# ESIR 1 / Algorithmique et Complexité : TP 3
# Reduced Ordered Binary Decision Diagram

Pierre Maurel, pierre.maurel@irisa.fr

Au cours du TD n°6 nous avons étudié les *Reduced Ordered Binary Decision Diagram* (ROBDD) qui permettent de fournir une *représentation compacte* d'une formule booléenne. L'objectif de ce TP est de créer un programme qui calcule le ROBDD associé à une expression booléenne et de comparer à l'utilisation d'un arbre de décision (ou, de manière équivalente, d'une table de vérité). Vous utiliserez alors ces méthodes pour résoudre le problème des $N$ reines.

Vous disposez pour la réalisation de ce TP de l'archive `ROBDD.zip` disponible sur l'ENT. Cette archive est composée de 14 fichiers Java répartis en 3 packages.

Une expression booléenne est une fonction de variables booléennes (appelées atomes) faisant intervenir des opérateurs booléens ($\wedge, \vee, \neg, \Rightarrow, \Leftrightarrow$) et éventuellement les constantes `VRAI` et `FAUX`. Une telle expression booléenne sera représentée par un objet de la classe `Expression`, définie dans le fichier `Expression.java`. Les méthodes abstraites suivantes y sont déclarées :

- `atomes()`. Elle renvoie la liste des atomes associés à l'objet courant. Attention, cette fonction renvoie un objet `Set` qui n'est donc pas ordonné.
- `evalue()`. Elle renvoie la valeur (booléenne) de l'objet courant. Si celui-ci ne peut pas être évalué (présence d'atomes), elle lève une exception.
- `remplace(s,b)`. Elle permet de remplacer un atome `s` de l'objet courant par un booléen `b`.
- `simplifier()`. Elle permet de simplifier l'objet courant selon les règles présentes dans le tableau de la partie 3.2 du TD 6.

Ces méthodes sont abstraites et doivent donc être implémentées pour chacune des classes filles de la classe `Expression`. Les classes filles de `Expression` définissent les opérateurs booléens $\wedge, \vee, \neg, \Rightarrow, \Leftrightarrow$ ainsi que les constantes booléennes (`Constante.java`) et les variables booléennes (`Atome.java`).

Dans la classe `Expression` est également fournie une méthode `arbre` qui renvoie l'arbre de Shannon correspondant à l'expression courante (`this`) et à l'ordre des atomes indiqué en paramètre.

## 1 Prise en main des expressions booléennes

**Exercice 1** En vous inspirant des classes `Et`, `Implique`, `Non` et `Ou`, complétez les méthodes à implémenter des classes `Atome` et `Equiv` (marquées par `//TODO`). Vous pourrez alors exécuter les quelques lignes déjà présentes dans le `main` de la classe `Main`.

**Exercice 2** Dans le `main` de la classe `Main`, créez l'expression booléenne

$$f(x_1, y_1, x_2, y_2) = (x_1 \Leftrightarrow y_1) \wedge (x_2 \Leftrightarrow y_2).$$

Testez les différentes fonctions de la classe `Expression`. Par exemple on pourra tenter de l'évaluer avec la méthode `evalue` après avoir remplacé chacun de ses atomes par des valeurs booléennes, ou afficher l'ensemble de ses atomes. Calculez et affichez l'arbre de cette expression pour les 2 ordres d'atomes vu en TD et vérifiez le résultat.

# 2 Construction du ROBDD

Vous allez maintenant construire le ROBDD associé à une expression booléenne. On va pour cela implémenter l'algorithme récursif vu dans la partie 3.2 du TD6.

**Exercice 3** Commencez par ajouter les deux sous-fonctions suivantes dont nous aurons besoin ensuite.

- une fonction `estVrai()` qui renvoie `true` si, et seulement si, l'expression courante est la constante VRAI et renvoie `false` sinon.
- une fonction `estFaux()` qui renvoie `true` si, et seulement si, l'expression courante est la constante FAUX et renvoie `false` sinon.

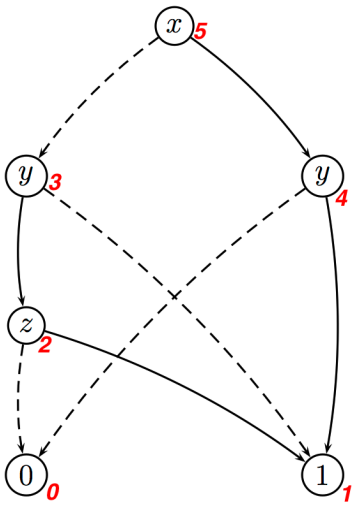
Exemple :

```
Expression exp = new Constante(true);
System.out.println(exp.estVrai());   //affiche true
exp = new Non(new Constante(false));
System.out.println(exp.estVrai());   //affiche false
```

Le deuxième exemple renvoie `false` car `exp` n'est pas égale à la constante VRAI.

La classe `Noeud_ROBDD.java` va représenter les noeuds des ROBDD. Un noeud ROBDD est défini par un nom (correspondant à la variable qu'il représente), un index (de type entier) et les index de ses fils gauche et droit. Le noeud correspondant à la feuille true est indicé par 1, et celui correspondant à la feuille false est indicé par 0. Les autres noeuds du ROBDD auront donc un indice strictement supérieur à 1. Un diagramme complet associé à une expression booléenne est alors représenté par une instance de la classe `ROBDD` contenant une liste de `Noeud_ROBDD`.
Ainsi le `ROBDD` contenant la liste suivante :

```
[2:bdd(z,0,1), 3:bdd(y,1,2), 4:bdd(y,0,1),
               5:bdd(x,3,4)]
```




représente ce ROBDD (en ***rouge*** : les index des noeuds)

**Exercice 4** Dans l'algorithme de construction du ROBDD, on devra vérifier, lorsque l'on obtient un nouveau noeud, s'il existe déjà ou non dans le diagramme courant.

Complétez `obtenirROBDDIndex(String nom, int fg, int fd)` de la classe `ROBDD` qui doit renvoyer l'index, dans le ROBDD courant, du noeud ROBDD associé à la variable `nom` et dont les fils droit et gauche sont `fd` et `fg`. Si ce noeud n'existe pas dans le diagramme, la fonction doit renvoyer -1.

**Exercice 5**

- Complétez la fonction `construireROBDD(ROBDD G, List<String> atomes_ordonnes)` de la classe `Expression` pour implémenter l'algorithme récursif vu dans la partie 3.2 du TD6.
- Vérifiez le bon fonctionnement de votre fonction sur l'expression de l'exercice 2 de ce TP ainsi que sur celle de l'exercice 10 du TD. Vous pouvez afficher la liste des noeuds constituant le diagramme ROBDD en utilisant le `toString` des objets `ROBDD` puis tracer à la main le ROBDD sur une feuille.

**Exercice 6**

(a) Dans `ROBDD.java`, ajoutez une fonction `trouve_sat` :

```
public String trouve_sat()
```

qui renvoie :

- une chaîne de caractère indiquant quelle valeur donner à quelle variable pour que la formule représentée par le `ROBDD` courant soit satisfaite, si elle est satisfaisable. *Remarque :* Cette fonction renverra n'importe quelle solution (i.e. n'importe quel jeu de paramètres satisfaisant la formule).
- un message indiquant que l'expression associée au ROBDD courant n'est pas satisfaisable le cas échéant

(b) Testez cette fonction, par exemple sur l'expression définie à l'exercice 2.

# 3 Application : le problème des $N$ reines

Le but du problème des $N$ dames est de déterminer s'il est possible de placer $N$ dames d'un jeu d'échecs sur un échiquier de $N \times N$ cases sans que les dames ne puissent se menacer mutuellement, conformément aux règles du jeu d'échecs (la couleur des pièces étant ignorée). Par conséquent, deux dames ne devraient jamais partager la même rangée, colonne, ou diagonale.

**Exercice 7** Comment peut-on modéliser ce problème sous la forme d'un problème de satisfaisabilité d'une expression booléenne (à $N^2$ variables) ? Comment peut-on se servir d'un constructeur de ROBDD pour déterminer s'il existe une solution au problème ?

**Exercice 8**

(a) Créez l'expression booléenne correspondant au problème des $N$ reines et servez vous des fonctions précédemment implémentées pour chercher le premier $N$ tel qu'il existe une solution. Affichez une de ces solutions. Vous pouvez dans un premier temps commencer par considérer le cas de tours à la place des reines (i.e. ne pas tenir compte des diagonales)

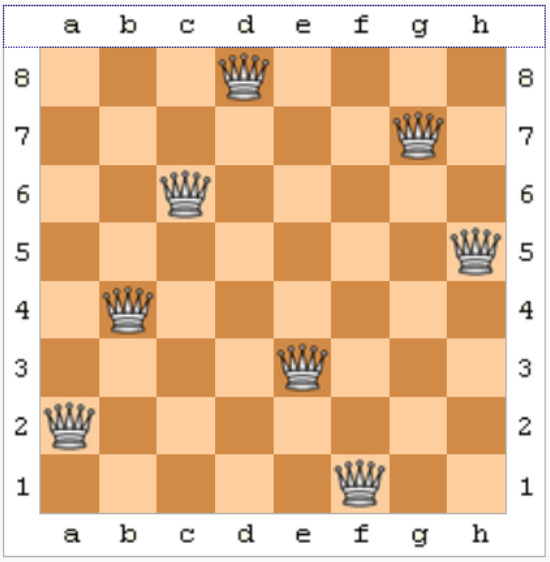



FIGURE 1 – Exemple de solution pour n=8 (tiré de wikipedia)

(b) Comparez la taille de l'arbre de Shannon et celle du ROBDD correspondant, ainsi que le temps de calcul nécessaire à la construction de chacun des deux.

**Exercice 9** Dans `ROBDD`, ajoutez une fonction `void reines_affiche_sat(int n)` similaire à la fonction `trouve_sat` de l'exercice 6 mais qui affiche une représentation graphique (en mode texte dans un premier temps) de la solution trouvée. Vous trouverez ci-dessous un exemple de ce que peut afficher cette fonction :

```
 _____ _____ _____ _____ _____ _____ _____ _____
|     |     |     |  X  |     |     |     |     |
|_____|_____|_____|_____|_____|_____|_____|_____|
|     |     |     |     |     |     |  X  |     |
|_____|_____|_____|_____|_____|_____|_____|_____|
|     |     |  X  |     |     |     |     |     |
|_____|_____|_____|_____|_____|_____|_____|_____|
|     |     |     |     |     |     |     |  X  |
|_____|_____|_____|_____|_____|_____|_____|_____|
|     |  X  |     |     |     |     |     |     |
|_____|_____|_____|_____|_____|_____|_____|_____|
|     |     |     |     |  X  |     |     |     |
|_____|_____|_____|_____|_____|_____|_____|_____|
|  X  |     |     |     |     |     |     |     |
|_____|_____|_____|_____|_____|_____|_____|_____|
|     |     |     |     |     |  X  |     |     |
|_____|_____|_____|_____|_____|_____|_____|_____|
```

*Indication :* On pourra supposer que le ROBDD a été construit en utilisant un ordre d'atomes particulier et donc choisir cet ordre de manière à faciliter l'affichage.