MINIMIZACIÓN HEURÍSTICA

(86:44) Técnica Digital Avanzada- Unidad 4.

Profesor: Ing. Miguel Antonio Martínez.

Minimización Heurística.

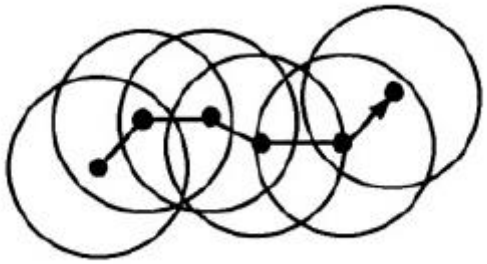
El principal problema de la minimización exacta es el número potencialmente grande de Implicantes Primos. Para la mayoría de las funciones con muchas entradas, hay muchos implicantes primos y demasiados términos mínimos. Los métodos heurísticos evitan calcular todos los implicantes primos y todos los términos mínimos en un intento de evitar el costo computacional. El enfoque habitual es modificar sucesivamente una cobertura inicial dada de la función, hasta una detención adecuada de acuerdo a algún criterio cumplido.

El precio a pagar es la incapacidad de garantizar la solución más barata, aunque la elección cuidadosa de los algoritmos puede proporcionar soluciones muy cercanas a la solución óptima en un tiempo razonable.

Los algoritmos que comienzan con una solución inicial y tratan de encontrar una mejor aplicando modificaciones sucesivas se llaman algoritmos de **búsqueda local.** El algoritmo de búsqueda local trata con un espacio de búsqueda, que es el conjunto de todos los valores posibles que las variables de entrada pueden tomar. El conjunto de todos los puntos del espacio de búsqueda que son soluciones válidas forman la **región factible.** El complemento de esa zona es la **región inviable.**

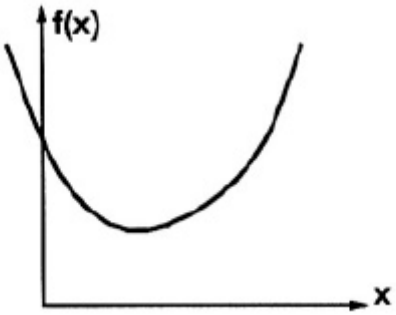
La búsqueda local se basa en la definición de una **distancia** entre dos puntos de la región factible. Una vez que definimos distancia podemos introducir el concepto de **vecindad** de un punto de **radio r,** dentro del espacio factible, como el conjunto de puntos cuya distancia **s** sea menor que **r.**

A partir de una solución inicial, el algoritmo examina su vecindario para encontrar un punto factible cuyo costo es menor que el costo actual. Si se encuentra uno, se supone como el nuevo punto de partida y el proceso se repita hasta que se cumpla algún criterio de detención. Esto se muestra en la siguiente figura:

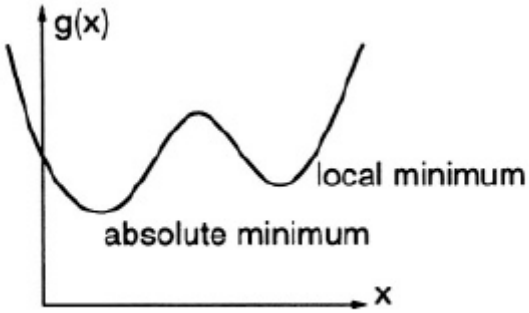


Hay infinitas posibilidades como variantes de este esquema, dependiendo de cómo se elija el nuevo punto factible, donde hay más vecinos, etc.

Dependiendo del problema y de la definición de vecinos, varios grados de optimización de la solución pueden ser garantizados. Supongamos que estamos interesados en el mínimo de una función convexa de una variable real como la de la figura siguiente. La búsqueda local consiste en moverse hacia abajo a lo largo de la curva y está garantizado el encontrar un mínimo verdadero.



Sin embargo si la función no es convexa, la búsqueda local se puede atascar en un mínimo relativo. Esta solución es mínima aunque no óptima. Este tipo de función se muestra en el siguiente gráfico.



En general, una propiedad interesante de la solución de un algoritmo de búsqueda local es la llamada **Optimidad Local**. Esto quiere decir que una solución es localmente óptima si su vecindario no contiene otra solución de menor costo. Para garantizar la optimización local se puede usar el siguiente criterio de detención: **"Deténgase cuando no haya en el vecindario una solución más barata que la actual".**

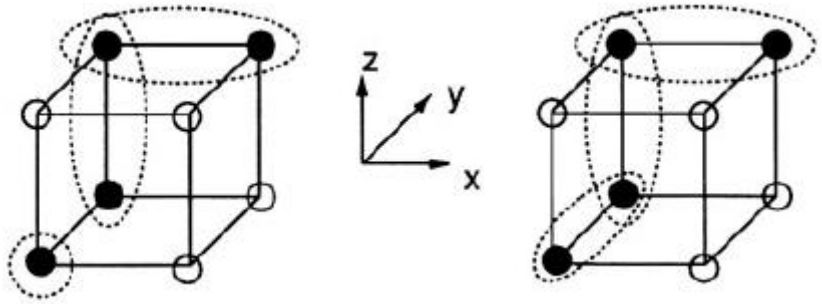
Ahora veremos cómo usamos la búsqueda local en el caso de minimización lógica. Nuestro punto de partida es una cobertura para una función de salida múltiple y el costo será el número de cubos (o número de términos producto) con el número de literales como desempate.

El vecindario que consideramos inicialmente se define como en conjunto de coberturas que se obtienen de la cobertura inicial agregando (o eliminando) exactamente un literal a uno de los cubos. La nueva cobertura así obtenida es un punto factible en el espacio de búsqueda si la función es equivalente a la original, o sea representa la misma función.

Cuando un cubo tiene todos ceros en su parte de salida, se puede descartar. Por lo tanto nuestra definición de vecindario nos permite obtener soluciones con menos cubos que la cobertura inicial. Por otro lado, las soluciones con más cubos que la cobertura inicial no son parte de nuestro espacio de búsqueda. Consideremos la función que representa la siguiente tabla donde las variables con "rayitas" indican la primera cobertura.

xyz	f
000	1
01–	1
–11	1

Antes de delinear el algoritmo debemos considerar los efectos de nuestro movimiento dentro del espacio de búsqueda. Por ejemplo en la figura siguiente vemos dos cubos booleanos, la parte izquierda es la cobertura inicial dada por la tabla anterior. La parte de la derecha es el resultado de cambiar la palabra 0 0 0 por 0 – 0, es decir eliminamos un literal de entrada.



De la figura de la derecha vemos que podemos eliminar un literal de los cubos 0 1 0 y de 0 1 1 generando 0 1 - , con esto logramos remover un cubo y nos queda la solución óptima:

xyz	f
0–0	1
–11	1

Visto desde el punto de vista del consenso, vemos que la función de la primera tabla se puede escribir como:
f = $\bar{x}.\bar{y}.\bar{z}$ + $\bar{x}$.y + y.z
Al expandir el término $\bar{x}.\bar{y}.\bar{z}$ Reemplazándolo por $\bar{x}.\bar{z}$ nos queda la función como:
f = $\bar{x}.\bar{z}$ + $\bar{x}$.y + y.z
Y vemos que el término $\bar{x}$.y es el consenso entre el primer y tercer término, por lo tanto la función final queda:
f = $\bar{x}.\bar{z}$ + y.z
que corresponde a la segunda tabla de verdad.

En el caso de tener **funciones no totalmente especificadas,** podemos definir a estas funciones como:

F = (f, d, r) perteneciente a un espacio B^n , en el cual tenemos valores de 0, de 1 y de redundancias. O sea B^n contiene { 0, 1, * }.

Llamamos f a los valores que hacen la función verdadera, o sea si f(x) = 1 ↔ F(x) = 1.
Llamamos r a los valores que hacen la función falsa, o sea r(x) = 1 ↔ F(x) = 0.
Llamamos d a los valores irrelevantes o redundantes, o sea d(x) = 1 ↔ F(x) = *

(f,d,r) forman una partición de B^n , o sea f + d + r = B^n y fd = fr = dr = ϕ, o sea son disjuntos. Una función completamente especificada **g es una cobertura** de F = (f,d,r) si f está incluida en g y g está incluida en f + d, o sea :

f ™ g ™ f + d

Un cubo **c** es un **implicante primo** de F = (f,d,r) si c está incluido en f + d, y no existe otro implicante de f + d que contenga a c.

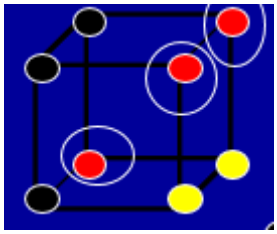
$\forall\ \bar{c}$ / $\bar{c}$ ™ f + d y c ◊ $\bar{c}$

Un cubo $\mathbf{c_j}$ de una cobertura F = {c_j} es **redundante** si f ™ F \{c_j}. O sea f está incluida en la función excluyendo el cubo $\mathbf{c_j}$. Si esto no ocurre se lo llama **irredundante.**

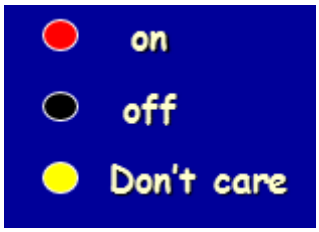
Ejemplo:

Consideremos la función F(a,b,c) = (f,d,r) donde f = {$\bar{a}b\bar{c}$, a$\bar{b}$c, abc} y d = {a$\bar{b}\bar{c}$, ab$\bar{c}$}.

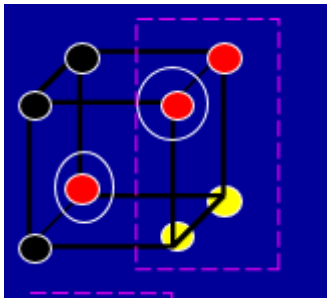
La función la representamos en el siguiente cubo booleano:

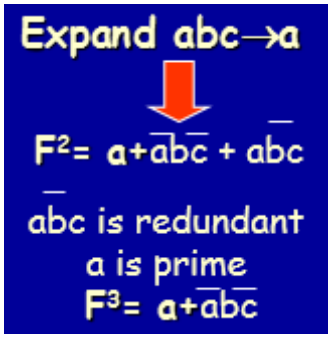


Donde:

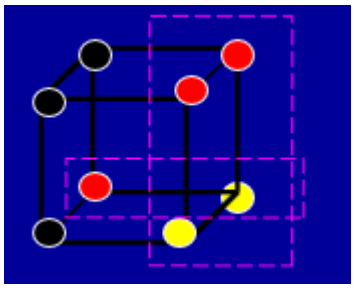


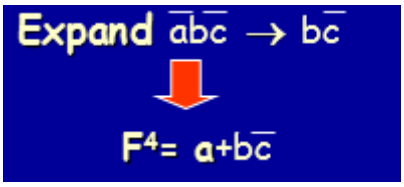
Expandiendo abc → a nos queda:





Expandiendo $\bar{a}b\bar{c}$ → b$\bar{c}$ llegamos al final con :

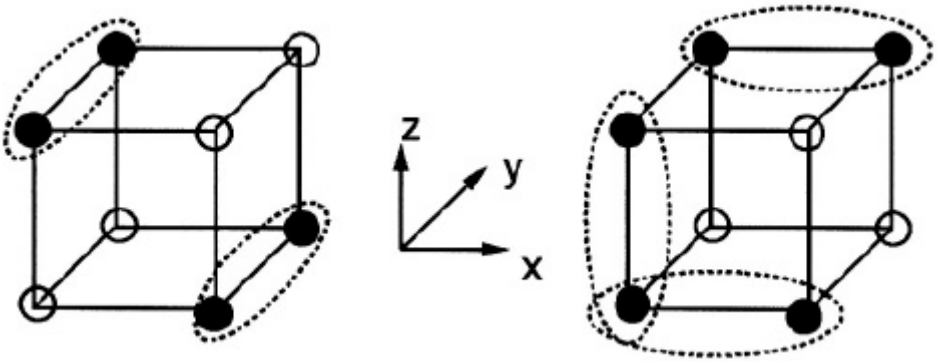




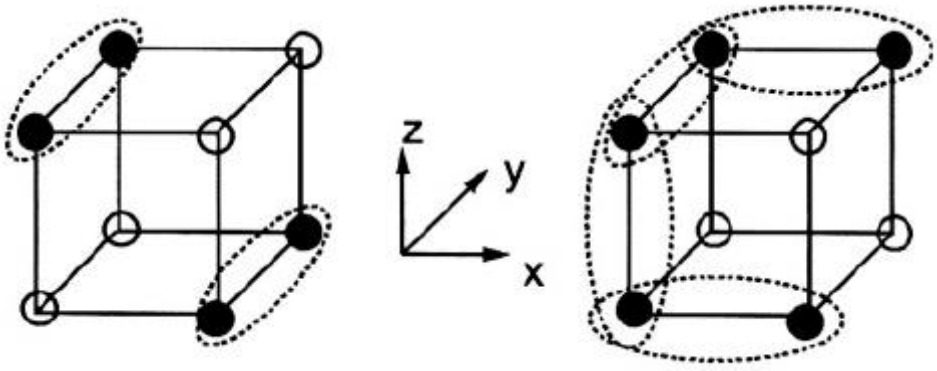
Otro ejemplo que vamos a ver es el caso que tenemos dos funciones llamadas f_1 y f_2 cuya tabla con una cobertura inicial vemos en la siguiente figura:

xyz	$f_1 f_2$
0–1	10
1–0	10
00–	01
–00	01
–11	01

La representación cúbica booleana de estas dos funciones se ven en el siguiente diagrama:



Expandiendo la expresión de f_2 tenemos el siguiente diagrama:



O sea ahora nos queda una tabla de cobertura de la siguiente manera:

xyz	$f_1 f_2$
0–1	11
1–0	10
00–	01
–00	01
–11	01

En el diagrama cúbico de f_2 se ve ahora que el cubo 00- es ahora **redundante.** Por lo tanto lo eliminamos y nos queda la siguiente tabla de cobertura más reducida (con una fila menos).

xyz	$f_1 f_2$
0–1	11
1–0	10
–00	01
–11	01

Visto de otra forma podemos decir que las dos funciones f_1 y f_2 son:

$\bar{x}$ .z + x.z

$\bar{x}.\bar{y} + \bar{y}.\bar{z}$ + y.z

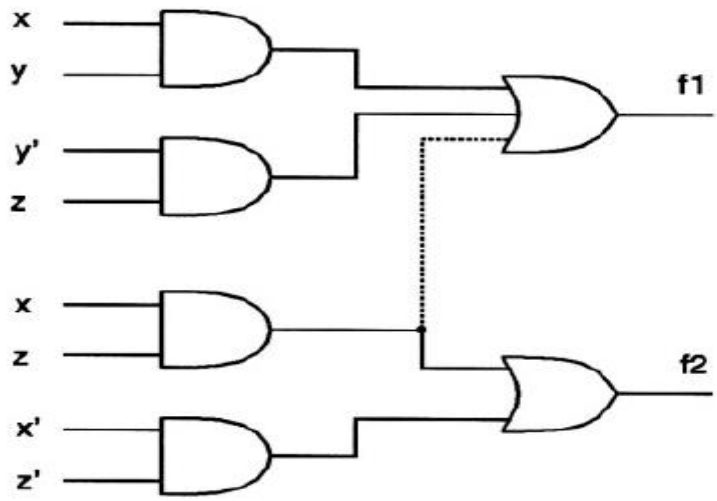
respectivamente, introduciendo en la segunda el término de consenso $\bar{x}$.z es la expansión de la salida del cubo correspondiente. Vemos que este término de consenso es común para las dos funciones y llegamos a la misma conclusión.

Los ejemplos vistos anteriormente sugieren que un procedimiento basado solo en los movimientos simples que disminuyen inmediatamente el costo puede no ser muy efectivo. Necesitamos considerar una variedad más amplia de movimientos y combinarlos en secuencias de movimientos que eventualmente conducen a las reducciones deseadas en el costo. En otras palabras, nosotros necesitamos expandir el vecindario que estamos buscando. La idea es iterar tres fases, primero expandimos todas las entradas y salidas. Esto tiene como efecto aumentar la superposición de cubos. Si tenemos suerte, uno o más cubos serán cubiertos completamente por la unión de otros cubos. Estos cubos cubiertos o redundantes se eliminan en esta segunda fase, al final de la cual estamos en un **mínimo local** o muy cerca de él. La tercera fase intenta salir de este mínimo local reduciendo los cubos. Las tres fases se repiten hasta que no haya más mejoras.

Consideremos un ejemplo con la siguiente tabla:

xyz	$f_1 f_2$
11–	10
–01	10
1–1	11
0–0	01

Luego de expandir entradas y salidas nos queda el circuito de la figura:



En el procedimiento la línea punteada es redundante ya que corresponde al término de consenso (**x.z**) entre **x.y** y $\overline{y}$ **.z.**

Hasta ahora hemos asumido que podíamos hacer expansiones y reducciones válidas. Aunque esto puede ser relativamente sencillo para los ejemplos simples que hemos visto hasta ahora, en realidad representa un problema no trivial para funciones muy grandes. Nuestra discusión se centra en dos aspectos, el primero es el que vinimos viendo, o sea encontrar el mejor movimiento dentro de muchos posibles. El segundo, que es lo que veremos ahora, es verificar la validez del movimiento hecho, es decir que la cobertura obtenida al expandir o reducir algunos cubos sea equivalente a la función original.

Cuando se expande un cubo, se le agregan nuevos términos. Queremos comprobar que ninguno de estos términos pertenece a los ceros de la función. Si esta condición está satisfecha, entonces estamos aumentando la superposición de términos existentes o cubriendo algunos términos previamente no utilizados del conjunto de redundancias. Del mismo modo cuando se reducen cubos se eliminan términos mínimos. Debemos asegurarnos de que esos términos mínimos sean cubiertos por algunos cubos o pertenece al conjunto de redundancias. En ambos casos, nuestra verificación de equivalencia implica verificar si el recién agregado o eliminado esté contenido en alguna otra cobertura.

Sea F = { c_j } una cobertura de los unos de la función y D una cobertura para las redundancias de la función. Sea c_i un cubo de F que es expandido o reducido y sea $\bar{c}_i$ el cubo que contiene el nuevo término agregado o eliminado. Para que la expansión o reducción sea válida se debe cumplir lo siguiente:

$$\bar{c}_i \leq (F - \{ c_j \}) \cup D$$

En esta fórmula F – { c_i } es la cobertura obtenida al remover c_i de la función F. Luego $\bar{c}_i$, (F – {c_i}) y D son considerados conjuntos de términos mínimos. Curiosamente, podemos usar un solo algoritmo para verificar la validez de las expansiones y las reducciones.

Consideremos la siguiente tabla de cobertura:

xyz	f
000	1
01–	1
–11	1
100	–

Supongamos que queremos verificar la validez de la expansión del primer cubo de 000 a 0-0. Entonces en este caso c_i = 000 y $\bar{c}_i$ = 010 porque la suma 000 + 010 = 0-0 . Debemos chequear si 010 está contenido en la siguiente tabla:

xyz	$(F - \{c_i\}) \cup D$
01–	1
–11	1
100	1

El último término es el de las redundancias, lo que contribuye a la contención como cualquier otro cubo. Por ejemplo x.z $\leq$ x.y + $\bar{y}$.z . Por lo tanto la prueba de contención no es trivial. No podemos simplemente escanear l alista de cubos en (F – { c_j }) $\cup$ D, y buscar
uno que contenga $\bar{c}_i$. Este es un método sencillo pero podría haber muchísimos términos mínimos para examinar.

Una posible mejora con respecto a la estrategia anterior podría ser la siguiente: inicialmente verificar la contención de un solo cubo de $\bar{c}_i$. Si esto falla, nos dividimos en mitades y verificamos las dos mitades por separado. Si ambos están contenidos hemos terminado. Si solo uno es contenido, el otro se divide de nuevo y así sucesivamente. Si eventualmente no encontramos minitérminos que no estén contenidos, la prueba de contención ha tenido éxito. Aunque este método es mejor que el anterior, podemos hacerlo considerablemente mejor. Sin embargo, necesitamos algunas definiciones que ahora presentaremos.

Recordando la definición de **cofactor de una función f(x_1, x_2, ……., x_n) respecto de una variable x_1 se escribe fx_1 es igual a f(1, x_2, ……, x_n).** Así mismo el cofactor de la misma función pero con respecto a $\bar{x}_i$ se escribe con **$f\bar{x}_i$ y es igual a f(0, x_2, ….., x_n).**

Recordando el teorema de Expansión de Boole podemos escribir:

$f = x_1.fx_1 + \bar{x}_1.f\bar{x}_1$

Es importante destacar que la cofactorización es conmutativa:

$(fx_1)x_2 = (fx_2)x_1 = fx_1x_2$

Esto nos permite establecer la siguiente definición:

El cofactor de una función f(x_1, x_2, ……., x_n) con respecto a un cubo "c" es la cofactorización sucesiva de la función f con respecto a todos los literales del cubo "c".

Finalmente, es conveniente introducir la siguiente definición:

- **Una función que es idénticamente uno es llamada tautología.**
- **Para una función "f" y un cubo "c":**

$c \leq f \Leftrightarrow f \equiv 1$

Volvamos al ejemplo anterior. Si c = x.z y f = x.y + $\bar{y}$.z . Tenemos que fc = y + $\bar{y}$ y verificamos que x.z $\leq$ x.y + $\bar{y}$.z. En vista del teorema anterior nuestra tarea es computar **((F – { c_j }) ∪ D)$\bar{c}_i$**
Y verificar si esta ecuación es una **tautología.**
Las ventajas de esta formulación son:

- La cofactorización puede simplificar enormemente el trabajo al reducir el número de variables y cubos.
- Tratamos un problema más uniforme, la tautología.

Necesitaremos un algoritmo para la verificación de tautologías. La base de este algoritmo es, una vez más, el teorema de expansión. En particular uno puede ver fácilmente que:
f = 1 ⟺ fx = f$\bar{x}$ = 1

En palabras, una función es **tautóloga** si y solo si **ambos cofactores** con respecto a cualquier variable son **tautólogos**. Por lo tanto, es posible dividir recursivamente la cobertura hasta que se encuentre un caso trivial. Seguimos con algunas definiciones.

Una función f(x_1, x_2, ……, x_n) es **monótona creciente** en una variable x_1 si y solo si:
f(0, x_2, ….., x_n) ≤ f(1, x_2,……, x_n), ∀ (x_2, ……, x_n)

Una función f(x_1, x_2, ……, x_n) es **monótona decreciente** en una variable x_1 si y solo si:
f(0, x_2, ….., x_n) ≥ f(1, x_2,……, x_n), ∀ (x_2, ……, x_n)
Por ejemplo, f = x.y + $\bar{y}$.$\bar{z}$ es monótona creciente en x, monótona decreciente en z y no es monótona en y. Se puede ver que f$\bar{x}$ = $\bar{y}$.$\bar{z}$ ≤ y + $\bar{z}$ = fx. Similarmente se puede verificar que
fz ≤ f$\bar{z}$. Mientras que fy e f$\bar{y}$ no se pueden comparar.

Recordando lo visto antes, una función f(x_1, x_2, ….., x_n) se denomina **unate** en la variable **x_1** si y solo si es **monótona creciente o monótona decreciente** en **x_1.** Se dice que una función es **unate** (sin especificaciones adicionales) si es **unate en todas las variables.**

Una función que es monótona creciente en todas sus variables será una tautología si y solo si **f(0, 0, ……., 0) = 1.** Claramente si **f(0, 0, ……, 0) ≠ 1,** entonces **f no es una tautología.**
Por otro lado si **f(0, 0, ……, 0) = 1**, por definición **f(x_1, x_2, …., x_n) = 1** para todos los valores de (x_1, x_2, ….., x_n). Análogamente si una función es **monótona decreciente** para todas sus variables solo debemos chequear el valor de (1, 1, ……, 1).

Decidir si una función es **unate** es general no sencillo, si queremos chequear el valor de la función para todas las entradas. Por suerte hay un caso especial donde ver si la función es **unate** se puede realizar por inspección. Lo vemos en la siguiente definición:

Una cobertura F es **monótona creciente** en la variable **x_1** si y solo si esta variable nunca aparece **complementada** en los términos de F. Una cobertura G es **monótona decreciente** respecto de la variable **x_1** y solo si esta variable nunca aparece **sin complementar** en los términos de G. Si ninguna de las dos condiciones anteriores se cumple, entonces ni F ni G son **monótonas respecto de x_1.**

Volviendo al ejemplo anterior x.y + $\bar{y}.\bar{z}$ es monótona creciente en x, es monótona decreciente en z y no es monótona respecto de y. Esto coincide con lo que vimos anteriormente para la función x.y + $\bar{y}.\bar{z}$. Sin embargo, si consideramos la función x + x.$\bar{y}$, nosotros vemos que la función es monótona creciente en ambas variables x e y, mientras la cobertura es solo monótona creciente en y. Si nos fijamos x + y es una cobertura alternativa de esta función y que x + y es monótona creciente en ambas variables. Este ejemplo da pie para el siguiente teorema.

Si una función f(x_1, x_2,….., x_n) es unate respecto de una variable x_1, entonces existe una cobertura de f que es unate en x_1. Si una cobertura F es unate en x_1, entonces la función F también es unate en x_1.

La segunda parte del teorema se puede demostrar observando que el producto y la suma de funciones monótonas crecientes es una función monótona creciente. Se deduce que una cobertura monótona creciente, que es la suma de productos de funciones monótonas crecientes (de x_i) representa una función monótona creciente.

Como consecuencia de este teorema, todos los resultados que probamos para funciones independientes se pueden aplicar a coberturas independientes. Además, podemos buscar cubiertas independientes que son fáciles de identificar y asegurarse que las correspondientes funciones son unate. Al hacerlo, no identificaremos algunas funciones como tales y, por lo tanto, perderemos algunas oportunidades de optimización. Sin embargo, este es un precio razonable a pagar para mantener la identificación de funciones unate independientes.

Para nosotros, el resultado más interesante en las coberturas unate tiene que ver con la verificación de tautologías.

Una cobertura unate F es una tautología si y solo si contiene un término constante 1.

Lo que queremos decir aquí es que el término aparece explícitamente entre los términos que constituyen F. Por ejemplo x + y no es una tautología, mientras que x + 1 si lo es. Viéndolo desde la representación cúbica, una cobertura tautóloga debe contener un cubo cuya parte de entrada es todo "-", o sea todas redundancias. El resultado de esto es que una rápida verificación de una función monótona creciente puede ser chequeada para tautología evaluándola en (0, 0, ….., 0). De hecho no existe un cubo que contenga literales no complementados que tenga el vértice (0, 0, …., 0). El teorema visto dice que la verificación de tautología para las coberturas no es trivial, La idea es adoptar el paradigma recursivo habitual basado en el Teorema de Expansión de Boole. En cada ciclo probamos la función para ver si es unate. Esto puede ser fácil mientras buscamos la variable para dividir, contando por separado el número de variables complementadas y no complementadas de cada variable. Como un ejemplo sencillo consideramos la siguiente cobertura no unate.

xyz	f
1–1	1
11–	1
00–	1
0–0	1

Separando en la variable “x”, nos queda:

yz	f_x
–1	1
1–	1

yz	f'_x
0–	1
–0	1

Los dos cofactores son unate, pero la función no es una tautología. Si una cobertura es unate solo en algunas variables no es posible contestar rápidamente si es una tautología. Sin embargo es posible simplificar considerablemente el cálculo gracias a la siguiente observación. Una cobertura F para una función $f(x_1, x_2, ..., x_n)$ que es monótona creciente respecto de x_1 se puede escribir como:

$$F(x_1,....., x_n) = x_1 . A(x_2,....., x_n) + B(x_2,....., x_n)$$

Donde A y B son fórmulas escritas como suma de productos, reagrupando términos y factorizando respecto de x_1, se ve claramente que:

$$Fx_1 = A + B \quad y \quad F\bar{x}_1 = B$$

Resulta que:

$$F\bar{x}_1 \leq Fx_1$$

De acuerdo con la definición, de manera equivalente tenemos que si:

$$F\bar{x}_1 \equiv 1 \Rightarrow Fx_1 \equiv 1$$

De donde:

$$F\bar{x}_1 \equiv 1 \Rightarrow F \equiv 1$$

$$F\bar{x}_1 \not\equiv 1 \Rightarrow F \not\equiv 1$$

Resumiendo si:

$$\mathbf{F\bar{x}_1 \equiv 1 \Leftrightarrow F \equiv 1}$$

Por lo tanto, es suficiente probar el cofactor negativo para chequear tautologías y así obtener la respuesta para la función completa. Del mismo modo, si una función es unate negativa en una variable, es suficiente probar el cofactor positivo para chequear tautología. Por lo tanto, para cada variable unate que encontramos, reducimos aproximadamente a la mitad el trabajo a realizar. Por ejemplo veamos el siguiente ejemplo:

$wxyz$	f
–1–1	1
0–10	1
010–	1

Todos los términos mínimos con w = 1 y x = 0 causan que f sea 0. Después de la simplificación concluimos que la función no es tautóloga.

Hay técnicas adicionales para acelerar la verificación de tautologías. Como vimos hemos visto hasta ahora la ejecución del algoritmo es exponencial. Si nos preguntamos si este comportamiento es óptimo, no tendríamos respuesta porque es extremadamente improbable que pudiéramos encontrar un algoritmo polinómico para chequera tautologías. El comportamiento es exponencial debido al que el número de nodos de un árbol de búsqueda es exponencial (recordar el método de branch and bound). En la práctica, sin embargo, el comportamiento no es tan malo ya que podemos podar mucho el árbol de búsqueda. Las funciones no utilizadas son un mecanismo que utilizamos para reducir la cantidad de nodos que necesitamos explorar. Primero trataremos de detectar casos especiales. Ellos son los siguientes:

- Una fila que tenga todas redundancias "-" en las entradas. Si se encuentra una fila de estas características, la función es **tautóloga para todas las salidas que tengan un "1" en esa fila.** Está condición es necesaria y suficiente.
- Una columna de entrada que tenga todos "1" o todos "0". Si se encuentra dicha columna, **la función no es tautóloga**. (B = 0 en la ecuación vista anteriormente).
- Si el número de entradas de la función es menor que 8, se genera la tabla de verdad y la pregunta si es tautóloga se responde por **inspección.** Esto se debe a que al haber un número de entradas lo suficientemente pequeño, es más fácil y rápido generar la tabla de verdad que aplicar este método recurrente.

  Otra técnica que utilizaremos se llama de **partición** y consiste en lo siguiente: Si una

Cobertura F puede escribirse como:

F = G + H

Donde G y H son funciones disjuntas (o sea no tienen variables en común), entonces F es **tautóloga si y solo si G ó H son tautólogas.** Sean x_1, ….., x_k variables dependientes de G y sean x_{k+1},……, x_n la variables dependientes de H. Si G($\bar{x}_1$, ……, $\bar{x}_k$) = 0 y H($\bar{x}_{k+1}$,….., $\bar{x}_n$) = 0, entonces F($\bar{x}_1$,……., $\bar{x}_k$, $\bar{x}_{k+1}$,……., $\bar{x}_n$) = 0. Particionar el problema es normalmente muy ventajoso. El número de nodos a chequear en el peor de los casos (para una partición uniforme) va desde 2^n a $2^{n/2+1}$ (para n par).

Encontrar una bipartición es relativamente fácil. Primero se elige un cubo de la cobertura (cualquier cubo sirve), se crea un conjunto C_1. Todas las columnas donde los cubos son 0 o 1 se agregan a C_1. Todos los cubos que se intersectan con cualquiera de los cubos elegidos se seleccionan (deben pertenecer al mismo grupo). El proceso se repite hasta que no se puedan agregar más cubos a C_1. Hay dos resultados posibles, que todas las columnas de la cobertura pertenezcan a C_1. En este caso el segundo conjunto está vacío. El segundo resultado es que las columnas y filas que no pertenecen a C_1 forman el segundo conjunto. Es posible iterar en el segundo conjunto para poder realizar más particiones. Como ejemplo de particionar una cobertura vemos el siguiente ejemplo:

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} 12345 \\ \left[\begin{array}{c} 1\text{–}1\text{—} \\ \text{–}1\text{—}0 \\ 0\text{—}0\text{–} \\ \text{—}01\text{–} \\ \text{———}1 \end{array}\right] \end{array}$$

Inicialmente se selecciona la fila 1 y viendo la cobertura de la figura nos queda inicialmente $C_1 = \{ 1, 3\}$. Escaneando las columnas de C_1 encontramos las filas 3 y 4 . Esto trae la columna 4 a C_1. No hay más adiciones posibles. Por lo tanto hay una bipartición identificada por $C_1 = \{ 1, 3, 4\}$ y $C_2 = \{ 2, 5\}$. Los subproblemas resultantes son:

$$\begin{array}{c} \\ 1 \\ 3 \\ 4 \end{array} \begin{array}{c} 134 \\ \left[\begin{array}{c} 11\text{–} \\ 0\text{–}0 \\ \text{–}01 \end{array}\right] \end{array} \qquad \begin{array}{c} \\ 2 \\ 5 \end{array} \begin{array}{c} 25 \\ \left[\begin{array}{c} 10 \\ \text{–}1 \end{array}\right] \end{array}$$

En este caso, se ve que ninguno de los subproblemas es **tautólogo.**

<u>Ejemplos de comprobación de tautologías.</u>

Nuestro primer ejemplo es para una cobertura con una sola salida como se ve en la siguiente figura:

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} wxyz\ f \\ \left[\begin{array}{c|c} \text{—}1\text{–} & 1 \\ 1\text{—}1 & 1 \\ \text{–}0\text{–}0 & 1 \\ \text{–}00\text{–} & 1 \end{array}\right] \end{array}$$

En este caso podemos ver que la cobertura es una función **unate** en **w** y en **x.** Calculando los cofactores respecto de estas dos variables nos queda:

$$\begin{array}{c} \\ 1 \end{array} \begin{array}{c} yz\ f \\ \left[\begin{array}{c|c} 1\text{–} & 1 \end{array}\right] \end{array}$$

Se nota en esta tabla final que esta cobertura **no es una tautología**. Por ende la cobertura de la **primer tabla tampoco.**

Como segundo ejemplo vemos una función de salida múltiple. La cobertura de salida múltiple es **tautóloga** si todas las **salidas son tautólogas.** Veamos la siguiente tabla:

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} wxyz\ fg \\ \left[\begin{array}{c} 1\text{–}01|01 \\ \text{–}110|10 \\ 1110|11 \\ \text{——}0|10 \\ \text{——}1|11 \end{array}\right] \end{array}$$

En este caso la cobertura es **unate** en **"w"** y en **"x"**. Factorizando obtenemos:

$$\begin{array}{c} \\ 4 \\ 5 \end{array} \begin{array}{c} yz\ fg \\ \left[\begin{array}{c} \text{–}0|10 \\ \text{–}1|11 \end{array}\right] \end{array}$$

Ahora podemos deshacernos de "y" porque la cobertura ya no depende de esta variable. Entonces la única variable que queda es "z". Si analizamos el cofactor positivo, el único cubo a la izquierda es el 5. Como este cubo tiene las dos salidas en "1", podemos decir que este semi espacio es tautólogo. Sin embargo para z = 0, del lado izquierdo nos quedamos solo con el cubo 4, para el cual la función g es cero. Por lo tanto no es una tautología y toda la cobertura no es tautóloga.

Cuando expandimos un cubo reemplazando un cero o un uno con una redundancia en una posición correspondiente a la variable x_i, nosotros decimos que estamos expandiendo el cubo en la dirección x_i. Del mismo modo obramos para la reducción. A menudo se da el caso que un cubo se puede expandir en más de una dirección, pero no en todas las direcciones simultáneamente. La elección de una dirección u otra, puede influir en la calidad de la solución, y por lo tanto, es importante poder decidir en qué direcciones es probable que se den mejores resultados. Si procedemos al azar es posible que reexpandamos en una dirección y existe la posibilidad de que regresemos a la cobertura inicial. Como objetivo secundario, es posible que queramos obtener el cubo expandido más grande, por lo tanto minimizando el número de literales. En ambos casos tener una representación explicita de los ceros de la función ayuda. Entonces veremos brevemente el caso de la complementación de la cobertura.

Complementación recursiva.

No debería sorprender que un algoritmo de complementación eficiente pueda estar basado en el teorema de expansión. Veremos la siguiente definición:

Para una **función booleana f,** se verifica que:

$$\bar{f} = x \cdot \bar{f}_x + \bar{x} \cdot \bar{f}\,\bar{x}$$

Podemos probar que esta función g = $\bar{f}$ es **complementaria de** f. Para probar esto es suficiente comprobar que **f . g = 0** y que **f + g = 1**. Esto se establece fácilmente expandiendo la **función f** respecto de **x** y sustituyendo dentro de **f . g** y dentro de **f + g**.

Cuando el subproblema se reduce a un solo cubo, simplemente podemos aplicar las leyes de De Morgan para obtener el complemento. Como normalmente queremos

tener el tamaño de los cubos lo más pequeño posible, lo que hacemos habitualmente es chequear los cubos que aparecen en cada cofactor y combinarlos dentro de uno (x . c + $\overline{x}$. c = c).

Como ejemplo consideramos la siguiente cobertura.

xyz	f
10–	1
110	1
0–1	1

Primero consideramos el cofactor positivo respecto de "x", quedando:

yz	f_x
0–	1
10	1

Ahora dividimos respecto de "y" y obtenemos f_{xy} y $f_{xy'}$. Estos cofactores son simples y suficientes para poder encontrar el complemento por inspección:

z	f_{xy}
0	1

z	f'_{xy}
1	1

z	$f_{xy'}$
–	1

z	$f'_{xy'}$

Vemos que $\bar{f}_{xy'} = 0$. Ahora podemos formar $\bar{f}_x$

yz	f'_x
11	1

Ahora calculamos $f_{x'}$ y su complemento:

yz	$f_{x'}$
–1	1

yz	$f'_{x'}$
–0	1

Finalmente, combinamos los dos resultados parciales dentro del complemento de f:

xyz	f'
111	1
0–0	1

Al igual que en la verificación de tautologías, podemos explorar las propiedades de las funciones unate. Especialmente, podemos probar que f es monótona creciente respecto de "x" si se cumple lo siguiente:

$$f' = f'_x + x' \cdot f'_{x'}$$

Y para f si fuera monótona decreciente se puede probar que:

$$f' = x \cdot f'_x + f'_{x'}.$$

Se puede acelerar el procedimiento para complementar funciones unate. Para ello vemos el siguiente ejemplo:

wxyz
11—
1—
–1–1
–11–
–111

Como la función es unate, adoptamos una estrategia especial de selección de variables. Nos damos cuenta que si seleccionamos "w", una fila de todos guiones aparecerá en f_w. Por lo tanto seleccionar "w" es bueno. Nosotros sabemos que $\bar{f}_w = 0$.
Necesitamos calcular $\bar{f}_{w'}$. Factorizando f, nos queda:

xyz
1–1
11–
111

Vemos que esta cobertura se puede escribir como x . G, donde G es:

yz
–1
1–
11

Aplicando De Morgan, $\bar{f}_{w'} = \bar{x} + \bar{G}$, o sea nos queda:

xyz
0—
–00

Llegando al siguiente resultado final:

wxyz
00—
0–00

Identificación de Implicantes Primos Esenciales (IPE):

Es ventajoso identificar los implicantes primos esenciales, porque sabemos que serán parte de la solución óptima. Curiosamente, después de una expansión inicial, todos los cubos en la cobertura son primos y, por lo tanto, todos los implicantes esenciales están presentes en la cobertura. Si podemos identificarlos, entonces podemos dejarlos de lado y así evitar reducirlos al querer expandir en una dirección diferente.

Si tuviéramos todos los primos de la función, podríamos saber fácilmente si un primo es esencial, verificando si está cubierto por la unión de otros primos. Cuando no tenemos todos los Implicantes Primos (IP), la identificación de los Implicantes Primos Esenciales (IPE) se basa en el siguiente teorema:

Teorema. Sea **F** una cobertura compuesta por implicantes primos. Sea **e** uno de los implicantes primos en F y sea **G** una cobertura compuesta por los implicantes primos restantes. Entonces **e** es un **implicante primo esencial** si y solo si no está cubierto por la unión de:

- **Los términos de consenso de e y cada elemento de G.**
- **La intersección de e con cada término de G.**

Como ejemplo, consideramos lo siguiente:

$$y'z' + xy' + xz$$

Y probamos con $\bar{x}.\bar{z}$. Este implicante primo intersecta a x.$\bar{y}$ (la intersección es x.$\bar{y}.\bar{z}$). Y también tiene un término consensuado con x.z (x.$\bar{y}$). Para ver si $\bar{y}.\bar{z}$ es esencial, debemos chequear si $\bar{y}.\bar{z} \leq$ x.$\bar{y}$ + x.$\bar{y}.\bar{z}$. Ya que x. $\bar{y}.\bar{z} \leq$ x.$\bar{y}$, nosotros solo necesitamos chequear si $\bar{y}.\bar{z} \leq$ x.$\bar{y}$. La respuesta es claramente que no, entonces $\bar{y}.\bar{z}$ es esencial. Si ahora consideramos x.$\bar{y}$, necesitamos chequear si x.$\bar{y}$ está cubierto por x.$\bar{y}.\bar{z}$ + x.$\bar{y}.\bar{z}$. La respuesta en este caso es positiva y por lo tanto x.$\bar{y}$ no es esencial.

Se puede vislumbrar más la utilidad de este método para identificar los implicantes primo esenciales en nuestro próximo ejemplo:

$$f = x'z' + x'y + xz.$$

Esta cobertura no es una suma completa. Por lo tanto no es inmediato ver si un implicante primo es esencial. Supongamos que queremos probar con $\bar{x}$.y. Su intersección con $\bar{x}.\bar{z}$ es $\bar{x}$.y.$\bar{z}$, mientras su término de consenso con x.z es y.z. Por lo tanto probamos si:

$$x'y \leq x'yz' + yz.$$

Factorizando el lado derecho de la desigualdad, obtenemos $\bar{z}$ + z, que claramente es una tautología. Por lo tanto $\bar{x}$.y no es esencial.

Comentario: Todo lo visto anteriormente es para interpretar como se usa la minimización heurística cuando las funciones son muy grandes, o sea con muchas variables de entrada y ocasionalmente, de salida. Obviamente que estos recursos no se usan a mano sino en programas computacionales. Lo visto es para comprender como funcionan estos softwares.

Sin embargo, vamos a ver algunos ejemplos simples de algunos temas vistos en esta unidad.

Por ejemplo, vamos a usar la **complementación recursiva** para hallar la función inversa de la ecuación que representa la siguiente tabla de verdad:

A	B	C	Z
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Como se desprende fácilmente de la tabla la función que representa es una EX – NOR.

$$F = \bar{A}.\bar{B} + A.B = \overline{A \oplus B}$$

Para resolver el problema tenemos que usar la siguiente fórmula vista anteriormente:

$$\bar{f} = x \,.\, \bar{f}_x + \bar{x} \,.\, \bar{f}\,\bar{x}$$

Calculamos los cofactores:

$$F_A = B$$
$$F_{A'} = \bar{B}$$
$$\bar{F}_A = \bar{B}$$
$$\bar{F}_{A'} = B$$

Reemplazando en el formula de complementación queda:

$$\bar{F} = A.\bar{B} + \bar{A}.B$$

Reescribiendo la tabla de verdad para esta función negada nos queda:

A	B	C	Z
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Como vemos la función que representa esta tabla es una EX – OR:

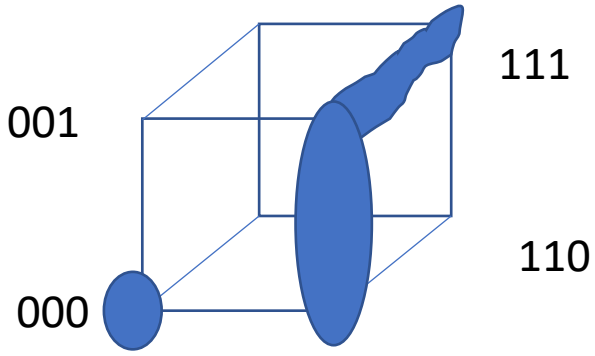
$$\bar{F} = A.\bar{B} + \bar{A}.B = A \oplus B$$

Como vemos es la función negada de la anterior dada.

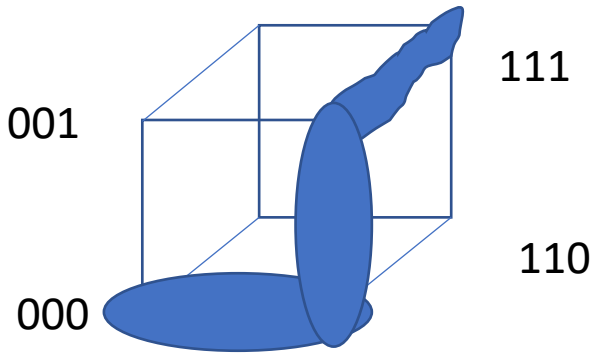
Importante: Todos los métodos de simplificación heurísticos se basan en expansiones y reducciones sucesivas como vimos. Simplemente como ejemplo muy básico de que mencionamos anteriormente. Vemos la siguiente ecuación:

$$F = \bar{A}.\bar{B}.\bar{C} + B.C + \bar{A}.B$$

Su representación cúbica es:



Expandiendo el minitermino 0 (ABC = 000) al minitérmino 3 (ABC = 011), nos queda:



Ahora vemos que el implicante primo formado por los minitérminos 2 y 3 no es esencial y lo descartamos, quedando:

La ecuación final queda:

$$F = \bar{A}.\bar{C} + B.C$$

Desde el punto de vista de la manipulación de la función original vemos que:

$$F = \bar{A}.\bar{B}.\bar{C} + B.C + \bar{A}.B$$

Expandiendo el término correspondiente al minitérmino 0 hacia el minitérmino 3 nos queda:

$$F = \bar{A}.\bar{C} + B.C + \bar{A}.B$$

Ahora se ve que el término de consenso entre el primer y el segundo término es justamente el tercero. Por lo tanto, lo eliminamos. Como se visualiza llegamos al mismo resultado.

Como ejemplo veremos diseñar un comparador de dos palabras (A y B) de tres bits c/u. El circuito dará un 1 a la salida cuando A sea mayor que B (A > B). Llamamos a los bits de cada palabra de la siguiente manera:

$$A = A_2A_1A_0 \text{ y } B = B_2B_1B_0$$

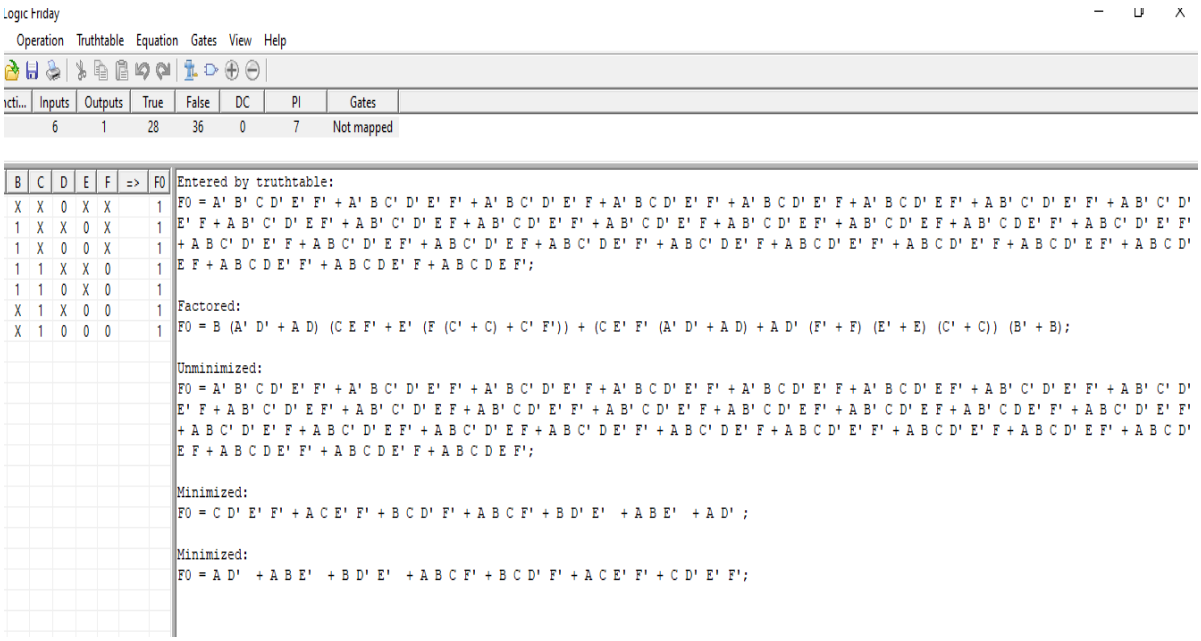
Una tabla de verdad para todas las posibilidades nos daría 64 filas ya que tenemos 6 variables independientes. Si pensamos lógicamente en qué condiciones se tienen que cumplir para que A sea mayor que B, nos quedan tres alternativas. Ellas son:

$$A_2 = B_2 \ \wedge \ A_1 = B_1 \ \wedge \ A_0 > B_0$$
$$\vee$$
$$A_2 = B_2 \ \wedge \ A_1 > B_1$$
$$\wedge$$
$$A_2 > B_2$$

Esto puesto en lógica proposicional cuando lo llevamos a una ecuación booleana nos queda:

$$F = (\overline{A2 \oplus B2}).(\overline{A1 \oplus B1}).A0.\overline{B0} + (\overline{A2 \oplus B2}).A1.\overline{B1} + A2.\overline{B2}$$

Si este problema lo hacemos con un programa de simplificación, tipo Logic Friday nos queda una pantalla como la siguiente:



Donde se aprecia que nos lista los minitérminos de la función, después da una versión factorizada de la misma y por último la función minimizada. Este programa arma la tabla de verdad y lo único que hay que cargarle es el valor de las salidas.

Para terminar, veremos un ejemplo donde diseñamos en forma de lógica proposicional y con la tabla de verdad con simplificación en dos niveles. Tomaremos un ejemplo muy sencillo donde haremos el mismo comparador anterior, pero para solo palabras de dos bits cada una, o sea dadas dos palabras de dos dígitos binarios cada una: $A = A_1A_0$ y $B = B_1B_0$. Determinaremos cuando A > B. Empezaremos por escribir la tabla de verdad de este circuito:

A1	A0	B1	B0	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

B1B0 \ A1A0	00	01	11	10
00	0	1	1	1
01	0	0	1	1
11	0	0	0	0
10	0	0	1	0

Acá vemos la tabla de verdad y el mapa de Karnaugh de la especificación dada, como se ve hay tres IP, los cuales son todos esenciales, por lo tanto, la función mínima queda:

$$F = A1.\overline{B1} + A0.\overline{B1}.\overline{B0} + A1.A0.\overline{B0} \qquad \text{(ecuación 1)}$$

Si usamos la lógica proposicional tenemos dos condiciones que satisfacen el problema, ellas son:

$$A_1 = B_1 \ \wedge \ A_0 > B_0$$

$$\vee$$

$$A_1 > B_1$$

Escribiendo en forma de ecuación estas dos condiciones nos queda:

$$F = (\overline{A1 \oplus B1}).A0.\overline{B0} + A1.\overline{B1} \quad \text{(ecuación 2)}$$

Aplicando teoremas del algebra de Boole a esta última ecuación nos queda:

$$F = (\overline{A1}.\overline{B1} + A1.B1).A0.\overline{B0} + A1.\overline{B1} =$$

$$= \overline{A1}.A0.\overline{B1}.\overline{B0} + A1.A0.B1.\overline{B0} + A1.\overline{B1} =$$

El consenso entre el primer término y el tercero no da la siguiente ecuación:

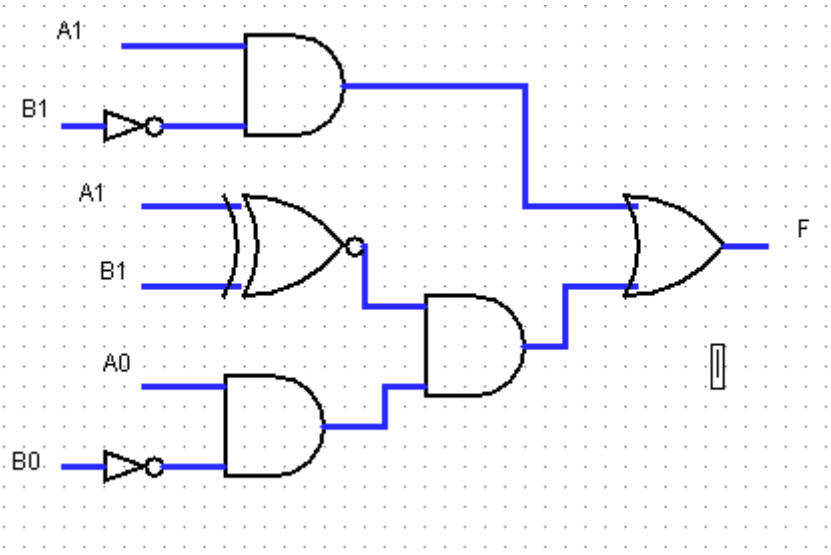
$$= \overline{A1}.A0.\overline{B1}.\overline{B0} + A1.A0.B1.\overline{B0} + A1.\overline{B1} + A0.\overline{B1}.\overline{B0} =$$

Este último término absorbe al primero, haciendo lo mismo entre el término segundo y el tercero queda la ecuación final:

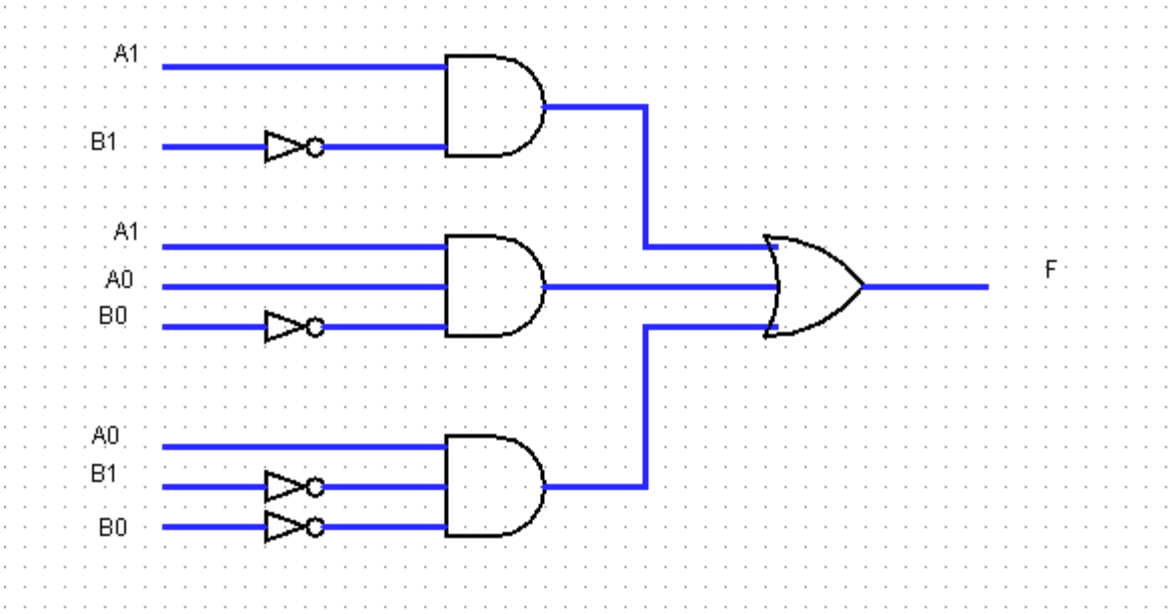
$$F = A1.\overline{B1} + A0.\overline{B1}.\overline{B0} + A1.A0.\overline{B0}$$

Vemos que por los dos métodos llegamos a lo mismo, lo que demostraremos ahora es la diferencia entre implementar la función desde el punto de vista proposicional y desde el punto de vista de la tabla de verdad. Esto nos dará un pantallazo de la ya visto sobre la diferencia entre implementar en dos niveles y la variación que se produce cuando implementamos con más de dos niveles.

En la siguiente figura vemos la implementación con compuertas de la ecuación 2:



La misma implementación en dos niveles nos queda:



La implementación proposicional necesita una compuerta menos, pero es más lenta (tiene mayor retardo de propagación), este hecho se ve reflejado si el número de compuertas crece.