

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique



Université des Sciences et de la Technologie Houari Boumédiène

Faculté d'Electronique et d'Informatique
Département Informatique

Master Systèmes Informatiques intelligents

Module : Représentation des connaissances et raisonnement.

Rapport des Tps RCR

Réalisé par :

BENKOUITEN Aymen, 191931046409

KENAI Imad Eddine, 191932017671

Année universitaire : 2022 / 2023

Table des matières

1	Introduction Générale	2
2	TP 1 : Inférence logique basée sur un solveur SAT	3
3	TP 2 : Inférence logique premier ordre	12
4	TP 3 : Logique Modale	15
5	TP 4 : Logique des défauts	22
6	TP 5 : Réseaux sémantiques	42
7	TP 6 : Logique des descriptions	48
8	Conclusion Générale	57

Chapitre 1

Introduction Générale

La représentation des connaissances désigne un ensemble d'outils et de procédés destinés d'une part à représenter et d'autre part à organiser le savoir humain pour l'utiliser et le partager. Il s'agit d'un domaine de l'intelligence artificielle (IA) consacré à la représentation des informations sur le monde sous une forme qu'un système informatique peuvent utiliser pour résoudre des tâches complexes. Dans cette série de Tps nous allons voir comment peut-on faire ceci tout en découvrant les différentes logiques et les types de raisonneurs qui font des déductions dans ces dernières. La représentation des connaissances englobe un ensemble de méthodes et d'outils visant à représenter et à structurer le savoir humain afin de faciliter son utilisation et son partage. Ce domaine de l'intelligence artificielle (IA) se concentre sur la représentation des informations sur le monde d'une manière exploitable par un système informatique pour résoudre des tâches complexes. Au cours de cette série de travaux pratiques, nous examinerons comment accomplir cela tout en explorant les différentes formes de logique et les types de raisonneurs utilisés pour effectuer des déductions dans ces systèmes.

Chapitre 2

TP 1 : Inférence logique basée sur un solveur SAT

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 copie des fichiers sous format CNF

DATA (D:) > M1 SII > Rep_connaissance > tp > tp1 >

Nom	Modifié le	Type	Taille
src	28/04/2023 09:54	Dossier de fichiers	
algorithms.txt	09/05/2008 18:36	Document texte	10 Ko
legal.txt	02/05/2008 10:11	Document texte	4 Ko
Makefile	02/05/2008 11:33	Fichier	2 Ko
readme.txt	09/05/2008 18:36	Document texte	3 Ko
revisions.txt	09/05/2008 18:20	Document texte	11 Ko
sample.cnf	02/05/2008 10:11	Fichier CNF	16 Ko
sample.wcnf	02/05/2008 10:11	Fichier WCNF	22 Ko
test1.cnf	28/04/2023 09:53	Fichier CNF	1 Ko
test2.cnf	28/04/2023 09:55	Fichier CNF	1 Ko
ubcsat	09/05/2008 18:37	Fichier	330 Ko
ubcsat.exe	09/05/2008 18:36	Application	356 Ko

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 suivantes (avec test.cnf le fichier contenant les règles SAT).

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

Listing 1 – SAT test

Voici le résultat.

```

D:\M1 SII\Rep_connaissance\tp\tp1>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
# -gtimeout 0
# -noimprove 0
# -target 0
# -wtarget 0
# -seed 1841231327
# -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.

```

#
# 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       1        1
#

```

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 = 9  
TotalLiterals = 23  
TotalCPUtimeElapsed = 0.002  
FlipsPerSecond = 500  
RunsExecuted = 1  
SuccessfulRuns = 1  
PercentSuccess = 100.00  
Steps_Mean = 1  
Steps_CoeffVariance = 0  
Steps_Median = 1  
CPUtime_Mean = 0.00200009346008  
CPUtime_CoeffVariance = 0  
CPUtime_Median = 0.00200009346008  
  
D:\M1 SII\Rep_connaissance\tp\tp1>
```

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.

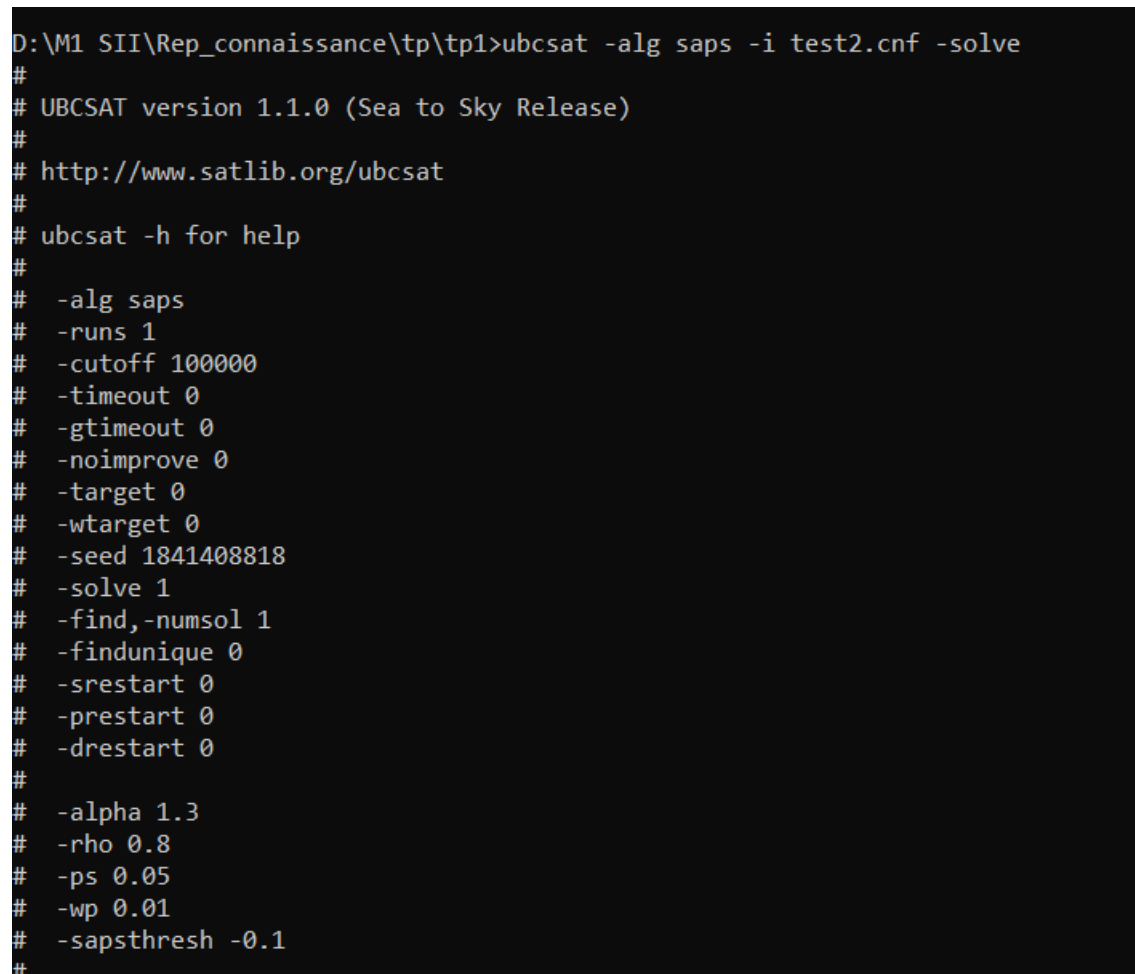
3.2 Le fichier Test1.cnf

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.



```
D:\M1 SII\Rep_connaissance\tp\tp1>ubcsat -alg saps -i test2.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 1841408818
# -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

```

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 :

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

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

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

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

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

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

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

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

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

$$Na; Ceb; Mc$$

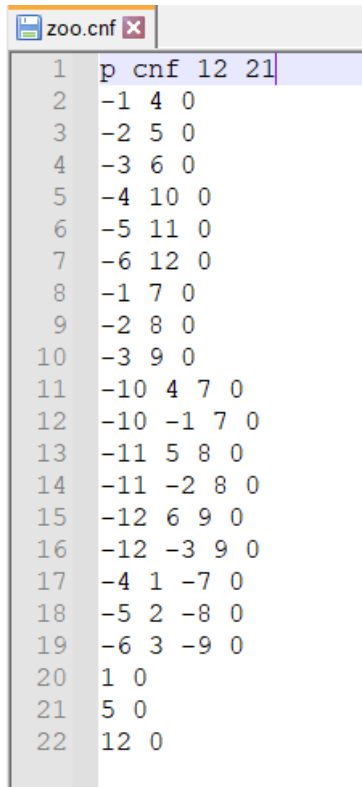
Nous avons 12 variables et 21 clauses au total. Représentation dans le fichier :

1 = Na ; 2 = Nb ; 3 = Nc ;

4 = Cea ; 5 = Ceb ; 6 = Cec ;

7 = Coa ; 8 = Cob ; 9 = Coc ;

10 = Ma ; 11 = Mb ; 12 = Mc ;



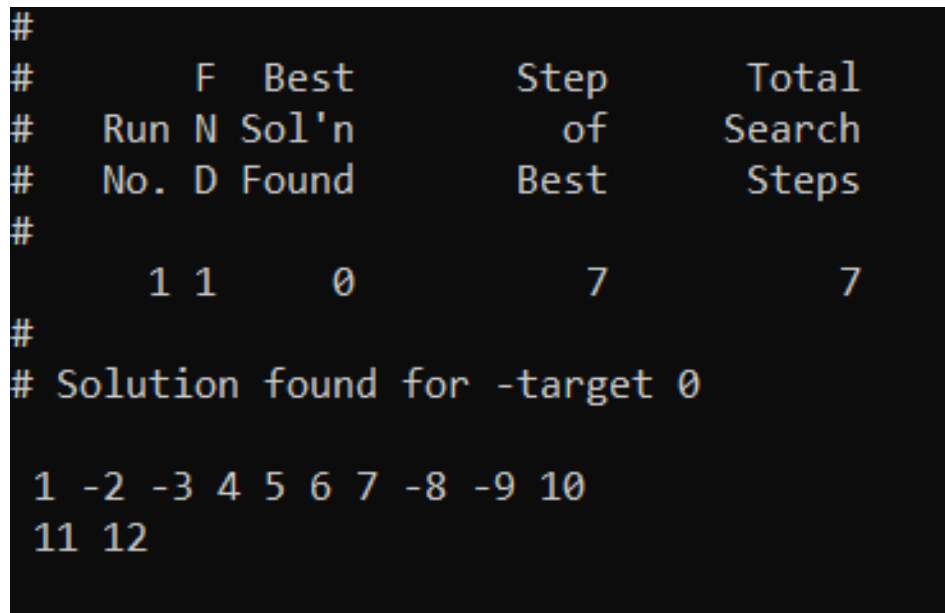
```

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 -1 7 0
9 -2 8 0
10 -3 9 0
11 -10 4 7 0
12 -10 -1 7 0
13 -11 5 8 0
14 -11 -2 8 0
15 -12 6 9 0
16 -12 -3 9 0
17 -4 1 -7 0
18 -5 2 -8 0
19 -6 3 -9 0
20 1 0
21 5 0
22 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           7           7
#
# Solution found for -target 0
#
# 1 -2 -3 4 5 6 7 -8 -9 10
# 11 12

```

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

```
D:\M1 SII\Rep_connaissance\tp\tp1>ubcsat -alg saps -i uf75-01.cnf -solve
#
```

```
#
# 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 10 – Commande et résultats du solveur SAT pour le premier fichier, partie 1

```
D:\M1 SII\Rep_connaissance\tp\tp1>ubcsat -alg saps -i uuf75-01.cnf -solve
#
```

```
1 0 1 535 100000
# No Solution found for -target 0
```

FIGURE 11 – Commande et résultats du solveur SAT pour le premier fichier, partie 2

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 "o" à la fin comme toutes les clauses).
- On execute les Solveur SAT et on affiche le résultat.

Code Source :

```
ubcnf = sys.argv[1]
newfile = ubcnf
base = os.path.splitext(ubcnf)[0]
os.rename(newfile, base + '.txt')
shutil.copy(base+'.txt', 'copy.txt')
os.rename(base+'.txt', base + '.cnf')
file = open('copy.txt', "r")
line = file.readline().split()

# extraire les information du fichier cnf
nbr_litteraux = int(line[2])
print("Le nombre des litteraux utilisé est: "+str(nbr_litteraux))
print("Le nombre de clauses originaux: "+line[3])
line[3] = str(int(line[3])+1)
print("Le nombre de clauses apres le traitement: "+line[3])
s = " "
newline = s.join(line)
newline = newline+'\n'
file.close()
but = 0

litteraux = {'Na': 1, 'Nb': 2, 'Nc': 3, 'Cea': 4, 'Ceb': 5, 'Cec': 6, 'Ma': 7, 'Mb': 8, 'Mc': 9, 'Coa': 10, 'Cob': 11, 'Coc': 12}
print("Les litteraux sont:", litteraux)
print("\n")
```

```

while True:
    litteral = input("Donner le nom de litteral que vous voulez tester: ")

    if litteral in litteraux.keys():
        print("litteral de test valide")
        but = litteraux[litteral]
        break
    print("Litteral n'existe pas")

non_but = -1 * but
with open('copy.txt') as f:
    lines = f.readlines()

lines[0] = newline
f1 = open('copy.txt', "w")
f1.writelines(lines)
f1.close()

# ajouter le non de litteral a la fin
f2 = open('copy.txt', "a")
f2.write('\n'+str(non_but)+' 0')
f2.close()

os.rename('copy.txt', 'copy.cnf')
os.system('ubcsat -alg saps -i copy.cnf -solve > tmp.txt')
check = "# No Solution found for"
infere = False
f3 = open('tmp.txt', "r")
for i in f3:
    if check in i:
        infere = True

```

```

if (infere == True):
    print("BC infère '" + litteral + "'")
else:
    print("BC n'infère pas '" + litteral + "'")
f3.close()
if os.path.isfile('copy_done.cnf'):
    os.remove('copy_done.cnf')
os.rename('copy.cnf', 'copy_done.cnf')

```

Voici l'exécution :

```
PS D:\M1 SII\Rep_connaissance\tp\tp1> python inference.py zoo.cnf
Le nombre des litteraux utilisé est: 12
Le nombre de clauses originaux: 21
Le nombre de clauses apres le traitement: 22
Les litteraux sont: {'Na': 1, 'Nb': 2, 'Nc': 3, 'Cea': 4, 'Ceb': 5, 'Cec': 6, 'Ma': 7, 'Mb': 8, 'Mc': 9, 'Coa': 10, 'Cob': 11, 'Coc': 12}

Donner le nom de litteral que vous voulez tester: Nc
litteral de test valide
BC n'infère pas 'Nc'
PS D:\M1 SII\Rep_connaissance\tp\tp1> 
```

Figure 14 – Résultats du programme, cas d'échec

```
PS D:\M1 SII\Rep_connaissance\tp\tp1> python inference.py zoo.cnf
Le nombre des litteraux utilisé est: 12
Le nombre de clauses originaux: 21
Le nombre de clauses apres le traitement: 22
Les litteraux sont: {'Na': 1, 'Nb': 2, 'Nc': 3, 'Cea': 4, 'Ceb': 5, 'Cec': 6, 'Ma': 7, 'Mb': 8, 'Mc': 9, 'Coa': 10, 'Cob': 11, 'Coc': 12}

Donner le nom de litteral que vous voulez tester: Na
litteral de test valide
BC infère 'Na'
PS D:\M1 SII\Rep_connaissance\tp\tp1> 
```

Figure 14 – Résultats du programme, cas positif

Chapitre 3

TP 2 : Logique Premier ordre

Code Source :

```
public static void main(String[] args) throws ParseException, IOException{
    /*
     * Example 1: Add sorts, constants and predicates to a first-order logic signature
     */
    //Create new FOLSignature with equality
    FolSignature sig = new FolSignature(true);

    //Add sort
    Sort sortAnimal = new Sort("Animal");
    sig.add(sortAnimal);

    //Add constants
    Constant constantEagle = new Constant("eagle",sortAnimal);
    Constant constantGoat = new Constant("goat",sortAnimal);
    sig.add(constantEagle, constantGoat);

    //Add predicates
    List<Sort> predicateList = new ArrayList<Sort>();
    predicateList.add(sortAnimal);
    Predicate p = new Predicate("Flies",predicateList);
    List<Sort> predicateList2 = new ArrayList<Sort>();
    predicateList2.add(sortAnimal);
    predicateList2.add(sortAnimal);
    Predicate p2 = new Predicate("Knows",predicateList2); //Add Predicate Knows(Animal,Animal)
    sig.add(p, p2);
    System.out.println("Signature: " + sig);

    FolParser parser = new FolParser();
    parser.setSignature(sig); //Use the signature defined above
    FolBeliefSet bs = new FolBeliefSet();
    FolFormula f1 = (FolFormula)parser.parseFormula("!Flies(goat)");
    FolFormula f2 = (FolFormula)parser.parseFormula("Flies(eagle)");
    FolFormula f3 = (FolFormula)parser.parseFormula("!Knows(eagle,goat)");
    FolFormula f4 = (FolFormula)parser.parseFormula("==(eagle,goat)");
    FolFormula f5 = (FolFormula)parser.parseFormula("goat == goat");
    bs.add(f1, f2, f3, f4, f5);
    System.out.println("\nParsed BeliefBase: " + bs);

    //Note that belief bases can have signatures larger (but not smaller) than their formulas' signature
    FolSignature sigLarger = bs.getSignature();
    sigLarger.add(new Constant("archaeopteryx",sortAnimal));
    bs.setSignature(sigLarger);
    System.out.println(bs);
    //System.out.println("Minimal signature: " + bs.getMinimalSignature());

    /*
     * Example 3: Use one of the provers to check whether various formulas can be inferred from the knowledge base parsed in Example 2.
     */
    FolReasoner.setDefaultReasoner(new SimpleFolReasoner()); //Set default prover, options are NaiveProver, EProver, Prover9
    FolReasoner prover = FolReasoner.getDefaultReasoner();
    System.out.println("ANSWER 1: " + prover.query(bs, (FolFormula)parser.parseFormula("Flies(goat)")));
    System.out.println("ANSWER 2: " + prover.query(bs, (FolFormula)parser.parseFormula("forall X: (exists Y: (Flies(X) && Flies(Y) && X==Y))")));
    System.out.println("ANSWER 3: " + prover.query(bs, (FolFormula)parser.parseFormula("goat== goat")));
    System.out.println("ANSWER 4: " + prover.query(bs, (FolFormula)parser.parseFormula("goat /= goat")));
    System.out.println("ANSWER 5: " + prover.query(bs, (FolFormula)parser.parseFormula("eagle /= goat")));
}
```

Voici un aperçu de ce que fait chaque partie du code :

1. **Signature** : La première partie du code crée une signature FOL en ajoutant des sorts, des constantes et des prédicats à la signature. Dans cet exemple, un sort "Animal" est ajouté, ainsi que deux constantes ("eagle" et "goat") de ce sort, et deux prédicats ("Flies" et "Knows") prenant des arguments de type "Animal".
2. **Parsage des formules** : La deuxième partie du code utilise un analyseur (parser) FOL pour analyser des formules en utilisant la signature définie précédemment. Les formules analysées sont ajoutées à un ensemble de croyances (FolBeliefSet). Les formules sont représentées à l'aide de la syntaxe FOL.
3. **Utilisation d'un raisonneur** : La troisième partie du code utilise un raisonneur FOL (prover) pour vérifier si différentes formules peuvent être déduites à partir de la base de connaissances (l'ensemble de croyances) analysée dans la partie précédente. Dans cet exemple, le raisonneur utilisé est le "SimpleFolReasoner". Des requêtes sont formulées en utilisant le raisonneur pour vérifier des formules telles que "Flies(goat)" (le goat vole-t-il ?), "forall X: (exists Y: (Flies(X) && Flies(Y) && X/=Y))" (existe-t-il deux animaux différents qui volent ?), etc.

En résumé, ce code montre comment définir une signature FOL, analyser des formules et utiliser un raisonneur pour effectuer des déductions logiques sur des connaissances représentées en logique du premier ordre.

Résultat de m'exécution :

```
Signature: [_Any = {}, Animal = {eagle, goat}], [Knows(Animal,Animal), ==(Any,Any), /==(Any,Any), Flies(Animal)], []
Parsed BeliefBase: { (goat==goat), Flies(eagle), !Knows(eagle,goat), (eagle/=goat), !Flies(goat) }
{ (goat==goat), Flies(eagle), !Knows(eagle,goat), (eagle/=goat), !Flies(goat) }
ANSWER 1: false
ANSWER 2: false
ANSWER 3: true
ANSWER 4: false
ANSWER 5: true
```

Chapitre 4

TP 3 : Logique Modale

Résumé

Dans ce deuxième TP, nous allons simplement vérifier la véracité de certaines formules en utilisant :

- La librairie Java tweety.

Code Source :

NB : On peut pas construire plusieurs monde (monde relations).

```
1 public static void main ( String [] args ) throws ParseException , IOException {
2     MlBeliefSet bs = new MlBeliefSet ();
3     MlParser parser = new MlParser ();
4     FolSignature sig = new FolSignature ();
5     sig.add( new Predicate ("p", 0));
6     sig.add( new Predicate ("q", 0));
7
8     parser.setSignature ( sig );
9
10    bs.add(( RelationalFormula )parser.parseFormula("<>(p)"));
11    bs.add(( RelationalFormula )parser.parseFormula(" [](q)"));
12
13    System.out.println(" Modal knowledge base : " + bs);
14    SimpleMlReasoner reasoner = new SimpleMlReasoner ();
15
16    System.out.println(" [](! p) " + reasoner.query(bs , ( FolFormula ) parser.parseFormula(" [](! p) " ))+"\n");
17    System.out.println(" <>(p && q) "+reasoner.query(bs , ( FolFormula ) parser.parseFormula(" <>( p && q)"))+"\n");
18    System.out.println(" [](p || q) "+reasoner.query(bs , ( FolFormula ) parser.parseFormula(" [](p || q)"))+"\n");
19    System.out.println(" <>(! (p && q)) "+reasoner.query(bs , ( FolFormula ) parser.parseFormula(" <>(! (p && q))"))+"\n");
20    System.out.println(" <>(! (q) && q)+reasoner.query(bs , ( FolFormula ) parser.parseFormula(" <>(! (q) && q)"))+"\n");
21
22 }
23 }
```

FIGURE 2 – Code source TP3

Ce code est un autre exemple d'utilisation de la logique modale à l'aide de la bibliothèque "TweetyProject".

Voici un aperçu de ce que fait chaque partie du code :

1. **Belief Set et Signature** : La première partie du code crée un ensemble de croyances modales (MIBeliefSet) et une signature (FolSignature). La signature contient deux prédicats, "p" et "q", sans aucun argument.
2. **Parsage des formules** : Le code utilise un analyseur (parser) modale (MIParser) pour analyser des formules modales en utilisant la signature définie précédemment. Les formules analysées sont ajoutées à l'ensemble de croyances.
3. **Utilisation d'un raisonneur** : Le code utilise un raisonneur modale (SimpleMIRasoner) pour vérifier différentes formules modales à partir de l'ensemble de croyances analysé. Les formules sont formulées en utilisant le raisonneur pour vérifier des formules modales telles que "" ($\neg p$ est nécessairement vrai), "<>(p && q)" (p et q sont possibles), "[](p || q)" (p ou q est nécessairement vrai), etc.

En résumé, ce code montre comment définir un ensemble de croyances modales, analyser des formules modales et utiliser un raisonneur pour effectuer des déductions logiques sur des connaissances représentées en logique modale.

Résultat de m'exécution :

```
Modal knowledge base : { <>(p), [](q) }
[](! p) false

<>(p && q) true

[](p || q) true

<>(! (p && q)) false

<>(! (q) && q) false
```

FIGURE 3 – Résultat d'exécution

Chapitre 5

TP 4 : Logique des défauts

Résumé

Dans ce TP, nous allons implémenter un exemple en utilisant la toolbox «Extension Calculator»

Toolbox 1 : Extension Calculator

Voici l'interface de l'outil

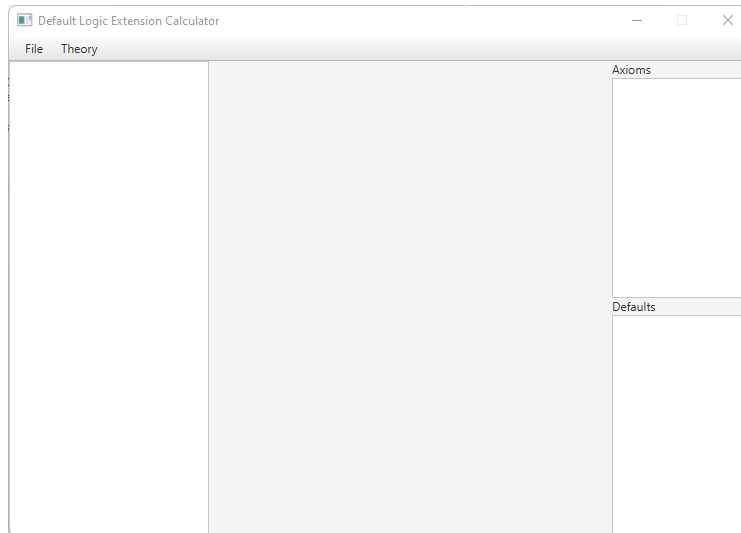


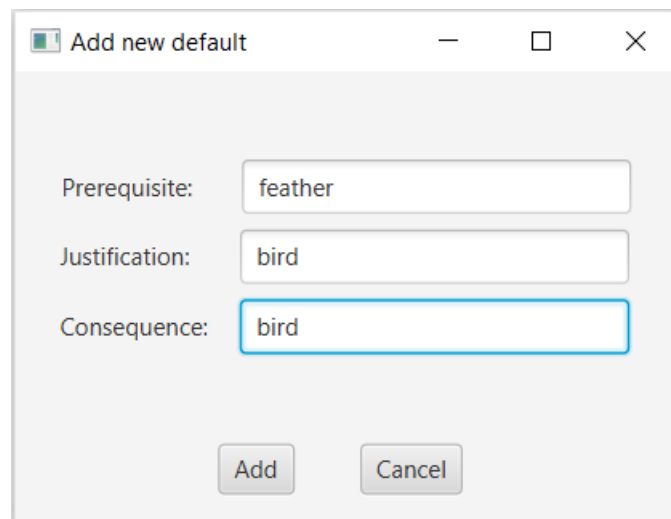
FIGURE 18 – Interface de l'outil Extension Calculator

Exemple :

Les défauts :

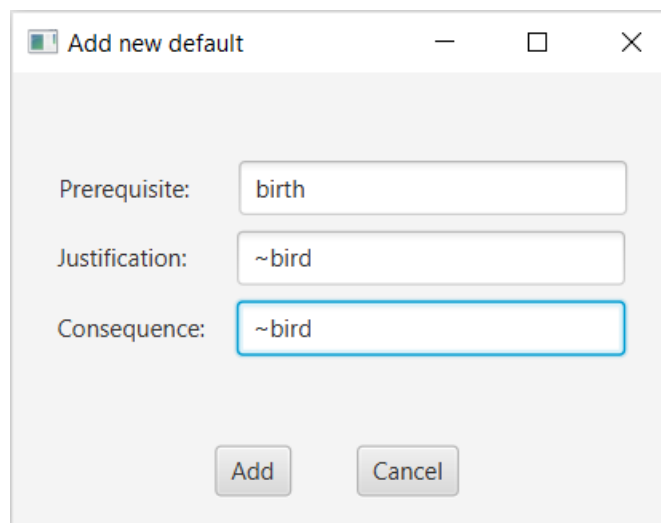
$D1 = \{ \text{feather} : \text{bird} / \text{bird} \}$

$D2 = \{ \text{birth} : \neg \text{bird} / \neg \text{bird} \}$



The screenshot shows a dialog box titled "Add new default" with a standard window control bar (minimize, maximize, close). Inside the dialog, there are three input fields labeled "Prerequisite:", "Justification:", and "Consequence:". The "Prerequisite:" field contains the text "feather". The "Justification:" field contains the text "bird". The "Consequence:" field contains the text "bird" and is highlighted with a blue border. At the bottom of the dialog, there are two buttons: "Add" and "Cancel".

FIGURE 19 – Création du défaut d1



The screenshot shows a dialog box titled "Add new default" with a standard window control bar (minimize, maximize, close). Inside the dialog, there are three input fields labeled "Prerequisite:", "Justification:", and "Consequence:". The "Prerequisite:" field contains the text "birth". The "Justification:" field contains the text "~bird". The "Consequence:" field contains the text "~bird" and is highlighted with a blue border. At the bottom of the dialog, there are two buttons: "Add" and "Cancel".

Figure 19 – Création du défaut d2

$$w = \{ birth, feather \}$$

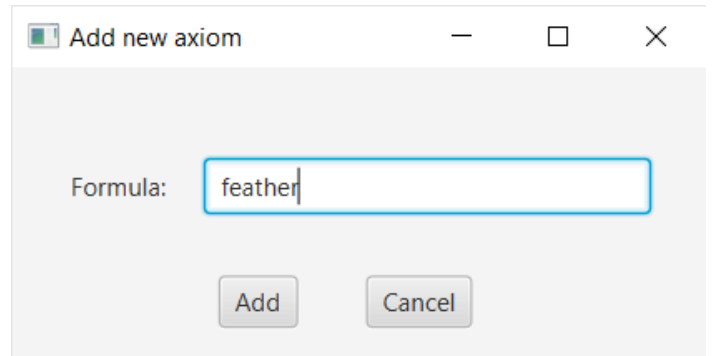
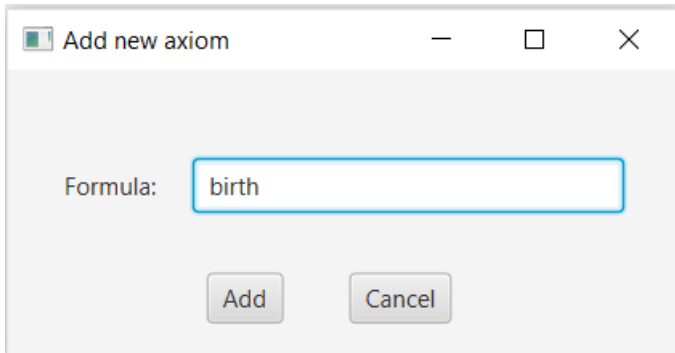


FIGURE 21 – Création du monde w_1

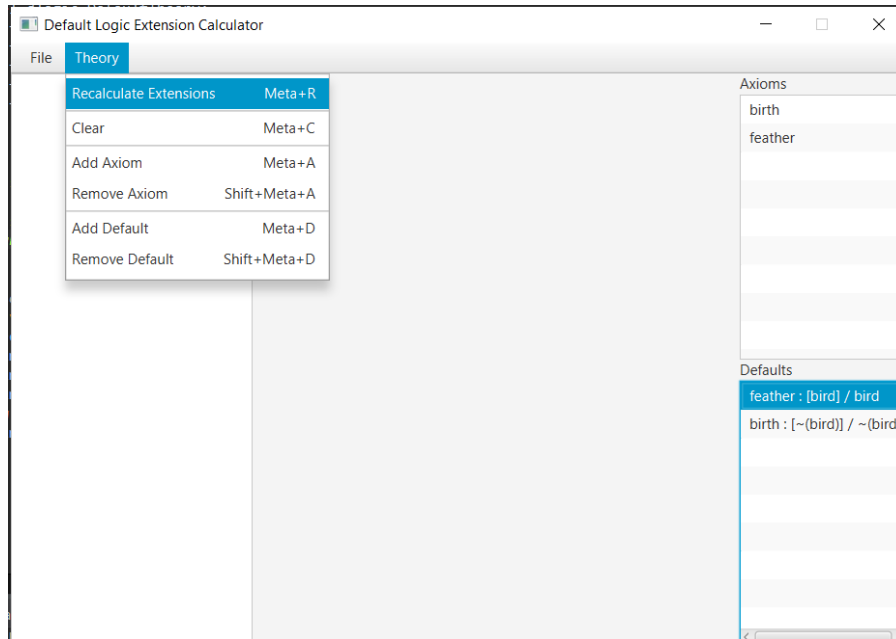


FIGURE 22 –Interface après la création w_1

Si la colonne à gauche est égale à la colonne des axiomes, alors il n'y a pas d'extension.

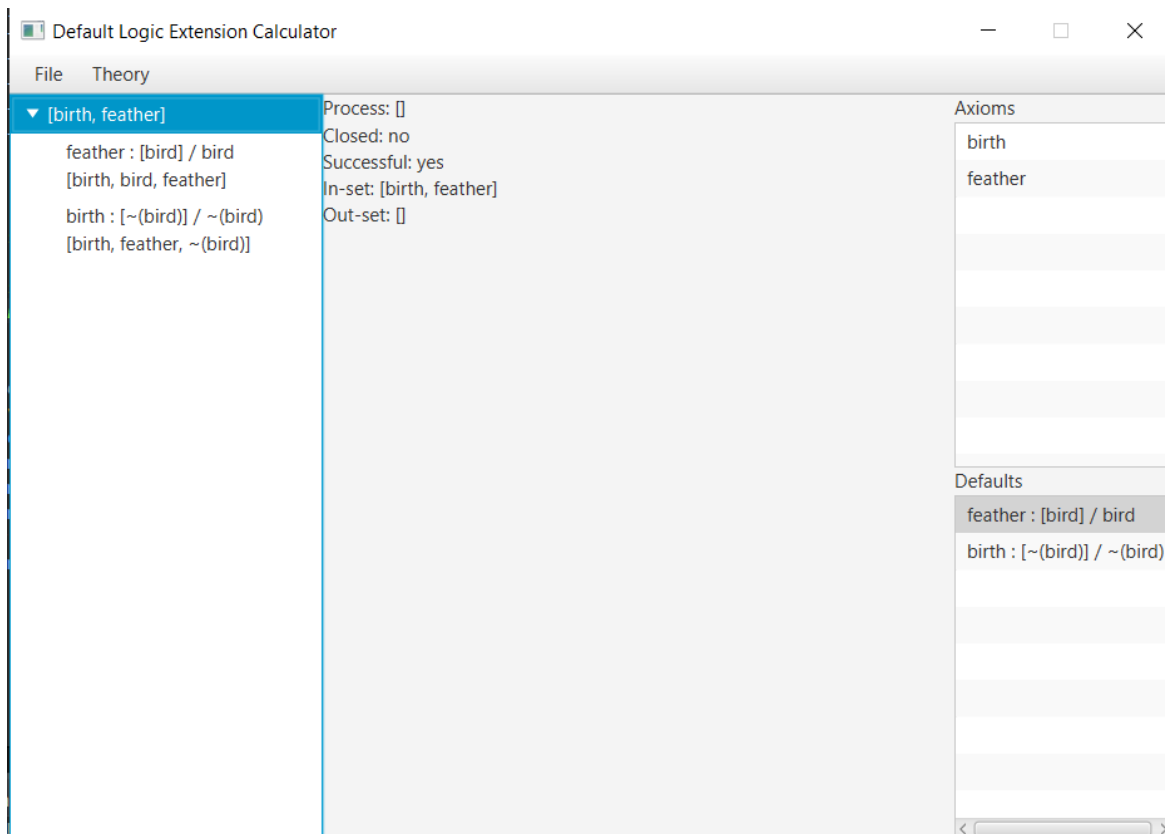


FIGURE 23 – Résultat de l'extension de w_2

Les extensions sont à gauche :

▼ [birth, feather]	Process: [feather : [bird] / bird]
feather : [bird] / bird	Closed: yes
[birth, bird, feather]	Successful: yes
	In-set: [birth, bird, feather]
birth : [~(bird)] / ~(bird)	Out-set: [~(bird)]
[birth, feather, ~(bird)]	

▼ [birth, feather]	Process: [birth : [~(bird)] / ~(bird)]
feather : [bird] / bird	Closed: yes
[birth, bird, feather]	Successful: yes
	In-set: [birth, feather, ~(bird)]
birth : [~(bird)] / ~(bird)	Out-set: [bird]
[birth, feather, ~(bird)]	

FIGURE 24 – Extension 1 et 2

Chapitre 6

TP 5 : Réseaux sémantiques

1 Introduction

Dans ce TP, nous allons nous intéresser à l'implémentation de réseaux sémantiques, et plus précisément à l'implémentation de différents algorithmes des réseaux sémantiques.

2 Réseaux sémantiques

Un réseau sémantique est un graphe marqué destiné à la représentation des connaissances. Pour notre TP, un réseau sémantique est défini dans un fichier JSON récupéré via le site qui a été joint dans le TP ou chaque nœud possède un label et un id qui sont par la suite utilisés pour représenter les différentes relations entre les nœuds. Nous allons réaliser les différents algorithmes avec le langage python étant particulièrement adapté pour travailler avec les fichiers JSON.

Voici l'exemple de réseau sémantique que nous allons utiliser pour le reste du TP.

3 Partie 1 : implémenter l'algorithme de propagation de marqueurs dans les réseaux sémantiques

Dans cette partie nous allons implémenter l'algorithme de propagation de marqueurs vu en cours :

```
def propagation_de_marqueurs(reseau_semantique, node1, node2, relation):
    nodes = reseau_semantique["nodes"]

    solutions_found = []

    for i in range(min(len(node1), len(node2))):
        solution_found = False

        try:
            M1 = [node for node in nodes if node["label"] == node1[i]][0]
            M2 = [node for node in nodes if node["label"] == node2[i]][0]

            edges = reseau_semantique["edges"]

            propagation_edges = [edge for edge in edges if (edge["to"] == M1["id"] and edge["label"] == "is a")]

            while len(propagation_edges) != 0 and not solution_found:

                temp_node = propagation_edges.pop()
                temp_node_contient_edges = [edge for edge in edges if (edge["from"] == temp_node["from"] and edge["label"] == relation)]
                solution_found = any(d["to"] == M2["id"] for d in temp_node_contient_edges)

                if not solution_found:
                    temp_node_is_a_edges = [edge for edge in edges if (edge["to"] == temp_node["from"] and edge["label"] == "is a")]
                    propagation_edges.extend(temp_node_is_a_edges)

            solutions_found.append(get_label(reseau_semantique, M2, relation) if solution_found else "il n'y a pas un lien entre les 2 noeuds")

        except IndexError:
            solutions_found.append("Aucune reponse n'est fournie par manque de connaissances.")

    return(solutions_found)

def get_label(reseau_semantique, node, relation):
    node_relation_edges = [edge["from"] for edge in reseau_semantique["edges"] if (edge["to"] == node["id"] and edge["label"] == relation)]
    node_relation_edges_label = [node["label"] for node in reseau_semantique["nodes"] if node["id"] in node_relation_edges]
    reponse = "il y a un lien entre les 2 noeuds : " + ", ".join(node_relation_edges_label)
    return reponse
```

Figure 1 – Algorithme de propagation de marqueur

Pour cet Algorithme nous allons utiliser le réseau sémantique fourni par le lien donnée dans la série de TP détaillé dans l'image suivante :

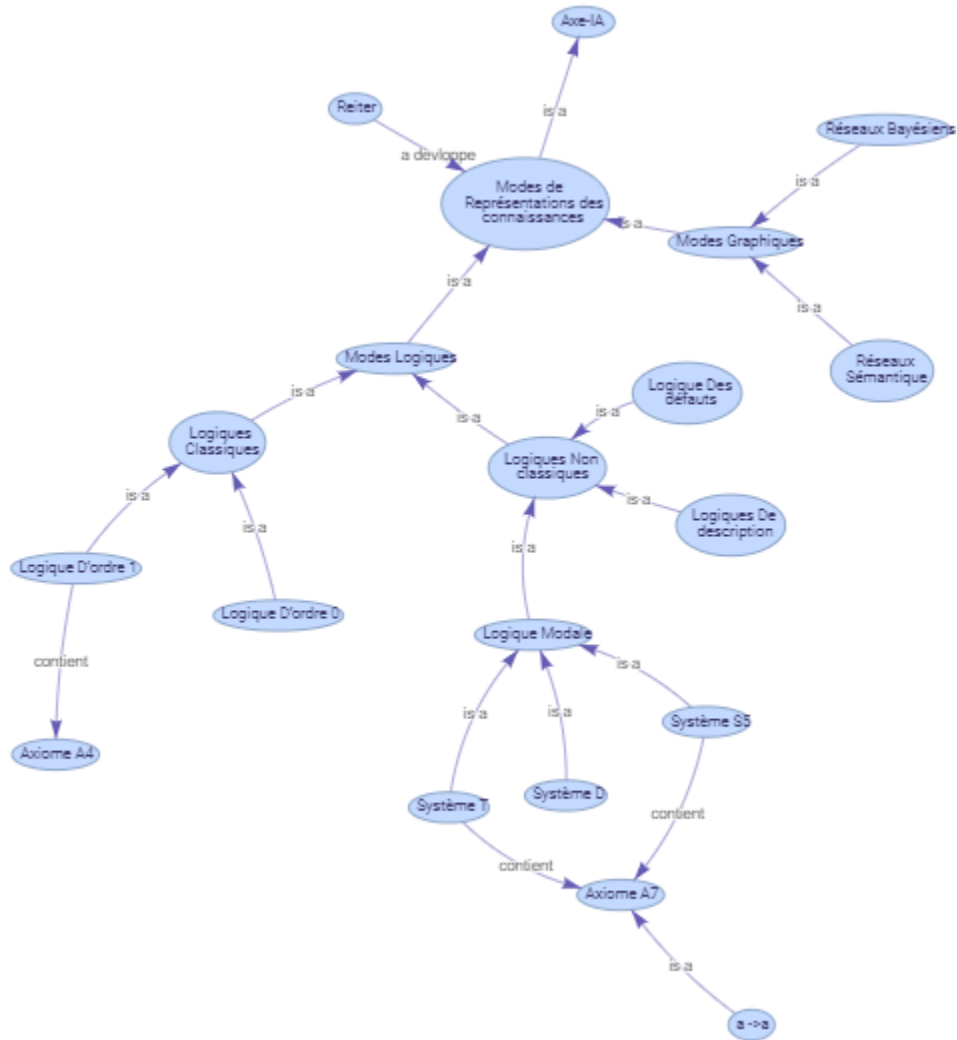


Figure 2 – Réseau sémantique utilisé pour la partie 1 du TP

Utilisant plusieurs nœuds marqués en entrée, on obtient les résultats suivants :

```
Modes de Representations des connaissances contient Axiome A7
il y a un lien entre les 2 noeuds : Systeme T, Systeme S5
Modes de Representations des connaissances contient Axiome A4
il y a un lien entre les 2 noeuds : Logique D ordre 1
Modes de Representations des connaissances contient Axe-IA
il n'y a pas un lien entre les 2 noeuds
Modes de Representations des connaissances contient Axiome A9
Aucune reponse n'est fournie par manque de connaissances.
```

Figure 3 – Résultat obtenu par l'algorithme de propagation de marqueurs

On voit qu'effectivement l'algorithme à trouver le lien entre ces deux nœuds.

4 Partie 2 : implémenter l'algorithme d'héritage

Dans cette partie, nous allons implémenter l'algorithme d'héritage vu en cours :

```
def get_label(reseau_semantique, node_id):
    label = [node["label"] for node in reseau_semantique["nodes"] if node["id"] == node_id]
    return " ,".join(label)

def heritage(reseau_semantique, name):
    fin = False

    nodes = reseau_semantique["nodes"]
    edges = reseau_semantique["edges"]

    #get node where given name
    node = [node for node in nodes if node["label"] == name][0]
    #get all inherited nodes IDs
    direct_edges = [edge["to"] for edge in edges if (edge["from"] == node["id"] and edge["label"] == "is_a")]
    all_edges = []
    properties = []
    while not fin:
        n = direct_edges.pop()
        #get inherited nodes label
        all_edges.append(get_label(reseau_semantique, n))
        #if the inherited are also inherited by other nodes
        direct_edges.extend([edge["to"] for edge in edges if (edge["from"] == n and edge["label"] == "is_a")])

        #for inference
        properties_nodes = [edge for edge in edges if (edge["from"] == n and edge["label"] != "is_a")]

        for pn in properties_nodes:
            properties.append(" : ".join([pn["label"], get_label(reseau_semantique, pn["to"])]))
        if len(direct_edges) == 0:
            fin = True

    return all_edges, properties
```

Figure 4 – Algorithme d'héritage

On obtient le résultat suivant :

```
starting node : Systeme D
Resultat de l'inference utiliser:
Systeme D
Logique Modale
Logiques Non classiques
Modes Logiques
Modes de Representations des connaissances
Axe-IA
Deduction des priorites:
empty
```

Figure 6 – Résultat obtenu par l’algorithme d’héritage

5 Partie 3 : implémentez un algorithme qui permet d’inhiber la propagation dans le cas des liens d’exception

Cet algorithme est très similaire à celui de la partie 1 la seule différence est dans le fait qu’on ne prend pas en compte les arcs de types exception.

Chapitre 7

TP 6 : Logique des descriptions

Résumé

Dans ce TP, nous allons manipuler des données ontologiques en logique des descriptions en optant pour le raisonneur **Hermit** sous java.

Nous commençons par l'exemple qu'on a utilisé :

Concepts

Vehicle:	Represents the concept of a vehicle.
Car:	Represents the concept of a car, which is a subclass of Vehicle.
Sedan:	Represents the concept of a sedan, which is a subclass of Car.
SUV:	Represents the concept of an SUV, which is a subclass of Car.

Roles

Electric:	Represents the property or role indicating whether a vehicle is electric or not.
Manufacturer:	Represents the role indicating the manufacturer of a vehicle.
Color:	Represents the role indicating the color of a vehicle.

Instances

Tipo:	Represents an individual instance of a sedan.
X6 :	Represents an individual instance of an SUV.
BMW:	Represents an individual instance of a vehicle manufacturer.
Toyota:	Represents another individual instance of a vehicle manufacturer.
RedSedan:	Represents an individual instance of a red sedan.

Voici le code source :

```
public static void main(String[] args) {  
    // Path to the .owl file  
    String FILE = "D:/desclogic5.owl";  
  
    // Create an instance of the OWL API manager  
    OWLOntologyManager manager = OWLManager.createOWLOntologyManager();  
  
    try {  
        File exemplFile = new File(FILE);  
        // charger le .owl  
        OWLOntology ontology = manager.loadOntologyFromOntologyDocument(exemplFile);  
  
        // T-BOX  
        System.out.println("concepts:");  
        for (OWLClass owlClass : ontology.getClassesInSignature()) {  
            System.out.println(owlClass.getIRI());  
        }  
  
        // roles  
        System.out.println("\nRoles:");  
        for (OWLObjectProperty objectProperty : ontology.getObjectPropertiesInSignature()) {  
            System.out.println(objectProperty.getIRI());  
        }  
  
        // A-BOX  
        System.out.println("\nIndividuals:");  
        for (OWLNamedIndividual individual : ontology.getIndividualsInSignature()) {  
            System.out.println(individual.getIRI());  
        }  
    } catch (OWLOntologyCreationException e) {  
        e.printStackTrace();  
    }  
}
```

Voici le résultat de l'exécution :

```
concepts:  
file:/D:/desclogic.owl#Vehicle  
file:/D:/desclogic.owl#Sedan  
file:/D:/desclogic.owl#Car  
file:/D:/desclogic.owl#SUV  
file:/D:/desclogic.owl#VehicleManufacturer  
  
Roles:  
file:/D:/desclogic.owl#Manufacturer  
  
Individuals:  
file:/D:/desclogic.owl#BMW  
file:/D:/desclogic.owl#RedSedan  
file:/D:/desclogic.owl#Tipo  
file:/D:/desclogic.owl#Toyota  
file:/D:/desclogic.owl#X6
```

FIGURE 1 – Résultats déduits sur les instances

On peut voir que le raisonneur a pu conclure *VehicleManufacturer*.

Chapitre 8

Conclusion Générale

Au cours de ces travaux pratiques, nous avons eu l'occasion d'explorer les diverses logiques sur lesquelles un système informatique peut se baser. Nous avons acquis des compétences en matière de modélisation de problèmes réels, de génération d'inférences, d'utilisation de raisonneurs pour évaluer la véracité des formules, de représentation graphique des informations, et enfin, de déduction de faits.