

## Cycle 9 – Modéliser, prévoir et vérifier des performances des systèmes combinatoires

### 1. INTRODUCTION

Dans la réalisation des commandes automatiques, les systèmes logiques ont un rôle relativement important aux côtés des systèmes linéaires, continus et invariants.

Les signaux d'entrée dans les systèmes logiques prennent des valeurs binaires 0 et 1. On parle parfois de «tout ou rien» (T.O.R.).

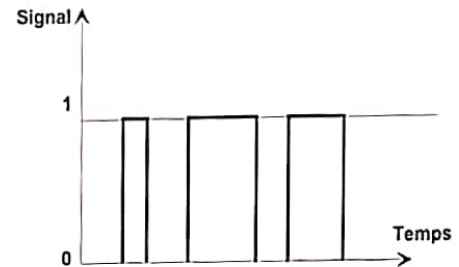


Fig. 1 : évolution temporelle d'un signal type TOR

La représentation ci-dessus est un modèle, et en réalité on constate des temps de réponse plus ou moins longs avant que le signal n'atteigne les valeurs 0 ou 1. Le comportement «tout ou rien» des composants ne correspond qu'au comportement normalement prévu en régime stabilisé et en l'absence de tout dysfonctionnement (composants parfaits).

On distingue deux types de systèmes logiques :

- **les systèmes combinatoires** (statiques ou sans mémoire) qui ne tiennent pas compte des états antérieurs du système.

Par exemple un distributeur de boissons, un afficheur à cristaux liquides peuvent être considérés comme des systèmes combinatoires.

- **les systèmes séquentiels** (dynamiques ou avec mémoire) qui tiennent compte des états antérieurs du système.

Par exemple, un ascenseur, une machine à laver peuvent être considérés comme des systèmes séquentiels.

En fait les automatismes, relatifs aux technologies électromagnétiques ou pneumatiques, existent depuis longtemps. On les définissait par les schémas de liaisons électriques ou pneumatiques, entre leurs différents organes. À l'heure actuelle, en raison de la diversité des moyens dont on dispose (commandes électromagnétique, électronique, programmée, pneumatique,...) et de la complexité de plus en plus grande des systèmes, il est indispensable de pouvoir traiter un automate par une méthode mathématique rigoureuse indépendante de la technologie utilisée. Pour cela on fait appel aux méthodes logiques : conduisant à l'établissement d'expressions algébriques binaires qui s'appliquent à tous les modes de réalisation.

Pour son étude, un système combinatoire est considéré comme une boîte noire : **on s'intéresse uniquement aux relations entrées-sorties.**

20  
20

## 2. LE LANGAGE BINAIRE

Coder une information revient à fixer une convention permettant de lire et écrire cette information sans ambiguïté. Les chiffres arabes sont notre code humain pour les données numériques. Mais ce code est peu approprié aux machines.

Les systèmes automatiques exploitent des codes basés sur le binaire pour des raisons d'architecture. Les qualités requises pour un code sont principalement :

- la taille du codage (nombre de bits (binary digit) nécessaires),
- la fiabilité de lecture,
- la simplicité de manipulation.

Le fait d'écrire un nombre décimal dans une autre base s'appelle **codage** (par exemple : conversion décimal  $\rightarrow$  binaire).

Le fait de ramener en décimal un nombre codé s'appelle **décodage** (par exemple : conversion binaire pur  $\rightarrow$  décimal).

Le fait de transcrire un nombre déjà codé dans un autre code s'appelle **transcodage** (par exemple : conversion binaire pur  $\rightarrow$  binaire réfléchi).

### 2.1 Code binaire pur ou code binaire naturel

Un nombre ou un chiffre peut être exprimé dans n'importe quelle base (le code binaire est associé à la base 2). Il faut tenir compte de cette remarque pour optimiser le codage de l'information. Il suffit alors de prendre la base la plus appropriée pour exprimer un chiffre ou un nombre.

La forme générale d'un nombre écrit dans une base  $X$  est :

$$\text{Nombre}_{(x)} = \alpha_n \cdot x^n + \alpha_{n-1} \cdot x^{n-1} + \alpha_{n-2} \cdot x^{n-2} + \dots + \alpha_2 \cdot x^2 + \alpha_1 \cdot x^1 + \alpha_0 \cdot x^0, \text{ avec } \alpha < x$$

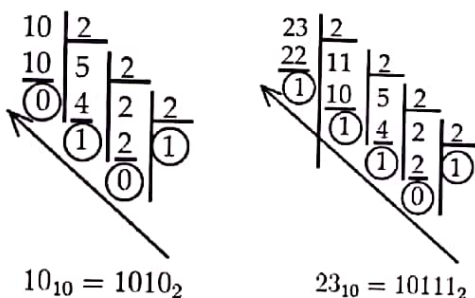
Donc dans la base 2, un nombre s'écrit :

$$\text{Nombre}_{(2)} = \alpha_n \cdot 2^n + \alpha_{n-1} \cdot 2^{n-1} + \alpha_{n-2} \cdot 2^{n-2} + \dots + \alpha_2 \cdot 2^2 + \alpha_1 \cdot 2^1 + \alpha_0 \cdot 2^0, \text{ avec } \alpha < 2$$

### 2.2 Principe de conversion

#### 2.2.1 Conversion décimal - binaire

Pour écrire un nombre décimal en binaire, il faut faire des divisions successives par 2. Par exemple pour écrire les chiffres décimaux 10 et 23 en binaire :



Le sens de lecture, et donc d'écriture, se fait selon la flèche. Donc :

$$10_{(10)} = 1010_{(2)}$$

$$23_{(10)} = 10111_{(2)}$$

Pour les nombres négatifs ou les réels, il faut consulter des manuels adéquats.

### 2.2.2 Conversion binaire - décimal

Pour cela, il faut utiliser la relation donnée précédemment, soit :

$$\text{Nombre}_{(2)} = \alpha_n \cdot 2^n + \alpha_{n-1} \cdot 2^{n-1} + \alpha_{n-2} \cdot 2^{n-2} + \dots + \alpha_2 \cdot 2^2 + \alpha_1 \cdot 2^1 + \alpha_0 \cdot 2^0, \text{ avec } \alpha < 2$$

Donc :  $10111_{(2)} = 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$

D'où :  $10111_{(2)} = 1 \cdot 16 + 0 \cdot 8 + 1 \cdot 4 + 1 \cdot 2 + 1 \cdot 1 = 23_{(10)}$

Nous obtenons donc le tableau ci-dessous :

Décimal	Binaire pur
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1

Décimal	Binaire pur
8	1 0 0 0
9	1 0 0 1
10	1 0 1 0
11	1 0 1 1
12	1 1 0 0
13	1 1 0 1
14	1 1 1 0
15	1 1 1 1

### 2.2.3 Opérations en code binaire pur (hors programme, donné à titre informatif pour votre culture générale)

**Addition :**

Les règles d'addition sont les suivantes :

0	0	1	1	11	11
+ 0	+ 1	+ 0	+ 1	+ 01	+ 11
= 0	= 1	= 1	= 10	= 100	= 110

Les calculs s'effectuent de droite à gauche. Prenons l'exemple ci-dessous :

Décimal	Binaire
23	1 <sup>1</sup> 0 <sup>1</sup> 1 <sup>1</sup> 11
+ 10	+ 1 0 1 0
33	10 0 0 0 1

- 1+0=1,
- 1+1=0 et une retenue de 1,
- 1+0=1, mais il faut ajouter à ce résultat une retenue de 1, soit 1+1+0=0 et une retenue de 1
- 0+1=1, calcul auquel on ajoute une retenue de 1, soit 0 et une retenue de 1,
- 1 et une retenue de 1 font 0 et une retenue de 1. Cette retenue étant la dernière, on la place à gauche du résultat.

Nous pouvons appliquer cette méthode à l'exemple ci-dessous :

Décimal	Binaire
63	1 1 1 1 1 1
+ 43	+ 1 0 1 0 1 1
106	1 1 0 1 0 1 0

**Soustraction :**

Les règles de soustraction sont les suivantes :

0	0	1	1	10
- 0	- 1	- 0	- 1	- 01
= 0	= ?	= 1	= 0	= 01

Les calculs s'effectuent de droite à gauche. Prenons l'exemple ci-dessous :

Décimal	Binaire
23	1 <sub>1</sub> 0111
- 10	- 1 1010
13	0 1101

- 1-0=0
- 1-1=0
- 1-0=0
- 0-1=1 et une retenue de 1,
- 1-(0+1)=0

Nous pouvons appliquer cette méthode à l'exemple ci-dessous :

Décimal	Binaire
63	111111
- 43	- 101011
20	010100

### Multiplication :

Les règles de multiplication sont les suivantes :

0	0	1	1	—	—	—	—
*0	*1	*0	*1				
=0	=0	=0	=1				

Les calculs s'effectuent de droite à gauche (voir exemple ci-dessous) :

Décimal	Binaire
23	1 0 1 1 1
× 3	× 1 1
69	1 1 0 1 1 1 1
	+ 1 0 1 1 1 0
	1 0 0 0 1 0 1

### Division :

Elle s'effectue comme une division ordinaire :

Décimal	Binaire
23   3	10111   11
- 2	- 11
020	0101
-18	- 11
2	101
	- 11
	100
	- 11
	100

### Remarque :

Le résultat obtenu en base 2 correspond à 7,66 en base 10. En effet 111,101010 s'écrit  $(1*2^2)+(1*2^1)+(1*2^0)+(1*2^{-1})+(0*2^{-2})+(1*2^{-3})+(0*2^{-4})+(1*2^{-5})+(1*2^{-6})$  en base 10.

- $(1*2^2)+(1*2^1)+(1*2^0)$  correspond à la partie entière et vaut 7
- $(1*2^{-1})+(0*2^{-2})+(1*2^{-3})+(0*2^{-4})+(1*2^{-5})+(1*2^{-6})$  correspond à la partie décimale et vaut 0,666.

## 2.3 Différents codes binaires

Il est possible de réaliser avec une suite de nombres binaires donnés, toute une série de codages en permutant les lignes (c'est le transcodage). Les valeurs binaires 0 et 1 sont appelées «bit».

*L'existence de ces différents codes est à connaître, certains seront étudiés plus en détail en TD.*

*Le code Gray (cf. 2.3.1.) est à savoir retrouver. Pour les autres codes, la méthode de codage/décodage vous sera redonnée si vous êtes amenés à les utiliser.*

### 2.3.1 Code binaire réfléchi ou code Gray

Ce code est un arrangement du système binaire. Il présente la propriété d'un seul changement de variable d'une ligne à l'autre.

Décimal	Binaire pur	Binaire réfléchi	
0	0 0 0 0	0 0 0 0	
1	0 0 0 1	0 0 0 1	Première symétrie
2	0 0 1 0	0 0 1 1	
3	0 0 1 1	0 0 1 0	Deuxième symétrie
4	0 1 0 0	0 1 1 0	
5	0 1 0 1	0 1 1 1	
6	0 1 1 0	0 1 0 1	
7	0 1 1 1	0 1 0 0	Troisième symétrie
8	1 0 0 0	1 1 0 0	
9	1 0 0 1	1 1 0 1	
10	1 0 1 0	1 1 1 1	
11	1 0 1 1	1 1 1 0	
12	1 1 0 0	1 0 1 0	
13	1 1 0 1	1 0 1 1	
14	1 1 1 0	1 0 0 1	
15	1 1 1 1	1 0 0 0	

Pour obtenir le code Gray, il faut faire toutes les deux, quatre, huit,... lignes une symétrie en commençant par le bit de droite et changer le bit de gauche. La première symétrie porte sur deux bits de la première colonne de droite. La deuxième symétrie porte sur huit bits des deux premières colonnes de droite. La troisième symétrie porte sur vingt-quatre bits des trois premières colonnes de droite et ainsi de suite.

Ce code est très utilisé pour décrire des automatismes (un seul bit change lors du changement d'état d'un composant), en particulier dans les codeurs (cf. figure ci-dessous). Il apporte une garantie d'interprétation avec une erreur maximale d'une incrémentation.

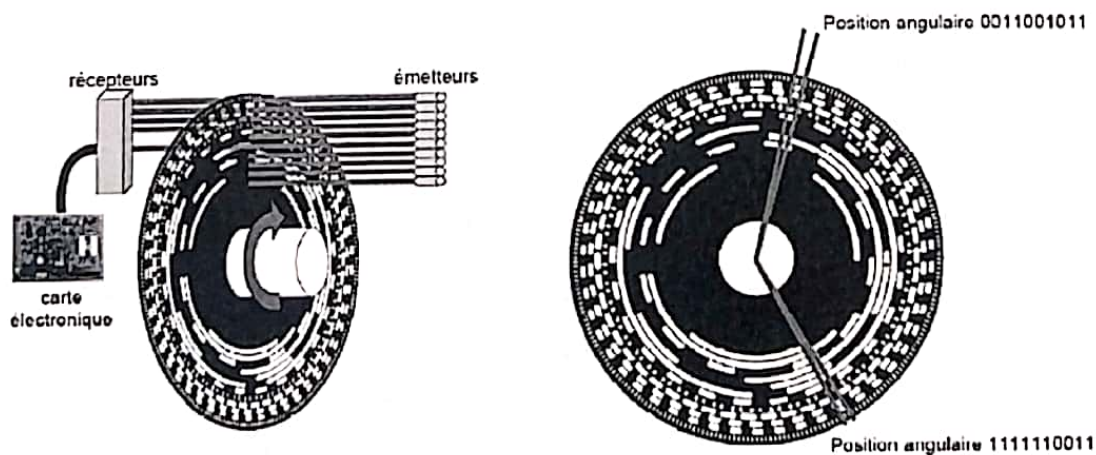


Figure 2 : utilisation du code Gray pour la réalisation d'un capteur de position absolu

### 2.3.2 Code décimal codé binaire (Code DCB)

C'est un code basé sur le code binaire pur, mais qui est adapté à la représentation des nombres en base 10. En effet le code binaire pur n'associe pas de bits spécifiques aux unités, dizaines, centaines,... La propriété du code DCB est d'associer quatre bits différents à chaque puissance de 10. Ainsi  $2005_{(10)}$  se code de la façon suivante :


Décimal	2	0	0	5
DCB	0010	0000	0000	0101

Décimal	Binaire	Décimal codé Binaire
$2005_{(10)}$	$11111010101_{(2)}$	0010000000000101

Ce code est par exemple utilisé sur les afficheurs 7 segments (chaque afficheur reçoit le code correspondant au chiffre à afficher).

Décimal	2	0	0	5
DCB	0010	0000	0000	0101



Cette représentation adaptée à la représentation binaire des nombres décimaux utilise un nombre de bits supérieurs à celui du binaire naturel (16 au lieu de 11, dans l'exemple traité ci-dessus), et donc une place plus importante en mémoire de l'ordinateur.

**Remarque :** jusqu'à présent, ce paragraphe ne fait référence qu'à la base 2. Mais pour le codage de l'information, d'autres bases sont utilisées, comme par exemple le code hexadécimal, le code 3 parmi 5 ou le codage ASCII.

### 2.3.3 Code hexadécimal

C'est un code en base 16, les caractères utilisés sont : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Les valeurs binaires de ces caractères correspondent respectivement aux seize premières lignes du code binaire pur.

base 10	base 2	base 16
0	0	0
1	1	1
2	10	2
3	11	3
4	100	4
5	101	5
6	110	6
7	111	7
8	1000	8

base 10	base 2	base 16
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Comme dans le cas de la conversion décimal - binaire, la conversion décimal - hexadécimal se fait par des divisions successives par 16, et la conversion hexadécimal - décimal se fait avec la formule suivante :

$$\text{Nombre}_{(16)} = \alpha_n \cdot 16^n + \alpha_{n-1} \cdot 16^{n-1} + \alpha_{n-2} \cdot 16^{n-2} + \dots + \alpha_2 \cdot 16^2 + \alpha_1 \cdot 16^1 + \alpha_0 \cdot 16^0$$

Ainsi, si l'on s'intéresse au codage du nombre décimal 3135 :

Décimal	Binaire	Hexadécimal
$3135_{(10)}$	$0000110000111111_{(2)}$	$0C3F_{(16)}$

Hexadécimal	0	C	3	F
Binaire	0000	1100	0011	1111

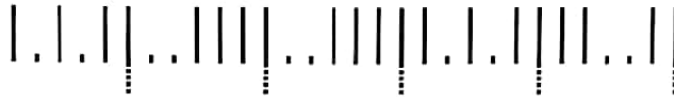
Ce code est très utilisé dans les automates programmables. C'est la base «naturelle» de l'informaticien, car elle permet de contracter les informations données en base 2 par paquets de quatre bits. Dans l'exemple ci-dessus, le nombre 3135 est ainsi codé sur 4 bits au lieu de 16.

### 2.3.4 Code 3 parmi 5

Ce code consiste à affecter choisir 3 bits à 1 parmi 5 bits. Le nombre de combinaisons possibles est 10, il permet donc de coder les dix chiffres décimaux.

Décimal	3 parmi 5
0	00111
1	01011
2	01101
3	01110
4	10011
5	10101
6	10110
7	11001
8	11010
9	11100

Ce code est celui utilisé à la poste pour la lecture du code postal. Par exemple, le code ci-dessous est celui relevé sur une lettre arrivée au lycée (et doit donc correspondre au code postal 75005 - la lecture se faisant de droite à gauche).



### 2.3.5 Codage ASCII (American Standard Code for Information Interchange)

Le code ASCII permet de coder sur 8 bits les 256 caractères classiques du clavier :

- l'alphabet majuscule et minuscule,
- les chiffres,
- les lettres accentuées ou spéciales des langues européennes,
- la ponctuation.

Les mails sont transférés par code ASCII.

### 3. ALGÈBRE ENSEMBLISTE

Il ne s'agit pas de faire un cours exhaustif de mathématiques sur l'algèbre ensembliste. Seules les définitions et les propriétés nécessaires à l'étude de systèmes combinatoires seront présentées.

#### 3.1 Parties d'un ensemble

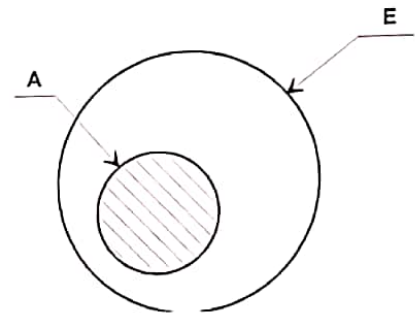
On appelle partie d'un ensemble E, tout sous-ensemble de E.

#### 3.2 Ensemble complémentaire - Ensemble vide

L'ensemble des éléments de E n'appartenant pas à A, sous-ensemble de E, est appelé **ensemble complémentaire** de A et noté  $\bar{A}$ . Dans le cas où A coïncide avec E,  $\bar{A}$  ne contient aucun élément et est appelé **ensemble vide** et noté  $\emptyset$ .

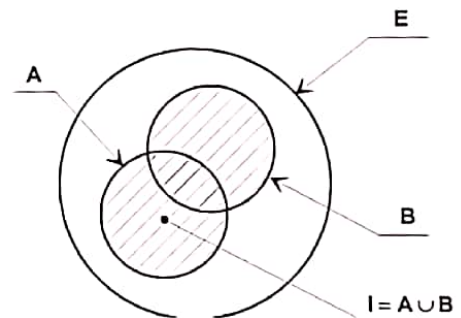
Nous avons donc les relations suivantes :

$$A \cup \bar{A} = E ; A \cap \bar{A} = \emptyset ; \bar{\bar{A}} = A$$



#### 3.3 Loi Union

On appelle union des sous-ensembles A et B de E, le sous-ensemble I formé des éléments de E qui appartiennent à l'un au moins des sous-ensembles A et B.



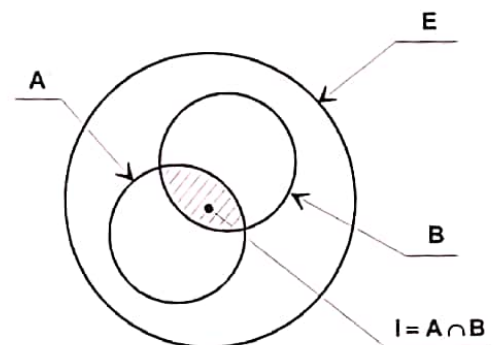
Il est clair que les relations ci-dessous sont vérifiées :

Commutativité	Associativité	Élément neutre
$A \cup B = B \cup A$	$A \cup (B \cup C) = (A \cup B) \cup C$	$A \cup \emptyset = \emptyset \cup A = A, \forall A$

La loi Union est donc une loi de composition interne.

#### 3.4 Loi Intersection

On appelle intersection des sous-ensembles A et B de E, le sous-ensemble I formé des éléments de E qui appartiennent simultanément aux deux sous-ensembles A et B.





Il est clair que les relations ci-dessous sont vérifiées :

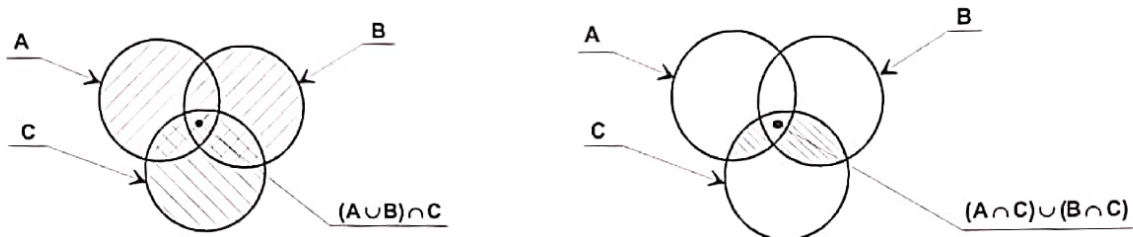
Commutativité	Associativité	Elément neutre
$A \cap B = B \cap A$	$A \cap (B \cap C) = (A \cap B) \cap C$	$A \cap E = E \cap A = A, \forall A$

La loi Intersection est donc une loi de composition interne.

### 3.5 Distributivité de chacune des lois par rapport à l'autre

#### 3.5.1 Distributivité de la loi Intersection par rapport à la loi Union

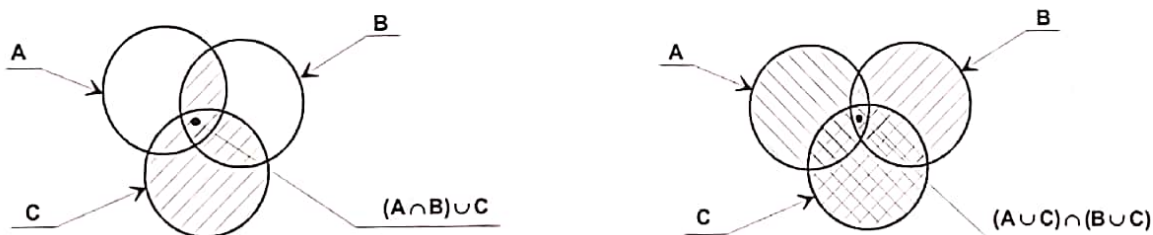
Les figures ci-dessous nous permettent de vérifier cette propriété.



Nous avons :  $(A \cup B) \cap C = (A \cap C) \cup (B \cap C)$

#### 3.5.2 Distributivité de la loi Union par rapport à la loi Intersection

Les figures ci-dessous nous permettent de vérifier cette propriété.



Nous avons :  $(A \cap B) \cup C = (A \cup C) \cap (B \cup C)$

### 3.6 Propositions contradictoires

#### 3.6.1 Définition

On appelle propositions contradictoires, deux propositions  $x$  et  $\bar{x}$  telles que :

- l'une des deux propositions est nécessairement vraie,
- les deux propositions ne sont pas vérifiées en même temps.

Par exemple dans un circuit électrique, «le courant passe» et «le courant ne passe pas» sont deux propositions contradictoires.

#### 3.6.2 Loi OU (ou somme) notée +

On appelle somme de deux propositions  $x$  et  $y$ , la proposition  $z$  définie par : l'une des deux propositions  $x$  et  $y$  est satisfaite.

$$z = x + y$$

### 3.6.3 Loi ET (ou produit) notée .

On appelle produit de deux propositions  $x$  et  $y$ , la proposition  $z$  définie par : **les deux propositions  $x$  et  $y$  sont simultanément satisfaites.**

$$z = x . y$$

### 3.6.4 Conséquences

Si  $x$ ,  $y$  et  $z$  sont respectivement des propositions vérifiées pour les ensembles  $A$ ,  $B$  et  $C$ , nous avons alors les relations :

$$(x + y) . z = x . z + y . z$$

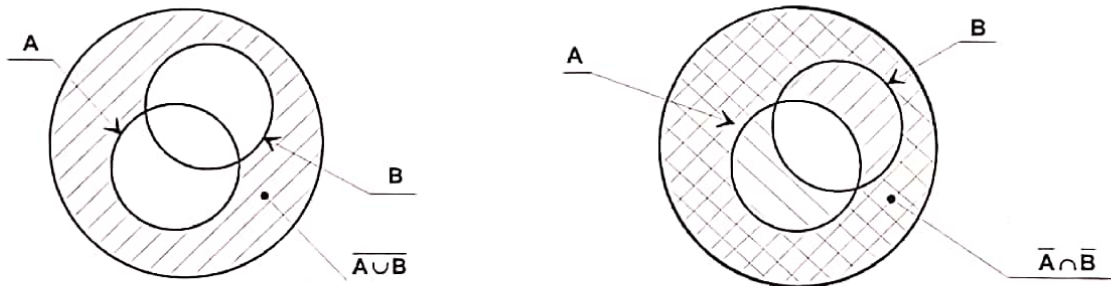
$$(x . y) + z = (x + z) . (y + z)$$

En effet ces deux relations sont déduites des relations respectivement démontrées aux paragraphes 3.5.1. et 3.5.2.

## 3.7 Théorèmes de De MORGAN

### 3.7.1 Premier théorème

Soient les figures ci-dessous :



On remarque que  $\overline{A \cup B} = \bar{A} \cap \bar{B}$ . Donc si  $x$  est une proposition vérifiée pour  $A$  et  $y$  une proposition vérifiée pour  $B$ , nous avons :

$$\overline{x + y} = \bar{x} . \bar{y}$$

Cette relation peut être généralisée et s'écrire :

$$\overline{\sum_{i=1}^n x_i} = \prod_{i=1}^n \bar{x}_i$$

#### Démonstration pour trois variables :

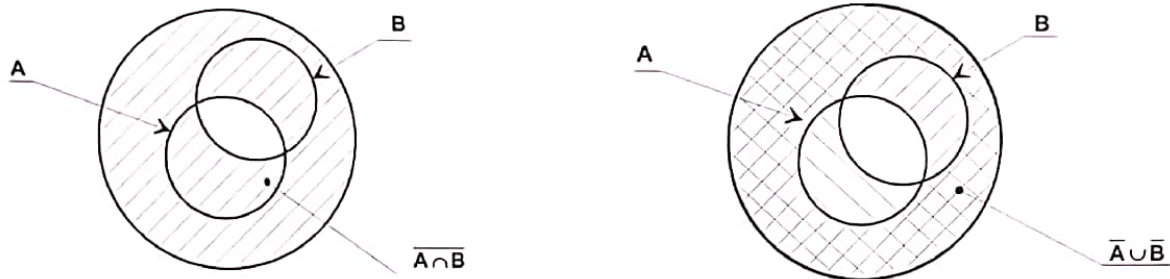
Si  $y = t + z$  alors,  $\overline{x + y} = \overline{x + (t + z)} = \bar{x} . \overline{(t + z)} = \bar{x} . (\bar{t} . \bar{z}) = \bar{x} . \bar{t} . \bar{z}$

Or :  $\overline{x + (t + z)} = \overline{x + t + z}$

Donc :  $\overline{x + t + z} = \bar{x} . \bar{t} . \bar{z}$

### 3.7.2 Deuxième théorème

Soient les figures ci-dessous :



On remarque que  $\overline{A \cap B} = \bar{A} \cup \bar{B}$ . Donc si  $x$  est une proposition vérifiée pour  $A$  et  $y$  une proposition vérifiée pour  $B$ , nous avons :

$$\overline{x \cdot y} = \bar{x} + \bar{y}$$

Cette relation peut être généralisée et s'écrire :

$$\overline{\prod_{i=1}^n x_i} = \sum_{i=1}^n \bar{x}_i$$

#### Démonstration pour trois variables :

Si  $y = t \cdot z$  alors,  $\overline{x \cdot y} = \overline{x \cdot (t \cdot z)} = \bar{x} + \overline{(t \cdot z)} = \bar{x} + (\bar{t} + \bar{z}) = \bar{x} + \bar{t} + \bar{z}$

Or :  $\overline{x \cdot (t \cdot z)} = \overline{x \cdot t \cdot z}$

Donc :  $\overline{x \cdot t \cdot z} = \bar{x} + \bar{t} + \bar{z}$

### 3.7.3 Troisième théorème

Comme  $\bar{\bar{A}} = A$  alors :  $\bar{\bar{x}} = x$

## 3.8 Formules d'absorption

La figure, ci-contre, prouve que  $A + A \cdot B = A$

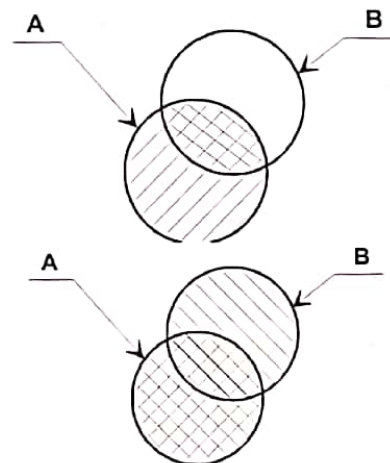
Donc si  $x$  est une proposition vérifiée pour  $A$  et  $y$  une proposition vérifiée pour  $B$ , nous avons :

$$x + x \cdot y = x$$

La figure, ci-contre, prouve que  $A \cdot (A + B) = A$

Donc si  $x$  est une proposition vérifiée pour  $A$  et  $y$  une proposition vérifiée pour  $B$ , nous avons :

$$x \cdot (x + y) = x$$



## 4. ALGÈBRE DE BOOLE

C'est une algèbre de propositions logiques mise au point par un mathématicien anglais Georges BOOLE (1815 - 1864).

Les variables de l'algèbre de BOOLE ne peuvent prendre que deux valeurs possibles, par convention : 0 et 1. Elles correspondent à des propositions ne comportant que deux réponses possibles : vrai ou faux. Ces variables sont appelées variables booléennes ou variables binaires.

### 4.1 Variable booléenne

Soit  $X$  une proposition et  $\bar{X}$  la proposition contradictoire correspondante. Associons à  $X$  la variable booléenne  $x$  et à  $\bar{X}$  la variable booléenne  $\bar{x}$ . Si la proposition  $X$  est satisfaite, alors  $x = 1 \Leftrightarrow \bar{x} = 0$ . Si la proposition  $X$  n'est pas satisfaite, alors  $x = 0 \Leftrightarrow \bar{x} = 1$ .

Tout état logique peut être représenté par une variable booléenne :

- proposition : vraie  $\rightarrow x = 1$ , fausse  $\rightarrow x = 0$ .
- circuit (électrique, pneumatique,...) : fermé  $\rightarrow x = 1$ , ouvert  $\rightarrow x = 0$ .
- lampe : allumée  $\rightarrow x = 1$ , éteinte  $\rightarrow x = 0$ .
- moteur : marche  $\rightarrow x = 1$ , arrêt  $\rightarrow x = 0$ .
- vérin : en pression  $\rightarrow x = 1$ , à l'échappement  $\rightarrow x = 0$ .
- action  $\rightarrow x = 1$ ; repos  $\rightarrow x = 0$ .
- niveau électrique : haut  $\rightarrow x = 1$ ; bas  $\rightarrow x = 0$ .

D'une manière générale, on peut associer une variable booléenne à toute situation modélisée en TOUT ou RIEN (T.O.R.), c'est-à-dire à des situations ne faisant pas intervenir des effets transitoires. Par exemple, si on s'intéresse à l'état de fonctionnement du moteur électrique, deux cas existent : soit le rotor du moteur tourne, soit il ne tourne pas.

Par contre, si on mesure la fréquence de rotation du rotor, une multitude de cas apparaissent entre l'immobilité et la fréquence nominale (fréquence d'utilisation optimale).

La variable décrivant la fréquence de rotation du rotor n'est plus une variable **logique** mais une variable :

- **analogique** si la mesure est continue,
- **échantillonnée** si la mesure est discontinue.

### 4.2 Fonction booléenne d'une variable quelconque

Une telle fonction transforme une variable quelconque (analogique ou numérique) en variable logique. On dit qu'on a affaire à un convertisseur analogique-logique ou numérique-logique.

Les deux exemples, ci-dessous, sont conçus à partir de cette fonction :

- un interrupteur thermostatique dont l'entrée est une température variable et la sortie entraîne la fermeture ou l'ouverture d'un circuit.
- un détecteur d'obscurité dont l'entrée est un flux lumineux variable et la sortie est un signal électrique haut ou bas.

### 4.3 Fonctions booléennes d'une variable binaire

Une variable binaire  $a$  peut prendre deux ( $2^1$ ) états. La fonction  $F(a)$  peut renvoyer 0 ou 1. Il existe donc quatre ( $2^{1 \cdot 2}$ ) fonctions booléennes d'une variable binaire.

Considérons l'ensemble des possibilités et consignons-les dans un tableau.

#### 4.3.1 Fonction identité, ou fonction oui

La fonction  $F(a) = a$  est la fonction identité, ou fonction « oui ».

$a$	$F(a)$
0	0
1	1

#### 4.3.2 Fonction complémentation

La fonction  $F(a) = \bar{a}$ , lue « a barre », est la fonction complémentation, ou fonction « non ».

$a$	$F(a)$
0	1
1	0

#### 4.3.3 Première fonction constante

La fonction  $F(a) = 1$  est la fonction constante « toujours vraie ».

$a$	$F(a)$
0	1
1	1

#### 4.3.4 Deuxième fonction constante

La fonction  $F(a) = 0$  est la fonction constante « toujours fausse ».

$a$	$F(a)$
0	0
1	0

### 4.4 Fonction booléenne de deux variables binaires

Un couple de variables binaires  $a$  et  $b$  peut prendre quatre ( $2^2$ ) états. La fonction  $F(a, b)$  peut renvoyer 0 ou 1. Il existe donc seize ( $2^{2 \cdot 2}$ ) fonctions booléennes de deux variables binaires.

Considérons l'ensemble des possibilités et consignons-les dans un tableau.

a	b	F <sub>1</sub>	F <sub>2</sub>	F <sub>3</sub>	F <sub>4</sub>	F <sub>5</sub>	F <sub>6</sub>	F <sub>7</sub>	F <sub>8</sub>	F <sub>9</sub>	F <sub>10</sub>	F <sub>11</sub>	F <sub>12</sub>	F <sub>13</sub>	F <sub>14</sub>	F <sub>15</sub>	F <sub>16</sub>
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0
1	1	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0
1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0

On retrouve les fonctions déjà vues au paragraphe précédent :

- $F_3(a, b) = 0$  et  $F_{11}(a, b) = 1$ , fonctions constantes,
- $F_3(a, b) = a$  et  $F_5(a, b) = b$ , fonction «oui»,
- $F_9(a, b) = \bar{a}$  et  $F_{15}(a, b) = \bar{b}$ , fonction «non».

Nous allons étudier les autres fonctions.

#### 4.4.1 La fonction OU (OR)

$F_6(a, b) = a + b$  est la fonction OU notée  $a + b$ , lue : « a ou b », qui vaut 1 si une au moins des variables vaut 1.

a	b	$F(a, b)$
0	0	0
0	1	1
1	1	1
1	0	1

#### 4.4.2 La fonction ET (AND)

$F_4(a, b) = a \cdot b$  est la fonction ET notée  $a \cdot b$ , lue : « a et b », qui vaut 1 si les deux variables valent 1.

a	b	$F(a, b)$
0	0	0
0	1	0
1	1	1
1	0	0

#### 4.4.3 La fonction complément de la fonction ET (NAND)

$F_{10}(a, b) = \overline{a \cdot b}$ , est le complément de la fonction ET, et est appelée fonction NAND.

a	b	$F(a, b)$
0	0	1
0	1	1
1	1	0
1	0	1

#### 4.4.4 La fonction complément de la fonction OU (NOR)

$F_{16}(a, b) = \overline{a + b}$ , est le complément de la fonction OU, et est appelée fonction NOR ou NI.

a	b	$F(a, b)$
0	0	1
0	1	0
1	1	0
1	0	0

#### 4.4.5 La fonction Inhibition

$F_8(a, b) = \bar{a}.b$  et  $F_2(a, b) = a.\bar{b}$  sont deux expressions de la fonction **Inhibition**.

a	b	$F_8(a, b)$	$F_2(a, b)$
0	0	0	0
0	1	1	0
1	1	0	0
1	0	0	1

La fonction  $F_8(a, b)$  est égale à b uniquement si  $a = 0$ . On dit que a inhibe b. De même la fonction  $F_2(a, b)$  est égale à a uniquement si  $b = 0$ . On dit que b inhibe a.

#### 4.4.6 La fonction Implication

$F_{14}(a, b)$  et  $F_{12}(a, b)$  sont les compléments respectifs des fonctions  $F_8(a, b) = \bar{a}.b$  et  $F_2(a, b) = a.\bar{b}$ , et sont appelées fonctions **Implication**. Elles s'écrivent respectivement  $a + \bar{b}$  et  $\bar{a} + b$ .

a	b	$F_{14}(a, b)$	$F_{12}(a, b)$
0	0	1	1
0	1	0	1
1	1	1	1
1	0	1	0

La fonction  $F_{12}(a, b)$  est égale à b uniquement si  $a = 1$ . On dit que a implique b. De même la fonction  $F_{14}(a, b)$  est égale à a uniquement si  $b = 1$ . On dit que b implique a.

#### 4.4.7 La fonction OU exclusif (XOR)

$F_7(a, b) = \bar{a}.b + a.\bar{b}$  est la fonction **OU exclusif**. Elle est notée aussi  $a \oplus b$ . Elle prend la valeur 1 si une seule des variables vaut 1.

a	b	$F(a, b)$
0	0	0
0	1	1
1	1	0
1	0	1

#### 4.4.8 La fonction complément de la fonction OU exclusif parfois appelée ET exclusif

$F_{13}(a, b) = a.b + \bar{a}.\bar{b}$  est la fonction complément de la fonction OU exclusif. En effet  $\overline{\bar{a}.b + a.\bar{b}} = \bar{a}.\bar{b} + a.b = (a + \bar{b}).(\bar{a} + b)$ .

Donc  $\overline{\bar{a}.b + a.\bar{b}} = a.\bar{a} + a.b + \bar{a}.\bar{b} + b.\bar{b}$ , soit  $\overline{\bar{a}.b + a.\bar{b}} = a.b + \bar{a}.\bar{b}$ . Cette fonction prend la valeur 0 si une seule des variables vaut 0.

a	b	$F(a, b)$
0	0	1
0	1	0
1	1	1
1	0	0

#### Remarque :

Les fonctions les plus courantes (et donc celles à retenir impérativement) sont les fonctions **OUI**, **NON**, **AND**, **OR**, **NAND** et **XOR**.

#### 4.5 Propriétés des opérateurs de base de l'algèbre de BOOLE

Les propriétés énoncées dans le tableau récapitulatif ci-dessous découlent directement de ce qui a été démontré précédemment.

Associativité de l'opérateur ET	$a.(b.c) = (a.b).c = a.b.c$
Associativité de l'opérateur OU	$a + (b + c) = (a + b) + c = a + b + c$
Commutativité de l'opérateur ET	$a.b = b.a$
Commutativité de l'opérateur OU	$a + b = b + a$
Distributivité de l'opérateur ET par rapport l'opérateur OU	$a.(b + c) = (a.b) + (a.c) = a.b + a.c$
Distributivité de l'opérateur OU par rapport l'opérateur ET	$a + (b.c) = (a + b).(a + c)$
Élément neutre de l'opérateur ET	$a.1 = 1.a = a$
Élément neutre de l'opérateur OU	$a + 0 = 0 + a = a$
Antisymétrie de l'opérateur ET	$a.0 = 0.a = 0$
Antisymétrie de l'opérateur OU	$a + 1 = 1 + a = 1$
Identités remarquables	$a.a = a$ (indempotence)
	$a + a = a$
	$a.\bar{a} = 0$
	$a + \bar{a} = 1$
	$\bar{\bar{a}} = a$
	$a + a.b = a$
Identités remarquables	$a.(a + b) = a$
	$a.(\bar{a} + b) = b.(\bar{b} + a) = a.b$
	$a + \bar{a}.b = \bar{b}.a + b = a + b$
	$(a + b).(c + d) = a.c + a.d + b.c + b.d$
Théorèmes de De MORGAN	$\overline{a + b} = \bar{a}.\bar{b}$
	$\overline{a.b} = \bar{a} + \bar{b}$



## 5. EXPRESSIONS LOGIQUES

### 5.1 Définition

Une expression logique est une combinaison cohérente de variables booléennes au moyen des trois opérations ET, OU, NON et de leurs compléments en nombre fini.

### 5.2 Table de vérité

La table de vérité d'une fonction logique est une écriture systématique qui consiste à décrire toutes les combinaisons des variables et d'y associer les valeurs correspondantes de la fonction.

C'est ce que nous avons déjà fait en établissant les tableaux des fonctions d'une et deux variables (voir paragraphes 0 et 4.4).

### 5.3 Ecriture d'une fonction logique

Soit la table de vérité d'une fonction quelconque à une variable :

$a$	$F(a)$
0	$F(0)$
1	$F(1)$

Nous allons chercher à déterminer l'expression logique de la fonction  $F(a)$ .

#### 5.3.1 Première écriture

La fonction  $F(a)$  est égale à  $F(0)$  quand  $a$  vaut 0, ou à  $F(1)$  quand  $a$  vaut 1. Mais,  $F(0)$  peut être égale à 0 ou 1. Pour connaître cette valeur, il suffit d'effectuer un ET logique entre  $F(0)$  et 1. De même, la valeur de  $F(1)$  est obtenue par ET logique entre  $F(1)$  et 1.

Cette proposition peut finalement s'écrire :  $F(a) = \bar{a} \cdot F(0) + a \cdot F(1)$

Cette écriture constitue la **première forme canonique** d'une fonction logique appelée «**somme de produits**» et peut être étendue à une fonction de  $n$  variables.

Par exemple, pour une fonction de deux variables, on obtient :

$$F(a, b) = \bar{a} \cdot \bar{b} \cdot F(0,0) + \bar{a} \cdot b \cdot F(0,1) + a \cdot \bar{b} \cdot F(1,0) + a \cdot b \cdot F(1,1)$$

#### 5.3.2 Deuxième écriture

La fonction contraire de  $F(a)$  est égale au contraire de  $F(0)$  quand  $a$  vaut 0, ou au contraire de  $F(1)$  quand  $a$  vaut 1.

Cette proposition peut finalement s'écrire :  $\overline{F(a)} = \bar{a} \cdot \overline{F(0)} + a \cdot \overline{F(1)}$

Or  $F(a) = \overline{\overline{F(a)}}$  d'où :  $F(a) = \overline{\bar{a} \cdot \overline{F(0)} + a \cdot \overline{F(1)}}$ . En appliquant les théorèmes de De MORGAN, il vient :  
 $F(a) = \overline{\bar{a} \cdot \overline{F(0)}} \cdot \overline{a \cdot \overline{F(1)}} = (\bar{a} + \overline{F(0)}) \cdot (\bar{a} + \overline{F(1)})$ .

Donc :  $F(a) = (a + F(0)) \cdot (\bar{a} + F(1))$

Cette écriture constitue la **deuxième forme canonique** d'une fonction logique appelée «**produit de sommes**» et peut être étendue à une fonction de  $n$  variables.

Par exemple, pour une fonction de deux variables, on obtient :

$$F(a, b) = (a + b + F(0,0)) \cdot (a + \bar{b} + F(0,1)) \cdot (\bar{a} + b + F(1,0)) \cdot (\bar{a} + \bar{b} + F(1,1))$$

## 5.4 Règles d'utilisation

On pourra utiliser l'une ou l'autre de ces deux écritures pour établir l'équation logique d'une fonction. Nous allons voir au travers d'exemples simples les façons d'utiliser ces écritures.

### 5.4.1 Fonction OUI

La table de vérité d'une fonction OUI s'écrit :

$a$	$F(a)$
0	0
1	1

#### Première écriture :

Nous avons :  $F(a) = \bar{a} \cdot F(0) + a \cdot F(1)$ , donc  $F(a) = \bar{a} \cdot 0 + a \cdot 1 = 0 + a = a$

#### Deuxième écriture :

Nous avons :  $F(a) = (a + F(0)) \cdot (\bar{a} + F(1))$ , donc  $F(a) = (a + 0) \cdot (\bar{a} + 1) = a \cdot 1 = a$

### 5.4.2 Fonction NON

La table de vérité d'une fonction NON s'écrit :

$a$	$F(a)$
0	1
1	0

#### Première écriture :

Nous avons :  $F(a) = \bar{a} \cdot F(0) + a \cdot F(1)$ , donc  $F(a) = \bar{a} \cdot 1 + a \cdot 0 = \bar{a} + 0 = \bar{a}$

#### Deuxième écriture :

Nous avons :  $F(a) = (a + F(0)) \cdot (\bar{a} + F(1))$ , donc  $F(a) = (a + 1) \cdot (\bar{a} + 0) = 1 \cdot \bar{a} = \bar{a}$

### 5.4.3 Fonction OU

La table de vérité d'une fonction OU s'écrit :

$a$	$b$	$F(a, b)$
0	0	0
0	1	1
1	1	1
1	0	1

**Première écriture :**

Nous avons :  $F(a, b) = \bar{a}.\bar{b}.F(0,0) + \bar{a}.b.F(0,1) + a.\bar{b}.F(1,0) + a.b.F(1,1)$ .

Donc  $F(a, b) = \bar{a}.\bar{b}.0 + \bar{a}.b.1 + a.\bar{b}.1 + a.b.1 = \bar{a}.b + a.\bar{b} + a.b$

D'où :  $F(a, b) = \bar{a}.b + a.(\bar{b} + b) = \bar{a}.b + a = a + b$

**Deuxième écriture :**

Nous avons :  $F(a, b) = (a + b + F(0,0)).(a + \bar{b} + F(0,1)).(\bar{a} + b + F(1,0)).(\bar{a} + \bar{b} + F(1,1))$ .

Donc  $F(a, b) = (a + b + 0).(a + \bar{b} + 1).(\bar{a} + b + 1).(\bar{a} + \bar{b} + 1) = (a + b).1.1.1 = a + b$

La deuxième écriture permet d'obtenir le résultat rapidement alors que la première est plus fastidieuse.

**5.4.4 Fonction ET**

La table de vérité d'une fonction ET s'écrit :

a	b	F(a, b)
0	0	0
0	1	0
1	1	1
1	0	0

**Première écriture :**

Nous avons :  $F(a, b) = \bar{a}.\bar{b}.F(0,0) + \bar{a}.b.F(0,1) + a.\bar{b}.F(1,0) + a.b.F(1,1)$ .

Donc  $F(a, b) = \bar{a}.\bar{b}.0 + \bar{a}.b.0 + a.\bar{b}.0 + a.b.1 = a.b$

**Deuxième écriture :**

Nous avons :  $F(a, b) = (a + b + F(0,0)).(a + \bar{b} + F(0,1)).(\bar{a} + b + F(1,0)).(\bar{a} + \bar{b} + F(1,1))$ .

Donc  $F(a, b) = (a + b + 0).(a + \bar{b} + 0).(\bar{a} + b + 0).(\bar{a} + \bar{b} + 1) = (a + b).(a + \bar{b}).(\bar{a} + b).1$ .

Soit :

$$F(a, b) = (a.a.\bar{a}) + (a.a.b) + (a.\bar{b}.\bar{a}) + (a.\bar{b}.b) + (b.a.\bar{a}) + (b.a.b) + (b.\bar{b}.\bar{a}) + (b.\bar{b}.b)$$

D'où :  $F(a, b) = 0 + a.b + 0 + 0 + 0 + a.b + 0 + 0 = a.b$ .

La première écriture permet d'obtenir le résultat rapidement alors que la deuxième est plus fastidieuse.

**5.4.5 Conclusions**

Des résultats précédents, on peut tirer les conclusions suivantes :

- lorsque la **fonction a moins de 1 que de 0, la première écriture est préférable**. Dans ce cas, seules les combinaisons de variables correspondant à des valeurs de la fonction égales à 1 sont à prendre en compte. On dit alors qu'on effectue un «**développement par les 1**».

- lorsque la fonction a moins de 0 que de 1, la deuxième écriture est préférable. Dans ce cas, seules les combinaisons de variables correspondant à des valeurs de la fonction égales à 0 sont à prendre en compte. On dit alors qu'on effectue un «développement par les 0».

#### 5.4.6 Exemple

Soit la table de vérité ci-dessous :

a	b	c	$F(a, b, c)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

$$\begin{aligned}
 F(a, b, c) &= \\
 &\bar{a} \cdot \bar{b} \cdot c \\
 &+ \\
 &\bar{a} \cdot b \cdot c \\
 &+ \\
 &a \cdot \bar{b} \cdot c
 \end{aligned}$$

Le nombre de 1 étant inférieur au nombre de 0, un développement par les 1 est préférable. Il vient de manière évidente :

$$F(a, b, c) = \bar{a} \cdot \bar{b} \cdot c + \bar{a} \cdot b \cdot c + a \cdot \bar{b} \cdot c$$

## 6. REPRÉSENTATION SYMBOLIQUE DES FONCTIONS BOOLÉENNES

Dans un système combinatoire :

- les variables d'entrée sont des informations qui décrivent à un moment donné l'état de certaines parties (position d'un chariot, d'un outil,...). Ces variables d'entrée sont notées en minuscules et sont binaires.
- les variables de sortie permettent de décrire l'état dans lequel doivent se trouver les organes récepteurs.

Afin d'obtenir les états des variables de sortie en fonction de toutes les combinaisons possibles des variables d'entrée, on utilise une table de vérité. Elle permet d'obtenir l'expression logique :

$$S = f(a, b, c)$$

La relation entre les variables de sortie et celles d'entrée est obtenue au moyen des fonctions élémentaires décrites aux paragraphes 0 et 4.4. Ces fonctions sont représentées par des symboles graphiques. Le tableau ci-dessous résume les fonctions élémentaires avec leur symbole, leur table de vérité et un schéma électrique associé. La représentation graphique des symboles est donnée selon la norme I.E.C. (International Electrotechnical Commission). Dans le schéma électrique, les entrées sont représentées par des interrupteurs et les sorties par des ampoules.

## 6.1 Tableau récapitulatif des fonctions logiques

Fonction	Table de vérité	Symbole	Schéma électrique															
<b>OUI</b> $S = a$	<table border="1"> <tr><td>a</td><td>S</td></tr> <tr><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td></tr> </table>	a	S	0	0	1	1											
a	S																	
0	0																	
1	1																	
<b>NON</b> $S = \bar{a}$	<table border="1"> <tr><td>a</td><td>S</td></tr> <tr><td>0</td><td>1</td></tr> <tr><td>1</td><td>0</td></tr> </table>	a	S	0	1	1	0	 $a = 1, S = 0$  $a = 0, S = 1$										
a	S																	
0	1																	
1	0																	
<b>OU</b> $S = a + b$	<table border="1"> <tr><td>a</td><td>b</td><td>S</td></tr> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> </table>	a	b	S	0	0	0	0	1	1	1	1	1	1	0	1		
a	b	S																
0	0	0																
0	1	1																
1	1	1																
1	0	1																
<b>ET</b> $S = a \cdot b$	<table border="1"> <tr><td>a</td><td>b</td><td>S</td></tr> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> </table>	a	b	S	0	0	0	0	1	0	1	1	1	1	0	0		
a	b	S																
0	0	0																
0	1	0																
1	1	1																
1	0	0																
<b>NAND</b> $S = \overline{a \cdot b} = \bar{a} + \bar{b}$	<table border="1"> <tr><td>a</td><td>b</td><td>S</td></tr> <tr><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> </table>	a	b	S	0	0	1	0	1	1	1	1	0	1	0	1		
a	b	S																
0	0	1																
0	1	1																
1	1	0																
1	0	1																
<b>NOR</b> $S = \overline{a + b} = \bar{a} \cdot \bar{b}$	<table border="1"> <tr><td>a</td><td>b</td><td>S</td></tr> <tr><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> </table>	a	b	S	0	0	1	0	1	0	1	1	0	1	0	0		
a	b	S																
0	0	1																
0	1	0																
1	1	0																
1	0	0																
<b>OU exclusif</b> $S = a \oplus b$ $S = a \cdot \bar{b} + \bar{a} \cdot b$	<table border="1"> <tr><td>a</td><td>b</td><td>S</td></tr> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> </table>	a	b	S	0	0	0	0	1	1	1	1	0	1	0	1		
a	b	S																
0	0	0																
0	1	1																
1	1	0																
1	0	1																
<b>ET exclusif</b> $S = \overline{a \oplus b}$ $S = \bar{a} \cdot \bar{b} + a \cdot b$	<table border="1"> <tr><td>a</td><td>b</td><td>S</td></tr> <tr><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> </table>	a	b	S	0	0	1	0	1	0	1	1	1	1	0	0		
a	b	S																
0	0	1																
0	1	0																
1	1	1																
1	0	0																

**Remarque :**

Ces fonctions sont matérialisées par des composants appelés **cellules logiques**. Les solutions les plus compliquées peuvent être réalisées en assemblant ces cellules logiques. Les cellules NOR se prêtent bien au « câblage » d'une fonction « produit de sommes », alors que les cellules NAND se prêtent mieux au câblage d'une fonction « somme de produits ».

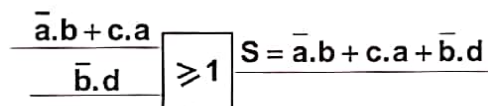
**6.2 Logigramme**

Un logigramme est une représentation graphique au moyen de leur symbole de cellules logiques décrivant une équation logique combinatoire. **Un logigramme n'est pas unique pour une équation donnée.**

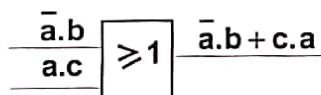
**Exemple :**

Soit la fonction logique :  $S = \bar{a}.b + c.a + \bar{b}.d$ . Le logigramme se trace de la sortie vers les variables d'entrée. Les différentes étapes *peuvent* être les suivantes :

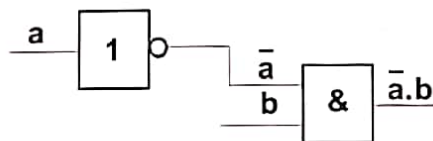
- une cellule **OU** donne  $S = \bar{a}.b + c.a + \bar{b}.d$  à partir de  $\bar{a}.b + c.a$  et de  $\bar{b}.d$ ,



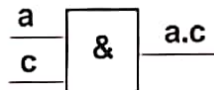
- une cellule **OU** donne  $\bar{a}.b + c.a$  à partir de  $\bar{a}.b$  et de  $c.a$ ,



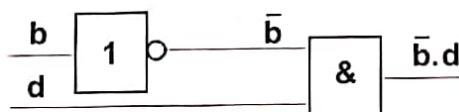
- une cellule **ET** donne  $\bar{a}.b$  à partir de  $\bar{a}$  et de  $b$ ,  $\bar{a}$  étant obtenu à partir de  $a$  par une cellule **NON**,



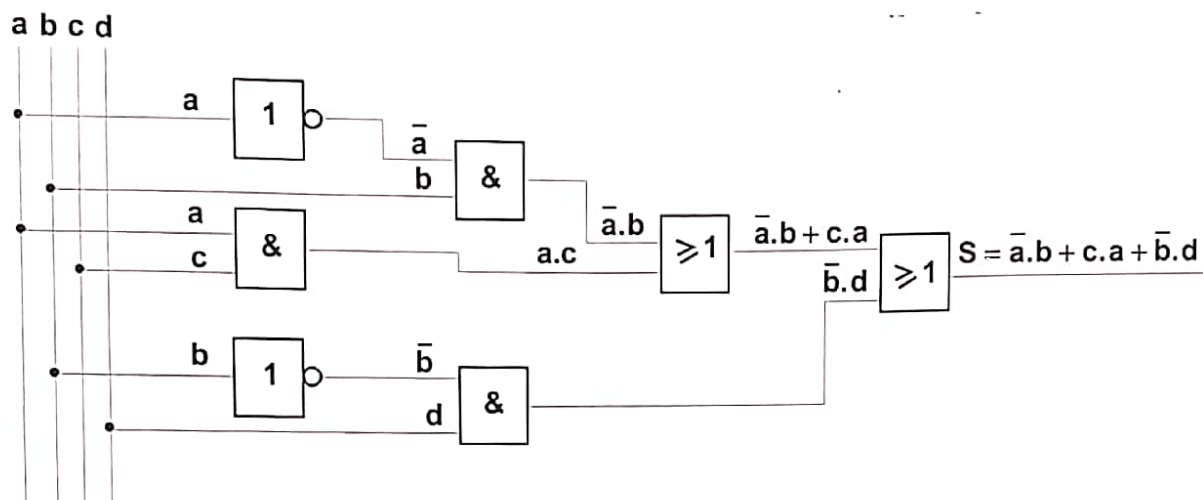
- une cellule **ET** donne  $c.a$  à partir de  $a$  et de  $c$ ,



- une cellule **ET** donne  $\bar{b}.d$  à partir de  $\bar{b}$  et de  $d$ ,  $\bar{b}$  étant obtenu à partir de  $b$  par une cellule **NON**,



Nous obtenons alors le logigramme ci-dessous :



## 7. SIMPLIFICATION DES EXPRESSIONS LOGIQUES

La simplification des expressions logiques est destinée à économiser le matériel nécessaire à la réalisation ou diminuer l'importance des équations programmées.

### 7.1 Méthode algébrique

Cette méthode de simplification repose sur l'application des propriétés énoncées précédemment. Elle demande beaucoup d'intuition.

#### 7.1.1 Premier exemple

Prenons l'expression du paragraphe 5.4.6 :  $F(a, b, c) = \bar{a}. \bar{b}. c + \bar{a}. b. c + a. \bar{b}. c$ .

En factorisant, on obtient :  $F(a, b, c) = (\bar{a}. \bar{b} + \bar{a}. b + a. \bar{b}). c$ . Or  $\bar{a}. \bar{b} + \bar{a}. b = \bar{a}. (\bar{b} + b) = \bar{a}$ .

Soit :  $F(a, b, c) = (\bar{a} + a. \bar{b}). c = (\bar{a} + \bar{b}). c$ .

Donc :  $F(a, b, c) = \bar{a}. \bar{b}. c + \bar{a}. b. c + a. \bar{b}. c = (\bar{a} + \bar{b}). c$

#### 7.1.2 Deuxième exemple

Soit l'expression :  $F(a, b, c) = a. b + \bar{a}. c + b. c$ .

En remarquant que  $a + \bar{a} = 1$ , nous pouvons écrire :  $F(a, b, c) = a. b + \bar{a}. c + b. c. (a + \bar{a}) = a. b + \bar{a}. c + b. c. a + b. c. \bar{a}$ .

En factorisant, nous obtenons :  $F(a, b, c) = a. b. (1 + c) + \bar{a}. c. (1 + b) = a. b. 1 + \bar{a}. c. 1 = a. b + \bar{a}. c$

Donc :  $F(a, b, c) = a. b + \bar{a}. c + b. c = a. b + \bar{a}. c$

## 7.2 Méthode graphique : Méthode de KARNAUGH

**Remarque :** cette méthode n'est plus au programme depuis la réforme de 2013, mais elle permet de simplifier très rapidement des expressions logiques complexes. Elle est donc quand même présentée dans ce polycopié.

C'est une technique graphique de simplification d'expressions logiques reprenant, sous une autre forme, la table de vérité.

### 7.2.1 Tableau de KARNAUGH

Un tableau de KARNAUGH comporte autant d'entrées qu'il y a de variables et associe une case à chaque combinaison de ces variables. Pour une fonction de  $n$  variables, un tableau de KARNAUGH comporte donc  $2^n$  cases.

Tableau à une variable :

	$a$
	$0 \quad 1$
$F(0)$	$F(1)$

Tableau à deux variables :

Ce tableau est issu du celui-ci-dessus

- par symétrie horizontale :

		$a,b$		
	$0,0$	$0,1$	$1,1$	$1,0$
$F(0,0)$	$F(0,1)$	$F(1,1)$	$F(1,0)$	

- ou par symétrie verticale :

		$a$	
		$0$	$1$
$0$	$b$	$F(0,0)$	$F(1,0)$
$1$	$b$	$F(0,1)$	$F(1,1)$

L'organisation des lignes et des colonnes respecte le code Gray : le passage d'une case à une case adjacente ne modifie donc que l'état d'une seule variable d'entrée. Cette particularité sera utilisée pour effectuer des simplifications.

Le processus de symétrie se poursuit pour trois puis  $n$  variables.

Tableau à trois variables :

- Un tableau à trois variables peut avoir l'allure suivante :

		$a,b$			
		$0,0$	$0,1$	$1,1$	$1,0$
$0$	$c$	$F(0,0,0)$	$F(0,1,0)$	$F(1,1,0)$	$F(1,0,0)$
$1$	$c$	$F(0,0,1)$	$F(0,1,1)$	$F(1,1,1)$	$F(1,0,1)$



- Le tableau de KARNAUGH peut aussi être présenté de manière légèrement différente :

		b		a	
		F(0,0,0)	F(0,1,0)	F(1,1,0)	F(1,0,0)
c		F(0,0,1)	F(0,1,1)	F(1,1,1)	F(1,0,1)

L'état des variables dans les différentes cases est indiqué par les barres verticales et horizontales situées à côté et au-dessus du tableau. Dans le tableau ci-dessus, par exemple, la ligne horizontale au-dessus de deux dernières colonnes indique que la variable  $a$  est à 1 dans ces colonnes. La ligne horizontale au-dessus des deux colonnes centrales indique que la variable  $b$  est à 1 dans ces deux colonnes et la ligne verticale à gauche de la dernière ligne indique que la variable  $c$  est à 1 dans cette ligne.

		b		a	
		$b = 1$	$a = 1$ $b = 1$	$a = 1$	
c		$b = 1$ $c = 1$	$a = 1$ $b = 1$ $c = 1$	$a = 1$ $c = 1$	

### 7.2.2 Simplification par tableau de KARNAUGH

Pour faire les simplifications, on procède à des **regroupements de cases adjacentes (de 0 ou de 1)**. On effectue des regroupements de  $2^n$  cases adjacentes (1, 2, 4, 8, 16, ... cases).

À l'intérieur de ces regroupements de 1 (ou 0), des variables apparaissant  $n$  fois sous leur forme normale et  $n$  fois sous leur forme complémentée peuvent être ignorées. Les variables qui restent fixes sont conservées.

On peut utiliser une même case pour plusieurs regroupements, mais on doit prendre au moins une fois tous les 1 du tableau. En pratique, on utilise cette méthode jusqu'à 4 ou 5 variables, pour plus de variables d'entrée, on réutilise l'algèbre de BOOLE ou d'autres méthodes qui ne seront pas abordées dans ce cours.

Ces regroupements doivent être les plus gros possibles et en nombre minimum afin d'obtenir l'expression la plus simple possible.

#### Lecture dans le cas de regroupement de 1 :

On déduit du tableau la fonction simplifiée en prenant tous les regroupements de 1 effectués. Pour chaque regroupement, on ne garde que les variables d'entrées en abscisse et en ordonnées qui restent fixes et on fait un ET logique entre chaque variable. Une variable à 0 est prise comme variable barre. On effectue ensuite un OU logique entre chaque regroupement.

#### Lecture dans le cas de regroupement de 0 :

Dans le cas de regroupements de 0, la technique est la même mais on effectue un OU logique entre les variables fixes dans les regroupements et un ET logique entre les différents regroupements. On obtient donc un produit de sommes au lieu d'une somme de produits. Attention, les calculs sont généralement plus fastidieux pour un produit de sommes.

### 7.2.3 Exemple

Considérons la fonction  $F(a, b, c) = \bar{a} \cdot \bar{b} \cdot c + \bar{a} \cdot b \cdot c + a \cdot \bar{b} \cdot c$  du paragraphe 5.4.6, décrite par sa table de vérité et établissons son tableau de KARNAUGH :

		a,b			
		0,0	0,1	1,1	1,0
c	0	0	0	0	0
	1	1	1	0	1

Le nombre de 1 étant inférieur au nombre de 0, un développement par les 1 est préférable. Il vient de manière évidente :

		a,b			
		0,0	0,1	1,1	1,0
c	0	0	0	0	0
	1	1	1	0	1

Un premier regroupement de 1 apparaît facilement. Dans ce regroupement, la variable  $b$  intervient sous forme de 1 et 0 et peut être ignorée.

Dans ce regroupement,  $a$  vaut toujours 0 et  $c$  vaut toujours 1. Le regroupement est donc équivalent à :  $\bar{a} \cdot c$ .

Un deuxième regroupement peut être effectué si on remarque que les colonnes extrêmes (tout comme les lignes extrêmes) sont adjacentes.

		a,b			
		0,0	0,1	1,1	1,0
c	0	0	0	0	0
	1	1	0	1	1

Ce regroupement est équivalent à :  $\bar{b} \cdot c$ .

Tous les 1 ayant été utilisés une fois, on peut effectuer la somme des regroupements. La fonction  $F(a, b, c)$  s'écrit donc :

$$F(a, b, c) = \bar{a} \cdot c + \bar{b} \cdot c = (\bar{a} + \bar{b}) \cdot c$$

Cette expression correspond bien à l'expression simplifiée obtenue en 7.1.1.

### 7.2.4 Remarques

Si on effectue un regroupement par les 0 mais que l'on écrit une somme de produit au lieu d'écrit un produit de somme, on obtient l'expression de  $\overline{F(a, b, c)}$ .

Pour effectuer les regroupements, il faut penser qu'un tableau de KARNAUGH se replie sur lui-même aussi bien verticalement qu'horizontalement.

### 7.2.5 Cas des fonctions incomplètes

Le cas des fonctions incomplètes (ou incomplètement spécifiées) peut apparaître comme un cas particulier. En pratique c'est le cas le plus fréquent. En effet :

- soit la valeur de la fonction n'a pas réellement d'importance pour le problème considéré,
- soit certaines combinaisons des variables sont physiquement impossibles.

*Pour illustrer ce dernier cas, considérons l'exemple d'un chariot qui se déplace entre deux contacts de fin de course (variables  $a$  et  $b$ ) associés aux points extrêmes du parcours. Si l'on doit traiter un problème combinatoire relatif au fonctionnement du chariot incluant éventuellement d'autres variables, il est clair que toutes les situations dans lesquelles les deux variables  $a$  et  $b$  vaudraient simultanément 1 sont physiquement impossibles (au moins en fonctionnement normal).*

*Pour les combinaisons correspondantes dans le tableau de KARNAUGH, la valeur de la fonction considérée n'est pas spécifiée puisque ces combinaisons ne se produisent jamais. Il est alors possible de lui donner une valeur quelconque 0 ou 1. Cette valeur sera notée  $\emptyset$  ou  $X$  dans les différentes cases concernées.*

*La démarche de simplification d'une telle fonction se déroule de la même manière que précédemment. Il faut choisir des regroupements de 2, 4, ... cases. Pour les simplifications, on peut utiliser les cases non spécifiées comme des 1 si cela facilite les regroupements, ou comme des 0 dans le cas contraire. Mais on ne peut attribuer qu'une seule valeur, à une case  $\emptyset$  donné.*

**Remarque :**

*Dans le cas d'une fonction incomplètement spécifiée, il est possible d'obtenir des expressions mathématiques différentes qui répondent au problème posé.*