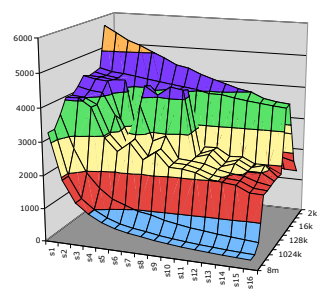
Architecture des systèmes informatiques

3 — Algèbre de Boole



Georges-Andre.Silber@ensmp.fr
CRI/ENSMP

MINES PARIS
CRI

Algèbre de Boole

- George Boole au 19ème siècle
- Représentation algébrique de la logique

NON
NOT
~A

~	
0	1
1	0

ET
AND
A&B

&	0	1
0	0	0
1	0	1

OU
OR
A|B

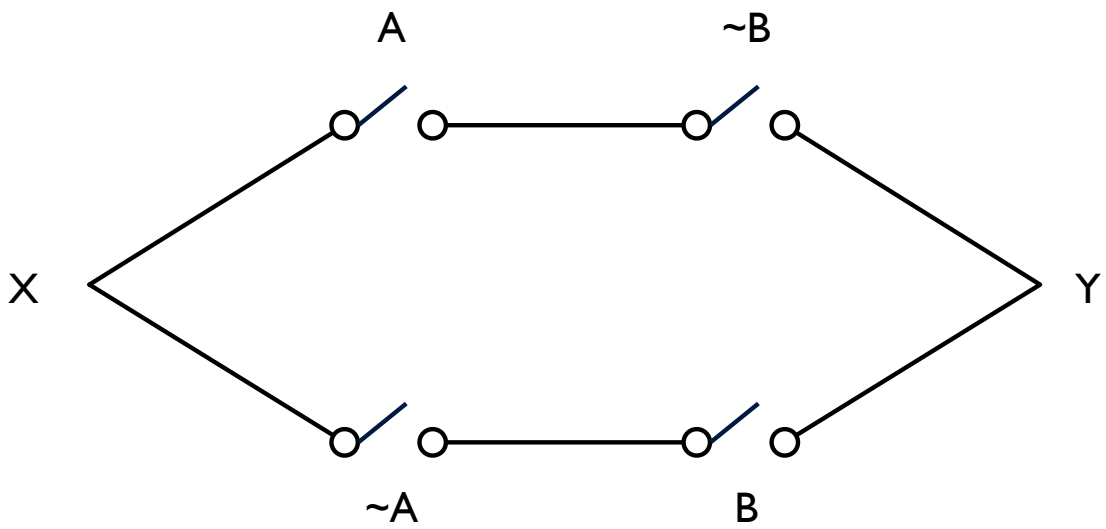
|	0	1
0	0	1
1	1	1

OU EXCLUSIF
XOR
A^B

^	0	1
0	0	1
1	1	0

Applications

- Appliqué aux systèmes digitaux par Claude Shannon
  - Thèse de Master au MIT en 1937
  - Raisonner à propos de réseaux d'interrupteurs
  - Fermé = 1, ouvert = 0



X et Y connectés quand

A&~B | ~A&B

=

A^B

Anneau des entiers

- <Z, +, *, -, 0, 1> est un anneau
  - + : somme
  - * : produit
  - - : opposé pour l'addition
  - 0 : élément neutre pour la somme
  - 1 : élément neutre pour le produit

Algèbre de Boole

- <{0,1}, |, &, ~, 0, 1> est une algèbre
  - | : somme
  - & : produit
  - ~ : complément (pas opposé)
  - 0 : élément neutre somme
  - 1 : élément neutre produit

Algèbre de Boole ≈ Anneau entier

ASSOCIATIVITÉ

booléens	entiers
(A|B)|C=A|(B|C)	(A+B)+C=A+(B+C)
(A&B)&C=A&(B|C)	(A*B)*C=A*(B*C)

ÉLÉMENTS NEUTRES

booléens	entiers
A|0=A	A+0=A
A&1=A	A*1=A

DISTRIBUTIVITÉ PRODUIT SUR SOMME

booléens	entiers
A&(B|C)=(A&B)|(A&C)	A*(B+C)=A*B+A*C

ÉLÉMENT ABSORBANT

booléens	entiers
A&0=0	A*0=0

COMMUTATIVITÉ

booléens	entiers
A|B = B|A	A+B=B+A
A&B=B&A	A*B=B*A

OPPOSÉ

booléens	entiers
~(~A)=A	-(-A)=A

MINES PARIS
CRI

Algèbre de Boole ≠ Anneau entier

ABSORPTION

booléens	entiers
A|(A&B)=A	A+(A*B)≠A
A&(A|B)=A	A*(A+B) ≠A

IDEMPOTENCE

booléens	entiers
A|A=A	A+A≠A
A&A=A	A*A≠A

DISTRIBUTIVITÉ SOMME SUR PRODUIT

booléens	entiers
A|(B&C)=(A|B)&(A|C)	A+(B*C)≠A+B*AC

LOI DES COMPLÉMENTS

booléens	entiers
A|~A=1	A+-A≠1

ANNEAU (ÉLÉMENT OPPOSÉ)

booléens	entiers
A|~A≠0	A+-A=0

Anneau de Boole

- <{0,1}, ^, &, Id, 0, 1> est un anneau
  - ^ : somme
  - & : produit
  - A ^ Id(A) = A ^ A = 0 (Identité)
  - 0 : élément neutre somme
  - 1 : élément neutre produit
  - Entiers modulo 2

Relations entre opérations

- Lois de Morgan
  - A&B = ~(~A|~B)
  - A|B=~(~A&~B)
- XOR avec OR
  - A^B=(~A&B)|(A&~B)
  - A^B=(A|B)&~(A&B)

Généralisation

- Vecteurs de bits
  - 01101001 & 01010101 = 01000001
  - 01101001 | 01010101 = 01111101
  - 01101001 ^ 01010101 = 00111100
  - ~01010101 = 10101010
- Les propriétés de l'algèbre de Boole s'appliquent

Représentation d'ensembles

- Vecteurs de w bits : sous-ensembles de {0,...,w-1}
  - a_j=1 si $j \in A$

```
01101001
76543210
```
{0, 3, 5, 6}

```
01010101
76543210
```
{0, 2, 4, 6}

- & intersection
- | union
- ^ différence symétrique
- ~ complément

```
01000001   {0, 6}
01111101   {0,2,3,4,5,6}
00111100   {2,3,4,5}
10101010   {1,3,5,7}
```

Opérations sur les bits en C

- Opérations `&`, `|`, `~`, `^`
  - s'applique aux types "entiers"
  - `long`, `int`, `short`, `char`
  - arguments : vecteurs de bits
  - application bit à bit

```
void showbits (char vec)
{
  int i;
  char mask = 0x80;
  for (i = 0; i < 8; i++)
    {
      printf ("%d", (vec&mask)?1:0);
      vec <<= 1;
    }
  printf ("\n");
}
```

```
int main (int ac, char *av)
{
  showbits (0x41);
  showbits (~0x41);
  showbits (0x69);
  showbits (0x55);
  showbits (0x69&0x55);
  showbits (0x69|0x55);
  showbits (0x69^0x55);
}
```

```
01000001
10111110
01101001
01010101
01000001
01111101
00111100
```

Voir showbits.c

Opérations logiques en C

- Opérations `&&`, `||`, `!`
  - 0 : faux
  - différent de 0 : vrai
  - retourne toujours 0 ou 1
  - A && B : B évalué ssi A est vrai
  - A || B : B évalué ssi A est faux

```
int main (int ac, char *av)
{
  showbits (!0x41);
  showbits (!0x00);
  showbits (!!0x41);
  showbits (0x69&&0x55);
  showbits (0x69||0x55);
}
```

```
00000000
00000001
00000001
00000001
00000001
```

Voir showlogical.c

Décalages (shift) en C

- Décalage `x << y`
  - Décale x de y positions vers la gauche
  - complète avec des 0
- Décalage `x >> y`
  - Décale x de y positions vers la droite
  - complète avec des 0
  - parfois avec des 1 quand signé

```
int main (int ac, char *av)
{
  char a = 0x05;
  char b = a<<3;
  char c = 0x80;
  unsigned char d = 0x80;

  showbits (a);      00000101
  showbits (a<<3);   00101000
  showbits (b>>3);   00000101
  showbits (c>>3);   11110000
  showbits (d>>3);   00010000
}
```

Voir showshift.c

Échange avec XOR

```
void swap (int *x, int *y)
{
  *x = *x ^ *y; /* #1 */
  *y = *x ^ *y; /* #2 */
  *x = *x ^ *y; /* #3 */
}
```

Voir funswap.c

	*x	*y
début	a	b
#1	a^b	b
#2	a^b	(a^b)^b=a
#3	(a^b)^a=b	a
fin	b	a