Capítulo 3 - Álgebra de proposições - Projetos de Circuitos

Objetivos

- Apresentar os conceitos básicos da álgebra de proposições;
- Mostrar as principais propriedades da álgebra de proposições;
- Estudar expressões e circuitos lógicos.

Álgebra de proposições

Em Matemática, chama-se ***proposição*** ao enunciado de uma verdade que se quer demonstrar, ou como usaremos: uma sentença que pode ser falsa (0), ou verdadeira (1), mas nunca ambos ao mesmo tempo.

- Correspondência entre as principais relações e portas lógicas

A conjunção determina que se duas proposições (p e q) forem verdadeiras (1), a conjunção de ambas também o será; basta que uma delas seja falsa (0), para que a conjunção (s) também o seja. A porta **AND** (E) implementa essa relação, pode ter duas (p, q), ou mais entradas, e a saída (s) assumirá o valor 1 se, e somente se, todas as entradas forem iguais a 1; caso uma, ou mais entradas sejam iguais a 0, a saída terá valor 0.

A disjunção determina que se duas proposições (p e q) forem falsas (0), a disjunção de ambas (s) também o será; basta que uma delas seja verdadeira (1), para que a disjunção também o seja. A porta **OR** (OU) implementa essa relação, pode ter duas (p, q), ou mais entradas, e a saída (s) assumirá o valor 0 se, e somente se, todas as entradas forem iguais a 0; caso uma, ou mais entradas forem iguais a 1, a saída terá valor 1.

A negação determina que se uma proposição (p) for falsa (0), a negação (s) será verdadeira (1), ou vice-versa. A porta **NOT** (NÃO) implementa essa relação, é também chamada de **INVERTER** (INVERSOR) e só tem uma entrada (p), e a saída assumirá o valor 1, se a entrada for igual a 0; senão, a saída terá valor 0.

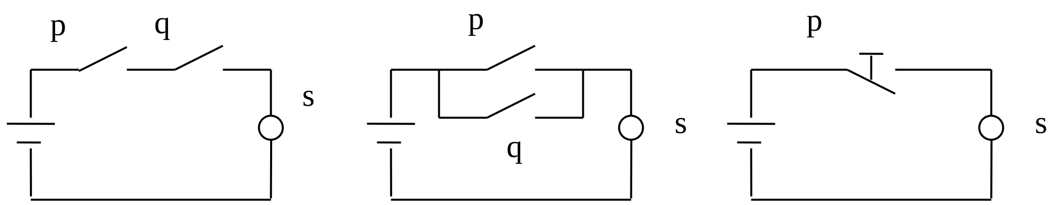
- Analogias com circuitos elétricos

O primeiro circuito a seguir (conjunção) determina que se duas chaves (p e q) forem fechadas (1), o resultado (s) será o de um circuito fechado com uma lâmpada acesa (1), por exemplo; basta que uma delas seja aberta (0), para que o circuito se abra, e a lâmpada apague (0). O circuito poderá ter duas (p, q), ou mais chaves, em série que a saída (s) terá o mesmo resultado (1) se, e somente se, todas as chaves forem fechadas (1); caso uma, ou mais chaves forem abertas (0), o resultado será um circuito aberto com a lâmpada apagada (0).

O segundo circuito a seguir (disjunção) determina que se duas chaves (p e q) forem abertas (0), o resultado (s) será o de um circuito aberto com uma lâmpada apagada (0), por exemplo; basta que uma delas seja fechada (1), para que o circuito se feche. O circuito poderá ter duas (p, q), ou mais chaves, em paralelo que a saída (s) terá o mesmo resultado (0), se, e somente se, todas as entradas forem abertas (0); caso uma, ou mais chaves forem fechadas (1), o resultado será um circuito fechado coma lâmpada acesa (1).

O terceiro circuito a seguir (negação) determina que se uma chave (p) for acionada (1), o resultado (s) será o de um circuito aberto com uma lâmpada apagada (0); caso contrário, o circuito permanecerá fechado, e a lâmpada se manterá acesa (1).

- Representações por circuitos



Circuito série (AND) (Conjunção) | Circuito paralelo (OR) (Disjunção) | Curto-circuito (NOT) (Negação)

- Representações de relações lógicas

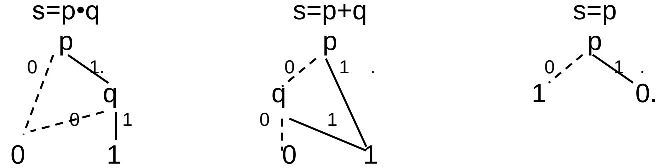
Conjunção (E)	Disjunção (OU)	Negação (NÃO)
(p e q)	(p ou q)	(não p)
$p \wedge q$	$p \vee q$	$\neg p$
p • q ou p q	p + q	/p ou $\bar{p}$ ou p'
p & q	p | q	!p ou ~p

Tabela-verdade

p q s	p q s	p s
0 • 0 = 0	0 + 0 = 0	
0 • 1 = 0	0 + 1 = 1	0' = 1
1 • 0 = 0	1 + 0 = 1	1' = 0
1 • 1 = 1	1 + 1 = 1	

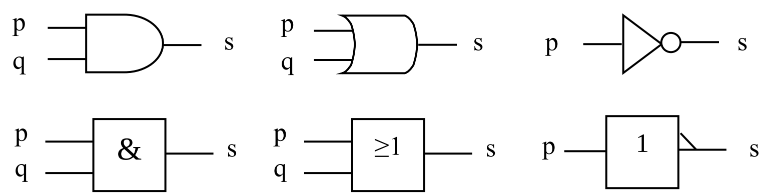
Diagrama de Decisão Binária (BDD)

s=p•q | s=p+q | s=p

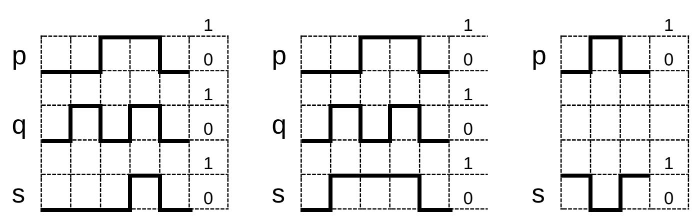


Portas Lógicas

s = AND (p, q) | s = OR (p, q) | s = NOT (p)



Notações para relações lógicas



Diagramas de tempo para as portas lógicas

- Prioridade de conectivos

Estabelece-se que a ordem de avaliação de uma expressão, envolvendo conectivos lógicos, será da esquerda para a direita, respeitando-se as prioridades dos conectivos na ordem mostrada abaixo, sendo a primeira a mais alta quando aplicada imediatamente a um valor.

NÃO
E
OU

Pode-se mudar a ordem de avaliação por meio de parênteses.

Exemplo:

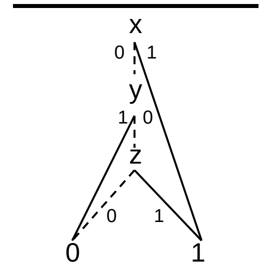
Considere a expressão: x + y' • z
A sua avaliação será feita na seguinte ordem de prioridade:

- negação de (y) : y'
- conjunção com (z) : y' • z
- disjunção com (x) : (y' • z) + x

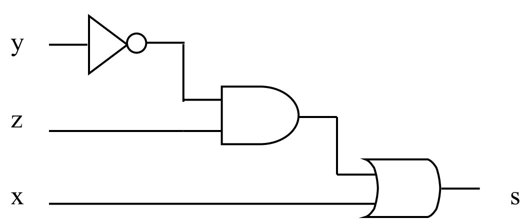
A expressão poderá ser representada na forma tabular (***tabela-verdade***) ou por BDD:

x y z	y'	(y' • z)	(y' • z) + x
0 0 0	1	0	0
0 0 1	1	1	1
0 1 0	0	0	0
0 1 1	0	0	0
1 0 0	1	0	1
1 0 1	1	1	1
1 1 0	0	0	1
1 1 1	0	0	1

BDD



A representação por um circuito lógico poderá ser a mostrada abaixo.



Se fosse desejado que a operação de disjunção ocorresse antes da conjunção, seria necessário o uso de parênteses, e o circuito seria diferente.

(x + y') • z

Se fosse desejado negar toda a expressão, também se usaria parênteses, e essa ação seria a última a ser avaliada. O circuito resultante seria o mesmo acima com mais um inversor.

(x + y' • z)'

Exercício - Construir tabela e circuito para as proposições:

a) $p+\overline{q}$
b) $\overline{p}\cdot\overline{q}$
c) $(p\cdot\overline{q})+r$
d) $\overline{(p\cdot q)}$
e) $\overline{(p+q)}$
f) $(p+\overline{r})(q+\overline{r})$

- Principais propriedades

Idempotência	Comutativa	Associativa
p + p = p	p + q = q + p	(p+q)+r = p+(q+r)
p • p = p	p • q = q • p	(p•q) •r = p• (q•r)

Distributiva	Absorção	Identidade	
p+(q•r)=(p+q) • (p+r)	$p+(\overline{p}\cdot q)=(p+q)$	p+0 = p	p•0 =0
p•(q+r)=(p•q)+(p•r)	$\overline{p}+(p\cdot q)=(\overline{p}+q)$	p+1 = 1	p•1 =p
	$p+(p\cdot q)=p$		

Complementar	De Morgan
$p+\overline{p}=1$ (tautologia)	$\overline{(p+q)}=\overline{p}\cdot\overline{q}$
$p\cdot\overline{p}=0$ (contradição)	$\overline{(p\cdot q)}=\overline{p}+\overline{q}$

Observação:
As regras de absorção são aplicadas, em geral, para se efetuar simplificações; normalmente, ao se levar em conta a precedência de operadores, a conjunção tem prioridade sobre a disjunção e as regras de absorção não se aplicam.

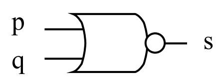
Exercício - Simplificar pelas propriedades da álgebra:

a) (p • q')'
b) (p' + q)'
c) ((p • q)' + (q • p)')'
d) (p • q) + (p'. (p + q'))
e) (p + q)' • (p • q)
f) (p • q) • (p'+q')' + r

- Outras relações lógicas importantes

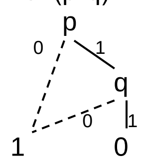
É comum usar as negações das portas principais e definir outras relações lógicas:

Porta NOR



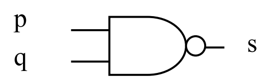
p q	s=(p + y)'
0 0	1
0 1	0
1 0	0
1 1	0

s=(p•q)'



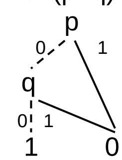
s= NAND(p,q)

Porta NAND



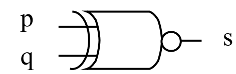
p q	s=(p • y)'
0 0	1
0 1	1
1 0	1
1 1	0

s=(p+q)'



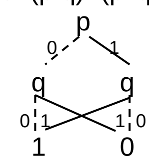
s=NOR(p,q)

Porta XNOR (NEXOR)



p q	s=(p xor y)'
0 0	1
0 1	0
1 0	0
1 1	1

s=(p.q)+(p'.q')



s=XNOR(p,q)

Além delas, também se poderão definir:

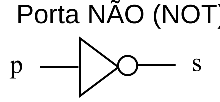
p q	$p \rightarrow q$	= (p' + q) (p implica q)	$p \Leftrightarrow q$	= $(p \rightarrow q) \bullet (q \rightarrow p)$ (p equivale a q)
0 0	1	(se p então q) ou (q, se p) ou	1	(se p então q, e se q então p) ou
0 1	1	(q, dado p) ou	0	(q é condição necessária
1 0	0	(p é condição suficiente para q) ou	0	e suficiente para p)
1 1	1	(q é condição necessária para p)	1	

As prioridades dessas últimas relações normalmente são inferiores às das anteriores.

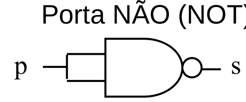
- Universalidade das portas NAND e NOR

A universalidade das portas NAND e NOR permite que todas as funções lógicas básicas possam ser substituídas por composições equivalentes, como se mostrará a seguir.

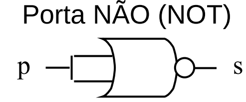
Porta NÃO (NOT)



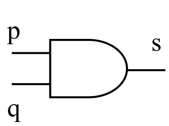
Porta NÃO (NOT)



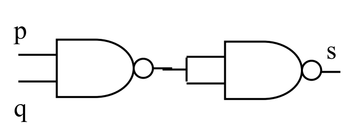
Porta NÃO (NOT)



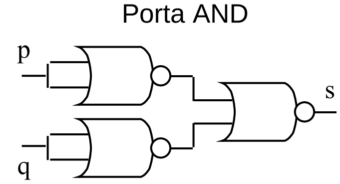
Porta AND



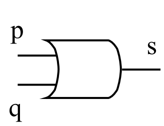
Porta AND



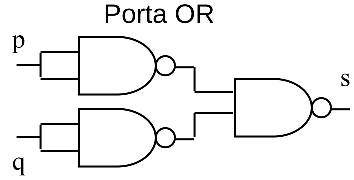
Porta AND



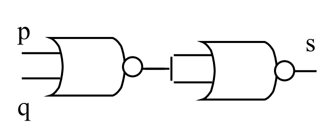
Porta OR



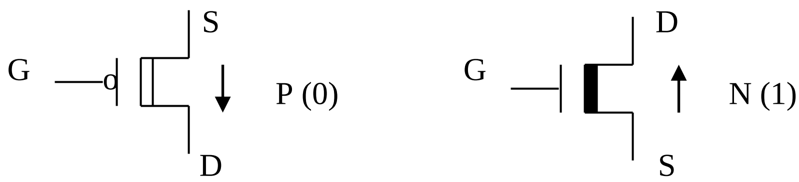
Porta OR



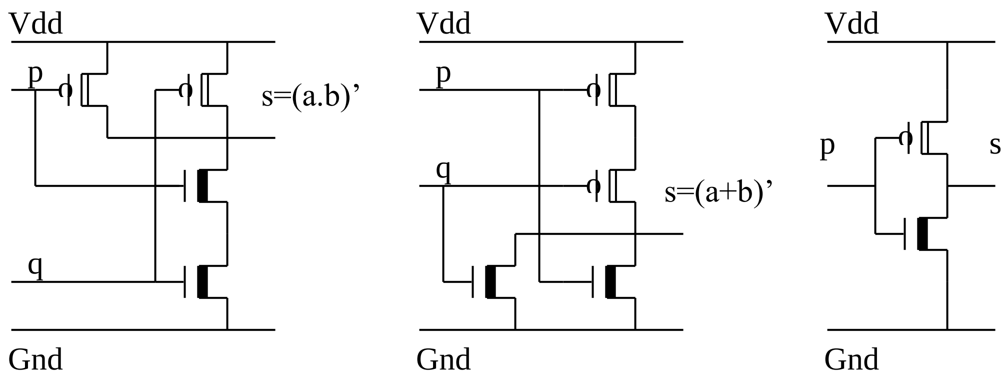
Porta OR



- Analogias com transistores (CMOS)



Tipo P　　　　Tipo N



NAND　　　　NOR　　　　NOT

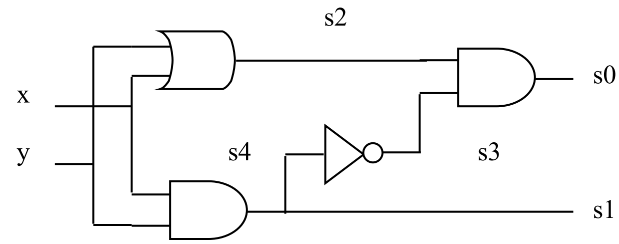
Projeto de circuitos lógicos

A síntese de circuitos lógicos pode ser executada em cinco níveis:

Nível	Atividades
Sistema	- especificação de requisitos - particionamento
Algoritmo	- especificação de comportamento - concorrência - complexidade - representação de dados
Arquitetura	- representação de dados - sinais e controle - paralelismo e *pipelining* - *data paths*
Lógico	- circuitos - otimizações em portas e transistores - mapeamento em bibliotecas
Físico	- otimização lógica - planejamento de *layout* - fabricação e encapsulamento

Aplicações aritméticas de expressões e circuitos lógicos

Dado o circuito lógico:



As relações abaixo descrevem os sinais de saída em função dos sinais de entrada:

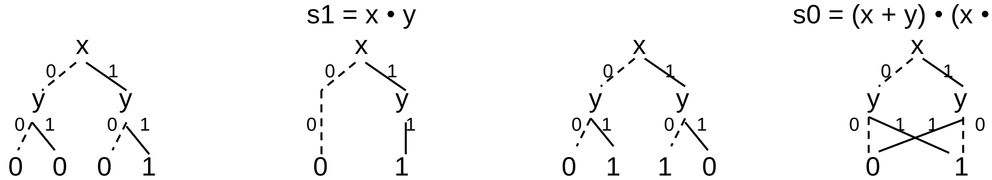
s0 = s2 • s3 = s2 • s4' = (x + y) • (x • y)'   (a)
s1 = s4 = x • y   (b)
s2 = x + y
s3 = s4'
s4 = x • y

A partir das relações (a) e (b) pode-se construir a tabela-verdade:

x y	x+y	s1=x•y	(x•y)'	s0=(x+y)•(x•y)'
0 0	0	0	1	0
0 1	1	0	1	1
1 0	1	0	1	1
1 1	1	1	0	0

e os respectivos diagramas de decisão binária (*BDDs – Binary Decision Diagrams*):

s1 | s1 compacto
s1 = x • y | s0 | s0 compacto
s0 = (x + y) • (x • y)'



O resultado também poderá ser apresentado de outra forma:

x + y =	s1 s0
0 + 0 =	0 0
0 + 1 =	0 1
1 + 0 =	0 1
1 + 1 =	1 0

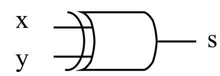
ou seja, o circuito é o responsável por uma soma binária de dois dígitos binários (bits), e é chamado de circuito de ***meia-soma***. Para se operar três ou mais bits, vários desses serão combinados em cascata para formar um circuito de ***soma-completa*** (descrito mais adiante).

A equação (a) que define uma das saídas do circuito (s0) poderá ainda ser simplificada pelas propriedades da álgebra:

$(x + y) \bullet (x \bullet y)'$	Propriedades
$(x + y) \bullet (x'+y')$	- De Morgan
$x\bullet x' + y\bullet x' + x\bullet y' + y\bullet y'$	- Distributiva
$0 + y\bullet x' + x\bullet y' + 0$	- Complementar
$(0 + y\bullet x') + (x\bullet y' + 0)$	- Associativa
$y\bullet x' + x\bullet y'$	- Identidade
$x'\bullet y + x\bullet y'$	- Comutativa

A relação obtida também serve para descrever uma outra porta lógica - ***OU exclusivo*** (XOR) - cuja representação encontra-se abaixo:

Porta OU-exclusivo (XOR)



x	y	$x \oplus y$	mintermos	MAXTERMOS	N
0	0	0	0	(X'+Y')	0
0	1	1	(x'•y)	1	1
1	0	1	(x•y')	1	2
1	1	0	0	(X + Y)	3

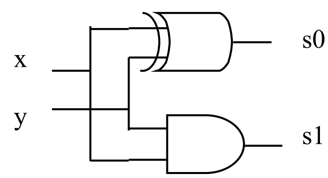
Essa relação poderá ser definida por uma soma dos produtos (SoP):

$$0 + (x'\bullet y) + (x\bullet y') + 0 = (x'\bullet y) + (x\bullet y') = \sum(1,\ 2)$$

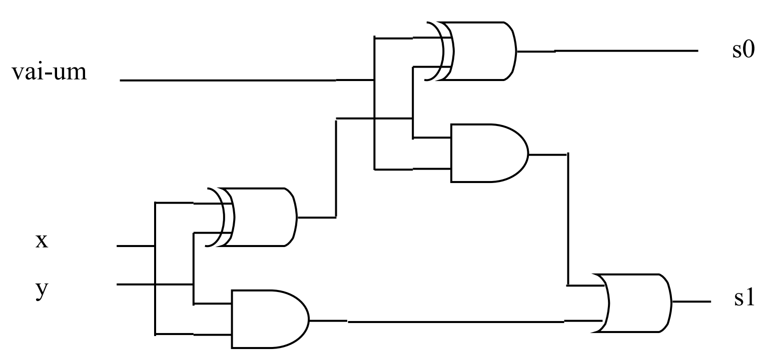
ou definida por um produto das somas (PoS):

$$(X'+Y')\bullet 1\bullet 1\bullet (X+Y)=(X'\bullet X)+(X'\bullet Y)+(Y'\bullet X)+(Y'\bullet Y)=(X'\bullet Y)+(Y'\bullet X) = \prod(0,\ 3)$$

O circuito de ***meia-soma*** poderá ser refeito com a porta **XOR** :



O circuito de ***soma-completa*** também poderá ser feito com essa porta:



As relações expressas pelo circuito de ***soma-completa*** poderão ser encontradas abaixo:

vai-um + x + y			s1	s0	mintermos	MAXTERMOS	N
0	0	0	0	0	$(v'\bullet x'\bullet y')$	$(V + X + Y)$	0
0	0	1	0	1	$(v'\bullet x'\bullet y)$	$(V + X + Y')$	1
0	1	0	0	1	$(v'\bullet x \bullet y')$	$(V + X' + Y)$	2
0	1	1	1	0	$(v'\bullet x \bullet y)$	$(V + X' + Y')$	3
1	0	0	0	1	$(v \bullet x'\bullet y')$	$(V' + X + Y)$	4
1	0	1	1	0	$(v \bullet x'\bullet y)$	$(V' + X + Y')$	5
1	1	0	1	0	$(v \bullet x \bullet y')$	$(V' + X' + Y)$	6
1	1	1	1	1	$(v \bullet x \bullet y)$	$(V' + X' + Y')$	7

A relação (**s0**) poderá ser definida por uma soma dos produtos (SoP):

$$(v'\bullet x'\bullet y) + (v'\bullet x \bullet y') + (v \bullet x'\bullet y') + (v \bullet x \bullet y) = \sum(1,\ 2,\ 4,\ 7)$$

ou definida por um produto das somas (PoS):

$$(V + X + Y)\bullet(V + X' + Y')\bullet(V' + X + Y')\bullet(V' + X' + Y) = \prod(0,\ 3,\ 5,\ 6)$$

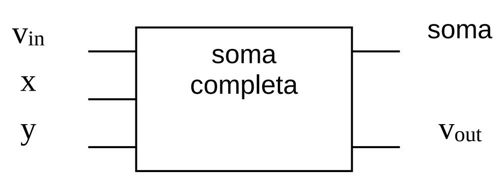
A relação (**s1**) também poderá ser definida por uma soma dos produtos (SoP):

$$(v'\bullet x \bullet y) + (v \bullet x'\bullet y) + (v \bullet x \bullet y') + (v \bullet x \bullet y) = \sum(3,\ 5,\ 6,\ 7)$$

ou definida por um produto das somas (PoS):

$$(V + X + Y)\bullet(V + X + Y')\bullet(V + X' + Y)\bullet(V' + X + Y) = \prod(0,\ 1,\ 2,\ 4)$$

O diagrama de blocos abaixo resume esse circuito:



A diferença entre dois ***bits*** também pode ser projetada de modo semelhante.

x - y	d	v	mintermos	MAXTERMOS	N
0 0	0	0	(x' • y')	(X + Y)	0
0 1	1	1	(x' • y)	(X + Y')	1
1 0	1	0	(x • y')	(X' + Y)	2
1 1	0	0	(x • y)	(X' + Y')	3

A diferença (d) poderá ser definida por uma soma dos produtos (SoP):

$$(x'\bullet y) + (x\bullet y') = \sum(1,\ 2)$$

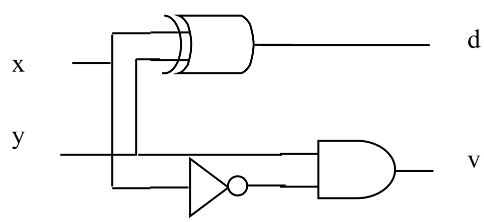
ou definida por um produto das somas (PoS):

$$(X+Y) \bullet (X'+Y') = \prod(0,\ 3)$$

A necessidade de empréstimo ("vem-um") poderá ser expressa pela relação:

$$(x'\bullet y) = v$$

O circuito de ***meia-diferença*** é mostrado a seguir:



O circuito de ***diferença-completa*** (x-y-empréstimo) terá a definição abaixo:

x – y – vem-um	s1	s0	mintermos	MAXTERMOS	N
0 0 0	0	0	(x'•y'•v')	(X + Y + V)	0
0 0 1	1	1	(x'•y'•v)	(X + Y + V')	1
0 1 0	1	1	(x'•y •v')	(X + Y' + V)	2
0 1 1	1	0	(x'•y •v)	(X + Y' + V')	3
1 0 0	0	1	(x •y'•v')	(X' + Y + V)	4
1 0 1	0	0	(x •y'•v)	(X' + Y + V')	5
1 1 0	0	0	(x •y •v')	(X' + Y' + V)	6
1 1 1	1	1	(x •y •v)	(X' + Y' + V')	7

A relação (**s0**) também poderá ser definida pela soma dos produtos (SoP):

$$(x'\bullet y'\bullet v) + (x'\bullet y \bullet v') + (x \bullet y'\bullet v') + (x \bullet y \bullet v) = \sum m(1,\ 2,\ 4,\ 7)$$

ou ainda definida pelo produto das somas (PoS):

$$(X + Y + V)\bullet(X + Y' + V')\bullet(X' + Y + V')\bullet(X' + Y' + V) = \prod M(0,\ 3,\ 5,\ 6)$$

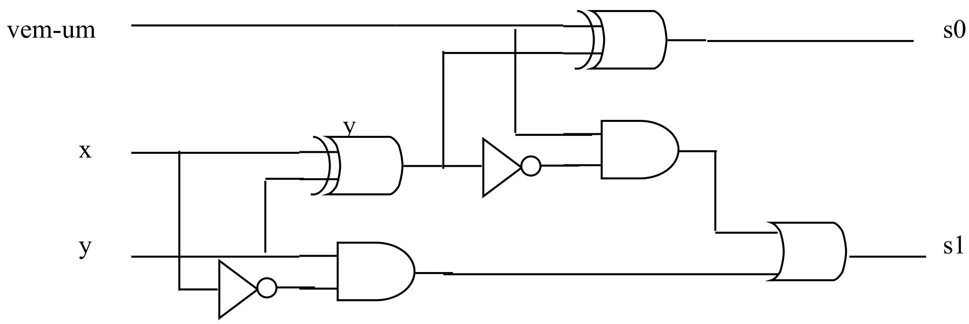
A relação (**s1**) poderá ser definida pela soma dos produtos (SoP):

$$(x'\bullet y'\bullet v) + (x'\bullet y \bullet v') + (x'\bullet y \bullet v) + (x \bullet y \bullet v) = \sum m(1,\ 2,\ 3,\ 7)$$

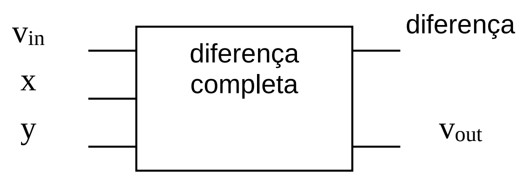
ou definida por um produto das somas (PoS):

$$(X + Y + V)\bullet (X' + Y + V)\bullet (X' + Y + V')\bullet (X' + Y' + V) = \prod M(0,\ 4,\ 5,\ 6)$$

O circuito equivalente é mostrado abaixo.



O diagrama de blocos descrito a seguir resume esse circuito:



Equivalências em lógica exclusiva

Axiomas	Complementares	Associativa
$p\oplus 0=p$	$(p\oplus q)=(\bar{p}\ q)+(p\ \bar{q})$	$(p\oplus q)\ \oplus r=p\oplus (q\oplus r)$
$p\oplus p=0$	$(p\oplus q)=(p+q)\ (\bar{p}+\bar{q})$	$p\oplus q\oplus p=q$
$p\oplus 1=\bar{p}$	$(p\oplus q)=(p+q)\ \overline{(p\cdot q)}$	Comutativa
$p\oplus \bar{p}=1$	$(p\oplus q)=\overline{(p\cdot q)}\ (p+q)$	$p\oplus q=q\oplus p$

Complementares

$p\oplus q=\bar{p}\oplus \bar{q}$	$p\oplus (\bar{p}\ q)=p+q$	$p\oplus (p\ \bar{q})=p\ q$
$\overline{(p\oplus q)}=\bar{p}\oplus q$	$\bar{p}\oplus (p+\bar{q})=p\ q$	$p\oplus (p+q)=\bar{p}\ q$
$\overline{(p\oplus q)}=p\oplus \bar{q}$	$p+(p\oplus q)=p+q$	$p\ (p\oplus \bar{q})=p\ q$

Exercício – Verificar pelas equivalências em lógica exclusiva:

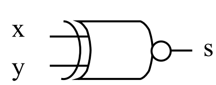
a) $(p\ \bar{q})+(\bar{p}\ q)=(p\ \bar{q})\oplus(\bar{p}\ q)$

b) $(p+q)\ (\bar{p}+\bar{q})=(p+q)\ (\bar{p}\oplus \bar{q})$

c) $(p+q)\ \overline{(p\cdot q)}=(p\oplus q)\ \overline{(p\cdot q)}$

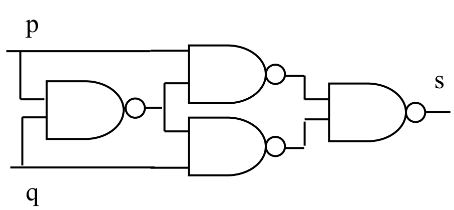
A relação complementar à relação XOR é chamada XNOR (ou também NEXOR) e descreve uma outra porta lógica cuja representação encontra-se abaixo:

Porta XNOR (NEXOR)

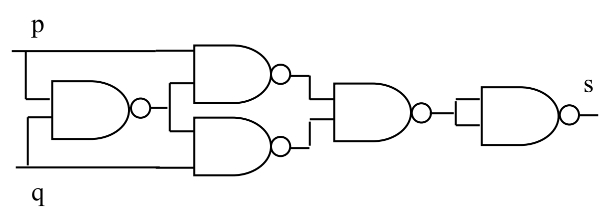


x	y	$x\Theta y$
0	0	1
0	1	0
1	0	0
1	1	1

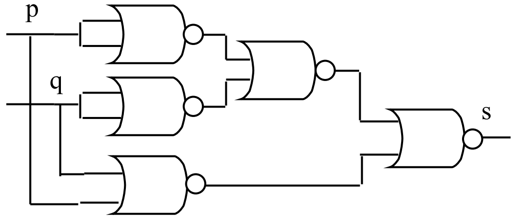
Porta XOR com NAND



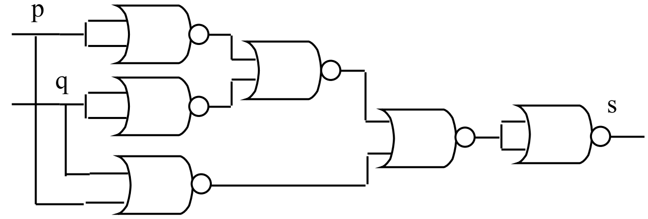
Porta XNOR com NAND



Porta XOR com NOR

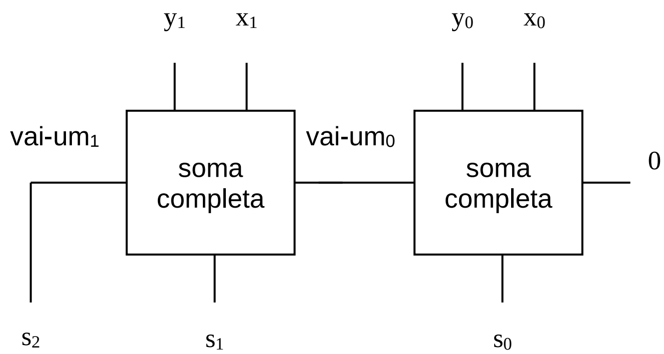


Porta XNOR com NOR



Operações aritméticas em paralelo

Um circuito para a adição de dois pares de ***bits*** poderia operar em paralelo:

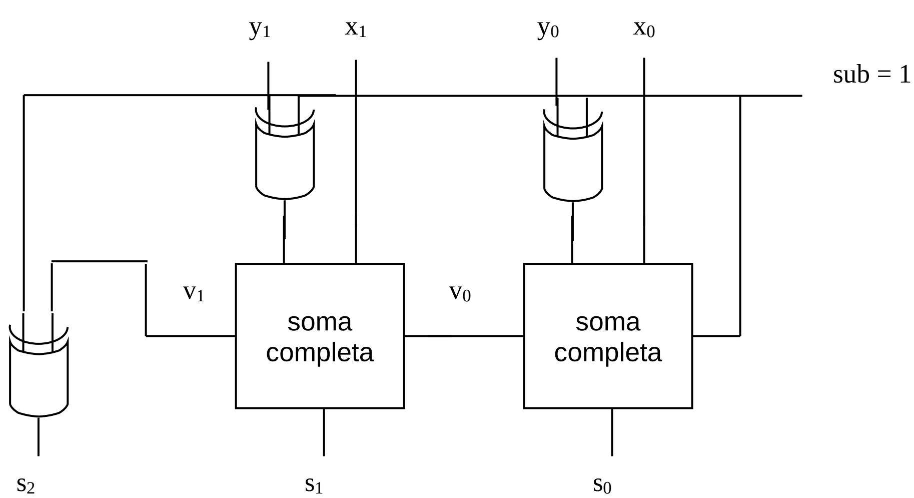


Um circuito para a subtração de dois pares de ***bits*** também pode usar o esquema acima. Entretanto, se for desejado otimizar os recursos envolvidos, a base do circuito de adição também poderá realizar a soma algébrica através da implementação da regra para a operação de soma utilizando o complemento de 2. Nesse caso, a saída de cada bloco implementará a relação:

s = x + inverso (y) + "vai-um"

para prover a soma adicional de uma unidade ao ***bit*** mais à direita (bit_0), a entrada em (0) do primeiro bloco à direita deverá alterada para (1). Esse valor poderá ser fornecido por uma linha adicional de controle (*sub*) que também servirá para tomar os inversos dos valores dos subtraendos (y_i), os valores dos minuendos (x_i) deverão ser tomados sem outras alterações.

O circuito abaixo ilustra essas modificações.



Códigos de condição (*flags*)

É conveniente observar que o circuito de soma algébrica (anterior) não considera as verificações de códigos de condições (*flags*) como:

- a possibilidade de ocorrência de erro por transbordamento de representação (*overflow*),
- ocorrência de resultado nulo (*zero flag*) e
- "vai-um" (*carry flag*).

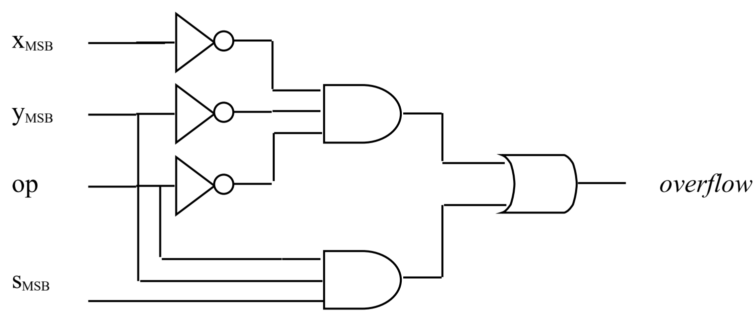
A situação de transbordamento (*overflow*), por exemplo, poderá ocorrer:

- na adição, quando os bits mais significativos (MSB) dos operandos (x, y) forem ambos iguais a 0 e o do resultado (s) for 1;
- na subtração, quando o bit mais significativo (MSB) do primeiro operando (x) for positivo, o do segundo operando (y) for negativo e o do resultado (s) também for negativo.

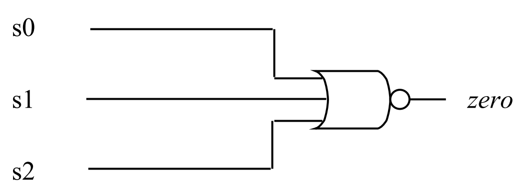
A tabela abaixo resume algumas das condições para ocorrência de transbordamento (*overflow*):

x_{MSB}	y_{MSB}	operação	s_{MSB}	*overflow*
0	0	0= soma	1	1
0	1	1=subtração	1	1

O circuito mostrado abaixo implementa essas condições para verificação de *overflow*.



A ocorrência de resultado nulo (*zero flag*) possui várias aplicações práticas como a verificação de testes ou o controle de repetições. Para implementar essa verificação basta testar se todos os bits do resultado são iguais a zero, como indicado no circuito abaixo.



Simplificação por agregação de produtos adjacentes (mapas de Veitch-Karnaugh)

Expressões do tipo X•Y+X•Y' podem ser simplificadas pelos mapas de Veitch-Karnaugh.

a\b	0	1
0	a'b'	a'b
1	a b'	a b

2 variáveis

a\bc	00	01	11	10
0	a'b'c'	a'b'c	a'b c	a'b c'
1	a b'c'	a b'c	a b c	a b c'

3 variáveis

ab\cd	00	01	11	10
00	a'b'c'd'	a'b'c'd	a'b'c d	a'b'c d'
01	a'b c'd'	a'b c'd	a' b c d	a'b c d'
11	a b c'd'	a b c'd	a b c d	a b c d'
10	a b'c'd'	a b'c'd	a b'c d	a b'c d'

4 variáveis

Exemplo 1: Simplificar a expressão abaixo.

$s_1 = a'b'+a'b$

a\b	0	1
0	1	1
1	0	0

$s_1 = a'(b'+b) = a'$

$s_2 = a'b'+a\ b'$

a\b	0	1
0	1	0
1	1	0

$s_2 = b'(a'+a) = b'$

Exemplo 2: Simplificar as expressões abaixo.

$s_{21} = a'b'c'+a'b'c+a'b\ c'+a\ b\ c'$

a\bc	00	01	11	10
0	1	1	0	1
1	0	0	0	1

$s_{21} = (a'b')(c'+c)+(b\ c')(a'+a) = (a'b')+(b\ c')$

$s_{22} = a'b'c'+ a'b'c+a\ b'c'+a\ b'c+a'b\ c'+a\ b\ c'$

a\bc	00	01	11	10
0	1	1	0	1
1	1	1	0	1

$s_{22} = (a'b')(c'+c)+(a\ b')(c'+c)+(b\ c')(a'+a)$
$= (a'b')+(a\ b')+(b\ c') = b'(a'+a)+(b\ c') = b'+(b\ c')$

Exemplo 3: Simplificar as expressões abaixo.

$s_{31} = a'b'c'd'+a'b'c\ d'+a\ b'c'd'+a\ b'c\ d'$

ab\cd	00	01	11	10
00	1	0	0	1
01	0	0	0	0
11	0	0	0	0
10	1	0	0	1

$s_{31} = (c'd')(a'b'+a\ b')+(c\ d')(a'b'+a\ b')$
$= ((c'd')+(c\ d'))(b'(a'+a))$
$= (d'(c'+c))(b')$
$= (d')(b') = b'd'$

$s_{32} = a'b'c\ d'+a'b\ c'd'+a'b\ c\ d'+a\ b\ c\ d'+a\ b'c\ d'$

ab\cd	00	01	11	10
00	0	0	0	1
01	1	0	0	1
11	0	0	0	1
10	0	0	0	1

$s_{32} = (c\ d')(a'b'+a'b+a\ b+a\ b')+(a'b\ c'd')$
$= (c\ d')(a'(b'+b)+a(b+b'))+(a'b\ c'd')$
$= (c\ d')(a'+a)+(a'b\ c'd') = (c\ d')+(a'b\ c'd')$
$= d'(c + a'b\ c') = d'(c+a'b) = (c\ d')+(a'b\ d')$

$s_{33} = a'b\ c'd+a'b\ c\ d+a\ b\ c'd+a\ b\ c\ d$

ab\cd	00	01	11	10
00	0	0	0	0
01	0	1	1	0
11	0	1	1	0
10	0	0	0	0

$s_{33} = (b\ c'd)(a'+a)+(b\ c\ d)(a'+a)$
$= (b\ c'\ d)+(b\ c\ d)$
$= (b\ d)(c'+c)$
$= b\ d$

Resumo dos mapas de Veitch-Karnaugh dependendo do número de variáveis:

a\b	0	1
0	(00)	(01)
1	(02)	(03)

2 variáveis

a\bc	00	01	11	10
00	(00)	(01)	(03)	(02)
01	(04)	(05)	(07)	(06)

3 variáveis

ab\cd	00	01	11	10
00	(00)	(01)	(03)	(02)
01	(04)	(05)	(07)	(06)
11	(12)	(13)	(15)	(14)
10	(08)	(09)	(11)	(10)

4 variáveis

ab\cde	000	001	011	010	110	111	101	100
00	(00)	(01)	(03)	(02)	(06)	(07)	(05)	(04)
01	(08)	(09)	(11)	(10)	(14)	(15)	(13)	(12)
11	(24)	(25)	(27)	(26)	(30)	(31)	(29)	(28)
10	(16)	(17)	(19)	(18)	(22)	(23)	(21)	(20)

5 variáveis

abc\def	000	001	011	010	110	111	101	100
000	(00)	(01)	(03)	(02)	(06)	(07)	(05)	(04)
001	(08)	(09)	(11)	(10)	(14)	(15)	(13)	(12)
011	(24)	(25)	(27)	(26)	(30)	(31)	(29)	(28)
010	(16)	(17)	(19)	(18)	(22)	(23)	(21)	(20)
110	(48)	(49)	(51)	(50)	(54)	(55)	(53)	(52)
111	(56)	(57)	(59)	(58)	(62)	(63)	(61)	(60)
101	(40)	(41)	(43)	(42)	(46)	(47)	(45)	(44)
100	(32)	(33)	(35)	(34)	(38)	(39)	(37)	(36)

6 variáveis

Simplificação pela lógica de Reed-Müller

Expressões do tipo X’•Y+X•Y’ podem ser simplificadas pela lógica de Reed-Müller.

Exemplo 1: Simplificar a configuração abaixo.

\ab	00	01	11	10
	0	1	0	1

s_{01} = (a’b)+(a b’) = a xor b

Exemplo 2: Simplificar a configuração abaixo.

c\ab	00	01	11	10
0	0	1	0	1
1	1	0	1	0

s_{12} = (a’b) c’ +(a b’) c‘ + (a’b’) c + (a b) c
= a xor b xor c

Exemplo 3: Simplificar a configuração abaixo.

cd\ab	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	0	1	0	1
10	1	0	1	0

s_{22} = (a’b c’d’)+(a b’c‘d’) + (a’b’c’d)+(a b c’d)
+(a’b c d)+(a b’c d) + (a’b’c d’)+(a b c d’)
= a xor b xor c xor d

Exemplo 4: Simplificar a configuração abaixo.

cd\ab	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	0	1	0	1
10	0	1	0	1

s_{22} = (a’b c’d’)+(a b’c‘d’) + (a’b c’d)+(a b’c’d)
+(a’b c d)+(a b’c d) + (a’b’c d’)+(a b’c d’)
= a xor b

Exemplo 5: Simplificar a configuração abaixo.

cd\ab	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	1	1	0	0
10	1	1	0	0

s_{22} = (a b c’d’)+(a b’c‘d’) + (a b c’d)+(a b’c’d)
+(a’b’c d)+(a’b c d) + (a’b’c d’)+(a’b c d’)
= a xor c

Exemplo 6: Simplificar a configuração abaixo.

cd\ab	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	1	0	1	0
10	1	0	1	0

s_{22} = (a’b c’d’)+(a b’c‘d’) + (a’b c’d)+(a b’c’d)
+(a’b’c d)+(a b c d) + (a’b’c d’)+(a b c d’)
= a xor b xor c

Simplificação pelo método de Quine-McCluskey

Mapas de Veitch-Karnaugh podem ser aplicados para até 5 ou 6 variáveis, mas para um número maior outros métodos deverão ser aplicados. Um método que pode ser implementado de maneira sistemática é o de Quine-McCluskey, o qual pode ser resumido pelos seguintes passos:

- produzir uma expansão de uma função na forma de soma de produtos (SoP);
- reduzir ao máximo os termos do tipo X•Y+X•Y' ;
- identificar um conjunto mínimo de fatores ***primos implicantes*** equivalente à função.

Exemplo:

Dada a função abaixo, identificar um conjunto de fatores ***primos implicantes***.

$$f\ (a,\ b,\ c,\ d\ ) = \sum m(0,\ 1,\ 2,\ 5,\ 6,\ 7,\ 8,\ 9,\ 10,\ 14)$$

1º passo: Identificar os grupos que possuem a mesma quantidade de 1's:

N	a	b	c	d	f (a,b,c,d)
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	1
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	0
13	1	1	0	1	0
14	1	1	1	0	1
15	1	1	1	1	0

Termos		N	Grupos	
0 0 0 0		0	0 0 0 0	0 bit em 1
0 0 0 1				
0 0 1 0		1	0 0 0 1	1 bit em 1
		2	0 0 1 0	
		8	1 0 0 0	
0 1 0 1				
0 1 1 0		5	0 1 0 1	2 bits em 1
0 1 1 1		6	0 1 1 0	
1 0 0 0		9	1 0 0 1	
1 0 0 1		10	1 0 1 0	
1 0 1 0				
		7	0 1 1 1	3 bits em 1
		14	0 1 1 1	
1 1 1 0				

2º passo: Agrupar os termos com os mesmos bits de diferença:

N	Grupos		Grupos I	1 bit		Grupos II	2 bits	OBS.:
0	0 0 0 0	#	0, 1	0 0 0 X	#	0, 1, 8, 9	X 0 0 X	
			0, 2	0 0 X 0	#	0, 2, 8, 10	X 0 X 0	
1	0 0 0 1	#	0, 8	X 0 0 0	#	0, 8, 1, 9		repetido
2	0 0 1 0	#				0, 8, 2, 10		repetido
8	1 0 0 0	#	1, 5	0 X 0 1				
			1, 9	X 0 0 1	#	2, 6, 10, 14	X X 1 0	
5	0 1 0 1	#	2, 6	0 X 1 0	#	2, 10, 6, 14		repetido
6	0 1 1 0	#	2, 10	X 0 1 0	#			
9	1 0 0 1	#	8, 9	1 0 0 X	#			
10	1 0 1 0	#	8, 10	1 0 X 0	#			
7	0 1 1 1	#	5, 7	0 1 X 1				
14	1 1 1 0	#	6, 7	0 1 1 X				
			6, 14	X 1 1 0	#			
			10, 14	1 X 1 0	#			

3º passo: Agrupar Identificar os termos não utilizados em algum agrupamento:

f (a, b, c, d) = (1,5) + (5,7) + (6,7) + (0,1,8,9) + (0,2,8,10) + (2,6,10,14)

= a’c’d + a’b d + a’ b c + b’c’ + b’d’ + c d’

4º passo: Montar a tabela de primos implicantes (não cobertos por outros termos):

primos	mintermos	0	1	2	5	6	7	8	9	10	14
0, 1, 8, 9	b’ c’	X	X					X	X		
0, 2, 8, 10	b’ d’	X		X				X		X	
2, 6, 10, 14	c d’			X		X				X	X
1, 5	a' c’ d		X		X						
5, 7	a' b d				X		X				
6, 7	a' b c					X	X				

5º passo: Identificar os mintermos cobertos por apenas um único conjunto de primos:

primos	mintermos	0	1	2	5	6	7	8	9	10	14
0, 1, 8, 9	b’ c’	X	X					X	X		
0, 2, 8, 10	b’ d’	X		X				X		X	
2, 6, 10, 14	c d’			X		X				X	X
1, 5	a' c’ d		X		X						
5, 7	a' b d				X		X				
6, 7	a' b c					X	X				

Há dois mintermos ***essenciais*** que satisfazem a condição: (b’c’) e (c d’)
e, por isso, devem estar presentes na resposta: f (a,b,c,d) = (b’c’) + (c d’) + . . .

6º passo: Eliminar todos os termos redundantes cobertos por esses:

primos	mintermos		0	1	2	5	6	7	8	9	10	14
0, 1, 8, 9	b’ c’	#	~~X~~	~~X~~					~~X~~	~~X~~		
0, 2, 8, 10	b’ d’		~~X~~		~~X~~				~~X~~		~~X~~	
2, 6, 10, 14	c d’	#			~~X~~		~~X~~				~~X~~	~~X~~
1, 5	a' c’ d			~~X~~		X						
5, 7	a' b d					X		X				
6, 7	a' b c						~~X~~	X				

7º passo: Escolher outros mintermos que possam eliminar redundâncias:

primos	mintermos		0	1	2	5	6	7	8	9	10	14
0, 1, 8, 9	b’ c’											
0, 2, 8, 10	b’ d’											
2, 6, 10, 14	c d’											
1, 5	a' c’ d					~~X~~						
5, 7	a' b d	#				~~X~~		~~X~~				
6, 7	a' b c							~~X~~				

A função equivalente será dada por: f (a,b,c,d) = (b’c’) + (c d’) + (a’ b d)

Um inconveniente deve ser indicado: devido ao crescimento exponencial do número de combinações, a solução encontrada pelo método de Quine-McCluskey poderá não ser a única.

Uma maneira de demonstrar a afirmação anterior é aplicando o método de Petrick:

1º passo: Eliminar as linhas com os mintermos essenciais e colunas correspondentes:

primos	mintermos		0	1	2	5	6	7	8	9	10	14
0, 1, 8, 9	b’ c’	#	X	X					X	X		
0, 2, 8, 10	b’ d’		X		X				X		X	
2, 6, 10, 14	c d’	#			X		X				X	X
1, 5	a' c’ d			X		X						
5, 7	a' b d					X		X				
6, 7	a' b c						X	X				

2º passo: Rotular as linhas restantes:

primos	mintermos		0	1	2	5	6	7	8	9	10	14
1, 5	a' c’ d	K				X						
5, 7	a' b d	L				X		X				
6, 7	a' b c	M						X				

3º passo: Construir a soma de produtos das linhas adicionando as colunas:

(K + L) . (L + M)

4º passo: Simplificar a soma de produtos (X + XY = X):

K.L + L.L + K.M + L.M = K.L + L + K.M + L.M = L + K.M

5º passo: Escolher as soluções com as menores quantidades de termos:

L + K.M → L

6º passo: Expandir a solução e contar o número de variáveis:

L = a’ b d → 3 termos

7º passo: Escolher as soluções com as menores quantidades de variáveis:

L = (a’ b d)

Após a escolha, montar a função equivalente final com todos os termos selecionados:

f (a,b,c,d) = (a’ b d) + (b’c’) + (c d’)

Prova de teoremas

As relações descritas podem ser utilizadas em Lógica Formal para provar teoremas.

Exemplo 1:
Provar (s') dados:

1. t	- premissa
2. $t \rightarrow q'$	- premissa
3. $q' \rightarrow s'$	- premissa
4. q'	- *modus ponens* entre 1 e 2
5. s'	- *modus ponens* entre 3 e 4 (c.q.d.)

Exemplo 2:
Provar (a) dados :

1. $a' \rightarrow b$	- premissa
2. $b \rightarrow c$	- premissa
3. c'	- premissa
4. b'	- *modus tolens* entre 2 e 3
5. (a')'	- *modus tolens* entre 1 e 4
6. a	- dupla negação (c.q.d.)

Quantificadores

A Lógica de Predicados, ou Lógica de Primeira Ordem, associa o uso de quantificadores.

Quantificador Universal

$(\forall x \in S) \ \ P(x)$ - ***para todo*** x em S, P(x) é verdadeiro

Quantificador Existencial

$(\exists x \in S) \ \ P(x)$ - ***para algum*** x em S, P(x) é verdadeiro

Os quantificadores admitem complementação :

$$\text{não}\left((\forall x) \ \ P(x)\right) = (\exists x)\,(\text{não } P(x))$$

$$\text{não}\left((\exists x) \ \ P(x)\right) = (\forall x)\,(\text{não } P(x))$$

Exemplo :
Supor o predicado abaixo, devidamente quantificado :

$$(\forall x \in R) \ \ (x^2 + 2x - 1 > 0)$$

$$\text{não}\left((\forall x \in R) \ \ (x^2 + 2x - 1 > 0)\right) \rightarrow (\exists x \in R) \ \ (x^2 + 2x - 1 \leq 0)$$

o que pode ser verificado quando x = 0 !

Exercícios propostos

1. Provar, pela álgebra, que :

a) $(p \bullet q') + p' = p' + q'$

b) $p \bullet (p'+ q) = p \bullet q$

c) $p \bullet q + (p' + q')' = (p' + q')'$

d) $(p \bullet q + r) \bullet ((p \bullet q)' \bullet r') = 0$

e) $(p \bullet q + r') \bullet (p \bullet q \bullet r' + p \bullet r') = p \bullet r'$

2. Fazer as tabelas e os circuitos do exercício anterior.

3. Demonstrar as relações abaixo através de tabelas:

a) $x + x \bullet y = x$ (absorção)

b) $x \bullet (x + y) = x$ (absorção)

c) $(x + y) \bullet (x + z) = x + y \bullet z$

d) $x + x' \bullet y = x + y$

e) $x \bullet y + y \bullet z + y' \bullet z = x \bullet y + z$

4. Verificar o resultado das seguintes relações :

a) $(x \rightarrow y) \bullet x \rightarrow y$ ("*modus ponens*")

b) $(x \rightarrow y) \bullet y' \rightarrow x'$ ("*modus tolens*")

c) $(x + y) \bullet x' \rightarrow y$ (silogismo disjuntivo)

d) $(x \rightarrow y) \bullet (y \rightarrow z) \rightarrow (x \rightarrow z)$ (silogismo hipotético)

e) $((x \rightarrow y) \bullet (w \rightarrow z)) \bullet (x+w) \rightarrow (y+z)$ (dilema)

f) $(x \rightarrow y) \rightarrow (x \rightarrow x \bullet y)$ (absorção)

g) $x \bullet y \rightarrow x$ (simplificação)

h) $x \rightarrow x + y$ (adição)

i) $(x' \rightarrow x) \rightarrow x$ (contradição)

j) $(x'\rightarrow y) \rightarrow ((x' \rightarrow y') \rightarrow x)$ (contradição)

h) $y \bullet (y \bullet x' \rightarrow z) \bullet (y \bullet x'\rightarrow z') \rightarrow x$ (contradição)

i) $(x \rightarrow y) = (x \bullet y') \rightarrow (y \bullet y')$ (redução ao absurdo)

5. Verificar o resultado das seguintes relações

a) $x \bullet (y \bullet z) \Leftrightarrow (x \bullet y) \bullet z$	(associatividade)
b) $x + (y + z) \Leftrightarrow (x + y) + z$	(associatividade)
c) $x \bullet y \Leftrightarrow y \bullet x$	(comutatividade)
d) $x + y \Leftrightarrow y + x$	(comutatividade)
e) $x \bullet (y+z) \Leftrightarrow (x \bullet y) + (x \bullet z)$	(distributividade)
f) $x+(y \bullet z) \Leftrightarrow (x+y) \bullet (x+z)$	(distributividade)
g) $(x \bullet y)' \Leftrightarrow x' + y'$	(De Morgan)
h) $(x+y)' \Leftrightarrow x' \bullet y'$	(De Morgan)
i) $(x')' \Leftrightarrow x$	(involução)
j) $x \bullet x \Leftrightarrow x$	(idempotência)
k) $x + x \Leftrightarrow x$	(idempotência)
l) $x \rightarrow y \Leftrightarrow x' + y$	(implicação)
m) $(x \Leftrightarrow y) \Leftrightarrow (x \rightarrow y) \bullet (y \rightarrow x)$	(equivalência)
n) $x \rightarrow y \Leftrightarrow x' \rightarrow y'$	(contraposição)
o) $(x \bullet y) \rightarrow z \Leftrightarrow x \rightarrow (y \rightarrow z)$	(exportação)

6. Transformar as afirmativas abaixo em proposições:

a) A soma de dois inteiros é maior que o primeiro.
b) A soma de dois inteiros é maior que a diferença entre eles.
c) O produto de um inteiro por zero é menor que outro inteiro.
d) Se um de dois números é nulo, o produto deles é zero.
e) Se dois números são iguais a um terceiro, então são iguais.

7. Construir um circuito para identificar se dois bits são iguais.

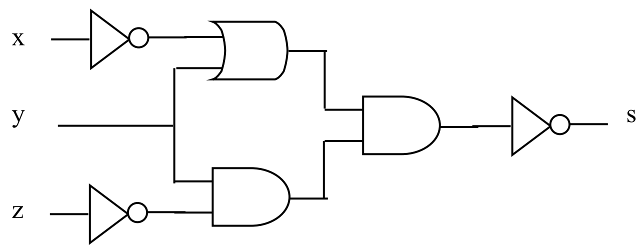
8. Construir um circuito capaz de fornecer a tabela abaixo:

x y z	f (x,y,z)
0 0 0	1
0 0 1	0
0 1 0	0
0 1 1	0
1 0 0	0
1 0 1	0
1 1 0	0
1 1 1	1

9. Simplificar a expressão lógica abaixo:

(x > 0 e y = 0) ou não (x < 0 ou z > 0)

10. Identificar a equação do circuito abaixo pela soma de produtos (SoP):



11. Identificar a equação do circuito anterior pelo produto das somas (PoS).

12. Construir um circuito equivalente ao da questão (10) usando apenas portas NAND.

13. Construir um circuito equivalente ao da questão (10) usando apenas portas NOR.

14. Montar um diagrama de um somador completo de 2 palavras de 2 bits cada.

15. Construir um circuito equivalente ao da questão (14) usando apenas portas NAND.

16. Construir um circuito equivalente ao da questão (14) usando apenas portas NOR.

17. Simplificar por mapa de Veitch-Karnaugh:
a'b'c'd'+a'b'c'd+a b'c'd+a b c'd'+a b c d'+a b c'd+a b c d

18. Demonstrar a validade do seguinte argumento:

(1) x < 6
(2) y > 7 ou x = y $\rightarrow$ (y = 4 e x < y)
(3) y = 4 $\rightarrow$ x < 6
(4) x < 6 $\rightarrow$ x < y
$\therefore$ x = y

19. Verificar se as proposições são falsas ou verdadeiras:

a) $(\forall n \in N)$ (n+4 > 3)

b) $(\forall n \in N)$ (n+3 > 7)

c) $(\exists n \in N)$ (n+4 < 7)

d) $(\exists n \in N)$ (n+3 < 2)

20. Sendo A = {1,2,3,4,5} determinar a negação de:

a) $(\exists n \in A)$ (x+4 = 10)

b) $(\forall n \in N)$ (x+4 < 10)