Minimización de Funciones Lógicas
El algoritmo de Quine – McClusky
explicado y mejorado

(II)

Macluskey, 2023

En este segundo artículo de la serie dedicada al algoritmo de Quine-McClusky voy a definir con el máximo detalle posible el algoritmo de Quine-McClusky. En el primer artículo describí la nomenclatura que usaré durante toda la serie para las variables individuales (C1, C2...C*n* si las variables son afirmativas; N1, N2...N*n* si son sus complementarias, y '+', '*' –OR, AND- como operadores lógicos, y también expliqué cómo obtener la forma canónica de una cierta expresión lógica, pues, como veremos, la forma canónica de la expresión es la única entrada para el algoritmo.

Consultar dicho artículo para mayor información.

El algoritmo de Quine-McClusky (QM como abreviatura) consta de una serie de pasos secuenciales, aunque cada uno de ellos tiene a su vez subpasos, iteraciones, selecciones y todo tipo de instrucciones.

Vamos ya con la definición del algoritmo.

Paso 1. Detección y eliminación de variables "implicantes"

Este paso es nuevo, no existe, que yo sepa, en la definición standard del algoritmo. Yo, al menos, no he encontrado nada parecido.

Se trata de buscar si una o varias variables individuales de las contenidas en la fórmula original implican a la expresión completa, sea en su forma afirmativa, sea negada. Se buscan, en concreto, variables *Cx* o *Cy* que cumplan alguna de estas dos expresiones: *Cx*➔*Forig*, o bien, en su caso, *Ny*➔*Forig'*, siendo *Forig* la fórmula original, que de momento desconocemos, dado que sólo conocemos su forma canónica, y *Forig'* la negación de dicha fórmula original.

Veamos qué significa que exista una variable de este estilo en la fórmula original.

Cx➔*Forig*, es decir, *Cx* implica a *Forig* ocurre cuando la fórmula original tiene la forma *Forig=Cx+E,* es decir, cuando la variable *Cx* está sumando (OR) a otra cierta expresión *E* que da como resultado *Forig*; en cuanto a *Ny*➔*Forig'*, ocurre cuando la fórmula original tiene la forma *Forig=Cy*E*, o sea, la variable *Cy* está multiplicando (AND) a otra cierta expresión *E* que da como resultado *Forig*.

Si se localizara alguna de estas variables, entonces **dicha variable puede eliminarse completamente de la forma canónica**, reduciendo a la mitad su tamaño y, por consiguiente, reduciendo de forma dramática el uso de recursos necesarios para la ejecución del algoritmo.

Obviamente, no siempre este paso encuentra variables "implicantes" que eliminar, pero el coste de ejecución del paso es ridículo comparado con la posible ganancia conseguida en caso de encontrar una o varias variables que puedan eliminarse.

Este Paso 1 funciona de la forma siguiente:

1) Determinar cuántas variables individuales contiene la forma canónica, en base al número de bits que la componen, teniendo en cuenta que dicho número de bits es siempre 2^n. Conocido dicho número *n*, componer la tabla de verdad de las *n* variables implicadas, tanto en su forma afirmativa como negada, y habilitar una tabla de variables implicantes en la que se anotarán las existentes con su operador relacionado.

2) Revisar la forma canónica original con las tablas de verdad de las *n* variables, tanto en su forma afirmativa como en su forma negada, comprobando si se cumple alguna de las dos condiciones siguientes, o ambas simultáneamente:
   a) Que todos los unos de la tabla de verdad de la variable tratada sean también unos en la forma canónica original.
   b) Que todos los ceros de la tabla de verdad de la variable tratada sean también ceros en la forma canónica original.

3) Si ninguna de las variables originales cumple alguna de las dos condiciones a ó b, entonces terminar el paso y continuar el algoritmo por el Paso 2 con la forma canónica resultante, la original si no se ha encontrado ninguna variable implicante o la resultante tras la ejecución de este Paso 1.

4) En caso de que para una variable individual *Cx* se cumpla alguna de las dos condiciones citadas, entonces:
   4.1) Si se cumple la condición a) anterior, entonces llevar a la tabla de variables implicantes la propia variable *Cx* con el operador '+'.
   4.2) Si se cumple la condición b) anterior, entonces llevar a la tabla de variables implicantes la variable *Cx,* pero ahora con el operador '*'.

4.3) Téngase en cuenta que la variable que cumpla alguna de las dos condiciones puede ser negativa, *Nx*. El procedimiento es idéntico, se lleva a la tabla de variables implicantes a esa *Nx* con el operador ('+', '*') que corresponda.
4.4) Se eliminan de la forma canónica original todos aquellos unos o ceros que coinciden con los de la variable Cx, según se haya cumplido respectivamente la condición a) o b) anteriores. La forma canónica se reduce, por tanto, a la mitad, dado que todas las tablas de verdad de las variables individuales tienen el mismo número de ceros que de unos.
4.5) Repetir todo el proceso desde el número 1, para intentar localizar nuevas variables implicantes en la forma canónica resultante.

5) En el caso particular de que en la comprobación de los ceros y los unos de una cierta variable individual *Cx* se cumplieran simultáneamente ambas condiciones a) y b), esto quiere decir que la tabla de verdad de la expresión original, o la resultante si se hubiera reducido mediante el paso 4 anterior, es exactamente la misma que la de la variable individual *Cx*. En este caso la fórmula buscada es simplemente *Cx*; hay que pasar dicha variable *Cx* a la tabla de variables implicantes, pero sin signo, y parar inmediatamente el proceso: aunque resten variables sin eliminar en la forma canónica, son irrelevantes y pueden ignorarse.

6) Con la forma canónica resultante se ejecutan normalmente los pasos que restan del algoritmo QM, el cual obtiene finalmente una determinada fórmula. En el caso de que en este Paso 1 haya encontrado y extraído una o más variables implicantes, entonces deben ser incluidas en la fórmula final, extrayéndolas de la tabla de variables implicantes en el orden inverso al que fueron introducidas, es decir, dicha tabla es en realidad una pila (LIFO) en la que el último elemento que se introdujo será el primero en salir (Last Input, First Output). Las variables se incluyen sumando o multiplicando a la totalidad de la fórmula obtenida, según sea el operador que las acompaña.

Veamos un ejemplo de este Paso 1. Sea la forma canónica siguiente: **0000000010101110**. Son 16 bits, y por lo tanto tendremos en la expresión resultante 4 variables individuales, C1, C2, C3 y C4, puesto que $16=2^4$.

En primer lugar se compone la tabla de verdad de las cuatro variables implicadas, tanto en su forma afirmativa como negativa. En este ejemplo de cuatro variables serían:

C1: 0000000011111111
N1: 1111111100000000
C2: 0000111100001111
N2: 1111000011110000
C3: 0011001100110011
N3: 1100110011001100
C4: 0101010101010101
N4: 1010101010101010

En la comparación de las diferentes tablas de verdad de las variables individuales, afirmativas o negativas, con la forma canónica, al compararla con la tabla de verdad de C1 vemos que en todas las posiciones en las que hay un cero en dicha tabla de verdad también hay un cero en la forma canónica de la expresión original:

Forma canónica: 0000000010101110
Tabla de verdad de C1: **00000000**11111111

Esto quiere decir que la variable C1 es implicante de la función original. Se lleva a la tabla de variables implicantes con el operador "*", dado que lo que coinciden son los ceros, y se eliminan de la forma canónica todos los bits correspondientes a dichos ceros de C1,que en este caso son los ocho primeros. El resultado es:

Nueva forma canónica: 10101110

Var. implicante	Operador
C1	*

Como la nueva forma canónica tiene ahora 8 bits, sólo tiene 3 variables, luego también se deben eliminan las dos variables C1 y N1 ya procesadas de las tablas de verdad a comprobar, dado que esa variable ha sido incorporada a la tabla de variables implicantes y extraída de la función. Las tablas de verdad de las variables restantes, una vez eliminadas las variables C1 y N1, quedan así:

C2: 00001111
N2: 11110000
C3: 00110011
N3: 11001100
C4: 01010101
N4: 10101010

El paso se repite nuevamente hasta que no se encuentren más variables implicantes. Vemos ahora en esta segunda iteración que, al comparar la nueva forma canónica con las tablas de verdad, todos los unos de la variable N4 coinciden con unos en la forma canónica actual:

Forma canónica actual: 10101110
Tabla de verdad de N4: **1**0**1**0**1**0**1**0

Luego N4 es implicante también, sumando, dado que lo que coinciden esta vez son los unos. Así, tras extraer la variable N4 (y también la C4, claro está) de la forma canónica y de la tabla de verdad de las variables restantes, y añadirla a la tabla de variables implicantes, queda:

Nueva forma canónica: 0010

Var. implicante	Operador
C1	*
N4	+

Y las tablas de verdad resultantes son:

C2: 0011
N2: 1100
C3: 0101
N3: 1010

Una nueva iteración del paso anterior encuentra ahora que todos los ceros de C2 coinciden con ceros en la forma canónica, luego C2 es implicante, multiplicando:

Forma canónica actual: 0010
Tabla de verdad de C2: **00**11

Aplicando de nuevo los cambios preconizados, queda:

Nueva forma canónica: 10

Tablas de verdad resultantes:
C3: 01
N3: 10

Var. implicante	Operador
C1	*
N4	+
C2	*

La nueva forma canónica coincide completamente con N3, tanto los ceros como los unos, por lo que esta variable se debe incorporar también, pero sin signo. En este caso N3 es la última variable a tratar; toda la fórmula ha sido resuelta sin llegar siquiera a necesitar que se ejecute el resto del algoritmo.

La tabla de variables implicantes ha quedado definitivamente así:

Var. implicante	Operador
C1	*
N4	+
C2	*
N3	

Ahora, una vez finalizado este Paso 1, se pasaría a ejecutar el resto del algoritmo QM; en este ejemplo concreto, sin embargo, no hace falta, dado que toda la fórmula ha sido determinada en este paso y es obviamente mínima.

Por fin, a la hora de componer la expresión final, en el Paso 8, a la fórmula devuelta por el algoritmo **se añaden las variables implicantes, con su operador, en orden inverso al de introducción en la tabla**. En el ejemplo que hemos seguido primero se extrae N3, sin operador; posteriormente, C2 con el operador '*', lo que da N3*C2; a continuación se extrae N4 con el operador '+', lo que deja N3*C2+N4; y finalmente se extrae C1 con su operador '*', resultando así la fórmula definitiva: C1*(N3*C2+N4).

Si una vez extraídas todas las variables posibles la forma canónica resultante sí que tiene contenido, lo habitual, se ejecuta el resto de pasos del algoritmo QM. Tras su ejecución completa, a la expresión resultado del algoritmo se le deben añadir las variables implicantes de la forma antes descrita. Supongamos que la tabla de variables implicantes final ha quedado así:

Var. implicante	Operador
C7	*
N3	+
C4	+

Supongamos también que el algoritmo ha devuelto una expresión *Eqm,* en la que, obviamente, no pueden estar C7, N3 ni C4, ni tampoco sus complementarias N7, C3 y N4. En ese caso la expresión final quedará:

(*Eqm*+C4+N3)*C7

Hasta aquí la descripción del Paso 1, Detección y eliminación de Variables Implicantes.

Este proceso de eliminación de variables implicantes, como ya he indicado, no lo he encontrado en ninguna publicación de las que he leído, y han sido decenas. Ignoro si soy el primero que lo describe o si está ya descrito en algún lugar al que yo no haya accedido. En cualquier caso, ahí queda su definición por si algún colega desea implementarlo.

El coste de ejecución de este paso es mínimo comparado con la posible reducción a la mitad (o la cuarta parte, etc.) de la forma canónica a tratar, reduciendo en consonancia el tiempo necesario para tratarla a la raíz cuadrada del necesario, por lo que claramente compensa ejecutarlo aun cuando no encuentre ninguna variable implicante.

Con formas canónicas de 8 bits y tres variables individuales se encuentra al menos una variable implicante en 134 de las 256 formas canónicas posibles de ese tamaño; en formas canónicas de 16 bits y cuatro variables individuales se encuentran variables implicantes en 5.638 de las 65.536 formas canónicas posibles. Con tamaños mayores de forma canónica la proporción de éxito baja pero, dado su escaso coste, continúa siendo rentable buscar y eliminar las posibles variables implicantes en términos de consumo de recursos.

A continuación describiré el resto de pasos del algoritmo de Quine-McClusky tal como se define en la literatura, explicando en cada paso, si procede, lo que he ido aprendiendo en mis investigaciones y pruebas.

Para describir los pasos restantes del algoritmo me apoyaré en principio en un ejemplo, uno cuya forma canónica es la siguiente:

10111001111111011111111111111111

Son 32 bits, cuatro ceros y 28 unos, y por tanto contiene cinco variables individuales, C1...C5, dado que $2^5 = 32$.

En el primer paso del algoritmo, la eliminación de variables implicantes que ya hemos visto, comprobamos que todos los unos de la tabla de verdad de la variable C1 coinciden con unos en la forma canónica dada:

Forma canónica: 10111001111111011111111111111111
Tabla de verdad de C1: 00000000000000001111111111111111

Por lo tanto C1 es implicante de la función completa. Se extrae C1 de la forma canónica (en este caso son sus últimos 16 unos), y se lleva, sumando, a la tabla de Variables Implicantes:

Var. implicante	Operador
C1	+

En la forma canónica de 16 bits y cuatro variables resultante no es ya posible encontrar más variables implicantes, por lo que dicha forma canónica resultante, que por consiguiente es la que se toma como entrada en los pasos restantes del algoritmo, es:

1011100111111101

Ahora son 16 bits, de los que 12 de ellos son unos y los 4 restantes, ceros. También son cuatro las variables individuales, de C2 a C5: recordemos que la variable C1 ha resultado ser implicante y por ello ha sido extraída de la original; no se puede reutilizar su nombre, pues en caso contrario la expresión final no será correcta cuando se añada como último paso dicha variable C1 sumando a la expresión obtenida por el algoritmo.

Paso 2- Carga inicial de los *minterms*

Antes de describir este paso necesito aclarar una cierta terminología que se usa continuamente en el algoritmo: *minterm* e *implicante primo*.

Un *minterm* (término mínimo) es cada una de las combinaciones que dan origen a un 1 en la forma canónica. Así, el tercer uno de la forma canónica anterior, que se encuentra en la cuarta posición de la forma canónica, da origen biunívocamente al *minterm* 0011 (N2*N3*C4*C5). En el algoritmo sólo se usa el índice 0011, su conversión al término N2*N3*C4*C5 está implícita. Posteriormente se verá en detalle esta conversión. El nombre "*minterm*" viene porque basta con que uno solo de estos términos se cumpla para que la función completa se cumpla, consecuencia obvia de que todos ellos están sumados (OR) entre sí para conformar la función global.

Existen también los "*maxterms*", en caso de que la función se definiera en Forma Normal Conjuntiva, un Producto de Sumas: cada término, una suma de variables individuales, es un *maxterm* porque, al estar todos ellos multiplicados (AND) entre sí, se precisa que todos y cada uno sean ciertos para que sea cierta la función global.

Un *implicante primo* (IP a partir de ahora) es el resultado de fusionar varios *minterms*, y se llama de esta forma porque cada uno de ellos implica a la función original, es decir, cada uno de los IPs cumple que *IP➔Exp.Original*. Veremos cómo se forman un poco más adelante.

Definición del Paso 2: Para cada *minterm* de la forma canónica dada, es decir, **para cada uno que allí aparece**, se carga un elemento en una tabla base de implicantes primos, calculando adicionalmente el número de unos que tiene el *minterm*.

En nuestro ejemplo la forma canónica tiene doce unos, por lo que doce son los elementos que se cargarán inicialmente en la tabla. Son estos:

Num.unos	Implicante primo
0	0000
1	0010
1	0100
1	1000
2	0011
2	1001
2	1010
2	1100
3	0111
3	1011
3	1101
4	1111

Como se puede ver, además del valor del *minterm* se ha cargado también el número de unos que tiene dicho implicante primo. "0000" no tiene ningún uno, todos son ceros, mientras "0111" tiene tres, etc. Esto será importante para el paso siguiente, como veremos en un momento.

La tabla se clasifica por el número de unos y el valor del implicante primo, quedando como se puede ver más arriba. Sí, en este punto un *minterm* y un *implicante primo* es lo mismo; a partir de ahora ya serán diferentes.

Paso 3 – Cálculo de los implicantes primos

Para cada uno de los implicantes primos de la tabla base, compararlo con todos y cada uno de los IPs de la propia tabla que tengan un número de unos exactamente mayor en uno al del IP comparado.

Así, el IP "0000", con cero unos, se comparará con todos los IPs que tengan un solo uno, es decir, con "0010", "0100" y "1000", pero no con el resto. Y así con todos y cada uno de los Implicantes Primos de la tabla.

En el proceso se generan nuevos elementos en otra tabla similar, llamémosla "tabla nueva" para distinguirla de la tabla base que se procesa, en la que **se genera un nuevo elemento cuando los dos elementos comparados sean exactamente iguales excepto en una única posición**.

Si eso ocurre se genera en la tabla nueva un elemento igual a cualquiera de ellos, pero con un guión (–) en la única posición en la que ambos difieren. Así, de la comparación entre "0000" y "0010" se obtiene el nuevo elemento "00–0", donde se ha sustituido el tercer carácter, que es el que difiere, por un guión. En dicho elemento de la tabla nueva se añade también el número de unos que tiene dicho nuevo elemento, calculado con el mismo criterio que en la carga inicial. Si hubiera más de una diferencia entre los dos elementos comparados no se genera ningún elemento en la tabla nueva.

El fundamento de este paso es que comparar dos elementos de la tabla equivale a tratar dos términos, dos productos de la Forma Normal Disyuntiva (FND) de la expresión dada. Cuando se encuentra que sólo se diferencian en un bit, en un elemento es un uno y un cero en el otro, esto implica que se han encontrado dos términos que pueden ser fusionados aplicando la propiedad distributiva, es decir, sacando factor común entre ambos y eliminando la variable cuyos valores difieren.

Sean, por ejemplo, los elementos 0101 y 1101 los que estamos comparando. Sólo se diferencian en un único bit, el primero de ellos, y darán origen al IP –101. Estos dos elementos representan respectivamente a N2*C3*N4*C5 y C2*C3*N4*C5. Como ambos términos están sumados en dicha FND, se puede aplicar la reducción siguiente, sacando como factor común el término común C3*N4*C5 de la forma:

N2*C3*N4*C5 + C2*C3*N4*C5 = (N2+C2)*C3*N4*C5 = (1)*C3*N4*C5 = C3*N4*C5;

Esto es exactamente lo que representa el implicante primo –101, que es el resultado de la comparación: que pueden fundirse estos dos términos en uno más sencillo que los engloba a ambos.

Volvamos a nuestro ejemplo.

La primera pasada de comparaciones y sus resultados son los siguientes (recordemos que cada elemento sólo se compara con todos los elementos que tienen exactamente un uno más que el comparado):

(0) 0000 – (1) 0010 = 00–0
(0) 0000 – (1) 0100 = 0–00
(0) 0000 – (1) 1000 = –000
(1) 0010 – (2) 0011 = 001–
(1) 0010 – (2) 1001 = **NO** (hay más de una diferencia, no se genera elemento en la tabla nueva)
(1) 0010 – (2) 1010 = –010
(1) 0010 – (2) 1100 = **NO**
(1) 0100 – (2) 0011 = **NO**
(1) 0100 – (2) 1001 = **NO**
(1) 0100 – (2) 1010 = **NO**
(1) 0100 – (2) 1100 = –100
(1) 1000 – (2) 0011 = **NO**
(1) 1000 – (2) 1001 = 100–
(1) 1000 – (2) 1010 = 10–0
(1) 1000 – (2) 1100 = 1–00
(2) 0011 – (3) 0111 = 0–11
(2) 0011 – (3) 1011 = –011
(2) 0011 – (3) 1101 = **NO**
(2) 1001 – (3) 0111 = **NO**
(2) 1001 – (3) 1011 = 10–1
(2) 1001 – (3) 1101 = 1–01
(2) 1010 – (3) 0111 = **NO**
(2) 1010 – (3) 1011 = 101–
(2) 1010 – (3) 1101 = **NO**
(2) 1100 – (3) 0111 = **NO**
(2) 1100 – (3) 1011 = **NO**
(2) 1100 – (3) 1101 = 110–
(3) 0111 – (4) 1111 = –111
(3) 1011 – (4) 1111 = 1–11
(3) 1101 – (4) 1111 = 11–1

La nueva tabla ha quedado, por lo tanto, y una vez clasificada, de la forma siguiente:

Num.unos	Implicante primo
0	–000
0	0–00
0	00–0
1	–010
1	–100
1	001–
1	1–00
1	10–0
1	100–
2	–011
2	0–11
2	1–01
2	10–1
2	101–
2	110–
3	–111
3	1–11
3	11–1

En el caso de que tras todas las comparaciones hubiera quedado algún elemento de la tabla base que no haya podido ser fusionado con ningún otro, entonces también se copian todos ellos a la nueva tabla. En este ejemplo concreto todos los elementos se han fusionado con otros una o varias veces, por lo que no hay que añadir ninguno adicional a la tabla nueva.

Ya se puede ver en el ejemplo que eran doce los IPs originales, pero ha habido que hacer 30 comparaciones en esta primera iteración del paso y se han generado 18 elementos, en los que se sigue añadiendo el número de unos que contiene cada elemento.

La nueva tabla se ordena de nuevo por el número de unos y el valor del IP, se eliminan los elementos duplicados que pudiera haber (en esta primera iteración del paso no hay ninguno) y se copia la nueva tabla resultante a la tabla base, para repetir el proceso de este Paso 3 tantas veces como sea necesario hasta que no se haya conseguido reducir ni un solo IP más.

La nueva tanda de comparaciones es tediosa de escribir (esta vez son 72 comparaciones), así que sólo escribiré la nueva tabla de los IPs resultantes. Habrá que creerme, o mejor dicho, al programa que lo hace. Nuevamente, en esta iteración todos los elementos de la tabla han sido fusionados con algún otro, por lo que no se debe copiar ningún elemento adicional no tratado.

El resultado de esta segunda iteración del Paso 3 es que en la nueva tabla quedan 14 IPs:

Num.unos	Implicante primo
0	–0–0
0	—00
0	—00
0	–0–0
1	–01–
1	–01–
1	1–0–
1	10—
1	10—
1	1–0–
2	—11
2	—11
2	1—1
2	1—1

Pero en realidad son sólo 7 IPs diferentes, que en este caso se repiten dos veces cada uno. Una vez ordenada la tabla y eliminados los duplicados, la tabla queda así:

Num.unos	Implicante primo
0	—00
0	–0–0
1	–01–
1	1–0–
1	10—
2	—11
2	1—1

La tercera iteración del paso realiza 12 comparaciones pero, como puede verse fácilmente, ninguna encuentra que los dos elementos comparados sólo difieran en sólo una posición, por lo que no hay más fusiones y, por lo tanto, ésta es la tabla de implicantes primos definitiva.

Este paso es terrible en uso de recursos en cuanto el número de variables individuales crece. Entre las muchas pruebas que he hecho, usé una expresión de 10 variables individuales cuya forma canónica tenía 1.006 unos de los 1.024 (2^{10}) posibles.

Según la información existente en la red, el número de implicantes primos a calcular para una función de *n* variables puede llegar a ser de 3^n dividido por la raíz cuadrada de *n*, es decir, según esta fórmula en esta función de 10 variables podemos esperar que el número de implicantes primos llegue a ser de hasta 18.673. Veamos ahora la realidad de esta función:

Como el valor 1 se da en 1.006 ocasiones en la forma canónica de la expresión, la expresión tiene 1.006 *minterms*, y por ello la carga inicial de implicantes primos cargó exactamente 1.006 implicantes primos. Para tratarla fueron necesarias diez iteraciones del bucle de este Paso 3, y la tabla de implicantes primos llegó a tener en la cuarta iteración más de 46.000 implicantes primos. En concreto, la dimensión de la tabla de implicantes primos en cada iteración fue de 1.006, 4.964, 21.854, 42.180, 46.496, 31.900, 13.884, 3.724, 560, 40 y, finalmente, la tabla de implicantes primos simplificados resultado de este Paso 3 tenía solamente 8 implicantes primos. Pero para llegar a esos magros ocho implicantes primos finales se han tenido que realizar por el camino muchos millones de operaciones. El método de ordenación utilizado debe ser Quicksort; otro método daría resultados desastrosos. Y, por cierto, el número máximo de implicantes primos que se requieren en este caso para completar el Paso 3 en este caso, 46.496, es dos veces y media mayor que el supuesto límite superior teórico establecido de dicho número.

En mis búsquedas por la Red he encontrado algunas publicaciones donde se explican diversas estrategias para mejorar el rendimiento de este Paso 3 y que consuma menos recursos, alguna basada en comparar los elementos de la tabla sólo cuando se cumplan unas ciertas restricciones para no tratar elementos que de antemano se determine que no van a fusionarse, o bien, en otro caso, en aplicar matrices de tamaño 3^k para reducir el número de comparaciones, y con toda probabilidad existen más aproximaciones que yo no he encontrado para reducir el coste de este Paso 3. Sin embargo, aunque no he implementado ninguna de estas ideas en mis programas, mi instinto de viejo informático me hace pensar que el coste de determinar si se compara o no un elemento con otro debe ser casi el mismo que el de hacer físicamente la comparación. Lo cierto es que en la práctica totalidad de publicaciones sobre el algoritmo QM no se cita ninguna de estas mejoras.

Lo que sí se cita de forma casi universal es que cuando se fusionan dos IPs para dar origen a un nuevo IP se debe apuntar cuidadosamente qué *minterms* dan origen a este nuevo IP fusionado, para posibilitar o simplificar el paso siguiente. Por ejemplo, al fusionar el IP 0000 con el 0100 para originar el nuevo IP 0–00, se debe anotar asociado a este IP 0–00 de dónde viene: de los *minterms* 0000 y 0100. Lo normal es que se guarden estos números en notación decimal, 0 y 4, supongo que para hacerlo más sencillo a la vista que en binario. En iteraciones sucesivas, por ejemplo cuando se fusiona 0–00 con 1–00 para dar origen a —00 se debe anotar en dicho IP los cuatro *minterms* que cubre: 0000 y 0111, herencia de 0–00, y 1000 y 1100, herencia de 1–00. Según se van haciendo más y más fusiones, el tamaño de estas listas de *minterms* crece y crece... listas de un número variable de elementos que se vuelven muy costosas de mantener, tanto por su configuración variable como por el tamaño de cientos de miles o millones de elementos que puede alcanzar la suma de las diferentes listas.

Sin embargo, **esta costosa técnica de llevar las listas de *minterms* no sirve para nada**, como veremos en seguida, y no debe implementarse de ninguna forma en el algoritmo.

Paso 4 - Carga de la tabla de cobertura de *minterms*

En base a la lista de implicantes primos simplificados por el paso anterior se rellena una tabla bidimensional en la que, en columnas, tendremos una por cada *minterm* que tenga la expresión original, es decir, una por cada uno de la forma canónica de la expresión, doce en nuestro ejemplo, y en filas tendremos una por cada implicante primo simplificado resultante. En las celdas de la tabla se marca con una X cuando el implicante primo de la fila cubre al *minterm* de la columna. Para saber qué *minterms* cubre cada IP basta con desarrollar el propio IP, sustituyendo cada guión por sus dos posibles valores 0 y 1.

Así, un IP como el --00, que tiene dos guiones, cubre 4 *minterms* (2^2), que son: 0000, 0100, 1000 y 1100, resultado de sustituir ambos guiones por las 4 combinaciones posibles de 0 y 1. Con esta sencilla técnica, como avanzaba en el Paso anterior, se evitan las interminables listas de *minterms* cubiertos por cada implicante primo, pues ésta es su única utilidad: conocer qué *minterms* cubre cada IP para marcar en consecuencia las equis en la tabla de cobertura.

Es importante añadir a la tabla las etiquetas, tanto en filas como en columnas, dado que, como veremos, en los siguientes pasos se irán eliminando tanto filas como columnas de la tabla.

En el ejemplo que hemos seguido hasta ahora, con siete implicantes primos finales y doce *minterms*, y una vez ordenados los IPs, la estructura de la tabla de cobertura quedaría así:

IP	0000	0010	0011	0100	0111	1000	1001	1010	1011	1100	1101	1111
—00	X			X		X				X		
—11			X		X				X			X
–0–0	X	X				X		X				
–01–		X	X					X	X			
1—1							X		X		X	X
1–0–						X	X			X	X	
10—						X	X	X	X			

Los pasos siguientes intentarán cubrir todos los *minterms*, las columnas, utilizando para ello el menor número posible de implicantes primos, aunque para nuestro objetivo final de optimizar el código fuente en realidad lo que más interesa es usar el menor número posible de variables individuales, es decir, el número de caracteres que no sean guiones de los IPs seleccionados. Recordemos que cada guión de un implicante primo representa una variable que ha sido eliminada, mientras que cada cero o uno de un implicante primo representa a una variable individual, negada o no, es decir, una condición a comprobar en la función lógica; cuantas menos haya de éstas, más rápida será la evaluación de la expresión total.

Es claro que en este caso el tamaño, fundamentalmente el número de IPs de la tabla de cobertura recién creada, importa. Cuando crece por encima de ciertos valores se necesita un enorme consumo de recursos para solventarlo. Es por eso que los dos siguientes pasos del algoritmo lo que intentan es reducir en lo posible el número de elementos de la tabla, tanto filas como columnas, para simplificar así la obtención de la solución final.

Paso 5 - Detección de Implicantes Primos Esenciales

Si nos fijamos en la tabla anterior nos damos cuenta de que hay algunos *minterms* que tienen una única X en su columna. Es el caso, por ejemplo, del *minterm* 0100: sólo el IP —00 lo cubre, así que obligatoriamente dicho IP tiene que estar en la lista final de IPs, pues en otro caso quedaría sin cubrir dicho *minterm* 0100. A estos IPs se los denomina *Implicantes Primos Esenciales*, y deben todos ellos formar parte obligatoriamente de la lista final de IPs buscada.

Una vez seleccionado un implicante primo esencial, se lleva a la tabla de IPs seleccionados, se elimina su fila de la tabla y también se eliminan todas las columnas en las que ese IP tenga una X, dado que los *minterms* correspondientes ya están cubiertos por ese IP seleccionado.

Bien: el IP —00 es el único que cubre el *minterm* 0100, luego es esencial y se selecciona. Se lleva a la tabla de IPs seleccionados, se elimina la fila del propio implicante primo y también las columnas de los *minterms* que cubre, y las tablas quedan así:

IP	0010	0011	0111	1001	1010	1011	1101	1111
10—				X	X	X		
—11		X	X			X		X
–0–0	X				X			
–01–	X	X			X	X		
1—1				X		X	X	X
1–0–				X			X	

IPs Seleccionados
—00

Un pequeño truco aquí es desplazar el último IP de la tabla a la posición del IP eliminado en vez de mover todas las filas, ahorrando algo de tiempo de CPU. Por eso el IP 10— aparece ahora en la primera fila en vez de la séptima original. Sigamos:

Nuevamente el IP —11 es ahora el único que cubre el *minterm* 0111, por lo tanto es también esencial y debe ser seleccionado. Se elimina dicho implicante primo y las columnas de los *minterms* que cubre, y por lo tanto quedan ahora en la tabla de cobertura cinco IPs:

IP	0010	1001	1010	1101
10—		X	X	
1–0–		X		X
–0–0	X		X	
–01–	X		X	
1—1		X		X

IPs Seleccionados
—00
—11

Ya no hay más implicantes primos esenciales, todos los *minterms* están cubiertos por al menos dos implicantes primos. Y todos los IPs esenciales localizados hasta ahora han sido trasladados a la tabla de IPs seleccionados.

Por supuesto es posible, y ocurre con frecuencia que, dada una cierta forma canónica y llegados a este momento, no haya ningún IP esencial que pueda ser eliminado; sin embargo, tal como comentaba en el Paso 1, el coste de realizar esta búsqueda de implicantes primos esenciales es muy pequeño comparado con la posible reducción de tamaño en la tabla de cobertura, que minorará significativamente el tiempo necesario para ejecutar el resto de pasos.

Paso 6 – Eliminación de Implicantes Primos redundantes

En este Paso se intenta eliminar de la tabla de cobertura aquellos IPs restantes que sean redundantes con otros implicantes primos de la propia tabla. Un IP IP_1 es redundante con otro IP_2 cuando dicho IP_2 cubre al menos todas las columnas (*minterms*) que cubre IP_1, pudiendo IP_2 cubrir, además algunos otros adicionales.

La justificación es sencilla: supongamos que tenemos la tabla de cobertura siguiente, donde IP_1 cubre dos columnas (*minterms*), la columna 2 y la 3, mientras que IP_2 cubre las columnas 2, 3 y 4.

IP	1	2	3	4
IP_1		X	X	
IP_2		X	X	X

Aquí todos los *minterms* que cubre IP_1, 2 y 3, los cubre también IP_2, que adicionalmente cubre un *minterm* más, el 4. Por lo tanto, es seguro que seleccionando IP_2 se cubren eficazmente todas las columnas que cubre IP_1. La consecuencia es que el implicante primo IP_1 es redundante con IP_2, que *lo domina*, según la terminología usada en la literatura, y por lo tanto se puede suprimir tranquilamente el implicante primo IP_1 de la tabla de cobertura.

Esta eliminación sólo es posible si en el Paso 5 se hubiera encontrado algún implicante primo esencial; si no hubiera ninguno entonces es seguro, debido al método de fusión de IPs seguido en el Paso 3, que no hay IPs redundantes y por tanto que ningún IP cubre a ningún otro.

Además, una consecuencia de esta posible eliminación de IPs redundantes es que ciertos IPs que antes no lo eran se conviertan en esenciales, denominados *implicantes primos esenciales secundarios*, pero que por muy secundarios que sean son igual de esenciales que los anteriores.

Esto quiere decir que la forma correcta de implementar los Paso 5 y 6 es en realidad juntos, en una iteración que se repite hasta que no se hayan encontrado más IPs esenciales, de la forma:

Paso conjunto 5-6:

1) Ejecutar el proceso de búsqueda y eliminación de IPs esenciales (Paso 5).
2) Si no se ha encontrado ningún IP esencial, terminar el Paso conjunto 5-6.
3) Ejecutar el proceso de búsqueda y eliminación de IPs redundantes (Paso 6).
4) En el caso de que se haya eliminado al menos un IP redundante, volver al paso 1; en el caso de no haber eliminado ninguno, terminar el Paso conjunto 5-6.
5) Terminar también en el caso de que no queden ya columnas en la tabla de cobertura, por haber sido cubiertas todas ellas por los implicantes esenciales seleccionados.

Volvamos ahora a nuestro ejemplo, donde restaban cinco IPs y cuatro *minterms* en la tabla de cobertura, y donde habían sido seleccionados dos IPs esenciales hasta el momento. Ésta era la situación:

IP	0010	1001	1010	1101
10—		X	X	
1–0–		X		X
–0–0	X		X	
–01–	X		X	
1—1		X		X

IPs Seleccionados
—00
—11

Al buscar IPs redundantes vemos que el IP 1–0– cubre exactamente las mismas columnas que el IP 1—1, los *minterms* 1001 y 1101. Puede eliminarse uno cualquiera de ellos, dado que en este punto son idénticos, pues cubren las mismas columnas. Eliminemos el último, el 1—1.

Por otra parte, el IP –0–0 cubre a su vez las mismas columnas que el IP –01–, que es también eliminado.

Como consecuencia de ambas eliminaciones la tabla de cobertura de IPs queda ahora así:

IP	0010	1001	1010	1101
10—		X	X	
1–0–		X		X
–0–0	X		X	

Ahora los IPs 1–0– y –0–0 se han convertido en esenciales secundarios, pues son los únicos que cubren a los *minterms* 1101 y 0010, respectivamente.

Por tanto, se seleccionan ambos y se llevan a la tabla de IPs seleccionados, que queda así:

IPs Seleccionados
—00
—11
1–0–
–0–0

Y ahora, al eliminar estos dos últimos implicantes primos, todos los *minterms* han resultado cubiertos. Ya no quedan columnas en la tabla. En ella queda un solo IP, el 10—, que es irrelevante y se ignora. No es necesario proseguir con el resto del algoritmo, lo que sí sería necesario en el caso habitual de que quede en la tabla de cobertura un cierto número de *minterms* sin cubrir por un cierto conjunto de IPs restantes. Habrá entonces que seleccionar de alguna manera un conjunto mínimo de IPs que cubran todos los *minterms* restantes.

Qué hacer en el caso de que la tabla de cobertura sí que tenga contenido tras la ejecución de los Pasos 5 y 6, lo que es el caso normal, es decir, lo que será el **Paso 7** del algoritmo de Quine-McClusky, **Búsqueda de la Solución Óptima**, será tratado más adelante, en un próximo artículo.

Los Pasos 2 a 6 que he explicado hasta ahora son, con mínimos ajustes, los mismos pasos descritos en la literatura sobre el algoritmo QM, aunque numerados de 1 a 5, dado que el Paso 1 tal como yo lo he definido no existe o al menos no lo he encontrado en la literatura.

Ahora, antes de seguir el desarrollo del resto del algoritmo debo comentar algunas de las conclusiones y hechos que he encontrado en mis pruebas e investigaciones. Para ello necesito adelantar la definición del **Paso 8, Obtención de la Solución Final Óptima**. De hecho, en casos como el del ejemplo que hemos seguido hasta aquí no es necesario ejecutar el antes citado Paso 7, dado que todos los *minterms* han sido cubiertos con los IPs esenciales, por lo que el devenir natural del algoritmo pasaría en este caso concreto por la ejecución directa de dicho Paso 8, que paso a definir inmediatamente.

Paso 8 – Obtención de la expresión óptima final

En base a los implicantes primos finalmente seleccionados para formar parte de la solución final, para obtener la expresión resultante como suma de productos basta con representar, multiplicando (AND) entre ellas, las variables referenciadas por cada implicante primo de la tabla final, en afirmativo si hay un 1, y en negativo si hay un cero, y obviando los guiones que no dan origen a ninguna variable, y luego sumar (OR) todos los términos resultantes.

Una vez obtenida de esta forma la expresión final, hay que añadir a la expresión todas las variables implicantes encontradas en el Paso 1 – Detección de Variables Implicantes. Estas variables implicantes se añaden a la expresión en el orden inverso al que fueron detectados, conectándolas con la expresión existente sumando/multiplicando a la totalidad de la expresión existente, según sea '+' o '*', respectivamente, el operador almacenado junto a la variable implicante en su tabla correspondiente.

En nuestro ejemplo, recordemos que la primera posición de los IPs corresponde a la variable C2, la segunda, a C3, la tercera a C4 y la cuarta y última a C5, ya que la variable C1 era implicante y había sido eliminada de la forma canónica en el Paso 1. Como los IPs finalmente seleccionados son cuatro, cuatro serán los productos obtenidos, y como todos ellos tienen dos bits válidos y dos guiones, todos los productos tendrán dos variables.

En el primer IP seleccionado, que es —00, la primera posición, correspondiente a C2, es un guión, luego se ignora, lo mismo que el segundo guión, correspondiente a C3. El tercer bit, el de C4, es un cero, luego la variable está negada, es N4, y lo mismo con el cuarto bit, el de C5, que al ser otro cero está negada también, luego es N5. El término generado por —00 es, por tanto, N4*N5. Los términos resultantes son los siguientes:

IPs Seleccionados	Términos generados
—00	**N4*N5**
—11	**C4*C5**
1–0–	**C2*N4**
–0–0	**N3*N5**

Los cuatro términos son, pues: N4*N5, C4*C5, C2*N4 y N3*N5 y, como paso final, todos estos productos se suman entre sí para dar origen a la expresión definitiva:

N4*N5+C4*C5+C2*N4+N3*N5

Aquí se quedan la práctica totalidad de publicaciones sobre el algoritmo QM, indicando que la expresión así obtenida es la mínima posible. Sin embargo, es obvio que **una suma de productos puede reducirse de tamaño simplemente aplicando la propiedad distributiva**, es decir, sacando como factores comunes aquellas variables que estén en diferentes términos, cosa que es habitual que se dé en la expresión obtenida, una Suma de Productos.

Así, esta expresión de 8 variables individuales es susceptible de ser reducida un poco más sacando factor común, en este caso a N5 (también se podría hacer con N4, pero no con ambas a la vez). Reordenando, esta expresión mínima final es:

C2*N4+N5*(N3+N4)+C4*C5

Expresión final que consta de tan solo 7 variables individuales, y que, ahora sí, ya no es posible reducir más.

Recordemos, sin embargo, que aún faltan por incluir las variables implicantes detectadas y eliminadas en el Paso 1. Como paso final, éstas deben ser incluidas en la fórmula final en el orden inverso a su eliminación (LIFO), conectadas con la expresión final por el operador que acompaña a dichas variables implicantes en su tabla correspondiente.

En nuestro ejemplo había una sola variable implicante, C1, con el operador '+', variable que hay que sumar a la totalidad de la expresión encontrada hasta el momento, por lo que la expresión definitiva es: **C1+C2*N4+N5*(N3+N4)+C4*C5.**

Ésta sí que es, definitivamente, la expresión mínima absoluta correspondiente a la forma canónica de ejemplo, 10111001111111011111111111111111.

O al menos *eso es lo que dicta la teoría unánimemente aceptada*.

Pero no es así, porque, atención: **ésta no es la expresión mínima equivalente a la forma canónica dada**. Para obtener de una vez la que sí que es la expresión mínima absoluta de la forma canónica dada en el ejemplo, 10111001111111011111111111111111, es preciso utilizar el mismo algoritmo QM, pero *de una forma ligeramente distinta*.

Pero todo esto lo veremos en el próximo artículo.