

TP n°4

Utilisation de *minisat*

minisat Dans ce TP, nous allons utiliser le solveur *minisat*. Il s'agit d'un solveur gratuit, qui permet de determiner comme son nom l'indique la satisfaisabilite de formules propositionnelles. *minisat* accepte en entree des formules representees au format *DIMACS*, que nous avons decrit dans le TP precedent.

minisat est installe sur toutes les machines de TP, et s'utilise en ligne de commande dans un terminal. Sa syntaxe d'utilisation est

```
minisat entree.dimacs sortie
```

Considerons par exemple la formule suivante :

$$x_3 \wedge (x_1 \vee \neg x_3) \wedge (x_2 \vee x_3 \vee \neg x_1)$$

Lorsqu'il est appele sur le fichier *DIMACS* correspondant a cette formule, *minisat* donne le resultat suivant :

```
SAT
1 -2 3 0
```

SAT signifie que la formule est satisfaisable. 1 -2 3 signifie que l'affectation $[x_1 \mapsto 1; x_2 \mapsto 0; x_3 \mapsto 1]$ est une solution au probleme. En revanche, avec la formule

$$x_3 \wedge \neg x_3$$

minisat donnera

```
UNSAT
```

qui signifie que la formule n'est pas satisfaisable.

Exercice 1 Testez *minisat* sur les formules suivantes :

$$(x_5 \vee \neg x_3 \vee x_1 \vee \neg x_2) \wedge (\neg x_5 \vee \neg x_1 \vee x_2 \vee \neg x_2)$$

$$(x_3 \vee x_1 \vee \neg x_2) \wedge (\neg x_2 \vee \neg x_1 \vee \neg x_2) \wedge (\neg x_3 \vee \neg x_1 \vee x_2)$$

Pour l'exercice suivant, on utilisera les classes d'entree/sortie `Lecteur` et `Ecrivain` disponibles sur la page du cours. Vous aurez egalement besoin de methodes pour manipuler des chaines de caracteres, par exemple `String.split` ou `Integer.parseInt`. Reportez-vous a l'API pour plus de precisions sur ces methodes ! On creera une nouvelle classe `OutilsMinisat`.

Exercice 2 [Obtenir toutes les solutions avec minisat]

minisat ne renvoie qu'une seule affectation rendant la formule vraie. Dans cet exercice, nous allons essayer d'obtenir à partir de *minisat* toutes les affectations rendant la formule vraie.

1. Ecrire une méthode boolean `isReponseMiniSat(String fichier)` qui lit le fichier de sortie de *minisat* `fichier`. Elle renverra vrai si la formule est satisfaisable, faux sinon.
2. écrivez une méthode String `clauseFausse(String fichier)` qui lit le fichier de sortie de *minisat* `fichier`. Si le fichier contient "UNSAT", elle renverra null. Sinon, la deuxième ligne de `fichier` représente une affectation. En ce cas, la méthode renverra une ligne DIMACS obtenue en changeant de signe à tous les nombres de la deuxième ligne de `fichier`. Cette ligne DIMACS représente une clause disjonctive. Il s'agit de la clause qui est fausse par rapport à l'affectation contenue dans la deuxième ligne de `fichier`, et vraie pour toutes les autres affectations.
3. En vous aidant des questions précédentes, écrire une méthode void `afficheMiniSat(String fichier)`, qui prend en argument un fichier en format DIMACS, et par des appels successifs à *minisat* (effectués à l'aide de la commande `Execution.exec` disponible sur la page web du cours), affiche toutes les affectations qui satisfont la formule.

Exercice 3 [Comparaison des performances de minisat et de isSatisfaisable]

Cet exercice est plus long et il est conseillé seulement si vous en avez le temps. Nous allons comparer les performances de *minisat* et de la fonction `isSatisfaisable` que nous avons codée dans les TPs précédents.

1. Ecrire une méthode String `randomCNF(int nc, int nl, int nv)` qui génère aléatoirement le contenu d'un fichier DIMACS avec `nc` clauses, `nl` littéraux par clause, avec un alphabet de `nv` variables.
Par exemple, `randomCNF(2, 4, 4)` pourrait renvoyer la formule suivante :

```
p cnf 4 2
1 -3 1 -2 0
-4 -1 2 -2 0
```
2. écrire une méthode `Formule lireDIMACS(String s)` qui lit un fichier DIMACS et construit la formule correspondante
3. Comparer les temps d'exécution de *minisat* et de votre méthode `isSatisfaisable` sur de grandes formules aléatoires en FNC (on utilisera la commande `java.lang.System.currentTimeMillis()`, qui donne le nombre de *ms* écoulées depuis le 1^{er} janvier 1970.)