,ecc. valori di ingresso). L’aumento di una variabile di controllo porta a una duplicazione delle dimensioni del dispositivo.

Esempio:

