PROCEDIMIENTO MÉTODO McCLUSKEY

EVELYN TABARES VALENCIA

A continuación, daremos una breve explicación del código que permite realizar el método de minimización de Quine-McCluskey.

Empezamos con la función main(), la cual nos redirigirá a diferentes funciones. Como primera instancia, permite ingresar al usuario a través del teclado los minterminos a evaluar, estos se almacenarán en la variable 'minterms' para luego ser separados y organizados de menor a mayor en una lista llamada 'a'.

A través de un ciclo for recorremos la lista de 'a' de menor a mayor y en una variable '*n_var*' almacenamos la cantidad de dígitos que se pueden encontrar en la representación binaria del último número de la lista. De esta manera obtenemos la cantidad de variables con la cual trabajaremos.

Clasificamos los minterminos según la cantidad de números '1' que se puedan encontrar en su representación binaria, esto lo logramos usando una lista 'grupo', donde su índice '*index*' se le asigna la cantidad de '1' que contiene el binario. Así, grupo[index] → index = a[i].count('1') .

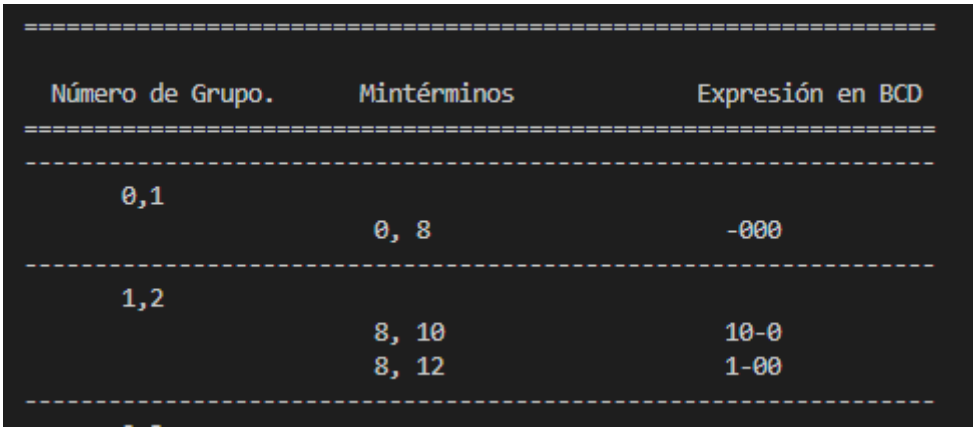
Con estos datos podemos imprimir la tabla de minterminos, clasificados según su cantidad de números 1 y a su lado la representación decimal y binaria.

Luego de esto empezaremos a combinar los binarios, entramos a un ciclo while que solo se detendrá cuando no haya más números que combinar, esto lo verificamos a través de una función llamada '*comprobar_vacia'*. Al entrar al ciclo nos redirige a la función '*combinar_pares'*, que toma como parámetros 'grupo' y 'desmarcado'. Dicha función toma un mintermino de un grupo y lo compara con cada número del siguiente grupo para esto hace uso de varios ciclos for.

La función anterior nos redirige a '*compara_binarios'* que recibe como parámetros dos minterminos, uno del grupo actual y el otro del grupo siguiente. Estos números se encuentran en representación binaria con cada digito por separado, como elementos de una lista. Al terminar de evaluar, la función nos retorna dos variables: 'b', la cual será True en caso de que se puedan combinar los números o False, en el caso contrario. La otra variable recibida, '*pos*', me indica la posición en la que se deberá hacer el cambio de digito binario por el carácter '-'.

Los números que puedan combinarse se almacenarán en una lista llamada '*lista_verificacion'* y el elemento combinado con el carácter '-' incluido, se almacena en la lista '*siguiente_grupo'*. Allí mismo empezamos a imprimir la tabla donde se muestran a qué grupos pertenecían ambos números combinados, su representación en decimal y binario.

Ejemplo:



Número de Grupo.	Mintérminos	Expresión en BCD
0,1		
	0, 8	-000
1,2		
	8, 10	10-0
	8, 12	1-00

Luego de esto, recorremos la lista de todos los minterminos (‘*grupo’*), en busca de aquellos elementos que no se hayan podido combinar, es decir que no se encuentren en ‘*lista_verificacion*’ y los agregamos a una nueva lista titulada ‘*desmarcado*’. Para finalizar la función ‘*compara_binarios’*, retornamos la lista de números combinados (‘*siguiente_grupo’*) y lista de números que no se combinaron (‘desmarcado’).

Volvemos a la función ‘*main*’ y nos encontramos con un llamado a la función ‘*eliminar_redundantes’*, donde como lo indica, eliminamos las combinaciones repetidas que se encuentren en la lista de grupos a combinar (‘*grupo*’).

Ejemplo:

grupo[i] **->** [[‘8 10’, ‘10-0’] , [‘8 10’ ,‘10-0’]] Se elimina uno de los elementos.

El salir del ciclo while indica que ya se han hecho todas las combinaciones posibles entre los minterminos y lo único que resta por hacer es eliminar aquellas combinaciones redundantes dentro de la lista de primos, es decir a aquellos elementos que no se combinaron, los cuales se almacenan en ‘*desmarcado*’.

Ejemplo:

desmarcado[i] **->** [[’10 14 11 15’, ‘1-1-‘] , [10 11 14 15’ , ‘1-1-‘]] Se elimina uno de los elementos.

Imprimimos los primos

Con esto damos por terminada la búsqueda de primos e imprimimos la lista ‘*desmarcado*’. Nos aseguramos de que en caso de que todos los minterminos se puedan combinar entre sí, el resultado sea igual a 1, de lo contrario convertimos cada elemento de la lista ‘desmarcado’ en su representación en letras, de A-Z para hacer mas sencillo su seguimiento. Para esta tarea llamamos a la función ‘*binario_a_letras’*, que nos retorna en formato string cada digito del binario convertido en una letra, siempre y cuando no sea el carácter ‘-‘.

En busca de primos implicantes esenciales

Para hallar los primos implicantes esenciales implementamos la lista de verificación, llamada a través de la función '*tabla_verificacion'* a la que le entregamos como parámetros los primos ('*primos*') y los minterminos ('*a*').

Creamos una nueva lista llamada 'decimales', cada fila la reconoceremos como representación de un primo y en cada columna de dicha fila almacenamos un mintermino que lo compone.

Creando la tabla

Instanciamos una nueva lista llamada 'tabla' a la cual en su fila y columna 0, le asignaremos el string 'Expresiones'. Ahora lo que queda de la fila 0, lo usaremos para almacenar cada mintermino en una columna diferente.

Nos encontramos con otro for que va de 0 a la longitud de la lista 'decimales', con cada iteración imprimiremos un primo con su representación en letras y lo almacenaremos en la columna 0. Anidamos otro ciclo for para recorrer cada columna de la tabla (los cuales son minterminos) y verificamos con un if, si el mintermino/columna actual se encuentra en la lista de decimales que componen al primo de dicha fila. En dicho caso, debe almacenarse una 'X' en la posición o un espacio ' ' de lo contrario. Luego de esto imprimimos la tabla de validación con la función importada '*tabulate',* que permite imprimir la lista 'tabla' con todos los datos de la tabla de validación de manera ordenada y prolija.

Ejemplo:

TABLA DE VALIDACIÓN

Expresiones	0	5	6	7	8	10	11	12	14	15
B'C'D'	X				X					
A'BD		X		X						
AD'					X	X		X	X	
BC			X	X					X	X
AC						X	X		X	X

Para verificar que un elemento sea un implicante primo esencial, es necesario analizar la tabla que hemos llenado. Si se encuentra una columna con una sola 'X', se agrega el primo al que pertenece a la lista de posibles primos implicantes, en este caso llamada 'valido'. Usamos un contador de caracteres 'X' llamado *'contmin'* y a través de un if verificamos que '*contmin*' == 1 para agregar el primo a la lista 'valido', siempre y cuando no se encuentre ya en esta.

El siguiente paso es buscar aquellas columnas en las que hay mas de una 'X', aquí es donde debemos a través de un contador, calcular la cantidad de primos validos e inválidos. Si el primo ya se ha almacenado en validos usamos el contador '*cont_valido'* y le agregamos 1, sino usamos 'cont_inval' y le agregamos 1, pero además creamos una lista temporal ('*temp_inval'*) donde almacenaremos específicamente la fila, donde encontramos cada primo invalido de dicha columna/mintermino.

A través del uso de if comprobaremos qué primos se relacionan con expresiones validas, descartándolo de ser un primo esencial y cuales se relacionan solo con expresiones invalidas, convirtiéndolo en un primo esencial.

Usamos:

if cont_valido >= 1 and cont_inval >=1: #Se convierten en inválidos las expresiones que se relacionan con al menos un valido

Agregamos la expresión a la lista de términos inválidos 'invalido'

elif cont_valido == 0 and cont_inval >= 1: #Las expresiones que solo se relacionan con inválidos, se convierten en validos

Para este paso agregamos un for, en el cual recorreremos la lista temporal 'temp_inval', donde almacenamos la fila de cada termino invalido que se encuentra en dicha columna/mintermino.

'temp_inval' se usa como coeficiente de 'primos' para acceder al primo al que nos referimos con dicha fila guardada.

if primos[int(temp_inval[z])][1] not in invalido: #Si la expresión/primo no existe en la lista de invalidos

Si este if se cumple agregamos la expresión a la lista de válidos. Por otro lado, si el primo existe tanto en la lista de 'validos' como de 'invalidos', se remueve de la lista de 'validos'.

Al final de esta función retornamos la lista de primos validos ('valido') y a continuación de vuelta en la función 'main', imprimimos dicha lista en su representación en letras. Esto es lo que veremos en pantalla como primos implicantes esenciales.