Rapport de Projet Problèmes Logiques : Sudoku

Auteurs :
KHADEMLOU Kimia
HAJMOHAMMADKASHI Shaghayegh
Groupe MIN3
année 2023-2024

Avant de Commencer :
Assurez-vous d'avoir installé la bibliothèque z3-solver.
Utilisation du code avec la commande :
python main.py <nom_du_puzzle>

1. Introduction

Le projet consiste à résoudre automatiquement des puzzles Sudoku en utilisant un SAT-solveur. Le Sudoku est un jeu de logique où l'objectif est de remplir une grille de $n^2 * n^2$ avec des chiffres de 1 à n^2 selon des règles spécifiques :

- Chaque case doit contenir un chiffre de 1 à n^2, sans répétition.
- Chaque ligne doit avoir chaque chiffre de 1 à n^2 une seule fois.
- Chaque colonne doit également avoir chaque chiffre de 1 à n^2 une seule fois.
- Chaque bloc 3x3 (ou sous-grille) doit contenir chaque chiffre de 1 à 9 sans répétition.
- Toute valeur donnée dans la grille initiale doit rester constante dans la solution finale.

Pour résoudre ces contraintes logiques, ce projet utilise le SAT-solveur Z3 qui prend un ensemble de formules logiques et tente de déterminer s'il existe une affectation qui rend ces formules vraies. Le but de ce projet est de modéliser les règles du Sudoku sous forme logique, puis de laisser Z3 déterminer une solution satisfaisante.

2. Conversion du Sudoku en CNF et Traduction en Logique

Pour convertir un problème de Sudoku en forme normale conjonctive (CNF), nous devons d'abord comprendre comment représenter un puzzle Sudoku uniquement avec des valeurs booléennes.

Prenons n comme étant la taille d'un puzzle Sudoku, correspondant à la longueur de chaque sous-grille. Pour un Sudoku 9x9 classique, n = 3.

Le puzzle Sudoku a n^2 lignes, n^2 colonnes, et n^2 valeurs possibles pour chaque entrée (combinaison ligne-colonne). Pour représenter ces combinaisons, nous créons des littéraux booléens. Nous utilisons la notation « sabc » où « a » représente le numéro de ligne, « b » le numéro de colonne, et « c » la valeur du 1 au 9. Par exemple, s123 indique que le chiffre 3 se trouve dans la ligne 1, colonne 2.

Maintenant, nous définissons des clauses qui imposent les contraintes du Sudoku pour garantir la validité du puzzle. Voici les étapes de la conversion :

2.1 Remplissage des Cases
Chaque case doit avoir au moins un chiffre pour que le puzzle soit entièrement rempli. Pour chaque combinaison (a, b), nous devons avoir au moins une variable sabc vraie, ce qui se traduit par une clause CNF :

$s\_ab1 \vee s\_ab2 \vee ... \vee s\_ab9$

2.2 Unicité dans les Lignes
Chaque chiffre doit apparaître au plus une fois dans chaque ligne. Pour garantir cela, nous créons des clauses qui empêchent que le même chiffre apparaisse plus d'une fois dans la même ligne. Par exemple, si la valeur « c » apparaît deux fois dans la ligne « a » une clause doit invalider cette combinaison :
$\neg(s\_ab1c \wedge s\_ab2c)$

où b1 et b2 sont des colonnes différentes.

2.3 Unicité dans les Colonnes
Le même concept s'applique aux colonnes. Chaque chiffre ne doit apparaître qu'une fois dans chaque colonne. Pour cela, nous créons des clauses qui assurent que le même chiffre n'apparaît pas plus d'une fois dans une colonne :

$\neg(s\_a1bc \wedge s\_a2bc)$

où a1 et a2 sont des lignes différentes.

2.4 Unicité dans les Sous-grilles
Pour garantir qu'un chiffre n'apparaît qu'une seule fois par sous-grille (bloc 3x3), nous utilisons des clauses pour les paires de cases dans la même sous-grille. Par exemple, pour chaque chiffre « c » et sous-grille (i, j), nous comparons les cases au sein de cette sous-grille pour s'assurer qu'elles ne contiennent pas le même chiffre :

$\neg(s\_(3i+a)(3j+b)(c) \wedge s\_(3i+a)(3j+k)(c))$

Avec ces clauses, le Sudoku peut être représenté en CNF, prêt à être résolu par un SAT-solveur.

2.5 Respect des Indices Donnés
Enfin, les indices fournis au début du puzzle doivent être respectés. Si une case doit contenir un chiffre spécifique, nous utilisons des clauses qui le garantissent :

s_abc

Si une valeur n'est pas autorisée dans une case, nous utilisons :

$\neg s\_abc$

2.6 Conversion en Forme Normale Conjonctive (CNF)
Une fois toutes ces clauses créées, nous pouvons les convertir en CNF, une collection de clauses où chaque clause est une série de littéraux connectés par des opérateurs "OU", et toutes les clauses sont connectées par des opérateurs "ET" (∧). Cela permet de modéliser le Sudoku de manière à ce qu'il puisse être résolu par un SAT-solveur comme Z3.

3. Implémentation Logicielle

3.1 Approche d'Implémentation du Code du Sudoku
L'approche d'implémentation du Sudoku repose sur la conversion du puzzle en une représentation logique qui peut être résolue par un SAT-solveur. Le processus commence par la modélisation du puzzle Sudoku avec des variables booléennes pour chaque combinaison de ligne, colonne, et valeur. Le code traduit ensuite ces variables en clauses logiques sous forme normale conjonctive (CNF). Ces clauses CNF sont utilisées par le SAT-solveur pour déterminer une solution satisfaisante.

Le code implémente des fonctions qui gèrent différentes contraintes du Sudoku, puis utilise un SAT-solveur comme Z3 pour résoudre le puzzle.

3.2 Rôle des Différentes Fonctions du Code
Le code fourni comporte plusieurs fonctions qui jouent des rôles spécifiques dans l'implémentation du Sudoku :

- clauses_indices : Cette fonction crée des clauses pour s'assurer que les indices donnés du Sudoku sont respectés. Elle ajoute des clauses CNF qui fixent des valeurs spécifiques aux cases correspondantes.

- clauses_remplissage : Cette fonction garantit qu'il y a au moins un chiffre dans chaque case du Sudoku. Elle crée des clauses qui permettent de remplir chaque case avec un chiffre de 1 à 9.

- clauses_unique_ligne et clauses_unique_colonne : Ces fonctions assurent que chaque chiffre apparaît au plus une fois dans chaque ligne et chaque colonne. Elles créent des clauses qui interdisent que le même chiffre apparaisse plus d'une fois dans une ligne ou une colonne.

- clauses_unique_zone : Cette fonction assure que chaque chiffre apparaît au plus une fois dans chaque sous-grille (bloc 3x3). Elle crée des clauses qui évitent les doublons dans les sous-grilles.

- cnf_to_dimacs : Cette fonction convertit les clauses CNF en format DIMACS, qui est un format standard pour les SAT-solveurs. Elle écrit les clauses dans un fichier que le SAT-solveur peut utiliser pour résoudre le puzzle.

- dimacs_to_3sat : Cette fonction convertit les clauses DIMACS en format 3-SAT, un sous-ensemble de CNF où chaque clause a trois littéraux ou moins. Elle assure la compatibilité avec les solveurs SAT qui travaillent spécifiquement avec des clauses 3-SAT.

- solve_sat : Cette fonction utilise le SAT-solveur pour résoudre le puzzle Sudoku. Elle lit les clauses DIMACS et retourne une solution si elle existe.

Afin d'éviter de rendre ce rapport trop long, veuillez voir les autres fonctions dans le code source.

3.3 Structures de Données Utilisées

Pour représenter le puzzle Sudoku et les structures associées, le code utilise plusieurs types de structures de données :

- Tableau 3D : Le tableau s est une structure 3D où chaque élément s[a][b][c] représente une combinaison de ligne, colonne, et valeur. Cette structure permet de modéliser chaque case du Sudoku avec des variables booléennes.

- Tableau de Clauses : Les clauses logiques sont stockées dans un tableau 2D où chaque sous-tableau représente une clause CNF. Les clauses contiennent des littéraux, qui sont des valeurs entières représentant les variables ou leur négation.

- Dictionnaire : Les indices du Sudoku sont stockés dans un dictionnaire clues, où les clés représentent les positions (x, y), et les valeurs représentent les chiffres associés.

- Résultat du SAT-solveur : Le résultat du SAT-solveur est stocké dans un dictionnaire qui indique quelles variables booléennes sont vraies, permettant de reconstruire la solution finale du Sudoku.

4. Résolution avec le SAT-solveur

4.1 Processus de Résolution avec le SAT-solveur

Le processus de résolution du Sudoku avec un SAT-solveur implique plusieurs étapes :

Modélisation en CNF : Avant de résoudre le Sudoku, il doit être converti en forme normale conjonctive (CNF). Cette conversion est effectuée en créant des clauses logiques qui respectent les contraintes du Sudoku. Une fois le problème modélisé en CNF, il peut être traité par le SAT-solveur.
Création du Fichier DIMACS: Après avoir généré les clauses CNF, elles sont écrites dans un fichier au format DIMACS, un format standard pour les SAT-solveurs. Le fichier DIMACS contient des informations sur le nombre de variables et de clauses, ainsi que les clauses elles-mêmes.
Résolution avec le SAT-solveur: Le fichier DIMACS est ensuite soumis au SAT-solveur pour déterminer si une solution existe. Le SAT-solveur analyse les clauses et tente de trouver une affectation des variables qui satisfait toutes les contraintes.
Interprétation des Résultats: Si le SAT-solveur trouve une solution, elle est interprétée pour reconstituer le puzzle Sudoku. Le résultat est une grille remplie avec les valeurs correctes, respectant toutes les contraintes du Sudoku.

5. Tests et Validation

Pour valider le code du Sudoku, nous avons effectué une série de tests pour assurer son bon fonctionnement et sa capacité à résoudre différents puzzles. Ces tests permettent de vérifier que le programme répond aux attentes et respecte les contraintes du Sudoku.

Dans la suite, vous trouverez quelques captures d'écran qui illustrent l'environnement du programme et montrent son interface ou ses résultats.

Un jeu avec seulement une solution possible :

```
PS C:\Users\kimia khanoom\OneDrive\پاتکس‌د\ProjetSudoku> python .\main.py .\jeu2-vrai-9x9.txt
Voici le puzzle initial :
8   1 | 5     | 2
6     |     1 | 4
  7   |     8 |     6
--------------------
      | 1   2 |
3     |   6   | 8
1     | 7 5   |
--------------------
  9 8 |       | 6 1 7
4 2   |       |   8
      |       |

Le puzzle est solvable !
Voici la solution :
8 4 1 | 5 7 6 | 2 9 3
6 3 5 | 9 2 1 | 4 7 8
2 7 9 | 3 4 8 | 1 5 6
--------------------
9 6 4 | 1 8 2 | 7 3 5
3 5 7 | 4 6 9 | 8 2 1
1 8 2 | 7 5 3 | 9 6 4
--------------------
5 9 8 | 2 3 4 | 6 1 7
4 2 3 | 6 1 7 | 5 8 9
7 1 6 | 8 9 5 | 3 4 2
```

Un jeu qui n’a aucune réponse :

```
PS C:\Users\kimia khanoom\OneDrive\پاتکس‌د\ProjetSudoku> python .\main.py .\jeu1-faux-9x9.txt
Voici le puzzle initial :
      |       | 6   1
1     |     4 |
    5 | 3   6 | 8 2 1
--------------------
    4 | 6 7   |   5
    7 |       | 9
      | 5 4   |
--------------------
3 7   | 4   5 | 2   6
      |       | 5 1
  6   |   2   |   3 7

Le puzzle est insolvable.
```

Un puzzle 4*4 :

```
PS C:\Users\kimia khanoom\OneDrive\پاتکس‌د\ProjetSudoku> python .\main.py .\jeu3-4x4.txt
Voici le puzzle initial :
2   | 1
    |   2
---------
4   |
  3 |

Le puzzle est solvable !
Voici la solution :
2 4 | 1 3
3 1 | 4 2
---------
4 2 | 3 1
1 3 | 2 4
```

Conclusion

Dans ce projet, nous avons réussi à automatiser la résolution de puzzles Sudoku en utilisant un SAT-solveur. Nous avons modélisé le Sudoku en forme normale conjonctive (CNF), généré des clauses logiques pour respecter les contraintes du puzzle, puis utilisé le SAT-solveur Z3 pour trouver des solutions. Les résultats obtenus ont confirmé que cette approche est efficace pour résoudre des puzzles de Sudoku de différentes tailles et niveaux de complexité.

Ce projet nous a permis de développer des compétences en modélisation logique, en programmation avec Python, et en utilisation de solveurs SAT. Nous avons également appris l'importance de tester et valider le code pour assurer des résultats fiables