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

(IV)

Macluskey, 2023

En los artículos anteriores de esta serie había explicado cómo es el algoritmo de Quine-McClusky y de qué modo se puede incrementar su eficacia, proporcionando siempre y en toda ocasión la expresión mínima correspondiente a la forma canónica dada, por el método de calcular la expresión resultante tratando, por un lado, los unos, los *minterms*, y por el otro tratando los ceros, los *maxterms*, y complementando la expresión resultante, para quedarse por fin con la expresión más pequeña de las dos.

Pero esa definición todavía no está completa: falta por describir el penúltimo paso del algoritmo que, dadas las características de los ejemplos utilizados en esos artículos, hasta ahora no ha hecho falta, pero en el caso general ya lo creo que hace falta. El caso es que había dejado su definición para más adelante... ahora es el momento.

En este artículo y en el siguiente voy a entrar en detalle en la descripción de este Paso 7 - Búsqueda de la Solución Óptima, que es el que, como pronto veréis, de verdad consume una cantidad indecente de recursos informáticos porque, como decía en el artículo anterior, la minimización de expresiones lógicas es un problema *NP-completo* y es precisamente en este Paso 7 donde la *NP-Completitud* se muestra en todo su esplendor.

Paso 7 – Búsqueda de la Solución Óptima

Recordemos que tras la ejecución del paso 6 (en realidad, del paso conjunto 5-6) puede que resten en la tabla de cobertura algunos (o muchos) implicantes primos que cubren un determinado número de columnas (*minterms*), al no ser ninguno de ellos esencial ni estar dominado por otro. Es normal que sí que resten una serie de IPs en la tabla, que hay que tratar para obtener la función mínima buscada, seleccionando los implicantes primos adecuados e incluyéndolos en la tabla de IPs seleccionados.

Antes de entrar en harina, ¿cómo son estos implicantes primos restantes, qué características tiene la tabla de cobertura resultante en este momento? En base al resultado de los pasos anteriores podemos hacer algunas afirmaciones útiles sobre ella:

1) Todas las columnas (*minterms*) de la tabla tienen al menos dos equis.
   Efectivamente, si alguna columna tuviera una sola equis, entonces el IP correspondiente sería esencial, habría sido seleccionado en el Paso 5 y ya no estaría en la tabla.

2) Todos los IPs (filas) de la tabla tienen al menos dos equis.
   Si hubiera alguno con una sola equis, y dado que según el punto anterior todas las columnas tienen al menos dos equis marcadas, entonces ese IP sería redundante con, al menos, el otro IP que tiene también marcada esa misma columna, y por tanto habría sido eliminado por el Paso 6.

3) Hay al menos tres IPs (filas) en la tabla de cobertura.
   Si quedaran solamente dos IPs, entonces o son iguales o uno de ellos es redundante con el otro, debido a los puntos 1 y 2 anteriores, y uno de ellos, el redundante, habría sido eliminado por el Paso 6.

4) Hay al menos tres *minterms* (columnas) en la tabla.
   Si hubiera sólo dos, entonces todos los IPS serían iguales, dado que todos los IPs tienen que tener al menos dos equis cada uno.

De esta forma la dimensión mínima de la tabla de cobertura en este punto del algoritmo QM es de 3 filas (IPs) y 3 columnas (*minterms*), y sería algo parecido a esto:

IP	m1	m2	m3
IP1		X	X
IP2	X	X	
IP3	X		X

Antes de comenzar a debatir las complejidades de este Paso 7 voy a definir un ejemplo que ayudará a fijar las ideas. Se trata de la forma canónica **1101111111111011**, de 16 bits y, por lo tanto, con cuatro variables involucradas, que son C1, C2, C3 y C4. La forma canónica tiene catorce unos y dos ceros; me centraré en lo que sigue en los resultados del algoritmo QM original, es decir, tratando los *minterms* (los unos). Advierto aquí que si hubiera elegido en su lugar la forma canónica *espejo*, la 0010000000000100, hubiera llegado a idénticas conclusiones tratando los *maxterms*, los ceros. Pero aquí lo mejor para nuestros intereses es tratar los unos, los *minterms*, tal como preconiza el algoritmo de Quine-McClusky habitual.

En el Paso 1 no se detecta ninguna variable implicante. En consecuencia no se cambia la forma canónica para la ejecución del resto de pasos, ni se incluye ninguna variable en la tabla de variables implicantes, que por lo tanto queda vacía.

Los *minterms* originales que se cargan en la tabla de IPs en el Paso 2 son, pues, catorce:
0000, 0001, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1110 y 1111.
Sólo faltan el 0010 y el 1101, que corresponden a los dos únicos ceros de la forma canónica.

Ahora se ejecutaría el Paso 3, Cálculo de los Implicantes Primos. Vais a tener que creerme, o mejor, al programa que gentilmente calcula estas cosas: este paso deja finalmente doce IPs que, una vez cargados en la tabla de cobertura según preconiza el Paso 4, quedan así:

	***Minterms* de la expresión**													
IPs	**0000**	**0001**	**0011**	**0100**	**0101**	**0110**	**0111**	**1000**	**1001**	**1010**	**1011**	**1100**	**1110**	**1111**
--00	X			X				X				X		
--11			X				X				X			X
-0-1		X	X						X		X			
-00-	X	X						X	X					
-1-0				X		X						X	X	
-11-						X	X						X	X
0--1		X	X		X		X							
0-0-	X	X		X	X									
01--				X	X	X	X							
1--0								X		X		X	X	
1-1-										X	X		X	X
10--								X	X	X	X			

Bien, sobre esta tabla de cobertura de 12 IPs (filas) y 14 *minterms* (columnas) se procesa el Paso 5, Selección de Implicantes Primos Esenciales... ¡No hay ninguno! Y como consecuencia el Paso 6 tampoco encuentra ningún IP redundante. De hecho, si no se ha seleccionado y eliminado ningún implicante primo esencial, y dado el método de construcción de la tabla, es imposible que haya IPs redundantes.

En una palabra, ésta es definitivamente la tabla de cobertura que tiene que tratar este Paso 7.

El objetivo del Paso 7 es encontrar una combinación de IPs tales que, en primer lugar, queden cubiertas todas las columnas de la tabla (necesario para asegurar que la fórmula encontrada es correcta, pues cubre todos los *minterms* de la forma canónica) y, luego, de todas las posibles combinaciones de IPs que cumplen esta condición, encontrar la que contenga un menor número de variables individuales, es decir, donde la suma del número de ceros y unos de los IPs de la combinación sea mínima, la menor de todas ellas.

A cualquiera se le ocurre un método infalible para encontrar esta solución: se generan todas las combinaciones de IPs posibles y se verifica para cada una de ellas si efectivamente todas las columnas han sido cubiertas por la combinación. Si quedan columnas por cubrir se descarta la combinación, pero si todas las columnas están cubiertas se cuentan las variables totales de dicha solución; nos quedamos con la combinación que menor número de variables tiene, y listo.

Un método éste, el de **fuerza bruta**, sencillo y eficaz... siempre que el número de IPs de la tabla esté en un cierto rango. En el caso del ejemplo son 12 IPs, por lo tanto las combinaciones a probar son 4.095 (2^{12}-1), desde 000000000001 hasta 111111111111, donde un 1 en la posición *i-ésima* significa que el IP correspondiente, el que a su vez está en la fila *i-ésima* de la tabla de cobertura, forma parte de la combinación, y 0, que no forma parte: el número de combinaciones posibles con n IPs es, obviamente, 2^n, y como la combinación de todo ceros, es decir, no seleccionar ningún IP, no tiene sentido, el número de combinaciones a probar es de 2^n-1.

Cierto que hay algunos trucos que permiten que ciertas combinaciones puedan ser ignoradas de antemano, como las que sólo tienen un único uno (es imposible que con un solo IP se cubran todas las columnas; si eso pasara no habríamos llegado a este Paso 7), o bien aquellas que tengan más unos que la mejor combinación seleccionada hasta el momento, etc., pero esto no varía en demasía el número de combinaciones a probar, que crece exponencialmente con el número de IPs de la tabla.

En nuestro ejemplo el coste de realizar 4.095 comprobaciones es, seguramente, asumible, pero ¿qué ocurre si en la tabla de cobertura hay, por ejemplo, 100 IPs? Pues que comprobar 2^{100} combinaciones, es decir, del orden de 10^{30}, está completamente fuera del alcance no sólo de mí, sino de cualquier ordenador que yo conozca. Esta cifra de 100 IPs es bastante habitual en cuanto la forma canónica tiene 256 o más bits; para formas canónicas de 1.024 bits lo normal es que la tabla de cobertura tenga al menos 150 IPs y 120 columnas; yo he detectado alguna forma canónica de 1.024 bits que genera en este punto una gigantesca tabla de cobertura de 556 filas y 464 columnas. Tratar esta monstruosidad por fuerza bruta es algo completamente fuera de las posibilidades de cálculo de todo el mundo. Y eso con solamente 1.024 bits en la forma canónica; imaginad lo que ocurre con formas canónicas mayores.

Por lo tanto, el método de fuerza bruta queda descartado como método factible en cuanto la tabla tiene más allá de quizás 20 ó 25 filas. Y como consecuencia hay que buscar métodos alternativos que encuentren la combinación ideal en un tiempo razonable, o por lo menos que sea en tiempo polinómico, no exponencial. Y en este caso la "*combinación ideal*" no tiene por qué ser la mínima absoluta, sino una que se le acerque lo suficiente: ya hemos visto que asegurar que se ha encontrado la solución mínima absoluta es prácticamente imposible en cuanto el número de IPs de la tabla supera una cierta, y relativamente modesta, cifra.

En la literatura sobre el algoritmo de Quine-McClusky que he revisado siempre se citan en este punto dos posibles métodos, aparte de la fuerza bruta, para resolver el problema de encontrar la combinación de implicantes primos que, cubriendo todas las columnas (los *minterms*), sea mínima, en el sentido de o bien necesitar la menor cantidad posible de IPs, o bien de involucrar la menor cantidad posible de variables individuales en la solución. Estos dos métodos son el *branch&bound* y el método de Petrick. Veamos cómo son cada uno de ellos.

El algoritmo Branch&Bound (ramificación y poda, en español) es un algoritmo de programación lineal entera, una variante del backtracking donde a partir de una combinación inicial válida se va construyendo un árbol de soluciones en el que cada rama conduce a una posible solución más evolucionada que la actual. El algoritmo elimina en cada paso aquellas ramas que se alejan de la solución óptima, para progresar sólo en aquellos nodos que van paulatinamente mejorando la solución, hasta llegar a un punto donde o no es posible mejorarla más o bien se alcanza un cierto límite de recursos prefijado.

Es una técnica similar a la *poda alfa-beta* tan utilizada en los programas que juegan al ajedrez, donde primero se elabora el árbol de jugadas posibles, se evalúan y entonces se podan del árbol las jugadas malas, para continuar expandiendo el árbol solamente con las jugadas prometedoras hasta alcanzar el límite fijado.

El algoritmo *branch&bound* corre en tiempo polinómico y, al igual que todos los algoritmos de programación lineal, no siempre devuelve la solución óptima, sino una que es "lo suficientemente óptima". Eso sí, por mucho que necesite un tiempo polinómico, cuando la tabla de cobertura tiene muchos implicantes primos el tiempo necesario para resolverlo se incrementa mucho. Muchísimo. Y además, cuanto más grande es dicha tabla de cobertura más dificultades tiene para encontrar la solución óptima.

Dado que hay mucha información en la Red sobre este tipo de algoritmos de programación lineal, herederos en su mayoría del venerable Simplex, no voy a incidir más en él, dado que, como se verá, hay otros métodos más eficientes y sencillos para obtener resultados similares.

En cuanto al **método de Petrick**, se trata en realidad de un algoritmo cuya única función es convertir una función booleana expresada como un producto de sumas, es decir, en Forma Normal Conjuntiva, en una equivalente expresada como una suma de productos, y por tanto en Forma Normal Disyuntiva. Nada más. Y nada menos.

Se basa en una inteligente interpretación del contenido de la tabla de cobertura de implicantes primos, que voy a explicar en detalle: a mí me ha costado lo suyo entender bien los intríngulis del método, así que intentaré que a quienes lean estas líneas no les cueste tanto como a mí me costó. Veamos primero el artificio booleano en que se basa, es muy interesante.

Pongamos antes que nada nombres a las diferentes filas y columnas de la tabla de cobertura. Denominaremos las columnas (los *minterms*) como M1, M2... M*n*. En la tabla del ejemplo antes citado hay 14 columnas, luego sus nombres serán M1...M14. M1 corresponde al *minterm* 0000, M2 al 0001, y así hasta M14, que corresponde al *minterm* 1111. En cuanto a los implicantes primos, los denominaremos IP1, IP2,... IP*k*. En el ejemplo hay 12 IPs, cuyos nombres serán IP1, el correspondiente a --00; IP2, a --11; hasta IP12, que corresponde al último de ellos, 10--.

En primer lugar, fijémonos bien en cuál es el objetivo buscado con el tratamiento de la tabla de cobertura anterior. Se trata de encontrar una combinación de implicantes primos tal que:

1) Cubra la totalidad de *minterms*, es decir, todas las columnas de la tabla, y
2) Que, además, sea mínima, es decir, que contenga el menor número de implicantes primos posible.

Vamos a fijarnos en la primera de las condiciones necesarias, que todas las columnas estén cubiertas por la combinación dada. Llamemos *Z* a esta función lógica. ¿Qué se necesita para que se cumpla *Z*, es decir, que su valor de verdad sea uno? Es de Perogrullo: ha de cumplirse que todas y cada una de las columnas M*i* queden cubiertas. Siendo F*i* la función de cobertura de la columna M*i*, la *i-ésima*, entonces debe ocurrir que todas las funciones F*i* deben cumplirse, es decir, tener un valor de verdad de uno, para que la función global *Z* se cumpla a su vez.

O sea: *Z* = (la columna M1 debe estar cubierta) **Y** (la columna M2 debe estar cubierta) **Y** ...
... **Y** (la columna M*n* debe estar cubierta)

Como hemos llamado F*i* a la función "la columna M*i* debe estar cubierta", entonces la función *Z* es, naturalmente, un producto booleano de todas las funciones F*i* referidas a las columnas de la tabla. Es decir, *Z*=F1*F2*....*F*n*. El valor de verdad de *Z* es uno si y sólo si los valores de verdad de todas las funciones F*i* son uno a su vez, es decir, si todas las columnas están cubiertas.

Fijémonos ahora en cómo son cada una de estas funciones F*i*. Para que la columna *i-ésima* esté cubierta en la combinación dada es preciso que al menos uno de los distintos implicantes primos IP*k* que la cubren esté incluido en la solución, es decir, que al menos uno de los IPs que tienen una X en la intersección con dicha columna esté incluido en la solución. Puede haber en dicha solución más de un IP que cubra la columna , pero basta con que solamente uno de ellos esté incluido en la solución para que la columna *i-ésima* esté cubierta, y eso se representa, claro está, con una suma booleana.

Si la columna *i* es cubierta por *k* implicantes primos distintos, es decir, hay *k* equis en la columna *i-ésima*, para que F*i* se cumpla debe cumplirse que:
(el IP1 esté seleccionado en la solución) **O** (el IP2 esté seleccionado en la solución) **O** ...
... **O** (el IP*k* esté seleccionado en la solución), es decir, la función (IP1+IP2+...IP*k*).

Éste es, por lo tanto, el contenido de la función F*i* antes definida. Sustituyendo estos valores de F*i* en la expresión anterior, *Z*=F1*F2*....F*n*, lo que resulta es un producto de sumas, una expresión en Forma Normal Conjuntiva que expresa las condiciones necesarias para que se cumpla la condición necesaria de que todas las columnas de la tabla estén cubiertas.

Vale, ya sé que todo esto no es fácilmente comprensible. Recurramos al ejemplo anterior a ver si ayuda a comprender la base de todo esto. Miremos la tabla de cobertura del ejemplo. Para que las 14 columnas de la tabla estén cubiertas es preciso que todas y cada una de ellas lo estén. Comencemos por la primera de ellas, la columna M1 correspondiente al *minterm* 0000. Para que resulte cubierta, la combinación de IPs elegida debe obligatoriamente incluir al menos a uno de los tres IPs que cubren esta columna: el IP1 (--00), el IP4 (-00-) ó el IP8 (0-0-). Son ellos tres los únicos que tienen una X en la intersección con la primera columna.

Por lo tanto, podemos definir esta función F1 como (IP1+IP4+IP8). Análogamente podemos definir la función F2, correspondiente a la columna M2, como (IP3+IP4+IP7+IP8), pues estos son los cuatro IPs que tienen una equis en la columna M2. F3 será (IP2+IP3+IP7), y así hasta llegar a F14, que será (IP2+IP6+IP11). Finalmente todas estas funciones se combinan entre sí multiplicándose entre ellas mediante productos lógicos, dando origen a la larguísima expresión *Z* siguiente:

Z=(IP1+IP4+IP8)*(IP3+IP4+IP7+IP8)*(IP2+IP3+IP7)*(IP1+IP5+IP8+IP9)*
(IP7+IP8+IP9)*(IP5+IP6+IP9)*(IP2+IP6+IP7+IP9)*(IP1+IP4+IP10+IP12)*
(IP3+IP4+IP12)*(IP10+IP11+IP12)*(IP2+IP3+IP11+IP12)*(IP1+IP5+IP10)*
(IP5+IP6+IP10+IP11)*(IP2+IP6+IP11)

Son catorce términos, uno por columna, en este caso concreto con entre 3 y 4 IPs cada uno, dependiendo del número de equis que hay en cada columna.

¿Qué significa esta expresión? Sencillo: la expresión *Z* anterior, derivada de la propia tabla de cobertura de implicantes primos y *minterms*, contiene todas las posibles formas de cubrir todas las columnas de la tabla con los IPs de las filas.

Por ejemplo, una posible forma de cubrir todas las columnas sería elegir IP1, porque cubre la columna M1; IP4, porque cubre la columna M2; IP7, la columna 3 ... y por fin IP11, que cubre la columna M14. Por tanto, por ejemplo, la siguiente combinación de IPs:

{IP1, IP4, IP7, IP5, IP8, IP6, IP2, IP1, IP3, IP10, IP3, IP5, IP11, IP11} es una combinación válida que garantiza que todas las columnas están cubiertas. En ella hay IPs repetidos que obviamente deben eliminarse para dejar así la combinación:

{IP1, IP4, IP7, IP5, IP8, IP6, IP2, IP3, IP10, IP11}

Como ésta hay otras muchas combinaciones que cubren todas las columnas: basta con escoger uno cualquiera de los IPs de cada término para conformar una solución perfectamente válida.

Conocida esta *Z*, que asegura cuando su valor de verdad es uno que todos los *minterms* están cubiertos, ahora hay que seleccionar la combinación de IPs que, haciendo verdadera la expresión *Z*, contenga el menor número posible de implicantes primos. Y esto no es sencillo.

En el caso del ejemplo anterior existen 26.873.856 formas de crear combinaciones válidas de IPs: 8 de las columnas tienen 3 equis, mientras que las otras 6 tienen 4 equis; por lo tanto el número de combinaciones posibles es $3^8 \cdot 4^6 = 6561 \cdot 4096 = 26.873.856$. Aunque ese número se reducirá mucho debido a la existencia de numerosos casos con IPs repetidos, de todos modos los restantes siguen siendo muchos miles. En cualquier caso, encontrar la solución mínima no es obvio, ni mucho menos. ¿Cómo podemos encontrar de forma viable a partir de la expresión *Z* la o las combinaciones válidas que contengan el menor número posible de implicantes primos?

Pues utilizando otro inteligente artificio booleano basado en que, por construcción, la expresión *Z* es un Producto de Sumas, es decir, está en Forma Normal Conjuntiva. Las reglas del álgebra de Boole, en concreto la propiedad distributiva, permiten operar con dicha expresión hasta convertirla en otra equivalente cuya forma sea una Suma de Productos, es decir, ponerla en Forma Normal Disyuntiva. Se trata de una expresión lógica expresada de forma diferente, pero con la misma tabla de verdad, es decir, se trata en realidad la misma expresión lógica que *Z*. En un momento veremos para qué sirve esta sorprendente transformación.

Recordemos aquí, por un lado, que la Idempotencia indica que a*a=a y que a+a=a; y por otro, que la Ley de Absorción indica que a+a*b=a y que a*(a+b)=a. Ambas leyes booleanas se usarán en las transformaciones que siguen, además de la propiedad distributiva, en concreto (a+b)*(c+d)=a*c+a*d+b*c+b*d.

El método para transformar un Producto de Sumas en una Suma de Productos no puede ser más sencillo. Largo, sí, pero sencillo. Vamos a verlo con un pequeño ejemplo, distinto del utilizado hasta ahora, pero que por su sencillez permite comprender bien el proceso:

Sea *Z*=(IP1+IP2)*(IP2+IP3+IP5)*(IP2+IP4)

Es decir, hay cinco IPs en filas, de IP1 a IP5, y tres *minterms* en columnas, pues tres son los términos de *Z*, las sumas. En primer lugar aplicamos la propiedad distributiva a los dos primeros términos. Para ello se combinan de dos en dos todos los términos del primer término (IP1+IP2) con cada uno de los tres términos del segundo término (IP2+IP3+IP5):

(IP1+IP2)*(IP2+IP3+IP5) =
(IP1*IP2+IP1*IP3+IP1*IP5+IP2*IP2+IP2*IP3+IP2*IP5)

Aplicando ahora las reglas del álgebra de Boole en la expresión resultante del paso anterior, por Idempotencia queda que IP2*IP2=IP2, y por la Ley de Absorción, IP2+IP2*IP3=IP2, o también IP1*IP2+IP2=IP2, etc. Entonces, operando de esta forma en la expresión anterior, resulta la fórmula siguiente:

(IP1*IP3+IP1*IP5+IP2)

Éste es, pues, el resultado de multiplicar los dos primeros términos de *Z*; ahora este resultado obtenido hay que multiplicarlo a su vez por el tercer y último término de *Z*, (IP2+IP4).

(IP1*IP3+IP1*IP5+IP2)*(IP2+IP4) =
(IP1*IP3*IP2+IP1*IP3*IP4+IP1*IP5*IP2+IP1*IP5*IP4+IP2*IP2+IP2*IP4), y reduciendo:
(IP1*IP3*IP4+IP1*IP5*IP4+IP2)

Esto es largo y tedioso, pero sencillo: únicamente hemos aplicado las reglas del álgebra de Boole a rajatabla. Así, la expresión *Z* original, un Producto de Sumas obtenido en principio a partir de una cierta tabla de cobertura, que era *Z*=(IP1+IP2)*(IP2+IP3+IP5)*(IP2+IP4), es idéntica a la expresión recién encontrada, *Z*=IP1*IP3*IP4+IP1*IP5*IP4+IP2, es decir, ambas tienen la misma tabla de verdad, pero en cambio ahora la expresión tiene la forma de Suma de Productos.

¿Qué significa ahora esa nueva función *Z* expresada como Suma de Productos? Es sencillo: ahora las tornas han cambiado, pues para que la función *Z* se cumpla basta con que se cumpla uno solo de los términos, es decir, uno de los productos que están sumados en la expresión. Con que el valor de verdad de un único término sea uno, el valor de verdad de la función *Z* es uno. Traduciendo esta información a nuestro problema, lo que quiere decir esta función *Z* como Suma de Productos es lo siguiente: **Cada uno de los términos de esa Suma de Productos representa una combinación diferente de implicantes primos, combinaciones que cumplen todas ellas la condición necesaria de cubrir todas las columnas de la tabla, los *minterms***.

En el pequeño ejemplo tratado, la combinación de IPs {IP1, IP3, IP4} que forman el primer término cubre todas las columnas de la tabla y es, por lo tanto, válida. También lo es la combinación {IP1, IP5, IP4} formada por el segundo término, y lo es también la combinación {IP2}, que aunque tenga un solo implicante primo es tan válida como las demás. A partir de aquí, seleccionar la combinación mínima, la que contiene menos IPs, es trivial: es la combinación o combinaciones que tengan un menor número de IPs multiplicados dentro de los productos. En este caso es evidentemente IP2 la solución mínima de este miniejemplo, ejemplo que no tiene mucho sentido en el mundo real dado que el implicante primo IP2 está en todas las sumas, y eso quiere decir que cubre él solito todos los *minterms*... Valga exclusivamente como ejemplo para facilitar la comprensión del proceso, que, repito, no es sencillo de comprender.

Hasta aquí el método de Petrick que, como anticipaba, en realidad lo que hace es convertir una expresión en Forma Normal Conjuntiva (Producto de Sumas) en otra equivalente expresada en Forma Normal Disyuntiva (Suma de Productos). Hasta ahora lo he definido exclusivamente en base a las reglas del álgebra de Boole: a diferencia del resto de Pasos, no he descrito algoritmo informático alguno para hacerlo. Vamos ya a ver cómo puede implementarse esta conversión de Petrick de forma sencilla.

En primer lugar hay que asignar a cada implicante primo de la tabla un código que tendrá tantos bits como implicantes primos tenga la tabla de cobertura. En nuestro ejemplo de unas páginas más arriba son doce los implicantes primos de la tabla, luego ésa, doce, será precisamente la longitud en bits de los códigos de IP. Estos códigos de IPs se crean de la siguiente manera: el código del IP *i-ésimo* tendrá todos sus bits a cero excepto el bit *i-ésimo*, que será un uno. De ahí que la longitud del código sea el número de IPs de la tabla, pues todos estos códigos tienen un único bit a uno y todos los demás bits a cero.

En el ejemplo, el primer IP de la tabla de cobertura, el --00, tendrá un uno en su primer bit y ceros en el resto, es decir, 100000000000; el segundo, el --11, un uno en su segundo bit y ceros en el resto de bits, o sea, 01000000000, y así hasta el duodécimo y último, el 10--, cuyo código será 000000000001.

Así preparados los códigos, se necesitan para implementar el algoritmo tres tablas de códigos de IPs, que llamaremos Tabla 1, Tabla 2 y Tabla Resultado. Y el algoritmo es:

1) Se inicializa la Tabla 1 con un único elemento ficticio de valor todo ceros.
2) Para cada columna de la tabla de cobertura se ejecuta el proceso siguiente:
   2.1 Se cargan en la Tabla 2 todos los códigos de los IPs de la tabla de cobertura que tengan una X en la columna tratada.
   2.2 Se fusionan dos a dos todos los códigos contenidos en la Tabla 1 con todos los códigos contenidos en la Tabla 2, llevando el resultado a la Tabla Resultado. La fusión entre códigos se hace mediante un OR lógico de los bits de ambos códigos, que son de la misma longitud por construcción. Es decir, si cualquiera de los dos bits en la misma posición es 1, el bit resultante es un 1; si ambos bits son 0, el resultado es 0.
   En este paso, si la Tabla 1 contiene *n* elementos y la Tabla 2 contiene *m* elementos, el número de elementos resultantes en la Tabla Resultado será de $n \cdot m$, dado que cada elemento de la Tabla 1 debe fusionarse con todos los elementos de la Tabla 2, dando origen a un elemento en la Tabla Resultado por cada pareja de elementos.
   2.3 Se procesa la Tabla Resultado para eliminar elementos redundantes, en base a la aplicación de la Ley de Absorción y la Idempotencia. Para ello se compara cada elemento de la Tabla Resultado con todos los otros elementos de la propia Tabla Resultado.
   Para cada pareja de elementos comparados, se fusionan con un OR lógico ambos elementos en un área de trabajo. Se compara entonces dicha área de trabajo con los dos elementos tratados. Si fuera igual a uno cualquiera de los dos, se elimina dicho elemento. Si fuera igual a ambos, se elimina solamente uno de ellos.
   O, visto de otra manera:
   - Si los dos elementos comparados son exactamente iguales, salvo que uno de ellos tiene algunos unos adicionales donde el otro tiene ceros, entonces puede suprimirse el que tiene más unos (se trata de una Absorción).
   - Si los dos elementos comparados tienen exactamente los mismos unos y en las mismas posiciones, entonces puede suprimirse uno cualquiera de ellos (se trata de una Idempotencia), pero solamente uno.

   2.4 Se copian desde la Tabla Resultado todos los códigos restantes, aquellos que no han sido suprimidos, a la Tabla 1, y se continúa con la siguiente columna, hasta haber tratado todas las columnas de la tabla de cobertura.
3) La interpretación del resultado final en la Tabla 1 es la siguiente: cada elemento es un término, un producto de IPs, y todos los elementos de la Tabla 1 están sumados entre sí. Es decir, cada código presente en la Tabla 1 final representa a una cierta combinación de IPs que cubre todas las columnas de la tabla de cobertura. Los IPs que forman dicha combinación son los que corresponden a los unos del código presente en dicha Tabla 1. Cualquiera de los términos, es decir, cada uno de los elementos contenidos en la Tabla 1 al final del proceso genera una solución válida que cubre todas las columnas.
   Por tanto, <u>para localizar la solución mínima basta con revisar la Tabla 1 contando el número de unos de los elementos allí presentes, y quedarse con el que tenga un menor número de unos</u>, o uno cualquiera de ellos si hay varios.
4) Una vez seleccionado el elemento con menor número de unos, se revisan los bits de dicho elemento. Si en la posición *i-ésima* hay un 0, se ignora; si hay un 1, entonces se lleva el implicante primo de la fila *i-ésima* a la Tabla de IPs seleccionados, aquella que se había ido rellenando con los implicantes primos esenciales que se habían encontrado, y a partir de la cual se genera en el Paso 8 la expresión final buscada por el algoritmo.

Una característica a destacar de este algoritmo de Petrick es que en el proceso de fusión y comparación de los elementos de las diferentes tablas se utilizan sentencias OR, que son mucho más rápidas de ejecución que, por ejemplo, la fusión de implicantes primos que se efectúa en el **Paso 3 – Cálculo de los implicantes primos**, dado que los IPs a fusionar en ese Paso 3 pueden tener tres valores diferentes: 0, 1 y guión, lo que obliga a realizar un cierto y no muy eficiente proceso iterativo sobre los valores de los IPs.

Veamos cómo funciona el método de Petrick en el ejemplo que había definido antes, con 12 IPs en filas y 14 *minterms* en columnas.

Repito a continuación la tabla de cobertura del ejemplo:

	***Minterms* de la expresión**													
IPs	**0000**	**0001**	**0011**	**0100**	**0101**	**0110**	**0111**	**1000**	**1001**	**1010**	**1011**	**1100**	**1110**	**1111**
--00	X			X				X				X		
--11			X				X				X			X
-0-1		X	X						X		X			
-00-	X	X						X	X					
-1-0				X		X						X	X	
-11-						X	X						X	X
0--1		X	X		X		X							
0-0-	X	X		X	X									
01--				X	X	X	X							
1--0								X		X		X	X	
1-1-										X	X		X	X
10--								X	X	X	X			

En primer lugar se deben asignar los códigos a cada implicante primo tal como se indica a continuación. El tamaño de los distintos códigos será en este caso de 12 bits, puesto que 12 son los IPs de la tabla.

IP	**Código asignado**
--00	100000000000
--11	010000000000
-0-1	001000000000
-00-	000100000000
-1-0	000010000000
-11-	000001000000
0--1	000000100000
0-0-	000000010000
01--	000000001000
1--0	000000000100
1-1-	000000000010
10--	000000000001

Se inicializa la Tabla 1 con un único elemento a ceros, técnica que sirve en realidad para que la primera ejecución del bucle de columnas se limite a copiar el contenido de la Tabla 2 a la Tabla 1.

Por tanto, en la ejecución correspondiente a la segunda columna el contenido inicial de las Tablas 1 y 2 es el siguiente:

Tabla 1
100000000000
000100000000
000000010000

Tabla 2
001000000000
000100000000
000000100000
000000010000

La Tabla 1 contiene los códigos de los tres IPs que cubren la primera columna, es decir, que tienen una equis en la intersección entre dicho IP y el *minterm* correspondiente a la columna tratada, la primera. Estos IPs son IP1, IP4 e IP8.

En cuanto a la Tabla 2, contiene los códigos de los cuatro IPs que cubren la segunda columna, que son IP3, IP4, IP7 e IP8. Comprobad que es así en la tabla de cobertura anterior, si lo deseáis.

Ahora se fusionan todos los códigos de la Tabla 1 con todos los de la Tabla 2. El resultado será una tabla de 12 elementos (3·4), y su contenido es:

Tabla Resultado
101000000000
100100000000
100000100000
100000010000
001100000000
000100000000
000100100000
000100010000
001000010000
000100010000
000000110000
000000010000

En efecto, el primer elemento de la Tabla 1, que es 100000000000, al fusionarse con el primero de Tabla 2, 001000000000, recordemos que mediante un OR lógico, da como resultado el elemento 101000000000; con el segundo de Tabla 2, 000100000000, da 100100000000; etc. Luego se fusiona el segundo elemento de la Tabla 1, 000100000000, con los cuatro de la Tabla2, y luego el tercero, 000000010000, también con los cuatro de la Tabla 2, y así hasta la fusión de los dos últimos elementos de cada tabla, que como ambos son idénticos, 000000010000, dan como resultado exactamente el mismo código, 000000010000 (esto es así como consecuencia de la Idempotencia: IP8*IP8=IP8).

Ahora, antes de seguir con la iteración de la tercera columna hay que reducir en lo posible esta Tabla Resultado, eliminando aquellos elementos cuyos unos coinciden en número y posición con los de otro elemento, salvo que pueden tener un número adicional de unos.

Sean los elementos 100100000000 (el segundo de la Tabla) y 000100000000 (el sexto). Todos los unos del sexto de ellos son también unos en las mismas posiciones en el segundo de ellos, que además tiene algún uno más, en concreto el situado en la primera posición. Por lo tanto, el primero de ellos, 100100000000, puede ser eliminado, pues es absorbido por el segundo. Esta situación concreta representa la fórmula IP1*IP4+IP4, que se convierte en IP4, eliminando el término IP1*IP4 debido a la Ley de Absorción. La forma más sencilla de comprobarlo es fusionando ambos elementos con un OR lógico, lo que da como resultado 100100000000, que es exactamente igual que el segundo elemento de la tabla, y por tanto se puede eliminar dicho elemento, el segundo, pues es absorbido por el sexto.

Aplicada sistemáticamente esta comprobación a todos los elementos de la Tabla Resultado, se eliminan los elementos siguientes: 100100000000, 100000010000, 001100000000, 000100100000, 000100010000, 001000010000, 000100010000 y 000000110000, por lo que la Tabla Resultado queda finalmente con los cuatro elementos siguientes:

Tabla Resultado
101000000000
100000100000
000100000000
000000010000

Esta Tabla Resultado donde se han eliminado los elementos absorbidos se copia ahora completa a la Tabla 1, y se continúa el proceso tratando la siguiente columna, en este caso la tercera de la tabla de cobertura, que tiene tres equis, por lo que la Tabla 2 queda como se puede ver a continuación:

Tabla 1
101000000000
100000100000
000100000000
000000010000

Tabla 2
010000000000
001000000000
000000100000

El resultado de realizar la fusión entre ambas tablas en la Tabla Resultado, que nuevamente tendrá doce elementos (4·3, al ser cuatro los elementos en la Tabla 1 y tres los de la Tabla 2), es el siguiente:

Tabla Resultado
111000000000
101000000000
101000100000
110000100000
101000100000
100000100000
010100000000
001100000000
000100100000
010000010000
001000010000
000000110000

Tabla que, una vez eliminados los elementos redundantes, los que son absorbidos por algún otro, queda con los 8 elementos siguientes:

Tabla Resultado
101000000000
100000100000
010100000000
001100000000
000100100000
010000010000
001000010000
000000110000

Como la cuarta columna, que es la siguiente a tratar, tiene 4 equis, ahora la Tabla Resultado tendrá 32 elementos (8·4), que quedan reducidos a 12 tras eliminar los redundantes. El algoritmo sigue procesando el resto de columnas, con cada vez más elementos involucrados en las Tablas, hasta que tras tratar la ultima columna, la número 14, y eliminados los elementos redundantes quedan finalmente 58 elementos en la Tabla Resultado... habrá que creerme, o más bien creer al programa que lo calcula, porque no voy yo a detallar aquí todos los cálculos hasta llegar a esa cifra, que el lector armado de paciencia puede realizar si lo desea para comprobar que efectivamente la cifra es correcta.

Es decir, la expresión *Z* inicial, calculada como Producto de Sumas en base a la tabla de cobertura y convertida por obra y gracia del método de Petrick en una Suma de Productos, tiene 58 términos, 58 productos lógicos que se suman entre sí para obtener dicha expresión *Z*. Cada uno de esos 58 términos representa una combinación de implicantes primos que cubren todas las columnas de la tabla de cobertura. Por ejemplo, uno de los términos finales es 010101010100, que representa a la combinación de IPs {IP2, IP4, IP6, IP8, IP10}, combinación que en efecto cubre todos los *minterms*; otro de los términos que hay en la tabla es el 111010001001, que representa a la combinación {IP1, IP2, IP3, IP5, IP9, IP12}, etc.

Ahora basta con encontrar cuál de estos 58 elementos requiere menos IPs, es decir, tiene menos unos, para localizar la solución mínima que cubre todas las columnas, que es el objetivo final del algoritmo QM.

En este ejemplo hay cinco de los 58 términos totales que contienen solamente cuatro unos, y por tanto necesitan de solamente cuatro implicantes primos para cubrir la totalidad de las columnas. Son los siguientes: 101000001010, 100001100001, 010100001100, 010010010001 y 001001010100.

Queda clara aquí una característica del método de Petrick: encuentra no sólo *una* solución mínima, sino que las encuentra *todas ellas*; en algunos casos puede ser conveniente elegir una u otra en función de ciertos condicionantes. En nuestro caso, el de optimizar el código de un programa fuente, elegiremos la combinación que menos variables individuales precise, es decir, que tenga un menor número total de ceros y unos o, si lo preferimos, un mayor número de guiones entre todos los implicantes primos de la solución, pues así se garantiza que el programa fuente resultante tiene el menor número posible de condiciones a evaluar.

En el ejemplo, todos los IPs tienen dos guiones, por lo que a priori da igual una que otra solución, podemos elegir cualquiera de ellas para obtener la solución mínima buscada, por ejemplo el primero de ellos, el 101000001010, que da origen a la combinación {IP1, IP3, IP9, IP11}, y entonces estos cuatro implicantes primos, que son{--00, -0-1, 01--, 1-1-}, se llevan a la Tabla de IPs seleccionados, que en este ejemplo concreto estaba de momento vacía, dado que no se había detectado ningún implicante primo esencial en los pasos anteriores.

Así queda finalmente dicha tabla:

IPs Seleccionados
--00
-0-1
01--
1-1-

Y esta lista de IPs seleccionados da por fin origen, según el Paso 8 del algoritmo de Quine-McClusky explicado anteriormente, a la expresión siguiente: N3*N4+N2*C4+N1*C2+C1*C3. Ordenando la fórmula queda **C1*C3+N1*C2+N2*C4+N3*N4**.

No es posible extraer aquí ningún factor común, ni tampoco había ninguna variable implicante, por lo que ésta es, definitivamente, la expresión mínima que tiene como forma canónica a 1101111111111011. Una de ellas, puesto que como hemos visto existen otras cuatro combinaciones más que usan también cuatro IPs, el mínimo absoluto.

Y, no lo olvidemos, ésta es la solución mínima si se ejecuta el algoritmo QM tratando los *minterms*, los unos. Faltaría comprobar qué ocurriría si se tratan los ceros, los *maxterms*, para obtener la que es efectivamente la expresión mínima que tiene la forma canónica dada.

Resumiendo: El método de Petrick es un método muy sencillo de implementar una vez se entienden los procelosos artificios booleanos en que se basa, método que siempre proporciona una expresión mínima cuya forma canónica es la dada; es más, devuelve todas las posibles combinaciones que contienen un número mínimo de variables.

De hecho devuelve *todas* las combinaciones de implicantes primos de cualquier tamaño que cumplen la condición de cubrir todas las columnas de la tabla de cobertura.

El método de Petrick sólo tiene un pero, un "pero" ciertamente importante: **corre en tiempo exponencial**, aunque atemperado por la eliminación de términos derivada de la aplicación de la Ley de Absorción o la Idempotencia, que eliminan ciertos elementos de las tablas, un número apreciable de ellos en muchas ocasiones. Pero conforme crece el número de implicantes primos o, aún peor, el número de columnas de la tabla de cobertura, el número de elementos necesarios en las tres Tablas definidas crece y crece hasta volverse completamente inmanejable.

Supongamos una sencilla tabla de cobertura con 16 implicantes primos y 20 columnas, y supongamos que cada columna tiene cuatro equis, números habituales, incluso pequeños, si la forma canónica tiene 32 bits o más. Si no se eliminara ningún elemento de las tablas debido a la Ley de Absorción o a la Idempotencia, el número de productos de que se compondría la Suma de Productos final sería de 4, número de equis por columna elevado a 20, número de columnas. Y 4^{20} son muchísimos elementos para poder tratarlos, ni tan siquiera almacenarlos, pues son más de un billón (europeo), nada menos que 1.099.511.627.776 elementos, concretamente. Por mucho que los procesos de eliminación de elementos reduzcan sensiblemente el tamaño de las tablas, que efectivamente lo hacen, inexorablemente éstas crecen y crecen exponencialmente hasta desbordar todos los buffers plausibles.

Además, hay que tener en cuenta que el propio proceso de comprobación de elementos en la Tabla Resultado para eliminar los elementos redundantes también necesita de un tiempo considerable cuando crece el número de elementos: siendo n el número de elementos de dicha Tabla Resultado tras la fusión de los elementos de las Tablas 1 y 2, es preciso realizar $n\cdot(n\text{-}1)/2$ comparaciones de elementos para determinar si es posible eliminar uno u otro, pues deben compararse todas las posibles parejas de códigos. Por tanto, cuando el número de elementos crece, el tiempo de verificación de elementos también crece de acuerdo al cuadrado del número de elementos de la tabla.

Para una Tabla Resultado de, por ejemplo, 5.000 elementos, cifra que se alcanza con facilidad en cuanto la tabla de cobertura tiene más allá de 25 ó 30 columnas, este paso de comprobación debe comparar casi 12 millones y medio de pares de elementos, es decir, el tiempo requerido por este proceso no es despreciable en absoluto cuando el número de elementos contenidos en las tablas crece por encima de algunos miles.

En definitiva, al igual que ocurre con el método de fuerza bruta, y aunque requiera menos capacidad de cómputo para realizarlo, intentar aplicar el método de Petrick para resolver tablas de cobertura con más de 35, 40 ó quizás 50 columnas se vuelve totalmente impracticable, por muchas eliminaciones que se hagan por el camino.

Para terminar este artículo dedicado sobre todo al método de Petrick sólo queda indicar que, como todo en álgebra de Boole es dual, este método puede usarse también, sin cambio alguno, para realizar la transformación contraria, es decir, para convertir una Suma de Productos en un Producto de Sumas. No me extiendo más sobre esta conversión, dado que no es el objeto de estos artículos sobre minimización de funciones lógicas, pero el lector no hallará dificultad alguna en corroborarlo.

Todas las referencias al método de Petrick que he encontrado en la Red lo hacen como parte del algoritmo de Quine-McClusky. Se cita en la literatura como si su única utilidad fuera resolver la tabla de cobertura de implicantes primos que genera al algoritmo QM, lo que no es poco, pero es que en realidad **el método de Petrick para lo que sirve es para transformar una expresión en Forma Normal Conjuntiva en una equivalente en Forma Normal Disyuntiva, y viceversa,** siempre que sólo haya variables afirmativas, sin complementar, en la expresión. Nada más. Y nada menos.

Para poder utilizarlo para resolver la tabla de cobertura se realizan, como ya he explicado, astutos artificios lógicos asimilando las filas, los IPs, a las variables de la expresión, y las columnas, los *minterms*, a los términos, es decir, las sumas, de la expresión. Una vez realizada esta asimilación, el algoritmo hace su trabajo convirtiendo la expresión en FNC a FND, y un nuevo inteligente artificio lógico permite seleccionar la combinación que menor número de IPs contiene para dar respuesta al Paso 7 del algoritmo de Quine-McClusky.

Leyendo los artículos que hay en la Red sobre el método de Petrick no me queda claro si el método se circunscribe a los artificios booleanos descritos o abarca también el método en sí para transformar una expresión que sea un Producto de Sumas a otra equivalente expresada como Suma de Productos, siempre que todas las variables estén en modo afirmativo. Porque esta transformación se puede utilizar en cualquier problema que lo requiera, no sólo para encontrar la expresión mínima equivalente a una dada.

Aquí se acaba el algoritmo de Quine-McClusky tal como se describe en la literatura, con un endemoniado penúltimo paso devorador de recursos en cuanto la forma canónica supera un cierto tamaño. Esto es lo que hay, según todas las publicaciones que he revisado.

Entonces, si queremos minimizar una expresión lógica de cierto tamaño... ¿se acaba aquí el camino? ¿No hay manera de encontrar la solución mínima, o al menos una lo suficientemente cercana a la mínima correspondiente a una determinada forma canónica cuando ésta crece por encima de un determinado número de bits?

Cuando esto pasa lo que generalmente implica es que el número de implicantes primos y el de columnas de la tabla de cobertura crece más allá del límite a partir del cual el consumo de recursos hace virtualmente imposible calcularla ni por fuerza bruta ni por el método de Petrick; tan sólo queda probar el incierto camino de los algoritmos de programación entera, como el *branch&bound*... por cierto, en el artículo de la Wikipedia inglesa sobre el método de Petrick se afirma como si tal cosa que este método también es denominado "*branch&bound*"... ¡Lo que hay que leer!

En fin, según toda la documentación que yo he encontrado del algoritmo de Quine-McClusky, ya no hay más. No hay más algoritmos o procesos que permitan tratar la tabla de cobertura. Aquí se acaba, efectivamente, el camino.

Sin embargo, sí que hay otros caminos, otros métodos, otros algoritmos procedentes de otras áreas de la informática que son aplicables a este problema. A ellos dedicaré el quinto y último artículo de la serie.