

## Projet Info LDD2 – TD 11

**Objectifs du TD :** Réécriture : évaluation de circuits.



Les circuits booléens peuvent être utilisés pour réaliser un calcul (e.g. l'additionneur du TD précédent).


Petit rappel pour convertir binaire en décimal :

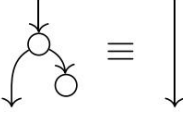
- `bin(n)[2:]` renvoie la représentation binaire de `n` sous la forme d'une chaîne de caractères, avec le poids fort à gauche
- `int(bitstring, 2)` renvoie l'entier décimal qui correspond à sa représentation binaire `bitstring`.

(on considérera dans ce TD les entiers non signés, c'est-à-dire que par exemple sur un octet (= 8 bits) on peut représenter tous les entiers de 0 à 255 inclus).

Pour pouvoir représenter les données dynamiquement dans nos circuits, nous allons rajouter des "primitives" i.e. des morceaux de graphe qui sont autorisés :

 qui correspond à un bit dans l'état 0, et  dans l'état 1 (on a déjà entraperçu 0 dans le TD 10).

Au passage, on rappelle que les "copies sans sorties"  sont autorisées, elles représentent en fait un effacement de données. Par ailleurs, sémantiquement,

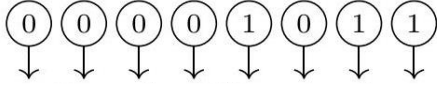
on a , i.e. si on copie un bit et qu'on efface une de ces copies, c'est équivalent à ne rien faire.

Exercice 1 :

Modifier la méthode `is_well_formed` pour accepter les nouvelles primitives (y compris le OU EXCLUSIF) si ça n'est déjà fait.

## Exercice 2 :

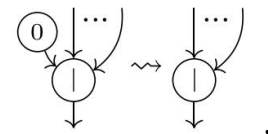
Implémenter une méthode de `bool_circ` qui étant donnés un entier et une taille de registre (i.e. le nombre de bits pour encoder l'entier) retourne un circuit booléen qui représente le registre instancié en l'entier. La taille du registre peut être donnée comme étant 8 par défaut.

Par exemple :  $11 \mapsto$  

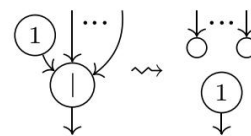
Comme on l'a déjà mentionné, différentes formules propositionnelles peuvent être équivalentes, le résultat sera le même pour toute évaluation des variables. Cela nous incite à faire de la simplification de circuits.

Dans ce TD, on s'intéresse au cas simple où toutes les variables sont instanciées. Il s'agit ici de transformer le circuit jusqu'à ce qu'il n'y ait plus de porte logique (elles auront toutes été évaluées).

Par exemple, on sait que  $(0 \mid P) \equiv P$  ce qui sera traduit par



Un peu plus subtil,  $(1 \mid P) \equiv 1$  sera traduit par



On donne dans la Table suivante toutes les transformations qui vont nous intéresser dans ce TD.






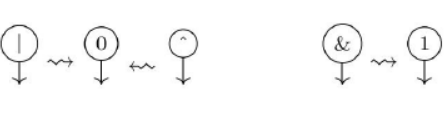
Copies	porte NON
	
porte ET	Porte OU
	
porte OU EXCLUSIF	éléments neutres
	

TABLE 1 – Les règles de transformation pour l'évaluation.

On note que les transformations données ci-dessus fonctionnent même lorsque "... " représente 0 arête.

Exercice 3 :

Dans `bool_circ`, pour chacune des règles de la Table, implémenter une méthode qui étant donnés les ids des noeuds concernés applique la transformation.

Exercice 4 :

Implémenter une méthode `evaluate(self)` qui va appliquer les règles précédentes tant qu'il y a des transformations à appliquer (i.e. ici tant qu'il y a des co-feuilles qui ne sont pas directement reliées à une sortie). Pour gagner en efficacité, on peut remarquer qu'il n'y a pas besoin de rechercher une co-feuille de zéro à chaque fois.

Exercice 5 :

Tester cette dernière méthode sur l'additionneur.