
Valeurs booléennes, opérateurs booléens et portes logiques

BARBIER J.M. - LGT Dumezil - NSI

septembre 2026

1 [B.4] Valeurs booléennes, opérateurs booléens et portes logiques

1.1 Les valeurs booléennes

1.1.1 Vrai ou faux

Un ordinateur ne manipule que des 0 et des 1 (il y a de l'électricité ou il n'y en a pas). Ces deux valeurs peuvent aussi être vues comme des "valeurs de vérité" :

- 0 = **faux** (en anglais *False*)
- 1 = **vrai** (en anglais *True*)

Ce type de donnée (qui ne peut prendre que ces deux valeurs) est une **valeur booléenne** (ou un **booléen**), du nom du mathématicien anglais **George Boole** (1815-1864). Un booléen tient donc, en théorie, sur **un seul bit** : c'est le plus petit type de donnée possible.

C'est la réponse naturelle à une **question fermée** : « l'interrupteur est-il allumé ? », « 3 est-il plus grand que 2 ? », « la porte est-elle ouverte ? »

1.1.2 Deux notations pour la même chose

Électronique / circuits	Mathématiques / tables	Python
tension haute (5V, ou 3.3V)	1	True
tension basse (0V)	0	False

Dans ce cours nous utiliserons surtout 0 et 1, plus lisibles dans les tableaux.

1.2 Opérateurs booléens et portes logiques

1.2.1 Opérateurs et portes

On peut **combinaison** des booléens avec des **opérateurs booléens** (ou opérateurs logiques) : **NON**, **ET**, **OU**, auxquels on ajoute le **OU exclusif**.

Exemple : quelle est la réponse à la question : "la porte est-elle fermée ET la lumière allumée ?"

Dans un ordinateur, chaque opérateur est réalisé physiquement par un composant électronique : une **porte logique**. Elle reçoit un ou deux signaux en entrée et produit un signal en sortie. Un processeur moderne en contient plusieurs **milliards**. Chaque porte a un **symbole** utilisé dans les schémas.

Dans un schéma, les entrées sont à gauche, la sortie à droite, et les traits qui les relient sont les **fils** : ils transportent une valeur, 0 ou 1. Un fil peut se **diviser** pour alimenter plusieurs portes, mais deux fils ne peuvent **jamais** arriver sur une même entrée.

Pour décrire complètement un opérateur (ou une porte), il suffit d'énumérer *tous* les cas possibles dans une **table de vérité**.



Important

Une **table de vérité** est un tableau qui donne la valeur de sortie pour **chacune** des combinaisons possibles des valeurs d'entrée.

Avec n entrées, il y a 2^n lignes : 2 lignes pour 1 entrée, 4 lignes pour 2 entrées, 8 lignes pour 3 entrées, 16 pour 4 entrées...

1.2.2 NON (NOT)

Le **NON** est **unaire** : il ne prend qu'une entrée, et l'inverse.

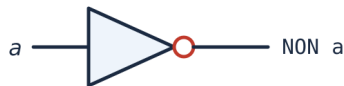


FIGURE 1 – Porte NON

a	NON a
0	1
1	0

Notations : NON a, not a (Python), $\bar{a}$ ou $\neg a$ (maths).



Important

Le **petit rond** en sortie d'une porte signifie toujours « et on inverse le résultat ».

1.2.3 ET (AND)

Le **ET** vaut 1 **si et seulement si** ses deux entrées valent 1 : « il faut le badge **et** le code ».



FIGURE 2 – Porte ET

a	b	a ET b
0	0	0
0	1	0
1	0	0
1	1	1

Notations : $a \text{ ET } b$, $a \text{ and } b$ (Python), $a \wedge b$ ou $a \cdot b$ (maths).



Astuce

Le ET se comporte comme une **multiplication** : $a \cdot b$ ne vaut 1 que si a et b valent 1.

1.2.4 OU (OR)

Le **OU** vaut 1 dès qu'**au moins une** des deux entrées vaut 1. Attention : c'est un OU **inclusif**, il vaut aussi 1 quand les deux valent 1.



FIGURE 3 – Porte OU

a	b	a OU b
0	0	0
0	1	1
1	0	1
1	1	1

Notations : $a \text{ OU } b$, $a \text{ or } b$ (Python), $a \vee b$ ou $a + b$ (maths).

1.2.5 OU exclusif (XOR)

Le **OU exclusif** vaut 1 quand les deux entrées sont **différentes**. C'est le « ou » du restaurant : fromage ou dessert, mais **pas les deux**.



FIGURE 4 – Porte XOR

Le symbole est celui du OU, avec une **barre supplémentaire** à l'arrière.

a	b	a XOR b
0	0	0
0	1	1
1	0	1
1	1	0

Notations : $a \text{ XOR } b$, $a \neq b$ (Python, il n'y a pas de mot-clé `xor`), $a \oplus b$ (maths).



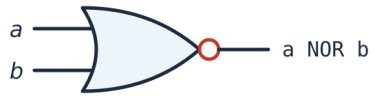
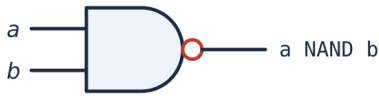
Astuce

Le XOR est un « détecteur de différence », mais aussi l'**addition binaire d'un bit** : $0 + 0 = 0$, $0 + 1 = 1$, $1 + 0 = 1$, $1 + 1 = 10$ dont on ne garde que le chiffre des unités, 0. Nous y reviendrons avec le demi-additionneur.

1.2.6 Opérateurs dérivés : NAND, NOR

On rencontre souvent les versions **niées** de ET et OU, plus simples à fabriquer avec des transistors. Leur symbole est celui du ET ou du OU, suivi du **petit rond** :

- $a \text{ NAND } b = \text{NON } (a \text{ ET } b)$
- $a \text{ NOR } b = \text{NON } (a \text{ OU } b)$



a	b	a NAND b	a NOR b
0	0	1	1
0	1	1	0
1	0	0	0
1	1	0	0

Culture



La porte NAND est *universelle* : avec uniquement des portes NAND, on peut reconstruire NON, ET, OU, XOR... et donc **n'importe quel circuit logique**. Idem pour NOR.

1.2.7 Récapitulatif des symboles



NON (NOT)
not a



ET (AND)
a and b



OU (OR)
a or b



OU exclusif (XOR)
 $a \neq b$



NON-ET (NAND)
not (a and b)



NON-OU (NOR)
not (a or b)

FIGURE 5 – Symboles des portes logiques

1.2.8 Récapitulatif des tables de vérité

Toutes les tables de vérité du cours réunies. Les deux premières colonnes sont les entrées, chacune des suivantes est un opérateur :

a	b	NON a	a ET b	a OU b	a XOR b	a NAND b	a NOR b
0	0	1	0	0	0	1	1
0	1	1	0	1	1	1	0
1	0	0	0	1	1	1	0
1	1	0	1	1	0	0	0

Quelques lectures utiles :

- la colonne NON a ne dépend **que** de a : l'opérateur est unaire, les deux premières lignes donnent le même résultat, et les deux dernières aussi,
- a NAND b est exactement la colonne a ET b **inversée**, et a NOR b la colonne a OU b inversée : c'est la définition de ces deux portes,
- a OU b et a XOR b ne diffèrent que sur la **dernière** ligne, celle où les deux entrées valent 1 : c'est toute la différence entre le OU inclusif et le OU exclusif,
- a ET b ne vaut 1 qu'une fois sur quatre, a OU b trois fois sur quatre.



Méthode

Ce tableau est un **aide-mémoire**, pas une chose à apprendre par cœur : les six colonnes se retrouvent en revenant à la définition de chaque opérateur (« le ET, c'est les deux », « le OU, c'est au moins un », etc.).

1.2.9 Simulation

Nous utiliserons un simulateur de circuits logiques (depuis **Capytale**) : on y place des entrées, des portes, une sortie, puis on bascule les entrées pour observer la sortie.



Méthode

Le simulateur permet de **vérifier** une table de vérité, pas de la remplacer : en devoir, il faut savoir la construire à la main.

1.3 Combinaison de portes logiques

1.3.1 Principe

Dès qu'on assemble plusieurs portes, la sortie de l'une devient l'entrée d'une autre. Pour comprendre ce que fait le circuit, on établit sa table de vérité.



Méthode

Pour dresser la table de vérité d'un circuit :

1. **compter les entrées** (n) et dessiner un tableau de 2^n lignes,
2. écrire **toutes** les combinaisons d'entrées, dans l'ordre, en comptant en binaire de 0 à $2^n - 1$ (c'est le plus sûr moyen de n'en oublier aucune),
3. ajouter **une colonne intermédiaire par porte**, dans l'ordre où le signal les traverse (de la gauche vers la droite du schéma),
4. remplir chaque colonne ligne par ligne, en n'utilisant que les colonnes déjà remplies,
5. la ou les dernières colonnes sont la ou les sorties

Les colonnes intermédiaires sont le point essentiel : on ne cherche **jamais** à deviner la sortie directement, on avance porte par porte.

1.3.2 Exemple détaillé

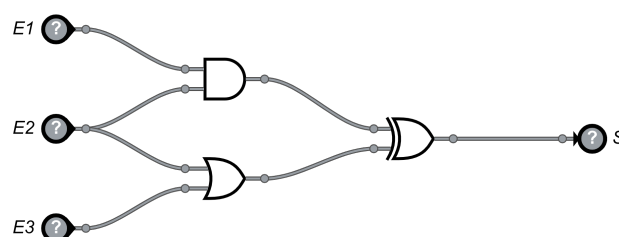


FIGURE 6 – Circuit 1 du cours

Ce circuit a 3 entrées et 3 portes :

- une porte ET qui reçoit E1 et E2, de sortie A,
- une porte OU qui reçoit E2 et E3, de sortie B,

- une porte XOR qui reçoit A et B, et donne la sortie S.

E2 alimente **deux** portes : le fil se divise. Avec 3 entrées, la table a $2^3 = 8$ lignes :

E1	E2	E3	A = E1 ET E2	B = E2 OU E3	S = A XOR B
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	1	1
0	1	1	0	1	1
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	1	1	0
1	1	1	1	1	0

La sortie vaut donc 1 dans 5 cas sur 8.



Dressez, sur le même modèle (avec les colonnes intermédiaires), la table de vérité des circuits 2 et 3 de la feuille Exos portes logiques.

1.4 Expressions booléennes

1.4.1 Écrire un circuit sous forme d'expression

Plutôt qu'un schéma, on peut écrire une **expression booléenne** : une formule qui combine des variables booléennes avec NON, ET, OU et XOR. Le circuit 1 précédent s'écrit :

$$S = (E_1 \text{ ET } E_2) \text{ XOR } (E_2 \text{ OU } E_3)$$

Schéma et expression sont deux écritures du **même** objet ; il faut savoir passer de l'une à l'autre :

- **du schéma vers l'expression** : on part de la sortie et on remonte, en remplaçant chaque porte par son opérateur et ses entrées,
- **de l'expression vers le schéma** : on place une porte par opérateur, en commençant par les parenthèses les plus internes.



Feuille Exos expressions booléennes :

- **exercice 1** : écrire l'expression de trois circuits (du schéma vers l'expression),
- **exercice 2** : dessiner le schéma de trois expressions (de l'expression vers le schéma).

1.4.2 Règles de priorité

Comme en mathématiques ($2 + 3 \times 4 = 14$ et non 20), les opérateurs booléens ont des **priorités** :



Important

Par ordre de priorité **décroissante** :

1. les **parenthèses**,
2. NON,
3. ET,
4. OU (et XOR).

À priorité égale, on évalue de **gauche à droite**.

Pour retenir : ET se comporte comme la multiplication, OU comme l'addition — et la multiplication est prioritaire.



- $a \text{ OU } b \text{ ET } c$ se lit $a \text{ OU } (b \text{ ET } c)$
- $\text{NON } a \text{ ET } b$ se lit $(\text{NON } a) \text{ ET } b$ — et **pas** $\text{NON } (a \text{ ET } b)$, qui est un NAND!

La table de vérité montre la différence :

a	b	(NON a) ET b	NON (a ET b)
0	0	0	1
0	1	1	1
1	0	0	1
1	1	0	0

Les deux colonnes sont différentes : **les parenthèses comptent**.



Conseil

En cas de doute, **mettez des parenthèses** : c'est toujours correct, et plus lisible.

1.4.3 Dresser la table d'une expression

Même méthode que pour un circuit : une colonne intermédiaire par opérateur, en respectant les priorités.

Exemple : table de vérité de $S = (a \text{ ET } \bar{b}) \text{ OU } (\bar{a} \text{ ET } b)$:

a	b	$\bar{a}$	$\bar{b}$	$a \text{ ET } \bar{b}$	$\bar{a} \text{ ET } b$	S
0	0	1	1	0	0	0
0	1	1	0	0	1	1
1	0	0	1	1	0	1
1	1	0	0	0	0	0

La colonne S est celle du **XOR** : nous venons de reconstruire un OU exclusif avec des NON, des ET et un OU.



Important

Deux expressions sont **équivalentes** si elles ont la **même table de vérité**. C'est le seul critère, peu importe leur écriture ou leur nombre de portes.



Exercice 3 de la feuille Exos expressions booléennes : dresser la table de vérité de quatre expressions, avec les colonnes intermédiaires.

1.4.4 Quelques équivalences utiles (pour info)



Non exigible

Ce qui est au programme, c'est de **dresser la table de vérité d'une expression booléenne**, et de passer d'un schéma à une expression.

Le calcul sur les expressions booléennes — les règles ci-dessous, la simplification d'une expression, les lois de De Morgan — n'est **pas exigible** et ne sera pas demandé en devoir. C'est donné ici pour information, et parce que c'est joli.

Elles permettent de **simplifier** une expression, donc de fabriquer le circuit avec moins de portes :

Règle	Écriture
élément neutre	$a \text{ ET } 1 = a; a \text{ OU } 0 = a$
élément absorbant	$a \text{ ET } 0 = 0; a \text{ OU } 1 = 1$
idempotence	$a \text{ ET } a = a; a \text{ OU } a = a$
complément	$a \text{ ET } (\text{NON } a) = 0; a \text{ OU } (\text{NON } a) = 1$
double négation	$\text{NON } (\text{NON } a) = a$
lois de De Morgan	$\text{NON } (a \text{ ET } b) = (\text{NON } a) \text{ OU } (\text{NON } b)$ $\text{NON } (a \text{ OU } b) = (\text{NON } a) \text{ ET } (\text{NON } b)$

1.5 Application : le demi-additionneur

1.5.1 Le problème

Comment faire faire une addition binaire à un circuit? Commençons par le plus simple : additionner **deux bits** E1 et E2. Reprenons la table d'addition binaire vue au chapitre B.1 :

- $0 + 0 = 0$
- $0 + 1 = 1$
- $1 + 0 = 1$
- $1 + 1 = 10$

Le dernier cas est la clé : le résultat ne tient pas sur un bit. Il faut donc **deux sorties** :

- la **somme** S (le chiffre des unités du résultat),
- la **retenue** R (le chiffre suivant, à reporter sur la colonne de gauche).

1.5.2 La table de vérité

E1	E2	Résultat de l'addition	Retenue R	Somme S
0	0	00	0	0

E1	E2	Résultat de l'addition	Retenue R	Somme S
0	1	01	0	1
1	0	01	0	1
1	1	10	1	0

Regardons chaque colonne de sortie séparément :

- R vaut 1 uniquement quand E1 et E2 valent 1 : c'est la table du **ET**,
- S vaut 1 quand E1 et E2 diffèrent : c'est la table du **XOR**.



Important

$$R = E_1 \text{ ET } E_2 \quad S = E_1 \text{ XOR } E_2$$

Deux portes suffisent donc à additionner deux bits. Ce circuit s'appelle le **demi-additionneur** (*half adder*).

1.5.3 Le schéma

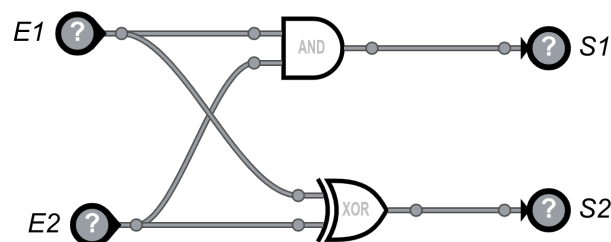


FIGURE 7 – Demi-additionneur

E1 et E2 alimentent chacune les deux portes : les fils se divisent.



Construisez ce circuit dans le simulateur, et vérifiez les 4 lignes de sa table de vérité en basculant les entrées.

1.5.4 Pourquoi « demi » ?

Parce qu'il ne fait que la moitié du travail : quand on additionne deux nombres de plusieurs bits, chaque colonne doit additionner **trois** bits, les deux bits du rang **plus la retenue de la colonne précédente**.

```

1 1 1 1      <- retenues
 1 0 1 1
+ 0 1 1 1
-----
1 0 0 1 0

```

Le circuit qui additionne trois bits (deux bits + une retenue entrante) s'appelle l'**additionneur complet** (*full adder*), et se fabrique... avec deux demi-additionneurs et une porte OU.

1.5.5 L'additionneur complet

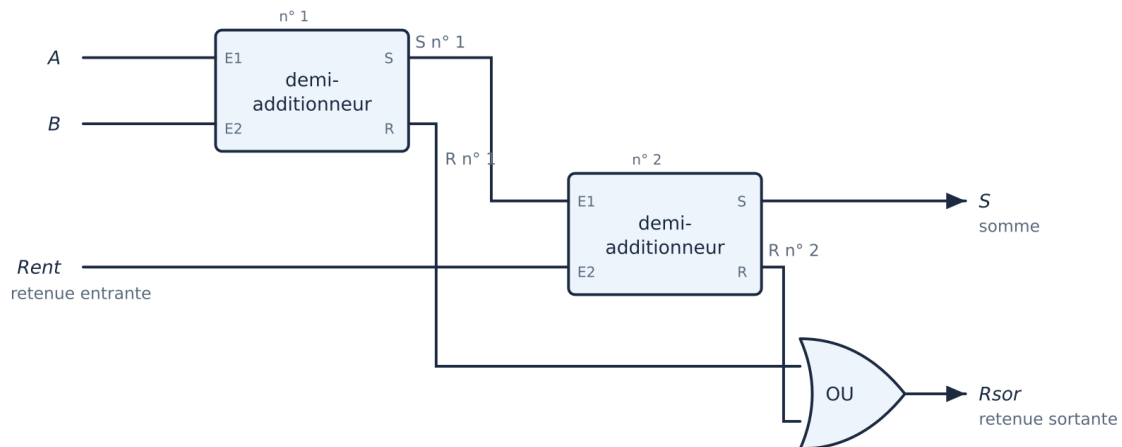


FIGURE 8 – Additionneur complet : deux demi-additionneurs et une porte OU

Il additionne **trois** bits : A, B, et la retenue entrante Rent.

- le demi-additionneur n° 1 additionne A et B,
- le demi-additionneur n° 2 ajoute la retenue entrante au résultat du premier : sa somme est la **somme finale S**,
- la porte OU rassemble les deux retenues : il suffit que **l'un** des deux demi-additionneurs en produise une.



Pourquoi un simple OU suffit

Les deux retenues ne peuvent jamais valoir 1 **en même temps** : si A et B valent tous les deux 1, le premier demi-additionneur sort une somme nulle, et le second ne peut donc plus produire de retenue. Le cas « 1 + 1 » ne se présente jamais à l'entrée du OU.

1.5.6 Additionner deux octets

Pour additionner deux nombres de 8 bits, on aligne **huit** additionneurs complets, un par colonne — exactement comme on pose une addition à la main :

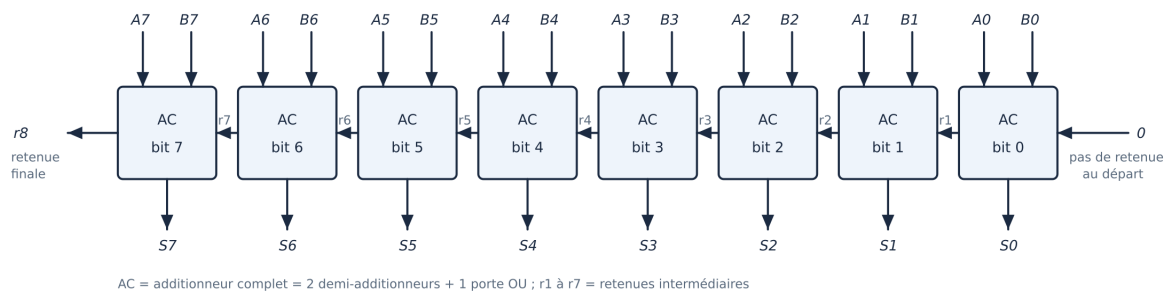


FIGURE 9 – Additionneur 8 bits : huit additionneurs complets en cascade

- la retenue sortante de chaque étage devient la retenue **entrante** de l'étage de gauche : c'est la chaîne r1, r2, ... r7,
- l'étage du bit 0 ne reçoit aucune retenue, son entrée vaut 0 (un simple demi-additionneur suffirait),
- la retenue finale r8 est le **9e bit** du résultat : si elle vaut 1, la somme ne tient pas sur 8 bits, c'est un **dépassement de capacité**.

On parle d'additionneur à **propagation de retenue** : chaque étage doit attendre la retenue de son voisin de droite. C'est ce qui limite la vitesse du circuit, et les processeurs utilisent des montages plus rapides (mais beaucoup plus gros).



Exercice 4 de la feuille Exos expressions booléennes : compter les portes de cet additionneur 8 bits, et y poser une addition qui déborde.

Bout à bout, 8, 32 ou 64 additionneurs complets forment l'unité de calcul d'un processeur.



À retenir

L'addition a été entièrement ramenée à de la **logique booléenne**. C'est le principe fondamental de l'ordinateur : tout ce qu'il calcule est fait avec des portes logiques.