

Expressions booléennes

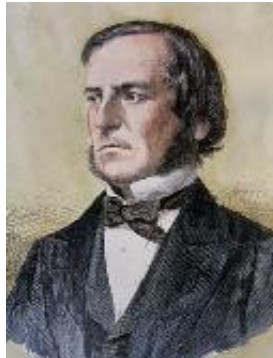
BRUNO DARID

14 août 2019

PLAN

1 Repères historiques	2
2 Quelques définitions	2
3 Les opérations logiques élémentaires	3
3.1 Les symboles	3
3.2 Simulations	3
3.2.1 E1C3 Opérateurs et, ou et non	3
3.2.2 E2C3 Lois de De Morgan	3
3.2.3 E3C3 Une opération logique très utilisée	3
4 Problèmes	4
4.1 P1C3 Addition binaire	4
4.1.1 Demi additionneur	4
4.1.2 Additionneur complet	5
4.1.3 Additionneur complet en python	5
4.2 P2C3 Un circuit important : le multiplexeur (<i>pour les plus rapides ET curieux</i>)	6
4.2.1 Principe	6
4.2.2 Simulation	7
5 À retenir	7

1 Repères historiques



[George Boole](#) (1815-1864) : mathématicien, logicien britannique, auteur d'une algèbre **binaire** dite **booléenne** n'acceptant que deux valeurs 0 et 1.

[Lien](#) vers la présentation de Marie Duflot-Kremer, chercheuse en informatique l'Université de Lorraine.

Aujourd'hui, l'algèbre de Boole trouve de nombreuses applications en informatique et dans la conception des circuits électroniques

2 Quelques définitions

On appelle **valeur logique** ou **valeur booléenne** toute valeur notée par deux symboles. On peut utiliser par exemple un des couples de valeurs suivants : $\{0,1\}$, $\{\text{vrai}, \text{faux}\}$, $\{\text{true}, \text{false}\}$ ou $\{\text{ouvert}, \text{fermé}\}$.

Exemple : l'état d'un interrupteur a une valeur booléenne, il peut être *ouvert* ou *fermé*.

Une **variable booléenne** (ou variable logique) est une grandeur représentée par un nom et pouvant prendre une valeur booléenne.

L'algèbre de Boole est caractérisée par la donnée :

- de deux opérations binaires **OU** et **ET** (*correspondant à la somme "+" et au produit "."*)
- d'une opération unaire **NON** (*correspondant au complémentaire "* \neg *"*).

Ces opérations doivent vérifier certaines conditions qui ne seront pas exposées ici. En mathématique on trouve aussi les notations \vee (*disjonction*), \wedge (*conjonction*) et \neg (*négation*).

L'association de variables booléennes et d'opérateur(s) booléen(s) produit une **expression booléenne**.

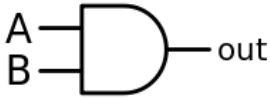
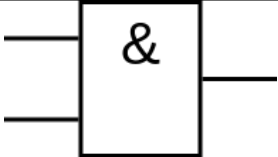
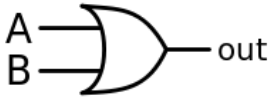
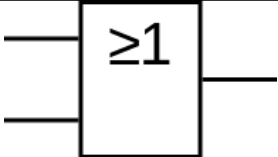
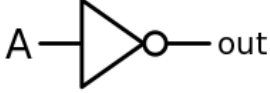
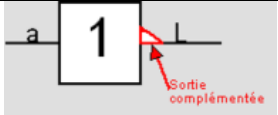
Exemple : si a , b et c sont trois variables booléennes, $a \cdot b \cdot \bar{c} + a \cdot \bar{b} \cdot c$ est une expression booléenne.

Dresser la **table de vérité** d'une expression booléenne signifie construire une table ayant autant de colonnes que de variables d'entrée plus une colonne donnant le résultat (*vrai* ou *faux*, 0 ou 1) pour chaque combinaison possible des variables d'entrée.

3 Les opérations logiques élémentaires

3.1 Les symboles

Les opérations logiques sont réalisées simplement avec des circuits électroniques (à base de transistors) appelés **portes logiques**. Voici les symboles des portes logiques utilisées pour les opérations ET, OU et NON.

Type	Symbole US	Symbole EU
ET		
OU		
NON		

Note : les symboles *EU* sont très peu utilisés (même en France!).

3.2 Simulations

3.2.1 E1C3 Opérateurs et, ou et non

Simuler avec le logiciel *Digital* du [professeur Helmut Neemann](#) les opérations logiques ET, OU et NON. Établir leur table de vérité.

3.2.2 E2C3 Lois de De Morgan

1. Dresser la table de vérité de l'expression $\overline{A + B}$
2. Vérifier les résultats par simulation
3. Proposer une expression équivalente
4. Reprendre les questions 1, 2 et 3 avec l'expression $\overline{A \cdot B}$

3.2.3 E3C3 Une opération logique très utilisée

1. Ouvrir le circuit "Dilemme.dig" (voir fig. 1)
2. Réaliser la simulation. Établir et commenter sa table de vérité. Cette opération très utilisée, appelée **OU Exclusif** (voir fig. 2) a pour symbole (*US*) :

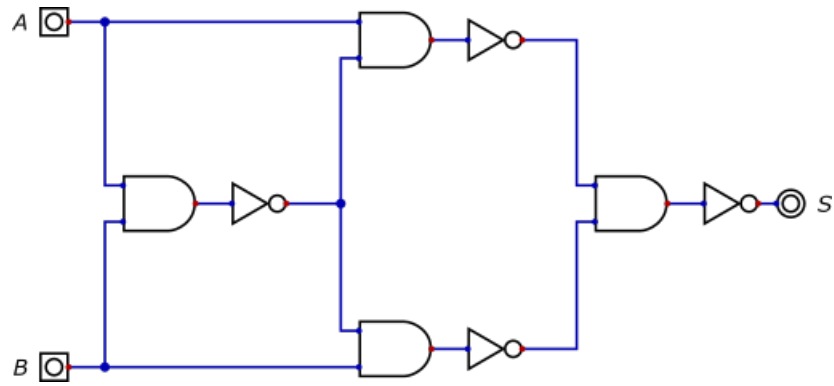


Fig. 1 – Dilemme!



Fig. 2 – Ou exclusif

4 Problèmes

4.1 P1C3 Addition binaire

4.1.1 Demi additionneur

Le demi additionneur fig. 3 est un circuit combinatoire qui permet de réaliser la somme arithmétique de deux nombres A et B chacun sur un bit. À la sortie on va avoir la somme S et une éventuelle retenue R.

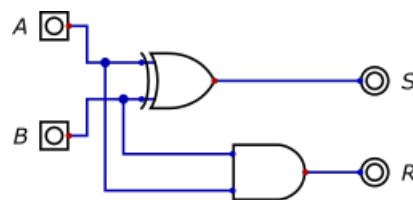


Fig. 3 – Demi additionneur

1. Ouvrir le fichier `halfAdder.dig` et passer en mode simulation.
2. Dresser alors la table de vérité de ce circuit.
3. En déduire la table d'addition binaire.
4. En utilisant la fonction d'analyse du logiciel, donner l'expression booléenne de la sortie S et de la retenue R

4.1.2 Additionneur complet

Lorsqu'on souhaite effectuer l'addition binaire sur un bit de deux nombres A_i et B_i , il faut tenir compte d'une éventuelle retenue R_{i-1} provenant du calcul du rang précédent. On a alors un additionneur complet, qui est réalisé avec deux demi-additionneurs (voir fig. 4). À la sortie on va avoir la somme S_i et une éventuelle retenue R_i . A la sortie on va avoir la somme S_i et une

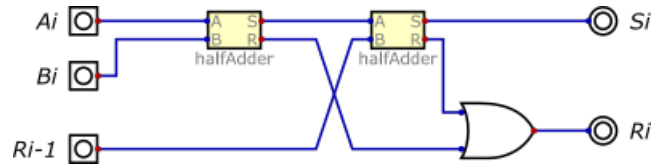


Fig. 4 – Additionneur complet

éventuelle retenue R_i .

1. Ouvrir le fichier `fullAdder.dig` et passer en mode simulation.
2. Dresser la table de vérité de l'additionneur complet.

4.1.3 Additionneur complet en python

Python possède un type booléen nommé `bool`. Cependant, les valeurs booléennes `True` et `False` peuvent être associées aux entiers 1 et 0. On peut le vérifier sur l'exemple suivant :

```
In [45]: A = True
         B = False
         print("A == 1 ? : ", A == 1)
         print("B == 1 ? : ", B == 1)
         print("B == 0 ? : ", B == 0)
```

```
A == 1 ? : True
B == 1 ? : False
B == 0 ? : True
```

L'expression booléenne de la sortie S_i de l'additionneur complet est :

$$S_i = (\overline{A_i} \cdot \overline{B_i} \cdot R_{i-1}) + (\overline{A_i} \cdot B_i \cdot \overline{R_{i-1}}) + (A_i \cdot \overline{B_i} \cdot \overline{R_{i-1}}) + (A_i \cdot B_i \cdot R_{i-1})$$

Celle de la retenue est :

$$R_i = (A_i \cdot R_{i-1}) + (A_i \cdot B_i) + (B_i \cdot R_{i-1})$$

En python, les opérateurs booléens sont : `and`, `or` et `not`. Ainsi l'expression booléenne $\overline{A} \cdot (B + \overline{C})$ se traduit par :

```
not A and (B or not C)
```

Remarque : une expression du type

```
expr1 and expr2
```

ou

`expr1 or expr2`

conduit à une évaluation séquentielle c'est-à-dire que python va évaluer *expr1* puis suivant sa valeur, va évaluer ou non *expr2*. Voyez-vous pourquoi ?

On définit une fonction en python ayant le code suivant :

```
In [51]: def addb(n,a,b):
         """
         Retourne ....
         On suppose que a et b sont des chaines constitués des caractères
         appartenant à {'0','1'}; n est un entier naturel supérieur à zéro.
         """
         ri_1 = False
         res = ""
         for i in range(n-1,-1,-1):
             ai = int(a[i])
             bi = int(b[i])
             si = (not ai and not bi and ri_1) or (not ai and bi and not ri_1) or \
                 (ai and not bi and not ri_1) or (ai and bi and ri_1)
             ri_1 = (ai and ri_1) or (ai and bi) or (bi and ri_1)
             res = str(int(si)) + res
         if ri_1:
             res = '1' + res
         return res
```

1. Consulter la [documentation officielle](#) de python sur la fonction range afin de comprendre la construction qui apparait à la ligne 9.
2. Les spécifications étant incomplètes, expliquer ce que réalise cette fonction.
3. Vérifier vos hypothèses avec quelques tests ; par exemple :
 - `addb(4, '0001', '0001')`
 - `addb(8, '11111111', '00000001')`
4. Quelle est l'utilité du test ligne 16 ?

4.2 P2C3 Un circuit important : le multiplexeur (pour les plus rapides ET curieux)

Le [multiplexeur](#) est un circuit logique très important en architecture machine. Il permet de sélectionner une entrée parmi N et de transférer sa valeur sur la sortie. La sélection se fait à l'aide d'une entrée *Select*.

4.2.1 Principe

La figure 5 présente le schéma de principe d'un multiplexeur à 4 entrées $A_0...A_3$ ainsi que sa réalisation avec les portes logiques élémentaires.

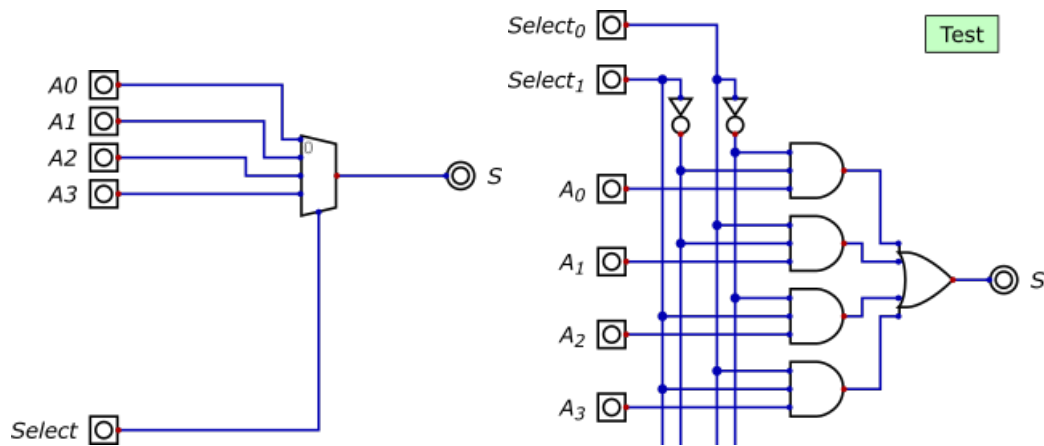


Fig. 5 – Multiplexeur

4.2.2 Simulation

1. Ouvrir le fichier mux2.dig et passer en mode simulation.
2. On souhaite sélectionner la deuxième entrée, quelle valeur logique doit-on affecter à $Select_0$ et $Select_1$?
3. Réaliser la sélection précédente. Entrer la combinaison (1, 1, 0, 0) pour (A_0, A_1, A_2, A_3). Que vaut S ? Expliquer.
4. Garder la même sélection. Donner une combinaison qui conduit à $S = 1$. Expliquer.

5 À retenir

L'algèbre de Boole est caractérisée par les trois opérations OU, ET et NON. Avec ces opérateurs et des variables booléennes on peut construire des expressions logiques. Dresser la table de vérité d'une expression signifie construire un tableau qui donne la valeur logique de l'expression pour chaque combinaison des variables d'entrées. Quelques résultats importants :

A	B	A OU B	A ET B	NON A
0	0	0	0	1
0	1	1	0	1
1	0	1	0	0
1	1	1	1	0

Ce(tte) œuvre est mise à disposition selon les termes de la Licence [Creative Commons Attribution - Pas d'Utilisation Commerciale 4.0 International](#).

