Algèbre booléenne et circuits logiques

5.1. L'algèbre de Boole

George **Boole** (1815-1864)

L'**algèbre de Boole**, ou **calcul booléen**, est la partie des mathématiques qui s'intéresse aux opérations et aux fonctions sur les variables logiques. Elle fut inventée par le mathématicien britannique George **Boole**. Aujourd'hui, l'algèbre de Boole trouve de nombreuses applications en informatique et dans la conception des circuits électroniques.

On appelle B l'ensemble constitué de deux éléments appelés **valeurs de vérité** {FAUX, VRAI}. Cet ensemble est aussi noté $B = \{0, 1\}$.

Sur cet ensemble on peut définir les lois **ET** et **OU** et une transformation appelée « **complémentaire** » (parfois « inversion » ou « contraire »).

ET

Elle est définie de la manière suivante : a ET b est VRAI si et seulement si a est VRAI et b est VRAI. Cette loi est aussi notée :

- $a \cdot b$
- $a \wedge b$ (dans quelques notations algébriques, ou en **APL**)
- a & b ou a && b (**Perl**, **C**, **PHP**, ...)
- a AND b (**Ada**, **Pascal**, **Python**, ...)

OU

Elle est définie de la manière suivante : a OU b est VRAI si et seulement si a est VRAI ou b est VRAI, ou si a et b sont VRAIS. Cette loi est aussi notée :

- $a+b$
- $a \vee b$ (dans quelques notations algébriques ou en **APL**)
- $a \mid b$ ou $a \parallel b$ (**Perl**, **C**, **PHP**, ...)
- a OR b (**Ada**, **Pascal**, **Python**, ...)

NON

Le contraire de « a » est VRAI si et seulement si a est FAUX. Le contraire de a est noté :

- $\bar{a}$
- $\neg a$
- $\sim a$ (dans quelques notations algébriques ou en **APL**)
- $!a$ (**C**, **C++**...)
- NOT a (**ASM**, **Pascal**, **Python**, ...)

5.2. Fonctions logiques et tables de vérité

Une **table de vérité** est un tableau qui représente des entrées (en colonne) et des états binaires (0 et 1). Le résultat, exprimé lui aussi sous forme binaire, se lit dans la dernière colonne.

	Symbole de la porte logique (voir § 5.4)	Opération booléenne	Table de vérité
ET (AND)		$A \cdot B$	**Entrées** A B — **Sortie** A AND B: 0 0 → 0; 0 1 → 0; 1 0 → 0; 1 1 → 1
OU (OR)		$A + B$	**Entrées** A B — **Sortie** A OR B: 0 0 → 0; 0 1 → 1; 1 0 → 1; 1 1 → 1
NON (NOT)		$\overline{A}$	**Entrée** A — **Sortie** NOT A: 0 → 1; 1 → 0
NON-ET (NAND)		$\overline{A \cdot B}$	**Entrées** A B — **Sortie** A NAND B: 0 0 → 1; 0 1 → 1; 1 0 → 1; 1 1 → 0
NON-OU (NOR)		$\overline{A + B}$	**Entrées** A B — **Sortie** A NOR B: 0 0 → 1; 0 1 → 0; 1 0 → 0; 1 1 → 0
OU exclusif (XOR)		$A \oplus B$ $= A \cdot \overline{B} + \overline{A} \cdot B$	**Entrées** A B — **Sortie** A XOR B: 0 0 → 0; 0 1 → 1; 1 0 → 1; 1 1 → 0

En électronique, une porte NON est plus communément appelée *inverseur*. Le cercle utilisé sur la représentation est appelé « bulle », et montre qu'une entrée ou une sortie est inversée.

Priorité

Pour faciliter leur compréhension, il a été décidé que ces opérations seraient soumises aux mêmes règles que les opérations mathématiques. La fonction ET (multiplication logique) est ainsi prioritaire par rapport à la fonction OU (somme logique).

On peut évidemment placer des parenthèses dans les opérations pour changer la priorité.

Exercice 5.1

Écrivez les tables de vérité des expressions suivantes :

a. $\overline{A \cdot B}$ b. $A + B \cdot \overline{C}$ c. $A \cdot \overline{B} + (\overline{C} \oplus D)$

Quelques propriétés

Toutes ces propriétés peuvent être facilement démontrées à l'aide de tables de vérité.

Associativité

Comme avec les opérations habituelles, certaines parenthèses sont inutiles :

$$(a + b) + c = a + (b + c) = a + b + c$$

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

Commutativité

L'ordre est sans importance :

$$a + b = b + a$$

$$a \cdot b = b \cdot a$$

Distributivité

Comme avec les opérations mathématiques habituelles, il est possible de distribuer :

$$a \cdot (b + c) = a \cdot b + a \cdot c$$

Attention : comportement différent par rapport aux opérateurs + et · habituels :

$$a + (b \cdot c) = (a + b) \cdot (a + c)$$

Élément neutre

$$a + 0 = a$$

$$a \cdot 1 = a$$

Élément nul

$$0 \cdot a = 0$$

$$1 + a = 1$$

Idempotence

$$a + a + a + [\ldots] + a = a$$

$$a \cdot a \cdot a \cdot [\ldots] \cdot a = a$$

Complémentarité

$$a = \neg(\neg a)$$

$$a + \neg a = 1$$

$$a \cdot \neg a = 0$$

Augustus de Morgan (1806-1871)

Lois de De Morgan

$$\overline{a \cdot b} = \overline{a} + \overline{b}$$

$$\overline{a + b} = \overline{a} \cdot \overline{b}$$

Maurice **Karnaugh**
(né en 1924)

5.3. Tables de Karnaugh

Une table de Karnaugh est une méthode inventée par Maurice **Karnaugh** en 1954 et qui sert à simplifier des équations logiques ou à **trouver l'équation logique correspondant à une table de vérité**

Cette méthode est graphique. Elle fonctionne très bien avec 3 ou 4 variables, beaucoup moins bien avec 5 ou 6 variables, et plus du tout au-delà !

5.3.1. Construction de la table

Soit la table de vérité de **S** suivante avec les variables **A**, **B**, **C** et **D** :

La première colonne ne figure normalement pas dans la table de vérité

Elle est là pour vous aider à remplir les colonnes A, B, C et D. En effet, le nombre ABCD est la représentation binaire du numéro de la ligne.

S'il y a n entrées, il y a 2^n lignes dans la table de vérité

	A	B	C	D	S
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	1
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	0
10	1	0	1	0	1
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	1

La table de Karnaugh correspondante se présente ainsi :

Par exemple, la case sur la ligne 10 et sur la colonne 01 correspond à la valeur de S pour laquelle A=1, B=0, C=0 et D=1

S CD AB	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	0	1	1	1
10	1	0	1	1

5.3.2. Méthode de recherche de l'équation de la table de vérité

- Pour trouver une équation logique, il faut regrouper les valeurs de S égales à 1 dans des rectangles ayant comme nombre de cases une puissance de 2 (16, 8, 4, 2 ou 1 cases).

Tore

- Les groupes formés doivent être les moins nombreux possibles, mais ils doivent **englober tous les 1**. On peut faire des chevauchements.
- On a intérêt à dessiner des rectangles **les plus grands possibles**.
- Cette table est en fait un **tore** (comme dans Pac-Man) : la dernière ligne est adjacente à la première et la première colonne est adjacente à la dernière. On peut ainsi regrouper des 1 se trouvant à ces emplacements.

Pour les tables à 4 variables, il faut de préférence procéder dans l'ordre suivant :

1. les rectangles 8 cases, puis
2. les rectangles 4 cases, puis
3. les rectangles 2 cases, et enfin
4. les cases uniques.

Dans l'exemple ci-contre : on peut former un rectangle de 8 cases (en bleu), puis un carré de 4 (en vert) et enfin on peut rassembler les deux 1 restants dans un carré de 4 (en rouge).

S

CD / AB	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	0	1	1	1
10	1	0	1	1

Mise en formule

Pour chaque rectangle, on regarde les coordonnées des lignes (pour les lettres A et B) et des colonnes (pour les lettres C et D) qui touchent ce rectangle.

S

CD / AB	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	0	1	1	1
10	1	0	1	1

Ci-dessus, on a coloré en bleu les coordonnées des lignes et des colonnes touchées par le rectangle bleu.

On voit que A, B et D prennent les valeurs 0 ou 1 : elles ne figureront pas dans la formule. Par contre, C vaut toujours 1. La formule de ce rectangle sera simplement « C ».

Les rectangles de taille 8 auront 1 lettre dans leur formule, ceux de taille 4 auront 2 lettres, ceux de taille 2 auront 3 lettres et ceux de taille 1 auront 4 lettres.

S

CD / AB	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	0	1	1	1
10	1	0	1	1

Pour le carré vert, on voit que A et C valent soit 0, soit 1 : ils ne figureront pas dans la formule. Par contre, B et D valent toujours 1. Le carré vert correspondra à la formule « B et D ».

S

AB \ CD	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	0	1	1	1
10	1	0	1	1

Pour le carré rouge, on voit que A et C valent soit 0, soit 1 : ils ne figureront pas dans la formule. Par contre, B et D valent toujours 0. La formule du carré rouge sera « $\overline{B}$ et $\overline{D}$ ». **En effet, quand la valeur constante est 0, on met une barre sur la lettre concernée.**

On fait ensuite l'union des trois formules : « S = C ou (B et D) ou ($\overline{B}$ et $\overline{D}$) », que l'on peut aussi écrire sous la forme de l'équation « $S = C + B \cdot D + \overline{B} \cdot \overline{D}$ ». On peut au besoin simplifier la formule en utilisant les propriétés de la page 3. On remarque au passage que l'entrée A n'a aucune influence sur la sortie S. On peut donc la supprimer purement et simplement de l'équation.

Exercice 5.2

Trouvez les équations des tables de vérité de S, T et U avec les variables A, B, C et D.

A	B	C	D	S	T	U
0	0	0	0	0	1	0
0	0	0	1	0	1	1
0	0	1	0	0	0	1
0	0	1	1	0	1	1
0	1	0	0	0	0	0
0	1	0	1	1	1	1
0	1	1	0	1	1	0
0	1	1	1	1	1	0
1	0	0	0	0	0	0
1	0	0	1	1	1	1
1	0	1	0	0	1	1
1	0	1	1	1	1	1
1	1	0	0	0	1	0
1	1	0	1	0	1	1
1	1	1	0	1	1	0
1	1	1	1	1	1	0

Vérifiez vos formules à l'aide d'un programme Python ou d'un tableur.

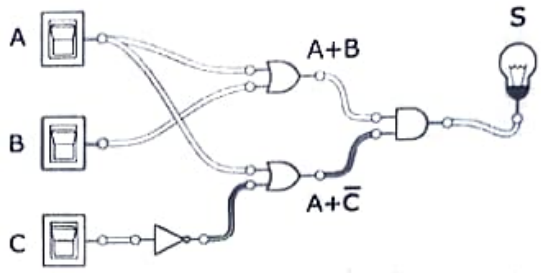
5.4. Circuits logiques

On appelle circuit logique un assemblage de portes logiques (voir § 5.4) reliées entre elles pour schématiser une expression algébrique.

Tout ordinateur est conçu à partir de circuits intégrés qui ont tous une fonction spécialisée. Ces circuits sont fait à partir de circuits logiques dont le but est d'exécuter des opérations sur des variables logiques (binaires).

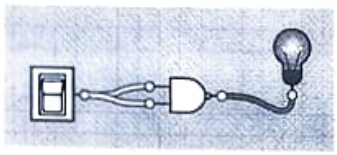
Par exemple, l'expression algébrique $S=(A+B)\cdot(A+\overline{C})$ sera schématisée comme suit :

Les circuits logiques ont été dessinés grâce au programme Logicly. http://logic.ly/

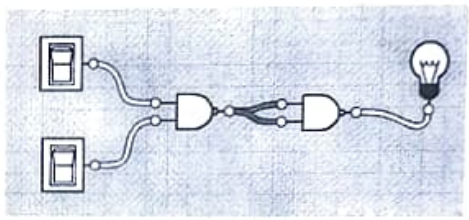


La porte NAND

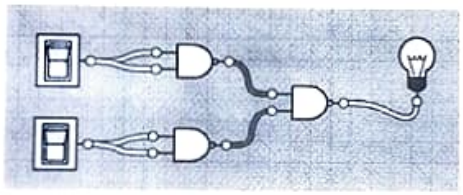
La porte NAND est la plus simple à réaliser du point de vue technologique. Il est possible de réaliser toutes les fonctions logiques en utilisant uniquement ce type de porte.



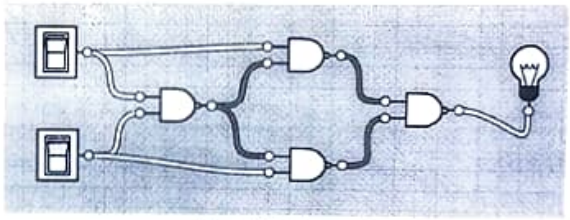
Porte NON



Porte ET



Porte OU



Porte OU exclusif

Exercice 5.3

Reprenez les équations trouvées à l'exercice 5.2, simplifiez-les si besoin grâce aux propriétés des fonctions logiques, puis construisez les circuits logiques correspondants.

Exercice 5.4

Réalisez une porte XOR avec des portes AND, OR et NOT.

Exercice 5.5

On a trois interrupteurs pouvant être en position 0 ou 1 et trois ampoules pouvant être allumées ou éteintes. On veut créer un circuit logique où le nombre de lampes allumées correspond au nombre d'interrupteurs positionnés sur 1, mais on ne veut pas savoir quels interrupteurs le sont.

1. Établissez les tables de vérité de ce problème.
2. Trouvez l'équation la plus simple possible pour chaque table de vérité.
3. Dessinez le circuit logique correspondant.