Logique

I Du transistor à l'ordinateur

Le transistor est à la base de la plupart des composants d'un ordinateur. Pour faire simple c'est un composant avec une entrée, une sortie, et une alimentation.Quand il est alimenté, le transistor laisse passer le courant de l'entrée vers la sortie et dans le cas contraire le courant ne passe pas.
C'est l'élément de base des *circuits logiques*.

Définition : circuit logique

Un circuit logique prend en entrée un ou plusieurs signaux électriques. Chacun de ses signaux peut être dans l'état 0 ou l'état 1.
En sortie, le circuit logique produit un signal (0 ou 1) obtenu en appliquant des *opérations booléennes* aux signaux d'entrée.

Un circuit logique est une implémentation matérielle d'une *fonction logique*. La fonction logique est, quant à elle, la version « mathématique » du circuit. Nous confondrons ces deux notions par la suite.

1 Opérateurs logiques de base

À l'aide des opérateurs suivants, on peut construire toutes les fonctions logiques.

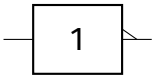
L'opérateur « non »

C'est un opérateur **unaire** : il ne prend qu'une seule variable booléenne en entrée.

Table de valeurs

x	*non* x
0	1
1	0

Symbole de porte logique européen



Cet opérateur renvoie « le contraire de ce qu'il a reçu ».
Parmi les notations que l'on rencontre pour noter « non x » il y a

- `NOT x`
- $\overline{x}$
- $!x$

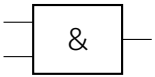
L'opérateur « et »

C'est un opérateur **binaire** : il prend deux variables booléennes en entrée.

Table de valeurs

x	y	x *et* y
0	0	0
0	1	0
1	0	0
1	1	1

Symbole de porte logique européen



Un « et » ne renvoie vrai que si ses deux entrées sont vraies.
Parmi les notations que l'on rencontre pour noter « x et y » il y a

- `x AND y`
- $x \wedge y$
- *x*&&*y*

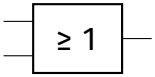
L'opérateur « ou »

C'est également un opérateur **binaire**.

Table de valeurs

x	y	x *ou* y
0	0	0
0	1	1
1	0	1
1	1	1

Symbole de porte logique européen



Un « ou » ne renvoie faux que si ses deux entrées sont fausses.
Parmi les notations que l'on rencontre pour noter « x ou y » il y a

- `x OR y`
- $x \vee y$
- $x || y$

On peut montrer qu'il est possible de se passer de la fonction *et* et que toutes les fonctions logiques peuvent s'écrire à l'aide de fonctions *non* et *ou* (on peut même n'utiliser qu'une seule fonction : la fonction « non ou »). Le choix de ces trois fonctions *et*, *ou* et *non* est donc, en quelque sorte, arbitraire.

Définition : Équivalence de deux circuits/fonctions logiques

On dira que deux fonctions logiques sont équivalentes lorsqu'elles prennent le même nombre de variables en entrée (le même nombre de signaux si on parle de circuits) et si ces deux fonctions donnent le même résultat lorsque les variables d'entrées ont les mêmes valeurs : on dit que les fonctions ont même *table de vérité*. Lorsque deux fonctions logiques sont équivalentes, on dit aussi que leurs expressions booléennes sont équivalentes.

Exemples

- Les expressions booléennes

$$not(A\ and\ B)$$

et

$$(not\ A)\ or\ (not\ B)$$

sont équivalentes.
Pour le prouver il suffit de vérifier que leurs tables de vérité sont les mêmes : on fait varier les valeurs de A et de B selon toutes les possibilités et on regarde le résultat de chaque expression.

A	B	A and B	not (A and B)
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

A	B	not A	not B	(not A) or (not B)
0	0	1	1	1
0	1	1	0	1
1	0	0	1	1
1	1	0	0	0

Les dernières colonnes de chaque tableau sont les mêmes : les expressions sont donc

équivalentes.

- En procédant de même on montre très facilement que « non non x » et « x » sont équivalentes.

Exercice 1

Montrer que

$$not(A \text{ or } B)$$

et

$$(not\ A)\ and\ (not\ B)$$

sont équivalentes.

Exercice 2

On peut définir le « ou exclusif » noté xor comme ceci : A xor B n'est vrai que si A est vrai ou B est vrai, mais pas les deux.

1. Donner la table de vérité de xor.
2. Montrer que A xor B équivaut à (A or B) and (not(A and B)).
3. Montrer que A xor B équivaut à (A and (not B)) or ((not A) and B).
4. Représenter A xor B avec un circuit logique, en utilisant les symboles de porte logique européens.

Exercice 3

On définit l'opération « nor », notée $\downarrow$ par :

$$A \downarrow B = \textit{not}(A \textit{ or } B)$$

Cette opération est dite *universelle* car elle permet de retrouver toutes les autres opérations.

1. Écrire la table de vérité de nor.
2. Montrer que $A \downarrow A = \textit{not } A$.
3. En utilisant les exemples de la page précédente, en déduire que $(A \downarrow B) \downarrow (A \downarrow B) = A \textit{ or } B$.
4. Comment à partir de A, B et $\downarrow$ obtenir $A \textit{ and } B$?

2 Un exemple détaillé : le multiplexeur

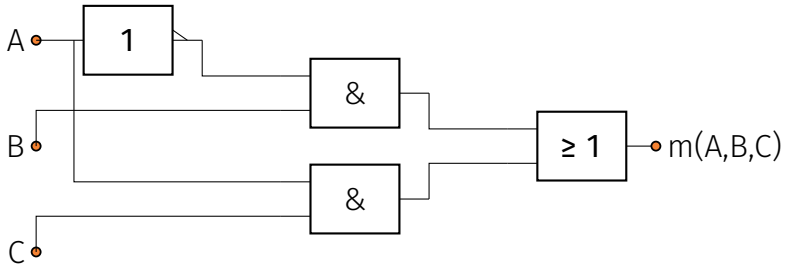
Il s'agit d'une fonction très importante : soient A, B et C trois variables logiques, alors m(A,B,C)=B si A vaut 0 et C si A vaut 1.
m permet donc de sélectionner B ou C suivant la valeur de A. Voici la table de valeurs de m :

A	B	C	m(A,B,C)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

On va décomposer m à l'aide des opérateurs de base. Ici, un raisonnement simple permet d'y arriver :

- quand A vaut 0, on peut garder la valeur de B en faisant (*non* A) *et* B ;
- quand A vaut 1, on peut garder la valeur de C en faisant A *et* C ;
- il se trouve que ces deux observations vont bien ensemble car quand A vaut 0, A *et* C vaut automatiquement 0, et quand A vaut 1, (*non* A) *et* B vaut automatiquement 0 ;
- on en conclut que l'**expression symbolique** de m est m(A,B,C) = (A *et* C) ou ((*non* A) *et* B).

Le circuit logique modélisant m est le suivant :



Exercice 4

On veut construire un « additionneur » selon le principe suivant :

- A et B représentent deux bits à ajouter
- S et R sont respectivement
  - la somme (sur un bit, donc) de A et B ;
  - la retenue.

Par exemple si A et B valent 1, alors S vaudra 0 et R vaudra 1.

1. Donner les tables de vérités de R et de S.
2. Exprimer R et S en fonction de A et B et des opérations « non », « ou » et « et ».
3. Représenter R et S avec un (ou des) circuit(s) logique(s), en utilisant les symboles de porte logique européens.

Exercice 5

Pour chiffrer un message, une méthode, dite du masque jetable, consiste à le combiner avec une chaîne de caractères de longueur comparable. Une implémentation possible utilise l'opérateur XOR (ou exclusif) dont voici la table de vérité :

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

Dans la suite, les nombres écrits en binaire seront précédés du préfixe `0b`.

1. Pour chiffrer un message, on convertit chacun de ses caractères en binaire (à l'aide du format UNICODE), et on réalise l'opération XOR bit à bit avec la clé.

   Après conversion en binaire, et avant que l'opération XOR bit à bit avec la clé n'ait été effectuée, Alice obtient le message suivant :

`m = 0b 0110 0011 0100 0110`

a. Le message `m` correspond à deux caractères codés chacun sur 8 bits : déterminer quels sont ces caractères. On fournit pour cela la table ci-dessous qui associe à l'écriture hexadécimale d'un octet le caractère correspondant.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	space	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Exemple de lecture : le caractère correspondant à l'octet codé `4A` *en hexadécimal est la lettre* `J`.

b. Pour chiffrer le message d'Alice, on réalise l'opération XOR bit à bit avec la clé suivante :

```
k = 0b 1110 1110 1111 0000
```

Donner l'écriture binaire du message obtenu.

2. **a.** Donner la table de vérité de l'expression booléenne `(a XOR b) XOR b`.

b. Bob connaît la chaîne de caractères utilisée par Alice pour chiffrer le message. Quelle opération doit-il réaliser pour déchiffrer son message ?