Les booléens

PAR CFAURY · PUBLIÉ 9 JUILLET 2019 · MIS À JOUR 21 MARS 2020

Les machines traitent et mémorisent l'information au moyen de **circuits logiques binaires** : leurs entrées et sorties se caractérisent exclusivement par deux états : l'**état logique bas** et l'**état logique haut**.

> *Ceci s'explique par la technologie employée : les microprocesseurs sont constitués d'une multitude de composants électroniques que l'on appelle des **transistors** et qui ne peuvent prendre que deux états, **bloqué** ou **saturé**, et se comportent comme des interrupteurs. Ce sont des circuits logiques, le plus souvent à base de transistors, qui réalisent toutes les opérations dans les processeurs des machines (voir le modèle d'architecture de Von Neumann) ...*
>
> *Source : https://fr.wikibooks.org/wiki/Fonctionnement_d%27un_ordinateur/Les_transistors_et_portes_logiques*

Informations binaires et Variables booléennes

Une **variable booléenne** ne peut prendre que deux valeurs, notées **0** et **1**.

Ces variables peuvent servir à constituer une **information binaire** *(oui/non, vrai/faux, égal/différent, marche/arrêt, allumé/éteint , ...)* ou à décrire l'état physique d'un composant d'un système *(alimentation d'un composant, action sur un bouton, ...)*

- La valeur **0** représente l'état physique d'un composant non alimenté, ou ne recevant pas d'action physique.
- La valeur **1** représente l'état physique d'un composant alimenté.

exemples : une lampe, résistance, un relais, un contacteur, sont à l'état 0 lorsqu'ils ne sont pas alimentés. Le circuit est alors ouvert.

Formes de l'information binaire

La valeur booléenne est donnée par l'état logique (haut ou bas) de la grandeur physique qui la porte : de l'électricité.

Il existe plusieurs types de circuits intégrés.

> *Exemple : dans le cas de l'utilisation de circuits intégrés en logique T.T.L. (Transistor-Transistor Logic), l'état logique HAUT correspond à une tension « proche » de 5V et l'état logique BAS à une tension « proche » de 0V.*

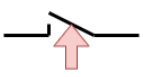
Tension	*État logique*	*Valeur booléenne*
5V	*HAUT*	*1*
	indéterminé	
0V	*BAS*	*0*

Il ne faut pas confondre :

- la valeur de **tension**, en Volt, aux bornes des composants d'un circuit logique,
- l'**état logique** HAUT ou BAS aux bornes des composants d'un circuit logique (qui dépend de la tension),
- la **valeur booléenne**, exprimée à l'aide des deux bits 0 et 1 qui représentent les deux états logiques bas et haut.

Le **bit** est l'unité des informations logiques (*bit* est l'abréviation de ***bi**nary digit*).

Les **transistors** ne peuvent prendre que deux états : **bloqué** ou **saturé** et se comportent comme des interrupteurs que l'on appellera **contacts**. On distingue deux types de contacts :

	au repos (état BAS)	**actionné (état HAUT)**
Contact à fermeture Contact qui est ouvert au repos (ou normalement ouvert) et qui se ferme lorsqu'il est actionné.		
Contact à ouverture Contact qui est fermé au repos et qui s'ouvre lorsqu'il est actionné.		

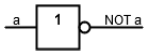
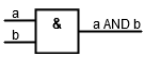
Un contact est à l'état 0 en l'absence d'action physique sur celui-ci, et à l'état 1 s'il est actionné.

Algèbre binaire (ou algèbre de Boole)

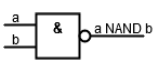
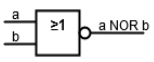
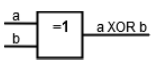
Source : https://fr.wikipedia.org/wiki/Algèbre_de_Boole_(logique)

Opérateurs

Les calculs sur les variables booléennes sont réalisés grâce à l'algèbre de ***Boole*** qui comporte **3 opérateurs élémentaires** :

Opérateur	Équation logique	Symbole logique	Table de vérité
NON (*NOT*)	$NOT\ a = \bar{a}$ $= \neg a$ $= !a$		a | $NOT\ a$ 0 | 1 1 | 0
ET (*AND*)	$a\ AND\ b = a \cdot b$ $= a\ \wedge\ b$ $= a\ \&\ b$		a | b | $a\ AND\ b$ 0 | 0 | 0 0 | 1 | 0 1 | 0 | 0 1 | 1 | 1
OU (*OR*)	$a\ OR\ b = a + b$ $= a\ \vee\ b$ $= a\ \|\|\ b$		a | b | $a\ OR\ b$ 0 | 0 | 0 0 | 1 | 1 1 | 0 | 1 1 | 1 | 1

A partir de ces trois opérateurs élémentaires, on peut également définir les opérateurs suivants :

Opérateur	Équation logique	Symbole logique	Table de vérité
NON-ET (*NAND*)	$a\ NAND\ b = \overline{a \cdot b}$ $= \neg(a\ \wedge\ b)$ $= !(a\ \&\ b)$		a | b | $a\ NAND\ b$ 0 | 0 | 1 0 | 1 | 1 1 | 0 | 1 1 | 1 | 0
NON-OU (*NOR*)	$a\ NOR\ b = \overline{a + b}$ $= \neg(a\ \vee\ b)$ $= !(a\ \|\|\ b)$		a | b | $a\ NOR\ b$ 0 | 0 | 1 0 | 1 | 0 1 | 0 | 0 1 | 1 | 0
OU exclusif (*XOR*)	$a\ XOR\ b = a \oplus b$ $= a\ \underline{\vee}\ b$		a | b | $a\ XOR\ b$ 0 | 0 | 0 0 | 1 | 1 1 | 0 | 1 1 | 1 | 0

> *Remarques :*
>
> - *les opérateurs NON-ET et NON-OU sont qualifiés d'**opérateurs complets** car ils permettent la réalisation des trois opérateurs élémentaires NON, ET et OU.*
> - *NAND est l'opérateur de base des circuits intégrés en logique T.T.L. (Transistor-Transistor Logic)*

Propriétés fondamentales de l'algèbre de BOOLE

Éléments neutres

- 0 est élément neutre de la fonction OU : $a + 0 = a$
- 0 est élément absorbant de la fonction ET : $a \cdot 0 = 0$
- 1 est élément neutre de la fonction ET : $a \cdot 1 = a$
- 1 est élément absorbant de la fonction OU : $a + 1 = 1$

Complémentarité

- $a + \bar{a} = 1$
- $a \cdot \bar{a} = 0$

Commutativité

- du produit logique : $a \cdot b = b \cdot a$
- de la somme logique : $a + b = b + a$

Distributivité

- de la fonction ET par rapport à la fonction OU : $a \cdot (b + c) = a \cdot b + a \cdot c$
- de la fonction OU par rapport à la fonction ET : $a + (b \cdot c) = (a + b) \cdot (a + c)$

Absorption

- $a + a \cdot b = a \cdot 1 + a \cdot b = a \cdot (1 + b) = a$

Idempotence

- $a + a = a$
- $a \cdot a = a$

Associativité

- du produit logique : $a \cdot (b \cdot c) = (a \cdot b) \cdot c = a \cdot b \cdot c$
- de la somme logique : $a + (b + c) = (a + b) + c = a + b + c$

Théorèmes de DE MORGAN

- Premier théorème : $\overline{a + b} = \bar{a} \cdot \bar{b}$
- Deuxième théorème : $\overline{a \cdot b} = \bar{a} + \bar{b}$

Table de vérité

Une **expression logique** contenant un nombre fini de variables booléennes ($a_1, \dots, a_n$), et chaque variable ne pouvant prendre qu'un nombre fini de valeurs (0 ou 1), il existe un nombre fini de combinaisons de ces variables, et par conséquent un nombre fini de valeurs pour l'expression. On peut représenter l'ensemble des combinaisons et des valeurs dans un tableau appelé **table de vérité** :

$\forall i \in [1, n],\ A = f(a_1, \dots, a_n)$

a_1	a_2	...	a_n	A
0	0	...	0	
0	0	...	1	
⋮	⋮	⋮	⋮	
1	1	...	1	

On peut utiliser une table de vérité pour décomposer les calculs algébriques, en procédant par étapes :

> *Exemples :*

- $A = a + \bar{b}$

a	b	$\bar{b}$	$A = a + \bar{b}$
0	*0*	*1*	*1*
0	*1*	*0*	*0*
1	*0*	*1*	*1*
1	*1*	*0*	*1*

- $B = \overline{a + b} \cdot c$

a	b	c	$a + b$	$\overline{a + b}$	$B = \overline{a + b} \cdot c$
0	*0*	*0*	*0*	*1*	*0*
0	*0*	*1*	*0*	*1*	*1*
0	*1*	*0*	*1*	*0*	*0*
0	*1*	*1*	*1*	*0*	*0*
1	*0*	*0*	*1*	*0*	*0*
1	*0*	*1*	*1*	*0*	*0*
1	*1*	*0*	*1*	*0*	*0*
1	*1*	*1*	*1*	*0*	*0*

> *Activité :*

- *Utiliser une table de vérité pour évaluer l'ensemble des valeurs possibles des expressions suivantes :*
  - $X = \overline{(a \cdot b)} + \bar{c}$
  - $Y = \overline{(a + \bar{c})} + \overline{(b \cdot c)}$

On peut aussi utiliser une table de vérité pour vérifier si deux expressions logiques sont équivalentes :

> *Activité :*
>
> - *Montrer que* $a + (\bar{a} \cdot b) = a + b$
> - *Simplifier l'expression* $(a + b) \cdot (a + \bar{b})$
> - *Utiliser une table de vérité pour démontrer la propriété d'absorption de l'algèbre de Boole.*

On peut encore utiliser une table de vérité pour obtenir une expression logique à partir d'une combinaison (complète ou pas) de valeurs logiques :

> *Activité :*
>
> - *A partir de la table de vérité de la fonction XOR, déterminer une expression booléenne à base d'opérateurs élémentaires.*
> - *Déterminer l'expression booléenne correspondant à la table de vérité ci-dessous :*

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

Règles et syntaxe dans les langages informatiques

	Python	C/C++
Valeurs booléennes	True False	true false
Opérateurs booléens		
NON	not a	!a
ET	a and b	a&&b
OU	a or b	a||b
XOR	a^b	a!=b
Comparaison Tous les types peuvent être comparés, une comparaison renvoie toujours un booléen.		
égalité	==	
différence	!=	
inférieur ou égal	<=	
strictement inférieur	<	
supérieur ou égal	>=	
strictement supérieur	>	

> *Remarques :*
>
> *Python et C++ évaluent les expressions logiques de manière* ***paresseuses*** *(on parle de lazy evaluation). Les opérateurs sont de type* ***court-circuit*** *:*
>
> - *OR n'évalue le deuxième argument que si le premier est faux.*
> - *AND n'évalue le deuxième argument si le premier est vrai.*
>
> *Exemple, en Python :*
>
> - *True or x : est vrai quelle que soit la valeur de x : la valeur de x n'est pas lue dans la mémoire et l'opération or n'est pas effectuée en entier.*
> - *False and not y : est faux quelle que soit la valeur de y : la valeur de y n'est pas lue dans la mémoire et l'opération and n'est pas effectuée en entier, l'opération not n'est pas effectuée du tout.*

> *Activité :*
>
> - *Ouvrir une console Python et montrer le caractère* ***lazy*** *de l'évaluation des expressions booléennes.*