

Rapport TP1
Représentation des Connaissances et Raisonnement 1
Inférence logique basée sur un solveur SAT

BOUROUINA Rania
181831052716

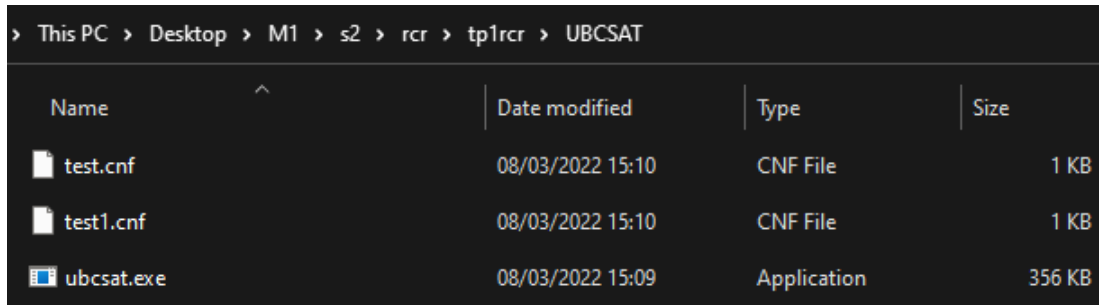
CHIBANE Ilies
181831072041

Résumé

Dans ce premier TP, nous allons d'abord utiliser un solveur SAT pour étudier la satisfiabilité de quelques Bases de Connaissance. Nous allons également traduire une BC relative aux connaissances zoologiques, tester sur des Benchmarking et simuler l'inférence d'une BC avec un algorithme.

Étape 1 : Création du répertoire

Dans l'étape 1, on crée un dossier contenant le SAT UBCSAT et on y copie des fichiers sous format CNF



Name	Date modified	Type	Size
test.cnf	08/03/2022 15:10	CNF File	1 KB
test1.cnf	08/03/2022 15:10	CNF File	1 KB
ubcsat.exe	08/03/2022 15:09	Application	356 KB

FIGURE 1 – Le répertoire UBCSAT

Étape 2 : Exécution du solveur SAT

Dans l'étape 2, on exécute le solveur SAT et teste de la satisfiabilité des deux fichiers : test.cnf et test1.cnf

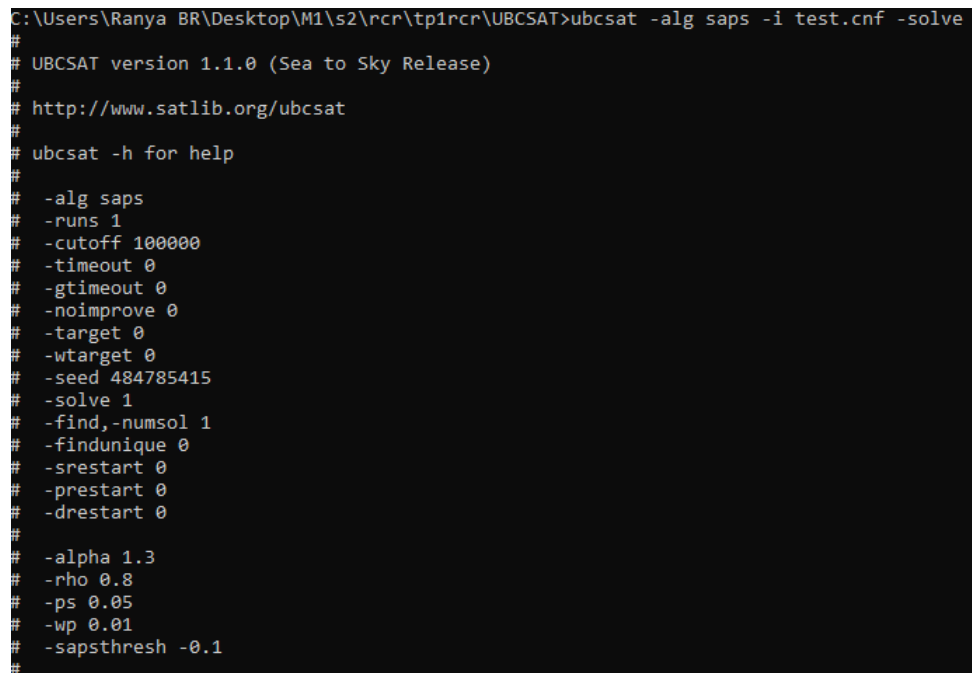
3.1 Le fichier Test.cnf

Pour l'essai numéro 1 nous avons exécuté la commande suivante (avec test.cnf le fichier contenant les règles SAT).

```
1 ubcsat -alg saps -i test.cnf -solve
```

Listing 1 – SAT test

Voici le résultat.



```
C:\Users\Ranya BR\Desktop\M1\s2\rcr\tp1rcr\UBCSAT>ubcsat -alg saps -i test.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
# -gtimout 0
# -noimprove 0
# -target 0
# -wtargt 0
# -seed 484785415
# -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 – Résultats du solveur SAT, partie 1 (test.cnf)

Cet affichage représente les informations sur la version de l'UBCSAT, un site et l'instruction d'aide, ainsi que plusieurs paramètres de notre Solveur.

```

# 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      7      7
#
#

```

FIGURE 3 – Résultats du solveur SAT, partie 2 (test.cnf)

L’affichage de la partie 2 représente le rapport de résultat d’exécution (sortie).

```

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

```

FIGURE 4 – Résultats du solveur SAT, partie 3 (solution test.cnf)

Cette partie affiche la sortie de l’UBCSAT. Puisque le solveur a trouvé une solution, nous pouvons dire que le fichier « test.cnf » est satisfiable. La solution trouvée est :

$$a \vee \neg b \vee \neg c \vee \neg d \vee \neg e \quad (1)$$

```

Variables = 5
Clauses = 11
TotalLiterals = 27
TotalCPUtimeElapsed = 0.001
FlipsPerSecond = 7001
RunsExecuted = 1
SuccessfulRuns = 1
PercentSuccess = 100.00
Steps_Mean = 7
Steps_CoeffVariance = 0
Steps_Median = 7
CPUtime_Mean = 0.000999927520752
CPUtime_CoeffVariance = 0
CPUtime_Median = 0.000999927520752

```

FIGURE 5 – Résultats du solveur SAT, partie 4 (rapport test.cnf)

La dernière partie, représente un rapport de statistiques de l’exécution. Parmi les paramètres contenus ici que nous avons vu en cours il y a, le nombre de variables (ici 5) ainsi que le nombre de clauses (ici 11). Il y a aussi le pourcentage de réussite (PercentSuccess) qui est 100% car la base de connaissance est satisfiable et donc exploitable.

3.2 Le fichier Test1.cnf

Pour Le fichier test1.cnf, Nous avons exécuté la commande suivantes (avec test1.cnf le fichier contenant les règles SAT du deuxième exemple donné dans le TP).

```
1 ubcsat -alg saps -i test1.cnf -solve
```

Listing 2 – SAT

Voici le résultats.

```
C:\Users\Ranya BR\Desktop\M1\s2\rcr\tp1rcr\UBCSAT>ubcsat -alg saps -i test1.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
# -wtarget 0
# -seed 485827900
# -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 6 – Résultats du solveur SAT pour le deuxième exemple (test1.cnf)

```
# 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 = 13
TotalLiterals = 29
TotalCPUtimeElapsed = 0.009
FlipsPerSecond = 11111328
RunsExecuted = 1
SuccessfulRuns = 0
PercentSuccess = 0.00
Steps_Mean = 100000
```

FIGURE 7 – Résultats du solveur SAT pour le deuxième exemple (test1.cnf, aucune solution trouvée)

Après l'exécution du solveur sur "test2.cnf", le message (No Solution found) s'affiche signifiant que la base n'est pas satisfiable et donc non exploitable. Notez que PercentSuccess ici est 0.00.

Étape 3 :

Ci-dessous, les énoncés vu en cours :

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

4.1 Partie 1 : Traduction la base de connaissances

Nos non logiques :

Na, Nb, Nc ; Où Na = Nautilus de a

Cea, Ceb, Cec ; Où Cea = Céphalopode de a

Ma, Mb, Mc ; Où Ma = Mollusque de a

Coa, Cob, Coc ; Où Coa = Coquille de a

Même chose en ce qui concerne b et c.

En ignorant les connaissances utilisant le mot « généralement » (vu en cours) :

1 - Les nautilus sont des céphalopodes.

$$(Na \supset Cea); (Nb \supset Ceb); (Nc \supset Cec);$$

2 - Les céphalopodes sont des mollusques.

$$(Cea \supset Ma); (Ceb \supset Mb); (Cec \supset Mc);$$

4- Les mollusques ont une coquille. on a enlevé le mot généralement

$$(Ma \supset Coa); (Mb \supset Cob); (Mc \supset Coc);$$

5- Les nautilus en ont une.(coquille)

$$(Na \supset Coa); (Nb \supset Cob); (Nc \supset Coc)$$

6- Les céphalopodes n'en ont pas.(coquille) on a enlevé le mot généralement

$$(Cea \supset \neg Coa); (Ceb \supset \neg Cob); (Cec \supset \neg Coc)$$

7- a est un nautilus, b est un céphalopode, c est un mollusque.

$$Na; Ceb; Mc$$

Si on ignore totalement le mot « généralement », notre système sera incohérent. Donc on procède à la transformation de ces clauses en CNF en transformant l'implication en une disjonction :

Solution : Application de la règle : $a \supset b \equiv \neg a \vee b$

7- a est un nautilus, b est un céphalopode, c est un mollusque.

Na

Ceb

Mc

1 - Les nautilus sont des céphalopodes.

$$(\neg Na \vee Cea); (\neg Nb \vee Ceb); (\neg Nc \vee Cec)$$

2 - Les céphalopodes sont des mollusques.

$$(\neg Cea \vee Ma); (\neg Ceb \vee Mb); (\neg Cec \vee Mc)$$

5- Les nautilus en ont une.(coquille)

$$(\neg Na \vee Coa); (\neg Nb \vee Cob); (\neg Nc \vee Coc)$$

4- Les mollusques ont une coquille. (on a enlevé le mot généralement)

$$(\neg Ma \vee Coa); (\neg Mb \vee Cob); (\neg Mc \vee Coc)$$

6- Les céphalopodes n'en ont pas.(coquille) on a enlevé le mot généralement

$$(\neg Cea \vee \neg Coa); (\neg Ceb \vee \neg Cob); (\neg Cec \vee \neg Coc)$$

On remplace les clauses qui comportaient le mot « généralement » avec une autre interprétation (les clauses 4 et 6)

en utilisant les autres clauses :

$$(\neg Na \vee Cea); (\neg Nb \vee Ceb); (\neg Nc \vee Cec)$$

$$(\neg Cea \vee Ma); (\neg Ceb \vee Mb); (\neg Cec \vee Mc)$$

$$(\neg Na \vee Coa); (\neg Nb \vee Cob); (\neg Nc \vee Coc)$$

$$(\neg Ma \vee Cea \vee Coa); (\neg Ma \vee \neg Na \vee Coa)$$

$$(\neg Mb \vee Ceb \vee Cob); (\neg Mb \vee \neg Nb \vee Cob)$$

$$(\neg Mc \vee Cec \vee Coc); (\neg Mc \vee \neg Nc \vee Coc)$$

$$(\neg Cea \vee Na \vee \neg Coa)$$

$$(\neg Ceb \vee Nb \vee \neg Cob)$$

$$(\neg Cec \vee Nc \vee \neg Coc)$$

$$Na; Ceb; Mc$$

Nous avons 12 variables et 21 clauses au total. Représentation dans le fichier :

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

```

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

```

FIGURE 8 – Contenu du fichier cnf

NB : nous avons 12 variables et 21 clauses. Si nous n'avons pas rajouté les autres clauses en réinterprétant les clauses avec «généralement», la BC n'aurait pas été satisfiable avec les 18 clauses initiales et donc non exploitable.


```

#      F Best      Step      Total
#      Run N Sol'n    of      Search
#      No. D Found    Best    Steps
#
#      1 1      0      12      12
#
# Solution found for -target 0
#
1 -2 -3 4 5 6 7 -8 -9 10
11 12
#
Variables = 12
Clauses = 21
TotalLiterals = 48
TotalCPUtimeElapsed = 0.002
FlipsPerSecond = 6000
RunsExecuted = 1
SuccessfulRuns = 1
PercentSuccess = 100.00
Steps_Mean = 12
Steps_CoeffVariance = 0
Steps_Median = 12
CPUtime_Mean = 0.0019998550415
CPUtime_CoeffVariance = 0
CPUtime_Median = 0.0019998550415
C:\Users\Ranya BR\Desktop\M1\s2\rcr\tp1rcr\UBCSAT>

```

FIGURE 9 – Résultats du solveur SAT pour le fichier

Solution : $Na \wedge \neg Nb \wedge \neg Nc \wedge Cea \wedge Ceb \wedge CeC \wedge Coa \wedge \neg Cob \wedge \neg CoC \wedge Ma \wedge Mb \wedge Mc$

4.1.a Teste de Benchmarking

On télécharge deux fichiers : un satisfiable et un non-satisfiable, et nous le testons.

```

C:\Users\Ranya BR\Desktop\M1\s2\rcr\tp1rcr\UBCSAT>ubcsat -alg saps -i uuf75-01.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
# -wtarget 0
# -seed 496330784
# -solve 1
# -find,-numsol 1
# -findunique 0
# -srestart 0
# -prestart 0
# -drestart 0

```

FIGURE 10 – Résultats du solveur SAT pour le premier fichier, partie 1

```

1 0 1 895 100000
# No Solution found for -target 0

Variables = 75
Clauses = 325
TotalLiterals = 975
TotalCPUtimeElapsed = 0.039
FlipsPerSecond = 2564100
RunsExecuted = 1
SuccessfulRuns = 0
PercentSuccess = 0.00
Steps_Mean = 100000
Steps_CoeffVariance = 0

```

FIGURE 11 – Résultats du solveur SAT pour le premier fichier, partie 2

La base est donc non satisfiable.

```

C:\Users\Ranya BR\Desktop\M1\s2\rcr\tp1rcr\UBCSAT>ubcsat -alg saps -i uf75-01.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
# -gtimeout 0
# -noimprove 0
# -target 0
# -wtarget 0
# -seed 496807074
# -solve 1
# -find,-numsol 1
# -findunique 0
# -srestart 0
# -prestart 0
# -drestart 0
#

```

FIGURE 12 – Résultats du solveur SAT pour le deuxième fichier, partie 1

```

#
# 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      144      144
#
# 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

```

FIGURE 13 – Résultats du solveur SAT pour le deuxième fichier, partie 2

La base est satisfiable.

Étape 4 : simulation de l'inférence d'une base de connaissances

En utilisant le le raisonnement par l'absurde, nous allons tester si une BC infère un but donné en optant pour le solveur UBSAT pour le teste de satisfiabilité d'une base.

5.1 Déroulement

- On prend le fichier de la BC choisie.
- On ouvre un fichier temporaire (pour copier la BC) et on y met les information de la BC (nombre de variable et nombre de clauses +1).
- Nous avons incrémenté le nombre de clauses pour rajouter le non_but dans le fichier à traiter (avec un "0" à la fin comme toutes les clauses).
- On execute les Solveur SAT et on affiche le résultat.

5.2 Tests

```
PS C:\Users\Ranya BR\Desktop\MI\s2\rcr\tp1rcr\UBCSAT> gcc main.c -o main.exe -stdc++
PS C:\Users\Ranya BR\Desktop\MI\s2\rcr\tp1rcr\UBCSAT> ./main
Entrez le nom de votre Fichier BC (sans extension) :
Zootest
Liste des variables de la BC (littereaux):
1: Na;      2: Nb;      3: Nc;
4: Cea;     5: Ceb;     6: Cec;
7: Coa;     8: Cob;     9: Coc;
10: Ma;     11: Mb;     12: Mc;

Donnez le nombre de littereaux :
1
Entrez le literal 1 :
1

BC U {Non but} est non satisfiable. La base n'infere pas le but.
-1 ne peut pas etre atteint
PS C:\Users\Ranya BR\Desktop\MI\s2\rcr\tp1rcr\UBCSAT> []
```

FIGURE 14 – Résultats du programme, cas d'echec

```
PS C:\Users\Ranya BR\Desktop\MI\s2\rcr\tp1rcr\UBCSAT> ./main
Entrez le nom de votre Fichier BC (sans extension) :
solu
Liste des variables de la BC (littereaux):
1: Na;      2: Nb;      3: Nc;
4: Cea;     5: Ceb;     6: Cec;
7: Coa;     8: Cob;     9: Coc;
10: Ma;     11: Mb;     12: Mc;

Donnez le nombre de littereaux :
1
Entrez le literal 1 :
3

BC U {Non but} est satisfiable. La base infere le but.
Solution :
1 -2 3 4 5 6 7 -8 9 10
11 12
```

FIGURE 15 – Résultats du programme, cas de réussite

5.3 Code Source

```
1
2 #include <stdio.h>
3 #include <stdlib.h>
```

```

4 #include <string.h>
5
6 int main(){
7     FILE * fich_base = NULL;
8     FILE * fich_temp = NULL;
9     int nbr_propositions, nbr_variables, nbr_clauses, i, but[20], non_but[20];
10    char nom_base[20], c;
11
12
13    printf("Entrez le nom de votre Fichier BC (sans extension) :\n");
14    gets(nom_base);
15    strcat(nom_base, ".cnf");
16    fich_base = fopen(nom_base, "r+");
17
18    if(fich_base == NULL)
19        printf("Impossible d'ouvrir BC...\n");
20    else{
21
22        fich_temp = fopen("Temp.cnf", "rw+");
23
24        if(fich_temp == NULL)
25            printf("Impossible de transferer BC \n");
26        else{
27            fscanf(fich_base, "p cnf %d %d ", &nbr_variables, &nbr_clauses); // on prend la premi re
28            ligne de la BC
29            nbr_clauses += 1 ; // pour rajouter le non_but
30            fprintf (fich_temp, "p cnf %d %d ""\n", nbr_variables, nbr_clauses); // on met les infos
31            dans le fichier qu'on va traiter
32        }
33        c = fgetc(fich_base);
34        while(c != EOF){
35            c = fgetc(fich_base);
36            if(c != EOF) fputc(c, fich_temp);
37        } // on a recopier le reste de la BC dans le fichier qu'on va traiter
38
39        printf("Liste des variables de la BC (litteraux):\n "); //afficher les codes pour l'utilisateur
40        printf("1: Na;\t\t 2: Nb;\t\t 3: Nc;\t \n 4: Cea;\t 5: Ceb;\t 6: Cec;\t \n 7: Coa;\t 8: Cob;\t
41        9: Coc;\t \n 10: Ma;\t 11: Mb;\t 12: Mc;\t\n\n");
42        printf("Donnez le nombre de litteraux : \n"); //prendre les buts inferer
43        scanf("%d", &nbr_propositions);
44
45        for(i = 1; i < nbr_propositions + 1; i++){
46
47            printf("Entrez le literal %d : \n", i);
48            scanf("%d", &but[i]);
49            if(but[i] > -13 && but[i] < 13)
50                non_but[i] = but[i] * (-1); //si les codes sot corrects on prend la negation du but (
51            absurde)
52            else
53                puts("Erreur, Vous avez entrer un code invalide");
54        }
55
56        fprintf(fich_temp, "\n"); //completion du fichier traiter
57        for(i = 1; i < nbr_propositions + 1; i++) fprintf(fich_temp, "%d ", non_but[i]); //Ajout des negations
58        au fichier
59        fprintf(fich_temp, "0"); //Ajout des 0 pour marquer la fin
60
61        system("ubcsat -alg saps -i Temp.cnf -solve > results.txt"); //execution solveur
62    }
63    fclose(fich_temp);
64
65    int termine = 0; //signaler fin affichage
66    FILE *fich = fopen("results.txt", "r+");
67    if(fich_base == NULL) printf("Impossible d'accéder aux resultats...\n");
68    else{
69        char texte[1000];
70        while(fgets(texte, 1000, fich) && !termine){
71            if(strstr(texte, "# Solution found for -target 0")){

```

```

71         printf("\n BC U {Non but} est satisfiable. La base infere le but. \n Solution : \n")
72     ;
73     fscanf(fich, "\n");
74     while(!strstr(fgets(texte, 1000, fich), "Variables"))
75         printf("%s", texte);
76     termine = 1;
77 }
78 if(termine == 0){ //cas de non satisfiablit
79     printf("\n BC U {Non but} est non satisfiable. La base n'infere pas le but.\n");
80     int j;
81     for(j = 1;j<nbr_propositions+1;j++){
82         printf("%d ",(-1)*but[j]);
83     }
84     if(j>2)
85         printf("ne peuvent pas etre atteints");
86     else
87         printf("ne peut pas etre atteint\n");
88 }
89 }
90
91 fclose(fich_temp);
92 }

```