

TP n°4 - Correction

Utilisation de *minisat*

minisat Dans ce TP, nous allons utiliser le résolveur *minisat*. Il s'agit d'un solveur gratuit, qui permet de déterminer comme son nom l'indique la satisfaisabilité de formules propositionnelles. *minisat* accepte en entrée des formules représentées au format *DIMACS*, que nous avons décrit dans le TP précédent.

minisat est installé 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
```

Considérons 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 appelé sur le fichier *DIMACS* correspondant à cette formule, *minisat* donne le résultat 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 problème. 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 :

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

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

Correction : Il suffit d'écrire les fichiers DIMACS suivant, et de lancer *minisat* dessus :

```
- p cnf 5 2
5 -3 1 -2 0
-5 -1 2 -2 0
ce qui donne
SAT
-1 -2 -3 -4 -5 0
```

```

- p cnf 3 3
  3 1 -2 0
  -2 -1 -2 0
  -3 -1 2 0
ce qui donne
SAT
-1 -2 -3 0

```

Pour l'exercice suivant, on utilisera les classes d'entrée/sortie `Lecteur` et `Ecrivain` disponibles à <http://www.pps.jussieu.fr/~buccia/COURS/OL3/>. Vous aurez également besoin de méthodes pour manipuler des chaînes de caractères, par exemple `String.split` ou `Integer.parseInt`. Reportez-vous à l'API pour plus de précisions sur ces méthodes! On créera 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. Écrire 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.

Correction :

```

public static boolean isReponseMinisat(String fichier)
{
    Lecteur l = new Lecteur(fichier);
    String s=l.readLine();
    if (s.equals("UNSAT")) return false;
    return true;
}

```

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 fautive par rapport à l'affectation contenue dans la deuxième ligne de `fichier`, et vraie pour toutes les autres affectations.

Correction :

```

public static String clauseFausse(String fichier)
{
    Lecteur l = new Lecteur(fichier);
    String s=l.readLine();
    if (s.equals("UNSAT")) return null;
    s=l.readLine();
    String[] tab = l.separe(s);
    int var;
    String rep="";
    for (int i=0; i<tab.length-1; i++) {
        var = - Integer.parseInt(tab[i]);
        rep = rep + var + " ";
    }
    rep = rep+"0";
    return rep;
}

```

3. En vous aidant des questions précédentes, écrire une méthode `void itereMinisat(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.

Correction :

```
public static void itereMinisat(String fichier)
{
    String[] commands = {"minisat", fichier, "reponse"};
    Execution.exec(commands);
    while(isReponseMinisat("reponse")) {
        Lecteur l = new Lecteur("reponse");
        System.out.println(l.toutLeFichier());
        l = new Lecteur(fichier);
        String [] prem = l.separe(l.readLigne());
        prem[3] = "" + (Integer.parseInt(prem[3])+1);
        String nouvelleprem = "p cnf " + prem[2] + " " +premier[3] + "\n";
        String reste = "";
        while (l.hasLigne())
            reste = reste + l.readLigne() + "\n";
        Ecrivain e=new Ecrivain(fichier);
        e.write(nouvelleprem);
        e.write(reste);
        e.write(clauseFausse("reponse"));
        Execution.exec(commands);
    }
}
```

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. Écrire 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
```

Correction :

Une recherche rapide indique par exemple que la classe `Random` permet de générer des nombres aléatoires.

```
public static int randomLiteral(int nv){
    Random generator = new Random();
    if (generator.nextInt(2)==0)
        return generator.nextInt(nv)+1;
    else
        return -(generator.nextInt(nv)+1);
}
```

```

public static String randomClause(int nl, int nv){
    String c = "";
    for (int i=0; i<nl; i++)
        c = c + randomLiteral(nv) + " ";
    c = c + "\n";
    return c;
}

public static String randomCNF(int nc, int nl, int nv){
    String f = "";
    for (int i=0; i<nc; i++)
        f = f + randomClause(nl,nv);
    return f;
}

```

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.)