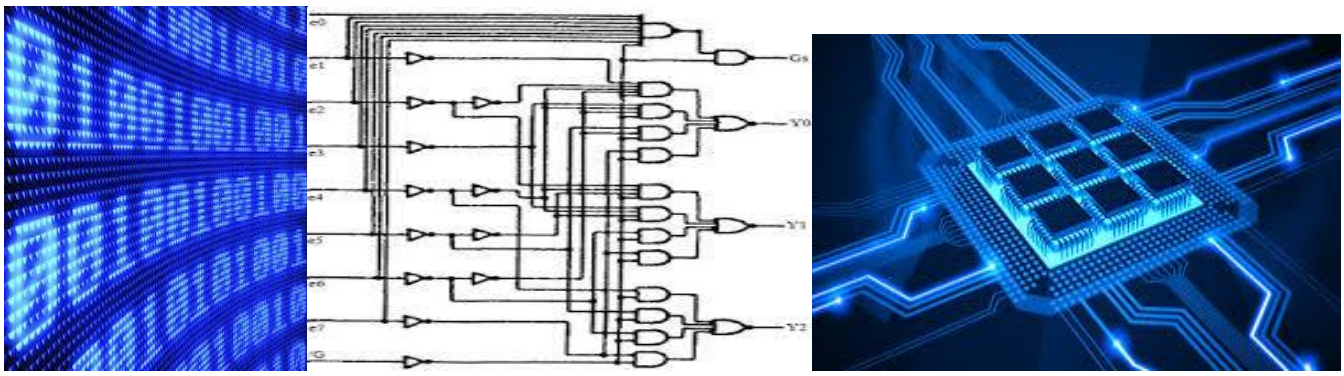
Université d'El-Oued
Faculté des sciences exactes
1ère MI. LMD

Module :

Codage et Représentation de l'Information



Support de cours

Chapitre 02

Algèbre de Boole

Et

Tableau de KARNAUGH

Préparé par : MEFTAH Mohammed Charaf Eddine

Algèbre de Boole

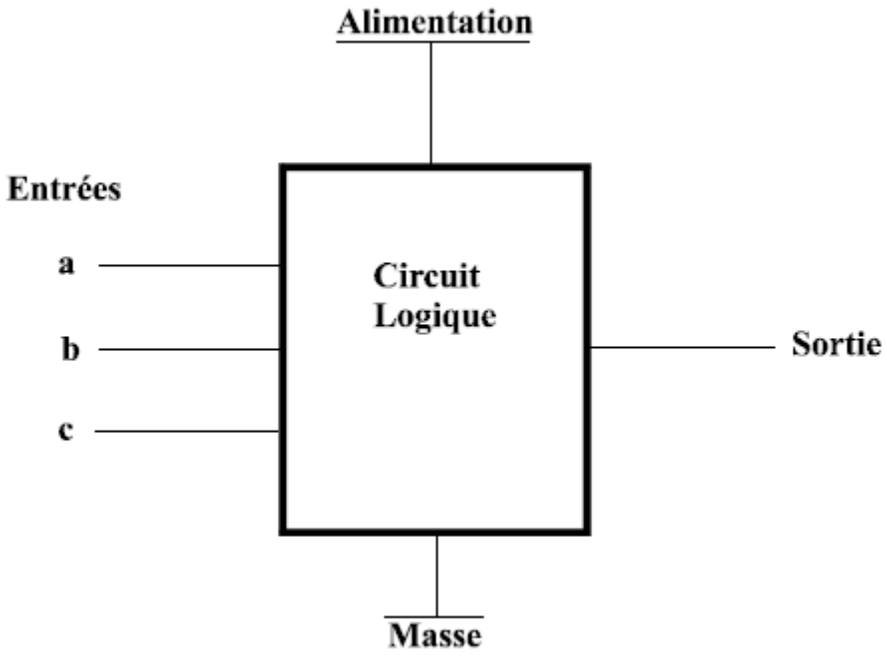
Introduction :

Pour exprimer et manipuler les nombres entiers, nous utilisons habituellement leur représentation en base 10 avec les dix chiffres 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
Matériellement, on peut coder facilement deux valeurs différentes par une tension électrique : une tension haute représentera l'une des deux valeurs et une tension basse, l'autre valeur. De ce fait, les ordinateurs utilisent principalement la base 2 et donc les deux chiffres 0, 1. Cette unité d'information qui peut valoir 0 ou 1 est appelée ***Bit***.

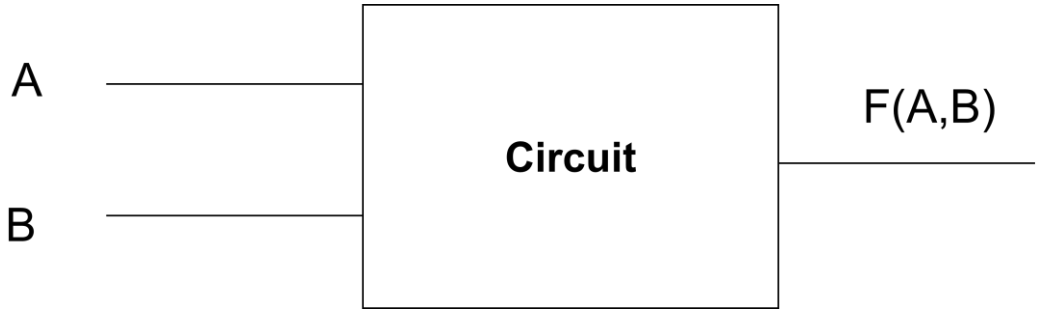
Remarque :
Une suite de 8 bits constitue un *octet* (en anglais : *byte*). C'est généralement la plus petite quantité d'information manipulable par un programmeur. Avec un octet on peut représenter 2^8, soit 256 valeurs différentes.
Vient ensuite la notion de *mot* (en anglais : *word*). Autrefois un mot était constitué de deux octets (16 bits) et on parlait d'ordinateur 16 bits. La taille du mot ayant évolué avec la technologie, on trouve maintenant couramment des mots de 32 bits (4 octets) et même 64 bits (8 octets).
Vu que la taille du mot n'est pas a priori bien fixée, toutes les quantités d'information seront exprimées à partir de l'octet qui est, quant à lui, de taille bien définie (8 bits).
Un *kilo-octet* (Ko) correspond à 2^{10} =1 024 octets.
Un mega-octet (Mo) correspond à 2^{10} = 1 048 576 octets ou encore 1 024 Ko.
Suite à la rapide évolution de la taille des mémoires, nous utilisons maintenant couramment :
le giga-octet (Go) qui correspond à 2^{30} octets, soit 2^{20} Ko, soit 2^{10} Mo. Est apparu récemment,
le tera-octet (To) qui correspond à 2^{40} octets, soit 2^{30} Ko, soit 2^{20} Mo, soit 2^{10}Go.

Un circuit digital est un circuit dans lequel les signaux ne peuvent avoir que deux niveaux, soit le niveau 1, soit le niveau 0. Un interrupteur, par exemple, est un circuit digital.

- **Les machines numériques (Exemple : un ordinateur…) sont constituées d'un ensemble de circuits électroniques.**



- Chaque circuit fournit une fonction logique bien déterminée (addition, comparaison ,….).



- La fonction F(A,B) peut être : la somme de A et B , ou le résultat de la comparaison de A et B ou une autre fonction

- ***Pour concevoir et réaliser ce circuit on doit avoir un modèle mathématique de la fonction réalisée par ce circuit . Ce modèle doit prendre en considération le système binaire.***

Le modèle mathématique utilisé est celui de **L' algèbre de Boole .**

1 . Définition

George Boole est un mathématicien anglais (1815-1864). Il a fait des travaux dont les quels les fonctions (expressions) sont constitués par **des variables qui peuvent prendre les valeurs 'OUI' ou 'NON'** .

Ces travaux ont été utilisés pour faire l'étude des systèmes qui possèdent deux états s'exclus mutuellement :

- Le système peut être uniquement dans deux états E1 et E2 tel que E1 est l'opposé de E2.
- Le système ne peut pas être dans l'état E1 et E2 en même temps
- Ces travaux sont bien adaptés au Système binaire (0 et 1).

Plusieurs valeurs logiques peuvent être combinées pour donner un résultat qui est lui aussi une valeur logique

Exemple de systèmes à deux états

- Un interrupteur est ouvert ou non ouvert (fermé)
- Une lampe est allumée ou non allumée (éteinte)
- Une porte est ouverte ou non ouverte (fermée)
- **Remarque :**

  On peut utiliser les conventions suivantes :

  OUI → VRAI (true)
  NON → FAUX (false)
  OUI → 1 (Niveau Haut)
  NON → 0 (Niveau Bas)

Deux notions fondamentales utilisées par cette algèbre :

1- ***Variable logique*** : est un variable que ne peut prendre que deux valeurs complémentaire 0 et 1 .(0 et 1 n'ont aucun caractère numérique) (Si X est une variable logique qui à pour valeur 1 alors sa négation X prend la valeur 0 et inversement 1)

2- ***Fonction logique*** : est une fonction binaire composé par n-variables logiques, de la forme F(X1,X2,…,Xn) (avec n finie) et son résultat est comme ses arguments peut –être 0 ou 1.

2. Définitions axiomatique de l'algèbre de Boole binaire

L'algèbre de Boole , comme tout autre système mathématique inductif peut être définie comme un **ensemble d'élément , un ensemble d'opérateurs et un nombre d'axiomes et de postulats** :

L'algèbre de Boole est une structure algébrique définie sur :

- un ensemble de deux éléments $\mathbf{B} = \{\mathbf{0}, \mathbf{1}\}$;
- Les opérateurs $+$ et $\times \; ou \; (.)$;
  Sont définis comme suite :

X	Y	X+Y	X.Y
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

L'algèbre de Boole est définie sur la base des postulats (فرضيات) suivants:

Les postulats de Huntington les plus communément utilisés pour formuler une structure algébrique sont :

1.Fermeture (عملية داخلية) : *Un ensemble S est fermé par rapport à un opérateur binaire * si :*
$\forall a,b \in S , \; a*b \in S$

Fermeture de l'ensemble $\mathbf{B}$ par rapport aux deux opérateurs $\cdot$ et $+$.

2. *Elément neutre (العنصر الحيادي) :* *Un ensemble S possède un élément neutre e par rapport à un opérateur binaire * si*
$\forall a \in S \; , a*e = e*a = a$

Existence d'un élément neutre noté $\mathbf{0}$ par rapport à l'opérateur $+$ et d'un élément neutre noté $\mathbf{1}$ par rapport à l'opérateur $\cdot$.

3. Commutativité (عملية تبديلية) : *Un opérateur binaire * est dit commutatif si :* $\forall a,b \in S , \; a*b = b*a.$

Les deux opérateurs $\cdot$ et $+$ sont ***commutatifs***

4 . Distributivité(عملية توزيعية) : *Un opérateur binaire * est dit distributif par rapport à un opérateur binaire + si*
$\forall a,b,c \in S, \; a * (b+c) = (a*b) + (a*c) ,$

L'opérateur $\cdot$ est distributif par rapport à l'opérateur $+$ et

L'opérateur $+$ est distributif par rapport à l'opérateur $\cdot$.

5 . $\forall x \in B$, $\exists x' \in B$ tel que **x+ x' = 1 et x.x' = 0** .

Ce postulat définit un nouvel opérateur unaire appelé ***complément (المتمم).***

6. Il existe au moins deux éléments distincts dans B.

3. Théorèmes et propriétés de l'algèbre de Boole

3.1 Principe de dualité (ازدواجية)

Dans une algèbre de Boole, tout résultat se présente sous deux **formes duales**.
Soit P un résultat, son dual (P) s'obtient en permutant systématiquement :

- les symboles « + » et « . »
- les symboles 0 et 1.

Si un résultat P est vrai dans une algèbre de Boole, il en est de même pour son dual.

Tous les axiomes (ou propriétés) sont toujours **vrais** si on remplace les $+$ par des $\cdot$ et inversement , et les 0 par des 1 et inversement : *Exemple* : l'expression duale de ***(y+1).x*** est ***(y.0)+x***.

Exemples :

- $\overline{\overline{a}} = a$ duale: $\overline{\overline{a}} = a$
- $a+1 = 1$ duale: $a.0 = 0$
- $a+a = a$ duale: $a.a = a$
- $a + a.b = a$ duale: $a.(a+b) = a$
- $a.b + \overline{a}.b = b$ duale: $(a+b).(\overline{a}+b) = b$
- $a.b + \overline{a} = b + \overline{a}$ duale: $(a+b).\overline{a} = b.\overline{a}$
- $a.b + \overline{a}.c + b.c = a.b + \overline{a}.c$

3.2 Théorèmes fondamentaux

	(a)	duale	(b)
Post 1	$x+0 = x$		$x.1=x$
Post 2 (commutativité)	$x+y = y+x$		$x.y = y.x$
Post 3 (distributivité)	$x(y+z) = xy + xz$		$x+yz = (x+y)(x+z)$
Post 4 (complément)	$x+x' = 1$		$x.x' = 0$
Théorème 1	$x+x = x$		$x.x = x$
Théorème 2	$x+1 = 1$		$x.0 = 0$
Théorème 3(involution)	$(x')' = x$		
Théorème 4(associativité)	$x+(y+z) = (x+y)+z$		$x.(y.z) = (x.y).z$
Théorème 5(de Morgan)	$(x+y)' = x'.y'$		$(x.y)' = x'+y'$
Théorème 6(absorption إمتصاص)	$x+x.y = x$		$x . (x+y) = x$

3.3 Précédence des opérateurs

La précédence des opérateurs établit des règles de priorité sur les opérateurs pour évaluer une expression booléenne . Sans ces règles , une expression peut apparaître ambiguë; son calcule peut se faire de plusieurs façons possibles et fournir plusieurs résultats différents par exemple , l'expression simple **(x.y+z)** peut être évaluée :

- Soit en effectuant l'opération • ensuite l'opération +,
- Soit en effectuant l'opération + ensuite l'opération •,

Pour lever l'ambiguïté, on définit une priorité sur les opérateurs . Pour l'algèbre de Boole binaire , la précédence des opérateurs donnée dans l'ordre décroissante est la suivante :

(1) Les parenthèses
(2) Le complément
(3) L'opérateur •
(4) L'opérateur +

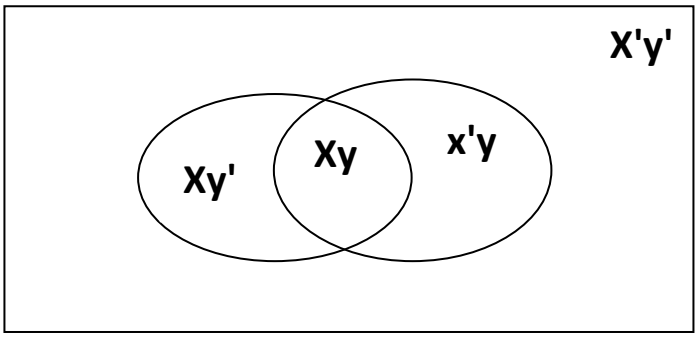
Pour calculer une expression booléenne, il faut effectuer les sous –expressions qui se trouvent entre parenthèses d'abord , ensuite le complément , en suite le produit booléen et enfin en dernier lieu l'addition booléenne.

3.4 Diagramme de Venn

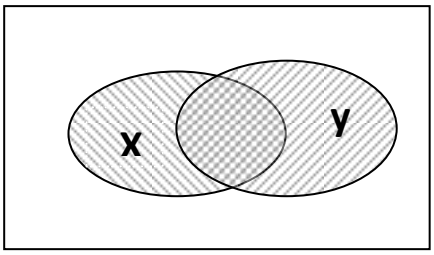
Pour visualiser la relation entre les variables d'une expression booléenne, on utilise le diagramme de Venn . Ce moyen ce consiste à considérer les classes de Boole comme des ensembles :

- La somme logique de deux classes se traduit ***par l'union*** (∪) entre les deux ensembles correspondants,
- Le produit logique ***par l'intersection (∩),***
- La complémententation par... ***le complément.***

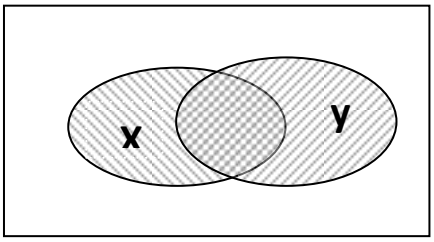
pour deux variables x et y , le diagramme de Venn est montré sur la figure suivante :



Remarquons qu'un diagramme de Venn n'est qu'une représentation schématique de la table vérité . Il permet aussi de valider certaines lois . Par exemple les diagrammes de Venn de la figure suivante illustrent la loi DeMorgan.



(x+y)'



x'.y'

4. Fonctions booléennes

Une fonction booléenne est une expression formée avec des **variables binaires** , **les opérateurs binaires OU (OR)** et **ET(AND)** , **l'opérateur unaire NOT** , **des parenthèses** et **le signe égale (=).** Pour des valeurs attribuées aux variables , **la fonction** vaut **soit 0 , soit 1:**

Exemple :

Soit **F1 = XYZ'**

Si X=1, Y=1 et Z= 0 alors F1=1

Une fonction booléenne peut être **représentée** par une **table de vérité** .

Une table de vérité est un tableau qui représente des entrées (en colonne) et des états binaire (0 / 1, faux / vrai, éteint / allumé, etc.). Une sortie, également représentée sous forme de colonne, est la résultante des états d'entrée, elle-même exprimée sous forme d'état binaire.

Entrées	Sortie
États	États

Considérons la fonction F1 et les fonctions :

F2=x+y'z

F3=x'y'z+x'yz+xy'

F4= xy'+x'z

Les tables de vérité correspondantes sont montrées dans la table suivante :

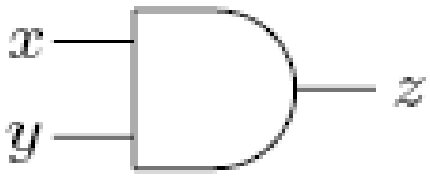
X	Y	Z	F1	F2	F3	F4
0	0	0	0	0	0	0
0	0	1	0	1	1	1
0	1	0	0	0	0	0
0	1	1	0	0	1	1
1	0	0	0	1	1	1
1	0	1	0	1	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	0

Tables de vérité des fonctions booléenne F1,F2,F3et F4

Le nombre de lignes dans une table de vérité est égale à 2^n, n étant le nombre de variables de la fonction .
F3 = F4 car ont les mêmes valeurs pour toutes les combinaisons des valeurs assignées aux variables x, y et z.

Porte : ET (AND)

Une porte-et peut avoir un nombre arbitraire d'entrées. Sa sortie vaut 1 si et seulement si toutes les entrées valent 1. Donc, la sortie vaut 0 si et seulement si au moins une des entrées vaut 0.
L'appellation «et » traduit le fait que la sortie vaut 1 si la première entrée vaut 1 et la deuxième entrée vaut 1 **et** . . . **et** la n-ième entrée vaut 1.
Dans les diagrammes représentant des circuits (portes et interconnexions entre les portes), la porte-et est dessinée de la manière suivante :



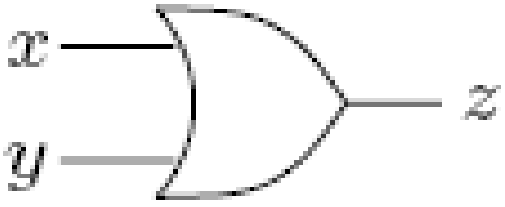
Voici la table de vérité d'une porte-et avec deux entrées :

x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

Porte OU (OR) :

Comme la porte-et, la porte-ou peut avoir un nombre arbitraire d'entrées. La sortie vaut 1 si et seulement si au moins une des entrées vaut 1. Autrement dit, la sortie vaut 0 si et seulement si toutes les entrées valent 0.
L'appellation « ou » vient du fait que la sortie vaut 1 si la première entrée vaut 1 ou la deuxième entrée vaut 1 **ou** . . . **ou** la n-ième entrée vaut 1.

Dans les diagrammes de circuits, la porte-ou est dessinée de la façon suivante :

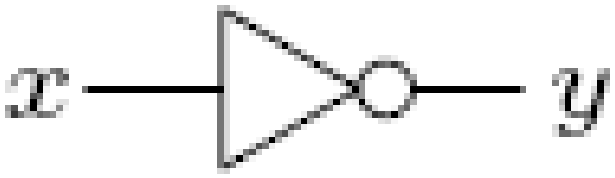


Voici la table de vérité d'une porte-ou avec deux entrées :

x	y	z
0	0	0
0	1	1
1	0	1
1	1	1

Inverseur NON (NOT) :

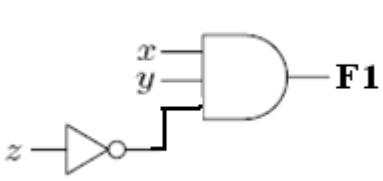
Un inverseur a exactement une entrée et une sortie. Sa sortie vaut 1 si et seulement si l'entrée vaut 0. Sinon la sortie vaut 0. Autrement dit, la valeur de la sortie est exactement l'inverse de la valeur de l'entrée.
Dans les diagrammes de circuits, l'inverseur est dessiné de la façon suivante :

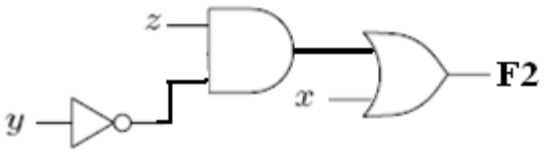


Voici la table de vérité de l'inverseur :

x	y
0	1
1	0

Une fonction booléenne peut être transformée de l'expression algébrique en un diagramme logique appelé **logigramme** composé des portes logiques ET (AND) , OU (OR) et NON (NOT)..
Les figures suivantes montrent l'implémentation des fonctions F1 et F2 :





F1 = xyz' **F2 = x+y'z**

Logigrammes des fonctions F1 et F2

Autres opérateurs binaires:

Les seuls opérateurs binaires logiques définis jusqu'à présent sont l'opérateur OR et L'opérateur AND. Nous allons prendre connaissance de tous les opérateurs binaires possibles et ceci en analysant chaque fonction booléenne à 2 variables obtenue par la table de vérité suivante :

x	y	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15
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	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Table . Recherche de tous les opérateurs binaires possibles.

F0 = 0 **Fonction nulle**
F1 = xy **ET (AND)**
F2 = xy' **inhibition** **x mais pas y**
F3 = x
F4 = x'y **inhibition** **y mais pas x**
F5 = y
F6 = x'y+xy' = x ⊕ y **XOR** **or exclusif**
F7 = x+y OR
F8 = (x+y)' = x'y' **NOR** **NOT OR**
F9 = x'y' + xy **Equivalence** **x égale à y**
F10 = y' **NOT**
F11 = x+y'
F12 = x' **NOT**
F13 = x'+y
F14 = (xy)' **NAND** **NOT AND**
F15 = 1 **Identité**

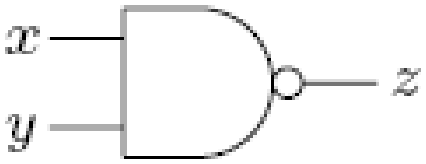
PORTES COMBINÉES

Souvent, il est pratique de combiner plusieurs fonctions de base (ET,OU, NON) dans une seule porte plus complexe, par exemple afin de conserver l'espace de dessin dans un diagramme de circuits. Dans cette section, nous présentons quelques portes combinées fréquemment utilisées avec leur table de vérité.

Porte NON-ET (NAND)

La porte-non-et est une porte-et avec un inverseur sur la sortie. Donc, au lieu de dessiner l'enchaînement de portes suivant,

on dessine une porte-*et* avec un cercle sur la sortie, comme ceci :

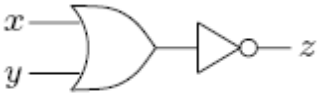


Comme la porte-*et*, la porte-*non-et* peut avoir un nombre arbitraire d'entrées. La table de vérité de la porte *non-et* est similaire à celle de la porte-*et* sauf que toutes les valeurs de la sortie sont inversées :

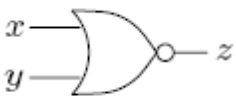
x	y	z
0	0	1
0	1	1
1	0	1
1	1	0

Porte NON-OR (NOR)

La porte-non-ou est une porte-ou avec un inverseur sur la sortie. Donc, au lieu de dessiner l'enchaînement de portes suivant,



on dessine une seule porte-*ou* avec un cercle sur la sortie comme ceci :

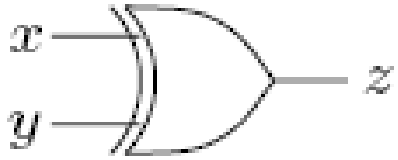


De même que la porte-*ou*, la porte-*non-ou* peut avoir un nombre arbitraire d'entrées. La table de vérité de la porte-*non-ou* est similaire à celle de la porte-*ou* sauf que la valeur de la sortie est inversée :

x	y	z
0	0	1
0	1	0
1	0	0
1	1	0

Porte OU-Exclusif (XOR)

La porte-*ou-exclusif* est apparentée à la porte-*ou*. Elle peut avoir un nombre arbitraire d'entrées. Sa sortie vaut 1 si et seulement si *exactement une des entrées* vaut 1. Sinon la sortie vaut 0. On dessine la porte-*ou-exclusif* comme ceci :



Voici la table de vérité d'une porte-*ou-exclusif* avec deux entrées :

x	y	z
0	0	0
0	1	1
1	0	1
1	1	0

5 – Manipulations algébriques

Un littéral (X,Y,….) est une variable booléenne complémentée ou non. Quand une fonction est implémentée par un circuit logique , chaque littéral dans la fonction représente une entrée à un circuit élémentaire et chaque terme (+ , . ,….) est implémenté à l'aide d'un circuit élémentaire.
La minimisation (تصغير أو تقليل) du nombre de littéraux et du nombre de termes donne un circuit avec peu d'équipements donc moins coûteux (أقل تكلفة) . Par conséquent , ***il est important de simplifier (تبسيط) une fonction booléenne avant de l'implémenter.***
La simplification des fonctions booléennes peut s'effectuer par ***manipulation algébrique*** faisant appel aux postulats et théorèmes évoqués précédemment.

Exemples de manipulation (simplification) algébriques :

1- x+x'y =(x+x')(x+y) = x+y
2- x(x'+y) = xx'+xy = xy
3- x'y'z + x'yz + xy' = x'z (y' + y) + xy' = x'z + xy'
4- xy + x'z + yz = xy + x'z + yz (x+x')
= xy + x'z + xyz + x'yz
= xy(1+z)+x'z(1+y)
= xy+x'z

Complément d'une fonction

Le complément d'un fonction booléenne F noté F' est obtenu en inversant les 0 en 1 et les 1 en 0 dans la table de vérité de F . Il peut être aussi obtenu par application du théorème de DeMorgane étendu à plusieurs variables .

Exemples : Considérer F1 = x'yz'+x'y'z et F2 = x(y'z'+yz)

F1' = (x'yz'+x'y'z)'
= (x'yz')' (x'y'z)'
= (x+y'+z) (x+y+z')
F2' = (x(y'z'+yz))'
= x'+ (y'z'+yz)'
= x'+ (y'z')'(yz)'
= x'+ (y+z) (y'+z')

Une procédure simple pour déterminer le complément d'une fonction consiste à prendre la forme duale de celle-ci et de complémenter chaque littéral de la forme duale obtenue.

Les formes canoniques :

Il existe deux formes canoniques pour chaque fonction booléenne : la forme ***disjonctive*** et la forme ***conjonctive.*** Ces deux formes sont basée respectivement sur la notion de ***minterme*** et de ***maxterme*** . Ces deux termes sont définis comme suit.

Minterme et Maxtermes :

Un minterme est défini comme étant un produit (جداء) booléenne de n littéraux . Il fait donc intervenir toutes les variables booléennes considérées. Le nombre total de mintermes est aussi égal à 2^n
Par dualité , **un maxterme est définit comme étant une somme (جمع) booléenne de n littéraux**. Le nombre total de maxtermes est aussi égal à 2^n.
La table suivante fournit tous les mintermes et les maxtermes ainsi leurs symboles respectifs pour 3 variables.

X	Y	Z	Minterme	Maxterme
0	0	0	M0 = x'y'z'	M0 = x+y+z
0	0	1	M1 = x'y'z	M1 = x+y+z'
0	1	0	M2 = x'yz'	M2 = x+y'+z
0	1	1	M3 = x'yz	M3 = x+y'+z'
1	0	0	M4 = xy'z'	M4 = x'+y+z
1	0	1	M5 = xy'z	M5 = x'+y+z'
1	1	0	M6 = xyz'	M6 = x'+y'+z
1	1	1	M7 = xyz	M7 = x'+y'+z'

Une fonction booléenne peut être exprimée algébriquement à partir de la table de vérité . Elle est égale à la somme des mintermes pour lesquels la fonction vaut 1 .

Les fonctions F1 et F2 de la table s'expriment comme :

X	Y	Z	F1	F2
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	0	0
1	0	0	0	1
1	0	1	0	1
1	1	0	1	1
1	1	1	0	1

F1 = xyz'
F2 = x'y'z+xy'z'+xy'z+xyz'+xyz
= M1+M4+M5+M6+M7
= $\sum$ (1,4,5,6,7)

Propriété 1 : Tout fonction booléenne peut s'exprimer comme **une somme de minterme**. C'est la **F**orme **C**anonique **D**isjonctive (**FCD** = expression des **1** de la fonction).
F2 (FCD) = x'y'z+xy'z'+xy'z+xyz'+xyz

Propriété 2 : Toute fonction booléenne peut s'exprimer comme **un produit de maxtermes** . C'est la **F**orme **C**anonique **C**onjonctive (**FCC** = expression des **0** de la fonction).
F2 (FCC) = (x+y+z)(x+y'+z)(x+y'+z')

Les deux formes canoniques (FCD (F=1), FCC(F=0)) sont équivalentes.

On choisit celle qui donne le résultat le plus simple :

Peu de 0 => FCC
Peu de 1 => FCD

Formes standard :
En parllèle des formes canoniques , il existe aussi plusieurs formes standard . Une forme standard disjonctive est tout simplement une somme de produits de littéraux quelconques qui ne sont pas nécessairement des mintermes. Et une forme standard conjonctive est un produit de sommes de littéraux qui ne sont pas un produit de sommes de littéraux qui ne sont pas nécessairement des maxtermes.

Exemples :
F1 = xy + xy'z' + yz
F2 = x (y' + z) (x'+ y + z')

Tableau de KARNAUGH

Dans ce cours sur la logique booléenne nous avons vu qu'il était possible de simplifier une équation en utilisant les relations de l'algèbre de Boole. Le tableau de KARNAUGH va nous permettre d'effectuer des simplifications par une méthode semi graphique parfois plus rapide que la méthode algébrique.

Maurice KARNAUGH, Mathématicien, professeur d'informatique à l'institut polytechnique de New York (de 1980 à 1999) est l'inventeur du diagramme de KARNAUGH en logique (1950) et le co-inventeur des premiers circuits logiques.

Construction du tableau de KARNAUGH

- C'est un tableau de 2^n cases, n étant le nombre de variables de la fonction logique.
- Sur les lignes et colonnes, on représente l'état des variables d'entrée codées en binaire réfléchi (code GRAY)
- Le tableau est construit pour une variable de Sortie
- Chaque case représente une combinaison des entrées.

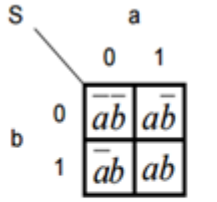


Tableau à 2 variables

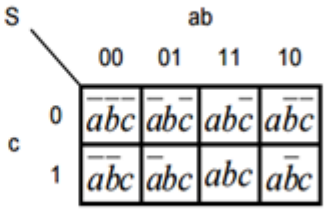


Tableau à 3 variables

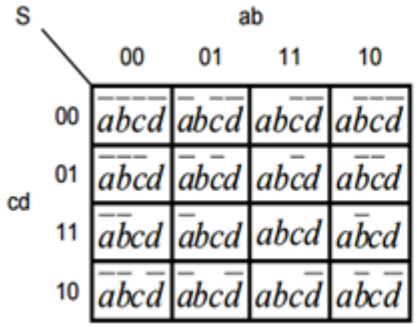


Tableau à 4 variables

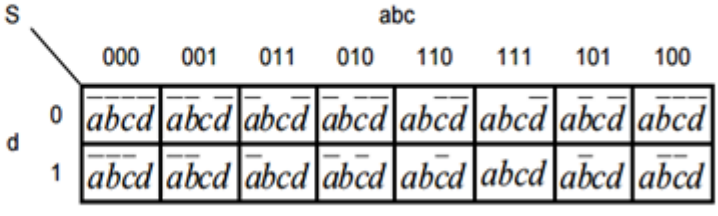
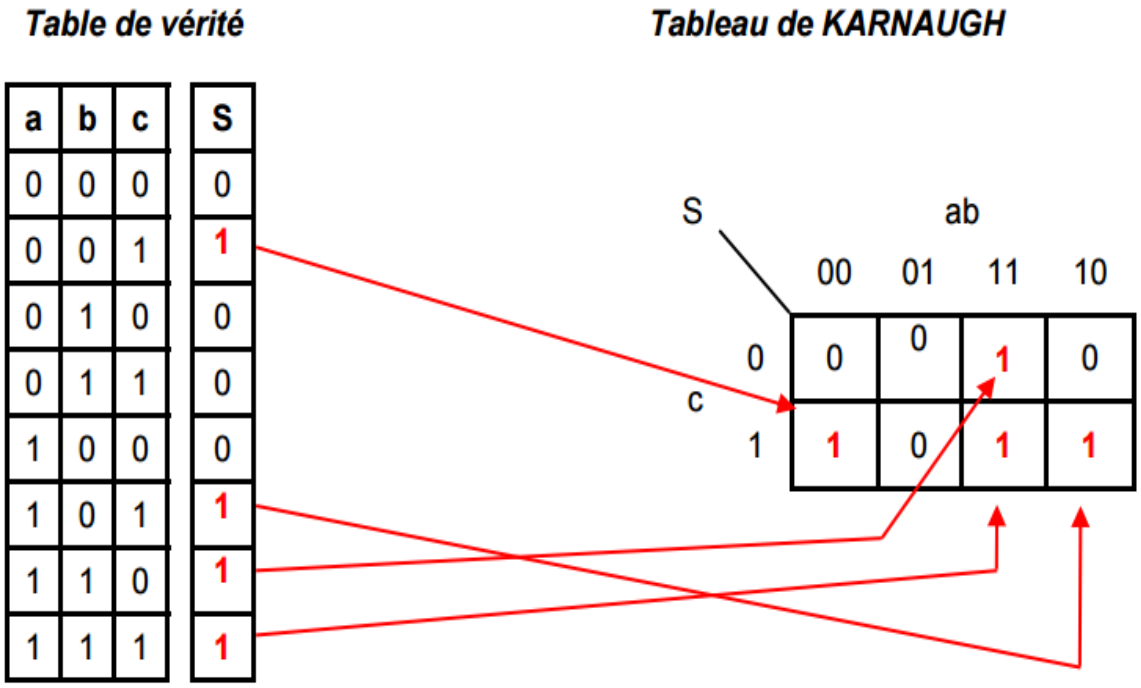


Tableau à 4 variables

Passage de la table de vérité au tableau de KARNAUGH

Une équation logique peut être représentée par une table de vérité ou un tableau de KARNAUGH.

Soit l'équation : S = ab + ab'c +b'c



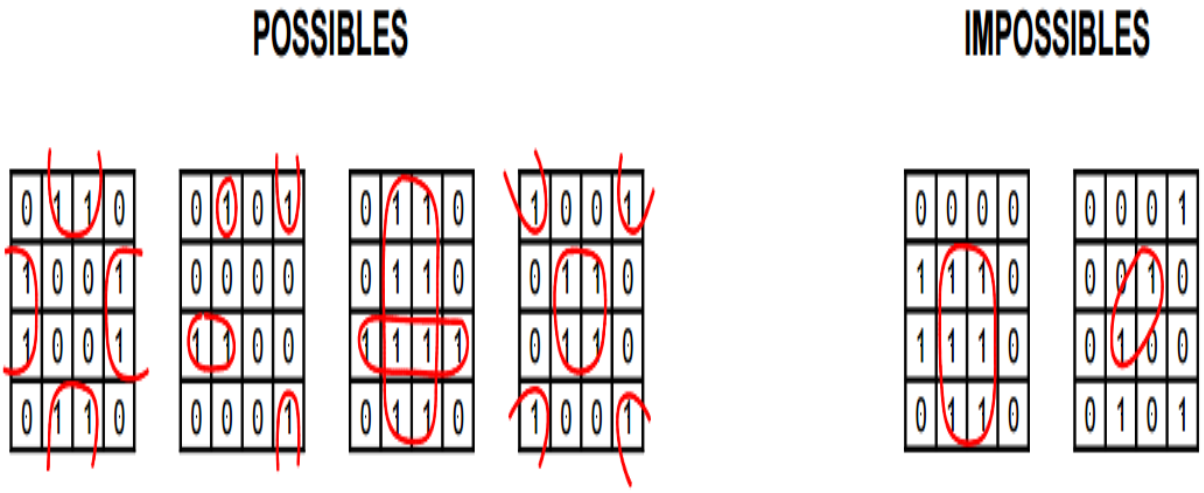
Simplification d'équation

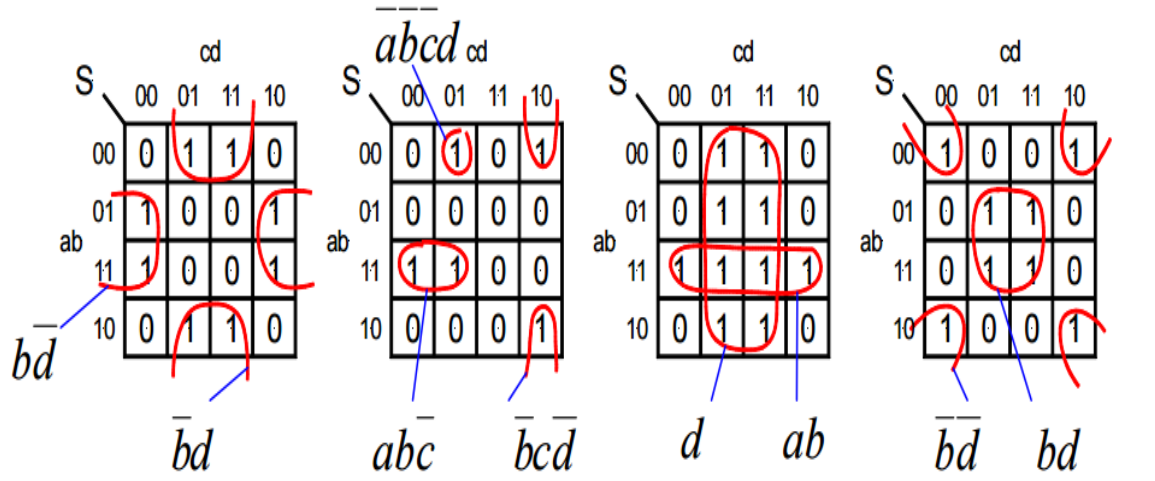
La méthode consiste à réaliser des groupements de CASES ADJACENTES contenant des 1 ou des 0. Un groupement de 1 permet d'obtenir l'équation de S , un groupement de 0 permet d'obtenir l'équation S'

Les règles de groupement :

- Le nombre de cases d'un groupement doit être égal à 1, 2, 4, …2^n
- Les groupements doivent être les plus grands possibles
- Les groupements peuvent se chevaucher pour être les plus grands possibles.
- Pour extraire l'équation de la fonction logique on ne retient que les variables dont l'état ne change pas à l'intérieur d'un groupement et on effectue la somme logique (OU logique) de toutes les expressions trouvées.

Exemples de groupements





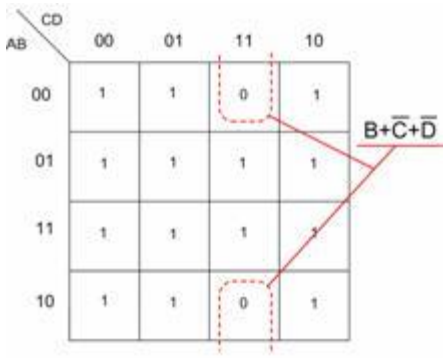
$S = b\bar{d} + \bar{b}\bar{d}$ $\quad S = ab\bar{c} + \bar{b}c\bar{d} + \bar{a}\bar{b}\bar{c}\bar{d}$ $\quad S = b\bar{d} + \bar{b}\bar{d}$ $\quad S = b\bar{d} + \bar{b}d$

- Un groupement de 1 case n'élimine aucune variable
- Un groupement de 2 cases élimine 1 variable
- Un groupement de 4 cases élimine 2 variables
- Un groupement de 8 cases élimine 3 variables
- …

Unir les *maxterms :*

Il est également possible d'unir les *maxterms* d'une table de Karnaugh plutôt que les *minterms* comme nous l'avons fait jusqu'ici. Pour ce faire, il suffit de procéder de manière inverse.

On cherchera à réunir les 0 plutôt que les 1 dans la table. Les variables dans les termes ainsi obtenus sont exprimés sous forme positive (non inversée) si elles ne prennent que la valeur 0, autrement, elles sont exprimées sous forme négative (inversée). La fonction simplifiée est exprimée sous forme d'un produit de sommes.



Dans notre exemple, il n'est possible de réunir qu'un seul rectangle. Ce dernier comprend la variable A sous ses deux formes (ligne 1 et ligne 4). Elle est éliminée du terme obtenu par l'union. Les autres variables se présentent sous une seule forme : B prend uniquement la valeur 0 et sera représentée sous forme positive dans le terme ; C prend uniquement la valeur 1 et sera représentée sous forme négative dans le terme ; D prend uniquement la valeur 1 et sera représentée sous forme négative dans le terme.

Karnaugh avec cas facultatifs (indéfinie) :

Il arrive parfois qu'une fonction logique ne soit pas définie pour l'ensemble des combinaisons possibles de ses variables d'entrée. Ces combinaisons, pour lesquelles les sorties sont dites facultatives, résultent souvent de la pratique où nombre de cas ne sont pas à considérer. Ce qu'on entend par le fait de dire que la sortie est facultative, c'est qu'elle peut prendre indifféremment la valeur 0 ou 1 sans affecter le comportement attendu du système.
Les techniques d'optimisation profitent alors de cette tolérance pour optimiser davantage le résultat. Grâce à leur qualité d'illustration graphique, les tables de Karnaugh permettent une exploitation élégante de ces conditions favorables.
Considérons l'exemple donné par la table de vérité suivante. La dernière ligne englobe l'ensemble des combinaisons non énumérée (représentés par deux tirets --) et correspondant aux minterms 10 à 15 de la fonction f. Les valeurs facultatives de f sont représentés par un seul tiret (-).

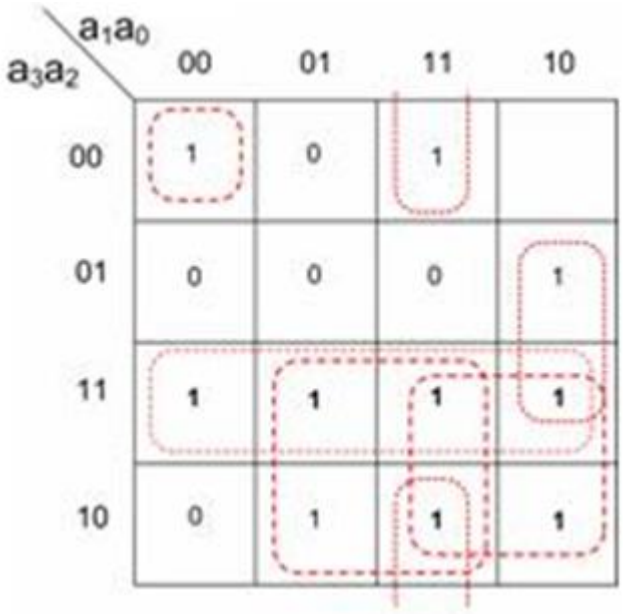
Minterm	a_3	a_2	a_1	a_0	*f*
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	1
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	1
10-15	--	--	--	--	-

L'équivalent de cette table de vérité en table de Karnaugh suit :

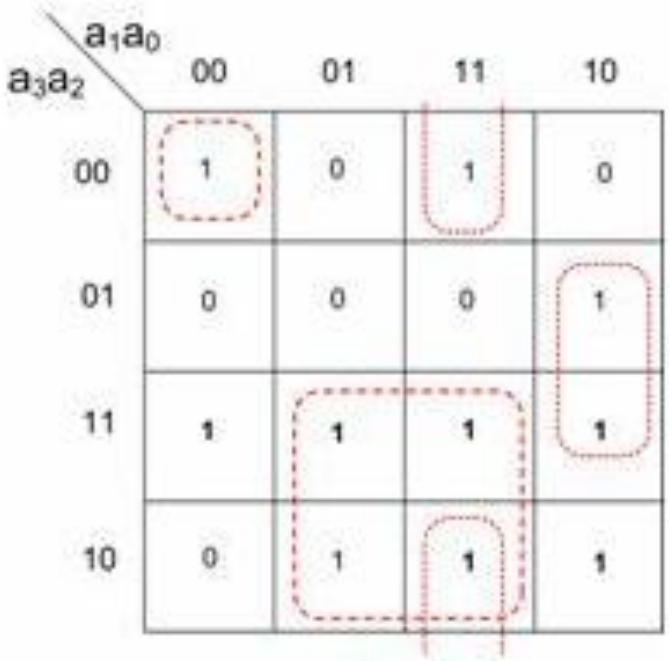
a_3a_2 \ a_1a_0	00	01	11	10
00	1	0	1	0
01	0	0	0	1
11	-	-	-	-
10	0	1	-	-

Afin de résoudre ce problème, il est conseillé d'agir comme si tous les cas facultatifs étaient des minterms (1) de la fonction à simplifier. Il suffit alors d'appliquer la méthode de simplification par table de Karnaugh .

On commence par recenser les impliquants premiers :



1- On élimine les impliquants premiers ne comportant que les cas facultatifs :



Technique de Quine-McCluskey

La technique de Quine-McCluskey est proche de celle de la table de Karnaugh mais recourt à un formalisme permettant de manipuler plus que 6 variables. Ainsi, la technique de Quine-McCluskey permet de résoudre des problèmes à 10 voire 15 variables indépendantes. La technique de Quine-McCluskey étant principalement une méthode algorithmique destinée à une exécution logicielle. Néanmoins, il importe aux étudiants de comprendre le fonctionnement de la technique et de pouvoir l'exécuter à la main pour faire le parallèle avec la méthode de Karnaugh.