[IA02] TP feuille 3 – Résolution de Sudoku en SAT

Information	Valeur
Auteur	Sylvain Lagrue (sylvain.lagrue@utc.fr (mailto:sylvain.lagrue@utc.fr))
Licence	Creative Common CC BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0)
Version document	1.0.5

I. Présentation du sujet

L'objectif de ce TP est de résoudre n'importe quel Sudoku à l'aide d'un solver SAT, via Python. Une description complète de la réécriture d'un Sudoku est accessible sur le site de Gophersat : https://github.com/crillab/gophersat/blob/master/examples/sat-for-noobs.md#sudoku (https://github.com/crillab/gophersat/blob/master/examples/sat-for-noobs.md#sudoku)

Un squelette de programme attendu, avec des fonctions prédéfinies, est accessible sur le cours Moodle.

Questions préliminaires indispensables à la suite…

1. Combien de variables propositionnelles sont nécessaires ?
2. Étant donné un ensemble de variables propositionnelles, que signifient les contraintes `at_least_one` et `unique` sur cet ensemble ? Quel rapport avec un `XOR` ? Comment les écrit-on avec des clauses ?
3. Comment modélise-t-on les règles génériques du Sudoku en logique propositionnelle à l'aide de ces contraintes ?
4. Comment peut-on intégrer les valeurs d'une grille donnée à la précédente modélisation ?

Choix de représentation des données

Les grilles de Sudokus seront représentées sous forme de listes de liste d'entiers (des matrices numpy sont également possibles).

```
# Exemple de grille de Sudoku
grid_example = [
    [5, 3, 0, 0, 7, 0, 0, 0, 0],
    [6, 0, 0, 1, 9, 5, 0, 0, 0],
    [0, 9, 8, 0, 0, 0, 0, 6, 0],
    [8, 0, 0, 0, 6, 0, 0, 0, 3],
    [4, 0, 0, 8, 0, 3, 0, 0, 1],
    [7, 0, 0, 0, 2, 0, 0, 0, 6],
    [0, 6, 0, 0, 0, 0, 2, 8, 0],
    [0, 0, 0, 4, 1, 9, 0, 0, 5],
    [0, 0, 0, 0, 8, 0, 0, 7, 9],
]
```

Les variables propositionnelles seront représentées par des entiers strictement postifs. Un littéral est un entier positif ou négatif. Une clause est une liste de littéraux. Une base de clauses sera vue comme une liste de listes d'entier. Un modèle sera vu comme une liste de clauses.

Pour résumer :

```
# aliases de type
Grid = List[List[int]]
Propositionnal_Variable = int
Literal = int
Clause = List[Literal]
Clause_Base = List[Clause]
Model = List[Literal]
```

Quelques outils/packages Python

- La fonction `pprint` du package du même nom permet un affichage lisible des tableaux à 2 dimensions
- L'opérateur `+` associé à 2 listes renvoie la concaténation de ces 2 listes.

```
>>> [1, 2, 3] + [4, 5, 6]
[1, 2, 3, 4, 5, 6]
```

- L'utilisation de `black` et de `mypy` est plus que jamais d'actualité…

II. Création des fonctions utilitaires de conversion

Écrire la fonction `cell_to_variable(i: int, j: int, val: int) -> int` qui, étant données les coordonnées d'une cellule (entre 0 et 9 exclu) et une valeur (entre 1 et 9 inclus) renvoie le numéro de variable correspondant.

```
>>> cell_to_variable(1, 3, 4)
112
```

Écrire la fonction `variable_to_cell(var: int) -> Tuple[int, int, int]`, inverse de la précédente, qui étant donné un numéro de variable renvoie le triplet ligne/colonne/valeur associé.

```
>>> variable_to_cell(1)
(0, 0, 1)
>>> variable_to_cell(112)
(1, 3, 4)
>>> variable_to_cell(729)
(8, 8, 9)
```

III. Génération du problème

Création des fonctions utilitaires de contraintes génériques

At least one

Écrire la fonction `at_least_one(vars: List[int]) -> List[int]` qui, étant donné une liste de variables propositionnelles, renvoie la clause `at least one` associée. Attention à la nature mutable des listes et à leur passage par référence !

```
>>> at_least_one([1, 3, 5])
[1, 3, 5]
```

Unique

Écrire la fonction `unique(vars: List[int]) -> List[List[int]]` qui, étant donné une liste de variables propositionnelles, renvoie la base de clauses `unique` associée. On utilisera pour cela la fonction `combinations` du package `itertools`.

```
>>> unique([1, 3, 5])
[[1, 3, 5], [-1, -3], [-1, -5], [-3, -5]]
```

Création des contraintes liées aux règles Sudoku

Écrire la fonction `create_cell_constraints() -> List[List[int]]` renvoyant une base de clauses représentant la contrainte d'unicité de valeur pour toutes les cases du Sudoku.

Écrire les fonctions suivantes renvoyant une base de clauses représentant représentant réciproquement les contraintes sur les lignes, colonnes et carrés du Sudoku.

- `create_line_constraints() -> List[List[int]]`
- `create_column_constraints() -> List[List[int]]`
- `create_box_constraints() -> List[List[int]]`

Création des règles spécifiques à une grille de Sudoku donné

Écrire la fonction `create_value_constraints(grid: List[List[int]]) -> List[List[int]]` qui, étant donné une grille de Sudoku, retourne une base de clauses représentant les contraintes induites par les valeurs inscrites dans la grille.

Génération globale du problème

Écrire la fonction `generate_problem(grid: List[List[int]] -> List[List[int]]` qui, étant donné une grille de Sudoku, renvoie la base de clause associée à l'ensemble des contraintes associées à cette grille.

IV. Appel au solver et affichage de la solution

1. Écrire une fonction `clauses_to_dimacs(clauses: List[List[int]], nb_vars: int) -> str` qui, étant donné une base de clauses et le nombre de variables à considérer, renvoie une chaîne de caractères codant la base de clauses au format Dimacs.

```
>>> clauses_to_dimacs([[-1, -2], [1, 2], [1, 3], [2, 4], [-3, 4], [-4, 5]],5)
'p cnf 5 6\n-1 -2 0\n1 2 0\n1 3 0\n2 4 0\n-3 4 0\n-4 5 0\n'
```

2. En utilisant la fonction précédente et la fonction fournie `write_dimacs_file(dimacs: str, filename: str)`, créer sur le disque le fichier `sudoku.cnf` encodant le problème.

3. Résoudre le problème avec Gophersat. On utilisera pour cela la fonction fournie `exec_gophersat(filename: str, cmd: str = "gophersat", encoding: str = "utf8") -> Tuple[bool, List[int]]`.

```
>>> exec_gophersat("licorne.cnf")
(True, [-1, 2, 3, 4, 5, 0])

# en cas de problème de chemin de l'exécutable...
>>> exec_gophersat("licorne.cnf", "/Users/sylvain/go/bin/gophersat")
(True, [-1, 2, 3, 4, 5, 0])
```

4. Écrire la fonction `model_to_grid(model: List[int], nb_vals: int = 9) -> List[List]` qui, étant donné un modèle renvoyé par Gophersat, renvoie un tableau 2D représentant la grille de Sudoku complétée.

5. Écrire la fonction principale permettant de résoudre entièrement les exemples de Sudoku.

V. Vous avez fini ?

Affichage *user friendly*

Écrire des fonctions permettant des sorties plus agréables au format ASCII, markdown ou html. Exemple de sortie possible :

```
Problème initial
-------------------------
| 5 3 . | . 7 . | . . . |
| 6 . . | 1 9 5 | . . . |
| . 9 8 | . . . | . 6 . |
-------------------------
| 8 . . | . 6 . | . . 3 |
| 4 . . | 8 . 3 | . . 1 |
| 7 . . | . 2 . | . . 6 |
-------------------------
| . 6 . | . . . | 2 8 . |
| . . . | 4 1 9 | . . 5 |
| . . . | . 8 . | . 7 9 |
-------------------------
```

```
Problème résolu
-------------------------
| 5 3 4 | 6 7 8 | 9 1 2 |
| 6 7 2 | 1 9 5 | 3 4 8 |
| 1 9 8 | 3 4 2 | 5 6 7 |
-------------------------
| 8 5 9 | 7 6 1 | 4 2 3 |
| 4 2 6 | 8 5 3 | 7 9 1 |
| 7 1 3 | 9 2 4 | 8 5 6 |
-------------------------
| 9 6 1 | 5 3 7 | 2 8 4 |
| 2 8 7 | 4 1 9 | 6 3 5 |
| 3 4 5 | 2 8 6 | 1 7 9 |
-------------------------
```

Questions subsidiaires

- Que veut dire un résultat UNSAT pour une grille ?
- Comment vérifier la règle *une grille de Sudoku n'a qu'une seule solution possible* ?
- Implémenter ces fonctions de vérification.