



République Algérienne Démocratique et populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université des Sciences et de la Technologie Houari Boumediene

Faculté d'Electronique et d'informatique
Département d'informatique

Rapport TP n°01

Représentation des connaissances et raisonnement

Inférence logique utilisant le solveur SAT

Réalisé par :
OUCHAOU Chafaa
TAOUCI Kenza

Systèmes Informatiques Intelligents

Année scolaire : 2023/2024

Table des matières

TABLE OF CONTENTS	
LIST OF FIGURES	
CHAPITRE 1 Étude de l'art	1
1.1 Introduction	1
1.2 Définition de l'inférence	1
1.3 Définition du problème SAT	1
1.4 Définition du solveur SAT	2
CHAPITRE 2 Realisation du TP	3
2.1 Étape 1	3
2.2 Étape 2	3
2.2.1 exemple 1.cnf	3
2.2.2 exemple 2.cnf	5
2.3 Étape 3	7
2.3.1 Traduction de la base de connaissances zoologique sous forme CNF :	7
2.3.2 Transformation en CNF avec application de la règle : $a \supset$ $b \equiv \neg a \vee b$	8
2.3.3 Représentation des clauses dans le fichier CNF	8
2.3.4 Après amélioration	9

2.3.5	Test de Benchmarks	12
2.4	Étape 4	14
2.4.1	Partie 1	14
2.4.2	Partie 2	15
2.5	Conclusion	17

Table des figures

2.1	Dossier avec les fichiers UBCSAT et CNF	3
2.2	La première partie du résultat	4
2.3	La deuxième partie du résultat	4
2.4	La troisième partie du résultat	5
2.5	La dernière partie du résultat	5
2.6	Résultat d'exécution du fichier exemple 2.cnf	6
2.7	contenu du fichier CNF	8
2.8	Résultat d'execution du fichir cnf	9
2.9	contenu du fichier CNF après amélioration	10
2.10	Résultat d'execution du fichir cnf après amélioration	11
2.11	Test de Benchmarks 1	12
2.12	Test de Benchmarks 2	13
2.13	Exécution du programme avec entré égale a 9	14
2.14	Exécution du programme avec entré égale a 7	15
2.15	Contenu du fichier cripto	16
2.16	résultat d'exécution avec le fichier cripto	17

CHAPITRE 1

Étude de l'art

1.1 Introduction

Dans le domaine de la logique et des systèmes informatiques, la notion de connaissance revêt une importance capitale. Comprendre les mécanismes par lesquels nous acquérons, traitons et utilisons l'information est fondamental pour de nombreux domaines, de l'intelligence artificielle à la résolution de problèmes complexes. L'inférence qui est l'un des principaux moyens de manipulation des connaissances joue un rôle central dans cette dynamique.

1.2 Définition de l'inférence

L'inférence est un processus par lequel de nouvelles informations ou conclusions sont déduites à partir de données ou de faits existants. Plus précisément, l'inférence consiste à tirer des conclusions logiques ou des implications à partir de prémisses ou d'observations préalables.

En d'autres termes, l'inférence est la capacité de raisonner à partir des informations disponibles afin d'aboutir à de nouvelles connaissances ou de confirmer des hypothèses. Ce processus peut prendre différentes formes, allant de la déduction logique formelle à l'induction empirique.

1.3 Définition du problème SAT

Le problème SAT (Satisfiability Problem), également connu sous le nom de problème de satisfiabilité booléenne, est l'un des problèmes fondamentaux de la théorie de la complexité algorithmique et de la logique propositionnelle. Il consiste à déterminer si une formule booléenne donnée peut être évaluée à vrai en attribuant des valeurs de vérité à ses variables.

Formellement, étant donnée une formule booléenne sous forme conjonctive normale (CNF), le problème SAT consiste à déterminer s'il existe une assignation de valeurs de vérité (vrai ou faux) aux variables qui rend la formule entière vraie.

1.4 Définition du solveur SAT

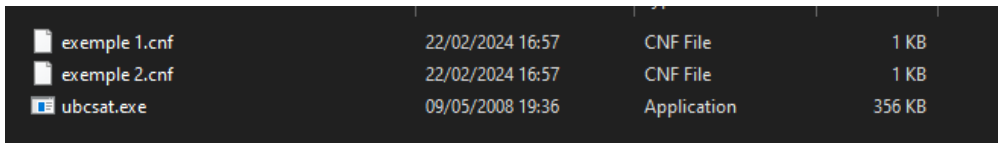
Les solveurs SAT (Satisfiability Solvers), également connus sous le nom de solveurs de satisfiabilité booléenne, sont des programmes informatiques conçus pour résoudre les problèmes SAT. Ces outils utilisent une variété d'algorithmes et de techniques pour rechercher une assignation de valeurs de vérité qui rend la formule booléenne donnée vraie, ou pour déterminer qu'aucune telle assignation n'existe (c'est-à-dire que la formule est insatisfiable).

CHAPITRE 2

Realisation du TP

2.1 Étape 1

Création d'un répertoire UBCSAT et copie des fichiers sous format CNF :






 exemple 1.cnf	22/02/2024 16:57	CNF File	1 KB
 exemple 2.cnf	22/02/2024 16:57	CNF File	1 KB
 ubcsat.exe	09/05/2008 19:36	Application	356 KB

FIGURE 2.1 – Dossier avec les fichiers UBCSAT et CNF

2.2 Étape 2

Exécution du solveur SAT et teste de la satisfiabilité des deux fichiers : exemple 1.cnf et exemple 2.cnf

2.2.1 exemple 1.cnf

Résultat de l'exécution :

La première partie du résultat de l'exécution affiche les détails concernant la version d'UBCSAT, des liens vers des ressources d'aide en ligne, ainsi que divers paramètres du solveur utilisé.

```

D:\Etude S2\rep connaissance\ubcsat\ubcsat v2>ubcsat -alg saps -i "exemple 1.cnf" -solve
#
# UBCSAT version 1.1.0 (Sea to Sky Release)
#
# http://www.satlib.org/ubcsat
#
# ubcsat -h for help
#
# -alg saps
# -runs 1
# -cutoff 100000
# -timeout 0
# -gttimeout 0
# -noimprove 0
# -target 0
# -wtargt 0
# -seed 1535967926
# -solve 1
# -find,-numsol 1
# -findunique 0
# -srestart 0
# -prestart 0
# -drestart 0
#
# -alpha 1.3
# -rho 0.8
# -ps 0.05
# -wp 0.01
# -sapsthresh -0.1

```

FIGURE 2.2 – La première partie du résultat

La deuxième partie, représente le rapport de résultat de traitement (Output report).

```

# UBCSAT default output:
# 'ubcsat -r out null' to suppress, 'ubcsat -hc' for customization help
#
#
# Output Columns: |run|found|best|beststep|steps|
#
# run: Run Number
# found: Target Solution Quality Found? (1 => yes)
# best: Best (Lowest) # of False Clauses Found
# beststep: Step of Best (Lowest) # of False Clauses Found
# steps: Total Number of Search Steps
#
#      F  Best      Step      Total
#      Run N Sol'n    of      Search
#      No. D Found    Best    Steps
#
#      1 1      0      2      2

```

FIGURE 2.3 – La deuxième partie du résultat

En ce qui concerne la troisième partie, elle présente le résultat du traitement, à savoir la sortie de l'UBCSAT. Dans ce cas, une solution a été identifiée, confirmant ainsi la satisfaisabilité du fichier "exemple 1.cnf".


```
#  
# Solution found for -target 0  
  
1 -2 -3 -4 -5
```

FIGURE 2.4 – La troisième partie du résultat

La section finale présente un rapport statistique sur l'exécution du solveur sur le problème donné. Ce rapport inclut des informations telles que le nombre de variables dans le fichier CNF (5 variables dans ce cas), le nombre de clauses (9 clauses dans notre exemple), le pourcentage de réussite (100% pour ce test), et d'autres données pertinentes.

```
Variables = 5  
Clauses = 9  
TotalLiterals = 23  
TotalCPUtimeElapsed = 0.004  
FlipsPerSecond = 500  
RunsExecuted = 1  
SuccessfulRuns = 1  
PercentSuccess = 100.00  
Steps_Mean = 2  
Steps_CoeffVariance = 0  
Steps_Median = 2  
CPUtime_Mean = 0.00399994850159  
CPUtime_CoeffVariance = 0  
CPUtime_Median = 0.00399994850159
```

FIGURE 2.5 – La dernière partie du résultat

2.2.2 exemple 2.cnf

Résultat de l'exécution :

L'exécution affiche : "No solution found" indiquant que l'ensemble des formules en entrée n'est pas satisfiable. Il est également notable que le taux de réussite est effectivement de 0%.

```

# -drestart 0
#
# -alpha 1.3
# -rho 0.8
# -ps 0.05
# -wp 0.01
# -sapsthresh -0.1
#
# UBCSAT default output:
# 'ubcsat -r out null' to suppress, 'ubcsat -hc' for customization help
#
#
# Output Columns: |run|found|best|beststep|steps|
#
# run: Run Number
# found: Target Solution Quality Found? (1 => yes)
# best: Best (Lowest) # of False Clauses Found
# beststep: Step of Best (Lowest) # of False Clauses Found
# steps: Total Number of Search Steps
#
#      F  Best      Step      Total
#      Run N Sol'n    of      Search
#      No. D Found    Best    Steps
#
#      1 0      1      2      100000
# No Solution found for -target 0

Variables = 5
Clauses = 12
TotalLiterals = 28
TotalCPUTimeElapsed = 0.018
FlipsPerSecond = 5555590
RunsExecuted = 1
SuccessfulRuns = 0
PercentSuccess = 0.00
Steps_Mean = 100000
Steps_CoeffVariance = 0
Steps_Median = 100000
CPUTime_Mean = 0.0179998874664
CPUTime_CoeffVariance = 0
CPUTime_Median = 0.0179998874664

```

FIGURE 2.6 – Résultat d'exécution du fichier exemple 2.cnf

2.3 Étape 3

2.3.1 Traduction de la base de connaissances zoologique sous forme CNF :

Reprenant l'énoncé vu en cours

- 1 - Les nautilus sont des céphalopodes
- 2 - Les céphalopodes sont des mollusques
- 3 - Les mollusques ont généralement une coquille
- 4 - Les céphalopodes n'en ont généralement pas
- 5 - Les nautilus en ont une
- 6 - a est un nautilus
- 7 - b est un céphalopode
- 8 - c est un mollusque

Pour la représentation des faits on considère les symboles non logiques suivants :

$\{\mathbf{Na}, \mathbf{Nb}, \mathbf{Nc}\}$; Où Na = Nautilus de a
 $\{\mathbf{Cea}, \mathbf{Ceb}, \mathbf{Cec}\}$; Où Cea = Céphalopode de a
 $\{\mathbf{Moa}, \mathbf{Mob}, \mathbf{Moc}\}$; Où Ma = Mollusque de a
 $\{\mathbf{Coa}, \mathbf{Cob}, \mathbf{Coc}\}$; Où Coa = Coquille de a
Même chose en ce qui concerne b et c.

En cours on est arrivé aux traductions suivantes :

- 1- $(\mathbf{Na} \supset \mathbf{Cea}); (\mathbf{Nb} \supset \mathbf{Ceb}); (\mathbf{Nc} \supset \mathbf{Cec});$
- 2 - $(\mathbf{ea} \supset \mathbf{Ma}); (\mathbf{Ceb} \supset \mathbf{Mb}); (\mathbf{Cec} \supset \mathbf{Mc});$
- 3 - $(\mathbf{Na} \supset \mathbf{Coa}); (\mathbf{Nb} \supset \mathbf{Cob}); (\mathbf{Nc} \supset \mathbf{Coc});$
- 4 - $\mathbf{Na}; \mathbf{Ceb}; \mathbf{Mc};$
- 5 - $(\mathbf{Ma} \supset \mathbf{Coa}); (\mathbf{Mb} \supset \mathbf{Cob}); (\mathbf{Mc} \supset \mathbf{Coc});$
- 6 - $(\mathbf{Cea} \supset \neg \mathbf{Coa}); (\mathbf{Ceb} \supset \neg \mathbf{Cob}); (\mathbf{Cec} \supset \neg \mathbf{Coc});$

2.3.2 Transformation en CNF avec application de la règle : $a \supset b \equiv \neg a \vee b$

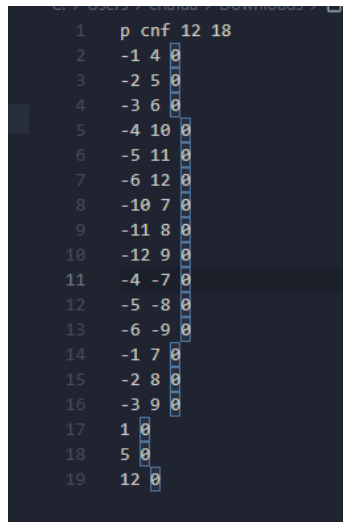
les clauses :

- 1- $(\neg Na \vee Cea); (\neg Nb \vee Ceb); (\neg Nc \vee Cec);$
- 2- $(\neg Cea \vee Ma); (\neg Ceb \vee Mb); (\neg Cec \vee Mc);$
- 3- $(\neg Na \vee Coa); (\neg Nb \vee Cob); (\neg Nc \vee Coc);$
- 4- Na
- 5- Ceb
- 6- Mc
- 7- $(\neg Ma \vee Coa); (\neg Mb \vee Cob); (\neg Mc \vee Coc);$
- 8- $(\neg Cea \vee \neg Coa); (\neg Ceb \vee \neg Cob); (\neg Cec \vee \neg Coc);$

2.3.3 Représentation des clauses dans le fichier CNF

variables non logique :

- 1 = Na; 2 = Nb; 3 = Nc;
4 = Cea; 5 = Ceb; 6 = Cec;
7 = Coa; 8 = Cob; 9 = Coc;
10 = Ma; 11 = Mb; 12 = Mc;



```
1 p cnf 12 18
2 -1 4 0
3 -2 5 0
4 -3 6 0
5 -4 10 0
6 -5 11 0
7 -6 12 0
8 -10 7 0
9 -11 8 0
10 -12 9 0
11 -4 -7 0
12 -5 -8 0
13 -6 -9 0
14 -1 7 0
15 -2 8 0
16 -3 9 0
17 1 0
18 5 0
19 12 0
```

FIGURE 2.7 – contenu du fichier CNF

Test SAT

Il n'existe pas de solution satisfaisant l'ensemble des clauses en entrée, le Zoo.cnf n'est donc pas satisfiable.

```
# No Solution found for -target 0

Variables = 12
Clauses = 18
TotalLiterals = 33
TotalCPUtimeElapsed = 0.016
FlipsPerSecond = 6249987
RunsExecuted = 1
SuccessfulRuns = 0
PercentSuccess = 0.00
Steps_Mean = 100000
Steps_CoeffVariance = 0
Steps_Median = 100000
CPUtime_Mean = 0.0160000324249
CPUtime_CoeffVariance = 0
CPUtime_Median = 0.0160000324249
```

FIGURE 2.8 – Résultat d'exécution du fichier cnf

2.3.4 Après amélioration

On obtien les clause suivant :

- 1 - $(\neg Na \vee Cea); (\neg Nb \vee Ceb); (\neg Nc \vee Cec);$
- 2 - $(\neg Cea \vee Ma); (\neg Ceb \vee Mb); (\neg Cec \vee Mc);$
- 3 - $(\neg Na \vee Coa); (\neg Nb \vee Cob); (\neg Nc \vee Coc);$
- 4 - Na
- 5 - Ceb
- 6 - Mc
- 7 - $(\neg Ma \vee Cea \vee Coa); (\neg Ma \vee \neg Na \vee Coa);$
- 8 - $(\neg Mb \vee Ceb \vee Cob); (\neg Mb \vee \neg Nb \vee Cob);$
- 9 - $(\neg Mc \vee Cec \vee Coc); (\neg Mc \vee \neg Nc \vee Coc);$
- 10 - $(\neg Cea \vee Na \vee \neg Coa);$

11 – $(\neg Ceb \vee Nb \vee \neg Cob)$;

12 – $(\neg Cec \vee Nc \vee \neg Coc)$;

Représentation des clauses dans le fichier CNF

```
1  p cnf 12 21
2  -1 4 0
3  -2 5 0
4  -3 6 0
5  -4 10 0
6  -5 11 0
7  -6 12 0
8  -10 4 7 0
9  -10 -1 7 0
10 -11 5 8 0
11 -11 -2 8 0
12 -12 6 9 0
13 -12 -3 9 0
14 -4 1 -7 0
15 -5 2 -8 0
16 -6 3 -9 0
17 -1 7 0
18 -2 8 0
19 -3 9 0
20 1 0
21 5 0
22 12 0
```

FIGURE 2.9 – contenu du fichier CNF après amélioration

Test SAT

Après amélioration, on peut observer que le système possède une solution (l'ensemble des clauses est satisfiable).

```

#
# Solution found for -target 0

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

Variables = 12
Clauses = 21
TotalLiterals = 48
TotalCPUtimeElapsed = 0.001
FlipsPerSecond = 8001
RunsExecuted = 1
SuccessfulRuns = 1
PercentSuccess = 100.00
Steps_Mean = 8
Steps_CoeffVariance = 0
Steps_Median = 8
CPUtime_Mean = 0.000999927520752
CPUtime_CoeffVariance = 0
CPUtime_Median = 0.000999927520752

```

FIGURE 2.10 – Résultat d'exécution du fichier cnf après amélioration

La solution qui satisfait le système est la suivante :

$N_a \wedge N_b \wedge \neg N_c \wedge C_{ea} \wedge C_{eb} \wedge C_{ec} \wedge C_{oa} \wedge C_{ob} \wedge \neg C_{oc} \wedge M_a \wedge M_b \wedge M_c$

2.3.5 Test de Benchmarks

Pour ce test, nous avons choisi deux fichiers de 250 variables et 1065 clauses. L'un est satisfiable et l'autre non.

Test du fichier satisfiable « uf250-01.cnf »

```
-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

Variables = 250
Clauses = 1065
TotalLiterals = 3195
TotalCPUtimeElapsed = 0.001
FlipsPerSecond = 889852
RunsExecuted = 1
SuccessfulRuns = 1
PercentSuccess = 100.00
Steps_Mean = 890
Steps_CoeffVariance = 0
Steps_Median = 890
CPUtime_Mean = 0.00100016593933
CPUtime_CoeffVariance = 0
CPUtime_Median = 0.00100016593933
```

FIGURE 2.11 – Test de Benchmarks 1

Il existe bien une solution satisfaisant la base de clauses.

Test du fichier non satisfiable « uuf250-01.cnf »

```
# No Solution found for -target 0

Variables = 250
Clauses = 1065
TotalLiterals = 3195
TotalCPUtimeElapsed = 0.051
FlipsPerSecond = 1960780
RunsExecuted = 1
SuccessfulRuns = 0
PercentSuccess = 0.00
Steps_Mean = 100000
Steps_CoeffVariance = 0
Steps_Median = 100000
CPUtime_Mean = 0.0510001182556
CPUtime_CoeffVariance = 0
CPUtime_Median = 0.0510001182556
```

FIGURE 2.12 – Test de Benchmarks 2

pas de solution pour cette base de clauses.

Interprétation des résultats

pour le premier fichier CNF satisfiable, le temps total d'exécution était beaucoup plus court, seulement 0.005 secondes. Le solveur UBCSAT a réussi à trouver un modèle rapidement, avec un pourcentage de réussite de 100%. Cela démontre une performance impressionnante en termes de temps et de succès,

Pour le deuxième fichier CNF non satisfiable, le temps total d'exécution était de 0.050 secondes. Le solveur UBCSAT n'a pas réussi à trouver de modèle satisfaisant. Ce temps relativement long est dû au fait que ne pas trouver de solution est considéré comme étant le pire cas en termes de complexité.

2.4 Étape 4

L'algorithme utilise le Solveur UBCSAT pour évaluer la satisfiabilité d'une base de connaissances. En utilisant le raisonnement par l'absurde, il teste si la base de connaissances BC déduit un but donné. Le processus de l'algorithme se déroule comme suit : tout d'abord, il vérifie si le fichier contenant la base de connaissances existe. Ensuite, il récupère la base de connaissances qui copie dans un nouveau fichier en incrémentant de 1 le nombre de clauses. Ensuite, l'algorithme récupère le but entré par l'utilisateur. Le complément du but est ensuite écrit dans le fichier précédemment créé, suivi de l'ajout de "0" à la fin de la nouvelle clause. Le Solveur SAT est ensuite exécuté sur le fichier créé. Si le Solveur SAT indique que la base de connaissances est non satisfiable, l'algorithme affiche "La base de connaissances est non satisfiable la BF infere le But" ; sinon, il affiche "La base de connaissances est satisfiable la BF n'infere pas le But".

2.4.1 Partie 1

Dans cette partie, nous allons d'abord expérimenter avec l'exemple de la base de faits améliorée du cours.

But : Coc

```
Entrez le nom du fichier contenant la base de connaissances: Zoo2.cnf
1: Na;      2: Nb;      3: Nc;
4: Cea;     5: Ceb;     6: Cec;
7: Coa;     8: Cob;     9: Coc;
10: Ma;     11: Mb;     12: Mc;

Entrez le littéral 1 (précédé de '-' si négatif): 9
La base de connaissances est non satisfiable la BF infere le But.
```

FIGURE 2.13 – Exécution du programme avec entré égale a 9

But : Coa

```
D:\Etude S2\rep connaissance\ubcsat\ubcsat v2>python etap4.py
Entrez le nom du fichier contenant la base de connaissances: Zoo2.cnf
1: Na;          2: Nb;          3: Nc;
4: Cea;         5: Ceb;         6: Cec;
7: Coa;         8: Cob;         9: Coc;
10: Ma;         11: Mb;         12: Mc;

Entrez le littéral 1 (précédé de '-' si négatif): 7
La base de connaissances est satisfiable la BF n'infere pas le But.
```

FIGURE 2.14 – Exécution du programme avec entrée égale a 7

Les résultats du test confirment les enseignements du cours : Aucune solution n'est trouvée pour le Nnon Coc, ce qui signifie que Coc est vrai pour tous les modèles et est donc prouvé. Cependant, il existe des modèles où Coa est vrai et d'autres où Coa est faux. Cela indique que Coa n'est pas prouvé.

2.4.2 Partie 2

Dans cette partie, nous allons voir un exemple de problème SAT pour la recherche de vulnérabilités dans des algorithmes de chiffrement.

Voici un exemple simplifié de problème SAT avec 20 clauses et 20 variables :

(C1) $\neg K1 \vee \neg K2 \vee \neg K3$

(C2) $K4 \vee K5 \vee K6$

(C3) $\neg K7 \vee \neg K8 \vee K9$

(C4) $K10 \vee K11 \vee K12$

(C5) $K13 \vee \neg K14 \vee K15$

(C6) $\neg K16 \vee K17 \vee K18$

(C7) $K19 \vee \neg K20 \vee \neg K1$

(C8) $\neg K2 \vee \neg K3 \vee K4$

(C9) $K5 \vee K6 \vee \neg K7$

(C10) $\neg K8 \vee K9 \vee K10$

(C11) $K11 \vee K12 \vee K13$

(C12) $\neg K14 \vee K15 \vee \neg K16$

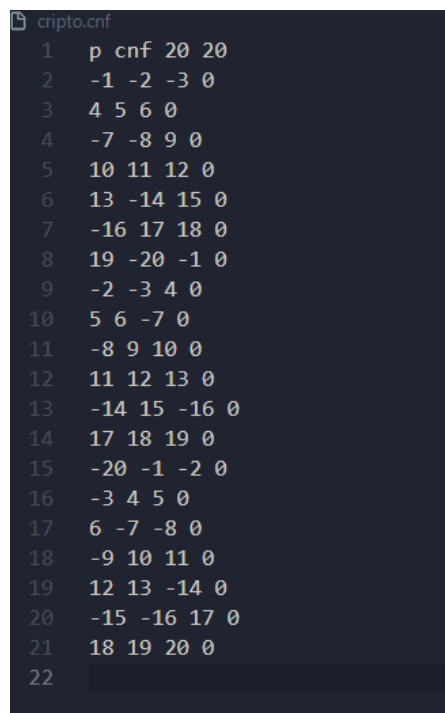
(C13) $K17 \vee K18 \vee K19$

(C14) $\neg K20 \vee \neg K1 \vee \neg K2$

(C15) $\neg K3 \vee K4 \vee K5$

$(C16) K6 \vee \neg K7 \vee \neg K8$
 $(C17) \neg K9 \vee K10 \vee K11$
 $(C18) K12 \vee K13 \vee \neg K14$
 $(C19) \neg K15 \vee \neg K16 \vee K17$
 $(C20) K18 \vee K19 \vee K20$

Représentation des clauses dans le fichier CNF



```

cripto.cnf
1  p cnf 20 20
2  -1 -2 -3 0
3  4 5 6 0
4  -7 -8 9 0
5  10 11 12 0
6  13 -14 15 0
7  -16 17 18 0
8  19 -20 -1 0
9  -2 -3 4 0
10 5 6 -7 0
11 -8 9 10 0
12 11 12 13 0
13 -14 15 -16 0
14 17 18 19 0
15 -20 -1 -2 0
16 -3 4 5 0
17 6 -7 -8 0
18 -9 10 11 0
19 12 13 -14 0
20 -15 -16 17 0
21 18 19 20 0
22

```

FIGURE 2.15 – Contenu du fichier cripto

Résultat d'exécution du programme

Après l'exécution pour toutes les variables de K, allant de 1 à 20, le résultat de l'exécution est le suivant :

```
Entrez le littéral 1 (précédé de '-' si négatif): 2  
La base de connaissances est non satisfiable la BF infere le But.
```

FIGURE 2.16 – résultat d’exécution avec le fichier crypto

2.5 Conclusion

En conclusion, cette étude sur les solveurs SAT, en particulier UBCSAT, ainsi que sur les principes d’inférence logique et de raisonnement par l’absurde, nous a permis de mieux comprendre les mécanismes fondamentaux de la résolution de problèmes logiques. Nous avons pu expérimenter avec des bases de faits au format CNF et observer comment les méthodes de raisonnement par l’absurde peuvent être utilisées pour tester l’inférence de ces bases.