Python (M1) – Des objets et des circuits combinatoires

TD #2

Rédacteur: Stéphane Vialette

Circuits combinatoires

Le but de ce TD est développer un module python élémentaire permettant de créer et d'évaluer des circuits combinatoires.

Quelques définitions et rappels. Un *circuit combinatoire* est un circuit à n entrées et m sorties, où l'état de chaque sortie (Faux ou Vrai) dépend uniquement de l'état des entrées. Les circuits combinatoires les plus simples sont aussi appelés des *portes logiques*. Nous considérerons dans un premier temps les portes suivantes :

- **Porte Non** : La porte Non est la plus simple qui soit. Ce circuit a une entrée e et une sortie s. L'état de s est l'opposé de celui de e.
- **Porte Et** : La porte Et est pratiquement aussi simple. Ce circuit comporte en effet deux entrées e_1 et e_2, et une sortie s. La sortie est à l'état Vrai si et seulement si e_1 et e_2 sont à l'état Vrai, d'où le nom de cette porte.
- **Porte Ou** : La porte Ou comporte également deux entrées e_1 et e_2 et une sortie s. Comme l'indique le nom de cette porte, la sortie est à l'état Vrai si et seulement si e_1 est à l'état Vrai ou e_2 est à l'état Vrai.
- **Porte Diff** : La porte Diff comporte deux entrées e_1 et e_2 et une sortie s. Comme l'indique le nom de cette porte, la sortie est à l'état Vrai si et seulement si l'état de e_1 et l'état de e_2 sont différents.

Premiers éléments

Entrées, sorties, portes, ... il va nous falloir organiser tout ceci dans un formalisme objet. Commençons par définir les entrées. Nous utilisons pour cela un module que nous nommons `entreebinaire` (fichier `entreebinaire.py`) donné Figure **??**. Le test `if self.__class__ is EntreeBinaire:` de la méthode `__init_` de la classe `EntreeBinaire` est une astuce classique en python pour simuler (et uniquement simuler) une classe abstraite.

[FIGURE 1 about here.]

Passer aux portes logiques (unaires et binaires). Nous utiliserons la hiérarchie donnée dans le squelette du module `portelogique` (fichier `squelette_portelogique.py`) (voir Figure **??**).

[FIGURE 2 about here.]

Le circuit combinatoire en lui-même est au final un objet assez simple. Il est défini par un ensemble d'entrées binaires et un ensemble de sorties binaires. Nous donnons le code complet du module `circuitcombinatoire` en Figure **??** (disponible via fichier `circuitcombinatoire.py`). Si vous examinez attentivement ce module, vous remarquerez que la classe `CircuitCombinatoire` ne propose que deux méthodes : `__init__(self, entrees, sorties)` (pour initialiser le circuit combinatoire aves ses entrées et ses sorties) et `sortie(self, i)` (pour calculer la valeur booléenne de la i-ème sortie de ce circuit combinatoire).

[FIGURE 3 about here.]

Avant de coder, il nous reste à comprendre comment ce module est utiliser. Un exemple d'utilisation est donné en Figure fig :main (fichier `main.py`). Assurez vous de bien comprendre ce code (en particulier la logique de construction de n'importe quel circuit combinatoire) avant de poursuivre. Il est important de voir que la création d'une instance de la classe `CircuitCombinatoire` passe par la création d'entrées (binaires) et de sorties (binaires également).

[FIGURE 4 about here.]

Question 1. Compléter les classes PorteLogique, Non, PorteLogiqueBinaire, Ou, Et et Diff. Tester intensivement votre implémentation avant de poursuivre.

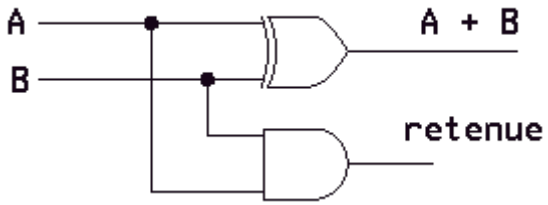
Question 2. La porte NonEt est la négation d'une porte Et. Proposer deux façons d'implémenter cette classe. Une première version calquée sur l'implémentation de la porte Et (il vous faudra donc placer la classe `NonEt` dans la hiérarchie des portes logiques), et une seconde version qui utilise les classes `Et` et `Non` précédemment définies.

Question 3. La porte Mux comporte trois entrées e_1, e_2 et e_3 et une sortie s. Si e_3 est à l'état Faux, l'état de la sortie est égal à celui de e_1. Si e_3 est à l'état Vrai, l'état de la sortie est égal à celui de e_2. Cette porte est donc une sorte d'aiguillage, qui permet de faire passer en sortie soit l'entrée e_1 soit l'entrée e_2, en fonction de l'état de l'entrée de commande e_3. Proposer une implémentation de la classe `Mux` qui représente une porte Mux (vous implémentation devra en particulier permettre l'introduction aisée de nouvelles portes logiques ternaires). Tester votre implémentation.

1 Construction d'un additioneur

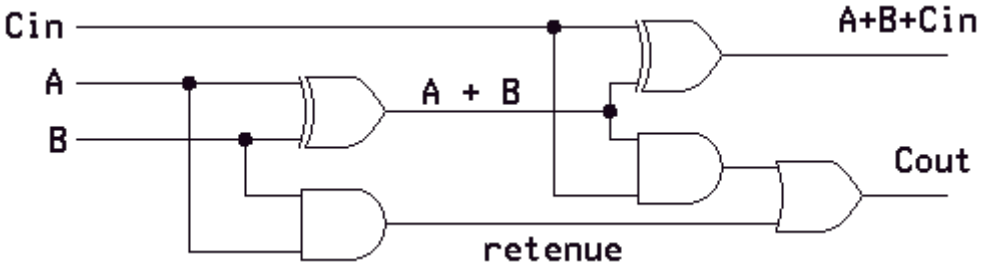
Un *additionneur* est un circuit admettant en entrée deux mots de n bits, et fournissant en sortie le résultat de l'addition binaire des deux mots d'entrée, sur $n + 1$ bits.

Un *demi-additionneur* 1 *bit* est un circuit combinatoire à 2 entrées et 2 sorties réalisant l'addition binaire de 2 bits (résultat et retenu).



Question 4. En associant Faux à 0 et Vrai à 1 (la convention naturelle), écrire une fonction `DemiAdditionneurUnBit` retournant un circuit combinatoire réalisant ce demi-additionneur 1 bit.

Un module de n bits est dit *cascadable* s'il tient compte d'une retenue en entrée (Carry in) et fournit une retenue en sortie (Carry out). Ceci permet de construire des modules d'une largeur de deux fois n bits en cascadant deux modules n bits, de trois fois n bits en cascadant trois modules, ...



Pour obtenir un additionneur n bits, il suffit de cascader ce module n fois, en forçant l'entréee Cin du premier module à 0, et en cascadant la sortie Cout à l'entrée Cin d'un module à l'autre. La sortie Cout du dernier module correspond au $n+$*-ème bit de l'additionneur n bits.

Question 5. Écrire une fonction `AdditionneurCascadable` retournant un circuit combinatoire réalisant cet additionneur cascadate.

Question 6. Écrire une fonction `Additionneur` retournant un circuit combinatoire réalisant un additionneur n bits.

```
##############################################################################

class EntreeBinaire(object):
        '''
        Classe de base pour les entrees binaires.
        '''
        def __init__(self, valeur):
                '''
                Initialise cette entree binaire a partir d'une valeur donnee binaire
                '''
                if self.__class__ is EntreeBinaire:
                        raise TypeError('EntreeBinaire est une classe abstraite')
                self._valeur = valeur

        def evaluer(self):
                '''
                Retourne la valeur binaire de cette entree
                '''
                return self._valeur

class Faux(EntreeBinaire):
        '''
        Entree constante Faux
        '''
        def __init__(self):
                super(Faux, self).__init__(False)

class Vrai(EntreeBinaire):
        '''
        Entree constante Vrai
        '''
        def __init__(self):
                super(Vrai, self).__init__(True)

##############################################################################
```

FIGURE 1 –

```
##########################################################################

class PorteLogique(object):
        '''
        Classe de base (vide) pour les portes logiques
        '''
        pass

class Non(PorteLogique):
        '''
        Porte logique (unaire) Non
        '''
        def __init__(self, entree):
                '''
                Initialisation de la porte logique unaire
                '''
                pass # A COMPLETER

        def evaluer(self):
                '''
                Evaluation: negation de l'entree binaire
                '''
                pass # A COMPLETER

class PorteLogiqueBinaire(PorteLogique):
        '''
        Classe de base (abstraite) pour les portes logiques binaires
        '''
        def __init__(self, entree_1, entree_2):
                '''
                Initialisation de la porte logique binaire
                '''
                pass # A COMPLETER

class Ou(PorteLogiqueBinaire):
        '''
        Porte logique (binaire) Ou
        '''
        def evaluer(self):
                '''
                Evaluation de la porte logique Ou
                '''
                pass # A COMPLETER

class Et(PorteLogiqueBinaire):
        '''
        Porte logique (binaire) Et
        '''
        def evaluer(self):
                '''
                Evaluation de la porte logique Et
                '''
                pass # A COMPLETER

class Diff(PorteLogiqueBinaire):
        '''
        Porte logique (binaire) Diff
        '''
        def evaluer(self):
                '''
                Evaluation de la porte logique Diff: retourne vrai ssi e1 et e2 sont differents
                '''
                pass # A COMPLETER

##########################################################################
```

FIGURE 2 –

```
##########################################################################

class CircuitCombinatoire(object):
        '''
        Representation generique des circuits combinatoire
        '''
        def __init__(self, entrees, sorties):
                '''
                Initialise un circuit combinatoire a partir d'entrees et de sorties
                '''
                try:
                        # les entrees et les sorties doivent etre iterables
                        iter(entrees)
                        iter(sorties)
                except TypeError:
                        raise
                self._entrees = tuple(entrees)
                self._sorties = tuple(sorties)

        def sortie(self, i):
                '''
                Retourne la i-eme sortie du circuit combinatoire
                '''
                return self._sorties[i].evaluer()

##########################################################################
```

FIGURE 3 –

```
################################################################################

from entreebinaire import Vrai, Faux
from portelogique import Non, Et, Ou, Diff
from circuitcombinatoire import CircuitCombinatoire

if __name__ == '__main__':

        print 'test #1'
        entree = Vrai()
        sortie = Non(entree)
        c = CircuitCombinatoire((entree,), (sortie,))
        print 'sortie:', c.sortie(0).evaluer()
        assert(c.sortie(0).evaluer() == False)

        print
        print 'test #2'
        entree = Faux()
        sortie = Non(entree)
        c = CircuitCombinatoire((entree,), (sortie,))
        print 'sortie:', c.sortie(0).evaluer()
        assert(c.sortie(0).evaluer() == True)

        print
        print 'test #3'
        entrees = (Vrai(), Vrai())
        c = CircuitCombinatoire(entrees,
                                (Ou(entrees[0], entrees[1]),
                                Et(entrees[0], entrees[1]),
                                Diff(entrees[0], entrees[1])))
        print 'sortie:', ', '.join([str(c.sortie(i).evaluer()) for i in range(3)])
        assert(c.sortie(0).evaluer() == True and
               c.sortie(1).evaluer() == True and
               c.sortie(2).evaluer() == False)

        print
        print 'test #4'
        entrees = (Vrai(), Faux())
        c = CircuitCombinatoire(entrees,
                                (Ou(entrees[0], entrees[1]),
                                Et(entrees[0], entrees[1]),
                                Diff(entrees[0], entrees[1])))
        print 'sortie:', ', '.join([str(c.sortie(i).evaluer()) for i in range(3)])
        assert(c.sortie(0).evaluer() == True and
               c.sortie(1).evaluer() == False and
               c.sortie(2).evaluer() == True)

        print
        print 'test #5'
        entrees = (Faux(), Vrai())
        c = CircuitCombinatoire(entrees,
                                (Ou(entrees[0], entrees[1]),
                                Et(entrees[0], entrees[1]),
                                Diff(entrees[0], entrees[1])))
        print 'sortie:', ', '.join([str(c.sortie(i).evaluer()) for i in range(3)])
        assert(c.sortie(0).evaluer() == True and
               c.sortie(1).evaluer() == False and
               c.sortie(2).evaluer() == True)

        print
        print 'test #6'
        entrees = (Faux(), Faux())
        c = CircuitCombinatoire(entrees,
                                (Ou(entrees[0], entrees[1]),
                                Et(entrees[0], entrees[1]),
                                Diff(entrees[0], entrees[1])))
        print 'sortie:', ', '.join([str(c.sortie(i).evaluer()) for i in range(3)])
        assert(c.sortie(0).evaluer() == False
               and c.sortie(1).evaluer() == False
               and c.sortie(2).evaluer() == False)

        print
        print 'test #7'
        V, F = (Vrai(), Faux())
        c = CircuitCombinatoire((V, F),
                                (Ou(V, Non(F)),
                                Et(Et(Ou(V, F), Non(Diff(V, Non(F)))), V)))
        print 'sortie:', ', '.join([str(c.sortie(i).evaluer()) for i in range(2)])
        assert(c.sortie(0).evaluer() == True and c.sortie(1).evaluer() == True)

################################################################################
```