Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université des Sciences et de la Technologie Houari Boumediene
Faculté d'électronique et d'informatique
Département d'informatique

Rapport de TP

Module : Compilation II

Master 1 SII

Mini projet

Problème de satisfiabilité (SAT)

• Réalisé par :

BENHADDAD Wissam

BOURAHLA Yasser

23-02-2018

Table des matières

Première partie

Introduction définition

Problème SAT En informatique théorique, le problème SAT ou problème de satisfaisabilité est un problème de décision, qui, étant donné une formule de logique propositionnelle, détermine s'il existe une assignation des variables propositionnelles qui rend la formule vraie.

Ce problème est très important en théorie de la complexité. Il a été mis en lumière par le théorème de Cook [1], qui est à la base de la théorie de la NP-complétude. Le problème SAT a aussi de nombreuses applications notamment en satisfaction de contraintes, planification classique, model checking, diagnostic, et jusqu'au configurateur d'un PC ou de son système d'exploitation[2] : on se ramène à des formules propositionnelles et on utilise des solveurs SAT.

Solveur SAT Un solveur SAT est un algorithme qui va essayer de trouver(si elles existent) les instances des variables booléennes pour les quelles toutes les clauses sont **Vraies**, dans ce TP nous utiliserons le solveur **UBCSAT** disponible ICI, en téléchargement, le code source est aussi disponible sur Github.

Deuxième partie

Expérimentations

1 Bases de connaissances au format CNF

Le fichier se présente sous le format standard **.CNF**, donc un ensembles de clauses en **Forme normale conjonctive**.

1.1 Examples 1 :

Le fichier **Test.cnf** contient les symboles $\neg$ logiques suivants : X1,X2,X3,X4,X5
Ainsi que les connaissances sous forme FNC :
$\{X_2 \vee \neg X_3\} = \{2, -3, 0\}$
$\{\neg X_3\} = \{-3, 0\}$
$\{X_1 \vee \neg X_2 \vee \neg X_3 \vee X_4\} = \{1, -2, -3, 4, 0\}$
$\{\neg X_1 \vee \neg X_4\} = \{-1, -4, 0\}$
$\{X_2 \vee \neg X_4\} = \{2, -4, 0\}$
$\{X_1 \vee X_3\} = \{1, 3, 0\}$
$\{\neg X_1 \vee \neg X_2 \vee X_3 \vee X_5\} = \{-1, -2, 3, 5, 0\}$
$\{X_2 \vee \neg X_5\} = \{2, -5, 0\}$
$\{\neg X_3 \vee X_4 \vee \neg X_5\} = \{-3, 4, -5, 0\}$
$\{X_1 \vee X_2 \vee X_5\} = \{1, 2, 5, 0\}$
$\{\neg X_3 \vee X_5\} = \{-3, 5, 0\}$

En exécutant **UBCSAT** avec la commande :

```
wiss@quantum  ~/CODES/TP RC/TP1  ./ubcsat -alg saps  -solve -i test.cnf
```

Le résultat obtenu est le suivant :

```
#
# Solution found for -target 0

 1 2 -3 -4 5


Variables = 5
Clauses = 11
TotalLiterals = 27
TotalCPUTimeElapsed = 0.000
FlipsPerSecond = inf
RunsExecuted = 1
SuccessfulRuns = 1
PercentSuccess = 100.00
Steps_Mean = 3
Steps_CoeffVariance = 0
Steps_Median = 3
CPUTime_Mean = 0
CPUTime_CoeffVariance = 0
CPUTime_Median = 0
```

L'intérprétation de ce résultat est que la base de connaissances admet au moins un modèle ,dont les valeurs de vérité des litéraux sont les suivantes :
$v(X1) = VRAI$
$v(X2) = VRAI$
$v(X3) = FAUX$
$v(X4) = FAUX$
$v(X5) = VRAI$

1.2 Examples 2 :

Le fichier **Test1.cnf** contient les symboles ¬ logiques suivants : X1,X2,X3,X4,X5
Ainsi que les connaissances sous forme FNC :
$\{X_2 \vee \neg X_3\} = \{2, -3, 0\}$
$\{\neg X_3\} = \{-3, 0\}$
$\{X_1 \vee \neg X_2 \vee \neg X_3 \vee X_4\} = \{1, -2, -3, 4, 0\}$
$\{\neg X_1 \vee \neg X_4\} = \{-1, -4, 0\}$
$\{X_2 \vee \neg X_4\} = \{2, -4, 0\}$
$\{X_1 \vee X_3\} = \{1, 3, 0\}$
$\{\neg X_1 \vee \neg X_2 \vee X_3 \vee X_5\} = \{-1, -2, 3, 5, 0\}$
$\{X_2 \vee \neg X_5\} = \{2, -5, 0\}$
$\{\neg X_3 \vee X_4 \vee \neg X_5\} = \{-3, 4, -5, 0\}$
$\{X_1 \vee X_2 \vee X_5\} = \{1, 2, 5, 0\}$
$\{X_3 \vee X_5\} = \{3, 5, 0\}$
$\{\neg X_5\} = \{-5, 0\}$
$\{X_3\} = \{3, 0\}$

En exécutant **UBCSAT** avec la commande :

```
wiss@quantum  ~/CODES/TP RC/TP1  ./ubcsat -alg saps  -solve -i test1.cnf
```

Le résultat obtenu est le suivant :

```
# No Solution found for -target 0


Variables = 5
Clauses = 13
TotalLiterals = 29
TotalCPUTimeElapsed = 0.020
FlipsPerSecond = 5000000
RunsExecuted = 1
SuccessfulRuns = 0
PercentSuccess = 0.00
Steps_Mean = 100000
Steps_CoeffVariance = 0
Steps_Median = 100000
CPUTime_Mean = 0.02
CPUTime_CoeffVariance = 0
CPUTime_Median = 0.02
```

Le solveur a determiné qu'il ne pouvait exister une instance telle que toutes les clauses soient vraies en même temps.

Troisième partie

Construction et test d'une base de connaissances zoologiques

2 Langage naturel

Les connaissances sont présentées dans le langage naturel comme ceci :

- Les nautiles sont des céphalopodes.
- Les céphalopodes sont des mollusques.
- Les mollusques ont généralement une coquille.
- Les céphalopodes n'en ont généralement pas.
- Les nautiles en ont une.
- a est un nautile.
- b est un céphalopode.
- c est un mollusque.

3 En logique d'ordre sifron et format .CNF

Nous avons décidé de prendre en compte le sens du mot **généralement**, et nous avons donc obtenu la traduction en logique propositionnelle suivante :

Symboles ¬ logiques et leurs codifications :
Céa = 1
Céb = 2
Céc = 3
Na = 4
Nb = 5
Nc = 6
Coa = 7
Cob = 8
Coc = 9
Ma = 10
Mb = 11
Mc = 12

FBF en clauses prêtes à l'exécution sous ubcsat

- $\{FBF\} \equiv \{Clause\} \Longrightarrow \{Codification\}$
- $\{Na\} \equiv \{Na\} \Longrightarrow \{4, 0\}$
- $\{Ceb\} \equiv \{Ceb\} \Longrightarrow \{2, 0\}$
- $\{Mc\} \equiv \{Mc\} \Longrightarrow \{12, 0\}$
- $\{Na \Rightarrow Cea\} \equiv \{\neg Na \vee Cea\} \Longrightarrow \{-4, 1, 0\}$
- $\{Nb \Rightarrow Ceb\} \equiv \{\neg Nb \vee Ceb\} \Longrightarrow \{-5, 2, 0\}$
- $\{Nc \Rightarrow Cec\} \equiv \{\neg Nc \vee Cec\} \Longrightarrow \{-6, 3, 0\}$
- $\{Cea \Rightarrow Ma\} \equiv \{\not Cea \vee Ma\} \Longrightarrow \{-1, 10, 0\}$
- $\{Ceb \Rightarrow Mb\} \equiv \{\not Ceb \vee Mb\} \Longrightarrow \{-2, 11, 0\}$
- $\{Cec \Rightarrow Mc\} \equiv \{\not Cec \vee Mc\} \Longrightarrow \{-3, 12, 0\}$
- $\{Na \Rightarrow Coa\} \equiv \{\neg Na \vee Coa\} \Longrightarrow \{-4, 7, 0\}$
- $\{Nb \Rightarrow Cob\} \equiv \{\neg Nb \vee Cob\} \Longrightarrow \{-5, 8, 0\}$
- $\{Nc \Rightarrow Coc\} \equiv \{\neg Nc \vee Coc\} \Longrightarrow \{-6, 9, 0\}$
- $\{(Cea \wedge \neg Na) \Rightarrow \neg Coa\} \equiv \{\neg Cea \vee Na \vee \neg Coa\} \Longrightarrow \{-1, 4, -7, 0\}$
- $\{(Ceb \wedge \neg Nb) \Rightarrow \neg Cob\} \equiv \{\neg Ceb \vee Nb \vee \neg Cob\} \Longrightarrow \{-2, 5, -8, 0\}$
- $\{(Cec \wedge \neg Nc) \Rightarrow \neg Coc\} \equiv \{\neg Cec \vee Nc \vee \neg Coc\} \Longrightarrow \{-3, 6, -9, 0\}$
- $\{Ma \wedge \neg(Cea \wedge \neg Na) \Rightarrow Coa\} \equiv \{(\neg Ma \vee Cea \vee Coa) \wedge (\neg Ma \vee \neg Na \vee Coa)\} \Longrightarrow \{-10, 1, 7, 0\} \wedge \{-10, -4, 7, 0\}$

- $\{Mb \wedge \neg(Ceb \wedge \neg Nb) \Rightarrow Cob\} \equiv \{(\neg Mb \vee Ceb \vee Cob) \wedge (\neg Mb \vee \neg Nb \vee Cob)\} \Longrightarrow \{-11, 2, 8, 0\} \wedge \{-11, -5, 8, 0\}$
- $\{Mc \wedge \neg(Cec \wedge \neg Nc) \Rightarrow Coc\} \equiv \{(\neg Mc \vee Cec \vee Coc) \wedge (\neg Mc \vee \neg Nc \vee Coc)\} \Longrightarrow \{-12, 3, 9, 0\} \wedge \{-12, -6, 9, 0\}$

Fichier Zoo.cnf :

```
p cnf 12 21
4 0
2 0
12 0
-4 1 0
-5 2 0
-6 3 0
-1 10 0
-2 11 0
-3 12 0
-4 7 0
-5 8 0
-6 9 0
-1 4 -7 0
-2 5 -8 0
-3 6 -9 0
-10 1 7 0
-10 -4 7 0
-11 2 8 0
-11 -5 8 0
-12 3 9 0
-12 -6 9 0
```

4 Application du *ubcsat* sur le fichier zoo.cnf

```
#
# Solution found for -target 0

 1 2 -3 4 5 -6 7 8 9 10
 11 12


Variables = 12
Clauses = 21
TotalLiterals = 48
TotalCPUTimeElapsed = 0.000
FlipsPerSecond = inf
RunsExecuted = 1
SuccessfulRuns = 1
PercentSuccess = 100.00
Steps_Mean = 8
Steps_CoeffVariance = 0
Steps_Median = 8
CPUTime_Mean = 0
CPUTime_CoeffVariance = 0
CPUTime_Median = 0
```

Ce qui correspond a l'instance vue en cours :
Na{4}, Céa{1}, Céb{2}, Ma{10}, Mb{11}, Mc{12}, Coa{7} ont la valeur vrai Nb{5} et Cob{8} ont la même valeur ; Si Céc{-3} est faux, Nc{-6} est faux et Coc{9}vrai

5 Test sur un fichier Benchmark

Nous avons choisit deux fichiers pour tester le solveur **UBCSAT** qui sont :
uf250-1065 et uuf250-1065
Les deux archives contiennts 100 instances de 250 variables, 1065 clauses satisfiables (resp. non satisfiable) [3]

5.1 uf250-1065 :

La commande est :

```
wiss@quantum  ~/CODES/TP RC/TP1  ./ubcsat -alg saps  -solve -i UF250.1065.100/uf250-069.cnf
```

Le résultat obtenu est :

```
#
# Solution found for -target 0

 1 -2 -3 -4 5 6 7 8 -9 10
 11 12 13 -14 15 -16 17 18 -19 20
 -21 22 -23 -24 -25 26 27 -28 29 30
 31 32 33 34 -35 -36 37 38 -39 -40
 -41 42 -43 44 -45 46 47 48 49 50
 -51 52 -53 -54 55 56 57 58 -59 -60
 -61 -62 -63 -64 -65 66 -67 -68 69 70
 -71 -72 -73 -74 -75 76 -77 78 79 -80
 -81 82 83 -84 -85 -86 87 88 -89 -90
 91 92 93 94 95 -96 97 -98 -99 100
 -101 102 -103 -104 105 106 107 108 -109 -110
 -111 112 113 -114 -115 116 -117 -118 -119 120
 -121 -122 123 -124 125 126 -127 128 129 130
 -131 -132 133 134 135 136 137 138 -139 140
 141 142 -143 -144 -145 -146 -147 148 -149 -150
 -151 152 -153 -154 155 156 157 -158 -159 -160
 -161 -162 -163 164 -165 -166 167 168 -169 -170
 171 172 -173 174 175 176 177 -178 179 -180
 -181 -182 -183 -184 185 -186 -187 -188 -189 -190
 191 -192 193 -194 -195 -196 197 -198 -199 -200
 201 -202 -203 -204 -205 206 207 208 209 210
 211 -212 213 214 -215 -216 217 -218 219 -220
 221 -222 -223 224 225 -226 227 228 229 -230
 -231 -232 233 -234 235 -236 -237 -238 -239 -240
 -241 -242 243 -244 245 -246 247 -248 -249 -250
```

5.2 uuf250-1065 :

La commande est :

```
✘ wiss@quantum  ~/CODES/TP RC/TP1  ./ubcsat -alg saps  -solve -i UUF250.1065.100/uuf250-069.cnf
```

Le résultat obtenu est :

```
# No Solution found for -target 0


Variables = 250
Clauses = 1065
TotalLiterals = 3195
TotalCPUTimeElapsed = 0.060
FlipsPerSecond = 1666667
RunsExecuted = 1
SuccessfulRuns = 0
PercentSuccess = 0.00
Steps_Mean = 100000
Steps_CoeffVariance = 0
Steps_Median = 100000
CPUTime_Mean = 0.06
CPUTime_CoeffVariance = 0
CPUTime_Median = 0.06
```

5.3 Commentaires :

Le solveur SAT est extrêmement rapide, il arrive a trouver des instances positives en une fraction de secondes, ce qui est un exploit vu la taille du problème.

Quatrième partie
Simulation d'un moteur d'inférence

6 Problématique

Étant donné que le problème SAT est hautement combinatoire, le résoudre avec des techniques d'informatique classiques serait très couteux en temps d'exécution, puisque nous avons a nôtre disposition un solveur sat **UBCSAT**, qui effectue cette tâche de façon optimale, nous pouvons l'utiliser afin d'exploiter le raisonnement l'absurde, tout le challenge se trouve donc dans l'écriture d'un programme qui :

- Lit une formule bien formée **Q** en entrée.
- Transforme sa négation en FNC.
- En extrait les clauses.
- Codifie les clauses et les ajoute à la base de connaissances déjà existante.
- Lance le solveur **UBCSAT** sur l'ensemble des clauses BC $\cup$ $\{\neg\mathbf{Q}\}$
- Si le solveur ne trouve pas d'instance qui satisfait la base, alors **BC** infère **Q** sinon on ne peut pas obtenir **Q** à partir de la base **BC**.

7 Implémentation machine

Nous avons choisit d'implémenter ce pseudo-algorithme avec le langage **Python3**[4], sa flexibilité et son assosciation à l'outil Antrl4 [5] nous a permit d'analyser la formule bien formée en entrée et de la tranfsormer au format DIMACS, plus de détails seront données dans la partie suivante.

8 Fonctionnement interne

Le module de conversion d'une FBF en clauses nommé FBFToCNF est entièrement open source est disponible ICI sur **Github**. Le script Python principal est le suivant :
(Les commentaires servent d'explications à chaque étapes)

```python
import os
import sys
import re
import shutil
######################################
#Function to return the number of distinc values in a valid
    CNF format file
def distinctOccurInFile(outLines):
    allVar=[]
    for i in range(1,len(outLines)):
        spl=re.split("\s+",outLines[i].replace("-",""))
        # print spl
        for col in spl :
            if re.match("[1-9]+",col) and (col not in
               allVar) :
                # print col
                allVar.append(col)
    return(len(allVar))
######################################

import FormulaToCNF
#insert here any propositional logic formula
dimacs = FormulaToCNF.getDIMACS(sys.argv[2],False,False)
print dimacs
outF = open("out.cnf", "w+r")
outF.write(dimacs)
outF.seek(0)

inPutFile = str(sys.argv[1])
#Opening the file where the knowledge base is.
file = open(inPutFile,"r")
#Opening the output file which will containt the KB and the
    negation of the formula in CNF
wfile = open("new.cnf","w")
#Var to store the file new.cnf
outLines=[]
for line in file.readlines():
    outLines.append(line)
for line in outF.readlines():
    outLines.append(line)

#Modifying the header of the file to adjust the number of
   Variables/Clauses depending on the input formula
splitF = re.split("\s+",outLines[0]);
if len(sys.argv)>2 :
    splitF[3] = str(int(splitF[3])+len(sys.argv)-2)
splitF[2]=str(distinctOccurInFile(outLines))
outLines[0]=" ".join(splitF)+"\n"
```

```python
#Writing the new KB into the new.cnf file
for str in outLines:
    wfile.write(str)
wfile.close()
file.close()
outF.close()
#Run the SAT solver with the new KB
os.system(" ./ubcsat -alg saps -solve -i new.cnf" )
```

9 Test du mini moteur d'inférence

Prennons un example extrêmement simple, nous avons les symboles non logiques : **X1,X2**
Et la base de connaissance :
$X1 \longrightarrow X2$

$X1$
Il s'agit de déduire la formule :
$Q \equiv X2 \wedge X1$
Il est évident que la formule découle logiquement de la base, car en appliquant le Modus ponens nous pouvons obtenir $X2$, et $X1$ est déjà dans la base, il s'agit maintenant de voir si notre simulateur peut inférer Q.
La base est la suivante :

```
p cnf 2 2
-1 2 0
1 0

```

La commande a lancé est de la forme :
python main.py baseConnaissance.cnf FBF

```
wiss@quantum  ~/CODES/TP RC/TP1/FBFToCNF  master ● python main.py sahla.cnf "-(2 ^ 1)"
```

Le résultat est le suivant :

```
# No Solution found for -target 0


Variables = 2
Clauses = 3
TotalLiterals = 5
TotalCPUTimeElapsed = 0.000
FlipsPerSecond = inf
RunsExecuted = 1
SuccessfulRuns = 0
PercentSuccess = 0.00
Steps_Mean = 100000
Steps_CoeffVariance = 0
Steps_Median = 100000
CPUTime_Mean = 0
CPUTime_CoeffVariance = 0
CPUTime_Median = 0
```

Qui peut se traduire par :
$BC \cup \neg Q$ infered $\perp$
Donc en utilisant le raisonnement par l'absurde nous avons démontré qu'effectivement **BC** infère **Q**

Références

[1] Stephen A. Cook. *The Complexity of Theorem-proving Procedures*. 1971.

[2] https://en.wikipedia.org/wiki/Boolean_satisfiability_problem.

[3] Satlib - benchmark problems. http://www.cs.ubc.ca/~hoos/SATLIB/benchm.html.

[4] https://www.python.org/.

[5] http://www.antlr.org/.