Algèbre booléenne et Portes Logiques

Introduction

Point Histoire : *A partir de 1847, le Britannique* ***George Boole*** *propose un mode de calcul permettant de* ***traduire des raisonnements logiques*** *par des opérations algébriques.*

Il crée ainsi une branche des Mathématiques qui définit des opérations dans un ensemble qui ne contient que ***deux éléments*** *notés 0 ou 1, ou bien vrai ou faux...*

Cela a permis de mettre en place les ***fondements de l'informatique****.*

On utilise les booléens **en programmation**, notamment dans les comparaisons, les tests d'égalité.

Exemples :

```
>>> a = 2
>>> test = (a == 3)
>>> type(test)
<class 'bool'>
>>> test
False
```

```
0 def egal(n):
1     #l'instruction est testée et renvoi le booléen True ou False
2     if n >= 0:
3         #si le boléen est True
4         print("La valeur est positive ou nulle.")
5     else:
6         #si le booléen est False
7         print("La valeur est strictement negative.")
```

Opérateurs booléens

L'ensemble des booléens est associé à **trois opérations** booléennes permettant d'en combiner les valeurs.

Opérateurs

La conjonction	et (and)	c1 et c2 est réalisé lorsque les conditions c1 et c2 sont toutes les deux réalisées
La disjonction	ou (or)	c1 ou c2 est réalisé lorsqu'au moins une condition est réalisée parmi c1 et c2
La négation	non (not)	non c est réalisés lorsque la condition c n'est pas réalisée

Les notations sont nombreuses, en voici quelques unes :

a et b	$a \,\&\, b$	$a \wedge b$
a ou b	$a \mid b$	$a \vee b$
non a	$\sim a$	$\neg a$

Puisque l'algèbre de Boole ne contient que deux éléments, pour chacune de ces fonctions, on peut étudier **tous les cas possibles**. On les regroupe dans un tableau appelé **table de vérité**.

CC BY NC SA

Table de vérité de la conjonction :

Par exemple : La condition c1 n'est pas réalisée (0) et la condition c2 est réalisée (1), alors c1 et c2 n'est pas réalisée (0).

c1	c2	c1 et c2
0	0	0
0	1	0
1	0	0
1	1	1

Exercice 1 :

Compléter les tables de vérités de la disjonction et de la négation ci-dessous :

c1	c2	c1 ou c2
0	0	
0	1	
1	0	
1	1	

c	non c
0	
1	

Exercice 2 :

Écrire la comparaison $a < b < c$ à l'aide de deux comparaisons et d'un opérateur logique.

Remarque : En l'absence de parenthèses, lorsque l'on combine plusieurs opérateurs booléens, l'opérateur le plus prioritaire est la négation, puis vient la conjonction, et enfin l'opérateur le moins prioritaire la disjonction.

Exemple : L'expression a **or not** b **and** c doit être comprise a **or ((not** b**) and** c**)**.

Exercice 3 :

Écrire la table de vérité de l'expression a **or ((not** b**) and** c**)**.

(Procéder par étapes en tenant compte des priorités des opérateurs.)

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

Propriétés

a, b et c sont trois booléens :

- *Commutativité :* a et $b = b$ et a ; a ou $b = b$ ou a
- *Associativité :* a et (b et c) = (a et b) et c ; a ou (b ou c) = (a ou b) ou c
- *Distributivité :* a et (b ou c) = (a et b) ou (a et c) ; a ou (b et c) = (a ou b) et (a ou c)
- *Involution :* non (non a) = a
- *Lois de De Morgan :* non (a ou b) = (non a) et (non b) ; non (a et b) = (non a) ou (non b)

Exercice 4 :

Démontrer les lois de Morgan en utilisant des tables de vérité.

cc BY NC SA

Remarque : Lorsque Python évalue une expression booléenne, il le fait de façon **paresseuse**. C'est à dire que si la partie gauche d'un **OR** est vraie, il n'évalue pas la partie droite. De même si la partie gauche d'un **and** est fausse, la partie droite n'est pas évaluée.

Cela permet d'écrire les choses suivantes :

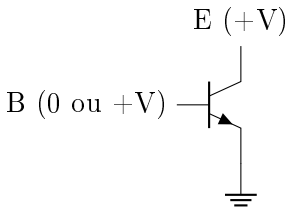
```
>>> x = 0
>>> x == 0 or 0 < 1/x < 1
True
>>> x != 0 and 0 < 1/x < 1
False
```

Si la division $1/x$ était évaluée, il y aurait une erreur, puisque l'on ne peut pas diviser par 0. Mais dans les deux cas, l'évaluation n'est pas faite puisque le résultat de l'expression a déjà pu être déterminée avec la partie gauche.

Portes logiques

Point Histoire : *En 1938, l'Américain Claude Shannon prouve que des circuits électriques peuvent résoudre tous les problèmes que l'algèbre de Boole peut résoudre.*

On va utiliser des **transistors** pour construire les opérateurs logiques de l'algèbre de Boole. Ils se comportent comme des **interrupteurs** qui laissent passer ou non un courant électrique, et ces deux états correspondent aux chiffres binaires 1 et 0.



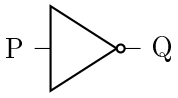
Avec plusieurs circuits en parallèle on peut ainsi représenter un nombre écrit en binaire.

Un simple transistor va permettre de réaliser une opération élémentaire appelée **porte logique NON (ou NOT)**.

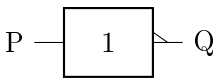
Elle n'a qu'un seul bit en entrée (P) et sa sortie (Q) vaut 0 si l'entrée vaut 1, et inversement elle vaut 1 quand son entrée vaut 0.

On la représente par le schéma suivant :

Notation américaine



Notation européenne



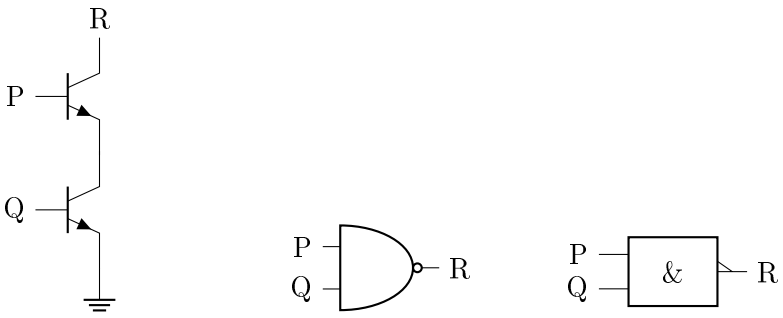
On représente le calcul réalisé par cette porte avec la *table de vérité* qui relie les valeurs entrées à la valeur du résultat.

Porte NOT	
P	Q
0	1
1	0

On peut fabriquer d'autres portes logiques en combinant plusieurs transistors.

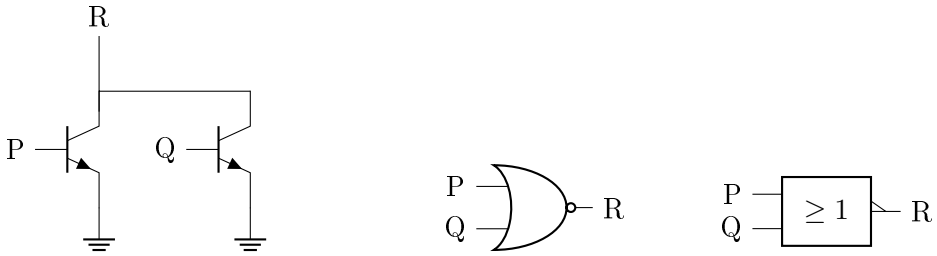
Par exemple, en combinant deux transistors en série on peut fabriquer la porte **NON ET (NAND)** qui, pour deux entrées P et Q, produit un résultat R.

Notation américaine Notation européenne



De la même manière en combinant deux transistors en parallèle, on obtient la porte **NON OU (NOR)** ci-dessous :

Notation américaine Notation européenne



Exercice 5 :

Compléter les tables de vérité suivantes :

Porte NAND		
P	Q	R
0	0	
0	1	
1	0	
1	1	

Porte NOR		
P	Q	R
0	0	
0	1	
1	0	
1	1	

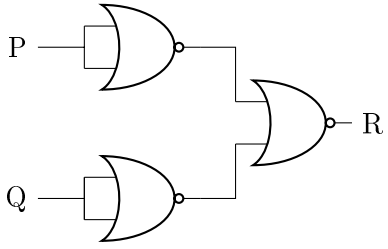
Les portes **NAND et NOR sont fondamentales** dans les circuits électroniques car elles sont **complètes**, c'est à dire que n'importe quel circuit peut être conçu en utilisant uniquement ces deux portes.

Exercice 6 :

1. La porte **NON (NOT)** peut être obtenue à partir d'un porte NAND en reliant les deux entrées de cette porte.

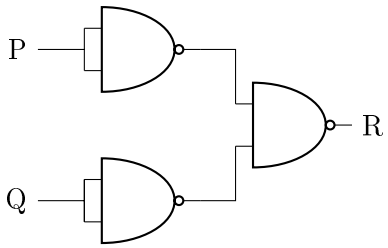
   Faire un schéma représentant la porte et dresser la table de vérité correspondante.

2. On donne le schéma ci-dessous constitué de trois portes NOR.



   Dresser la table de vérité associée ; de quelle porte logique s'agit-il ?

3. Cette fois, on utilise trois portes NAND.



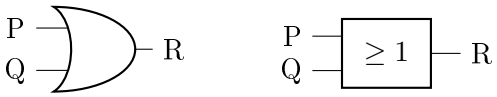
Dresser la table de vérité associée à ce schéma ; de quelle porte logique s'agit-il ?

Remarques :

• Voici les représentations des portes logiques **AND** et **OU** :

AND Notation américaine Notation européenne

OR Notation américaine Notation européenne



• Il existe aussi la porte logique **OU exclusif (XOR)** qui est un peu plus complexe à obtenir.

Elle correspond à au cas où l'on veut que l'une des deux conditions soit vraie, mais pas les deux en même temps.

Voici sa représentation et sa table de vérité :

XOR Notation américaine Notation européenne

Porte XOR		
P	Q	R
0	0	0
0	1	1
1	0	1
1	1	0

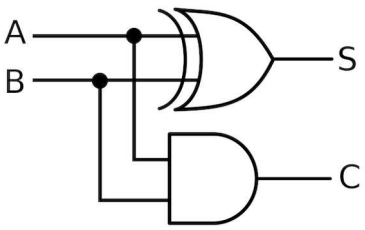
Circuits combinatoires

Lorsqu'on assemble des portes logiques entre elles on peut obtenir des **circuits combinatoires** : ce sont des circuits électroniques d'un peu plus haut niveau. Par exemple, des circuits pour comparer les entrées entre elles, ou bien faire des opérations arithmétiques comme l'addition, la soustraction....

Addition

Pour réaliser une **addition de deux bits, sans retenue**, il suffit d'utiliser une porte XOR.

Pour faire une addition, en indiquant s'il y a une retenue, il faut utiliser un **demi-additionneur** dont le circuit est représenté ci-dessous, ou S est la somme et C la retenue éventuelle :

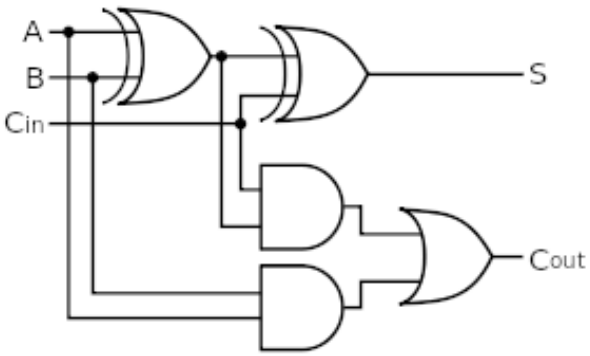


Demi-additionneur			
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Pour faire une addition complète, en tenant compte d'une éventuelle retenue en entrée, on utilise un **additionneur**, qui est décrit par le circuit ci-dessous. A et B sont les entrées correspondant aux deux bits à additionner, Cin est l'éventuelle retenue à ajouter à l'entrée, S est la sortie donnant la somme et Cout la retenue éventuelle de cette somme. (C pour Carry ou retenue en Anglais)

Exercice 7 :

Compléter la table de vérité de cet additionneur pour vérifier qu'il effectue bien l'addition de deux bits.



Additionneur				
A	B	Cin	S	Cout
0	0	0		
0	1	0		
1	0	0		
1	1	0		
0	0	1		
0	1	1		
1	0	1		
1	1	1		

Opérations bits à bits

En Python, l'expression a et b s'écrit a **and** b.

Il ne faut pas le confondre avec a & b, qui est un opérateur binaire agissant directement sur les nombres au niveau des bits.

Les autres opérateurs logiques bit-à-bit de Python sont $x \,|\, y$ et $x \wedge y$ pour respectivement le ou logique et le ou exclusif.

Ces opérateurs sont appelés **opérateurs bits-à-bits (bitwise)**.

Exemple : En Python 6 & 3 = 2.

On a $6 = 110_2$, $3 = 011_2$:

x	y	x & y
1	0	0
1	1	1
0	1	0

on a bien $010_2 = 2$

Dans le shell :

```
>>> 0b110 & 0b011
2
```

Remarque : Python n'affiche les bits d'un nombre binaire qu'à partir du premier bit non nul.

Exercice 8 :

Effectuer les calculs bits-à-bits ci-dessous :

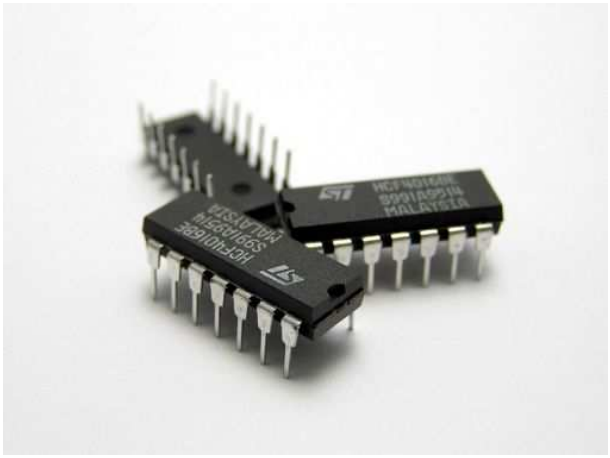
a = 12 & 5 ; b = 25 | 14 ; c = 9 $\wedge$ 13

Conclusion

Les circuits d'ordinateurs sont fabriqués à partir de portes logiques élémentaires, elles-mêmes construites à partir de transistors où les valeurs booléennes 0 et 1 sont matérialisées par des courants électriques.

Les portes logiques permettent d'obtenir des circuits combinatoires réalisant des fonctions booléennes qui permettent d'effectuer des additions binaires et, plus généralement, toute une gamme d'opérations arithmétiques et logiques.

Remarque : Dans nos ordinateurs, on ne trouve plus de transistor seuls, ils sont gravés dans du silicium pour constituer un circuit intégré (ou puce électronique).



CC BY NC SA