# OPC – Résolution

## 1 Arbre de Recherche

1. Dessiner l'arbre de recherche obtenu en exécutant l'algorithme MAC sur le problème suivant. On choisira les variables ainsi que les valeurs dans l'ordre lexicographique.

```
class Tree extends Problem {
  void model() {
    Var w = var("w", dom(1, 4));
    Var x = var("x", dom(1, 4));
    Var y = var("y", dom(1, 4));
    Var z = var("z", dom(1, 4));

    allDifferent(w, x, y, z);
    ctr(gt(w, z));
    ctr(le(x, y));
  }
}
```

2. On souhaite utiliser AbsCon pour résoudre l'instance et visualiser sa trace. Le plus simple est de procéder comme suit avec Eclipse.
   (a) Téléchargez le fichier `AbsCon.jar` depuis http://www.cril.fr/~lecoutre – onglet Enseignement.
   (b) Utiliser un ordinateur équipé avec Java SE 8 et Eclipse
   (c) Lancer et paramétrer Eclipse
   — Si c'est la première fois qu'Eclipse est exécuté, accepter le répertoire de travail par défaut proposé, et ensuite "Go to Workbench"
   — Créer un projet Java de nom `abscon` :
   — File - New - Java Project
   — Taper le nom du projet (ici, `abscon`), puis choisir Next
   — Dans l'onglet Libraires, choisir "add External Jars" , puis sélectionner abscon.jar
   (d) Créer une classe Java de nom `Tree.java` dans le répertoire `src` du projet, avec le code donné ci-dessus.
   (e) Au niveau de la configuration, on utilisera `abscon.Resolution` comme Main Class, et `Tree -trace` comme Program Arguments, comme suit :

```
Menu Run
Sous-menu Run configurations...
Onglet Main
  Project: abscon
  Main class: abscon.Resolution
Onglet Arguments
  Program arguments: Test -s=all
  VM arguments:
```

puis Run

## 2 Le problème des reines

1. Le code ci-dessous est à placer dans un fichier de nom `Queens.java`.

```
class Queens extends Problem {
  int n = askInt("Number of queens");
  int model = askInt("Model (1 to 3)", bounds(1, 3));

  void model() {
    Var[] q = array("q", dom(0, n - 1), n);
    if (model == 1) {
      group(range(n), range(n), (i, j) ->
        i < j ? ctr(ne(q[i], q[j])) : null);
      group(range(n), range(n), (i, j) ->
        i < j ? ctr(ne(Math.abs(i - j), dist(q[i], q[j]))) : null);
    }
  }
}
```

   Il est important de noter que pour définir des contraintes en intension, on peut construire des expressions/formules booléennes (prédicats) à l'aide de constantes (entières et booléennes) et des fonctions (opérateurs) décrits sur cette page, en sélectionnant `Constraint - Generic - intension`.

2. Lancer le solveur en mode interactif (il demandera la valeur des paramètres) avec :

   ```
   java abscon.Resolution Queens
   ```

   Sinon on peut directement les donner en ligne de commande

   ```
   java abscon.Resolution Queens 8 1
   ```

3. Noter le temps et le nombre de noeuds nécessaire pour trouver toutes les solutions de l'instance des 12 reines (avec le modèle 1) :

   ```
   java abscon.Resolution Queens 12 1 -s=all -v=0
   ```

4. Intégrer une nouvelle formulation des contraintes (`model = 2`) où les deux jeux de contraintes sont fusionnés (i.e. les contraintes de même portée sont fusionnées). Tester le nouveau modèle et comparer le temps de résolution ainsi que le nombre de noeuds explorés (pour toutes les solutions de l'instance des 12 reines).
5. Pour le modèle 3, ajouter simplement une contrainte globale redondante (utilisez la méthode `allDifferent` au modèle précédent.
6. Comparer les 3 modèles en terme de temps d'exécution et du nombre de noeuds explorés lorsque les 14 200 solutions au problème des 12 reines sont recherchées.

## 3 Le problème Sudoku

Intégrer le code suivant pour le problème du Sudoku (défini ici pour une grille vierge) :

```
class Sudoku extends Problem {
  int order = askInt("order (9 = classical Sudoku)");
  int base = (int) Math.sqrt(order);

  void model() {
    Var[][] x = array("x", dom(1, order), order, order);
    Var[][] xT = Var.transpose(x);

    group(range(order), i -> allDifferent(x[i])); // rows
    group(range(order), i -> allDifferent(xT[i])); // cols
    group(range(0, order, base), range(0, order, base), (i, j)
      -> allDifferent(Var.pieceOf(x, i, base, j, base))); // blocks
  }
}
```

1. Construire une nouvelle classe Sudoku3 pour modéliser l'instance Sudoku suivante :

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 4 | | | | | | | |
| 5 | 3 | 9 | | | 1 | | 6 | |
| | | 1 | | | 2 | | 5 | |
| 4 | | 7 | 2 | | 9 | | | 6 |
| | | 6 | | | | 5 | | |
| 8 | | | 6 | | 3 | 1 | | 7 |
| | 8 | | 7 | | | 2 | | |
| | 6 | | 3 | | | 4 | 1 | 8 |
| | | | | | | | 7 | |

   On pourra utiliser un tableau comme suit :

   private int[][] hints = 0, 1, 4 , 1, 0, 5 , 1, 1, 3 , 1, 2, 9 , 1, 5, 1 , 1, 7, 6 , 2, 2, 1 , 2, 5, 2 , 2, 7, 5 , 3, 0, 4 , 3, 2, 7 , 3, 3, 2 , 3, 5, 9 , 3, 8, 6 , 4, 2, 6 , 4, 6, 5 , 5, 0, 8 , 5, 3, 6 , 5, 5, 3 , 5, 6, 1 , 5, 8, 7 , 6, 1, 8 , 6, 4, 7 , 6, 6, 2 , 7, 1, 6 , 7, 4, 3 , 7, 6, 4 , 7, 7, 1 , 8, 7, 7 ; // triplets of the form (row,column,value)

2. La résoudre avec la commande :

   ```
   java abscon.Resolution Sudoku3
   ```

   Noter le nombre (et le type) de contraintes du problème ainsi que le nombre de valeurs détruites au préprocessing.

3. La résoudre avec la commande :

   ```
   java abscon.Resolution Sudoku3 -ugc=no
   ```

   Noter le nombre (et le type) de contraintes du problème ainsi que le nombre de valeurs détruites au préprocessing. Pourquoi est-ce différent ?

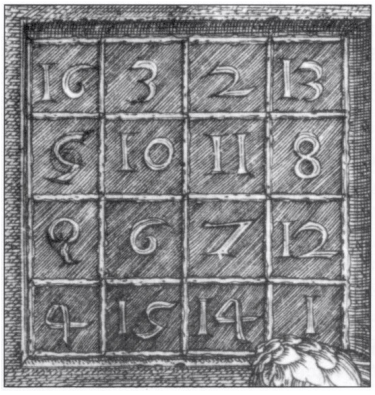
4. Ajouter 8 dans la case en position $(7, 8)$. Quelle est la conséquence ?

## 4 Séries de tous les intervalles

Donner le modèle de ce problème sous AbsCon.

## 5 Les Carrés magiques

Un carré magique d'ordre $n$ est un arrangement de $n^2$ nombres dans un carré tel que la somme des valeurs sur chaque ligne, chaque colonne et chaque diagonale (principale) est égale à une même constante. Un carré magique normal contient les entiers de 1 à $n^2$. Par exemple, voici une solution pour $n = 4$.



1. Donner le modèle de ce problème sous AbsCon Pour cela,
   — penser à intégrer un paramètre de type `int` pour définir l'ordre du carré (à l'aide de la méthode `askInt`)
   — penser à calculer à l'aide d'une formule (simple) la valeur à atteindre lorsqu'on somme chaque ligne, colonne, et diagonale.
   — poster les contraintes de somme en appelant la méthode `sum`
2. Lancer la résolution pour quelques instances (ordres) du carré magique avec et sans le paramètre `-ugc=no`
3. Lancer la résolution pour quelques instances (ordres) du carré magique avec et sans le paramètre `-rc=10`

# 6 Le tour des cavaliers

Le problème du tour du cavalier est de trouver une séquence de mouvements de cavalier (un cavalier peut bouger de 2 cases verticalement et d'une case horizontalement, ou de 2 cases horizontalement et d'une case verticalement) sur un plateau de $n$ sur $n$ cases. Il est nécessaire que :
— chaque case soit visitée une et une seule fois
— le dernier mouvement permet de revenir sur la case de départ

La figure suivante montrer une solution de tour pour $n = 8$, avec un tour commençant dans le coin supérieur gauche.

Créer une classe permettant de modéliser ce problème. Le résoudre à l'aide d'AbsCon.