# Projet Info LDD2 – TD 5

Renaud Vilmart -- vilmart@lsv.fr

**Objectifs du TP :** Début des circuits booléens.

On va commencer par créer une nouvelle classe `bool_circ` qui sera une sous-classe de `open_digraph`, puisque les circuits booléens sont des cas particuliers de graphes dirigés ouverts.

⌜ **Exercice 1 :** Créer la sous-classe `bool_circ` d'`open_digraph`. Faire en sorte que si `g` est une instance d'`open_digraph`, alors `bool_circ(g)` crée un circuit booléen en utilisant `g`. On ne se préoccupe pas pour l'instant de savoir si le graphe `g` est un circuit booléen valable.
(Astuce : le pense-bête contient des infos utiles sous la section "classes".) ⌟

On va déterminer dans la suite à chaque noeud la porte logique grâce à son étiquette (`label`). On propose d'utiliser '`&`' pour la porte ET, '`|`' pour la porte OU, et '$\sim$' pour la porte NON ; et de laisser l'étiquette vide `''` pour le symbole de copie.

Par la suite, on pourra également se permettre d'avoir des noeuds étiquetés '0' et '1' pour représenter les constantes 0 et 1, ou encore `'^'` pour représenter le OU EXCLUSIF.

Pour être un circuit booléen valide, tous les noeuds "copie" doivent avoir exactement une entrée (i.e. doivent avoir un degré entrant égal à 1) ; chaque porte ET et OU doit avoir exactement une sortie ; chaque porte NON doit avoir exactement une entrée et une sortie.

Les portes ET et OU étant associatives (par exemple, dans le cas de &, $(x_0 \& x_1) \& x_2 = x_0 \& (x_1 \& x_2)$), elles peuvent avoir plus de 2 entrées sans que ça soit ambigu. On admet aussi qu'elles peuvent avoir 0 entrée (ça représente l'élément neutre de l'opération) ou 1 entrée (c'est l'identité).

⌜ **Exercice 2 :** Coder pour `node` les méthodes `indegree`, `outdegree` et `degree` qui calculent respectivement le degré entrant, sortant, et total d'un noeud. ⌟

Pour être un circuit booléen valide, le graphe doit également être acyclique.

⌜ **Exercice 3 :** Coder une méthode `is_cyclic` qui teste la cyclicité d'un graphe dirigé (dont un algo est donné dans les slides de la "séance 0"). ⌟

⌜ **Exercice 4** (tests requis) **:** Implémenter une méthode `is_well_formed` de `bool_circ` qui teste si le circuit booléen est bien un circuit booléen. Il doit être acyclique et respecter les contraintes de degré données ci-dessus. ⌟

⌜ **Exercice 5 :** Modifier la méthode `__init__` de `bool_circ` pour qu'elle teste si le graphe donné est bien un circuit booléen. ⌟

Dans la suite, on va avoir besoin, lorsqu'on nous donne une paire de graphes, de faire en sorte qu'il n'y ait pas de chevauchement entre les indices de l'un et

ceux de l'autre. Par exemple, si on a $i$ comme `id` d'un noeud du premier, on voudrait que $i$ ne soit pas `id` d'un noeud dans le second. Une méthode consiste simplement à translater les indices de l'un des deux graphes suffisamment pour qu'il n'y ait plus de chevauchement. Les deux exercices suivant sont à faire dans la classe `open_digraph`.

⌜ **Exercice 6 :** Implémenter les méthodes `min_id` et `max_id` qui renvoient respectivement l'indice min et l'indice max des noeuds du graphe. ⌟

⌜ **Exercice 7** (tests requis) **:** Implémenter dans `open_digraph` une méthode `shift_indices (self,n)`, qui va ajouter `n` à tous les indices du graphe (`n` pouvant être négatif). ⌟