



UNIVERSITÉ DES SCIENCES ET DE LA TECHNOLOGIE
HOUARI BOUMEDIENE FACULTÉ INFORMATIQUE

TP REPRÉSENTATION DES CONNAISSANCES ET LE
RAISONNEMENT 1

TPs RCR

Étudiants :
Manel OUCHAR
Zakaria YOUSFI
SII G03

Table des matières

1	TP1 : Solveur SAT	5
1.1	Etape 1	5
1.1.1	Création du répertoire et des fichiers	5
1.2	Etape 2	6
1.2.1	Exécution du fichier test1.cnf	6
1.2.2	Exécution du fichier test2.cnf	6
1.3	Etape 3	9
1.3.1	Traduction de la base de connaissances zoologiques	9
1.3.2	Test de satisfiabilité de la base	11
1.3.3	Test des fichiers Benchmarks	12
1.3.3.1	Le fichier uf100-0430.cnf	12
1.3.3.2	Le fichier uuf100-0430.cnf	13
1.4	Etape 4	14
1.4.1	Simulation d'inférence d'une base de connaissances	14
1.4.1.1	Code source du programme	15
1.4.1.2	Tests du programme	17
2	TP2 : Logique des prédicats	19

2.1	La librairie Java Tweety	19
2.2	Exemple de logique des prédicats	19
2.3	Code source et tests	20
3	TP3 : Logique modale	22
3.1	Modèle modal	22
3.2	Code source et tests	23
4	TP4 : Logique des défauts	26
4.1	Exercice	26
4.2	Code Source	27
4.3	Exécution	30
5	TP5 : Réseaux sémantiques	32
5.1	Partie 01 : Algorithme de propagation de marqueurs	32
5.1.1	Algorithme	33
5.1.2	Exécution	34
5.2	Partie 02 : Algorithme d'héritage	35
5.2.1	Algorithme	35
5.2.2	Exécution	36
5.3	Partie 03 : Algorithme de propagation de marqueurs cas d'exceptions . . .	37
5.3.1	Algorithme	37
5.3.2	Exécution	38
6	TP6 : Logique de description	40
6.1	Web Protégé	40

Table des figures

1.1	Répertoire UBCSAT	5
1.2	Exécution du solveur SAT test1.cnf	7
1.3	Exécution du solveur SAT test2.cnf	8
1.4	Le fichier zoo.cnf	11
1.5	Solution de la base de connaissances zoologiques	12
1.6	Téléchargement des fichiers benchmarks.	13
1.7	Solution du fichier uf100-0430.cnf	13
1.8	Solution du fichier uuf100-0430.cnf	14
1.9	Succès.	17
1.10	Echec.	18
2.1	Résultats d'exécution du programme de logique du premier ordre	21
3.1	Exemple de modèle modal	23
3.2	Résultats d'exécution du programme de logique modale	24
4.1	Enoncé exercice 1	27
4.2	Résultat d'exécution de l'outil DefaultLogic sur l'exercice 1	31
5.1	Réseau sémantique utilisé.	34

5.2	Résultats de l'exécution de l'algorithme du propagation des marqueurs sur un réseau sémantique.	35
5.3	Réseau sémantique du le cours page 11.	36
5.4	Résultats d'exécution de l'algorithme d'heritage.	37
5.5	Réseau sémantique de la plateforme khzour modifié.	39
5.6	Résultats d'exécution de l'algorithme d'exceptions.	39
6.1	Interface de l'outil web protégé.	41
6.2	Les Concepts.	42
6.3	Les Roles.	43
6.4	Les individus.	44
6.5	Informations sur les individus.	45
6.6	Schéma d'un individu.	46

Chapitre 1

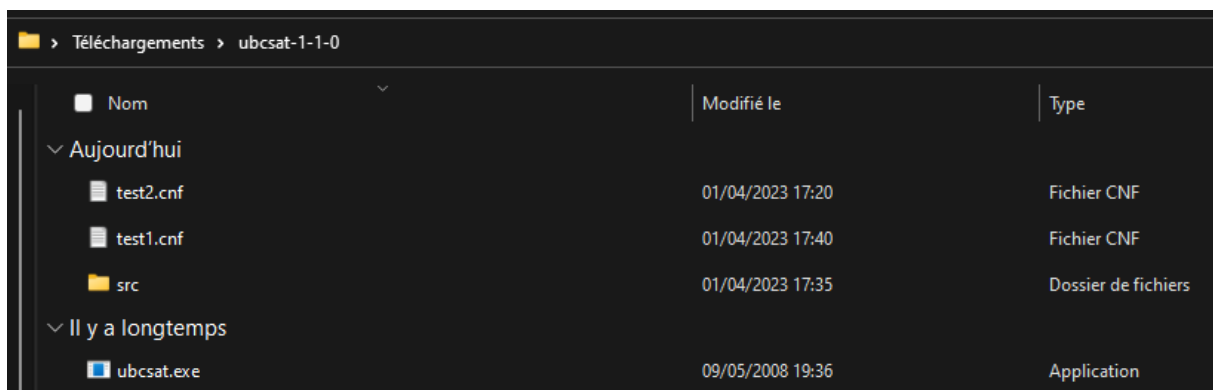
TP1 : Solveur SAT

Dans ce TP1, nous allons commencer par utiliser un solveur SAT afin d'examiner la satisfaisabilité de plusieurs bases de connaissances. En outre, nous allons traduire une base de connaissances portant sur les connaissances **zoologiques**, tester cela sur des référentiels de performance et simuler l'inférence d'une base de connaissances à l'aide d'un algorithme.

1.1 Etape 1

1.1.1 Création du répertoire et des fichiers

On commence par créer un dossier, puis on y copie le fichier ubcsat (le solveur) ainsi que les deux fichiers test1.cnf et test2.cnf.



Nom	Modifié le	Type
test2.cnf	01/04/2023 17:20	Fichier CNF
test1.cnf	01/04/2023 17:40	Fichier CNF
src	01/04/2023 17:35	Dossier de fichiers
ubcsat.exe	09/05/2008 19:36	Application

FIGURE 1.1 – Répertoire UBCSAT

1.2 Etape 2

Maintenant, nous allons tester la satisfiabilité des bases de connaissances des deux fichiers test1.cnf et test2.cnf. Pour cela, on va exécuter le solveur SAT.

1.2.1 Exécution du fichier test1.cnf

Pour ce fichier, on exécute la commande suivante :

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

Listing 1.1 – Commande d'exécution du solveur SAT pour le fichier test1.cnf

Une fois la commande exécutée, des informations et des paramètres sur l'UBCSAT s'affichent, ainsi qu'une solution : **1 2 -3 -4 5** qui correspond à la solution : $\mathbf{a} \wedge \mathbf{b} \wedge \neg \mathbf{c} \wedge \neg \mathbf{d} \wedge \mathbf{e}$

On voit également des informations relatives à l'exécution tels que : le nombre de variables, le nombre de clauses dans la base, le temps d'exécution, le pourcentage de succès qui ici est à 100 car la base de connaissance est satisfiable.

1.2.2 Exécution du fichier test2.cnf

Pour ce fichier, on exécute la commande suivante :

```
1 ubcsat -alg saps -i test2.cnf -solve
```

Listing 1.2 – Commande d'exécution du solveur SAT pour le fichier test1.cnf

Une fois la commande exécutée, des informations et des paramètres sur l'UBCSAT s'affichent, ainsi qu'une solution : **1 2 -3 -4 5** qui correspond à la solution : $\mathbf{a} \wedge \mathbf{b} \wedge \neg \mathbf{c} \wedge \neg \mathbf{d} \wedge \mathbf{e}$

On voit également des informations relatives à l'exécution tels que : le nombre de variables, le nombre de clauses dans la base, le temps d'exécution, le pourcentage de succès qui ici est à 100 car la base de connaissance est satisfiable.

```

#
# 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      3      3
#
# Solution found for -target 0

1 2 -3 -4 5

Variables = 5
Clauses = 9
TotalLiterals = 23
TotalCPUtimeElapsed = 0.001
FlipsPerSecond = 3000
RunsExecuted = 1
SuccessfulRuns = 1
PercentSuccess = 100.00
Steps_Mean = 3
Steps_CoeffVariance = 0
Steps_Median = 3
CPUtime_Mean = 0.000999927520752
CPUtime_CoeffVariance = 0
CPUtime_Median = 0.000999927520752

```

FIGURE 1.2 – Exécution du solveur SAT test1.cnf


```

# 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      5      5
#
# Solution found for -target 0

1 2 -3 -4 5

Variables = 5
Clauses = 11
TotalLiterals = 27
TotalCPUtimeElapsed = 0.000
FlipsPerSecond = 1
RunsExecuted = 1
SuccessfulRuns = 1
PercentSuccess = 100.00
Steps_Mean = 5
Steps_CoeffVariance = 0
Steps_Median = 5
CPUtime_Mean = 0
CPUtime_CoeffVariance = 0
CPUtime_Median = 0

```

FIGURE 1.3 – Exécution du solveur SAT test2.cnf

1.3 Etape 3

1.3.1 Traduction de la base de connaissances zoologiques

La base est la suivante :

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

Soient Na, Nb, Nc, Cea, Ceb, Cec, Ma, Mb, Mc, Coa, Cob, Coc 12 symboles non logiques que nous interprétons par "a (respectivement b, c) est un nautilus", "(respectivement, céphalopode, mollusque, coquille)". En ignorant pour l'instant les connaissances utilisant le mot "généralement" nous avons :

1. Les nautilus sont des céphalopodes :
 $(Na \supset Cea) ; (Nb \supset Ceb) ; (Nc \supset Cec) ;$
2. Les céphalopodes sont des mollusques :
 $(Cea \supset Ma) ; (Ceb \supset Mb) ; (Cec \supset Mc) ;$
3. Les mollusques ont une coquille :
 $(Ma \supset Coa) ; (Mb \supset Cob) ; (Mc \supset Coc) ;$
4. Les nautilus ont une coquille :
 $(Na \supset Coa) ; (Nb \supset Cob) ; (Nc \supset Coc) ;$
5. Les céphalopodes n'ont pas de coquille :
 $(Cea \supset \neg Coa) ; (Ceb \supset \neg Cob) ; (Cec \supset \neg Coc) ;$
6. a est un nautilus :
 $Na ;$
7. b est un céphalopode :
 $Ceb ;$
8. c est un mollusque :
 $Mc ;$

Si on ajoute les connaissances dans lesquelles figure le mot "généralement" sans prendre en compte la nuance introduite par ce mot, il vient :

- $(Ma \supset Coa); (Mb \supset Cob); (Mc \supset Coc);$
- $(Céa \supset \neg Coa); (Céb \supset \neg Cob); (Céc \supset \neg Coc);$

Par deux applications du modus ponens sur $Céb$, $(Céb \supset Mb)$ puis $(Mb \supset Cob)$ on peut conclure Cob et par $Céb$ et $Céb \supset \neg Cob$ on peut conclure $\neg Cob$. On obtient un système incohérent.

Pour éviter cette incohérence, la seule solution est de modifier la traduction de "généralement", ce qui nous donne :

- Les céphalopodes qui ne sont pas des nautilus n'ont pas de coquille :
 $((Céa \wedge \neg Na) \supset \neg Coa); ((Céb \wedge \neg Nb) \supset \neg Cob); ((Céc \wedge \neg Nc) \supset \neg Coc);$
- Les mollusques qui ne sont pas des céphalopodes non nautilus en ont une :
 $((Ma \wedge \neg (Céa \wedge \neg Na)) \supset Coa); ((Mb \wedge \neg (Céb \wedge \neg Nb)) \supset Cob); ((Mc \wedge \neg (Céc \wedge \neg Nc)) \supset Coc);$

Ce système est cohérent puisqu'il admet des modèles : $Na, Céa, Céb, Ma, Mb, Mc, Coa$ ont la valeur vrai.

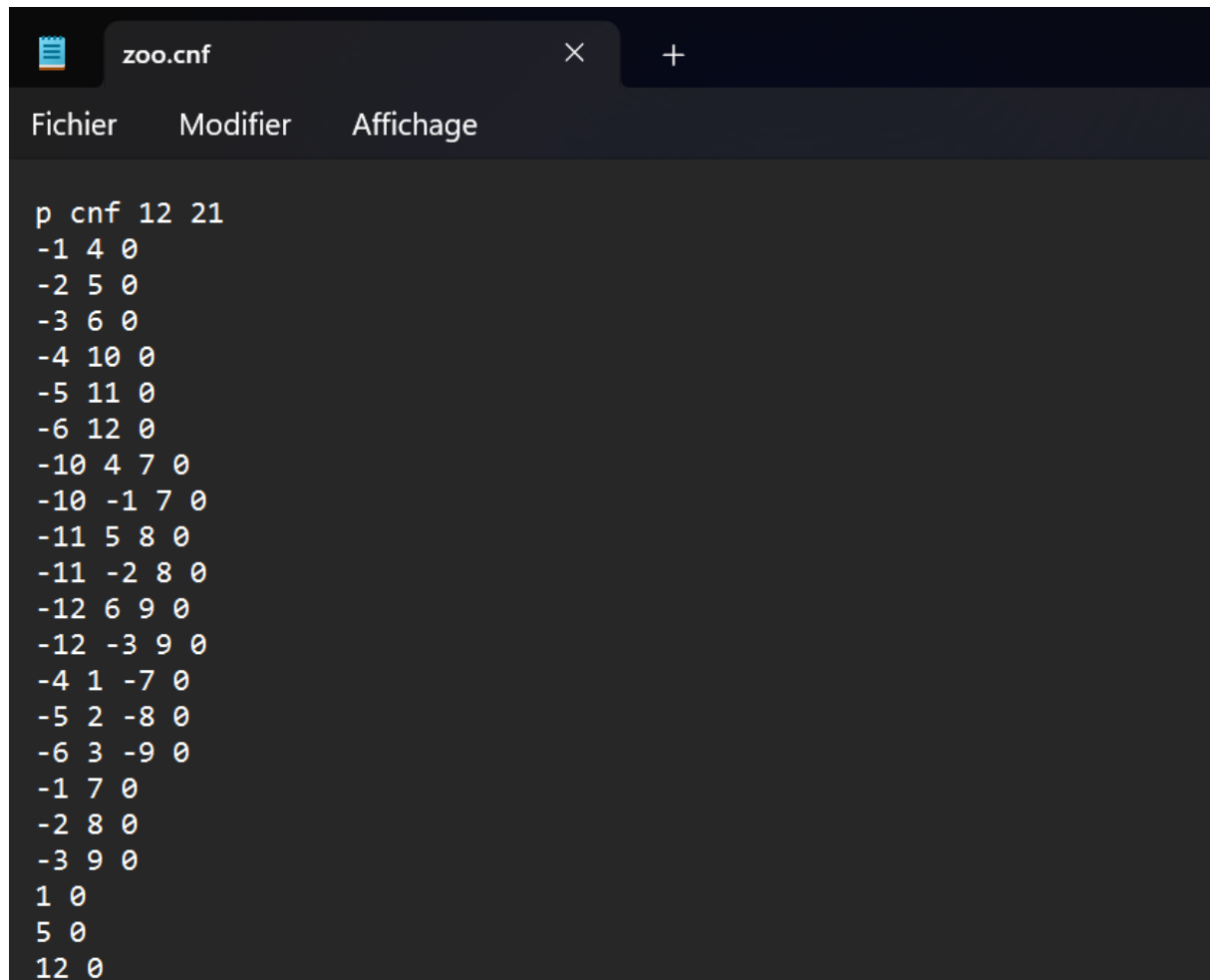
Donc on transforme ces clauses en CNF en transformant l'implication en une disjonction : $a \supset b \equiv \neg a \vee b$. On obtient :

1. Les nautilus sont des céphalopodes :
 $(\neg Na \vee Cea); (\neg Nb \vee Ceb); (\neg Nc \vee Cec);$
2. Les céphalopodes sont des mollusques :
 $(\neg Cea \vee Ma); (\neg Ceb \vee Mb); (\neg Cec \vee Mc);$
3. Les mollusques ont une coquille :
 $(\neg Ma \vee Coa \vee Cea); (\neg Ma \vee \neg Na \vee Coa); (\neg Mb \vee Cob \vee Ceb); (\neg Mb \vee \neg Nb \vee Cob); (\neg Mc \vee Coc \vee Cec); (\neg Mc \vee \neg Nc \vee Coc);$
4. Les nautilus ont une coquille :
 $(\neg Na \vee Coa); (\neg Nb \vee Cob); (\neg Nc \vee Coc);$
5. Les céphalopodes n'ont pas de coquille :
 $(\neg Cea \vee Na \vee \neg Coa); (\neg Ceb \vee Nb \vee \neg Cob); (\neg Cec \vee Nc \vee \neg Coc);$
6. a est un nautilus :
 $Na;$
7. b est un céphalopode :
 $Ceb;$
8. c est un mollusque :
 $Mc;$

1.3.2 Test de satisfiabilité de la base

La base contient 12 variables et 21 clauses qu'on va représenter comme suit : $1 \equiv \text{Na}$, $2 \equiv \text{Nb}$, $3 \equiv \text{Nc}$, $4 \equiv \text{Cea}$, $5 \equiv \text{Ceb}$, $6 \equiv \text{Cec}$, $7 \equiv \text{Coa}$, $8 \equiv \text{Cob}$, $9 \equiv \text{Coc}$, $10 \equiv \text{Ma}$, $11 \equiv \text{Mb}$, $12 \equiv \text{Mc}$,

Par conséquent, on aura la base suivante :



```
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 1.4 – Le fichier zoo.cnf

On exécute le solveur SAT pour ce fichier en utilisant la commande :

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

Listing 1.3 – Commande d'exécution du solveur SAT pour le fichier zoo.cnf

On obtient la solution suivante : **1 2 -3 4 5 -6 7 8 9 10 11 12** qui correspond à la solution :

$$\text{Na} \wedge \text{Nb} \wedge \neg \text{Nc} \wedge \text{Cea} \wedge \text{Ceb} \wedge \neg \text{Cec} \wedge \text{Coa} \wedge \text{Cob} \wedge \text{Coc} \wedge \text{Ma} \wedge \text{Mb} \wedge \text{Mc}$$

Ceci veut dire que la base est satisfiable.

```
# 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      8      8
#
# 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 = 1
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
```

FIGURE 1.5 – Solution de la base de connaissances zoologiques

1.3.3 Test des fichiers Benchmarks

On télécharge des fichiers benchmarks pour tester leurs satisfiabilité.

1.3.3.1 Le fichier uf100-0430.cnf

Cette base contient 100 variables et 430 clauses. On procède au test de sa satisfiabilité en exécutant la commande :

```
1 ubcsat -alg saps -i uf100-0430.cnf -solve
```

Listing 1.4 – Commande d'exécution du solveur SAT pour le fichier uf100-0430.cnf

Le solveur arrive à trouver une solution, ce qui veut dire que cette base est satisfiable.

Nom	Modifié le	Type
zoo.cnf	02/04/2023 19:06	Fichier CNF
Semaine dernière		
test2.cnf	01/04/2023 22:47	Fichier CNF
test1.cnf	01/04/2023 17:40	Fichier CNF
src	01/04/2023 17:35	Dossier de fichiers
Il y a longtemps		
uf100-01000.cnf	18/06/1998 14:05	Fichier CNF
uf100-0999.cnf	18/06/1998 14:08	Fichier CNF
uf100-0998.cnf	18/06/1998 14:08	Fichier CNF
uf100-0997.cnf	18/06/1998 14:08	Fichier CNF
uf100-0996.cnf	18/06/1998 14:08	Fichier CNF
uf100-0995.cnf	18/06/1998 14:08	Fichier CNF
uf100-0994.cnf	18/06/1998 14:08	Fichier CNF

FIGURE 1.6 – Téléchargement des fichiers benchmarks.

```
# 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      121      121
#
# 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

Variables = 100
Clauses = 430
TotalLiterals = 1290
TotalCPUtimeElapsed = 0.002
FlipsPerSecond = 60497
RunsExecuted = 1
SuccessfulRuns = 1
PercentSuccess = 100.00
Steps_Mean = 121
Steps_CoeffVariance = 0
Steps_Median = 121
CPUtime_Mean = 0.00200009346008
CPUtime_CoeffVariance = 0
CPUtime_Median = 0.00200009346008
```

FIGURE 1.7 – Solution du fichier uf100-0430.cnf

1.3.3.2 Le fichier uuf100-0430.cnf

Cette base contient 100 variables et 430 clauses. On procède au test de sa satisfiabilité en exécutant la commande :

```
1 ubcsat -alg saps -i uuf100-0430.cnf -solve
```

Listing 1.5 – Commande d'exécution du solveur SAT pour le fichier uuf100-0430.cnf

Le solveur n'arrive pas à trouver une solution, ce qui veut dire que cette base n'est pas satisfiable.

```
# 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      6108      100000
# No Solution found for -target 0

Variables = 100
Clauses = 430
TotalLiterals = 1290
TotalCPUTimeElapsed = 0.033
FlipsPerSecond = 3030304
RunsExecuted = 1
SuccessfulRuns = 0
PercentSuccess = 0.00
Steps_Mean = 100000
Steps_CoeffVariance = 0
Steps_Median = 100000
CPUTime_Mean = 0.032999923706
CPUTime_CoeffVariance = 0
CPUTime_Median = 0.032999923706
```

FIGURE 1.8 – Solution du fichier uuf100-0430.cnf

1.4 Etape 4

1.4.1 Simulation d'inférence d'une base de connaissances

Afin de simuler l'inférence d'une base de connaissance, nous avons écrit un programme en C qui traite la base de connaissances zoologiques, et qui utilise le solveur SAT "UBC-SAT" pour déterminer si un but est inféré ou non par la base.

Le programme commence par ouvrir le fichier cnf de la BC zoologiques qu'on a créé auparavant dans ce TP, et crée ensuite un fichier temporaire, dans lequel il copie tout le contenu du fichier BC. Il augmente le nombre de clauses de 1 pour y ajouter la négation du but. Le programme demande ensuite à l'utilisateur d'entrer les littéraux du but à inférer, et enregistre leur négation dans le fichier temporaire. Il exécute ensuite le solveur SAT en utilisant le fichier temporaire comme entrée, et stocke les résultats dans un fichier de sortie.

Le programme ouvre ensuite le fichier de sortie, recherche la chaîne de caractères "Solution found for -target 0", qui indique que le solveur a trouvé une solution satisfaisante pour la négation but. Si c'est est le cas, le programme affiche que le but n'est pas inférer par la base. Sinon, cela signifie que la formule est satisfaisable, et le programme indique que la base de connaissances infère le but.

1.4.1.1 Code source du programme

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 int main() {
6
7     int nbr_litteraux, nbr_variables, nbr_clauses, but[30], non_but[30];
8     FILE *fBase = NULL, *fTemp = NULL;
9     char c;
10
11     //ouvrir le fichier de la base zoologique
12     fBase = fopen("C:\\Users\\Asus Zenbook Flip\\Downloads\\ubcsat
-1-1-0\\zoo.cnf", "r");
13
14     if (fBase == NULL) {
15         printf("!Erreur lors de l'ouverture du fichier de base!\n");
16         return 1;
17     }
18
19     //ouvertir du fichier temporaire pour copier le contenu du fichier
de base et le modifier
20     fTemp = fopen("fTemp.cnf", "w+");
21
22     if (fTemp == NULL) {
23         printf("!Erreur lors de la cr ation du fichier temporaire!\n");
24         return 1;
25     }
26
27     fscanf(fBase, "p cnf %d %d", &nbr_variables, &nbr_clauses);
28     //on incr mente le nombre de clauses afin d'ajouter la n gation du
but
29     nbr_clauses += 1;
30
31     //on recopie tout le contenu dans le fichier temporaire
32     fprintf(fTemp, "p cnf %d %d\n", nbr_variables, nbr_clauses);
33     c = fgetc(fBase);
34     while (c != EOF) {
35         fputc(c, fTemp);

```



```

36     c = fgetc(fBase);
37 }
38
39 //On affiche les variables pour que l'utilisateur puisse choisir les
    litt raux du but
40 printf("Liste des variables de la BC :\n");
41 printf("1=Na; 2=Nb; 3=Nc; 4=Cea; 5=Ceb; 6=Cec; 7=Coa; 8=Cob; 9=Coc;
    10=Ma; 11=Mb; 12=Mc;\n\n");
42 printf("Entrez le nombre de litt raux de votre but =\n");
43 scanf("%d", &nbr_litteraux);
44
45 for (int i = 1; i <= nbr_litteraux; i++) {
46     printf("Entrez le litt eral numero %d:\n", i);
47     scanf("%d", &but[i]);
48     if (but[i] > -13 && but[i] < 13)
49         non_but[i] = -1 * but[i];
50     else {
51         printf("Le litt eral que vous avez introduit est faux !\n");
52         return 1;
53     }
54 }
55
56 //ajouter la clause de la n gation du but dans la base du fichier
    temporaire
57 fprintf(fTemp, "\n");
58 for (int i = 1; i <= nbr_litteraux; i++)
59     fprintf(fTemp, "%d ", non_but[i]);
60 fprintf(fTemp, "0\n");
61
62 fclose(fBase);
63 fclose(fTemp);
64
65 system("ubcsat -alg saps -i fTemp.cnf -solve > output.txt");
66
67 FILE *fResult = fopen("output.txt", "r");
68 if (fResult == NULL) {
69     printf("Erreur lors de l'ouverture du fichier de sortie!\n");
70     return 1;
71 }
72
73 //Affichage
74
75 int termine = 0;
76 char ligne[1000];
77
78 while (fgets(ligne, 1000, fResult) && !termine) {
79     if (strstr(ligne, "# Solution found for -target 0")) {

```

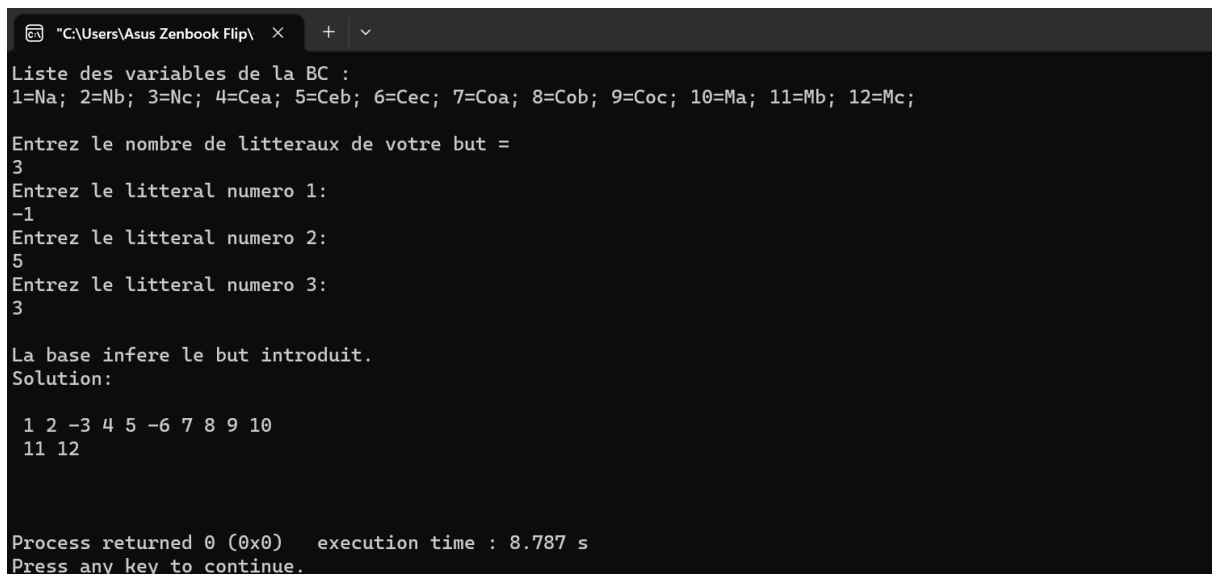
```

80         printf("\nLa base n'inferre pas le but introduit.\nSolution:\n");
81         while (!strstr(fgets(ligne, 1000, fResult), "Variables"))
82             printf("%s", ligne);
83         termine = 1;
84     }
85 }
86
87 fclose(fResult);
88
89 if (!termine)
90     printf("\nLa base inferre le but introduit.\n");
91
92 return 0;
93 }

```

Listing 1.6 – Programme C qui détermine si un but est inféré ou non par une base.

1.4.1.2 Tests du programme



```

"C:\Users\Asus Zenbook Flip\ >
Liste des variables de la BC :
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 nombre de litteraux de votre but =
3
Entrez le litteral numero 1:
-1
Entrez le litteral numero 2:
5
Entrez le litteral numero 3:
3

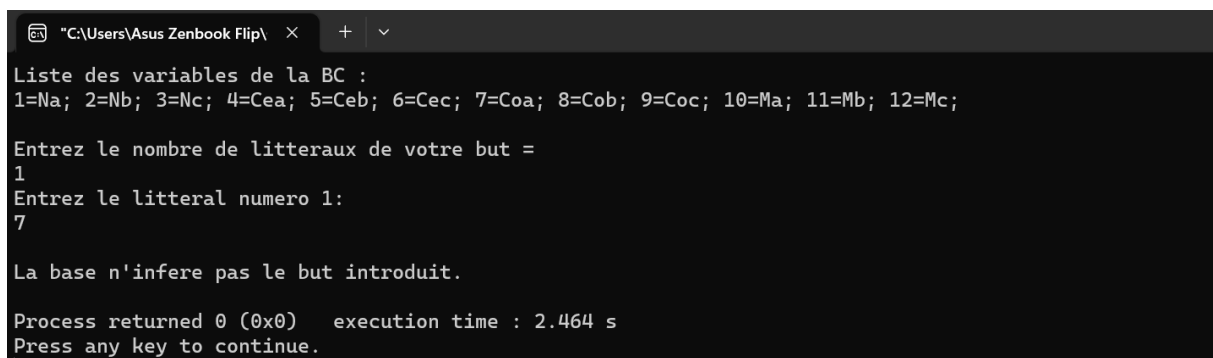
La base inferre le but introduit.
Solution:

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

Process returned 0 (0x0)   execution time : 8.787 s
Press any key to continue.

```

FIGURE 1.9 – Succès.



```
"C:\Users\Asus Zenbook Flip\  ×  +  v
Liste des variables de la BC :
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 nombre de litteraux de votre but =
1
Entrez le litteral numero 1:
7

La base n'infere pas le but introduit.

Process returned 0 (0x0)   execution time : 2.464 s
Press any key to continue.
```

FIGURE 1.10 – Echec.

Chapitre 2

TP2 : Logique des prédicats

Dans ce TP2, nous allons exploiter la librairie Java **Tweety** pour la modélisation des connaissances en logique des prédicats.

2.1 La librairie Java Tweety

Tweety est une bibliothèque Java open-source utilisée pour la modélisation des connaissances. Elle fournit des outils pour la représentation et la manipulation de différents types de connaissances, tels que les connaissances logiques et incertaines. Tweety prend également en charge l'inférence et la résolution de problèmes en utilisant des algorithmes de raisonnement tels que la logique propositionnelle et la logique des prédicats.

2.2 Exemple de logique des prédicats

Afin de d'exploiter cette librairie, on va écrire un programme pour effectuer des raisonnements sur des formules de logique du premier ordre. Pour cela, on doit d'abord définir un exemple de connaissances de cette logique pour qu'on puisse déduire si des formules sont vraies ou fausses.

Soit les connaissances suivantes :

1. Tout homme est mortel : $(\forall x) \text{Homme}(x) \supset \text{Mortel}(x)$;
2. Socrates est un homme : $\text{Homme}(\text{Socrates})$

On veut démontrer à l'aide de ce programme que $\text{Mortel}(\text{Socrates})$ est vraie.

2.3 Code source et tests

Le programme commence par importer les classes nécessaires de la bibliothèque **Tweety** et par définir une signature de FOL (Logique du Premier Ordre), qui est un ensemble de prédicats et de constantes que le programme peut utiliser pour construire des formules FOL.

Ensuite, il crée un ensemble de connaissances appelé BC, qui est utilisé pour stocker les axiomes et les faits de la base de connaissances. Il crée également un objet de type "FolParser", qui est utilisé pour analyser les formules FOL et les transformer en objets Java.

Le programme définit ensuite deux prédicats, "homme" et "mortel", ainsi qu'une constante "socrates", et les ajoute à la signature.

Il ajoute ensuite deux formules à l'ensemble BC. La première formule est une implication qui affirme que si quelque chose est un homme, alors il est mortel. La seconde formule est un fait qui affirme que Socrates est mortel.

Puis, il crée un raisonneur FOL utilisé pour répondre à une requête, qui est de savoir si Socrates est mortel.

Enfin, le programme affiche la réponse obtenue (" $\text{mortel}(\text{socrates})$ is : true") dans la console.

```
1 import java.io.IOException;
2 import org.tweetyproject.logics.common.syntax.Constant;
3 import org.tweetyproject.logics.common.syntax.Predicate;
4 import org.tweetyproject.logics.fol.parser.FolParser;
5 import org.tweetyproject.logics.fol.syntax.*;
6 import org.tweetyproject.logics.fol.reasoner.*;
7
8 public class FirstOrderLogic {
9     public static void main(String[] args) throws IOException {
10
11         FolSignature signature = new FolSignature();
12         FolBeliefSet BC = new FolBeliefSet();
13         FolParser parser = new FolParser();
14
15         Predicate homme = new Predicate("homme", 1);
16         signature.add(homme);
```

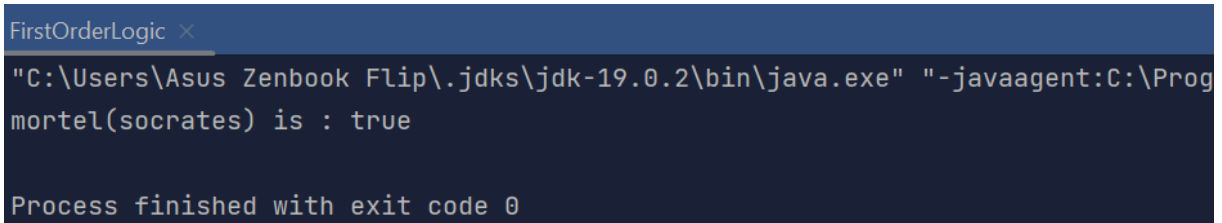
```

17
18     Predicate mortel = new Predicate("mortel", 1);
19     signature.add(mortel);
20
21     Constant socrates = new Constant("socrates");
22     signature.add(socrates);
23
24     BC.setSignature(signature);
25     parser.setSignature(signature);
26
27     BC.add(parser.parseFormula("forall X:(homme(X) => mortel(X)"));
28     BC.add(parser.parseFormula("mortel(socrates)"));
29
30     FolReasoner.setDefaultReasoner(new SimpleFolReasoner());
31     FolReasoner prover = FolReasoner.getDefaultReasoner();
32     System.out.println("mortel(socrates) is : " + prover.query(BC, (
33         FolFormula) parser.parseFormula("mortel(socrates)")));
34 }

```

Listing 2.1 – Programme Java de logique du premier ordre.

L'exécution de ce programme a donner les résultats suivants :



```

FirstOrderLogic x
"C:\Users\Asus Zenbook Flip\.jdk\jdk-19.0.2\bin\java.exe" "-javaagent:C:\Prog
mortel(socrates) is : true

Process finished with exit code 0

```

FIGURE 2.1 – Résultats d'exécution du programme de logique du premier ordre

Le résultat obtenu confirme que Mortel(Socrates) est **vraie**.

Chapitre 3

TP3 : Logique modale

Dans ce TP3, nous allons exploiter la librairie Java **Tweety** pour la modélisation des connaissances en logique modale.

3.1 Modèle modal

Afin de d'exploiter la librairie **Tweety**, on va écrire un programme qui utilise cette bibliothèque pour effectuer des raisonnements sur des formules d'un modèle de logique modale. Pour cela, on doit d'abord définir un modèle modal ainsi que des formules qui sont vraies dans ce dernier pour qu'on puisse déduire si d'autres formules sont vraies ou fausses.

Soit M le modèle modal suivant :

On définit des assertions qui sont vraies dans ce modèle :

1. $M, w4 \models \Box(a \wedge \neg\Diamond\neg b)$: Vrai Dans $w4$, a est vrai et il n'y a pas de monde accessible où $\neg b$ est nécessairement vrai. Par conséquent, $a \wedge \neg\Diamond\neg b$ est vrai dans tous les mondes accessibles à $w4$, ce qui rend $\Box(a \wedge \neg\Diamond\neg b)$ vrai dans $w4$.
2. $M, w1 \models \Diamond\neg a$: Vrai Dans $w1$, $\neg a$ est vrai. Il n'y a pas de monde accessible où a serait nécessairement vrai. Par conséquent, $\Diamond\neg a$ est vrai dans $w1$.
3. $M, w4 \models \neg\Diamond(a \wedge \neg b)$: Vrai Dans $w4$, a est vrai et il n'y a pas de monde accessible où $a \wedge \neg b$ serait nécessairement vrai. Par conséquent, $\neg\Diamond(a \wedge \neg b)$ est vrai dans $w4$.

On voudrait savoir si ces formules sont vraies ou fausses :

- $M, w3 \models \Box(a \supset b)$

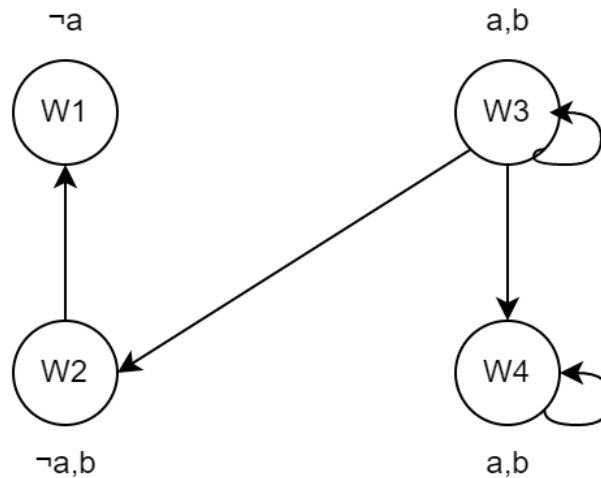


FIGURE 3.1 – Exemple de modèle modal

- $M, w1 \models \Diamond(a \wedge b)$
- $M, w2 \models \neg(\Box a)$

3.2 Code source et tests

Le programme ci-dessous commence par créer une signature pour la logique des prédicats du premier ordre et ajoute les deux prédicats unaires de notre modèle, a et b . Ensuite, il utilise un parseur "MlParser" pour créer un ensemble de connaissances BC contenant les trois formules propositionnelles modales qui sont vraies dans ce modèle.

Puis il utilise le raisonneur de logique modale "SimpleMlReasoner" pour répondre aux requêtes (formules). Enfin, il affiche le résultat true pour dire que la formule est vraie et false pour dire qu'elle est fausse.

```

1 import java.io.IOException;
2
3 import org.tweetyproject.commons.ParserException;
4 import org.tweetyproject.logics.commons.syntax.Predicate;
5 import org.tweetyproject.logics.commons.syntax.RelationalFormula;
6 import org.tweetyproject.logics.fol.syntax.FolFormula;
7 import org.tweetyproject.logics.fol.syntax.FolSignature;
8 import org.tweetyproject.logics.ml.parser.MlParser;
9 import org.tweetyproject.logics.ml.reasoner.SimpleMlReasoner;
10 import org.tweetyproject.logics.ml.syntax.MlBeliefSet;
11

```



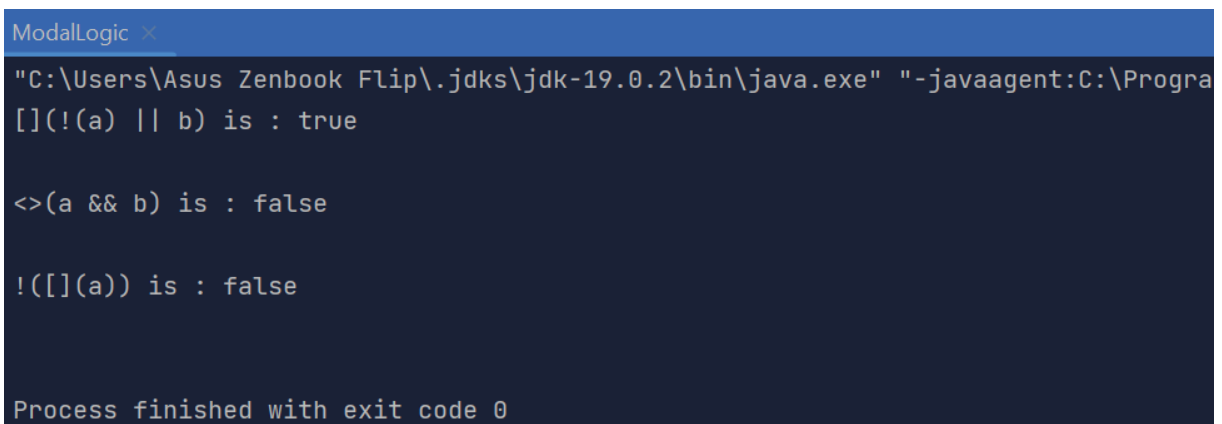
```

12
13 public class ModalLogic {
14     public static void main(String[] args) throws ParseException,
15     IOException {
16         MlParser parser = new MlParser();
17         FolSignature signature = new FolSignature();
18         signature.add(new Predicate("a", 0));
19         signature.add(new Predicate("b", 0));
20
21         parser.setSignature(signature);
22
23         MlBeliefSet BC = new MlBeliefSet();
24         BC.add((RelationalFormula) parser.parseFormula("[ (b && !( <> (!(a
25         ))))"));
26         BC.add((RelationalFormula) parser.parseFormula("<>(!(a))"));
27         BC.add((RelationalFormula) parser.parseFormula("!( <>(a && !(b)))
28         "));
29
30         SimpleMlReasoner reasoner = new SimpleMlReasoner();
31         System.out.println("[ (!(a) || b) is : " + reasoner.query(BC, (
32         FolFormula) parser.parseFormula("[ (!(a) || b)"))+"\n");
33         System.out.println("<>(a && b) is : " + reasoner.query(BC, (
34         FolFormula) parser.parseFormula("<>(a && b)"))+"\n");
35         System.out.println("!( [ (a) ) is : " + reasoner.query(BC, (
36         FolFormula) parser.parseFormula("[ (!(a))"))+"\n");
37     }
38 }

```

Listing 3.1 – Programme Java de logique modale.

L'exécution de ce programme a donner les résultats suivants :



```

ModalLogic x
"C:\Users\Asus Zenbook Flip\.jdk\jdk-19.0.2\bin\java.exe" "-javaagent:C:\Progra
[ (!(a) || b) is : true

<>(a && b) is : false

!( [ (a) ) is : false

Process finished with exit code 0

```

FIGURE 3.2 – Résultats d'exécution du programme de logique modale

Voici les résultats obtenus :

- $M, w3 \models \Box(a \supset b)$: Vrai Dans $w3$, nous avons a et b qui sont tous les deux vrais. La formule $a \rightarrow b$ est donc vraie dans $w3$. De plus, il n'y a pas de monde accessible où a est vrai et b est faux, ce qui signifie que $\Box(a \rightarrow b)$ est vrai dans tous les mondes accessibles à $w3$.
- $M, w1 \models \Diamond(a \wedge b)$: Faux Dans $w1$, $\neg a$ est vrai, donc $a \wedge b$ est faux. Il n'existe aucun monde accessible depuis $w1$ où $a \wedge b$ serait vrai. Par conséquent, $\Diamond(a \wedge b)$ est faux dans $w1$.
- $M, w2 \models \neg(\Box a)$: Faux Dans $w2$, $\neg a$ est vrai, donc $\Box a$ est faux. Par conséquent, $\neg(\Box a)$ est faux dans $w2$.

Chapitre 4

TP4 : Logique des défauts

La logique des défauts, également appelée logique non monotone, est un domaine de l'intelligence artificielle qui traite de la représentation des connaissances incomplètes ou incertaines. Contrairement à la logique classique, qui est monotone, c'est-à-dire que l'ajout de nouvelles informations ne peut que renforcer les conclusions précédentes, la logique des défauts permet la rétractation ou la modification de conclusions antérieures en présence de nouvelles informations.

En d'autres termes, dans la logique des défauts, les connaissances sont représentées sous forme de règles avec des exceptions ou des conditions spéciales qui peuvent être activées ou désactivées en fonction de la situation. Cela permet de modéliser des situations dans lesquelles une règle peut être vraie dans la plupart des cas, mais peut avoir des exceptions.

Dans ce TP, nous allons explorer la logique des défauts en implémentant le premier exercice de la série du TD.

4.1 Exercice

Nous allons utiliser l'outil **DefaultLogic** pour ce TP qui est un outil puissant utilisé dans la logique formelle pour le raisonnement par défaut. Il fournit un cadre pour la représentation des connaissances et le raisonnement dans des situations où l'information est incomplète ou incertaine.

On a testé cet outil avec l'énoncé de l'exercice 1 de la série du TD :

Exercice 1:

Soit l'ensemble de défauts $D=\{d_1, d_2\}$ avec $d_1 = A: B/C$ et $d_2 = A: \neg C/D$.

Quelles sont les extensions qui peuvent se déduire si on considère les ensembles de formules suivantes:

1. $W=\{ \neg A \}$
2. $W=\{ A, \neg B \}$
3. $W=\{ A, \neg C \vee \neg D \}$
4. $W=\{ A, \neg B \wedge C \}$

FIGURE 4.1 – Enoncé exercice 1

4.2 Code Source

```

1 package be.fnord.DefaultLogic;
2 import a.e;
3 import be.fnord.util.logic.DefaultReasoner;
4 import be.fnord.util.logic.WFF;
5 import be.fnord.util.logic.defaultLogic.DefaultRule;
6 import be.fnord.util.logic.defaultLogic.RuleSet;
7 import be.fnord.util.logic.defaultLogic.WorldSet;
8 import java.util.HashSet;
9
10 public static void EXERCISE1(){
11
12     RuleSet rules = new RuleSet(); // cr ation de l'ensemble des
    d fauts
13
14     DefaultRule d1 = new DefaultRule(); //cr ation du d faut d1
15     d1.setPrerequisite("A"); // prerequisite du d1
16     d1.setJustificatoin("B"); // justification du d1
17     d1.setConsequence("C"); // consequence du d1
18
19     rules.addRule(d1); // addition de d1 dans l'ensemble des defaults
20
21     DefaultRule d2 = new DefaultRule(); //meme chose pour d2
22     d2.setPrerequisite("A");
23     d2.setJustificatoin(e.NOT+"C");
24     d2.setConsequence("D");
25
26     rules.addRule(d2);
27
28     WorldSet w1= new WorldSet(); // definition d'un monde w1
29     w1.addFormula(e.NOT+"A");
30
31     WorldSet w2= new WorldSet(); // def w2
32     w2.addFormula("A");

```

```

33     w2.addFormula(e.NOT+"B");
34
35     WorldSet w3= new WorldSet(); // def w3
36     w3.addFormula("A");
37     w3.addFormula("(" + e.NOT+"C" + e.OR+e.NOT+"D)");
38
39     WorldSet w4= new WorldSet(); // def w4
40     w4.addFormula("A");
41     w4.addFormula("(" + e.NOT+"B" + e.AND+"C)");
42
43     //W1
44     try {
45         a.e.println("\nMonde 1:");
46         DefaultReasoner r = new DefaultReasoner(w1, rules); // raisonner
47         HashSet<String> scenarios = r.getPossibleScenarios(); //
48         extraction des extensions
49         a.e.println("w1: [" + w1.toString() + "]\nDefaults: [" + rules.
50         toString() + "]");
51         if(scenarios.isEmpty()) a.e.println("ces d faults ne g n re pas
52         d'extension");
53         for (String c : scenarios) {
54             a.e.println("E: Th(W U {" + c + "})");
55             WFF world_and_ext = new WFF("(" + w1.getWorld() + " ) & (" + c +
56             "));");
57             a.e.println(world_and_ext.getClosure());
58             a.e.decIndent();
59         }
60         a.e.println("");
61     }
62     catch(Exception e){
63         System.out.println(e);
64     }
65
66     //W2
67     try {
68         a.e.println("Monde 2:");
69         DefaultReasoner r = new DefaultReasoner(w2, rules);
70         HashSet<String> scenarios = r.getPossibleScenarios();
71         a.e.println("w2: [" + w2.toString() + "]\nDefaults: [" + rules.
72         toString() + "]");
73         for (String c : scenarios) {
74             a.e.println("E: Th(W U {" + c + "})");
75             WFF world_and_ext = new WFF("(" + w2.getWorld() + " ) & (" + c +
76             "));");
77             a.e.println(world_and_ext.getClosure());
78             a.e.decIndent();
79         }
80         a.e.println("");

```

```

74     }
75     catch(Exception e){
76         System.out.println(e);
77     }
78
79     //W3
80     try {
81         a.e.println("Monde 3:");
82         DefaultReasoner r = new DefaultReasoner(w3, rules);
83         HashSet<String> scenarios = r.getPossibleScenarios();
84         a.e.println("w3: [" + w3.toString() + "]\nDefaults: [" + rules.
toString() + "]");
85         for (String c : scenarios) {
86             a.e.println("E: Th(W U {" + c + "})");
87             WFF world_and_ext = new WFF("(" + w3.getWorld() + " ) & (" + c +
"))");
88             a.e.println(world_and_ext.getClosure());
89             a.e.decIndent();
90         }
91         a.e.println("");
92     }
93     catch(Exception e){
94         System.out.println(e);
95     }
96
97     //W4
98     try {
99         a.e.println("Monde 4:");
100         DefaultReasoner r = new DefaultReasoner(w4, rules);
101         HashSet<String> scenarios = r.getPossibleScenarios();
102         a.e.println("w3: [" + w4.toString() + "]\nDefaults: [" + rules.
toString() + "]");
103         for (String c : scenarios) {
104             a.e.println("E: Th(W U {" + c + "})");
105             WFF world_and_ext = new WFF("(" + w4.getWorld() + " ) & (" + c +
"))");
106             a.e.println(world_and_ext.getClosure());
107             a.e.decIndent();
108         }
109         a.e.println("");
110     }
111     catch(Exception e){
112         System.out.println(e);
113     }
114 }

```

Listing 4.1 – Code source de l'outil DefaultLogic sur l'exercice 1.

4.3 Exécution

Voici le résultat de l'exécution du code :

- **W1** : Le raisonneur ne trouve pas d'extensions car : les défauts d_1 et d_2 ne sont pas utilisables car leurs prérequis $A \in \Gamma_1(E)$. D'où, par clôture déductive et minimalité, les deux défauts ne sont pas générateurs d'extension.
- **W2** : Le raisonneur arrive à trouver une extension $TH(\{A, \neg B, D\})$ car : les deux défauts de la théorie d_1 et d_2 sont utilisables car leur prérequis $A \in \Gamma\Delta_2(E)$ (vu que $A \in W$ et $W \subset \Gamma\Delta_2(E)$). L'applicabilité de d_1 rend d_2 non applicable. Or d_1 est non applicable vu que la négation de sa justification $\neg B \in \Gamma\Delta_2(E)$ (vu que justification $\neg B \in W_2$ et $W_2 \subset \Gamma\Delta_2(E)$) et que si E est une extension $E = \Gamma\Delta_2(E)$. D'où, $\neg B \in E$. Pour d_2 , il est applicable.
D'où, par clôture déductive et minimalité, cette théorie admet une seule extension : $E = \Gamma\Delta_2(E) = TH(\{A, \neg B, D\})$. Seul d_2 est générateur d'extension.
- **W3** : Le raisonneur arrive à trouver une seule extension $E = \Gamma\Delta_3(E) = TH(A, \neg C \vee \neg D, C)$
- **W4** : le raisonneur affiche une erreur car la négation des justifications des défauts d_1 et d_2 $\neg B$ et $\neg\neg C$ appartiennent à E .

Demonstrating reasoners:

Monde 1:

w1: $[\neg A]$

Defaults: $[(A):(B) \Rightarrow (C)]$, $[(A):(\neg C) \Rightarrow (D)]$

ces défauts ne génère pas d'extension

Monde 2:

Trying eeee & A & $\neg B$

Trying eeee & A & $\neg B$

w2: $[A \text{ \& } \neg B]$

Defaults: $[(A):(B) \Rightarrow (C)]$, $[(A):(\neg C) \Rightarrow (D)]$

E: $\text{Th}(W \cup \{D\})$

$D \text{ \& } \neg B \text{ \& } \text{eeee} \text{ \& } A$

Monde 3:

Trying eeee & A & $(\neg C | \neg D)$

Trying eeee & A & $(\neg C | \neg D)$

Trying eeee & A & $(\neg C | \neg D)$

Trying eeee & A & $(\neg C | \neg D)$

w3: $[A \text{ \& } (\neg C | \neg D)]$

Defaults: $[(A):(B) \Rightarrow (C)]$, $[(A):(\neg C) \Rightarrow (D)]$

E: $\text{Th}(W \cup \{C\})$

$C \text{ \& } \neg D \text{ \& } \text{eeee} \text{ \& } (\neg D | \neg C) \text{ \& } A$

Monde 4:

Trying eeee & A & $(\neg B \& C)$

Trying eeee & A & $(\neg B \& C)$

error

FIGURE 4.2 – Résultat d'exécution de l'outil DefaultLogic sur l'exercice 1

Chapitre 5

TP5 : Réseaux sémantiques

Dans le domaine de la représentation des connaissances, les réseaux sémantiques sont une méthode populaire pour organiser et structurer l'information. Un réseau sémantique est un graphe orienté dans lequel les nœuds représentent des concepts et les arêtes représentent des relations entre ces concepts.

La technique de propagation de marqueurs est une méthode d'analyse des réseaux sémantiques. Elle consiste à associer des marqueurs à certaines unités lexicales (mots ou expressions) du réseau, afin de modéliser la diffusion d'une information ou d'un concept dans ce réseau. Les marqueurs peuvent être des valeurs numériques, des étiquettes, des couleurs, etc.

La technique d'héritage est une méthode d'analyse des réseaux sémantiques qui permet de modéliser les relations de sous-typage entre les concepts. Elle consiste à organiser les concepts en une hiérarchie de classes, où chaque classe représente un ensemble de concepts ayant des propriétés et des relations sémantiques communes.

Dans ce TP, nous allons explorer les réseaux sémantiques en implémentant les algorithmes de propagation des marqueurs et d'héritage.

5.1 Partie 01 : Algorithme de propagation de marqueurs

Il s'agit dans cette partie d'implémenter l'algorithme de propagation de marqueurs dans les réseaux sémantiques.

La propagation de marqueurs se fait en plusieurs étapes. Tout d'abord, on initialise les marqueurs sur certaines unités lexicales du réseau. Ensuite, on utilise des règles de

propagation pour transmettre les marqueurs d'une unité lexicale à une autre, en fonction de la similarité sémantique entre ces unités.

5.1.1 Algorithme

Voici l'algorithme implémenté en python :

```
1 def RSpropagationMarqueurs(resSem, noeud1, noeud2, relation):
2
3     # extraction des noeuds du reseau
4     noeuds = resSem["nodes"]
5     # extraction des arettes
6     arettes = resSem["edges"]
7
8     reponse = noeud1 + " " + relation + " " + noeud2 + ":\n\t\t\t"
9     solutionExiste = False
10
11     # si les noeuds n'existent pas dans le reseau alors manque de
12     # connaissance
13     try:
14         # on trouve les noeuds dans le reseau semantique depuis les
15         # noeuds donnees en entrees (les marqueurs)
16         marq1 = [node for node in noeuds if node["label"] == noeud1][0]
17         marq2 = [node for node in noeuds if node["label"] == noeud2][0]
18         # extraire les arettes qui representent un lien est-un vers
19         # marqueur 1
20         arettesPropagation = [arete for arete in arettes if (arete["to"]
21 == marq1["id"] and arete["label"] == "is a")]
22         # tanque on a pas trouver une solution et il existe des arettes
23         # a verifier
24         while len(arettesPropagation) != 0 and not solutionExiste:
25             noeudTemp = arettesPropagation.pop()
26             arettesTemp = [arete for arete in arettes if (arete["from"]
27 == noeudTemp["from"] and arete["label"] == relation)]
28             solutionExiste = any(foo['to'] == marq2["id"] for foo in
29 arettesTemp)
30             if not solutionExiste:
31                 arettesEstUn = [arete for arete in arettes if (arete["to"]
32 == noeudTemp["from"] and arete["label"] == "is a")]
33                 arettesPropagation.extend(arettesEstUn)
34                 arettesNoeudRelation = [arete["from"] for arete in arettes if (
35 arete["to"] == marq2["id"] and arete["label"] == relation)]
36                 labelsArettesNoeudRelation = [marq2["label"] for marq2 in noeuds
37 if marq2["id"] in arettesNoeudRelation]
38             if solutionExiste:
```

```

29     reponse += "lien existe entre les noeuds : " + " et ".join(
labelsArettesNoeudRelation)
30     else:
31         reponse += "pas de lien"
32 except IndexError:
33     return(reponse + "noeud absent dans le reseau\n")
34
35 return(reponse + "\n")

```

Listing 5.1 – Algorithme de propagation de marqueurs.

5.1.2 Exécution

Pour tester l'algorithme on a utilisé le fichier json utilisé dans la plateforme khazour qui est mentionné dans la serie du TP.

Voici le réseau utilisé :

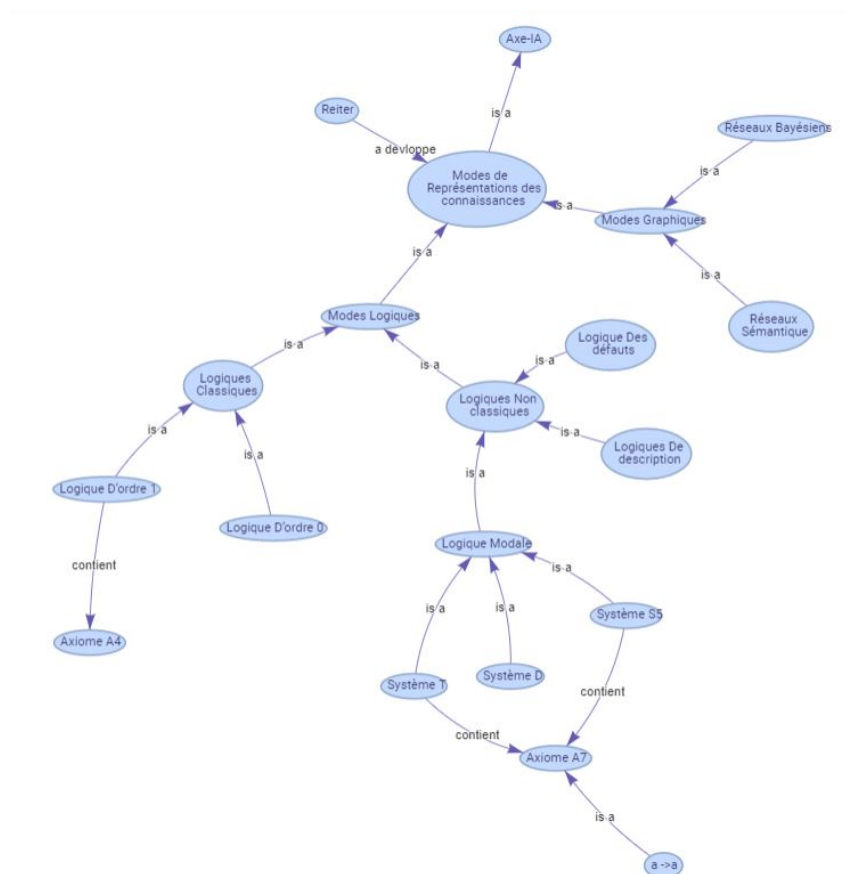


FIGURE 5.1 – Réseau sémantique utilisé.

Voici le résultat obtenu :

```

Modes de Representations des connaissances contient Axiome A4:
    lien existe entre les noeuds : Logique D ordre 1

Modes de Representations des connaissances contient Axiome A7:
    lien existe entre les noeuds : Systeme T et Systeme S5

Modes de Representations des connaissances contient Axe-IA:
    pas de lien

Modes de Representations des connaissances contient blablabla:
    noeud absent du reseau

```

FIGURE 5.2 – Résultats de l'exécution de l'algorithme du propagation des marqueurs sur un réseau sémantique.

5.2 Partie 02 : Algorithme d'héritage

Il s'agit dans cette partie du TP d'implémenter l'algorithme d'héritage qui est un processus d'inférence très utilisé dans les réseaux sémantiques.

5.2.1 Algorithme

Voici l'algorithme implémenté en python :

```

1 def heritage(resSem, nom):
2     noeuds = resSem["nodes"]
3     arretes = resSem["edges"]
4
5     stop = False
6
7     noeud = [noeud for noeud in noeuds if noeud["label"] == nom][0]
8     arretesHeritage = [arete["to"] for arete in arretes if (arete["from"]
9 ] == noeud["id"] and arete["label"] == "Est-un")]
10
11     heritage = []
12     proprietes = []
13
14     while not stop:
15         i = arretesHeritage.pop()
16         heritage.append(" ,".join([noeud["label"] for noeud in resSem["
17 nodes"] if noeud["id"] == i]))
18         arretesHeritage.extend([arete["to"] for arete in arretes if (
19 arete["from"] == i and arete["label"] == "Est-un")])
20         noeudsProprietes = [arete for arete in arretes if (arete["from"]
21 == i and arete["label"] != "Est-un")]
22         for p in noeudsProprietes:

```

```

19     proprietes.append(": ".join([p["label"], " , ".join([noeud["
label"] for noeud in resSem["nodes"] if noeud["id"] == p["to"]]))))
20     if len(arretesHeritage) == 0:
21         stop = True
22
23     return heritage, proprietes

```

Listing 5.2 – Algorithme d'héritage.

5.2.2 Exécution

Pour tester l'algorithme on a utilisé le réseau sémantique du cours :

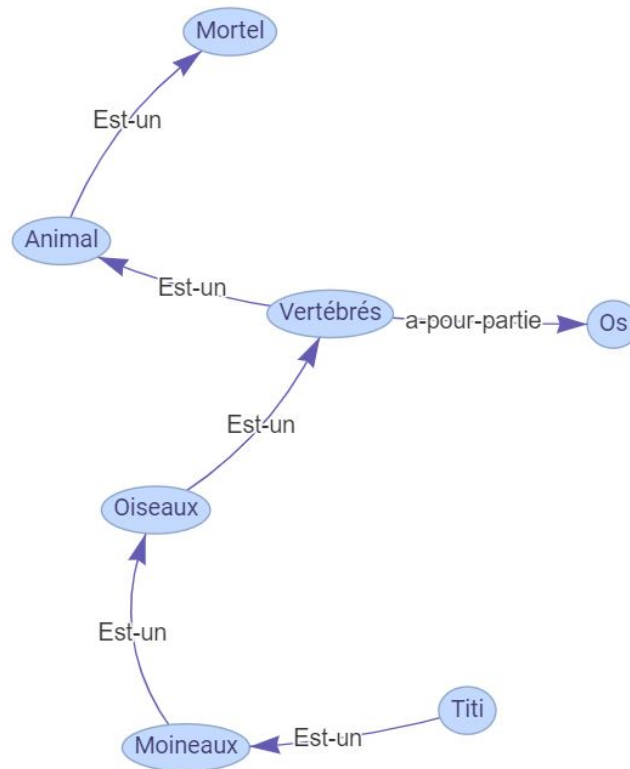


FIGURE 5.3 – Réseau sémantique du le cours page 11.

Voici les résultats obtenus :

```

Résultat d'Application de l'heritage:
Titi
Moineaux
Oiseaux
VertÃ©brÃ©s
Animal
Mortel

Deduction des proprietes:
a-pour-partie: 0s

```

FIGURE 5.4 – Résultats d'exécution de l'algorithme d'heritage.

5.3 Partie 03 : Algorithme de propagation de marqueurs cas d'exceptions

Dans cette partie, nous allons implémenter un algorithme qui permet d'inhiber la propagation dans le cas des liens d'exception dans un réseau sémantique.

5.3.1 Algorithme

Voici l'algorithme implémenté en python :

```

1 def RSpropagationMarqueursExceptions(resSem, noeud1, noeud2, relation):
2     # extraction des noeuds du reseau
3     noeuds = resSem["nodes"]
4     # extraction des arettes
5     arettes = resSem["edges"]
6
7     reponse = noeud1 + " " + relation + " " + noeud2 + ":\n\t\t\t"
8     solutionExiste = False
9
10    # si les noeuds n'existent pas dans le reseau alors manque de
    connaissance
11    try:
12        # on trouve les noeuds dans le reseau semantique depuis les
        noeuds donnees en entrees (les marqueurs)
13        marq1 = [node for node in noeuds if node["label"] == noeud1][0]
14        marq2 = [node for node in noeuds if node["label"] == noeud2][0]
15        # extraire les arettes qui represent un lien est-un vers
        marqueur 1
16        arettesPropagation = [arete for arete in arettes if (arete["to"]
        == marq1["id"] and arete["label"] == "is a" and arete["edge_type"]
        != "exception")]

```

```
17     # tanque on a pas trouver une solution et il existe des arettes
    a verifier
18     while len(arettresPropagation) != 0 and not solutionExiste:
19         noeudTemp = arettresPropagation.pop()
20         arettresTemp = [arete for arete in arettres if (arete["from"]
    == noeudTemp["from"] and arete["label"] == relation and arete["
    edge_type"] != "exception")]
21         solutionExiste = any(foo['to'] == marq2["id"] for foo in
    arettresTemp)
22         if not solutionExiste:
23             arettresEstUn = [arete for arete in arettres if (arete["to
    "] == noeudTemp["from"] and arete["label"] == "is a" and arete["
    edge_type"] != "exception")]
24             arettresPropagation.extend(arettresEstUn)
25             arettresExceptions = [arete["from"] for arete in arettres if (
    arete["edge_type"] == "exception")]
26             arettresNoeudRelation = [arete["from"] for arete in arettres if (
    arete["to"] == marq2["id"] and arete["label"] == relation and arete["
    from"] not in arettresExceptions)]
27             for i in arettresNoeudRelation:
28                 print(i)
29             labelsArettesNoeudRelation = [marq2["label"] for marq2 in noeuds
    if marq2["id"] in arettresNoeudRelation]
30             if solutionExiste:
31                 reponse += "lien existe entre les noeuds : " + " et ".join([
    label for label in labelsArettesNoeudRelation])
32             else:
33                 reponse += "pas de lien"
34             except IndexError:
35                 return(reponse + "noeud absent dans le reseau\n")
36
37     return(reponse + "\n")
```

Listing 5.3 – Algorithme de propagation de marqueurs avec exceptions.

5.3.2 Exécution

Pour tester cette algorithme on a utilisé l'exemple donné dans la serie du TP de la plateforme khazour. on a juste ajouter une exception entre Système S5 et logique modale pour tester.

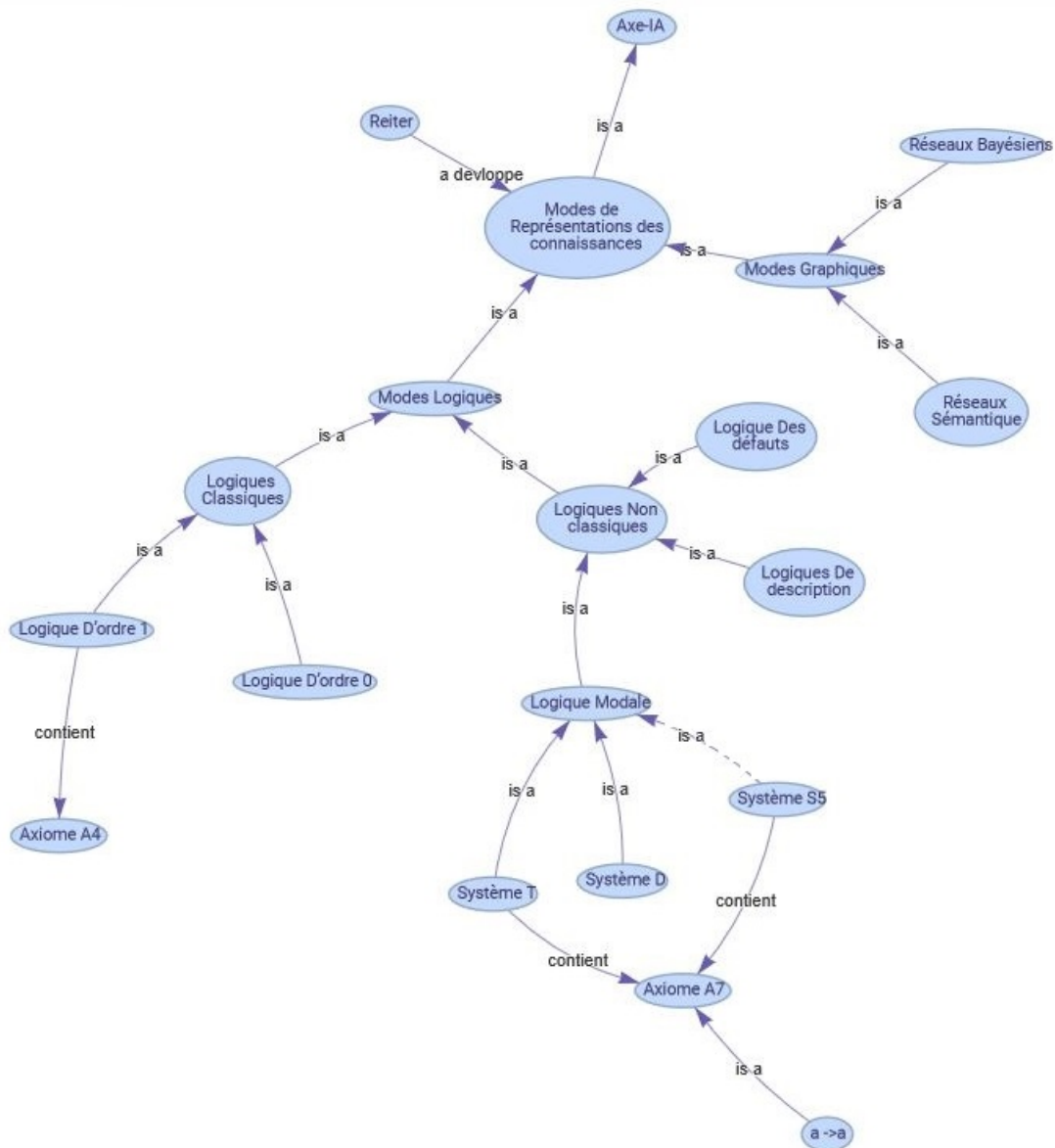


FIGURE 5.5 – Réseau sémantique de la plateforme khzour modifié.

Modes de Representations des connaissances contient Axiome A7:
lien existe entre les noeuds : Systeme T

FIGURE 5.6 – Résultats d'exécution de l'algorithme d'exceptions.

Chapitre 6

TP6 : Logique de description

La logique de description vise à représenter et à raisonner sur des connaissances complexes. Elle offre un cadre formel permettant de décrire des concepts, des relations et des contraintes dans un domaine spécifique. Elle repose sur une structure de langage qui permet de définir des classes, des propriétés et des relations entre ces classes. Elle offre des mécanismes pour spécifier des axiomes, des restrictions et des règles d'inférence, ce qui permet de déduire de nouvelles informations à partir des connaissances existantes. Ainsi, elle permet de représenter de manière précise et formelle des connaissances sur un domaine spécifique et de raisonner logiquement sur ces connaissances.

Dans ce TP, pour le raisonneur on a opté pour Pellet et pour l'outil l'app WebProtégé.

6.1 Web Protégé

Web Protégé est un outil puissant et convivial pour la modélisation des connaissances basée sur les ontologies. Il offre une interface web intuitive qui permet aux utilisateurs de collaborer et de construire des ontologies de manière efficace.

Pour la démonstration de l'outil, on a choisi les TBOX et ABOX suivantes :

TBOX : La TBOX représente les axiomes terminologiques, c'est-à-dire les définitions conceptuelles et les relations entre les concepts.

Concept : Animal

Sous-concept : Humain, Chien, Chat, Oiseau

Concept : Plante

Sous-concept : Fruit, Vegetable

Role : mange, possède

ABOX : L'ABOX représente les assertions individuelles, c'est-à-dire les faits concrets concernant les individus et leurs relations.

Individu : Goku

Type : Humain

Individu : Doug

Type : Chien

Individu : Fifi

Type : Oiseau

On a utilisé un fichier .owl pour la définition de notre TBOX et ABOX. Voici l'interface de l'outil Web Protégé :

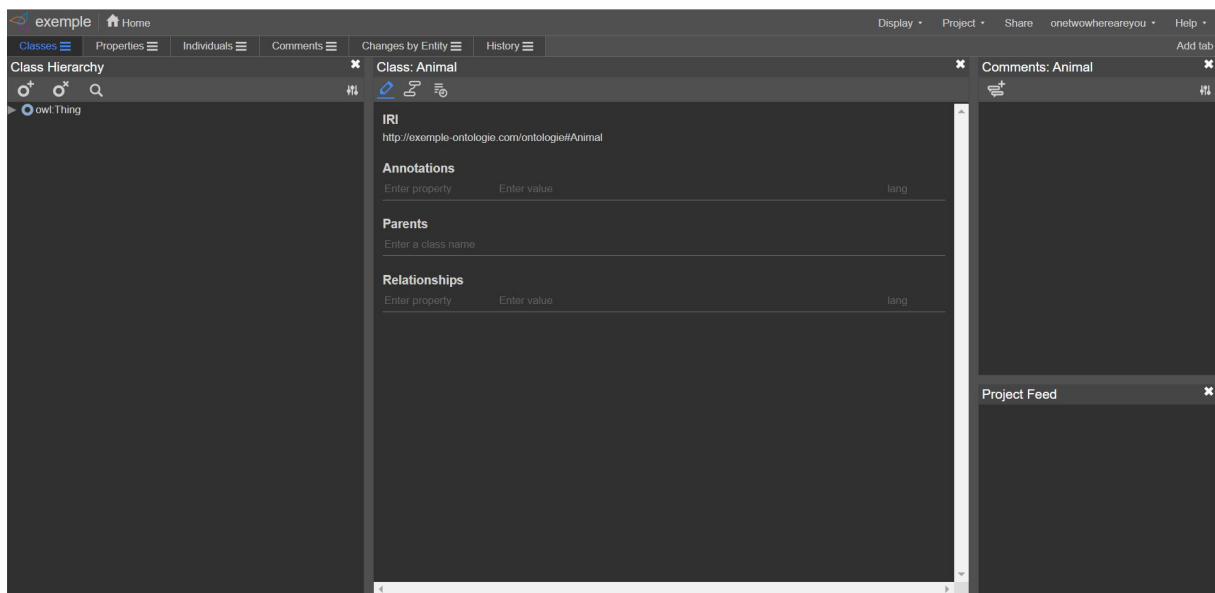


FIGURE 6.1 – Interface de l'outil web protégé.

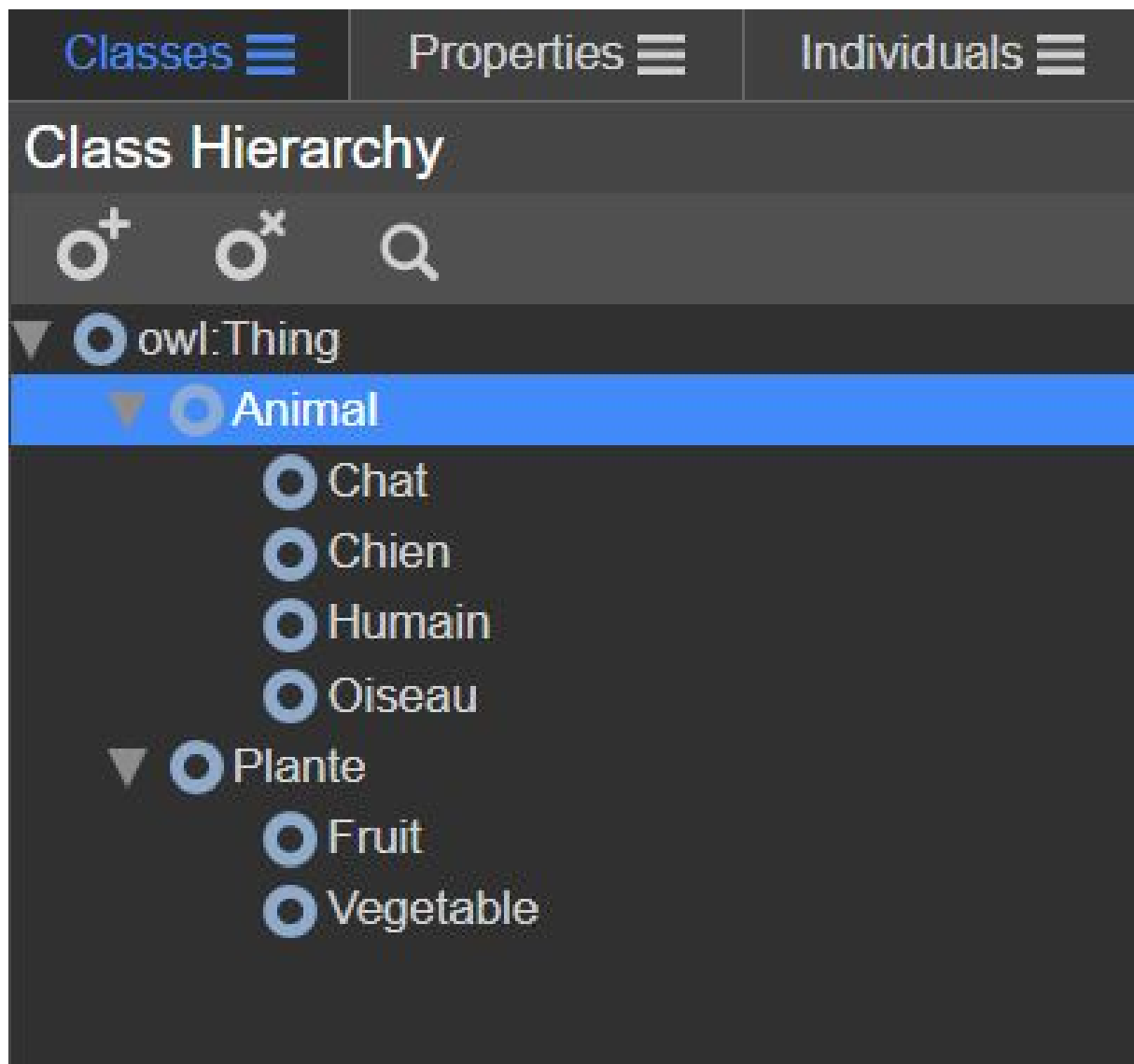


FIGURE 6.2 – Les Concepts.

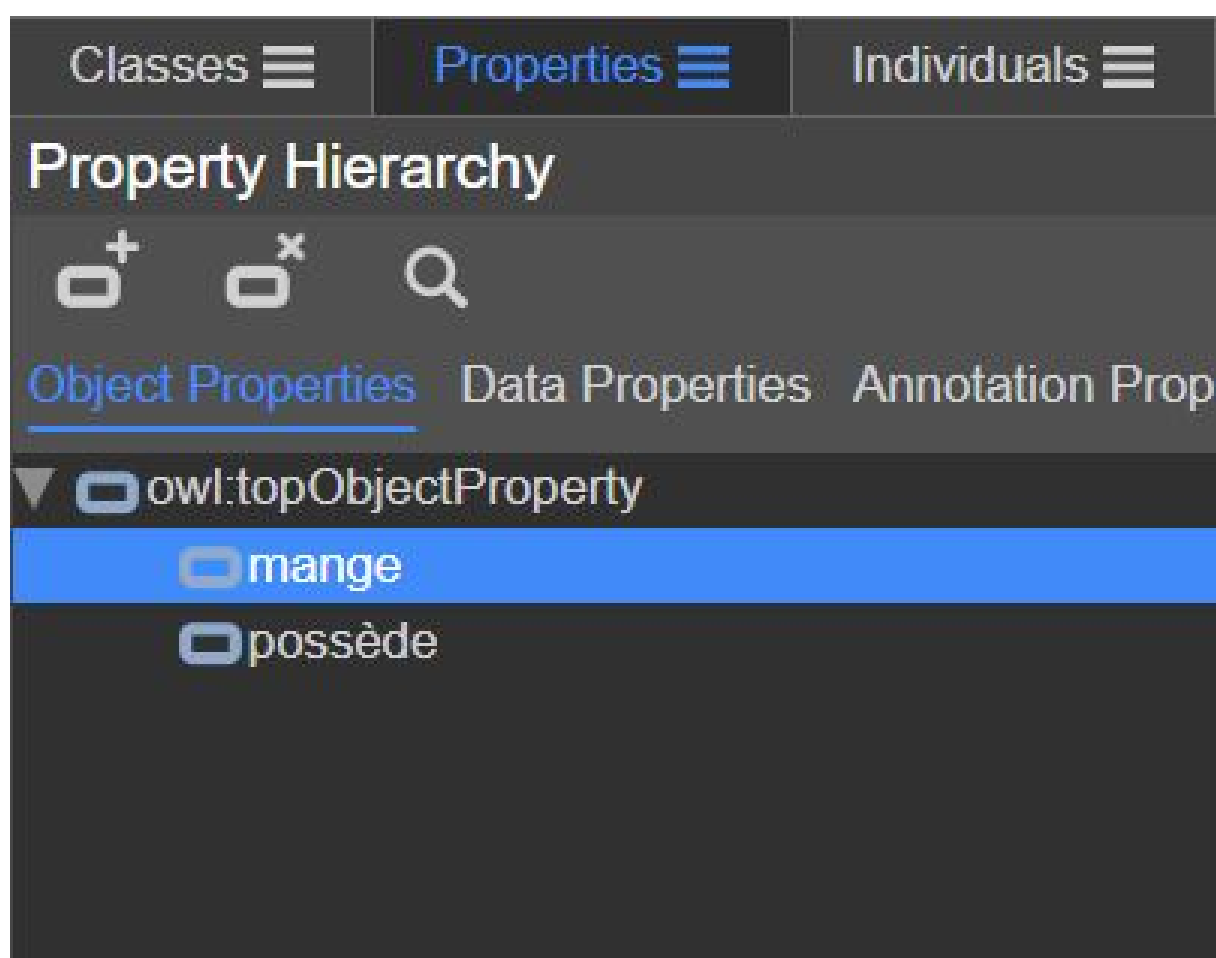


FIGURE 6.3 – Les Roles.

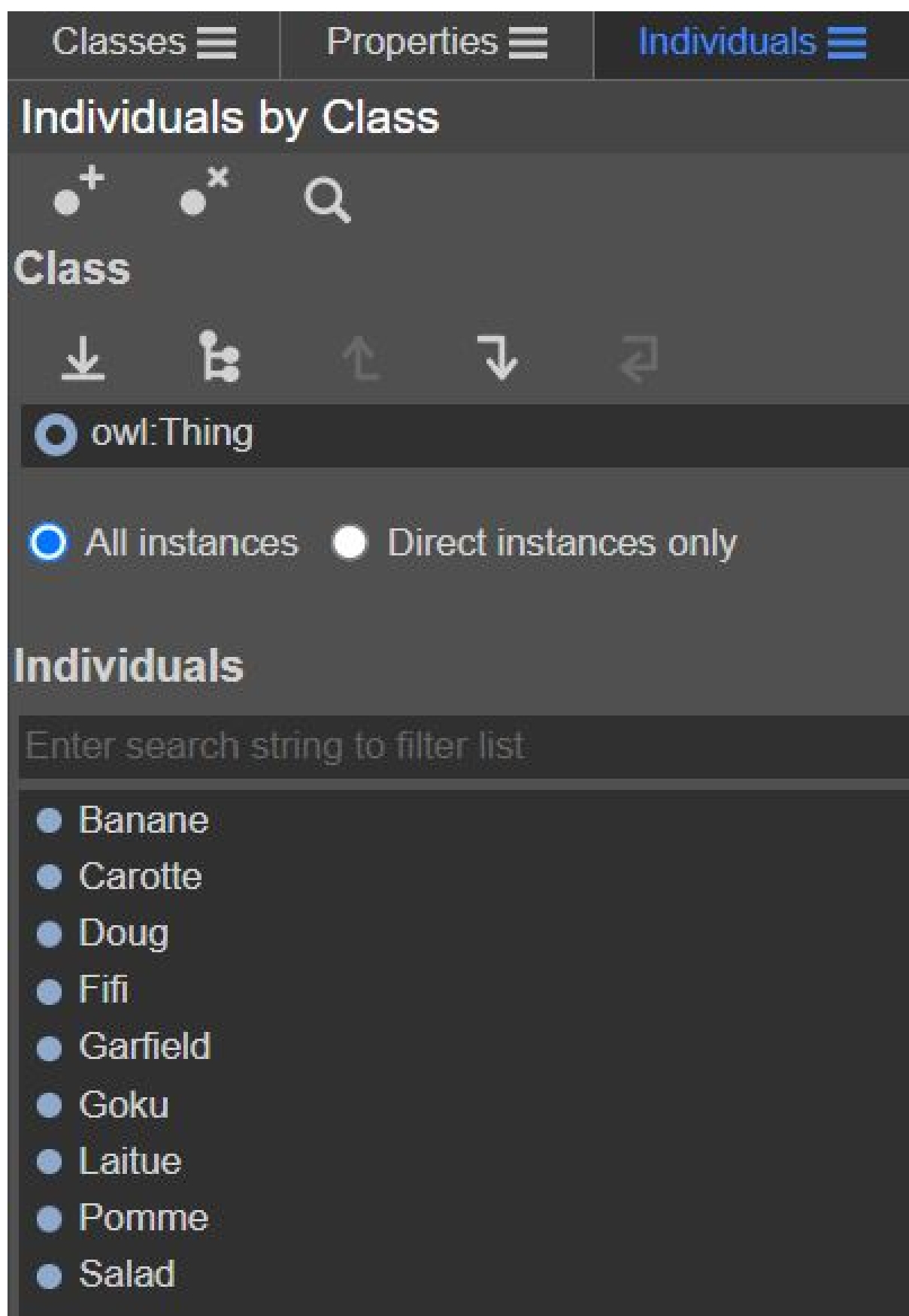





FIGURE 6.4 – Les individus.

Individual: Goku



IRI
`http://exemple-ontologie.com/ontologie#Goku`

Annotations
Enter property Enter value

Types
Enter a class name

Relationships

<input type="checkbox"/> mange	<input checked="" type="radio"/> Carotte
<input type="checkbox"/> mange	<input checked="" type="radio"/> Pomme
<input type="checkbox"/> possède	<input checked="" type="radio"/> Doug
<input type="checkbox"/> possède	<input checked="" type="radio"/> Fifi
<input type="checkbox"/> possède	<input checked="" type="radio"/> Garfield

Enter property Enter value

FIGURE 6.5 – Informations sur les individus.

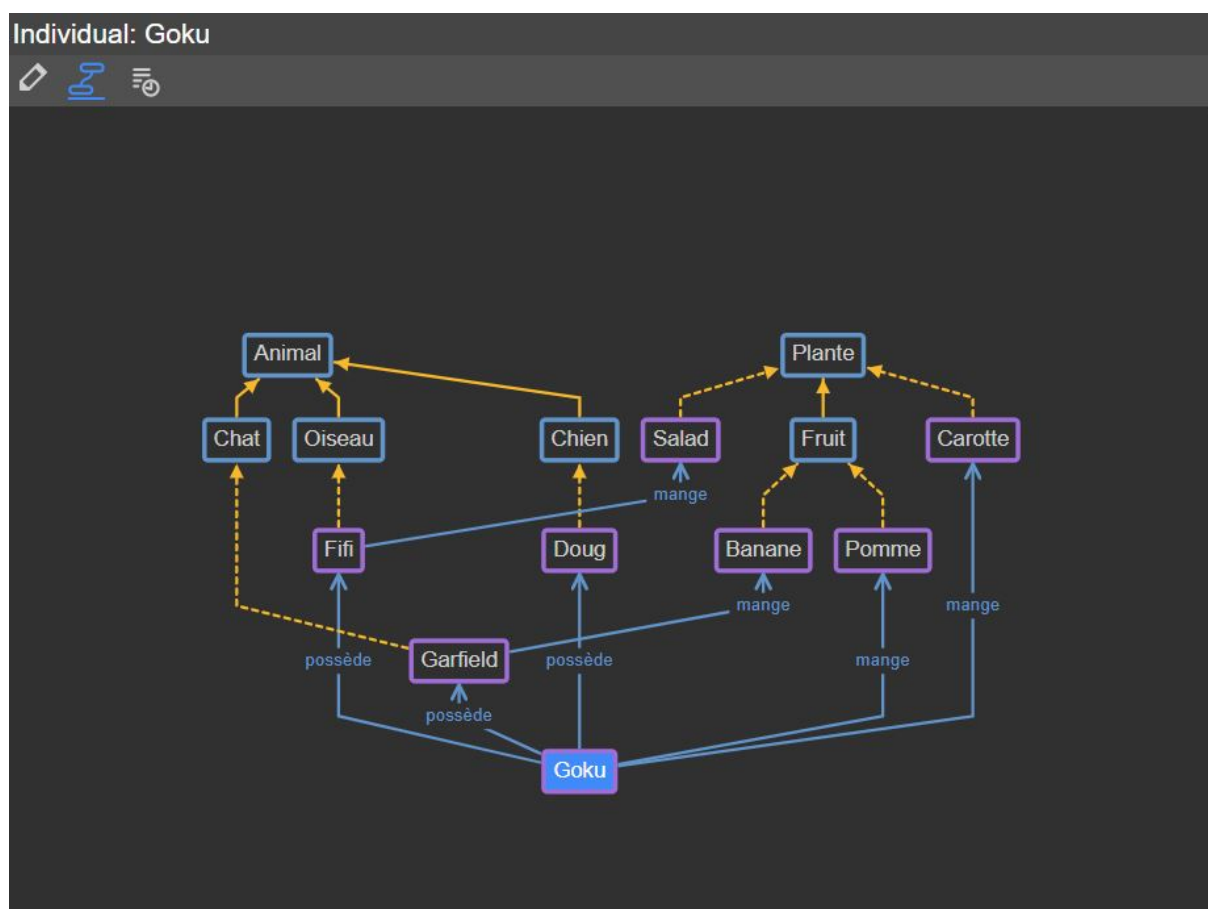


FIGURE 6.6 – Schéma d'un individu.

Conclusion générale

Au cours de ces TP, nous avons exploré les diverses logiques sur lesquelles un système informatique peut fonctionner. Nous avons acquis des compétences en modélisation de problèmes réels, en génération d'inférences, en utilisation de raisonneurs pour évaluer la validité de formules, en représentation graphique d'informations, et enfin en déduction de faits. Ces expériences nous ont permis d'élargir notre compréhension des possibilités offertes par les systèmes informatiques et de développer des compétences précieuses dans ces domaines.