Universidad de Huelva

Tema 2

Circuitos lógicos

Índice

2.1 Funciones sobre conjuntos finitos

2.2 Programación lineal

2.3 Circuitos lógicos

2.4 Formas normales

2.5 Circuitos espacializados

2.6 Límites en tamaño y profundidad

2.1 Funciones sobre conjuntos finitos

2.2 Programación lineal

2.3 Circuitos lógicos

2.4 Formas normales

2.5 Circuitos espacializados

2.6 Límites en tamaño y profundidad

- Computar es calcular funciones en base a la combinación de operaciones elementales.
- La teoría clásica de la Computación se centra en el cálculo de funciones sobre números naturales.

$$f: \mathbb{N} \rightarrow \mathbb{N}$$

- En realidad, este estudio abarca el cálculo de funciones sobre conjuntos numerables ($A \leftrightarrow \mathbb{N}$).
- La correspondencia entre el conjunto A y $\mathbb{N}$ representa una codificación del conjunto A.
- Un caso particular se refiere al cálculo de funciones sobre conjuntos finitos.

- Para trabajar sobre conjuntos finitos vamos a utilizar una codificación binaria.
- El número de bits necesarios para codificar N elementos es

$$n = \lceil \log_2(N) \rceil$$

- De esta forma se transforma una variable x, definida sobre el conjunto de N elementos, en n variables x_i, definidas sobre el conjunto B={0,1}
- Las funciones sobre conjuntos finitos se pueden expresar como funciones binarias

$$f : B^n \rightarrow B^m$$

- Estas funciones pueden tratarse también como m funciones booleanas independientes $f : B^n \rightarrow B$

2.1 Funciones sobre conjuntos finitos

2.2 Programación lineal

2.3 Circuitos lógicos

2.4 Formas normales

2.5 Circuitos espacializados

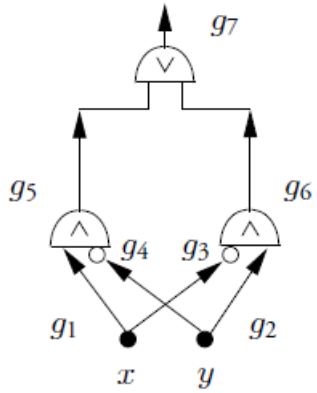
2.6 Límites en tamaño y profundidad

- Un programa lineal es una secuencia de instrucciones del tipo
  - Entrada: (s READ x)
  - Salida: (s OUTPUT i)
  - Operación: (s OP i … k)

  donde *s* indica el número de instrucción, *x* es una variable de entrada, OP es una operación elemental, *i … k* representan los resultados de las instrucciones i-ésima … k-ésima y se cumple que s > i … k.
- Se denomina ***base de un programa lineal*** al conjunto de operaciones elementales que utiliza.
- Para representar funciones binarias se utilizan programas lineales booleanos, que están formados por la base { AND, OR, NOT }

- La representación gráfica de un programa lineal consiste en dibujar un grafo en el que cada instrucción es un nodo (una puerta) y los operandos de ese nodo se representan mediante arcos.
- El resultado es un grafo dirigido acíclico. (La condición s > i…k garantiza la inexistencia de bucles).
- La programación lineal no está limitada a la representación de circuitos. Se puede extender la base del programa al uso de operaciones algebraicas (sumas, multiplicaciones, …), lo que da lugar a algunas propiedades diferentes de las que presentaremos en este tema.

- Ejemplo



- Programa:

```
(1 READ x)
(2 READ y)
(3 NOT 1)
(4 NOT 2)
(5 AND 1 4)
(6 AND 3 2)
(7 OR 5 6)
(8 OUTPUT 7)
```

2.1 Funciones sobre conjuntos finitos

2.2 Programación lineal

2.3 Circuitos lógicos

2.4 Formas normales

2.5 Circuitos espacializados

2.6 Límites en tamaño y profundidad

- Un circuito lógico es un grafo dirigido acíclico formado por puertas lógicas conectadas entre sí.
- La base Ω de un circuito lógico se refiere a los tipos de puertas lógicas incluidas en el circuito.
- El comportamiento de las puertas lógicas (y el de los circuitos completos) se puede representar mediante tablas de verdad.

a	b	a ∧ b
0	0	0
0	1	0
1	0	0
1	1	1

a	b	a ∨ b
0	0	0
0	1	1
1	0	1
1	1	1

a	b	a ⊕ b
0	0	0
0	1	1
1	0	1
1	1	0

a	ā
0	1
1	0

- El problema de la computabilidad de funciones binarias consiste en, dada una función $f : B^n \rightarrow B^m$, construir un circuito lógico con base Ω que reproduzca su tabla de verdad.
- La tabla de verdad de la función $f : B^n \rightarrow B^m$ tendrá n variables de entrada, m variables de salida y 2^n filas.

x0	x1	x2	y1	y2
0	0	0	1	1
0	0	1	0	1
0	1	0	1	0
0	1	1	0	1
1	0	0	1	1
1	0	1	1	0
1	1	0	0	1
1	1	1	1	1

- Se define el ***tamaño de un circuito*** como el número de puertas lógicas que contiene.
- Se define la ***profundidad de un circuito*** como el camino más largo que contiene el circuito.
- Se define el ***tamaño de circuito de una función booleana***, $C_{\Omega}(f)$, como el tamaño mínimo del circuito lógico que reproduce la función, construido sobre la base Ω.
- Se define la ***profundidad de circuito de una función booleana***, $D_{\Omega}(f)$, como la profundidad mínima del circuito lógico que reproduce la función, construido sobre la base Ω.

Propiedades

- Conmutatividad:

$$a \wedge b = b \wedge a \qquad a \vee b = b \vee a \qquad a \oplus b = b \oplus a$$

- Sustitución de constantes:

$$a \wedge 0 = 0 \qquad a \vee 0 = a \qquad a \oplus 0 = a$$

$$a \wedge 1 = a \qquad a \vee 1 = 1 \qquad a \oplus 1 = \bar{a}$$

- Reglas de absorción:

$$a \wedge a = a \qquad a \vee a = a \qquad a \oplus a = 0$$

$$a \wedge \bar{a} = 0 \qquad a \vee \bar{a} = 1 \qquad a \oplus \bar{a} = 1$$

$$a \wedge (a \vee b) = a \qquad a \vee (a \wedge b) = a$$

Propiedades

- Reglas de De Morgan:

$$\overline{(a \wedge b)} = \bar{a} \vee \bar{b} \qquad \overline{(a \vee b)} = \bar{a} \wedge \bar{b}$$

- Asociatividad:

$$a \wedge (b \wedge c) = (a \wedge b) \wedge c$$

$$a \vee (b \vee c) = (a \vee b) \vee c$$

$$a \oplus (b \oplus c) = (a \oplus b) \oplus c$$

- Distributividad:

$$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$$

$$a \wedge (b \oplus c) = (a \wedge b) \oplus (a \wedge c)$$

$$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$$

2.1 Funciones sobre conjuntos finitos

2.2 Programación lineal

2.3 Circuitos lógicos

2.4 Formas normales

2.5 Circuitos espacializados

2.6 Límites en tamaño y profundidad

- Una forma normal es una descripción de una función por medio de un circuito con una estructura estandarizada.
- Formas normales más conocidas:
  - Forma normal disyuntiva
  - Forma normal conjuntiva
  - Forma normal como suma de productos (SOPE)
  - Forma normal como producto de sumas (POSE)
  - Ring-sum expansion (RSE)

- Forma normal disyuntiva
  - Un ***mintérmino*** de las variables $x_1 \ldots x_n$ es una operación AND de cada variable o su negación.

    Por ejemplo, (a ∧ ē ∧ i ∧ ō ∧ ū) es un mintérmino de (a,e,i,o,u)
  - Por simplicidad, eliminaremos los símbolos ∧ de la notación. (a ē i ō ū)
  - Un **mintérmino *c* de una función booleana *f*** es un mintérmino que contiene todas las variables de *f* y para el que *f*(*c*)=1.
  - La ***forma normal disyuntiva*** (DNF) de una función booleana *f* es la operación OR de todos sus mintérminos.

- Ejemplo de forma normal disyuntiva (DNF)

x0	x1	x2	y1
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

$$f(x_0, x_1, x_2) = \bar{x}_0\bar{x}_1\bar{x}_2 \vee \bar{x}_0 x_1\bar{x}_2 \vee x_0\bar{x}_1\bar{x}_2 \vee x_0\bar{x}_1 x_2 \vee x_0 x_1 x_2$$

- Forma normal conjuntiva
  - Un ***maxtérmino*** de las variables $x_1 \ldots x_n$ es una operación OR de cada variable o su negación.

    Por ejemplo, $(a \vee \bar{e} \vee i \vee \bar{o} \vee \bar{u})$ es un maxtérmino de (a,e,i,o,u)

    El maxtérmino $(a \vee \bar{e} \vee i \vee \bar{o} \vee \bar{u})$ es siempre 1 salvo para (0,1,0,1,1).
  - Un **maxtérmino *c* de una función booleana *f*** es un maxtérmino que contiene todas las variables de *f* y para el que $f(c)=0$.
  - La ***forma normal conjuntiva*** (CNF) de una función booleana *f* es la operación AND de todos sus maxtérminos.

- Ejemplo de forma normal conjuntiva (CNF)

x_0	x_1	x_2	y_1
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

$$f(x_0, x_1, x_2) = (x_0 \vee x_1 \vee \bar{x}_2) \wedge (x_0 \vee \bar{x}_1 \vee \bar{x}_2) \wedge (\bar{x}_0 \vee \bar{x}_1 \vee x_2)$$

- Sum of products expansion (SOPE)
  - Un ***producto*** de las variables $x_1 \ldots x_n$ es una operación AND de cada variable o su negación donde no necesariamente aparecen todas las variables.
  - Un mintérmino es un tipo especial de producto en el que aparecen todas las variables.
  - A partir de la forma normal disyuntiva se puede obtener una representación como suma de productos simplificando los mintérminos

$$f(x_0, x_1, x_2) = \bar{x}_0\bar{x}_1\bar{x}_2 \vee \bar{x}_0 x_1\bar{x}_2 \vee x_0\bar{x}_1\bar{x}_2 \vee x_0\bar{x}_1 x_2 \vee x_0 x_1 x_2 = \bar{x}_0\bar{x}_2 \vee x_0\bar{x}_1 \vee x_0 x_2$$

- Product of sums expansion (POSE)
  - Una ***suma*** de las variables $x_1 \ldots x_n$ es una operación OR de cada variable o su negación donde no necesariamente aparecen todas las variables.
  - Un maxtérmino es un tipo especial de producto en el que aparecen todas las variables.
  - A partir de la forma normal conjuntiva se puede obtener una representación como suma de productos simplificando los maxtérminos

$$f(x_0, x_1, x_2) = (x_0 \vee x_1 \vee \bar{x}_2) \wedge (x_0 \vee \bar{x}_1 \vee \bar{x}_2) \wedge (\bar{x}_0 \vee \bar{x}_1 \vee x_2) = (x_0 \vee \bar{x}_2) \wedge (\bar{x}_0 \vee \bar{x}_1 \vee x_2)$$

- Ring-sum expansion (RSE)
  - La RSE de una función *f* está formada por operaciones EXCLUSIVE-OR de una constante (0 o 1) y de productos de las variables de *f* sin negaciones.
  - Se puede obtener la RSE de una función a partir de su forma normal disyuntiva de la siguiente forma.
    1. Sustituir los operadores OR por EXCLUSIVE-OR en la forma normal disyuntiva. Esto se puede hacer ya que dos mintérminos nunca podran valer 1 al mismo tiempo.
    2. Sustituir las variables negadas ($\bar{a}$) por ($a \oplus 1$).
    3. Aplicar las reglas de conmutatividad, distributividad y absorción para simplificar el resultado.

- Ring-sum expansion (RSE)

$$f(x_0,x_1,x_2)=\bar{x}_0\bar{x}_1\bar{x}_2\vee\bar{x}_0x_1\bar{x}_2\vee x_0\bar{x}_1\bar{x}_2\vee x_0\bar{x}_1x_2\vee x_0x_1x_2$$

$$f(x_0,x_1,x_2)=\bar{x}_0\bar{x}_1\bar{x}_2\oplus\bar{x}_0x_1\bar{x}_2\oplus x_0\bar{x}_1\bar{x}_2\oplus x_0\bar{x}_1x_2\oplus x_0x_1x_2$$

$$\begin{aligned} f(x_0,x_1,x_2) \;=\; & (x_0\oplus 1)(x_1\oplus 1)(x_2\oplus 1)\\ \oplus\; & (x_0\oplus 1)x_1(x_2\oplus 1)\\ \oplus\; & x_0(x_1\oplus 1)(x_2\oplus 1)\\ \oplus\; & x_0(x_1\oplus 1)x_2\\ \oplus\; & x_0x_1x_2 \end{aligned}$$

- Ring-sum expansion (RSE)

$$
\begin{aligned}
f(x_0,x_1,x_2) \;=\; & x_0x_1x_2 \oplus x_0x_1 \oplus x_0x_2 \oplus x_1x_2 \oplus x_0 \oplus x_1 \oplus x_2 \oplus 1 \\
\oplus\; & x_0x_1x_2 \oplus x_0x_1 \oplus x_1x_2 \oplus x_1 \\
\oplus\; & x_0x_1x_2 \oplus x_0x_1 \oplus x_0x_2 \oplus x_0 \\
\oplus\; & x_0x_1x_2 \oplus x_0x_2 \\
\oplus\; & x_0x_1x_2
\end{aligned}
$$

$$f(x_0,x_1,x_2) = x_0x_1x_2 \oplus x_0x_1 \oplus x_0x_2 \oplus x_2 \oplus 1$$

- Comparación entre formas normales
  - Las funciones pueden tener una representación muy compacta en una forma normal y representaciones mucho mayores en otras formas normales.
  - Por ejemplo, la función de paridad tiene tamaño exponencial en DNF y CNF y tamaño lineal en RSE. Por su parte, la función OR sobre n variables tiene tamaño lineal en CNF y exponencial en RSE.
  - Hay funciones que tienen tamaño exponencial en todas las formas normales, por ejemplo la función (x mod 3 == 0). Sin embargo se puede obtener una representación lineal en una forma no normal.

2.1 Funciones sobre conjuntos finitos

2.2 Programación lineal

2.3 Circuitos lógicos

2.4 Formas normales

2.5 Circuitos espacializados

2.6 Límites en tamaño y profundidad

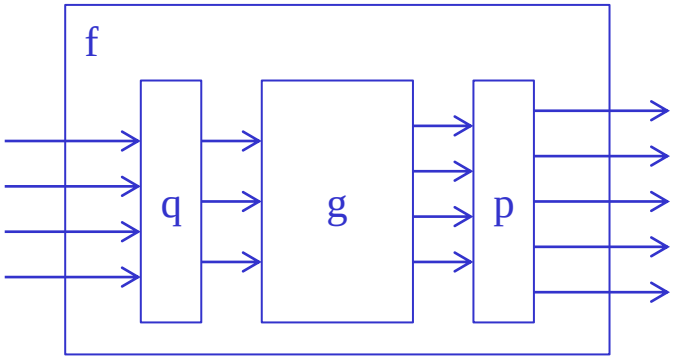
- Reducción entre funciones
  - Una función $f : A^n \rightarrow A^m$ es una ***reducción*** de la función $g : A^r \rightarrow A^s$ a través de las funciones $p : A^s \rightarrow A^m$ y $q : A^n \rightarrow A^r$ si,

$$\forall x \in A^n,\ f(x) = p\ (\ g\ (\ q\ (\ x\ )\ )\ )$$

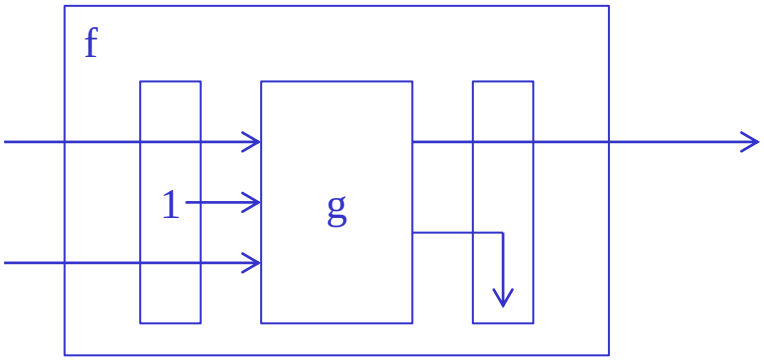
  - El tamaño y profundidad de f satisface las siguientes ecuaciones:

$$C(f) \leq C(p) + C(g) + C(q)$$

$$D(f) \leq D(p) + D(g) + D(q)$$



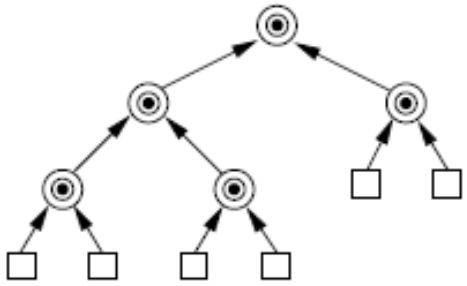
- Subfunciones
  - Una función *f* es una ***subfunción*** de la función *g* si *f* puede ser obtenida a partir de *g* asignando valores constantes a variables de entrada de *g*, reordenando sus variables de entrada o eliminando variables de dalida.



- Operaciones vectoriales
  - Una operación vectorial consiste en aplicar una operación lógica a un vector de variables de entrada:

$$x_0 \otimes x_1 \otimes x_2 \otimes x_3 \otimes x_4 \otimes \ldots \otimes x_n$$

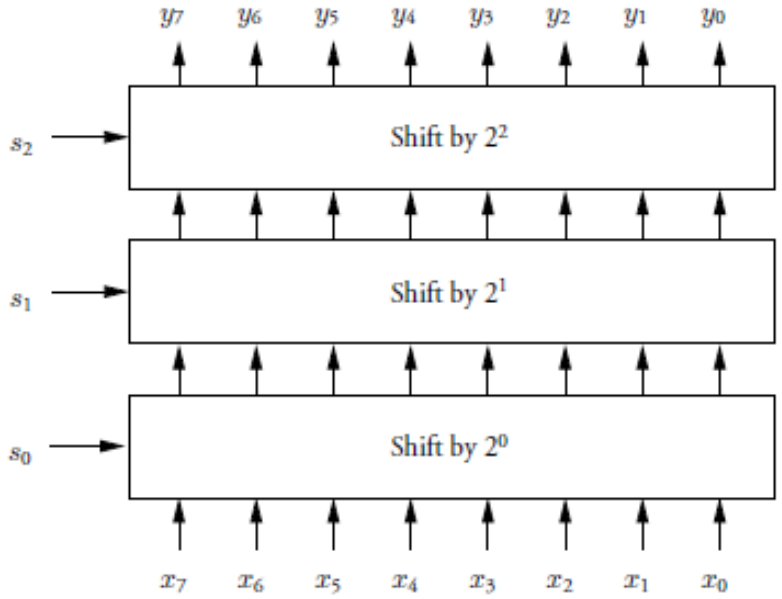
  - Por ejemplo, un mintérmino o un maxtérmino.
  - Un circuito eficiente para desarrollar estas funciones es un árbol binario balanceado.
  - Considerando la base $\Omega = \{\otimes\}$ el tamaño del circuito es $C_\Omega = (n-1)$ y su profundidad $D_\Omega = \lceil \log_2 n \rceil$



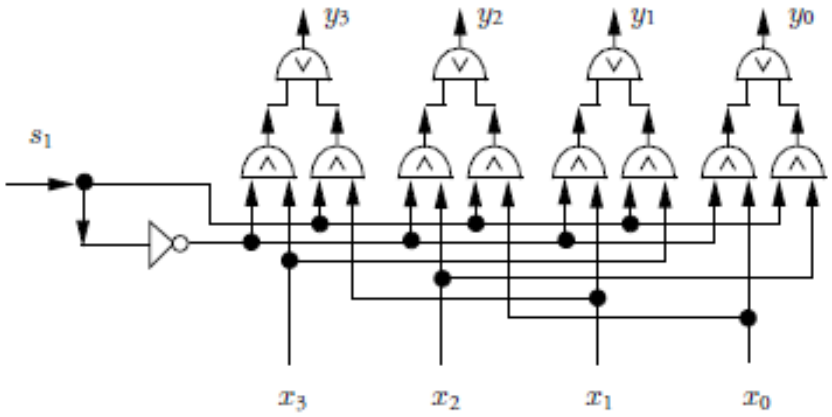
- Funciones de desplazamiento
  - Una ***operación de desplazamiento cíclico*** de tamaño s sobre una n-tupla $\boldsymbol{x}$ consiste en desplazar cíclicamente sus componentes s lugares a la izquierda. Por ejemplo, f(2, {a, b, c, d}) = {c, d, a, b}
  - Para poder realizar todos los desplazamientos posibles es necesario expresar s con $\lceil \log_2 n \rceil$ bits.

$$f^{(n)}_{cyclic}: B^{n+\lceil \log 2\, n \rceil} \rightarrow B^n$$

- Funciones de desplazamiento
  - El circuito que desarolla la función de desplazamiento cíclica se puede dividir en etapas relacionadas con los bits de s.
  - La primera etapa, asociada a s_0,realiza un desplazamiento de 1 bit (2^0). La segunda etapa, asociada a s_1, realiza un despazamiento de 2 bits (2^1). La tercera etapa , asociada a s_2, realiza un desplazamiento de 4 bits (2^2).



- Funciones de desplazamiento
  - Cada etapa puede desarrollarse mediante un circuito como:



  - Considerando la base $\Omega = \{\wedge,\vee,\neg\}$, el tamaño y profundidad de la función cumplen que:

$$C_{\Omega}(f) \leq (3 \cdot n+1) \lceil \log_2 n \rceil$$

$$D_{\Omega}(f) \leq 3 \lceil \log_2 n \rceil$$

- Funciones de desplazamiento
  - La ***función de desplazamiento lógico*** $f^{(n)}_{shift}$ consiste en desplazar los componentes de una n-tupla hacia la izquierda rellenando con 0s.
  - Se puede expresar la función de desplazamiento lógico sobre n bits como una subfunción de la función de desplazamiento cíclico sobre 2·n bits, colocando a 0 los n bits más significativos de la entrada y desechando los n bits menos significativos de la salida.
  - Se puede expresar la función de desplazamiento cíclico sobre n bits como una subfunción de la función de desplazamiento lógico sobre 2·n bits, repitiendo la entrada x en los bits más significativos y desechando los n bits menos significativos de la salida.

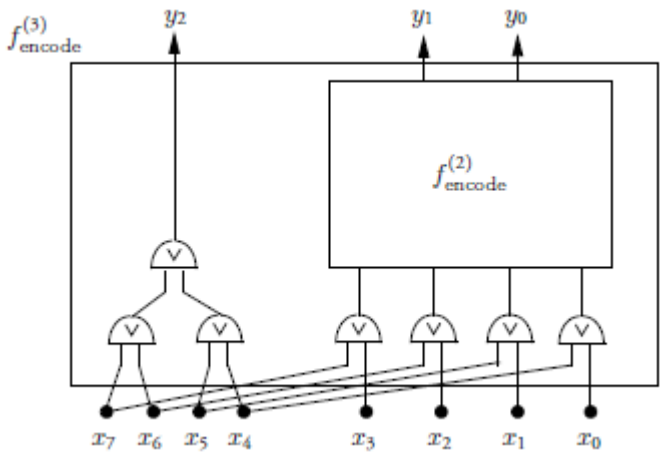
- Codificador
  - Un ***codificador*** o ***encoder*** $f^{(n)}_{encode}$ es una función de 2^n entradas en las que sólo una de ellas valdrá 1. La salida es la codificación binaria de la posición de la entrada activa.
  - Para n=1, $f^{(n)}_{encode}(x_1,x_0) = x_1$, por tanto,

$$C_{\Omega}(f^{(1)}_{encode}) = 0$$

$$D_{\Omega}(f^{(1)}_{encode}) = 0$$

  - Se puede construir el circuito genérico de forma recursiva, considerando que el bit más significativo de la salida sólo está activo cuando la entrada activa se encuentra en la mitad alta de la entrada

- Codificador



– Esta descomposición cumple que

$$C_{\Omega}(f^{(n)}_{encode}) \leq 2^n - 1 + C_{\Omega}(f^{(n-1)}_{encode})$$

$$D_{\Omega}(f^{(n)}_{encode}) \leq \max(n - 1, D_{\Omega}(f^{(n-1)}_{encode}) + 1)$$

– Sustituyendo los valores para n=1,

$$C_{\Omega}(f^{(n)}_{encode}) \leq 2^n - (n + 3)$$

$$D_{\Omega}(f^{(n)}_{encode}) \leq n - 1$$

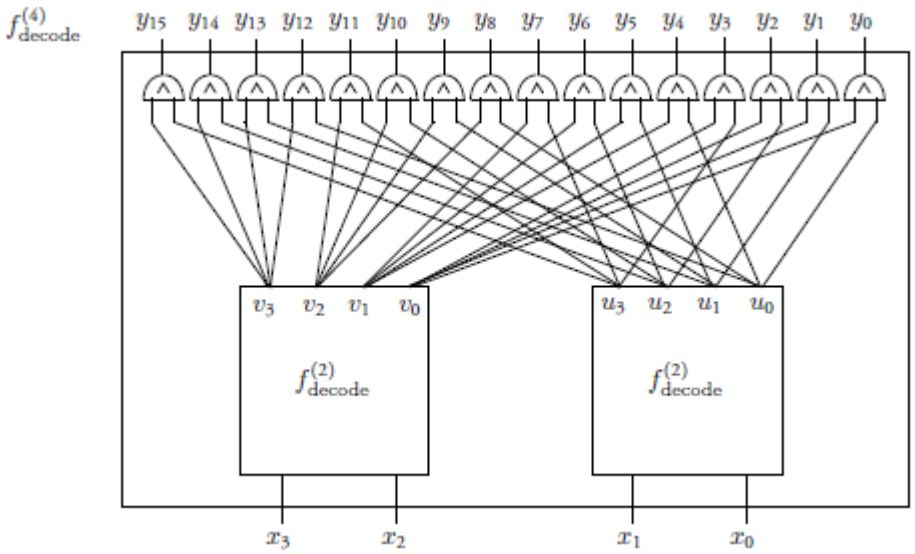
- Decodificador
  - Un ***decodificador*** o ***decoder*** $f^{(n)}_{\text{decode}}$ es una función de n entradas y 2^n salidas. Dado un valor de entrada, todas las salidas tendrán valor 0 excepto la que se encuentre en la posición indicada por la entrada.
  - Una forma de construir este circuito es representar cada salida mediante el mintérmino correspondiente. Para esta construcción se cumple que

$$C_{\Omega}(f^{(n)}_{\text{decode}}) \leq (2n - 1)\, 2^n$$

$$D_{\Omega}(f^{(1)}_{\text{encode}}) \leq \lceil \log_2 n \rceil + 1$$

  - Se puede compactar este circuito si troceamos los mintérminos y evitamos la repetición de trozos.

- Decodificador
  - Se puede compactar este circuito si troceamos los mintérminos y evitamos la repetición de trozos.



  - De esta forma se cumple que, si n es par,

$$2^n \leq C_{\Omega}(f^{(n)}_{\text{decode}}) \leq 2^n + (2n-2)\,2^{n/2}$$

$$D_{\Omega}(f^{(n)}_{\text{decode}}) \leq \lceil \log_2 n \rceil + 1$$

- Multiplexor
  - Un ***multiplexor*** $f^{(n)}_{mux}$ es una función con dos vectores de entrada: $\boldsymbol{x}$ (de n bits) y $\boldsymbol{z}$ (de 2^n bits). El valor de $\boldsymbol{x}$ se trata como una dirección. La salida de la función corresponde al bit de $\boldsymbol{z}$ indicado por la dirección $\boldsymbol{x}$.
  - Es fácil construir un circuito que desarrolle un multiplexor a partir de un decodificador. Un decodificador de n bits genera 2^n salidas. Para construir un multiplexor se añade una operación AND entre cada salida del decodificador y los bits de la entrada z. La salida final del multiplexor es la operación OR entre todas estas conjunciones.
  - De esta forma, el tamaño y profundidad de un multiplexor debe cumplir:

$$C_\Omega(f^{(n)}_{mux}) \leq C_\Omega(f^{(n)}_{decode}) + 2^n + 2^n - 1 \leq 3 \cdot 2^n + (2n-2)\, 2^{n/2} - 1$$

$$D_\Omega(f^{(n)}_{mux}) \leq D_\Omega(f^{(n)}_{decode}) + n + 1 \leq \lceil \log_2 n \rceil + n + 2$$

2.1 Funciones sobre conjuntos finitos

2.2 Programación lineal

2.3 Circuitos lógicos

2.4 Formas normales

2.5 Circuitos espacializados

2.6 Límites en tamaño y profundidad

- Número de funciones booleanas
  - Utilizando n entradas binarias podemos describir 2^n valores de entrada diferentes.
  - A partir de m= 2^n valores de entrada se pueden definir 2^m funciones booleanas distintas.

Bits	Entradas	Funciones
1	2	4
2	4	16
3	8	256
4	16	65.536
5	32	4.294.967.296
6	48	281.474.976.710.656
16	65.536	$2 \cdot 10^{19728}$
32	4.294.967.296	$3{,}1 \cdot 10^{1.292.913.986}$

- Número de funciones computables con G puertas
  - Supongamos que disponemos de G puertas lógicas. ¿Cuántas funciones diferentes podemos generar?
  - Para simplificar el estudio consideraremos que todas las puertas son iguales. Necesitamos una base compuesta por un único tipo de puerta, por ejemplo Ω={ NAND }
  - Las entradas de cada puerta pueden ser cualquiera de las variables de entrada (n) o las salidas de las otras puertas (G-1).
  - Como el comportamiento de la puerta es simétrico respecto a las entradas (conmutatividad), el número de opciones de cada puerta será:

$$N\_opciones = \frac{(n+G-1)\cdot(n+G)}{2}$$

- Número de funciones computables con G puertas
  - El número de opciones para las G puertas será

$$N\_opciones(G) = \left[\frac{(n+G-1)\cdot(n+G)}{2} \right]^{G}$$

  - Como el identificador que le demos a cada puerta no tiene importancia y hay G! formas de ordenar las puertas, el número de funciones que pueden definir está limitado por

$$N(G) \leq \left[\frac{(n+G-1)\cdot(n+G)}{2} \right]^{G} \cdot \frac{1}{G!}$$

  - Se trata de un límite superior ya que muchos de los circuitos generarán la misma función o contendrán ciclos (por lo que no deberían considerarse válidos).

- Número de funciones computables con G puertas
  - Considerando la aproximación del factorial

$$G! \geq G^G e^{-G}$$

  - Sustituyendo

$$N(G) \leq \left[\frac{(n+G-1)\cdot(n+G)}{2}\right]^G \cdot \left[\frac{e}{G}\right]^G \leq \left[\frac{e\cdot(n+G)^2}{2\cdot G}\right]^G$$

- Número de funciones computables con G puertas
  - Si consideramos n = 16, el número de funciones diferentes es de orden $10^{19.728}$

G	límite
10	$4{,}28 \cdot 10^{19}$
100	$1{,}65 \cdot 10^{226}$
1.000	$1{,}13 \cdot 10^{3.147}$
10.000	$3{,}40 \cdot 10^{41.346}$

- Límite en tamaño de las funciones booleanas
  - Toda función booleana puede ser realizada mediante la disjunción de todos sus mintérminos (DNF).
  - El cálculo de los mintérminos se puede realizar mediante un circuito decodificador.
  - La disjunción se realizará, a lo sumo, sobre el conjunto de todos los mintérminos , lo que requeriría 2^n-1 puertas OR distribuidas como arbol binario balanceado con una profundidad máxima n.

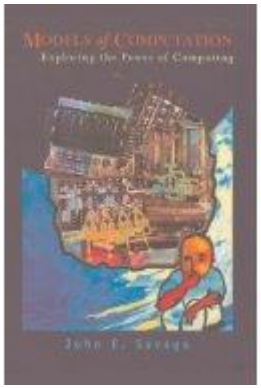
$$C_{\Omega}(f) \leq C_{\Omega}(f^{(n)}_{\text{decode}}) + 2^n - 1 \leq 2^{n+1} + (2n-2)\,2^{n/2} - 1$$

$$D_{\Omega}(f) \leq D_{\Omega}(f^{(n)}_{\text{decode}}) + n \leq n + \lceil \log_2 n \rceil + 1$$

- Límite en tamaño de las funciones booleanas
  - Se puede realizar un estudio más ajustado considerando la descomposición de una función de *n* bits en dos trozos de *k* y *s* bits, lo que se conoce como representación de Lupanov.
  - Según ese estudio, para valores grandes de *n* se verifica que

$$C_{\Omega}(f) \leq O\left(\frac{2^n}{n^2}\right) + O\left(\frac{2^n}{n^3}\right) + \frac{2^n}{n - 5\log_2 n}$$

Bibliografía



- Savage, John E. (1998). "Models Of Computation: Exploring the Power of Computing". Capítulo 2