IFT 615 : Devoir 2
Travail individuel

Remise : 11 juin 2014, 9h00 (**au plus tard**)

Ce devoir comporte 2 questions : 1 question théorique et 1 question de programmation. Vous trouverez tous les fichiers nécessaires pour ce devoir ici : `http://marccote.github.io/courses/ift615/devoir_2/devoir_2.zip`.

Veuillez remettre votre réponse à la question théorique sous forme papier et remettre votre fichier `solution_sudoku.py` à l'aide de l'outil **turnin** :

```
turnin -c ift615 -p devoir_2 solution_sudoku.py
```

1. [**5 points**] Soit l'énoncé des faits suivant :

   — Marie est une actrice.
   — Jean est un acteur.
   — Jean a joué dans un film et y a tenu le rôle d'un zombie.
   — Charles est directeur de film.
   — Tous les films que Charles dirige ont des créatures mythiques.
   — Marie a joué dans tous les films dans lesquels Jean a joué.
   — Tous les films qui contiennent des créatures mythiques sont des films d'horreur.
   — Un rôle de zombie est un rôle de créature mythique.
   — On est acteur lorsqu'on a joué dans un film.

   On aimerait savoir si Marie a déjà joué dans un film d'horreur.

   (a) Donnez l'ensemble des formules représentant tous les énoncés dans cette situation, sous forme de logique du premier ordre. Utilisez les prédicats suivants :

   — $Acteur(x)$: x est un acteur.
   — $Directeur(x)$: x est directeur de film.
   — $Dirige(x, f)$: x dirige le film f.
   — $Joue(x, f)$: x joue dans le film f.
   — $Role(r, f)$: r est un rôle dans le film f.
   — $JoueRole(x, r)$: x joue le rôle r.
   — $Zombie(r)$: r est un rôle de zombie.
   — $Creature(r)$: r est un rôle de créature mythique.
   — $Horreur(f)$: f est un film d'horreur.

   Veuillez également identifier clairement quelles sont les constantes et les variables utilisées dans vos formules.

   (b) Convertissez les formules de la question (a) sous forme normale conjonctive. Numérotez chaque clause.

   (c) Utilisez la preuve par résolution afin de démontrer que Marie a joué dans un film d'horreur. Vous devez faire explicitement référence aux numéros de clause en (b) lors de l'application de la règle de résolution et spécifier l'UPG utilisé.

2. [**5 points**] Programmez l'algorithme *backtracking-search* avec l'algorithme AC-3 comme technique d'inférence pour résoudre une grille de jeu Sudoku : `http://fr.wikipedia.org/wiki/Sudoku`.

   Le programme doit être écrit dans le langage Python. Plus spécifiquement, vous devez remettre un fichier `solution_sudoku.py` contenant les **5 fonctions** suivantes :

   — `backtracking_search(csp)` : Fonction prenant en entrée un objet `csp` de la classe `CSP` (voir ci-dessous pour une description de la classe `CSP`) et qui retourne un dictionnaire où une clé est une paire d'entiers (position (y, x) d'une case dans la grille du Sudoku, où y est le numéro de rangée et x est le numéro de colonne) et la valeur associée est un chiffre entre 1 et 9 sous forme de chaîne de caractères ou `str` (valeur à insérer à cette position (y, x) de la grille). Ce dictionnaire doit correspondre à une assignation complète de toutes les cases de la grille.
   — `backtrack(assignations, csp)` : Fonction qui a comme argument un dictionnaire `assignations` de clés qui sont des paires (variable) et de valeurs qui sont des valeurs (`str` d'un chiffre entre 1 et 9) et un objet `csp` de la classe `CSP`. Cette fonction correspond à la fonction `backtrack` dans les diapositives du cours. Vous devez utiliser l'algorithme AC-3 pour l'inférence. Également, vous devez choisir la première variable `X` non-assignée selon l'ordre des variables dans `csp.variables` et vous devez itérer sur les valeurs du domaine `csp.domaines[X]` dans l'ordre.
   — `est_compatible(X, v, assignations, csp)` : Fonction qui a comme argument une variable `X` (une paire d'entiers correspondant à une position dans la grille), une valeur `v` (un chiffre entre 1 et 9 sous forme de `str`), l'assignation courante `assignations` des variables (dictionnaire paire/`str`) et un objet `csp` de la classe `CSP`. La fonction doit retourner True si assigner la valeur `v` à la variable `X` est compatible (légal) ou pas.
   — `AC3(csp)` : Fonction qui prend en entrée un objet `csp` de la classe `CSP` et qui retourne une paire contenant ce même `csp` mais dont les domaines ont été réduits par l'algorithme AC-3 et un booléen indiquant si aucun conflit n'a été détecté (donc True s'il n'y a pas de conflit, False sinon). Voir le pseudocode de AC-3 dans les diapositives.
   — `reviser(Xi, Xj, csp)` : Fonction qui prend comme argument deux variables `Xi` et `Xj` (des paires d'entiers) et un objet `csp` de la classe `CSP`. Cette fonction doit retourner une paire contenant un booléen indiquant si le domaine de `Xi` a été réduit et l'objet `csp` dans lequel le domaine de la variable `Xi` a été réduit. Pour plus de détails, voir la diapositive du pseudocode d'AC-3.

   Un fichier initial `solution_sudoku.py` vous est fourni et contient la signature de toutes ces fonctions.

   Le fichier `sudoku.py` contient également la définition de la classe `CSP`, qui correspond à la spécification du problème de satisfaction de containtes. Spécifiquement, un objet de la classe `CSP` contient :

   — `variables` : La liste des variables du problème. Pour le Sudoku, cette liste contient l'ensemble des paires d'entiers (une paire de deux entiers entre 0 et 8) correspondant à toutes les positions (y, x) de la grille 9×9.
   — `domaines` : Un dictionnaire qui associe à chaque variable son ensemble de valeurs possibles. Spécifiquement, les clés du dictionnaire sont des paires d'entiers et la valeur associée est une liste contenant toutes les valeurs possibles pour cette case dans la grille. Pour la configuration initiale du Sudoku, le domaine de plusieurs cases ne contiendra qu'une seule valeur : ce sont les cases pré-spécifiées du Sudoku (mais techniquement pas encore assignée !). Les autres cases ont comme domaine l'ensemble des chiffres de 1 à 9 (`str`), dans l'ordre. À noter qu'un élément `X` peut être enlevé d'une liste à l'aide de sa méthode `remove(X)`. Vous pouvez donc manipuler directement le champs `domaines` lorsque vous reduisez les domaines d'un objet `CSP`.
   — `contraintes` : Un dictionnaire qui associe à chaque variable l'ensemble des autres variables du problème ayant une contrainte avec celle-ci. Spécifiquement, pour une variable `X` donnée, toutes les variables contenues dans `contraintes[X]` doivent être assignées à une valeur différente de celle de `X`.
   — `arcs()` : Méthode qui retourne la liste des arcs associés aux contraintes du problème. Chaque arc d'une variable `Xi` vers une variable `Xj` est représenté par une paire `(Xi,Xj)`.
   — `copy()` : Méthode qui retourne une copie de l'objet de la classe `CSP`. **Cette méthode doit être utilisée pour la ligne 6 du pseudocode de** *backtracking search***, qui crée un nouvel objet** csp^* **puis change le domaine de la variable en considération** X.

Le script Python `sudoku.py` importera la fonction `backtracking_search` contenue dans `solution_sudoku.py` (qui doit être dans le même répertoire) et l'utilisera afin de résoudre une grille de Sudoku.

Voici comment utiliser ce script :

```
Usage: python sudoku.py [-joueur JOUEUR] [-no_partie INT] [-valider FICHIER]

où -joueur    : "humain" ou le fichier contenant votre solution.
                Défaut = "solution_sudoku.py"
   -no_partie : un entier entre [0-3]
                -> 0: Très débutant
                -> 1: Débutant
                -> 2: Intermédiaire
                -> 3: Très difficile
                -> Défaut: Difficile + Évaluation avec "sudoku_validation.pkl"
   -valider   : un fichier permettant de valider votre joueur pour un jeu donné.
                Défaut = "sudoku_validation.pkl"
```

Lorsque `no_partie` n'est pas spécifié, une grille par défaut est utilisée et l'exécution de votre algorithme sera comparée avec notre solution, en comparant le nombre d'ajouts d'assignations et de retour en arrière (*backtrack*) ayant été fait. Afin d'obtenir les mêmes résultats, vous devez vous assurer que vous sélectionnez les variables dans `csp.variables` et les valeurs de domaine dans `csp.domaines[X]` selon l'ordre dans lequel ils apparaissent.

La procédure d'évaluation automatique utilisera également directement vos fonctions `backtrack(assignations, csp)`, `est_compatible(X, v, assignations, csp)`, `AC3(csp)` et `reviser(Xi, Xj, csp)`, ainsi assurez-vous de les utiliser et de bien les implémenter.