République Algérienne Démocratique et Populaire
Ministère de l'Enseignement supérieur et de la Recherche scientifique
Université des Sciences et de la Technologie Houari Boumediene

Master 1 SII
Rapport de TP
REPRÉSENTATION DE CONNAISSANCE ET RAISONNEMENT

Réalisé par :
FLICI Syrine Mahdia
TOLBA Yasmine
TOUMACHE Doudja Rania

2021/2022

TP1 :

I. Etape 1 :

Dans l'étape 1 nous avons créé un dossier contenant un solver ubcsat

II. Etape 2 :

Après le téléchargement du fichier ubcsat, on a exécuté le fichier : sample.cnf avec la requête ubcsat -alg saps -i sample.cnf -solve

```
C:\Users\HP\Desktop\SII\M1\S2\RCR1\TP -RCR1-\TP1\ubcsat>ubcsat -alg saps -i sample.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 1551140317
#  -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
#
# 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       7296       7296
#
# Solution found for -target 0
```

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


Variables = 250
Clauses = 1065
TotalLiterals = 3195
TotalCPUTimeElapsed = 0.012
FlipsPerSecond = 607996
RunsExecuted = 1
SuccessfulRuns = 1
PercentSuccess = 100.00
Steps_Mean = 7296
Steps_CoeffVariance = 0
Steps_Median = 7296
CPUTime_Mean = 0.0120000839233
CPUTime_CoeffVariance = 0
CPUTime_Median = 0.0120000839233
```

Voici l'exemple donné dans l'énoncé de ce TP :

test1.cnf

```
p cnf 5 9
2 -3 0
-3 0
1 -2 -3 4 0
-1 -4 0
-1 -2 3 5 0
2 -5 0
-3 4 -5 0
1 2 5 0
-3 5 0
%
0
```

Les résultats sur ubcsat:

```
C:\Users\YsPC\Desktop\school\TP RCR\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
#  -gtimeout 0
#  -noimprove 0
#  -target 0
#  -wtarget 0
#  -seed 1465832143
#  -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
#
# 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          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 = 0.00100016593933
CPUTime_CoeffVariance = 0
CPUTime_Median = 0.00100016593933
```

L’exécution se déroule comme prévu, puisque nous savons que cette base de connaissances est satisfiable.

On fait la même chose avec test2.cnf

```
p cnf 5 12
2 -3 0
-3 0
1 -2 -3 4 0
-1 -4 0
2 -4 0
1 3 0
-1 -2 3 5 0
2 -5 0
-3 4 -5 0
1 2 5 0
3 5 0
-5 0
%
0
```

Et les résultats sur ubcsat:

```
C:\Users\YsPC\Desktop\school\TP RCR\UBCSAT>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 1465973670
#  -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
#
# 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          4     100000
# No Solution found for -target 0


Variables = 5
Clauses = 12
TotalLiterals = 28
TotalCPUTimeElapsed = 0.011
FlipsPerSecond = 9090977
RunsExecuted = 1
SuccessfulRuns = 0
PercentSuccess = 0.00
Steps_Mean = 100000
Steps_CoeffVariance = 0
Steps_Median = 100000
CPUTime_Mean = 0.010999917984
CPUTime_CoeffVariance = 0
CPUTime_Median = 0.010999917984
```

Encore une fois, comme prévu, aucune solution n'a été trouvée, car nous savons que cette base de connaissance n'est pas satisfiable.

III. Etape 3 :

Traduction de la base de connaissances relative aux connaissances zoologiques :

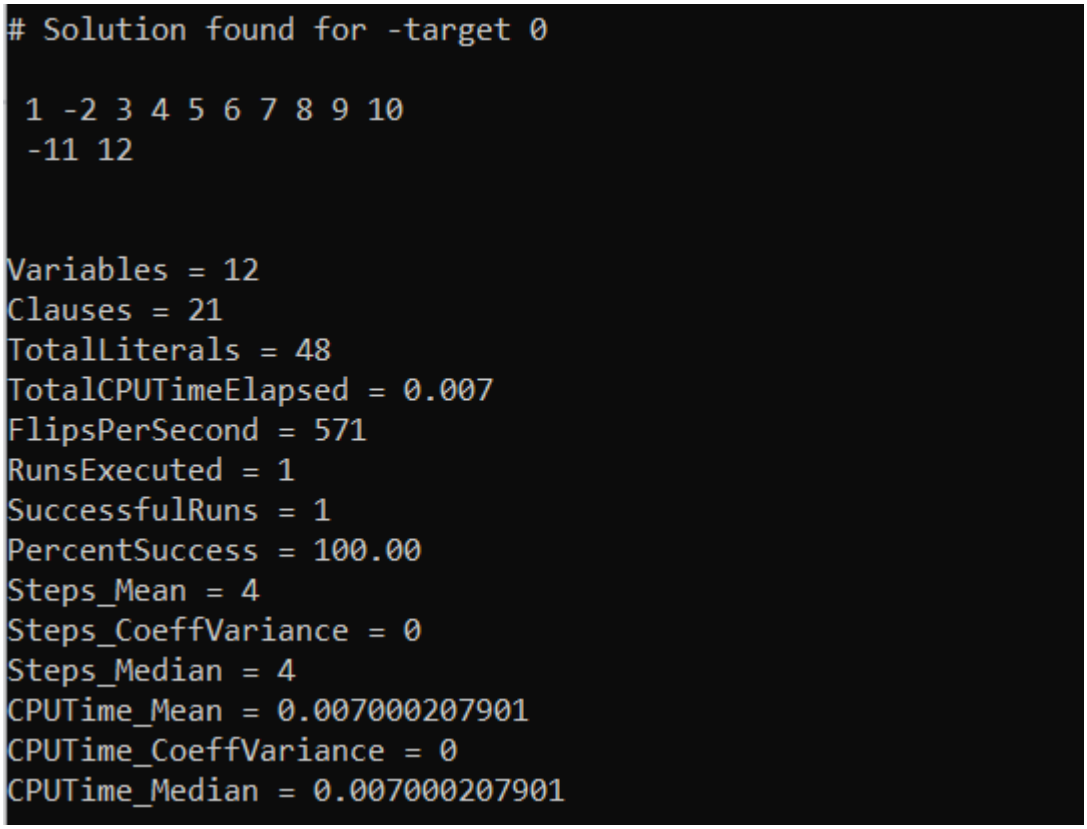
Na : 1, Nb : 2, Nc : 3, Cea : 4,
Ceb : 5, Cec : 6, Ma : 7, Mb : 8,
Mc : 9, Coa : 11, Cob : 12, Coc : 13.

¬Na ∨ Cea ; ¬Nb ∨ Ceb ; ¬Nc ∨ Cec; ¬Cea ∨ Ma;
¬Ceb ∨ Mb ; ¬Cec ∨ Mc ; ¬Na ∨ Coa ; ¬Nb ∨ Cob ;
¬Nac ∨ Coc ; ¬Cea ∨ Na ∨ ¬Coa; ¬Ceb ∨ Nb ∨ ¬Cob;
¬Cec ∨ Nc ∨ ¬Coc ; ¬Ma ∨ Cea ∨ Coa ; ¬Ma ∨ ¬Na ∨ Coa ;
¬Mb ∨ Ceb ∨ Cob ; ¬Mb ∨ ¬Nb ∨ Cob ; ¬Mc ∨ Cec ∨ Coc ;
¬Mc ∨ ¬Nc ∨ Coc.

On aura le fichier cnf suivant :

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

Avec l'exécution : la base de connaissances est satisfiable avec SAT solveur



Exécution de quelques fichiers cnf :

Un fichier satisfiable

```
# 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


Variables = 75
Clauses = 325
TotalLiterals = 975
TotalCPUTimeElapsed = 0.011
FlipsPerSecond = 40272
RunsExecuted = 1
SuccessfulRuns = 1
PercentSuccess = 100.00
Steps_Mean = 443
Steps_CoeffVariance = 0
Steps_Median = 443
CPUTime_Mean = 0.0110001564026
CPUTime_CoeffVariance = 0
CPUTime_Median = 0.0110001564026
```

Exécution d’un fichier non satisfiable

```
# No Solution found for -target 0


Variables = 50
Clauses = 218
TotalLiterals = 654
TotalCPUTimeElapsed = 0.045
FlipsPerSecond = 2222218
RunsExecuted = 1
SuccessfulRuns = 0
PercentSuccess = 0.00
Steps_Mean = 100000
Steps_CoeffVariance = 0
Steps_Median = 100000
CPUTime_Mean = 0.0450000762939
CPUTime_CoeffVariance = 0
CPUTime_Median = 0.0450000762939
```

IV. Etape 4 :

Après la récupération de la base de connaissance, il faut d'abord tester si elle est satisfiable, si ce n'est pas le cas enlève une exception, dans le cas où la BC est satisfiable, alors nous calculons la négation de la littérale, elle ainsi est inféré dans la BC selon sa satisfiabilité.

Il va falloir prendre en considération le nombre des clauses dans la première ligne du fichier cnf lors de l'ajout de la nouvelle littérale. Une autre chose à vérifier est si la nouvelle formule contient un chiffre plus grand que le nombre des formules.

Ci-dessus le code source en python.

1- La fonction de la négation de la littérale

```python
def negation_litterale(litterale):
    # premierement on separe le litterale en plusieurs partie
    litterale = litterale.split(" ")
    #on convertit en entier en enlevant le 0
    litterale = [int(i) for i in litterale if i != "0"]
    #la négation :
    litterale = [-i for i in litterale]
    negation_litterale = ""
    for i in litterale:
        negation_litterale += "{} 0\n".format(i)
    # on retoune la negation du litteralee, sa taille, ainsi que son nombre max
    return negation_litterale, len(litterale), max([abs(i) for i in litterale])
```

2- La fonction de l'insertion de la littérale dans la BC

```
def insertion_litterale(cnf_path_file, litterale):
    negation_litterale, nb_lignes, var_max = negation_litterale(litterale)
    #initialisation de la base de connaissance
    with open(cnf_path_file, 'r') as cnf_file:
        BC = cnf_file.read()
    #nous traitons le fichier cnf en enlevant tout ce qui est commentaire,
    # sauts de lignes, ou meme espacement
    BC = BC.split("\n")
    i = 0
    while (BC[i] == "" or BC[i][0] != "p"):
        i+=1
    BC[i].replace("\t", " ")

    #réajouter les clauses apres le traitement !
    f1, f2, var_nb, nb_clause = [f for f in BC[i].split(" ") if f!=""]
    nb_clause, var_nb = (int(nb_clause), int(var_nb))
    nb_clause += nb_lignes
    #test de nombre des clauses dans le fichier
    if var_max > var_nb:
        var_nb = var_max

    BC[i] = " ".join([f1, f2, str(var_nb), str(nb_clause)])
    BC = "\n".join(BC)
    if BC[-1] != '\n':
        BC += '\n'
    #ajout de la negation du litterale
    BC += negation_litterale
    temp_BC = os.path.basename(cnf_path_file)[:-4] + "_temporary.cnf"
    with open(temp_BC, 'w') as temporary_cnf:
        temporary_cnf.write(BC)
```

3- La fonction de satisfiabilité

```
def solver(cnf_path_file):
    return subprocess.check_output(['ubcsat', '-alg', 'saps', '-i', '{}'.format(cnf_path_file), '-solve'])

def satisfiable(cnf_path_file):
    return not (b"No Solution found" in solver(cnf_path_file))
```

4- La fonction main qui applique l'algorithme

```
def algorithme(cnf_path_file,litterale):
    #test de la satisfiabilite
    if satisfiable(cnf_path_file):
        #insertion du litterale
        insertion_litterale(cnf_path_file, litterale)
        temp_BC = os.path.basename(cnf_path_file)[:-4] + "_temporraire.cnf"
        #test de la satisfiabilite
        if satisfiable(temp_BC):
            print("la base de connaissance {} infére le litterale {}".format(os.path.basename(cnf_path_file), litterale))
        else:
            print("la base de connaissance {} n'infére pas le litterale {}".format(os.path.basename(cnf_path_file), litterale))
        # supprimer le fichier temporraire
        os.remove(temp_BC)
    else: #raise of error !
        raise ValueError("veuillez introduire une Base de Connaissance satisfiable svp !")
```

TP2 :

Introduction

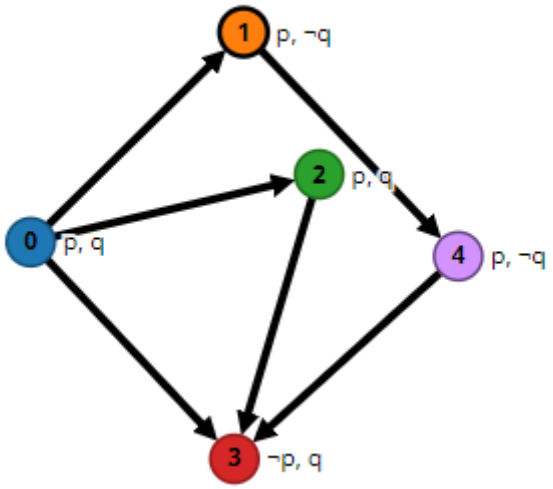
Dans ce TP nous allons vérifier la véracité des formules des exercices 2 et 4 vu en TD, en utilisant deux outils différents, ‘Modal logic playground’, ainsi que la librairie ‘Tweety’ du JAVA.

1. Modal Logic Playground

L’outil ne permet pas d’introduire un nombre différent de variable dans chaque monde, pour cela dans le cas où il y a une seule variable dans un monde nous avons mis les autres variables négatives.
Les résultats de cette partie peuvent alors ne pas être cohérents.

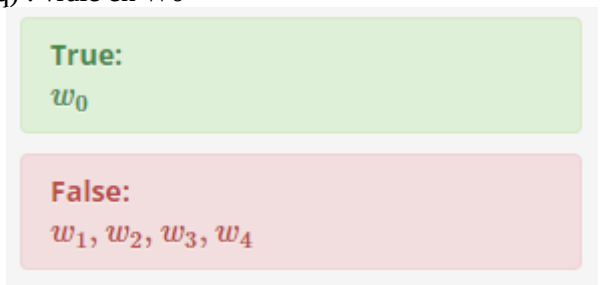
1.1. Exercice 2

Le monde :

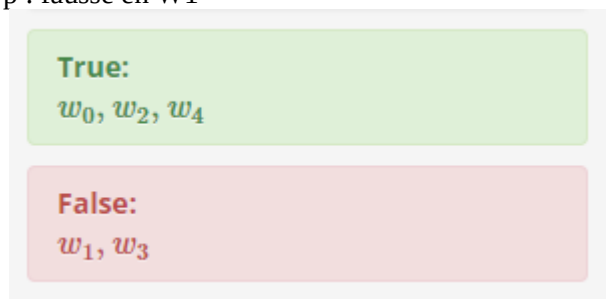


Les tests des différentes formules

- W0 |== ◊ (p ∧ q) : vraie en W0



- W1 |== ¬□p : fausse en W1



- W2 |== □(p ⊃ q) équivalente à □ (¬p ∨ q) : vraie en W2

True:
w_2, w_3, w_4

False:
w_0, w_1

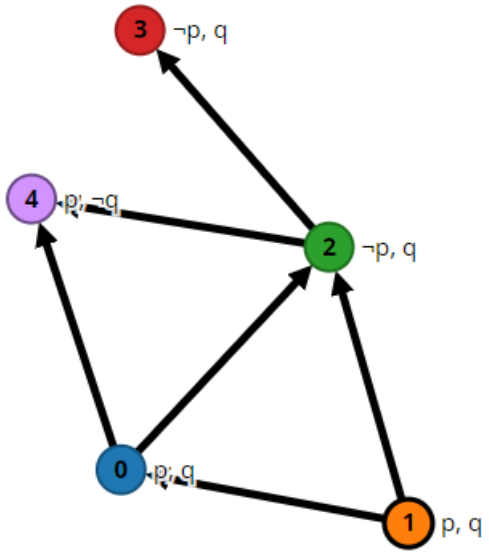
- □(q ∧◇¬p) : vraie en W3 et fausse en W4

True:
w_3

False:
w_0, w_1, w_2, w_4

1.2. Exercice 4

En représentera le ‘a’ par ‘p’ et ‘b’ par ‘q’

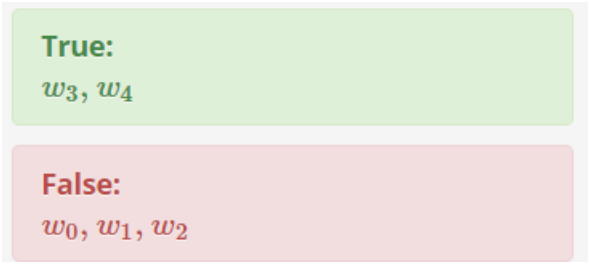


- W0 |== □ (¬a∨¬b) == □ (¬p ∨ ¬q) : vraie en W0

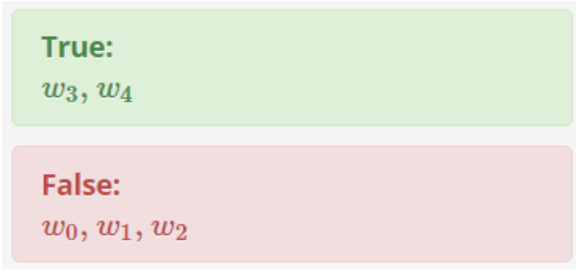
True:
w_0, w_2, w_3, w_4

False:
w_1

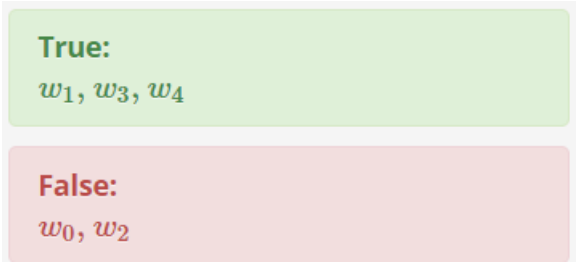
- W2 |== □◇¬b == □◇¬q : fausse en W2



- W1 |== ¬◇ (a⊃b) équivalente à ¬◇ (¬a ∨ b) == ¬◇ (¬p ∨ q) : fausse en W1



- W4|== □◇b == □◇q : vraie en W4



2. Tweety

2.1. Exercice 2

```
=======================EXERCICE 2 SERIE 3 : LEOGIQUE MODALE=============================================
M,w1 |== <>(p&&q) : true
M,w2 |== 7[](p) : false
M,w3 |== [](p=>q) : true
M,w5 |== [](q && <>7p) : false
```

```
MlParser parser = new MlParser();
FolSignature sig = new FolSignature();
sig.add(new Predicate("p",0));
sig.add(new Predicate("q",0));
parser.setSignature(sig);
MlBeliefSet w1 = parser.parseBeliefBase("type(p) \n type(q) \n"+"(p) \n"+"(q) \n");
MlBeliefSet w2 = parser.parseBeliefBase("type(p) \n type(q) \n"+"(p) \n");
MlBeliefSet w3 = parser.parseBeliefBase("type(p) \n type(q) \n"+"(p) \n"+"(q) \n");
MlBeliefSet w4 = parser.parseBeliefBase("type(p) \n type(q) \n"+"(q) \n");
MlBeliefSet w5 = parser.parseBeliefBase("type(p) \n type(q) \n"+"(p) \n");
SimpleMlReasoner reasoner = new SimpleMlReasoner();
FolFormula f1 = (FolFormula) parser.parseFormula("p&&q");
FolFormula f2 = (FolFormula) parser.parseFormula("[](p)");
FolFormula f3 = (FolFormula) parser.parseFormula("[](p=>q)");
FolFormula f4 = (FolFormula) parser.parseFormula("[](q&&(<>(!p)))");
MlBeliefSet[] Relation1= {w2,w3,w4};
MlBeliefSet[] Relation2= {w5};
MlBeliefSet[] Relation3= {w4};
MlBeliefSet[] Relation5= {w4};
```

TP3 :

I. Introduction

Nous allons étudier les différents outils de test basé sur le raisonnement de la logique des défauts sur les exercices de la série 4

II. Outil : DefaultLogicModelCheck

L'outil est disponible dans le répertoire : https://github.com/edm92/defaultlogic

Nous allons tester sur les exercices 1, 2, 3 et 6, vu que l'outil ne garantit pas la notion de priorité entre les défauts et la notion des prédicats.

Nous allons tester pour chaque monde les différents états possibles, les cas où il y aura un match et où il n'y aura pas.

- Exercice 1

Exercice 1:
Soit l'ensemble de défauts D={d1,d2} avec $d1 = A: B/C$ et $d2 = A:\neg C/D$.
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, $\neg B \wedge C$ }

- Avec le monde W1
  Avec ces défauts n'y aura pas d'extensions possibles

```
Given the world:
        ~A
 And the rules
        [(A):(B) ==> (C)] , [(A):(~C) ==> (D)]
Possible Extensions
-------------------------------------------------------------------
```

```java
public static void EXO1W1() {
    WorldSet myWorld = new WorldSet(); // Empty World
    myWorld.addFormula("~A");

    DefaultRule rule1 = new DefaultRule();
    rule1.setPrerequisite("A");
    rule1.setJustificatoin("B");
    rule1.setConsequence("C");

    DefaultRule rule2 = new DefaultRule();
    rule2.setPrerequisite("A");
    rule2.setJustificatoin("~C");
    rule2.setConsequence("D");

    RuleSet myRules = new RuleSet();
    myRules.addRule(rule1);
    myRules.addRule(rule2);

    DefaultReasoner loader = new DefaultReasoner(myWorld, myRules);
    HashSet<String> extensions = loader.getPossibleScenarios();

    a.e.println("Given the world: \n\t" + myWorld.toString()
            + "\n And the rules \n\t" + myRules.toString());

    a.e.println("Possible Extensions");
    for (String c : extensions) {
        a.e.println("\t Ext: Th(W U (" + c + "))");
        // Added closure operator
        a.e.incIndent();
        WFF world_and_ext = new WFF("(( " + myWorld.getWorld() + " ) & ("
                + c + "))");
        a.e.println("= " + world_and_ext.getClosure());
        a.e.decIndent();

    }
}
```

Résultat en TD :

Les deux défauts ne sont pas générateurs d'extension.

- Avec le monde W2

```
------------------------------------------------------------
Trying  A & ~B
Trying  A & ~B
This example is titled Commitment to assumptions
Given the world:
        A & ~B
 And the rules
        [(A):(B) ==> (C)] , [(A):(~C) ==> (D)]
Possible Extensions
         Ext: Th(W U (D))
        = D & ~B &  A
------------------------------------------------------------
```

```java
public static void EXO1W2() {
    WorldSet myWorld = new WorldSet(); // Empty World
    myWorld.addFormula("A");
    myWorld.addFormula("~B");

    DefaultRule rule1 = new DefaultRule();
    rule1.setPrerequisite("A");
    rule1.setJustificatoin("B");
    rule1.setConsequence("C");

    DefaultRule rule2 = new DefaultRule();
    rule2.setPrerequisite("A");
    rule2.setJustificatoin("~C");
    rule2.setConsequence("D");

    RuleSet myRules = new RuleSet();
    myRules.addRule(rule1);
    myRules.addRule(rule2);

    DefaultReasoner loader = new DefaultReasoner(myWorld, myRules);
    HashSet<String> extensions = loader.getPossibleScenarios();
    a.e.println("This example is titled Commitment to assumptions");
    a.e.println("Given the world: \n\t" + myWorld.toString()
            + "\n And the rules \n\t" + myRules.toString());

    a.e.println("Possible Extensions");
    for (String c : extensions) {
        a.e.println("\t Ext: Th(W U (" + c + "))");
        // Added closure operator
        a.e.incIndent();
        WFF world_and_ext = new WFF("(( " + myWorld.getWorld() + " ) & ("
                + c + "))");
        a.e.println("= " + world_and_ext.getClosure());
        a.e.decIndent();

    }
```

Résultat en TD :

D'où, par clôture déductive et minimalité, cette théorie admet une seule extension :
$E = \Gamma_{\Delta 2}(E) = TH(\{A, \neg B, D\})$.
Seul d2 est générateur d'extension.

- Avec le monde W3
  Avec ces défaut il y aura des extension possibles

```
------------------------------------------------------------------
Trying  A & (~C | ~D)
Trying  A & (~C | ~D)
Trying  A & (~C | ~D)
Trying  A & (~C | ~D)
Given the world:
        A & (~C | ~D)
 And the rules
        [(A):(B) ==> (C)] , [(A):(~C) ==> (D)]
Possible Extensions
         Ext: Th(W U (C))
        = C & ~D &  (~D | ~C) & A
------------------------------------------------------------------
```

```java
public static void EXO1W3() {
    WorldSet myWorld = new WorldSet(); // Empty World
    myWorld.addFormula("A");
    myWorld.addFormula("(~C | ~D)");

    DefaultRule rule1 = new DefaultRule();
    rule1.setPrerequisite("A");
    rule1.setJustificatoin("B");
    rule1.setConsequence("C");

    DefaultRule rule2 = new DefaultRule();
    rule2.setPrerequisite("A");
    rule2.setJustificatoin("~C");
    rule2.setConsequence("D");

    RuleSet myRules = new RuleSet();
    myRules.addRule(rule1);
    myRules.addRule(rule2);

    DefaultReasoner loader = new DefaultReasoner(myWorld, myRules);
    HashSet<String> extensions = loader.getPossibleScenarios();

    a.e.println("Given the world: \n\t" + myWorld.toString()
            + "\n And the rules \n\t" + myRules.toString());

    a.e.println("Possible Extensions");
    for (String c : extensions) {
        a.e.println("\t Ext: Th(W U (" + c + "))");
        // Added closure operator
        a.e.incIndent();
        WFF world_and_ext = new WFF("(( " + myWorld.getWorld() + " ) & ("
                + c + "))");
        a.e.println("= " + world_and_ext.getClosure());
        a.e.decIndent();

    }

}
```

Résultat en TD:

Conclusion : La théorie Δ_2 admet une et une seule extension :

$$\mathbf{E = \Gamma_{\Delta 3}(E) = TH(\{A, \neg C \vee \neg D, C\})}$$

- Exercice 2

  Il n'y a pas d'extension.

```
Given the world:
        A
 And the rules
        [(A):(~B) ==> (B)]
Possible Extensions
```

```java
public static void EX02() {
    WorldSet myWorld = new WorldSet(); // Empty World
    myWorld.addFormula("A");

    DefaultRule rule1 = new DefaultRule();
    rule1.setPrerequisite("A");
    rule1.setJustificatoin("~B");
    rule1.setConsequence("B");

    RuleSet myRules = new RuleSet();
    myRules.addRule(rule1);

    DefaultReasoner loader = new DefaultReasoner(myWorld, myRules);
    HashSet<String> extensions = loader.getPossibleScenarios();

    a.e.println("Given the world: \n\t" + myWorld.toString()
            + "\n And the rules \n\t" + myRules.toString());

    a.e.println("Possible Extensions");
    for (String c : extensions) {
        a.e.println("\t Ext: Th(W U (" + c + "))");
        // Added closure operator
        a.e.incIndent();
        WFF world_and_ext = new WFF("(( " + myWorld.getWorld() + " ) & ("
                + c + "))");
        a.e.println("= " + world_and_ext.getClosure());
        a.e.decIndent();

    }
}
```

- Exercice 3
    - Le monde W

```
Trying  A & B
Given the world:
        A & B
 And the rules
        [((A & B)):(~C) ==> (~C)]
Possible Extensions
         Ext: Th(W U (~C))
        = B & ~C &  A
```

```java
public static void EXO3W1() {
    WorldSet myWorld = new WorldSet(); // Empty World
    myWorld.addFormula("A");
    myWorld.addFormula("B");

    DefaultRule rule1 = new DefaultRule();
    rule1.setPrerequisite("(A & B)");
    rule1.setJustificatoin("~C");
    rule1.setConsequence("~C");

    RuleSet myRules = new RuleSet();
    myRules.addRule(rule1);

    DefaultReasoner loader = new DefaultReasoner(myWorld, myRules);
    HashSet<String> extensions = loader.getPossibleScenarios();

    a.e.println("Given the world: \n\t" + myWorld.toString()
            + "\n And the rules \n\t" + myRules.toString());

    a.e.println("Possible Extensions");
    for (String c : extensions) {
        a.e.println("\t Ext: Th(W U (" + c + "))");
        // Added closure operator
        a.e.incIndent();
        WFF world_and_ext = new WFF("(( " + myWorld.getWorld() + " ) & ("
                + c + "))");
        a.e.println("= " + world_and_ext.getClosure());
        a.e.decIndent();

    }
}
```

Résultat en TD :

- Donc, la seule extension de la théorie est $E=\Gamma_{\Delta}(E)=TH(\{A, B, \neg C\})$.

- Le monde W’

```
Given the world:
        A & B & C
 And the rules
        [((A & B)):(~C) ==> (~C)]
Possible Extensions
```

```java
public static void EXO3W2() {
    WorldSet myWorld = new WorldSet(); // Empty World
    myWorld.addFormula("A");
    myWorld.addFormula("B");
    myWorld.addFormula("C");

    DefaultRule rule1 = new DefaultRule();
    rule1.setPrerequisite("(A & B)");
    rule1.setJustificatoin("~C");
    rule1.setConsequence("~C");

    RuleSet myRules = new RuleSet();
    myRules.addRule(rule1);

    DefaultReasoner loader = new DefaultReasoner(myWorld, myRules);
    HashSet<String> extensions = loader.getPossibleScenarios();

    a.e.println("Given the world: \n\t" + myWorld.toString()
            + "\n And the rules \n\t" + myRules.toString());

    a.e.println("Possible Extensions");
    for (String c : extensions) {
        a.e.println("\t Ext: Th(W U (" + c + "))");
        // Added closure operator
        a.e.incIndent();
        WFF world_and_ext = new WFF("(( " + myWorld.getWorld() + " ) & ("
                + c + "))");
        a.e.println("= " + world_and_ext.getClosure());
        a.e.decIndent();

    }
}
```

Résultat en TD :

- Ainsi, par clôture déductive et minimalité, $\Gamma_{\Delta'}(E')=TH(\{A,B,C\})$
- Donc, la seule extension de la théorie est $E'=\Gamma_{\Delta'}(E')=TH(\{A, B, C\})$.

- Exercice 6

```
Trying  eeee
Trying  eeee
Trying  eeee
Trying  eeee
Trying  eeee
Trying  eeee
Given the world:

 And the rules
       [(a):(b) ==> (b)] , [([]):(a) ==> (a)] , [([]):(~a) ==> (~a)]
Possible Extensions
         Ext: Th(W U (~a))
        =  ~a
         Ext: Th(W U (b & a))
        = b &  a
```

```java
public static void EXO6() {
    WorldSet myWorld = new WorldSet(); // Empty World
    myWorld.addFormula(a.e.EMPTY_FORMULA);

    DefaultRule rule1 = new DefaultRule();
    rule1.setPrerequisite("a");
    rule1.setJustificatoin("b");
    rule1.setConsequence("b");

    DefaultRule rule2 = new DefaultRule();
    rule2.setPrerequisite(a.e.EMPTY_FORMULA);
    rule2.setJustificatoin("a");
    rule2.setConsequence("a");

    DefaultRule rule3 = new DefaultRule();
    rule3.setPrerequisite(a.e.EMPTY_FORMULA);
    rule3.setJustificatoin("~a");
    rule3.setConsequence("~a");

    RuleSet myRules = new RuleSet();
    myRules.addRule(rule1);
    myRules.addRule(rule2);
    myRules.addRule(rule3);

    DefaultReasoner loader = new DefaultReasoner(myWorld, myRules);
    HashSet<String> extensions = loader.getPossibleScenarios();

    a.e.println("Given the world: \n\t" + myWorld.toString()
            + "\n And the rules \n\t" + myRules.toString());

    a.e.println("Possible Extensions");
    for (String c : extensions) {
        a.e.println("\t Ext: Th(W U (" + c + "))");
        // Added closure operator
        a.e.incIndent();
        WFF world_and_ext = new WFF("(( " + myWorld.getWorld() + " ) & ("
                + c + "))");
        a.e.println("= " + world_and_ext.getClosure());
        a.e.decIndent();

    }
}
```

Résultat en TD :

Ainsi, cette théorie Δ admet deux extensions :

$E_1 = \Gamma_\Delta(E1) = TH(\{\neg a\})$: cas où d2 est déclenché en premier

$E_2 = \Gamma_\Delta(E2) = TH(\{a,b\})$: cas où d3 est déclenché en premier suivi de d1

TP4 :

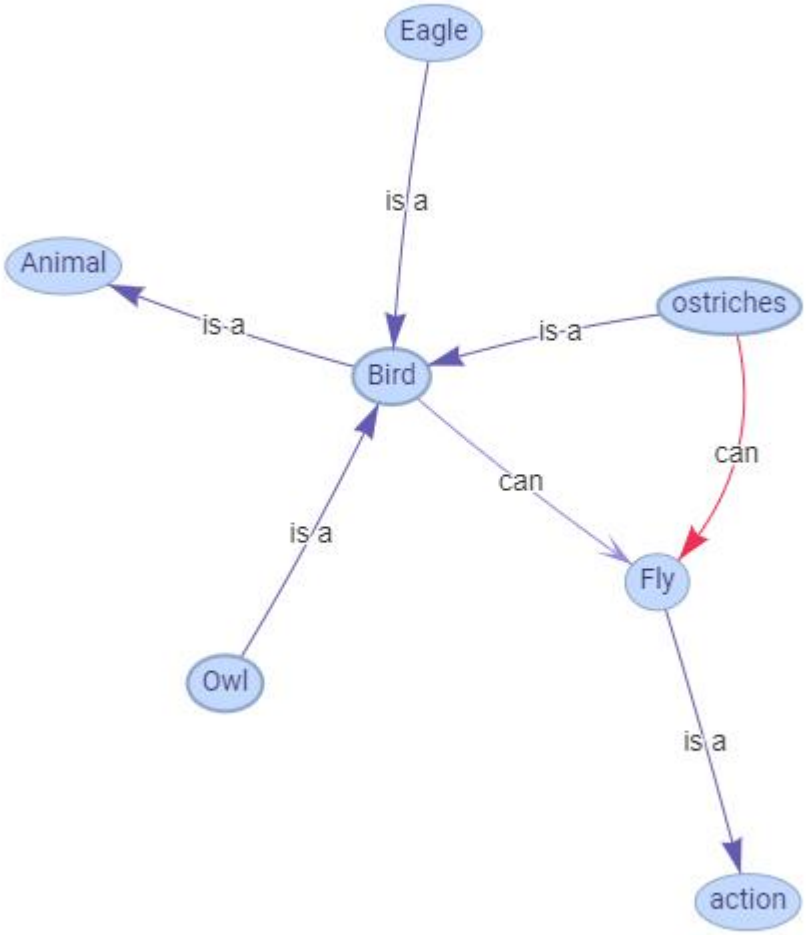
I. Introduction :

Dans ce TP, nous nous intéressons aux réseaux sémantiques.

En utilisant l'outil en ligne https://semantic-networks.netlify.app/, nous pourrons modéliser nos propres réseaux sémantiques. On va pouvoir dessiner le réseau nœud par nœud, et arc par arc. Il nous permet aussi d'utiliser l'algorithme de propagation des marqueurs et de l'héritage.

II. Le réseau sémantique utilisé :

Dans ce réseau nous représentons les règles reliées aux oiseaux.



- Propagation des marqueurs :
  L'algorithme nous permettra de déterminer quels nœuds ont une relation avec un nœud. Nous avons codé en python, cet algorithme qui va utiliser le fichier.json généré par le site:

network.json > ...


```
{
    "nodes":[
        {"id":"5d544bda-7d61-44dd-9609-ca47a5fb12d3","x":-245,"y":-138.5,"label":"Bird","color":null,"borderWidth":2,"M1":false,"M2":false},
        {"id":"2bc4d46e-c5eb-41ce-89b9-ef2ac8a5e78d","x":124,"y":-105.5,"label":"Fly"},
        {"id":"4f34c1a9-c93a-4f52-bc9f-6c7bcdf16b54","x":104,"y":-137.5,"label":"ostriches","color":null,"borderWidth":2,"M1":false,"M2":false},
        {"id":"b49b7f0b-4323-4ee5-92b6-9e426691170c","x":-271,"y":-86.5,"label":"Owl","color":null,"borderWidth":2,"M1":false,"M2":false},
        {"id":"b34850b0-27db-4f5f-9b39-a121e824b9dc","x":-104.308082532919,"y":-226.8035367778998,"label":"Eagle"},
        {"id":"5f13a1c1-52ad-4736-82f2-e5fcdc3c2488","x":-327.308082532919,"y":-144.8035367778998,"label":"Animal"},
        {"id":"e84c1495-6157-434c-8873-60eda57a98b3","x":80.691917467081,"y":202.1964632221002,"label":"action"}],

    "edges":[
        {"from":"5d544bda-7d61-44dd-9609-ca47a5fb12d3","to":"2bc4d46e-c5eb-41ce-89b9-ef2ac8a5e78d","label":"can","edge_type":"Most",
            "arrows":{"to":{"enabled":true,"type":"vee"}},"color":"#978dde","id":"ac85f547-1fec-45d9-a5e6-84a472be8ece"},
        {"from":"4f34c1a9-c93a-4f52-bc9f-6c7bcdf16b54","to":"5d544bda-7d61-44dd-9609-ca47a5fb12d3","label":"is a","edge_type":"All",
            "arrows":{"to":true},"id":"ef43864b-da8c-4a9e-ac60-b70c58abde3c","color":null,"width":null},
        {"from":"b49b7f0b-4323-4ee5-92b6-9e426691170c","to":"5d544bda-7d61-44dd-9609-ca47a5fb12d3","label":"is a","edge_type":"All",
            "arrows":{"to":true},"id":"5ee12038-85c2-4670-95eb-85e01a169633","color":null,"width":null},
        {"from":"4f34c1a9-c93a-4f52-bc9f-6c7bcdf16b54","to":"2bc4d46e-c5eb-41ce-89b9-ef2ac8a5e78d","label":"can","edge_type":"Not","arrows":{"to
        {"from":"b34850b0-27db-4f5f-9b39-a121e824b9dc","to":"5d544bda-7d61-44dd-9609-ca47a5fb12d3","label":"is a","edge_type":"All","arrows":{"t
        {"from":"5d544bda-7d61-44dd-9609-ca47a5fb12d3","to":"5f13a1c1-52ad-4736-82f2-e5fcdc3c2488","label":"is a","edge_type":"All","arrows":{"t
        {"from":"2bc4d46e-c5eb-41ce-89b9-ef2ac8a5e78d","to":"e84c1495-6157-434c-8873-60eda57a98b3","label":"is a","edge_type":"All","arrows":{"t
    }
```


```python
def get_label(sem_net, node, link):
    # we retrieve the info from the dict generator from the json file
    #separate the edges and nodes
    node_link_arcs=[]
    for edge in sem_net["edges"]:
        if (edge["to"] == node["id"] and edge["label"] == link):
            node_link_arcs.append(edge["from"])

    #we take their labels for an easy search
    node_link_arcs_label = []
    for node in sem_net["nodes"]:
        if node["id"] in node_link_arcs:
            node_link_arcs_label.append(node["label"])

    reponse = "there is a link between nodes : " + ", ".join(node_link_arcs_label)
    return reponse
```

```
def mark_propagation(sem_net, node1, node2, link):
    nodes = sem_net["nodes"]

    solutions_found = []

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

        for node in nodes:
            if node["label"] == node1:
                val1 = node
                break

        for node in nodes:
            if node["label"] == node2:
                val2 = node
                break

        edges = sem_net["edges"]
        mark_prop_arcs=[]
        for edge in edges:
            if (edge["to"] == val1["id"] and edge["label"] == "is a"):
                mark_prop_arcs.append(edge)

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

            temp_node = mark_prop_arcs.pop()
            temp_node_contains_edges=[]
            for edge in edges :
                if (edge["from"] == temp_node["from"] and edge["label"] == link):
                    temp_node_contains_edges.append(edge)

            solution_found = any(d['to'] == val2["id"] for d in temp_node_contains_edges)

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

                mark_prop_arcs.extend(is_a_temp)

        solutions_found.append(get_label(sem_net, val2, link) if solution_found else "no link found")

    return(solutions_found)
```

- Un test :

```
value = propagation.mark_propagation(dictio, 'Animal', 'Bird', 'is a')
print(value)
```

```
'there is a link between nodes : ostriches, Owl, Eagle'
```

- Héritage : pour déterminer tous les nœuds desquels le nœud donné a hérité :

La base de données utilisée est le même réseau sémantique, mais avec certaines exceptions :

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

```python
def heritage(sem_network, name):
    the_end = False

    nodes = sem_network["nodes"]
    edges = sem_network["edges"]


    for node_ in nodes:
        if node_["label"] == name:
            node=node_
            break

    inherited_arc=[]
    for edge in edges :
        if (edge["from"] == node["id"] and edge["label"] == "is a"):
            inherited_arc.append(edge["to"])
    legacy = []
    properties = []
    while not the_end:
        if(len(inherited_arc) == 0):
            legacy.append("no heritage")
            break
        n = inherited_arc.pop()

        legacy.append(get_label(sem_network, n))

        inherited_arc=[]
        for edge in edges:
            if (edge["from"] == n and edge["label"] == "is a"):
                inherited_arc.extend(edge["to"])

        properties_nodes=[]
        for edge in edges:
            if (edge["from"] == n and edge["label"] != "is a"):
                properties_nodes.append(edge)

        for pn in properties_nodes:
            properties.append(": ".join([pn["label"], get_label(sem_network, pn["to"])]))
        if len(inherited_arc) == 0:
            the_end = True

    return legacy, properties
```

- Un test :

```
value2 = heritage.heritage(dictio, 'Eagle')
print(value2)
```

```
(['Bird', ''], ['can: Fly'])
```

- Les exceptions : le fichier .json du réseau sémantique à été mis à jour pour faire l'exemple des exceptions :

```
{
    "nodes":[
        {"id":"5d544bda-7d61-44dd-9609-ca47a5fb12d3","x":-245,"y":-138.5,"label":"Bird","color":null,"borderWidth":2,"M1":false,"M2":false},
        {"id":"2bc4d46e-c5eb-41ce-89b9-ef2ac8a5e78d","x":124,"y":-105.5,"label":"Fly"},
        {"id":"4f34c1a9-c93a-4f52-bc9f-6c7bcdf16b54","x":104,"y":-137.5,"label":"ostriches","color":null,"borderWidth":2,"M1":false,"M2":false},
        {"id":"b49b7f0b-4323-4ee5-92b6-9e426691170c","x":-271,"y":-86.5,"label":"Owl","color":null,"borderWidth":2,"M1":false,"M2":false},
        {"id":"b34850b0-27db-4f5f-9b39-a121e824b9dc","x":-104.308082532919,"y":-226.8035367778998,"label":"Eagle"},
        {"id":"5f13a1c1-52ad-4736-82f2-e5fcdc3c2488","x":-327.308082532919,"y":-144.8035367778998,"label":"Animal"},
        {"id":"e84c1495-6157-434c-8873-60eda57a98b3","x":80.691917467081,"y":202.1964632221002,"label":"action"}],

    "edges":[
        {"from":"5d544bda-7d61-44dd-9609-ca47a5fb12d3","to":"2bc4d46e-c5eb-41ce-89b9-ef2ac8a5e78d","label":"can","edge_type":"Most",
            "arrows":{"to":{"enabled":true,"type":"vee"}},"color":"#978dde","id":"ac85f547-1fec-45d9-a5e6-84a472be8ece"},
        {"from":"4f34c1a9-c93a-4f52-bc9f-6c7bcdf16b54","to":"5d544bda-7d61-44dd-9609-ca47a5fb12d3","label":"is a","edge_type":"All",
            "arrows":{"to":true},"id":"ef43864b-da8c-4a9e-ac60-b70c58abde3c","color":null,"width":null},
        {"from":"b49b7f0b-4323-4ee5-92b6-9e426691170c","to":"5d544bda-7d61-44dd-9609-ca47a5fb12d3","label":"is a","edge_type":"All",
            "arrows":{"to":true},"id":"5ee12038-85c2-4670-95eb-85e01a169633","color":null,"width":null},
        {"from":"4f34c1a9-c93a-4f52-bc9f-6c7bcdf16b54","to":"2bc4d46e-c5eb-41ce-89b9-ef2ac8a5e78d","label":"can","edge_type":"Not","arrows":{"to":true},"color":"
        {"from":"b34850b0-27db-4f5f-9b39-a121e824b9dc","to":"5d544bda-7d61-44dd-9609-ca47a5fb12d3","label":"is a","edge_type":"All","arrows":{"to":true},"id":"7a
        {"from":"5d544bda-7d61-44dd-9609-ca47a5fb12d3","to":"5f13a1c1-52ad-4736-82f2-e5fcdc3c2488","label":"is a","edge_type":"exception","arrows":{"to":true},"id
        {"from":"2bc4d46e-c5eb-41ce-89b9-ef2ac8a5e78d","to":"e84c1495-6157-434c-8873-60eda57a98b3","label":"is a","edge_type":"exception","arrows":{"to":true},"id

}
```


```python
def mark_propagation_exception(sem_net, node1, node2, link):
    nodes = sem_net["nodes"]

    solutions_found = []

    for i in range(min(len(node1),len(node2))):
        solution_found = False
        for node in nodes:
            if node["label"] == node1:
                val1 = node
                break

        for node in nodes:
            if node["label"] == node2:
                val2 = node
                break

        edges = sem_net["edges"]
        mark_prop_arcs = []
        for edge in edges :
            if (edge["to"] == val1["id"] and edge["label"] == "is a" and (edge["edge_type"] != "exception" )) :
                mark_prop_arcs.append(edge)

        while len(mark_prop_arcs) != 0 and not solution_found:
            temp_node = mark_prop_arcs.pop()
            temp_node_contains_edges=[]
            for edge in edges :
                if (edge["from"] == temp_node["from"] and edge["label"] == link and (edge["edge_type"] != "exception" )):
                    temp_node_contains_edges.append(edge)

            solution_found = any(d['to'] == val2["id"] for d in temp_node_contains_edges)

            if not solution_found:
                is_a_temp=[]
                for edge in edges :
                    if (edge["to"] == temp_node["from"] and edge["label"] == "is a" and (edge["edge_type"] != "exception" )):
                        is_a_temp.append(edge)
                mark_prop_arcs.extend(is_a_temp)

        solutions_found.append(get_label(sem_net, val2, link) if solution_found else "no link found")


    return(solutions_found)
```

- Un test :

```python
exception = json.load(f1)
value3 = exceptions.mark_propagation_exception(exception, 'Animal', 'Bird', 'is a')
print(value3)
```

```
'no link found'
```

TP5 :

I. Introduction

Dans ce TP, nous allons utiliser l'outil Protégé et le raisonneur Pellet. Protégé nous aidera à créer notre base de connaissances et Pellet nous aidera à déduire de nouvelles informations à partir de celles qu'on a créé.

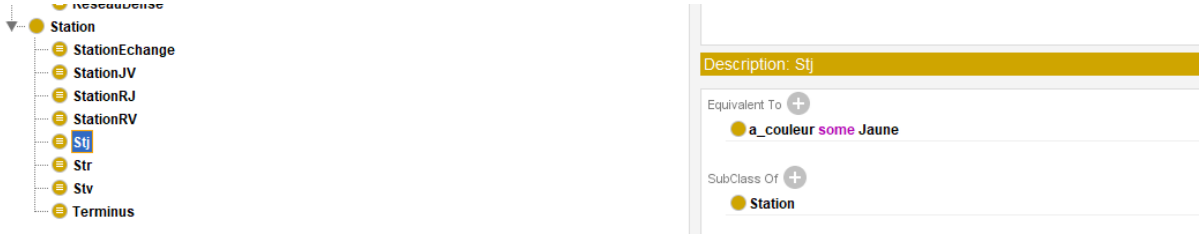
II. Base de connaissances

Tout d'abord, nous avons créé notre base de connaissances en se basant sur l'exercice 2 de la série 6 (la carte de métro de Milan).

Nous avons 3 Couleurs, 2 types de réseaux, et plusieurs types de stations.

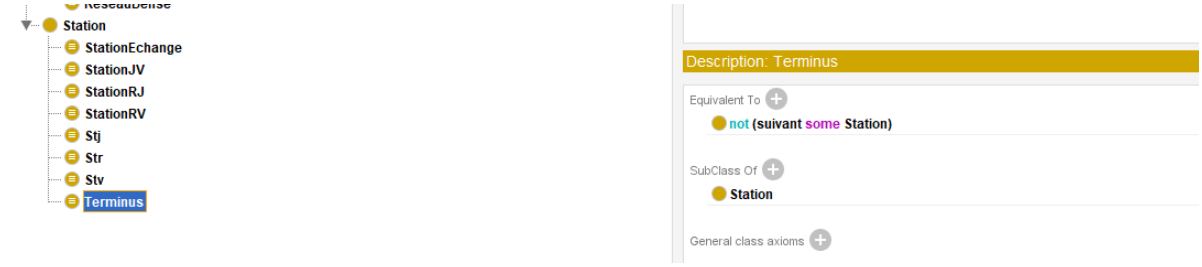
Par exemple, un petit réseau aura un maximum de 6 lignes.

Un réseau dense est composé d'au minimum 20 lignes

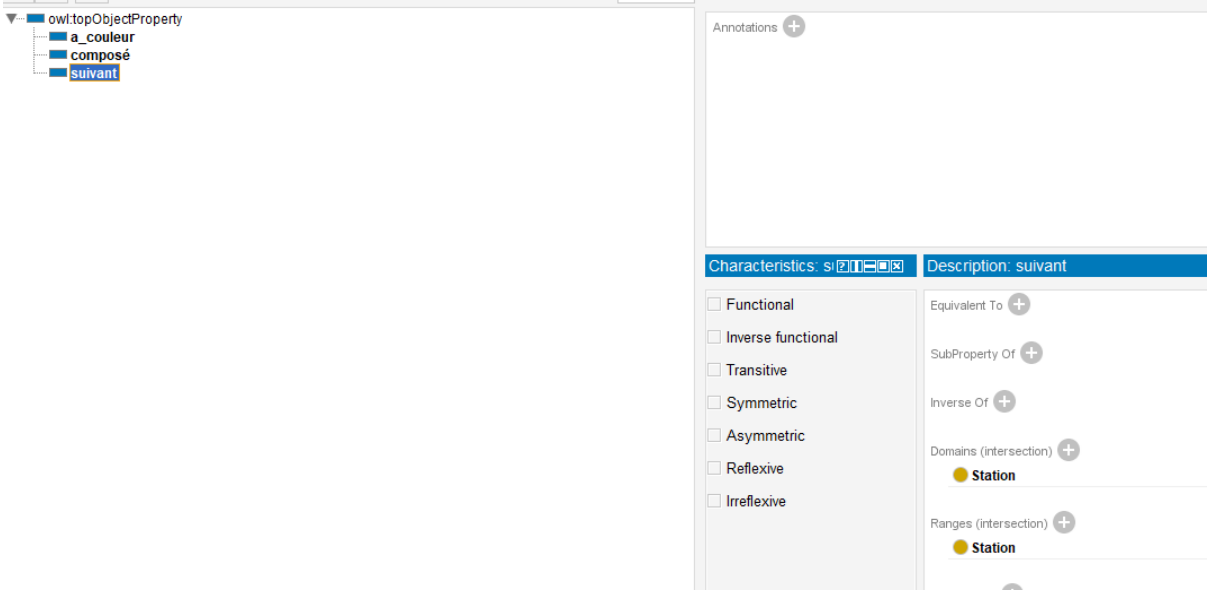


Une station Jaune est une station qui a la couleur Jaune.

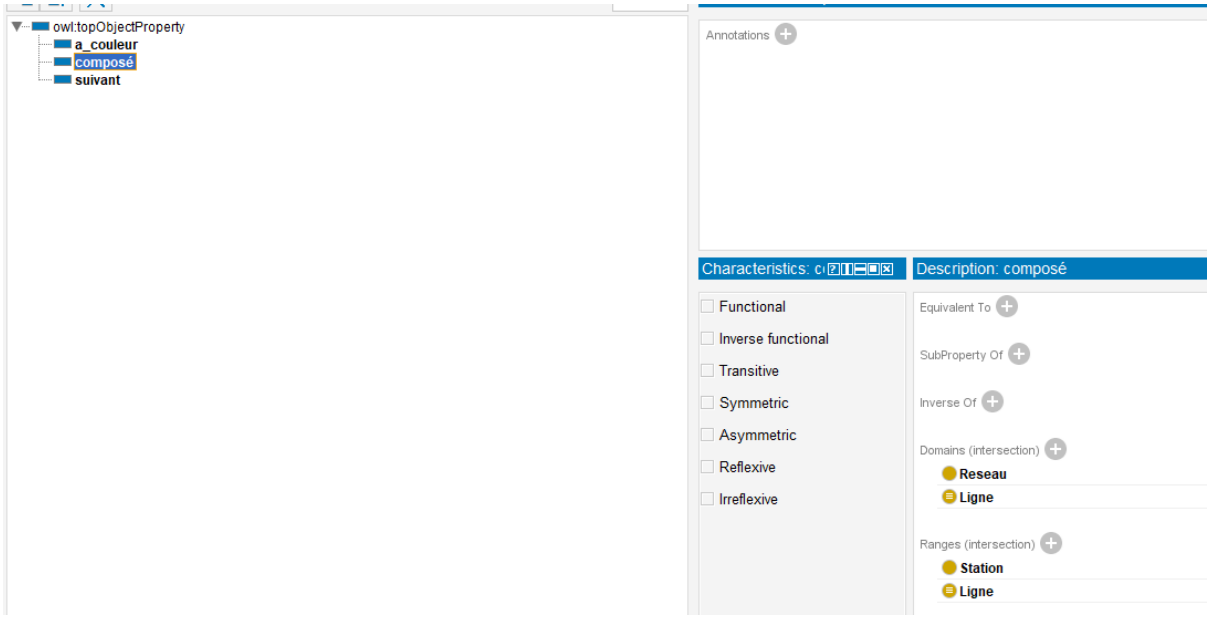
Une station rouge-jaune est une station qui est soit rouge soit jaune



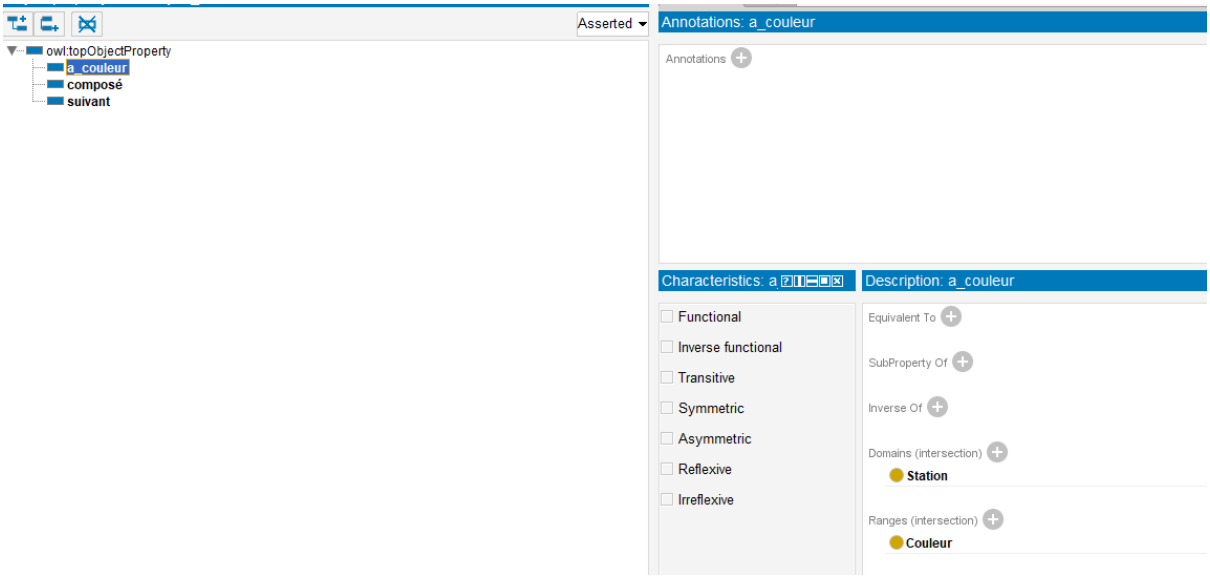
Un terminus est une station qui n'a pas de station suivante.
En ce qui concerne les rôles:



Le rôle suivant permet d'exprimer qu'une Station a une Station suivante.



Le rôle composé permet d'exprimer qu'un réseau est composé de Lignes ou une ligne est composée de Stations.

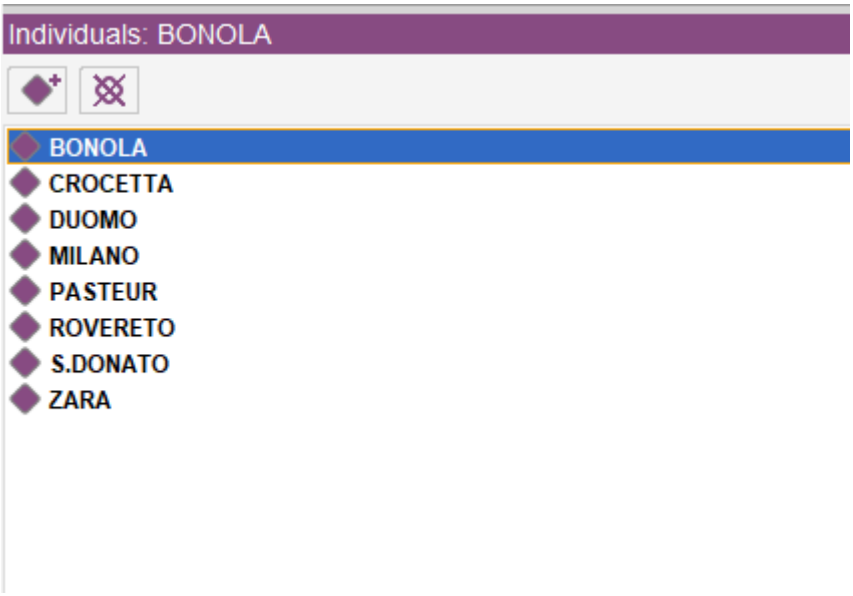


Le rôle a_couleur permet d'exprimer qu'une Station a une Couleur.

III. Base de faits

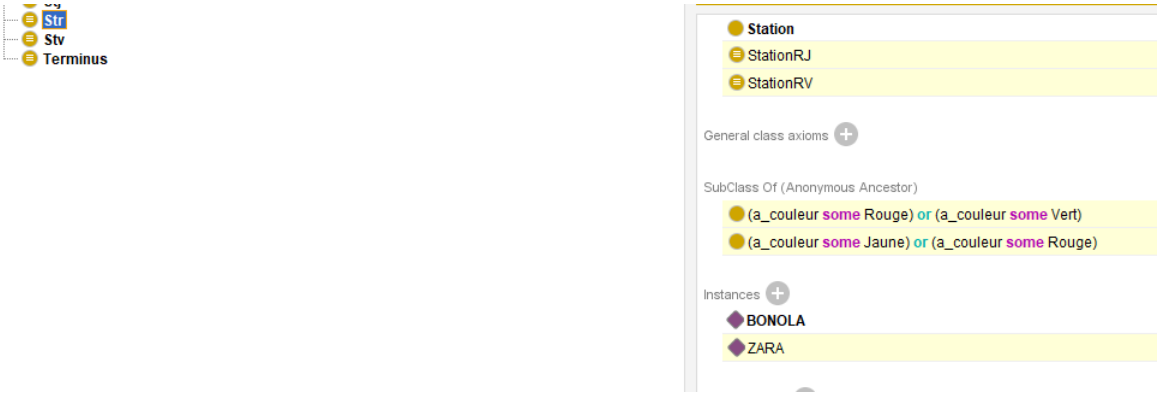
Pour notre base de faits, nous avons reconstitué les faits donnés de la A-BOX vue en td:

ABOX :
STR(BONOLA) ; STJ(CROCETTA) ; STJR(DUOMO) ; TERMINUS(S.DONATO)
a-couleur(ZARA,ROUGE) ; suivant(PASTEUR,ROVERETO) ; RESEAU(MILANO)

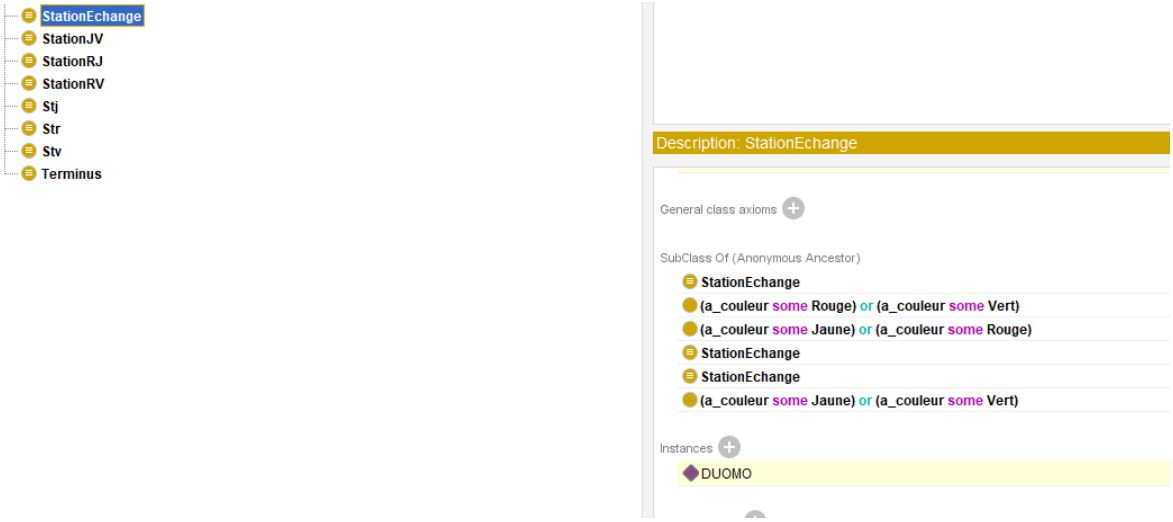


IV. Raisonnement

En utilisant le raisonneur Pellet, nous avons pu déterminer quelques nouvelles informations.



La station ZARA a été reconnue comme Station Rouge.



La station DUOMO est une station d'échange.