TP Architecture - En plus

M. Tellene

Les exercices concernant la construction de circuit ou de fonctions Python devront être sauvegardées dans le dossier **« TP_architecture »**.

1 Changeur de monnaie - Sur *SmartSim* et sur feuille

On désire réaliser un circuit pilotant un changeur de monnaie. Le changeur devant fonctionner ainsi : on introduit **une et une seule** pièce parmi `5€`, `2€`, `1€`, `0.5€` et il rend un nombre de pièces définit par le tableau suivant :

Pièce introduite	Pièces rendues	Nombre
5€	1€	3
	0.5€	2
	0.2€	3
	0.1€	4
2€	1€	1
	0.5€	1
	0.2€	1
	0.1€	3
1€	0.2€	2
	0.1€	6
0.5€	0.1€	5

Les variables d'entrées correspondant aux pièces introduites seront notées `C`, `D`, `U`, `Q` respectivement pour les pièces `5€`, `2€`, `1€`, `0.5€`

Les sorties du circuit correspondront aux nombres de pièces rendues codés en binaire de chacune des pièces. Les noms des sorties seront notés u_i, c_i, v_i, d_i respectivement pour les pièces `1€`, `0.5€`, `0.2€`, `0.1€`

Un peu d'aide à la compréhension : étant donné que les sorties correspondront aux nombres de pièces rendues codés en binaire, cela veut dire que si l'on donne une pièce de 5€ alors, on doit rendre 3 pièces de 1, 2 pièces de 0.5, 3 pièces de 0.2 et 4 pièces de 0.1. Pour le cas de la pièces de 1, la machine nous rendra 3 pièces soit $(11)_2$.

Comment utiliser les sorties ? Nous aurons une table de vérité de la forme (table à compléter) :

C	D	U	Q	u_1	u_0	...
1	0	0	0	1	1	...
...	...	...	...	...	...	...

1. Déterminer, pour chacune des pièces rendues, le nombre de sorties nécessaires
2. Écrire la table de vérité de toutes les fonctions correspondantes (afin de simplifier, nous considérerons que lorsqu'une entrée est à 1, les autres sont forcément à 0)
3. Dessiner le circuit

2 Un peu de Python

Vous sauvegarderez les réponses à cet exercice dans un fichier nommé **portes_logiques.py**
Nous allons créer des fonctions permettant de représenter des tables de vérités. Soit la fonction `porte_et()` :

```
def porte_et():
    print("a", "b", "S")
    for a in (False, True):
        for b in (False, True):
            print(int(a), int(b), int(a and b))
```

1. Recopier et exécuter cette fonction afin de comprendre ce qu'elle fait
2. Sur le modèle de la fonction `porte_et()`, écrire une fonction `porte_ou()`
3. Toujours en vous basant sur la fonction `porte_et()`, écrire une fonction `porte_xor()` (le mot-clé `xor` n'existe pas en Python)

3 Pour aller plus loin - Logique des prédicats - Sur feuille

Dans ce chapitre, nous avons vu les portes logiques et la logique combinatoire. Il existe bien d'autres logiques et cette partie s'intéressera à la logique des prédicats. Voici un exemple :

Nous avons une phrase de départ et le but est de donner à chaque fois l'interprétation des prédicats utilisés — par exemple `A(x,y) = x aime y`. Il est possible de lier plusieurs prédicats ensemble - par exemple `A(x,y) AND A(x,z) = x aime y et z`.

Pour chacune des phrases suivantes, les traduire en énoncé logique (la première est donnée à titre d'exemple) :

1. Jean est plus grand que Marie → `G(j,m)`
2. Paul a vu Léa et elle ne l'a pas vu
3. Un chat est entré
4. Certains enfants ne sont pas malades
5. Yoann aime les glaces à la menthe et au chocolat mais pas celle a la vanille

Il sera possible d'utiliser les prédicats suivants :

Prédicat	Nombre d'arguments	Signification
`∃(x)`	1	`∃(x) = il existe un x`
`C(x)`	1	`C(x) = x est un chat`
`Enf(x)`	1	`Enf(x) = x est un enfant`
`Ent(x)`	1	`Ent(x) = x est entré`
`GC(x)`	1	`GC(x) = x est une glace au chocolat`
`GM(x)`	1	`GM(x) = x est une glace à la menthe`
`GV(x)`	1	`GV(x) = x est une glace à la vanille`
`M(x)`	1	`M(x) = x est malade`
`A(x,y)`	2	`A(x,y) = x aime y`
`G(x,y)`	2	`G(x,y) = x est plus grand que y`
`V(x,y)`	2	`V(x,y) = x a vu y`