Projet SAT solver

Votre travail est d'écrire un SAT solver, c'est-à-dire un programme qui lit un fichier représentant une formule sous forme normale conjonctive et qui détermine si cette formule est satisfiable.

1 Fichier en entrée

Votre programme devra prendre **en argument sur la ligne de commande** un chemin vers un fichier représentant une formule sous forme normale conjonctive.

On supposera que le fichier est fourni au format DIMACS, ce qui implique :

- qu'il peut contenir des lignes de commentaires, de la forme « `c xxx` », où `xxx` peut-être n'importe quoi ;
- qu'il contient une ligne de description des propriétés de la formule, de la forme « `p cnf V C` », où `V` est le nombre de variables et `C` le nombre de clauses ;
- que les clauses sont chacunes représentées sur une ligne (nécessairement après la ligne de description) par une suite d'entiers non nuls, terminée par un 0.

  Par exemple, la ligne « `1 -9 -2 7 0` » représente la clause $x_1 \vee \neg x_9 \vee \neg x_2 \vee x_7$.

Vous trouverez des exemples de tels fichiers, qui pourront servir pour vos tests, sur la page suivante : `https://www.cs.ubc.ca/~hoos/SATLIB/benchm.html`

2 Sortie du programme

Votre programme écrira son résultat **sur la sortie standard**, en respectant le format suivant :

- si la formule est non satisfiable : « `s UNSATISFIABLE` » ;
- si la formule est satisfiable : « `s SATISFIABLE` ».

  Dans ce cas, des lignes supplémentaires seront imprimées, pour donner un modèle de la formule. Chaque ligne supplémentaire sera de la forme « `v N` », où l'entier `N` représente à la fois une variable et sa valeur de vérité. Par exemple, la ligne « `v 3` » indique que la variable x_3 est vraie alors que la ligne « `v -2` » indique que la variable x_2 est fausse.

3 Exigences de rendu

Chaque groupe devra rendre une archive **au plus tard le dimanche 29 mai 2022**, contenant :

- le code source (en OCaml ou en C) de votre programme, structuré de manière lisible et commenté ;
- un fichier README dans lequel vous expliquez comment compiler et exécuter votre programme et où vous présentez brièvement vos choix de conception (structures de données, éventuelles optimisations algorithmiques, ...) ;
- un répertoire contenant quelques fichiers de test de votre conception.

Il va sans dire que votre code doit compiler et s'exécuter sans erreur ni avertissement, au moins sur vos tests.

Votre code doit permettre de choisir entre l'algorithme de Quine et l'algorithme DPLL (dans sa version la plus naïve), via un jeu d'option sur la ligne de commande. Par exemple, vous pouvez proposer un programme tel que, s'il est compilé en un exécutable nommé `sat_solver`, la ligne de commande :

```
./sat_solver -algo quine fichier.cnf
```

exécute l'algorithme de Quine sur la formule représentée par le fichier `fichier.cnf`, et tel que si on remplace `quine` par `dpll` sur la ligne précédente, alors c'est l'algorithme DPLL qui sera implémenté. Ceci n'est bien-sûr qu'une proposition : vous choisissez vous-mêmes quelle forme doit prendre la ligne de commande, tant que cela est clairement documenté dans le fichier README.

Vous devrez également proposer, en option supplémentaire pour l'algorithme DPLL, le choix d'une heuristique visant à améliorer l'efficacité du programme. Voici une liste non exhaustive d'heuristiques possibles :

- lors du choix d'un littéral pour construire un modèle, on choisit le littéral le plus fréquent ;
- ou bien un littéral apparaissant dans une clause de taille minimale ;
- ou bien un littéral tiré aléatoirement uniformément ;
- pour les plus aguerris : l'heuristique « watched literals » (je vous laisse chercher sur internet) ;
- ...

Votre code doit proposer en option **au moins une** heuristique, et fournir une option par heuristique. Par exemple, la ligne de commande pourra s'écrire ainsi :

```
./sat_solver -algo dpll -watched-literals fichier.cnf
```

pour activer l'usage de l'heuristique « watched literals ». Là encore, vous avez le choix de la forme de la ligne de commande et des heuristiques que vous implémentez, tant que cela est clairement expliqué dans le fichier README.

4 Quelques critères d'évaluation

Voici une liste de critères qui pourront servir à l'évaluation de votre projet :

- Clarté : votre code est-il lisible ? bien documenté ? Le fichier README fournit-il toutes les informations nécessaires ?
- Modularité : cela va de pair avec le point précédent. Votre code est-il assez modulaire pour éviter les répétitions de morceaux de code ? pour éventuellement changer à moindre coût les structures de données ?
- Efficacité : le choix des structures de données et leur implémentation sont-ils pertinents, compte tenu de l'objectif de résoudre efficacement le problème SAT ? Votre programme résout-il en temps raisonnable des instances de SAT de taille raisonnable ?
- Pertinence des tests : le jeu de test fourni permet-il d'évaluer la correction et l'efficacité de votre programme ?