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

(III)

Macluskey, 2023

En el primer artículo de la serie expliqué mi viejo interés por la optimización de funciones lógicas en programas fuente, y que el algoritmo de Quine-McClusky es el algoritmo utilizado desde hace muchos años para, a partir de la forma canónica de una expresión dada, obtener la expresión equivalente que es mínima, en el sentido de que tiene el menor número posible de condiciones individuales.

A continuación, en el segundo artículo, describí con el mayor grado de precisión de que soy capaz los diferentes pasos del algoritmo, incluyendo un Paso 1, Detección de las Variables Implicantes, añadido por mí al algoritmo, dado que no he encontrado nada similar en la ingente documentación que al respecto he revisado en la Red.

En él dejé para más adelante la descripción del penúltimo Paso del algoritmo, el Paso 7 – Obtención de la Solución Óptima, y ya adelanto que tampoco en este artículo lo voy a definir, cosa que haré en los dos próximos artículos de la serie, dado que su explicación detallada nos llevará a interesantes conclusiones. Lo que sí haré en este artículo es resolver el enigma planteado en el último párrafo del artículo anterior, que reproduzco de nuevo:

"... **ésta** (la expresión final obtenida por el algoritmo en su forma original) **no es la expresión mínima equivalente a la forma canónica dada**. Para obtener definitivamente 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*."

Esta afirmación es bastante contundente, pues implicaría de ser cierta que el algoritmo de Quine-McClusky tal como se presenta en la literatura no siempre da como resultado la expresión mínima equivalente. Y eso hay que probarlo.

Pues a eso vamos. Aunque ya voy adelantando que en mis investigaciones he encontrado que, dependiendo del tamaño de la forma canónica, *hay una probabilidad cercana al 50% de que el algoritmo no obtenga la función mínima buscada*. Es decir, que casi una de cada dos veces el algoritmo falla en obtener la expresión mínima equivalente a la dada.

Describamos, pues, cuál es ese Procedimiento alternativo que, ahora sí, encuentra siempre la función mínima equivalente a la forma canónica dada.

Procedimiento alternativo del algoritmo Quine-McClusky:

El algoritmo, como hemos visto en los dos artículos anteriores, se basa en tratar la forma canónica de la expresión a minimizar: toma los unos de dicha forma canónica para generar los *minterms*, luego fusionarlos para conseguir los implicantes primos, etc.

Preguntémonos ahora qué ocurriría si **en vez de tomar los unos de la forma canónica tomáramos los ceros**, y luego se siguieran los pasos del algoritmo exactamente igual que lo descrito hasta ahora.

El resultado de esa forma de proceder sería obviamente **la fórmula mínima que daría origen a la *expresión complementaria a la buscada***. Entonces, una vez obtenida esta expresión mínima complementaria, **para obtener la expresión buscada bastaría con negar la expresión complementaria así calculada**, aplicando las Leyes de De Morgan. Recuerdo que las Leyes de De Morgan preconizan que (a*b)'=a'+b', y que (a+b)'=a'*b'.

En fin, la expresión resultante estaría en este caso no en Forma Normal Disyuntiva, sino en Forma Normal Conjuntiva, es decir, sería un Producto de Sumas en vez de una Suma de Productos. Y sería igual de "mínima" que la obtenida por el método tradicional... o no. Enseguida veremos si esto es así.

Ignoro las implicaciones que en el diseño de circuitos electrónicos tiene el que la expresión sea un producto de sumas en vez de una suma de productos, pero puedo asegurar que si lo que se desea es optimizar el código de una expresión lógica contenida en un programa fuente da exactamente igual una u otra estructura de la expresión: lo importante es reducir al máximo el número de condiciones de la fórmula para minimizar el número de comparaciones a realizar para determinar el flujo del programa. Si hay más ORs que ANDs o viceversa no importa lo más mínimo, sólo importa reducir el número de condiciones todo lo posible.

En fin, tratar los ceros en vez de los unos implica tratar no los *minterms* de la expresión, sino los *maxterms*. Más adelante hablaré sobre las referencias que he encontrado sobre el uso de *maxterms* en la literatura sobre el algoritmo de Quine-McClusky.

Llevando a la práctica esta orientación de tratar los ceros en vez de los unos de la forma canónica, vamos a ejecutar los diferentes pasos del algoritmo tal como se definieron en el artículo anterior para el mismo ejemplo que hemos seguido hasta ahora. Será rápido.

El Paso 1 se ejecuta exactamente igual que antes, puesto que si una variable resulta ser implicante, lo será independientemente de que el resultado del algoritmo sea una Suma de Productos o un Producto de Sumas. Por lo tanto, igual que antes, la variable C1, y sólo ella, es implicante, sumando, y se extrae de la forma canónica.

Tras esta extracción de C1, recordemos, la forma canónica resultante era la siguiente: **1011100111111101**, y las variables individuales involucradas son C2, C3, C4 y C5, dado que C1 ha sido eliminada al ser reconocida como implicante. La forma canónica contiene doce unos y cuatro ceros; en este procedimiento alternativo trataremos los ceros de la forma canónica en vez de los unos, como preconiza el algoritmo original.

PASO 2: Carga inicial de implicantes primos.

Se cargan cuatro, uno por cada cero de la forma canónica, que son:

Num.unos	Implicante primo
1	0001
2	0101
2	0110
3	1110

PASO 3: Fusión de *minterms* para obtener los Implicantes Primos

Ciertamente en este caso son *maxterms*, pero mantendré la terminología y me seguiré refiriendo a ellos como *minterms* para resaltar el hecho de que el funcionamiento del algoritmo en este procedimiento alternativo es exactamente el mismo que antes.

Comparaciones

(1) 0001 – (2) 0101 = 0–01
(1) 0001 – (2) 0110 = **NO** (hay más de una diferencia)

(2) 0101 – (3) 1110 = **NO**
(2) 0110 – (3) 1110 = –110

Todos los IPs, los cuatro, se han fusionado con algún otro, por lo que no hay que copiar ninguno adicional a la nueva tabla de implicantes primos, que queda:

Num.unos	Implicante primo
1	0–01
2	–110

Comparación (sólo hay una posible en la segunda ronda)

(1) 0–01 – (2) –110: **NO**

No hay más fusiones posibles. Ésta es la tabla final de implicantes primos. Con ella, en el **Paso 4** se genera la tabla bidimensional de implicantes primos y *minterms*, que es así de sencilla:

IP	0001	0101	1100	1110
0–01			X	X
–110	X	X		

No hay mucho más que hacer aquí: ambos implicantes primos son esenciales, por lo que el **Paso 5** selecciona ambos y a continuación en el **Paso 8** se genera la expresión resultante en base a estos dos únicos implicantes primos, 0–01 y –110. Esa expresión, según preconiza el **Paso 8**, es la siguiente:

N2*N4*C5+C3*C4*N5

Ahora bien, dado que hemos tratado los ceros de la forma canónica de la expresión original, **la fórmula obtenida es en realidad la de la expresión complementaria a la buscada**. Por lo tanto, para obtener la expresión real buscada basta con negar la fórmula obtenida aplicando las Leyes de De Morgan. Una vez hecho esto el resultado, que ahora es un producto de sumas, es el siguiente:

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

Que tiene seis variables individuales en lugar de las ocho que tenía la expresión obtenida por el algoritmo tradicional, o bien siete tras aplicarle la propiedad distributiva. Es decir, **esta expresión es**, ahora sí**, la expresión mínima absoluta que resuelve la forma canónica 1011100111111101**. En este caso no se puede aplicar la distributiva para extraer factores comunes y se debe añadir, igual que antes, la variable implicante C1 sumando, con lo que la expresión final obtenida es:

C1+(C2+C4+N5)*(N3+N4+C5)

Si la comparamos con la fórmula obtenida por el procedimiento tradicional, que era: **C1+C2*N4+N5*(N3+N4)+C4*C5**, vemos que la recién obtenida tiene una variable individual menos. Y si operamos algebraicamente con una de las fórmulas llegaremos a obtener la otra, ambas son correctas y representan a la misma expresión.

El ejemplo anterior demuestra que hay casos en los que aplicar el algoritmo a los *maxterms* de la expresión, es decir, a los ceros de la forma canónica y luego negar la expresión resultante obtiene un mejor resultado, uno que contiene menos variables individuales que aplicándolo directamente a los *minterms*, los unos.

Con mucha dificultad he encontrado en la Red alguna información sobre la posibilidad de usar los *maxterms* en lugar de la forma universalmente aceptada de usar los *minterms* como se describe hasta ahora en este documento. Se trata de un artículo sobre diseño de circuitos electrónicos que cita la posibilidad de usar *maxterms* para obtener un Producto de Sumas, en vez de la habitual Suma de Productos, explicando que esto puede ser útil si la tecnología utilizada hace más eficiente el uso de puertas lógicas OR que el de las puertas AND: al usar *maxterms* resultan más puertas OR en el circuito que puertas AND, mientras que en el caso habitual en el circuito obtenido ocurre lo contrario. Yo no sé casi nada sobre diseño de circuitos, por lo que no puedo opinar al respecto, pero repito que si lo que se desea es optimizar funciones lógicas en programas fuente, que es lo que a mí me interesa y de lo que sí que entiendo, lo crítico es obtener una función booleana con el menor número de variables (condiciones) posibles, y tanto da que haya más ORs que ANDs o viceversa.

Lo que no he encontrado en lugar alguno es que se cite el hecho de que tratando los *maxterms* se pueda llegar a una expresión de menor tamaño que si se tratan los *minterms,* y viceversa.

El caso es que queda demostrado que en ciertas ocasiones se obtiene una expresión mejor, con menos variables individuales representadas, tratando los *maxterm*s (los ceros de la forma canónica) y luego complementando la expresión resultante, que tratando los *minterms* (los unos) tal como se explica en la inmensa mayor parte de la literatura. Este hecho, el que tratar los ceros y aplicar las Leyes de De Morgan al resultado da en ocasiones mejores resultados que tratar los unos, no lo he encontrado así explicado en ninguna parte de la Red, por muchas búsquedas que he realizado. Ignoro si esta afirmación que hago aquí es realmente una primicia, aunque lo dudo; estoy prácticamente seguro de que en algún lugar alguien, quizás algún doctorando o investigador, ha llegado a la misma conclusión. Pero yo no le he encontrado.

Bien, en este punto de la investigación surgen naturalmente dos preguntas:

¿**Con cuánta frecuencia ocurre que dé mejores resultados**, es decir, una expresión de menor tamaño, **tratar los *maxterms* en lugar de los *minterms***?; y

¿**Es posible conocer a priori**, en base a la forma canónica, **si dará mejor resultado utilizar un método u otro**?

Dado que en principio no tenía ni idea de cómo podrían contestarse estas preguntas de forma teórica, la única forma que se me ocurrió para intentar responderlas es probando: ejecutar el algoritmo tanto con *minterms* como con *maxterms* con diferentes formas canónicas de distinto número de variables y estudiar el resultado comparando ambas soluciones, e intentar sacar conclusiones de estas pruebas. Para ello he construido diferentes programas de simulación que hacen exactamente esto: a partir de una determinada forma canónica se ejecuta el algoritmo de ambas formas, se comparan los resultados y se anota cuál de los dos métodos ha funcionado mejor en cada caso, o si ambos han dado como resultado fórmulas de igual tamaño. Y, desde luego, antes de nada he probado exhaustivamente el código del algoritmo en sí para asegurar que funciona correctamente en toda ocasión. Lo hace.

Los parámetros de las pruebas realizadas han sido los siguientes:

- He tratado formas canónicas de entre tres y diez variables individuales, es decir, entre 8 y 1024 bits en la forma canónica.
  Con dos variables diferentes los resultados son obvios, y mis medios materiales me imposibilitan tratar formas canónicas de más de 1.024 bits, y estas últimas con gran dificultad y consumo de recursos, tiempo de CPU, sobre todo.
  Espero que los resultados puedan extrapolarse con cierta fiabilidad a formas canónicas de mayor tamaño, pero desde luego que no puedo asegurar tal cosa.

- Para formas canónicas de tres o cuatro variables individuales (tamaños de 8 y 16 bits en la forma canónica) he realizado el proceso de comprobación para todas las posibles formas canónicas de dicha longitud, que son 254 en el caso de tres variables individuales y de 65.534 en el caso de cuatro (se excluyen las formas canónicas con todo ceros o todo unos, que dan como resultado cero y uno, respectivamente y no tienen ningún misterio).
  Más allá de eso es para mí imposible realizar la comprobación de todas las posibles formas canónicas: con cinco variables y 32 bits en la forma canónica existen casi 4.300 millones de combinaciones posibles: imposible atacar la comprobación exhaustiva en ese caso, y no digamos para seis o más variables individuales, son cifras astronómicas.

- Consecuentemente, para formas canónicas de 32 bits en adelante he programado una simulación. El programa de simulación genera formas canónicas pseudoaleatorias, esparciendo los ceros y los unos en la forma canónica en base a un procedimiento razonablemente aleatorio, y realiza las comprobaciones con estos valores generados.
  La generación aleatoria tiene el inconveniente de que, por decirlo de algún modo, la expresión resultante "no tiene sentido". Me refiero a que cuando un programador escribe una cierta expresión lógica busca representar la realidad, algo que debe suceder en los datos para tomar uno u otro camino en la ejecución del programa; sin embargo, una forma canónica aleatoria no representa en principio nada "lógico", y esto ciertamente tiene consecuencias al evaluar la expresión.

- En todos los casos, dada una forma canónica obtenida de una u otra manera he ejecutado el algoritmo de ambas formas, incluyendo, desde luego, el Paso 7 (Obtención de la Solución Óptima) que aún no he llegado a definir, y comprobado si es mejor uno u otro método contando las variables individuales contenidas en cada una de las dos expresiones devueltas, sea una Suma de Productos (*minterms*), sea un Producto de Sumas (*maxterms*).

- Además, para los tamaños más pequeños (hasta 6 variables diferentes) he realizado una simulación similar, pero esta vez aplicando a las expresiones resultantes la propiedad distributiva para sacar factores/sumandos comunes y reducir más aún las expresiones. La ejecución del proceso de extracción de factores o sumandos comunes es muy costosa cuando las expresiones son de gran tamaño, como las que se obtienen cuando la forma canónica generada aleatoriamente tiene, por ejemplo, 1.024 bits.

En total he tratado varios millones de formas canónicas de diferentes tamaños, como se muestra en la tabla siguiente:

Nº variables individuales	Nº bits de la forma canónica	Combinaciones probadas sin aplicar la distributiva	Combinaciones probadas aplicando la distributiva	Total de Combinaciones probadas
3	8	254	254	254
4	16	65.534	65.534	65.534
5	32	2.000.000	250.000	2.250.000
6	64	200.000	20.000	220.000
7	128	60.000	–	60.000
8	256	20.000	–	20.000
9	512	2.000	–	2.000
10	1024	250	–	250

Lógicamente, cuantas más variables (más bits en la forma canónica) más costoso es el proceso, y también es sensiblemente más costosa la búsqueda sistemática de factores/sumandos comunes; ésa es la razón de que cuanto mayor es el tamaño de la forma canónica, menor sea el número de simulaciones efectuadas, dado que mis medios materiales para la ejecución de estos procesos son, obviamente, limitados. Por ejemplo, en el caso de formas canónicas de 1.024 bits, la comprobación de ambos algoritmos para determinar cuál es la menor de las dos expresiones encontradas tarda en mi máquina de media cinco minutos por cada forma canónica... y eso sin intentar siquiera aplicar la propiedad distributiva, lo que podría llevar entre veinte y treinta minutos adicionales por cada forma canónica. Efectivamente, usar una muestra de solamente 250 formas canónicas aleatorias para intentar determinar lo que les ocurre a las del orden de 10^{310} (2^{1024}) restantes es irrisorio, pero habrá que conformarse. Ir más allá es para mí completamente imposible. Y no digamos para formas canónicas de más de 1.024 bits.

En cualquier caso, del estudio de los diferentes resultados se pueden extraer conclusiones valiosas, que paso a detallar a continuación, intentando responder en lo posible a las dos preguntas anteriores.

Pregunta 1: ¿Con cuánta frecuencia ocurre que dé mejores resultados tratar los ceros, es decir, obtener una expresión de menor tamaño tratando los *maxterms* en lugar de los *minterms*?

Veamos: Para las formas canónicas de tres y cuatro variables (8 y 16 bits, respectivamente), donde se han evaluado la totalidad de formas canónicas posibles, y fijándonos en las expresiones devueltas por el algoritmo (sumas de productos o productos de sumas), **el número de formas canónicas para las que se obtiene mejor resultado tratando los unos *es exactamente el mismo* que aquellas para las que se obtiene mejor resultado tratando los ceros**.

He aquí los datos:

En formas canónicas de 8 bits (tres variables, 254 formas canónicas en total), en 60 casos (un 23,6%) se obtiene una expresión más pequeña tratando los unos, en otros 60 casos (otro 23,6%) se obtiene una expresión más pequeña tratando los ceros y en los 134 casos restantes (un 52,8%) ambos métodos dan expresiones de la misma longitud.

En cuanto a las formas canónicas de 16 bits (cuatro variables, 65.534 formas canónicas en total), en 23.304 casos se obtiene una expresión más pequeña tratando los unos, en otros exactamente 23.304 casos se obtiene una expresión más pequeña tratando los ceros y en los restantes 18.926 casos ambos métodos dan expresiones de la misma longitud. En porcentaje, en el 35,6% de los casos da mejor resultado tratar los unos; en otro 35,6%, tratar los ceros, y en el 28,8% restante ambos métodos dan expresiones del mismo tamaño.

Viendo estos datos, lo primero que a uno se le ocurre es que algo muy similar, mejor dicho, exactamente lo mismo va a pasar con todos los tamaños de forma canónica, pero ¿cómo podría yo confirmarlo, o no?

¿Qué ocurre en formas canónicas de mayor longitud, aquellas que se han investigado generando formas canónicas aleatorias? Pues que lógicamente el número de casos en que resulta mejor uno u otro método no coincide, pero la diferencia entre ellos es ciertamente pequeña y asumo que es perfectamente factible atribuirla a la variabilidad generada por el proceso de generación pseudoaleatoria de formas canónicas. Las cifras concretas de las simulaciones de cada tamaño de forma canónica son las siguientes:

Formas canónicas de 32 bits: 930.364 casos donde es mejor tratar los unos; 952.825 casos donde es mejor tratar los ceros; y 116.811 donde da igual un método u otro. Los porcentajes de casos donde da mejor resultado un método u otro rondan ya el 47%.

Formas canónicas de 64 bits: 95.085 casos donde es mejor tratar los unos; 95.784 casos donde es mejor tratar los ceros; y 9.131 donde da igual un método u otro. Ahora los porcentajes de casos donde da mejor resultado bien un método, bien otro rondan el 48%.

Formas canónicas de 128 bits: 28.345 casos donde es mejor tratar los unos; 29.690 casos donde es mejor tratar los ceros; y 1.965 donde da igual un método u otro.

Formas canónicas de 256 bits: 10.066 casos donde es mejor tratar los unos; 9.518 casos donde es mejor tratar los ceros; y 416 donde da igual un método u otro.

Formas canónicas de 512 bits: 930 casos donde es mejor tratar los unos; 1.043 casos donde es mejor tratar los ceros; y 27 donde da igual un método u otro.

Formas canónicas de 1024 bits: 93 casos donde es mejor tratar los unos; 153 casos donde es mejor tratar los ceros; y 4 donde da igual un método u otro.

Dado el bajo número de casos simulados para formas canónicas de 1.024 bits, sólo 250 de muchos billones de cuatrillones, es imposible sacar consecuencia alguna de estos datos, y de hecho lo mismo podemos decir de las formas canónicas de menor tamaño. Sin embargo, los datos sí que muestran una clara tendencia: en unas simulaciones ha dado mejores resultados tratar los ceros; en otras, los unos; y en los únicos casos en los que el tamaño de la forma canónica permite tratar todas las formas canónicas posibles, las de 8 ó 16 bits, se constata que hay un número idéntico de formas canónicas que dan mejor resultado tratando los unos que tratando los ceros.

Mi conjetura es, pues, la siguiente:

Para todo tamaño de forma canónica, el número de casos en los que la expresión mínima absoluta se obtiene tratando los unos (*minterms*) es exactamente el mismo que el número de casos en los que la expresión mínima se obtiene tratando los ceros (*maxterms*) y luego complementando la expresión obtenida aplicando las leyes de De Morgan.

¿Será esta afirmación algo más que una simple conjetura, habrá alguna forma de demostrar que esto es así, convirtiendo la conjetura en un teorema?

Sí que la hay. Vamos a ello:

Para convertir esta plausible conjetura en un teorema hay que demostrar que para todo tamaño de forma canónica ocurre este hecho: que el número de formas canónicas en que resulta óptimo utilizar un método es idéntico al de formas canónicas en que es óptimo utilizar el otro. Sabemos que con tamaños de forma canónica de 4, 8 y 16 bits esta conjetura se cumple, constatándolo por fuerza bruta, es decir, probando la totalidad de formas canónicas posibles de estas longitudes. Sólo queda extender la prueba a formas canónicas de mayor tamaño, donde es imposible utilizar la fuerza bruta debido al gigantesco número de formas canónicas posibles.

Vamos a hacerlo en base a una serie de Lemas que iré demostrando hasta llegar a la demostración final. Algunos son evidentes, pero no quiero dar nada por sentado y ser lo más riguroso posible.

En primer lugar definiré el concepto de "*forma canónica espejo*" de una forma canónica dada, de la forma:

"**Dada una determinada forma canónica *FC*, existe una y sólo una forma canónica espejo de ella, *FE*, de su misma longitud, tal que todos los bits de *FE* tienen los valores contrarios a los de la forma canónica *FC* en su misma posición**".

Por ejemplo, dada la forma canónica *FC*: 1101010001010101, existe una y sólo una forma canónica espejo *FE* de ella y de igual longitud: 0010101110101010. Y desde luego, a su vez *FC* es la forma canónica espejo de *FE*. Obviamente, esta definición implica que *FE* es la forma canónica de la expresión complementaria a la de *FC*, y viceversa.

LEMA 1: Toda forma canónica tiene una única forma canónica espejo.

Como cada forma canónica es distinta a todas las demás de su misma longitud, invirtiendo los valores de todos sus bits se llega a otra forma canónica de las posibles que, a su vez, es única. Por lo tanto, todas las formas canónicas de una cierta longitud se organizan en parejas, donde cada una de ellas es espejo, es decir, complementaria, de la otra.

LEMA 2: Dada una determinada longitud *n* de formas canónicas, donde el número N de formas canónicas diferentes es $N=2^n$, existen N/2 parejas de formas canónicas donde una es espejo de la otra, y viceversa.

Dado que toda forma canónica de las 2^n posibles tiene una única forma canónica espejo y viceversa, el número total de parejas existentes es el número N dividido por el número de componentes de cada pareja, 2, y por tanto N/2, es decir $2^n/2$.

Habiendo quedado bien establecida la definición de las parejas de formas canónicas espejo, fijémonos ahora en cómo trata el algoritmo de Quine-McClusky a cada una de estas formas canónicas espejo. Observaremos qué ocurre tanto cuando para cada una de ellas se tratan los unos como cuando se tratan los ceros.

Recordemos en este punto que en el Paso 1 del algoritmo QM se extraen las posibles variables implicantes que pudiera tener la forma canónica dada.

LEMA 3: Si una cierta forma canónica contiene una o varias variables implicantes, entonces su forma canónica espejo contiene también el mismo número de variables implicantes.

De hecho, las de la forma espejo son las mismas variables que las de la inicial, pero complementadas, en su mismo orden y afectadas por el operador contrario: si una variable implicante C*x* está afectada por el operador '+', en la forma espejo estará complementada, es decir, será N*x*, y estará afectada por el operador contrario, '*'.

Si una determinada configuración de unos en una de las formas determina que existe una variable implicante sumando, entonces en la forma espejo existe una configuración idéntica de ceros en sus mismas posiciones, y por lo tanto la misma variable, pero complementada, es implicante, pero esta vez multiplicando. Y viceversa, si una determinada configuración de ceros en una de las formas determina que existe una variable implicante multiplicando, entonces en la forma espejo existe una configuración idéntica de unos en sus mismas posiciones, y así la misma variable, complementada, es implicando, pero ahora sumando.

Veamos un ejemplo; en concreto usaremos el mismo que hemos usado hasta ahora, de la forma canónica 1011100111111101111111111111111. Sabemos que C1 era una variable implicante, sumando, dado que todos sus unos eran unos también en la forma canónica:

Forma canónica: 1011100111111101111111111111111
Tabla de verdad de C1: 0000000000000000**1111111111111111**

Una vez extraída esta variable implicante C1, sumando, la forma canónica resultante quedaba: 1011100111111101.

Y por otro lado, la forma canónica espejo de 1011100111111101111111111111111 es, claro está: 01000110000000100000000000000000.

En esta forma canónica espejo podemos ahora comprobar que, al compararla con la tabla de verdad de N1, la complementaria de C1, todos sus ceros coinciden con ceros en la forma espejo:

Forma canónica espejo: 01000110000000100000000000000000
Tabla de verdad de N1: 1111111111111111**0000000000000000**

Por lo tanto N1, complementaria de C1, es implicante, pero en este caso multiplicando, de la forma espejo de la original. Y la forma canónica resultante tras extraer dicha variable implicante N1 es la siguiente: 0100011000000010.

LEMA 4: Tras la extracción de las variables implicantes de una forma canónica y de su forma espejo en el Paso 1 del algoritmo QM, las dos formas canónicas resultantes tras la extracción siguen siendo la una espejo de la otra.

La eliminación del mismo número de bits en ambas formas canónicas de la pareja de formas espejo, y en las mismas posiciones, deja una serie de bits que no son modificados y en sus mismas posiciones relativas, y por lo tanto continúan siendo unos los contrarios de los otros. Lo podemos ver en el caso anterior: las formas canónicas resultantes de la extracción en las dos formas canónicas espejo son: 1011100111111101 y
0100011000000010.

Ambas continúan, pues, formando una pareja de formas canónicas espejo.

LEMA 5: La expresión resultante de aplicar el algoritmo de Quine-McCluskey a una forma canónica determinada tratando los unos tiene idéntica longitud que la de la expresión obtenida por dicho algoritmo aplicado a su forma canónica espejo, pero tratando los ceros y complementando, vía las Leyes de De Morgan, la expresión resultante.

Y viceversa, la expresión resultante de aplicar el algoritmo de Quine-McCluskey a una determinada forma canónica tratando los ceros y complementando la expresión resultante tiene idéntica longitud que la de la expresión que obtiene dicho algoritmo aplicado a la forma canónica espejo, pero tratando los unos.

Fijémonos en cómo se produce la carga inicial de *minterms* en la tabla de implicantes primos, o propiamente *maxterms*, si se tratan los ceros. Sean las formas canónicas *FC* y *FE* que forman una pareja de formas espejo, que van a ser objeto de tratamiento por el algoritmo QM. Tomemos primero a la forma canónica *FC* tratando los unos. El Paso 2 del algoritmo carga en la tabla de IPs todos los *minterms* cuyo valor en dicha forma *FC* es un uno.

Comparemos ahora con lo que el propio Paso 2 carga en la tabla de IPs si tomamos ahora la forma canónica *FE*, pero tratando los ceros. Se cargarían en la tabla de IPs todos los *maxterms* cuyo valor en dicha forma *FC* es un cero. Comparemos ahora el contenido de ambas tablas de IPs tras esta carga inicial en uno y otro caso.

¡Son idénticas!

En efecto, al formar las formas canónicas *FC* y *FE* una pareja de formas espejo, todos los unos existentes en la forma *FC* son ceros en las mismas posiciones de la forma *FE*, y viceversa. Por lo tanto, tratando los unos con la forma canónica *FC* se cargan todos los *minterms* con valor uno, que son exactamente los mismos que tienen valor cero en la forma canónica *FE*. Pero en esta forma canónica *FE* se están tratando precisamente los ceros, los *maxterms*, por lo que los valores que en definitiva se cargan inicialmente en la tabla de IPs son los mismos.

Todo el resto del algoritmo, los Pasos 3 a 8, es idéntico en ambos casos, dado que sus datos de entrada, la tabla de IPs inicialmente cargada en dicho Paso 2, son los mismos. El único punto en que varía el algoritmo en ambos casos es que en el paso final, una vez obtenida la expresión resultante y antes de añadir las variables implicantes, en el caso de tratar los ceros la expresión debe ser complementada usando las Leyes de De Morgan, mientras que si se tratan los unos este paso no se realiza.

Sin embargo, la ejecución de dicha complementación o negación de la expresión resultante no varía el número de variables de la expresión: todas las variables son sustituidas por sus complementarias, C1 por N1, etc, y los operadores se intercambian, donde hay un '+' se cambia por '*' y viceversa. Puede ser necesario añadir paréntesis a la expresión, pero esto no altera el número de variables de la expresión.

Luego las longitudes de las expresiones así obtenidas son idénticas.

LEMA 6: Si en una determinada pareja de formas canónicas espejo {*FC*, *FE*} el algoritmo QM aplicado a la forma *FC* obtiene la solución mínima tratando los unos, el algoritmo aplicado a su forma canónica espejo *FE* obtendrá la solución mínima tratando los ceros, y viceversa.

Es consecuencia inmediata del LEMA 5. Las expresiones obtenidas por el algoritmo QM tratando los unos en una forma canónica y tratando los ceros en su forma canónica espejo son siempre de la misma longitud, y viceversa.

Por tanto, si en una de estas dos formas canónicas se obtiene mejor resultado tratando, por ejemplo, los unos, en su forma canónica espejo se obtendrán los mejores resultados tratando los ceros, y viceversa.

Y, por fin, en aquellas formas canónicas en las que se obtengan expresiones de la misma longitud bien aplicando el algoritmo a los unos, bien a los ceros, también ocurrirá lo mismo en sus respectivas formas canónicas espejo, dado que las longitudes de las expresiones están ligadas dos a dos: tratando los ceros en una y tratando los unos en la otra.

Veamos un ejemplo. Sea la pareja de formas canónicas espejo siguiente, de longitud 8 y, por tanto, de tres variables individuales C1, C2 y C3.

FC: 10011000 y
FE: 01100111.

Cada una es espejo de la otra. No hay variables implicantes, en ninguna de las dos. Recordemos que si en una de ellas se hubieran detectado una o varias variables implicantes, en la otra se habrían detectado el mismo número de ellas, debido al LEMA 3.

Apliquemos en primer lugar el algoritmo QM tratando los unos a la forma *FC*. La carga inicial de *minterms* en la tabla de IPs cargaría los siguientes: {000, 011 y 100}. A partir de ahí el algoritmo se ejecuta normalmente y el resultado es la expresión: N1*C2*C3+N2*N3, con cinco variables individuales en ella.

Si aplicamos ahora el algoritmo QM a la forma *FE*, pero tratando los ceros, vemos que los *maxterms* cargados inicialmente en la tabla de IPs son exactamente los mismos que en el caso anterior: {000, 011, 100}. Por lo tanto, el algoritmo devolverá exactamente la misma expresión N1*C2*C3+N2*N3, que debe en este caso ser complementada para obtener la expresión definitiva: (C1+N2+N3)*(C2+C3), también con cinco variables individuales y, por lo tanto, del mismo tamaño que tratando los unos en la forma espejo.

De forma similar, si aplicamos ahora el algoritmo QM tratando ceros a la forma *FC*, cargará inicialmente en la tabla de IPS los cinco *maxterms* correspondientes a los cinco ceros de *FC*, que son: {001, 010, 101, 110, 111}. Ejecutando ahora el algoritmo QM, obtiene la fórmula siguiente, ya complementada mediante las Leyes de De Morgan: (C2+N3)*(N2+C3)*(N1+N2), con seis variables individuales. Por tanto, es óptimo en esta forma canónica *FC* usar el algoritmo QM tratando los unos.

Aplicando ahora el algoritmo QM a la forma *FE*, pero esta vez tratando los unos, la carga inicial de la tabla de IPS termina con los mismos cinco *minterms* del caso anterior en la tabla, y tras ejecutar el algoritmo se obtiene la expresión: N2*C3+C2*N3+C1*C2, de seis variables, y por ello en el caso de la forma canónica *FE* el resultado óptimo se obtiene tratando los ceros en el algoritmo QM.

Es decir, en el caso de la pareja de formas espejo *FC* y *FE* del ejemplo, en una de las dos, *FC*, la expresión mínima se obtiene tratando los unos, mientras que en la otra, *FE*, el resultado óptimo se obtiene tratando el valor contrario, los ceros. Circunstancia que ocurre en toda pareja de formas canónicas espejo de cualquier tamaño, excepto que con ambos métodos (tratar unos o ceros) se obtengan expresiones del mismo tamaño, en cuyo caso es irrelevante usar uno u otro método para obtener la expresión mínima correspondiente a las dos formas canónicas espejo que forman la pareja.

TEOREMA: Hay un número idéntico de formas canónicas de una determinada longitud en las que el algoritmo QM obtiene la expresión mínima tratando los unos que formas canónicas de la misma longitud en las que el algoritmo QM obtiene la expresión mínima tratando los ceros.

Es consecuencia inmediata del LEMA 6, que garantiza que en toda pareja de formas canónicas espejo, si en una de ellas resulta óptimo tratar los unos, en la otra resultará siempre óptimo tratar el valor contrario, los ceros, y viceversa, y del LEMA 2, que indica que para formas canónicas de *n* bits de longitud existen N/2 parejas de formas canónicas espejo diferentes, siendo N el número total de formas canónicas posibles para dicha longitud *n*, es decir, $N=2^n$.

Cada pareja de formas canónicas espejo puede ser de dos tipos: 1) en la pareja es irrelevante qué método utilizar en el algoritmo QM, debido a que ambos métodos, tratar unos o ceros, obtienen expresiones del mismo tamaño; y 2) en una de las dos formas canónicas espejo es óptimo tratar los ceros y en la otra lo es tratar los unos.

Y por tanto, **el número total de formas canónicas de longitud *n* en las que la solución mínima se obtiene tratando los ceros (*maxterms*) y complementando la expresión final mediante las Leyes de De Morgan es idéntico al número total de formas canónicas de dicha longitud *n* en las que la solución mínima se obtiene tratando los unos (*minterms*)**.

QED

Hasta aquí, todo esto es fijándose en la expresión en bruto obtenida por el algoritmo QM, sea ésta una suma de productos, sea un producto de sumas. Es obvio que en una parte significativa de los casos esta expresión se puede reducir aplicando la propiedad distributiva para sacar factores/sumandos comunes. Entonces, ¿qué ocurre cuando se aplica la distributiva, cambian algo los resultados?

Bien, veamos ahora cuáles son los resultados de aplicar la propiedad distributiva en los cuatro tamaños de forma canónica que he estudiado (8, 16, 32 y 64 bits):

Formas canónicas de 8 bits (tres variables): Todos los 254 casos dan ahora expresiones de exactamente el mismo tamaño usando uno u otro método.

Formas canónicas de 16 bits (cuatro variables): Ahora la cifra de casos en que resulta mejor aplicar uno u otro método ya no es exactamente la misma: 15.527 casos donde es mejor tratar los unos; 15.590 casos donde es mejor tratar los ceros; y 34.417 donde da igual un método u otro. Han aumentado significativamente (se han más que cuadruplicado) los casos en los que ambos métodos dan expresiones del mismo tamaño, pero ahora ya no son exactamente iguales los casos en que da mejor resultado un método o el otro. La diferencia es pequeña, 15.527 vs. 15.590, pero ya no son iguales.

En vista del TEOREMA que acabo de definir esto no es posible, porque aplicar la propiedad distributiva a expresiones que ya hemos visto que son complementarias unas de otras debería obtener siempre expresiones del mismo tamaño, y los datos indican que no siempre lo hace, y eso no es teóricamente posible. Pero en el mundo real ocurre. Al menos en *mi* mundo real. La explicación de que se dé esta diferencia, explicación que he contrastado a base de reproducir de forma manual algún caso concreto, es la siguiente:

Todo algoritmo que trate de reducir el tamaño de una expresión lógica aplicando la propiedad distributiva para sacar factor/sumando común a ciertos términos tiene que, tras analizar la expresión, elegir un determinado término de todos los posibles para extraerlo el primero de la fórmula, ignorando el resto, y esto debe hacerlo iterativamente tantas veces como

pueda mientras queden términos repetidos en la fórmula. En el ejemplo citado en el artículo anterior de la serie, recordemos, se podía elegir N4 o N5 como variable para extraerla como factor común, pero no a ambas.

Pues bien, por muy sofisticado que sea el algoritmo elegido, que tenga en cuenta tanto el número de veces que se encuentra en la fórmula cada término repetido como su tamaño, y seleccione el más idóneo según sus parámetros, y por muy sofisticados que sean dichos parámetros, siempre existirán fórmulas que se hubieran podido reducir más escogiendo otros términos o haciéndolo en otro orden. Y esto es exactamente lo que ocurre aquí: esa diferencia de 63 casos en el número de casos (15.527 vs 15.590) se produce precisamente debido a este sesgo del algoritmo de aplicación de la propiedad distributiva.

Cierto, seguro que se puede escribir un algoritmo mejor, pero yo no sé hacerlo. Bueno, sí que sé, tratando todas las posibles formas de extraer factores/sumandos comunes y quedándose con el que mejor resultado dé, pero yo no pienso programar esa monstruosidad. Sigamos.

Formas canónicas de 32 bits (cinco variables): Hay 99.609 casos donde es mejor tratar los unos; 101.677 donde es mejor tratar los ceros; y 48.714 casos donde las expresiones obtenidas por uno u otro método son de idéntico tamaño.

Formas canónicas de 64 bits (seis variables): Hay 8.628 casos donde es mejor tratar los unos; 9.244 en los que es mejor tratar los ceros; y 2.128 donde da igual usar uno u otro método.

En resumen, se puede extender el TEOREMA antes definido si las expresiones obtenidas son reducidas mediante la aplicación de la propiedad distributiva. Así se pueden obtener fórmulas de menor tamaño que las obtenidas directamente por el algoritmo, si ése fuera el objetivo de la aplicación del algoritmo. Lo que sí ocurre tras la aplicación de la distributiva es que aumentan considerablemente los casos en que ambos métodos, tratar unos y tratar ceros, obtienen expresiones del mismo tamaño.

En cualquier caso, y según los datos de las simulaciones realizadas, el porcentaje de formas canónicas en que es indiferente usar uno u otro método se reduce conforme el tamaño de las formas canónicas aumenta: de un 52,8% de las formas canónicas de 8 bits de tamaño, se pasa a un 28,8% en las de 16 bits; a un 5,8% en las de 32 bits; un 4,5% en las de 64 bits; un 3,2% en las de 128 bits; a un magro 2,1% en las de 256 bits, etc. Y esto implica que conforme aumenta el tamaño de la forma canónica la probabilidad de que el resultado óptimo se consiga tratando bien los unos, bien los ceros, se aproxima al 100%.

Es decir, en base a los datos conseguidos mediante las diferentes simulaciones podemos asegurar que **aplicando el algoritmo de Quine-McClusky tal como se define en la literatura hay una alta probabilidad, cercana al 50% para formas canónicas de una cierta longitud, de que la expresión obtenida no sea la mínima absoluta correspondiente a la forma canónica dada**.

La reflexión inmediata tras toda este discurso es que para asegurar que se obtiene en todo caso la expresión mínima correspondiente a una determinada forma canónica es necesario ejecutar el algoritmo QM dos veces, una tratando los unos, la forma tradicional, y otra tratando los ceros y complementando la expresión resultante mediante las Leyes de De Morgan, para calcular el tamaño de cada una de las expresiones así obtenidas y quedarse con la de menor tamaño.

<u>Pregunta 2</u>: ¿Es posible conocer a priori, en base a la forma canónica, **si dará mejor resultado utilizar un método u otro**?

La respuesta corta es: **No**. O mejor, **casi seguro que no**. O mejor aún, yo no he encontrado la forma de hacerlo.

Por más suposiciones y pruebas que he hecho (que la forma canónica tenga más ceros que unos o viceversa, que se den en ella ciertas configuraciones de ceros y unos, etc.) no he encontrado ningún criterio que pueda determinar a priori si va a dar mejor resultado uno u otro método. Como mucho se puede seleccionar probabilísticamente qué método va a dar mejor resultado en base a observar algunos millones de casos, como yo he hecho.

Veamos un ejemplo de lo que ocurre en el caso de las formas canónicas de 16 bits (es decir, de cuatro variables), de las que he tratado todas y cada una de las 65.534 formas canónicas posibles:

Expresión directamente devuelta por el algoritmo (como suma de productos o producto de sumas)				
	Característica de la forma canónica			
Resultado	**Más unos que ceros**	**Más ceros que unos**	**Igual número**	**TOTALES**
Mejor tratar Unos	13.288	7.664	2.352	23.304
Mejor tratar Ceros	7.664	13.288	2.352	23.304
Mismo resultado	5.380	5.380	8.166	18.926
TOTALES	26.332	26.332	12.870	65.534

Dadas estas cifras, la mejor estrategia consiste en:
- Si la forma canónica tiene más unos que ceros, o los mismos, entonces tratar los **unos**;
- Si la forma canónica tiene más ceros que unos, entonces tratar los **ceros**.

Comprobadlo, si queréis. Esta estrategia acierta en el 73,02% de las ocasiones, y por lo tanto falla en el restante 26,98% de las veces. Es más corto calcular la probabilidad de fallo: 7.664+7.664+2.352=17.680 de 65.534, un 26,98%. Que no deja de ser una alta probabilidad de fallo, me parece a mí, como para grabar en piedra la estrategia.

¿Qué pasaría si a estas mismas expresiones en bruto entregadas por el algoritmo en cualquiera de sus dos formas, tratar unos o ceros, les aplicamos la propiedad distributiva para reducir su tamaño sacando factores/sumandos comunes? Veamos:

Expresión resultante tras aplicar la propiedad distributiva				
	Característica de la forma canónica			
Resultado	**Más unos que ceros**	**Más ceros que unos**	**Igual número**	**TOTALES**
Mejor Unos	8.111	4.668	2.748	15.527
Mejor Ceros	4.687	8.100	2.803	15.590
Da igual	13.534	13.564	7.319	34.417
TOTALES	26.332	26.332	12.870	65.534

La mejor estrategia ahora es casi la misma que en el caso anterior, pero ahora la tasa de acierto es mejor:
- Si la forma canónica tiene más unos que ceros, entonces tratar los **unos**;
- Si la forma canónica tiene más ceros que unos, o los mismos, entonces tratar los **ceros**.

Esta estrategia falla en obtener la expresión mínima en un 18,47% de las ocasiones (4.687+4.668+2.748=12.103 casos de 65.534), y por tanto acierta en un 81,53% de las veces.

Es decir, aplicar la distributiva mejora la tasa de acierto de la estrategia de elegir uno u otro método, pero conforme va aumentando el tamaño de las formas canónicas, la probabilidad de acertar eligiendo de antemano uno u otro método va disminuyendo en cualquier caso hasta acercarse al 50%, lo que básicamente es lo mismo que elegir uno u otro método de forma aleatoria, o incluso mejor, elegir siempre el mismo.

Así que si lo que se desea es **obtener siempre la expresión mínima** correspondiente a una determinada forma canónica no queda más remedio que **ejecutar dos veces el algoritmo conforme a ambos métodos**, tratando los unos y tratando los ceros, **y quedarse con la expresión que resulte de menor tamaño**. Y, desde luego, aplicar la propiedad distributiva a la expresión obtenida para extraer cuantos factores/sumandos comunes sea posible.

Bien. Terminada la disquisición sobre la conveniencia de usar un método (tratar los *minterms*) u otro (tratar los *maxterms*), es el momento de definir el paso restante del algoritmo, ése que había dejado para más adelante, el Paso 7, el que de verdad precisa cantidades astronómicas de recursos informáticos para ejecutarse. El problema de minimizar expresiones lógicas es un problema *NP-completo*, como ya dije, y es precisamente en este Paso 7 donde esa *NP-Completitud* se muestra en todo su esplendor.

Pero este **Paso 7** restante, **Obtener la Solución Óptima**, a diferencia de los Pasos vistos hasta ahora, tiene múltiples alternativas y además tiene implicaciones en otras áreas de la ciencia informática, por lo que lo desarrollaré a lo largo de los dos siguientes artículos.