

Université des Sciences et de la Technologie Houari Boumediene



Rapport des TPs RCR

Binôme

OMARI Hamza	181831030062	Groupe 01
HAMZAOUI Thameur	181831030061	Groupe 02

2023-2024

Table des matières

Table des matières	2
TP1 Partie 1 : Inférence Logique propositionnelle basée sur un solveur SAT	4
Introduction	4
Etape 1	4
Etape 2	4
TEST1.cnf	6
Etape 3	7
Traduction de la base de connaissances	7
codification:	7
Création du fichier cnf:	8
Test des fichiers benchmarks:	9
Etape 4	10
coude source	11
TP1 Partie 2 : Inférence logique des prédicats	13
setup de tweety	13
Code source	14
Résultat d'exécution	14
TP2 : logique modale	15
Exemple de modèle modal	15
code source	16
Résultat d'exécution	16
TP3 : Logique des défauts	17
Setup	17
Code source	18
Résultats	19
TP4 : Les logiques de description	20
Introduction	20
TBOX	20
ABOX	20
Code source	21
Résultats	22
TP5 : Théorie des Fonctions de Croyance	23
Introduction:	23
Exemple	23
Modélisation de connaissance	23
Code Source:	24
Représentation des fonctions de masse pour chaque expert :	24
Calcul de croyance et Plausibilité pour chaque expert	25
les de degré de croyance, plausibilité et degré de doute pour toutes les hypothèses	26
Combinaison de Dempster & Shaffer	27
TP6 : Réseaux Bayésiens	28
Introduction	28
	2

ÉTAPE 1 : Installation de PGMPY	28
ÉTAPE 2 : Poly Arbre	29
Matrices de probabilités :	29
Probabilités conditionnelles :	30
- $P(D=\text{Detection Likelihood} \mid O)$:	30
Local Indépendances :	30
Global Indépendances :	30
Code Source	31
résulte	32
Code	32
Code	33
Résultat	33
Code	33
résultat	34
code and résulte	34
Partie 2 : Conception d'un arbre multi-connected	35
Define the structure of the Bayesian Network for computer vision	35
Création de Graphe	36
TP7 : Contrôleurs flous	39
introduction	39
Exemple	39
Etape 1 : Définition des E/S du contrôleur	40
Etape 2 subdivision des E/S en sous ensembles flou (Fuzzification)	41
Etape 3 Définition de la base de règle floue	44
Etape 4 Application de la méthode d'inférence	45
Etape 5 Application de la défuzzification	46

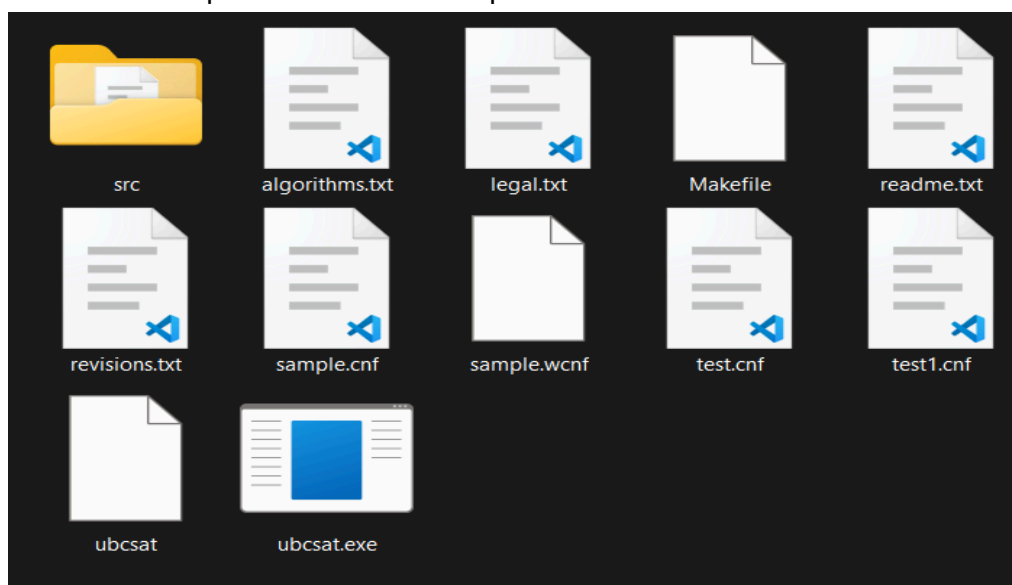
TP1 Partie 1 : Inférence Logique propositionnelle basée sur un solveur SAT

Introduction

Au cours de cette première séance pratique, nous allons commencer par appliquer un solveur SAT afin d'analyser la satisfaisabilité de différentes Bases de Connaissances. Ensuite, nous allons traduire une base de connaissances concernant la zoologie, effectuer des tests sur des références standards, et simuler l'inférence d'une base de connaissances en utilisant un algorithme.

Etape 1

création d'un répertoire **UBCSAT** et copie des fichiers



Etape 2

Maintenant nous allons exécuter le solveur SAT, pour tester la satisfaisabilité des deux fichiers : test.cnf et test1.cnf

en utilisant la commande

```
UBCSAT> ubcsat -alg saps -i test.cnf -solve
```

cette première partie affiche des informations sur UBCSAT, un site et l'instruction d'aide, ainsi que plusieurs paramètres de notre Solveur.

```

C:\Users\monms\Desktop\RCR TP\UBCSAT>ubcsat -alg saps -i test.cnf -solve
#
# UBCSAT version 1.1.0 (Sea to Sky Release)
#
# http://www.satlib.org/ubcsat
#
# ubcsat -h for help
#
# -alg saps
# -runs 1
# -cutoff 100000
# -timeout 0
# -gtimeout 0
# -noimprove 0
# -target 0
# -wtargt 0
# -seed 960124498
# -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

```

la deuxième partie d'affichage

```

# 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      4      4
#

```

cette partie représente le rapport des résultats de traitement

```

# Solution found for -target 0

1 2 -3 -4 5

```

cette partie représente les résultats, ici on peut voir qu'une solution a été trouvée

```

Variables = 5
Clauses = 11
TotalLiterals = 27
TotalCPUTimeElapsed = 0.000
FlipsPerSecond = 1
RunsExecuted = 1
SuccessfulRuns = 1
PercentSuccess = 100.00
Steps_Mean = 4
Steps_CoeffVariance = 0
Steps_Median = 4
CPUTime_Mean = 0
CPUTime_CoeffVariance = 0
CPUTime_Median = 0

```

Cette dernière partie, représente un rapport de statistiques sur l'exécution du SOLVER sur le problème.

TEST1.cnf

voici le résultat de l'exécution sur l'ensemble test1

```

1 0 1 4 100000
# No Solution found for -target 0

Variables = 5
Clauses = 13
TotalLiterals = 29
TotalCPUTimeElapsed = 0.000
FlipsPerSecond = 1
RunsExecuted = 1
SuccessfulRuns = 0
PercentSuccess = 0.00
Steps_Mean = 100000
Steps_CoeffVariance = 0
Steps_Median = 100000
CPUTime_Mean = 0
CPUTime_CoeffVariance = 0
CPUTime_Median = 0

```

Nous remarquons que pour ce fichier cnf, l'exécution affiche "no solution found for -target 0" cela signifie que l'ensemble des formules en entrées n'est pas satisfiable. On remarque que le pourcentage de réussite est bien égal à 0%.

Etape 3

Traduction de la base de connaissances

Ci-dessous, quelques énoncés :

- Les Loutre de mer sont des mammifères-marins
- Les mammifères-marins sont des animaux
- Les animaux ont généralement une pelage
- Les mammifères-marins n'en ont généralement pas
- Les Loutre de mer en ont un

- a est un Loutre de mer ,
- b est un mammifère marin,
- c est un animal.

En ignorant pour l'instant les connaissances utilisant le mot "généralement" nous avons :

- $(La \supset Ma); (Lb \supset Mb); (Lc \supset Mc);$
- $(Ma \supset Aa); (Mb \supset Ab); (Mc \supset Ac);$
- $(Aa \supset Pa); (Ab \supset Pb); (Ac \supset Pc);$
- $(Ma \supset \neg Pa); (Mb \supset \neg Pb); (Mc \supset \neg Pc)$
- $(La \supset Pa); (Lb \supset Pb); (Lc \supset Pc);$

- La; Mb; Ac.

Sachant que $A \supset B$ sous la forme CNF c'est $(\neg A \vee B)$ On a La liste des CNF :

- $(\neg louta \vee mamia); (\neg loutb \vee mamib); (\neg loutc \vee mamic)$
- $(\neg mamia \vee anima); (\neg mamib \vee animb); (\neg mamib \vee animb);$
- $(\neg anima \vee a-pela); (\neg animb \vee a-pelb); (\neg anima \vee a-pelc);$
- $(\neg louta \vee a-pela); (\neg loutb \vee a-pelb); (\neg loutc \vee a-pelc);$
- $(\neg mamia \vee \neg a-pela); (\neg mamib \vee \neg a-pelb); (\neg mamib \vee \neg a-pelc);$
- louta
- mamib
- animc

codification:

louta = 1,
loutb = 2,
loutc = 3 ,
mamia = 4 ,
mamib = 5,
mamic = 6,
a-pelaa= 7,
a-pelab = 8,
a-pelac = 9,
anima = 10,

animb = 11,
animc = 12

Création du fichier cnf:

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

Execution

UBCSAT> ubcsat -alg saps -i zoo.cnf -solve


```

# 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      2      5      100000
# No Solution found for -target 0

Variables = 12
Clauses = 18
TotalLiterals = 33
TotalCPUTimeElapsed = 0.005
FlipsPerSecond = 19999542
RunsExecuted = 1
SuccessfulRuns = 0
PercentSuccess = 0.00
Steps_Mean = 100000
Steps_CoeffVariance = 0
Steps_Median = 100000
CPUTime_Mean = 0.00500011444092
CPUTime_CoeffVariance = 0
CPUTime_Median = 0.00500011444092

```

Test des fichiers benchmarks:

On a choisi 2 fichiers, un contenant 20 variables et 91 clauses et l'autre contient 50 variables et 218 clauses Test du fichier satisfiable « uf75-01 »

```

# 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

```

On remarque qu'il existe bien une solution satisfaisant la base de clauses.

suite de l'affichage :

```
Variables = 75
Clauses = 325
TotalLiterals = 975
TotalCPUtimeElapsed = 0.000
FlipsPerSecond = 1
RunsExecuted = 1
SuccessfulRuns = 1
PercentSuccess = 100.00
Steps_Mean = 135
Steps_CoeffVariance = 0
Steps_Median = 135
CPUtime_Mean = 0
CPUtime_CoeffVariance = 0
CPUtime_Median = 0
```

Test du fichier satisfiable "uuf75-01"

```
1 0 1 588 100000
# No Solution found for -target 0

Variables = 75
Clauses = 325
TotalLiterals = 975
TotalCPUtimeElapsed = 0.031
FlipsPerSecond = 3225817
RunsExecuted = 1
SuccessfulRuns = 0
PercentSuccess = 0.00
Steps_Mean = 100000
Steps_CoeffVariance = 0
Steps_Median = 100000
CPUtime_Mean = 0.0309998989105
CPUtime_CoeffVariance = 0
CPUtime_Median = 0.0309998989105
```

pas de solution satisfaisant la base des clauses.

Etape 4

En utilisant 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.

avec l' **Algorithme de raisonnement par l'absurde**

coude source

```
C: > Users > monms > Desktop > RCR TP > UBCSAT > C code.c > main()
1  ∨ #include <stdio.h>
2  ∨ #include <stdlib.h>
3  ∨ #include <string.h>
4  ∨ int main() {
5      FILE *fich_base = NULL;
6      FILE *fich_temp = NULL;
7
8      int nbr_propositions, nbr_variables, nbr_clauses, i, but[20], non_but[20];
9      char nom_base[20], c;
10
11     printf("Enter CNF file name : ");
12     gets(nom_base);
13     strcat(nom_base, ".cnf");
14     fich_base = fopen(nom_base, "r+");
15
16     if (fich_base == NULL) {
17         printf("Cannot open the file :(");
18     } else {
19         fich_temp = fopen("zoo1.cnf", "rw+");
20         if (fich_temp == NULL) {
21             printf("Cannot transfer the file's content :/");
22         } else {
23             fscanf(fich_base, "p cnf %d %d", &nbr_variables, &nbr_clauses);
24             nbr_clauses += 1;
25             fprintf(fich_temp, "p cnf %d %d\n", nbr_variables, nbr_clauses);
26         }
27         c = fgetc(fich_base);
28         while (c != EOF) {
29             c = fgetc(fich_base);
30             if (c != EOF) {
```

```

C: > Users > monms > Desktop > RCR TP > UBCSAT > code.c > main()
30     if (c != EOF) {
31         fputc(c, fich_temp);
32     }
33 }
34
35 printf("BC variables:\n");
36 printf("1: Louta ;\t\t2: Loutb;\t\t3: Loutc;\n4: mamia;\t5: mamib;\t6: mamic;\n7: pela;\t8: pelb;\t9: pelc;\n10: anima ;\t11: animb;\t12: animc;\n");
37 printf("Enter number of literals : ");
38 scanf("%d", &nbr_propositions);
39
40 for (i = 1; i < nbr_propositions + 1; i++) {
41
42     printf("\nEnter literal number %d : ", i);
43     scanf("%d", &but[i]);
44
45     if (but[i] > -13 && but[i] < 13) {
46         non_but[i] = but[i] * (-1);
47     } else {
48         puts("ERROR, invalid code");
49     }
50 }
51
52 fprintf(fich_temp, "\n");
53
54 for (i = 1; i < nbr_propositions + 1; i++) {
55     fprintf(fich_temp, "%d ", non_but[i]);
56 }
57
58 fprintf(fich_temp, "0");
59 system("ubcsat -alg saps -i zoo1.cnf -solve > results.txt");
60 fclose(fich_temp);

```

```

C: > Users > monms > Desktop > RCR TP > UBCSAT > code.c > main()
60     system("ubcsat -alg saps -i zoo1.cnf -solve > results.txt");
61     fclose(fich_temp);
62     int termine = 0;
63
64     FILE *fich = fopen("results.txt", "r+");
65
66     if (fich == NULL) {
67         printf("Cannot access results :(\n");
68     } else {
69         char texte[1000];
70         while (fgets(texte, 1000, fich) && !termine) {
71             if (strstr(texte, "# Solution found for -target 0")) {
72
73                 printf("\nBC U {Non GOAL} is satisfied. BC infer the GOAL.\nSolution is:\n");
74                 fscanf(fich, "\n");
75                 while (!strstr(fgets(texte, 1000, fich), "Variables")) {
76                     printf("%s", texte);
77                 }
78                 termine = 1;
79             }
80         }
81         if (termine == 0) {
82             printf("\nBC U {Non GOAL} is not satisfied. BC does not infer the GOAL.\n");
83             int j;
84             for (j = 1; j < nbr_propositions + 1; j++) {
85                 printf("%d ", (-1) * but[j]);
86             }
87             if (j > 2) {
88                 printf("Cannot be achieved\n");
89             } else {
90                 printf("Cannot be achieved\n");

```

```

C: > Users > monms > Desktop > RCR TP > UBCSAT > code.c > main()
90         printf("Cannot be achieved\n");
91     }
92 }
93 }
94 fclose(fich_temp);
95 }
96
97 return 0;
98 }

```

Résultat de l'exécution de l'algorithme sur la base de connaissances zoo :

```

C:\Users\monms\Desktop\RCR TP\UBCSAT>code
Enter CNF file name : zoo
BC variables:
1: Louta ;          2: Loutb;          3: Loutc;
4: mamia;          5: mamib;          6: mamib;
7: pela;           8: pelb; 9: pelc;
10: anima ;        11: animb;         12: animc;

Enter number of literals : 1

Enter literal number 1 : 1

BC U {Non GOAL} is not satisfied. BC does not infer the GOAL.
-1 Cannot be achieved

```

TP1 Partie 2 : Inférence logique des prédicats

Il s'agit d'exploiter la librairie Tweety pour la modélisation des connaissances en logique modale.

setup de tweety

Pour installer la bibliothèque Tweety Projet avec Maven, on doit ajouter la dépendance correspondante à notre fichier pom.xml

de plus , généralement eclipse intègre maven de base.

```

1<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLoc
2  <modelVersion>4.0.0</modelVersion>
3  <groupId>mytweety.mytweetyapp</groupId>
4  <artifactId>mytweetyapp</artifactId>
5  <version>0.0.1-SNAPSHOT</version>
6  <dependencies>
7  <dependency>
8    <groupId>org.tweetyproject</groupId>
9    <artifactId>tweety-full</artifactId>
10   <version>1.24</version>
11 </dependency>
12 </dependencies>
13 </project>

```

Code source

```
public static void main(String[] args) throws IOException{

    System.out.println("STAAAAART : \n");

    FolSignature signature = new FolSignature();
    FolBeliefSet BC = new FolBeliefSet();
    FolParser parser = new FolParser();

    Predicate terminal = new Predicate("terminal", 1);
    signature.add(terminal);
    Predicate passe = new Predicate ("passe", 2);
    signature.add(passe);
    Constant master2 = new Constant ("master2");
    signature.add(master2);
    Constant thameur = new Constant("thameur");
    signature.add(thameur);
    BC.setSignature(signature);
    parser.setSignature(signature);

    FolFormula f1 = parser.parseFormula("forall X:(terminal(X) => passe(X,master2))");
    BC.add(f1);
    FolFormula f2 = parser.parseFormula("terminal(thameur)");
    BC.add(f2);
    FolReasoner.setDefaultReasoner(new SimpleFolReasoner());
    FolReasoner prover = FolReasoner.getDefaultReasoner();
    System.out.println("master2(thameur) is : "+prover.query(BC,(FolFormula) parser.parseFormula("terminal(thameur)")));
    System.out.println("passe(thameur,master2) is : "+prover.query(BC,(FolFormula) parser.parseFormula("passe(thameur,master2)"));
```

Résultat d'exécution

```
master2(thameur) is : true
passe(thameur,master2) is : true
```

On voit bien qu'il a évalué à 'true' les deux propositions, et effectivement elles sont vraies !

TP2 : logique modale

Il s'agit d'exploiter la librairie Tweety pour la modélisation des connaissances en logique modale.

Exemple de modèle modal

- 1- $M, w1 \models \Diamond(p \wedge q)$

Dans le monde $w1$, il existe une possibilité future où il fait beau et nous allons à la plage.

- 2- $M, w2 \models \neg \Box p$

Dans le monde $w2$, il n'est pas nécessairement vrai que partout il fait beau.

- 3- $M, w3 \models \Box(p \supset q)$

Dans le monde $w3$, il est nécessairement vrai que si il fait beau, alors nous allons à la plage.

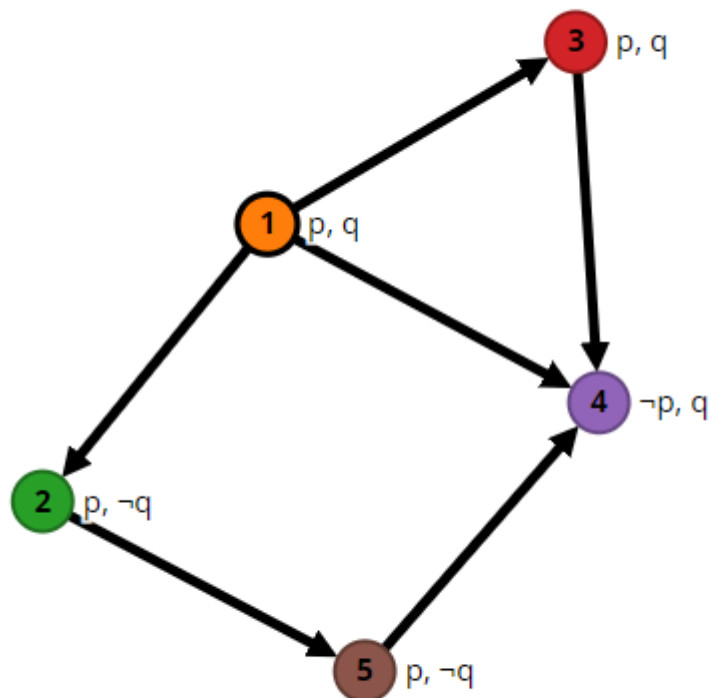
- 4- $M, w4 \models \Box(q \wedge \Diamond \neg p)$

Dans le monde $w4$, il est nécessairement vrai que nous allons à la plage et qu'il existe une possibilité future où il ne fait pas beau.

- 5- $M, w5 \models \Box(q \wedge \Diamond \neg p)$.

C'est une répétition de la formule précédente, mais appliquée au monde $w5$.

nous allons utiliser les formules déduites vraies et étudier la véracité des autres formules avec des raisonneurs. Nous avons donc : $\text{BliefSet} = \{ \text{Formules } 1 \ 3 \ 4 \}$



code source

```
// TP 2 : Logique modale
static void modal(String[] args) throws ParseException, IOException {
    MIBeliefSet bs = new MIBeliefSet();
    MIParser parser = new MIParser();
    FolSignature sig = new FolSignature();
    sig.add(new Predicate("p", 0));
    sig.add(new Predicate("q", 0));

    parser.setSignature(sig);
    bs.add((RelationalFormula) parser.parseFormula("<>(p && q)"));
    bs.add((RelationalFormula) parser.parseFormula("[](!p || q)"));
    bs.add((RelationalFormula) parser.parseFormula("[](q && <>(!q))"));

    System.out.println("Modal knowledge base: " + bs);
    SimpleMlReasoner reasoner = new SimpleMlReasoner();
    System.out.println("[](!p)      " + reasoner.query(bs, (FolFormula) parser.parseFormula("[](!p)"))+"\n");
    System.out.println("<>(p && q)      " + reasoner.query(bs, (FolFormula) parser.parseFormula("<>(p && q)"))+"\n");
}
```

Résultat d'exécution

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

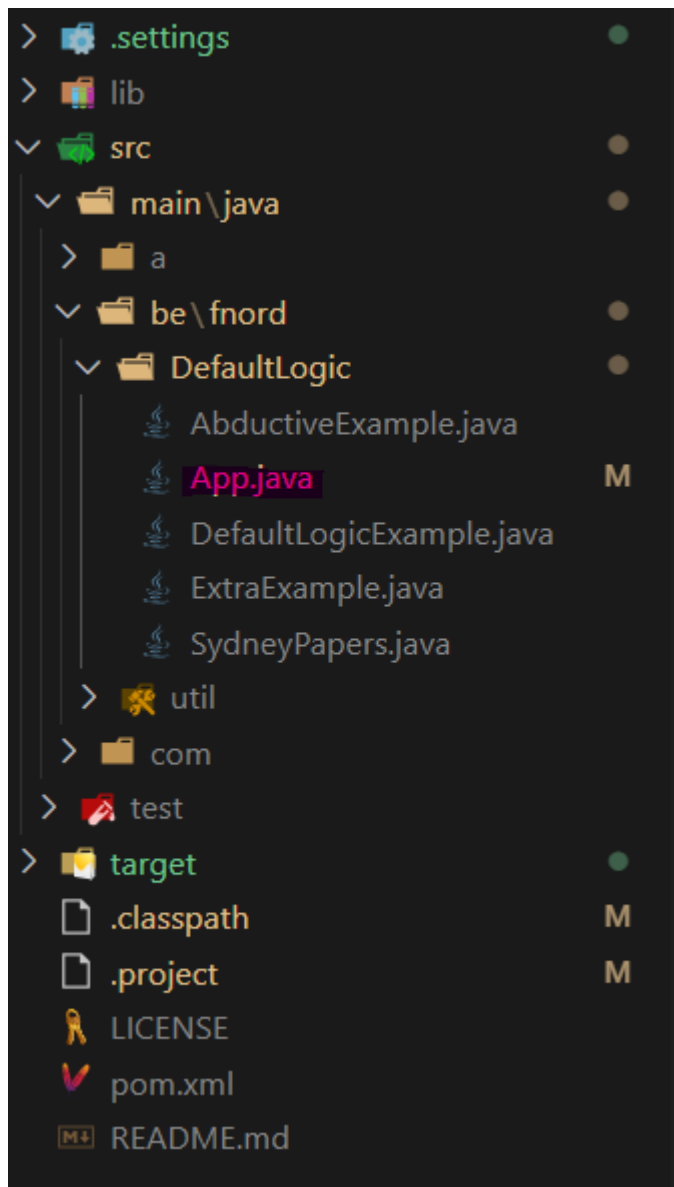

TP3 : Logique des défauts

Il s'agit d'exploiter une des boîtes à outils relatives au raisonnement basé sur la logique des défauts via différents liens

Dans ce TP, nous allons implémenter quelques exercices de la série de TD en utilisant la toolbox **"defaultlogic"** conçue en java par Evan Morrison.

Setup

Tout d'abord pour faire fonctionner la bibliothèque, il faut respecter la structure suivante



dans App.java, c'est le point d'entrée ou on va mettre du code.

Maintenant considérant les connaissances suivantes:

A : "Il y a des nuages dans le ciel"

B : "Les gens sortent avec des parapluies"

C: "pas d'alerte de pluies"

Nous avons donc : $W=\{A, B\}$ et $D=\{A \wedge B : \neg C/\neg C\}$.
 $W'=\{A, B, C\}$ et $D=\{A \wedge B : \neg C/\neg C\}$.

Code source

```
159 public static void world(){
160
161     RuleSet rules = new RuleSet () ; // pour mettre les d f a u t s
162     DefaultRule d = new DefaultRule () ; // c r a t i o n d'un d f a u t d1
163     d.setPrerequisite ("A"+e . AND +"B");
164     d.setJustificatoin ( e. NOT +"C");
165     d.setConsequence (e. NOT +"C") ;
166     rules.addRule (d) ;
167
168     WorldSet w1 = new WorldSet () ;
169     w1.addFormula (_wff:"A");
170     w1.addFormula (_wff:"B");
171
172     WorldSet w2 = new WorldSet () ;
173     w2.addFormula (_wff:"A");
174     w2.addFormula (_wff:"B");
175     w2.addFormula (_wff:"C");
176
177     /* ***** execution world 1***** */
178     try {
179         a. e. println (msg:" /===== execution World 1 =====/\n\n\n") ;
180         DefaultReasoner r = new DefaultReasoner (w1 , rules ) ; // c r a t i o n du raisonneur
181
182
183         HashSet < String > scenarios = r. getPossibleScenarios () ; // faire l'extension
184         a. e. println ("W1 : \n\t { " + w1 . toString () + " }\n D: \n\t { " + rules . toString () + " }");
185         a. e. println (msg:" Par c l t u r e d d u c t i v e e t m i n i m a l i t , c e t t e t h o r i e a d m e t u n e s e u l e e x t e n s i o n ");
186
187         for ( String c : scenarios ) {
188             a. e. println ("\t E: Th(W U ( " + c + " ) )");
189
190             // Added closure operator
191             a. e. incIndent () ;
192             WFF world_and_ext = new WFF ("(( " + w1 . getWorld () + " ) & ( " + c + " ) )");
193             a. e. println ("= " + world_and_ext . getClosure () );
194             a. e. decIndent () ;
195         }
196         a. e. println (msg:"");
197     } catch ( Exception e ){
198
199     }
200
201     /* ***** execution world 2 ***** */
202     try {
203         a. e. println (msg:" /===== execution World 2 =====/\n\n\n") ;
204         DefaultReasoner r = new DefaultReasoner (w2 , rules ) ; // c r a t i o n du raisonneur
205
206
207         HashSet < String > scenarios = r. getPossibleScenarios () ; // faire l'extension
208         a. e. println ("W1 : \n\t { " + w2 . toString ()
209             + " }\n D: \n\t { " + rules . toString () + " }");
210         a. e. println (msg:" Par c l o t u r e d d u c t i v e e t m i n i m a l i t , c e t t e t h o r i e a d m e t u n e s e u l e e x t e n s i o n ");
211         for ( String c : scenarios ) {
212             a. e. println ("\t E: Th(W U ( " + c + " ) )");
213
214             // Added closure operator
215             a. e. incIndent () ;
216             WFF world_and_ext = new WFF ("(( " + w2 . getWorld () + " ) & ( "
217             + c + " ) )");
218             a. e. println ("= " + world_and_ext . getClosure () );
219             a. e. decIndent () ;
220         }
221         a. e. println (msg:"");
```

Résultats

Pour world 1

```
/===== execution World 1 =====/
```

```
Trying eeee & A & B
```

```
W1 :
```

```
    { A & B }
```

```
D:
```

```
    { [(A&B): (~C) ==> (~C)] }
```

```
Par clture dductive et minimalit, cette thorie admet une seule extension
```

```
E: Th(W U (~C))
```

```
= B & ~C & eeee & A
```

Pour world 2

```
/===== execution World 2 =====/
```

```
W1 :
```

```
    { A & B & C }
```

```
D:
```

```
    { [(A&B): (~C) ==> (~C)] }
```

```
Par clture dductive et tminimalit, cette thorie admet une seule extension
```

TP4 : Les logiques de description

Introduction

La partie pratique consiste à utiliser un des outils pour simuler le raisonnement en exploitant les TBOX et les Abox des exercices précédents.

Nous allons utiliser la librairie python **OwlReady**

Comme fait, en série d'exercices et en cours. Nous allons d'abord présenter **TBOX** et **ABOX**

TBOX

Concepts

- Personne
- Aliment
- University

Rôles

- Mange
- Mange_par
- Enseigne
- Enseigne par
- PartieDe

Entités composées

- Faculty : Sous ensemble de University
- Departement : Sous ensemble de Faculty
- Etudiant :Sous ensemble de Personne
- Enseignant : Sous ensemble de Personne
- Malbouffe : Sous ensemble de Aliment

ABOX

- Personne(Khellaf)
- Pesonne(Mohamed)

Code source

```
from owlready2 import *

onto = get_ontology("http://testxyz.org/onto.owl") #create ontology using iri

with onto: #defining our ontology

    ##Defining concepts##
    class Personne (Thing) :
        pass

    class Aliment (Thing):
        pass

    class University (Thing):
        pass

    AllDisjoint ([Personne, Aliment, University]) #pour dire qu'un individu ne peut pas etre une personne et un aliment

    ##Defining roles##
    class mange (Personne >> Thing):
        pass

    class enseigne(Personne >> Thing): #persone est thing
        pass

    class enseigne_par(ObjectProperty):
        inverse_property = enseigne

    class mange_par(ObjectProperty):
        inverse_property = mange
```

```
class PartieDe(Thing >> Thing):
    pass

##Defining composed entities##
class Faculty (Thing):
    equivalent_to = [Thing & PartieDe. some (University) ]

class Departement (Thing):
    equivalent_to = [Thing & PartieDe. some (Faculty) ]

class Enseignant (Personne) :
    equivalent_to = [Personne & enseigne. only (Personne) ]

class Etudiant (Thing):
    equivalent_to = [Personne & enseigne_par. only (Enseignant)]

##defining instances ABOX##

class Hamza (Thing):
    equivalent_to = [Personne & mange. only (Aliment) ]

class Khellaf (Personne):
    equivalent_to = [Enseignant & mange. some (Aliment) & enseigne. only (Etudiant) ]

class MalBouffe (Thing) :
    equivalent_to = [Aliment & (mange_par.some(Personne) ) ]

AllDisjoint ([Etudiant, Enseignant])
AllDisjoint ([Khellaf, Hamza])
AllDisjoint ([MalBouffe, Departement, Faculty, University])
```

```

sync_reasoner_pellet (infer_property_values=True)
onto.save (file = "tp_rc1.owl", format = "rdxml")

```

with onto:

```

USTHB = onto.University ()
thameur = onto.Etudiant ()
Chocolat = onto.Aliment ()
belhadi = onto.Personne ()

SI = Thing () #department
INFO = Thing() #faculty

INFO.PartieDe.append(USTHB)
SI.PartieDe.append(INFO)

thameur.mange = [Chocolat]
belhadi.enseigne = [thameur]

sync_reasoner_pellet (infer_property_values=True)
onto.save(file = "tp_rc2.owl", format = "rdxml")

```

Résultats

Le résultat de notre exécution est

```

* Owlready2 * Pellet took 1.540024757385254 seconds
* Owlready * Reparenting onto.MalBouffe: {owl.Thing} => {onto.Aliment}
* Owlready * Reparenting onto.Khellaf: {onto.Personne} => {onto.Enseignant}
* Owlready * Reparenting onto.Etudiant: {owl.Thing} => {onto.Personne}
* Owlready * Reparenting onto.Hamza: {owl.Thing} => {onto.Personne}
* Owlready * (NB: only changes on entities loaded in Python are shown, other changes are done but not listed)
* Owlready2 * Running Pellet...

```

nous pouvons voir qu'il a déduit ceci

- La malbouffe est un aliment
- khellaf est enseignant
- Etudiant est une personne
- Hamza est une personne

```

* Owlready2 * Pellet took 1.4296553134918213 seconds
* Owlready * Reparenting onto.aliment1: {onto.Aliment} => {onto.MalBouffe}
* Owlready * Reparenting onto.thing1: {owl.Thing} => {onto.Departement}
* Owlready * Reparenting onto.thing2: {owl.Thing} => {onto.Faculty}
* Owlready * Reparenting onto.personne1: {onto.Personne} => {onto.Enseignant}
* Owlready * (NB: only changes on entities loaded in Python are shown, other changes are done but not listed)

```

- aliment1 (chocolat) est une malbouffe
- Thing1 (Object) est un département
- Thing2 (Object) est une Faculté
- Personne1 est un(e) enseignant(e)

TP5 : Théorie des Fonctions de Croyance

Introduction:

Ce rapport explore l'application de la théorie de Dempster-Shafer dans le contexte de la vision par ordinateur. Cette théorie offre un cadre mathématique pour gérer l'incertitude dans les données, une préoccupation majeure dans des domaines tels que la reconnaissance d'objets.

Nous examinons un scénario où trois experts discutent des sources possibles de bruit dans une image. Chacun propose des hypothèses, et la théorie de Dempster-Shafer est utilisée pour modéliser et fusionner ces connaissances. Le rapport se concentre sur l'analyse des degrés de croyance, de plausibilité et de doute pour chaque hypothèse, ainsi que sur la fusion des sources d'expertise.

Cette approche formelle fournit une méthodologie systématique pour traiter l'incertitude en vision par ordinateur, améliorant ainsi la fiabilité des analyses et des décisions dans des contextes complexes.

Exemple

Imaginons un scénario en vision par ordinateur où trois experts discutent des sources possibles d'erreur dans un système de reconnaissance d'objets. Les hypothèses pour les sources d'erreur pourraient être les suivantes :

Hypothèse A: Les erreurs proviennent du processus de capture d'image.

Hypothèse B: Les erreurs sont dues à des problèmes de traitement lors de la phase d'extraction de caractéristiques.

Hypothèse C: Les erreurs se produisent lors de la classification des objets.

Les trois experts partagent leurs opinions :

Expert 1: Atteste que 40% des erreurs proviennent du processus de capture d'image, 30% de la phase d'extraction de caractéristiques, et 30% de la classification.

Expert 2: Affirme que 20% des erreurs sont dues à la capture d'image, 50% à l'extraction de caractéristiques, et 30% à la classification.

Expert 3: Estime que les différentes hypothèses sont équiprobables.

Modélisation de connaissance

A: Capture d'image

B: Extraction de caractéristiques

C: Classification

Code Source:

Répresentation des fonctions de masse pour chaque expert :

```
from pyds import MassFunction

# Hypothèses
hypotheses = ['A', 'B', 'C']

# Expert 1
m1 = MassFunction({'A': 0.4, 'B': 0.3, 'C': 0.3})

# Expert 2
m2 = MassFunction({'A': 0.2, 'B': 0.5, 'C': 0.3})

# Expert 3
m3 = MassFunction({'A': 0.33, 'B': 0.34, 'C': 0.33})

# Affichage des masses pour chaque expert
print("Expert 1")
print(f"m_1 = {dict(m1)}")
print("m_1 =", m1)

print("\nExpert 2")
print(f"m_2 = {dict(m2)}")
print("m_2 =", m2)

print("\nExpert 3")
print(f"m_3 = {dict(m3)}")
print("m_3 =", m3)
```


résultat

```
Expert 1
m_1 = {frozenset({'A'}): 0.4, frozenset({'B'}): 0.3, frozenset({'C'}): 0.3}
m_1 = {{ 'A':0.4; 'B':0.3; 'C':0.3}}

Expert 2
m_2 = {frozenset({'A'}): 0.2, frozenset({'B'}): 0.5, frozenset({'C'}): 0.3}
m_2 = {{ 'B':0.5; 'C':0.3; 'A':0.2}}

Expert 3
m_3 = {frozenset({'A'}): 0.33, frozenset({'B'}): 0.34, frozenset({'C'}): 0.33}
m_3 = {{ 'B':0.34; 'A':0.33; 'C':0.33}}
```

Calcul de croyance et Plausibilité pour chaque expert

```
#affichage de croyance et plausibilité pour chaque expert
print("Croyance et possibilité pour l'expert 1")
print("bel_1 = ", m1.bel())
print("pl_1 = ", m1.pl())

print("Croyance et possibilité pour l'expert 2")
print("bel_2 = ", m2.bel())
print("pl_2 = ", m2.pl())

print("Croyance et possibilité pour l'expert 3")
print("bel_3 = ", m3.bel())
print("pl_3 = ", m3.pl())
```

résulte

```
Croyance et possibilité pour l'expert 1
bel_1 = {frozenset(): 0.0, frozenset({'B'}): 0.3, frozenset({'A'}): 0.4, frozenset({'C'}): 0.3,
frozenset({'B', 'A'}): 0.7, frozenset({'B', 'C'}): 0.6, frozenset({'A', 'C'}): 0.7, frozenset({'B',
'A', 'C'}): 1.0}
pl_1 = {frozenset(): 0.0, frozenset({'B'}): 0.3, frozenset({'A'}): 0.4, frozenset({'C'}): 0.3,
frozenset({'B', 'A'}): 0.7, frozenset({'B', 'C'}): 0.6, frozenset({'A', 'C'}): 0.7, frozenset({'B',
'A', 'C'}): 1.0}
```

les de degré de croyance, plausibilité et degré de doute pour toutes les hypothèses

```
# Affichage de degré de croyance, plausibilité et degré de doute pour toutes les hypothèses
print("Affichage de degré de croyance, plausibilité et degré de doute")

# Liste des experts
experts = [m1, m2, m3]

# Liste des hypothèses
hypothèses = ['A', 'B', 'C']

# Boucle pour chaque expert
for i, expert in enumerate(experts, start=1):
    print(f"\nExpert {i}")
    for hypothesis in hypotheses:
        print(f'bel_{i}({hypothesis}) =', expert.bel({hypothesis}))
        print(f'pl_{i}({hypothesis}) =', expert.pl({hypothesis}))
        print(f'Dis_{i}({hypothesis}) =', 1 - expert.pl({hypothesis}))
```

résulte:

```
Affichage de degré de croyance, plausibilité et degré de doute

Expert 1
bel_1(A) = 0.4
pl_1(A) = 0.4
Dis_1(A) = 0.6
bel_1(B) = 0.3
pl_1(B) = 0.3
Dis_1(B) = 0.7
bel_1(C) = 0.3
pl_1(C) = 0.3
```

Combinaison de Dempster & Shaffer

```
#combinaisons des masses par fusion de Dempster-Shafer
print("Dempster-Shafer Combinaison rule")
print("Dempster-Shafer Combinaison rule for m_1 and m_2 = ", m1 & m2)
print("Dempster-Shafer Combinaison rule for m_2 and m_3 = ", m2 & m3)
print("Dempster-Shafer Combinaison rule for m_1 and m_3 = ", m1 & m3)

print("Dempster-Shafer Combinaison rule for m_1, m_2 and m_3 = ", m1.combine_conjunctive(m2, m3))
print("Dempster-Shafer Combinaison rule for m_2, m_1 and m_3 = ", m2.combine_conjunctive(m1, m3))
print("Dempster-Shafer Combinaison rule for m_3, m_1 and m_2 = ", m3.combine_conjunctive(m1, m2))
```

Résultat:

```
Dempster-Shafer Combinaison rule
Dempster-Shafer Combinaison rule for m_1 and m_2 = {'B':0.46875; {'C':0.28125; {'A':0
.250000000000000006}
Dempster-Shafer Combinaison rule for m_2 and m_3 = {'B':0.5074626865671642; {'C':0
.2955223880597015; {'A':0.19701492537313434}
Dempster-Shafer Combinaison rule for m_1 and m_3 = {'A':0.3963963963963964; {'B':0
.3063063063063063; {'C':0.2972972972972973}
Dempster-Shafer Combinaison rule for m_1, m_2 and m_3 = {'B':0.47619047619047616; {'C':0
.27731092436974786; {'A':0.24649859943977595}
Dempster-Shafer Combinaison rule for m_2, m_1 and m_3 = {'B':0.47619047619047616; {'C':0
.27731092436974786; {'A':0.24649859943977595}
Dempster-Shafer Combinaison rule for m_3, m_1 and m_2 = {'B':0.4761904761904762; {'C':0
.2773109243697479; {'A':0.24649859943977595}
```

TP6 : Réseaux Bayésiens

Introduction

L'objectif de ce travail pratique (TP) est de mettre en œuvre des réseaux bayésiens en utilisant PGMPY, une bibliothèque Python dédiée aux réseaux bayésiens. Cette librairie est accessible sur le référentiel GitHub suivant : PGMPY. Dans les sections à suivre, nous allons élaborer l'implémentation d'un exemple de réseau bayésien avec une structure graphique de poly-arbre, ainsi que celle d'un réseau multi-connecté. Nous suivrons les étapes fournies dans l'énoncé du TP pour atteindre ces objectifs.

ÉTAPE 1 : Installation de PGMPY

Pour cela, il suffit de suivre les instructions indiquées dans la repository que nous avons partagé plutôt. Dans notre cas, nous avons juste exécuté la commande pip suivante au niveau de notre terminal.

```
Microsoft Windows [version 10.0.22631.3007]
(c) Microsoft Corporation. Tous droits réservés.

C:\Users\thame>pip install pgmpy
Collecting pgmpy
  Obtaining dependency information for pgmpy from https://files.pythonhosted.org/packages/eb/9a/2fcb6fd998a016cef29ca3eab30b98b6c232b6e9a0444df07f0ad47f8d/pgmpy-0.1.24-py3-none-any.whl.metadata
  Downloading pgmpy-0.1.24-py3-none-any.whl.metadata (6.3 kB)
Collecting networkx (from pgmpy)
  Obtaining dependency information for networkx from https://files.pythonhosted.org/packages/d5/f0/8fbc882ca80cf077f1b246c0e3c3465f7f415439bdea6b899f6b19f61f70/networkx-3.2.1-py3-none-any.whl.metadata
  Downloading networkx-3.2.1-py3-none-any.whl.metadata (5.2 kB)
Collecting numpy (from pgmpy)
  Obtaining dependency information for numpy from https://files.pythonhosted.org/packages/99/2b/f7114983d84303019385d93d24d729aeba67be7e083286f114188943cf3/numpy-1.26.3-cp311-cp311-win_amd64.whl.metadata
  Downloading numpy-1.26.3-cp311-cp311-win_amd64.whl.metadata (61 kB)
  61.2/61.2 kB 251.1 kB/s eta 0:00:00
Collecting scipy (from pgmpy)
  Obtaining dependency information for scipy from https://files.pythonhosted.org/packages/9a/25/5b30cb3efc9566f0ebeeac1976150316353c17031ad786ef46de5ab8dc/scipy-1.12.0-cp311-cp311-win_amd64.whl.metadata
  Downloading scipy-1.12.0-cp311-cp311-win_amd64.whl.metadata (60 kB)
  60.4/60.4 kB 1.6 MB/s eta 0:00:00
Collecting scikit-learn (from pgmpy)
  Obtaining dependency information for scikit-learn from https://files.pythonhosted.org/packages/a8/e9/3e4879974a7c4dcac2a746dde3df08d0ae8f14c74b03591616ce5f0a8b1/scikit_learn-1.4.0-1-cp311-cp311-win_amd64.whl.metadata
  Downloading scikit_learn-1.4.0-1-cp311-cp311-win_amd64.whl.metadata (11 kB)
Collecting pandas (from pgmpy)
  Obtaining dependency information for pandas from https://files.pythonhosted.org/packages/77/62/8e11962934e024a093758992bc82711e3e30efd5ea355cbfcd6e1ab5de76/pandas-2.2.0-cp311-cp311-win_amd64.whl.metadata
  Downloading pandas-2.2.0-cp311-cp311-win_amd64.whl.metadata (19 kB)
```

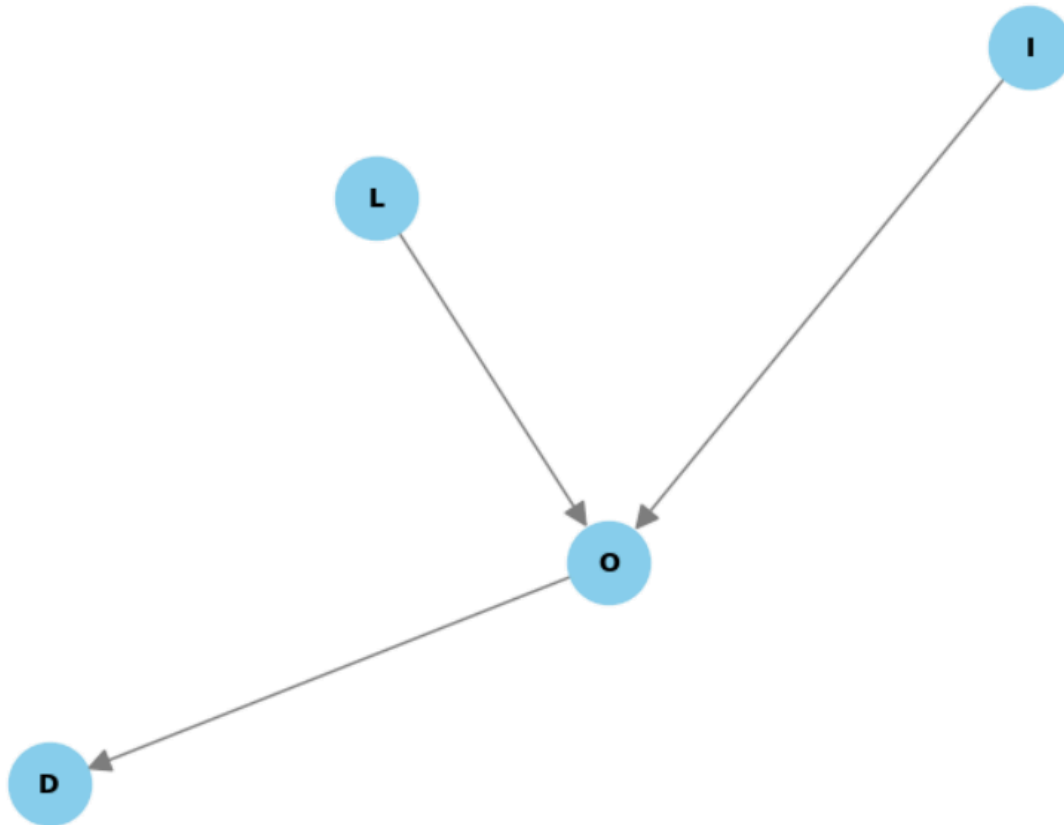
ÉTAPE 2 : Poly Arbre

I (Image Quality): Represents the quality of the image (High or Low).

L (Lighting Conditions): Represents the lighting conditions (Good or Poor).

O (Object Presence): Represents the presence of an object in the image.

D (Detection Likelihood): Represents the likelihood of detecting the object.



Matrices de probabilités :

Probabilités a priori :

- $P(I=\text{High Quality})$: 80%
- $P(I=\text{Low Quality})$: 20%
- $P(L=\text{Good Lighting})$: 70%
- $P(L=\text{Poor Lighting})$: 30%

Probabilités conditionnelles :

- $P(O=\text{Object Present} \mid I, L)$:
 - High Quality, Good Lighting: 90%
 - High Quality, Poor Lighting: 70%
 - Low Quality, Good Lighting: 40%
 - Low Quality, Poor Lighting: 20%

- $P(D=\text{Detection Likelihood} \mid O)$:

- Object Present: 95%
- Object Absent: 80%

Active Trails :

- Active trails for nodes 'I': {'I', 'O', 'D'}
- Active trails for nodes 'O': {'I', 'O', 'D'}
- Active trails for nodes 'D': {'I', 'O', 'D'}

Local Indépendances :

- Local independencies for 'I': {}
- Local independencies for 'O': {'I', 'L'}
- Local independencies for 'D': {'O'}

Global Indépendances :

- $I \perp L$
- $I \perp O \mid D$
- $I \perp D \mid O$
- $L \perp I$
- $L \perp D \mid O$
- $O \perp I \mid L$
- $O \perp L \mid I$
- $O \perp D \mid I, L$
- $D \perp I \mid O$
- $D \perp L \mid O$
- $D \perp O \mid I, L$

Code Source

```
import ...

# Define the structure of the Bayesian Network
computer_vision_model = BayesianNetwork([

    ('L', 'O'),
    ('O', 'D'),
])

# Define the relationships and probabilities
image_quality_cpd = TabularCPD(
    variable='I',
    variable_card=2,
    values=[[0.8], [0.2]], # High Quality: 80%, Low Quality: 20%
)

lighting_conditions_cpd = TabularCPD(
    variable='L',
    variable_card=2,
    values=[[0.7], [0.3]], # Good Lighting: 70%, Poor Lighting: 30%
)

object_presence_cpd = TabularCPD(
    variable='O',
    variable_card=2,
    values=[[0.9, 0.7, 0.4, 0.2], # High Quality, Good Lighting
            [0.1, 0.3, 0.6, 0.8]], # High Quality, Poor Lighting
    evidence=['I', 'L'],
    evidence_card=[2, 2]
)

detection_likelihood_cpd = TabularCPD(
    variable='D',
    variable_card=2,
    values=[[0.95, 0.2], # Object present
            [0.05, 0.8]], # Object absent
    evidence=['O'],
    evidence_card=[2]
)

# Add the relationships to the model
computer_vision_model.add_cpds(image_quality_cpd, lighting_conditions_cpd,
                                object_presence_cpd, detection_likelihood_cpd)

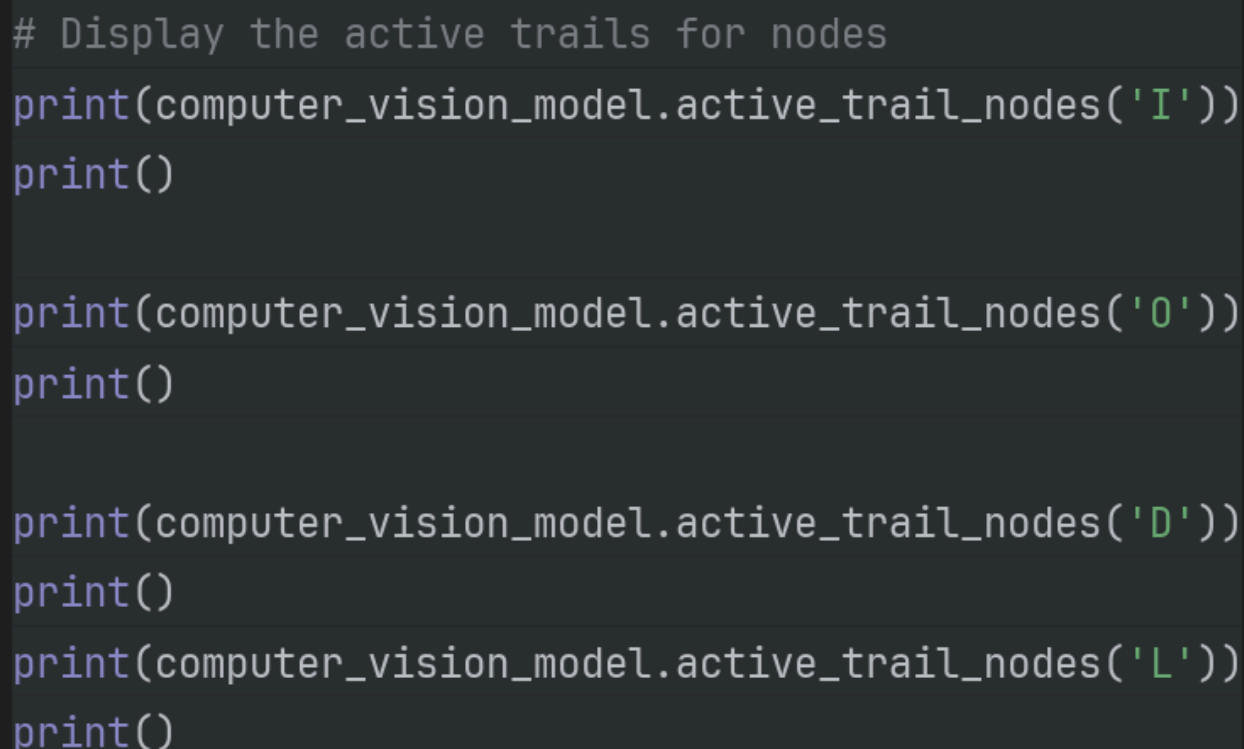
# Check the model structure
computer_vision_model.check_model()

# Display the local independencies
print(computer_vision_model.local_independencies('O'))
computer_vision_model.get_cpds()
```

résulte

```
[<TabularCPD representing P(I:2) at 0x29011646890>,  
<TabularCPD representing P(L:2) at 0x290112defd0>,  
<TabularCPD representing P(O:2 | I:2, L:2) at 0x290116463d0>,  
<TabularCPD representing P(D:2 | O:2) at 0x29011646450>]
```

Code



```
# Display the active trails for nodes  
print(computer_vision_model.active_trail_nodes('I'))  
print()  
  
print(computer_vision_model.active_trail_nodes('O'))  
print()  
  
print(computer_vision_model.active_trail_nodes('D'))  
print()  
print(computer_vision_model.active_trail_nodes('L'))  
print()
```

résulte

```
{'I': {'I', 'O', 'D'}}  
{'O': {'I', 'L', 'O', 'D'}}  
{'D': {'I', 'L', 'O', 'D'}}  
{'L': {'L', 'O', 'D'}}
```


Code

```
# Determine local independencies
print(computer_vision_model.local_independencies('I'))
print()

print(computer_vision_model.local_independencies('O'))
print()

print(computer_vision_model.local_independencies('D'))
print()
print(computer_vision_model.local_independencies('L'))
print()
```

Résultat

$(I \perp L)$

$(D \perp I, L \mid O)$

$(L \perp I)$

Code

```
# Get global independencies
independencies = computer_vision_model.get_independencies()
print(independencies)
```

résultat

(I ⊥ L)
(I ⊥ D | O)
(I ⊥ D | L, O)
(L ⊥ I)
(L ⊥ D | O)
(L ⊥ D | I, O)
(D ⊥ I, L | O)
(D ⊥ L | I, O)
(D ⊥ I | L, O)

code and résultat

```
: from pgmpy.inference import VariableElimination

# Perform inference using VariableElimination
computer_vision_infer = VariableElimination(computer_vision_model)

# Query the probability of object detection given certain conditions
prob_detection_given_high_quality = computer_vision_infer.query(variables=['D'], evidence={'I': 0})
print(prob_detection_given_high_quality)
```

```
+-----+-----+
| D   | phi(D) |
+-----+-----+
| D(0) | 0.8300 |
+-----+-----+
| D(1) | 0.1700 |
+-----+-----+
```

```
: from pgmpy.inference import VariableElimination

# Perform another inference using VariableElimination
prob_detection_given_low_quality = computer_vision_infer.query(variables=['D'], evidence={'I': 1})
print(prob_detection_given_low_quality)
```

```
+-----+-----+
| D   | phi(D) |
+-----+-----+
| D(0) | 0.4550 |
+-----+-----+
| D(1) | 0.5450 |
+-----+-----+
```

Partie 2 : Conception d'un arbre multi-connected

Define the structure of the Bayesian Network for computer vision

```
import ...

# Define the structure of the Bayesian Network for computer vision
computer_vision_model = BayesianNetwork([

    ('PixelQuality', 'Blur'),
    ('PixelQuality', 'Edges'),
    ('Blur', 'CatPresent'),
    ('Edges', 'CatPresent'),
])

# Define Conditional Probability Distributions (CPDs) for each node
lighting_cpd = TabularCPD(
    variable='Lighting',
    variable_card=2,
    values=[[0.8], [0.2]]
)

pixel_quality_cpd = TabularCPD(
    variable='PixelQuality',
    variable_card=2,
    values=[[0.9, 0.6], [0.1, 0.4]],
    evidence=['Lighting'],
    evidence_card=[2]
)

blur_cpd = TabularCPD(
    variable='Blur',
    variable_card=2,
    values=[[0.8, 0.4], [0.2, 0.6]],
    evidence=['PixelQuality'],
    evidence_card=[2]
)

edges_cpd = TabularCPD(
    variable='Edges',
    variable_card=2,
    values=[[0.7, 0.3], [0.3, 0.7]],
    evidence=['PixelQuality'],
    evidence_card=[2]
)

cat_present_cpd = TabularCPD(
    variable='CatPresent',
    variable_card=2,
    values=[[0.95, 0.2, 0.1, 0.05], [0.05, 0.8, 0.9, 0.95]],
    evidence=['Blur', 'Edges'],
    evidence_card=[2, 2]
)

# Add CPDs to the Bayesian Network model
computer_vision_model.add_cpds(lighting_cpd, pixel_quality_cpd, blur_cpd, edges_cpd,
                                cat_present_cpd)

# Check the CPDs of the model
computer_vision_model.get_cpds()
```

Création de Graphe

```
import networkx as nx
import matplotlib.pyplot as plt

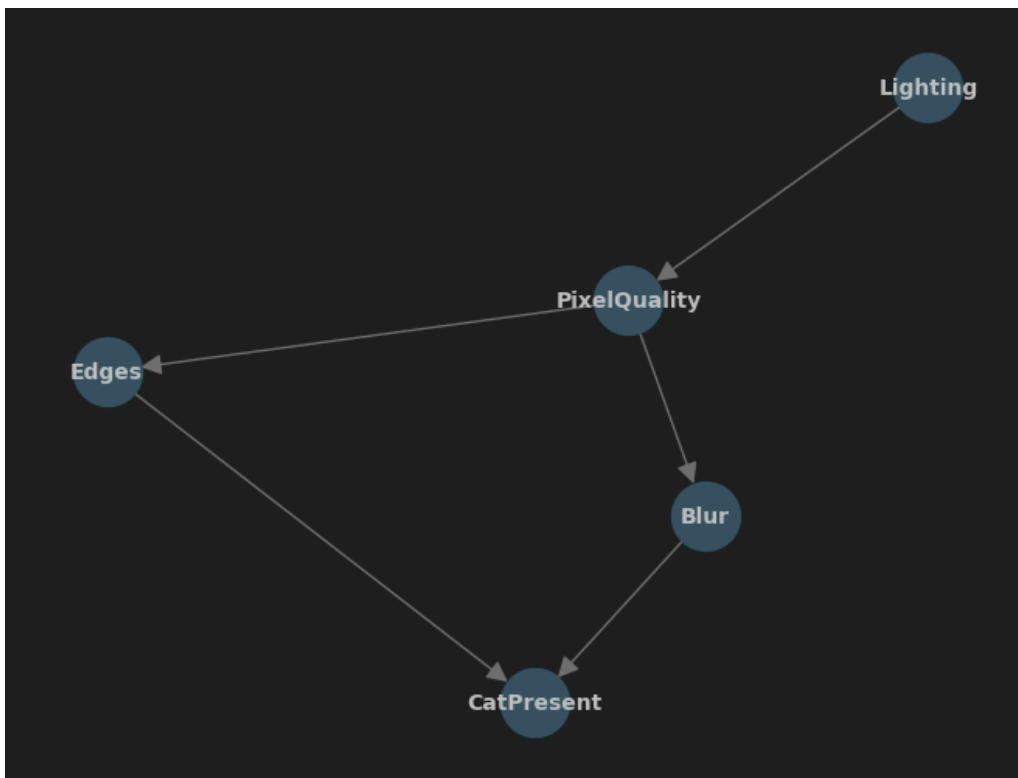
# Create a directed graph
G = nx.DiGraph()

# Define edges
edges = [ ('Lighting', 'PixelQuality'),
          ('PixelQuality', 'Blur'),
          ('PixelQuality', 'Edges'),
          ('Blur', 'CatPresent'),
          ('Edges', 'CatPresent'),
          ]

# Add edges to the graph
G.add_edges_from(edges)

# Draw the graph
pos = nx.spring_layout(G) # You can choose other layout algorithms
nx.draw(G, pos, with_labels=True, font_weight='bold', node_size=1000,
        node_color='skyblue', font_color='black', font_size=10, edge_color='gray',
        arrowsize=20)

# Show the graph
plt.show()
```



```

# Determine active trail nodes for specific nodes
print(computer_vision_model.active_trail_nodes('Lighting'))
print()
print(computer_vision_model.active_trail_nodes('CatPresent'))
print()

# Determine local independencies for specific nodes
print(computer_vision_model.local_independencies('Lighting'))
print()
print(computer_vision_model.local_independencies('Edges'))
print()
print(computer_vision_model.local_independencies('CatPresent'))
print()

computer_vision_model.get_independencies()

```

résultat

```

{'Lighting': {'Lighting', 'CatPresent', 'PixelQuality', 'Blur', 'Edges'}}
{'CatPresent': {'Lighting', 'PixelQuality', 'CatPresent', 'Blur', 'Edges'}}

(Edges ⊥ Lighting, Blur | PixelQuality)
(CatPresent ⊥ Lighting, PixelQuality | Blur, Edges)

: (Lighting ⊥ Blur, Edges, CatPresent | PixelQuality)
  (Lighting ⊥ Edges, CatPresent | Blur, PixelQuality)
  (Lighting ⊥ Blur, CatPresent | Edges, PixelQuality)
  (Lighting ⊥ Blur, Edges | CatPresent, PixelQuality)
  (Lighting ⊥ CatPresent | Blur, Edges)
  (Lighting ⊥ CatPresent | Blur, Edges, PixelQuality)
  (Lighting ⊥ Edges | CatPresent, Blur, PixelQuality)
  (Lighting ⊥ Blur | CatPresent, Edges, PixelQuality)
  (CatPresent ⊥ Lighting | PixelQuality)
  (CatPresent ⊥ Lighting, PixelQuality | Blur, Edges)
  (CatPresent ⊥ Lighting | Blur, PixelQuality)
  (CatPresent ⊥ Lighting | Edges, PixelQuality)
  (CatPresent ⊥ PixelQuality | Lighting, Blur, Edges)
  (CatPresent ⊥ Lighting | Blur, Edges, PixelQuality)
  (PixelQuality ⊥ CatPresent | Blur, Edges)
  (PixelQuality ⊥ CatPresent | Lighting, Blur, Edges)
  (Blur ⊥ Lighting, Edges | PixelQuality)
  (Blur ⊥ Edges | Lighting, PixelQuality)
  (Blur ⊥ Lighting | Edges, PixelQuality)
  (Blur ⊥ Lighting | CatPresent, PixelQuality)
  (Blur ⊥ Lighting | CatPresent, Edges, PixelQuality)
  (Edges ⊥ Lighting, Blur | PixelQuality)
  (Edges ⊥ Blur | Lighting, PixelQuality)

```

```

# Get global independencies for the entire model
computer_vision_model.get_independencies()

# Perform variable elimination for inference
vision_infer = VariableElimination(computer_vision_model)

# Query the probability of Lighting (no evidence provided)
prob_lighting = vision_infer.query(variables=['Lighting'])
print(prob_lighting)

# Query the probability of CatPresent (no evidence provided)
prob_cat_present = vision_infer.query(variables=['CatPresent'])
print(prob_cat_present)

# Query the conditional probability of CatPresent given evidence Blur=1 and Edges=0
proba_cat_present_blur_edges = vision_infer.query(variables=['CatPresent'], evidence={'Blur': 1,
'Edges': 0})
print(proba_cat_present_blur_edges)

```

```

+-----+-----+
| Lighting | phi(Lighting) |
+=====+=====+
| Lighting(0) | 0.8000 |
+-----+-----+
| Lighting(1) | 0.2000 |
+-----+-----+

+-----+-----+
| CatPresent | phi(CatPresent) |
+=====+=====+
| CatPresent(0) | 0.5349 |
+-----+-----+
| CatPresent(1) | 0.4651 |
+-----+-----+

+-----+-----+
| CatPresent | phi(CatPresent) |
+=====+=====+
| CatPresent(0) | 0.1000 |
+-----+-----+
| CatPresent(1) | 0.9000 |
+-----+-----+

```

TP7 : Contrôleurs flous

introduction

Le but de ce TP est de concevoir et d'implémenter un contrôleur flou. Pour réaliser cela, nous avons utilisé la librairie open source fuzzylogic de python et nous l'avons par la suite utilisé pour résoudre notre exemple

Exemple

Supposons que nous concevons un système de classification d'images basé sur la vision par ordinateur (CV) pour détecter des objets dans une scène en fonction de deux paramètres : la couleur de l'objet (CO) et la forme de l'objet (FO). Les ensembles flous spécifiés pour ces paramètres sont les suivants :

- Paramètre d'entrée CO :
 - Rouge (R) Triangle (20, 50, 80)
 - Vert (V) Triangle (20, 60, 90)
 - Bleu (B) Triangle (30, 80, 110)
- Paramètre d'entrée FO :
 - Carré (CA) Trapèze (2, 4, 6, 8)
 - Cercle (CE) Trapèze (6, 8, 10, 12)
 - Triangle (TR) Trapèze (10, 12, 14, 16)
- Paramètre de sortie DO (Détection de l'objet) :
 - Non détecté (ND) Triangle (10, 60, 90)
 - Faiblement détecté (FD) Triangle (20, 80, 110)
 - Moyennement détecté (MD) Triangle (40, 90, 130)
 - Fortement détecté (SD) Triangle (30, 100, 150)

La matrice d'inférence est la suivante :

CO/FO	CA	CE	TR
R	ND	ND	FD
V	ND	FD	MD
B	ND	MD	SD

Etape 1 : Définition des E/S du contrôleur

```
# Définition des E/S du contrôleur pour la vision par ordinateur
# Initialisation du premier contrôleur d'entrée pour la couleur de l'objet
C0 = Domain('Couleur de l objet', 20, 110)

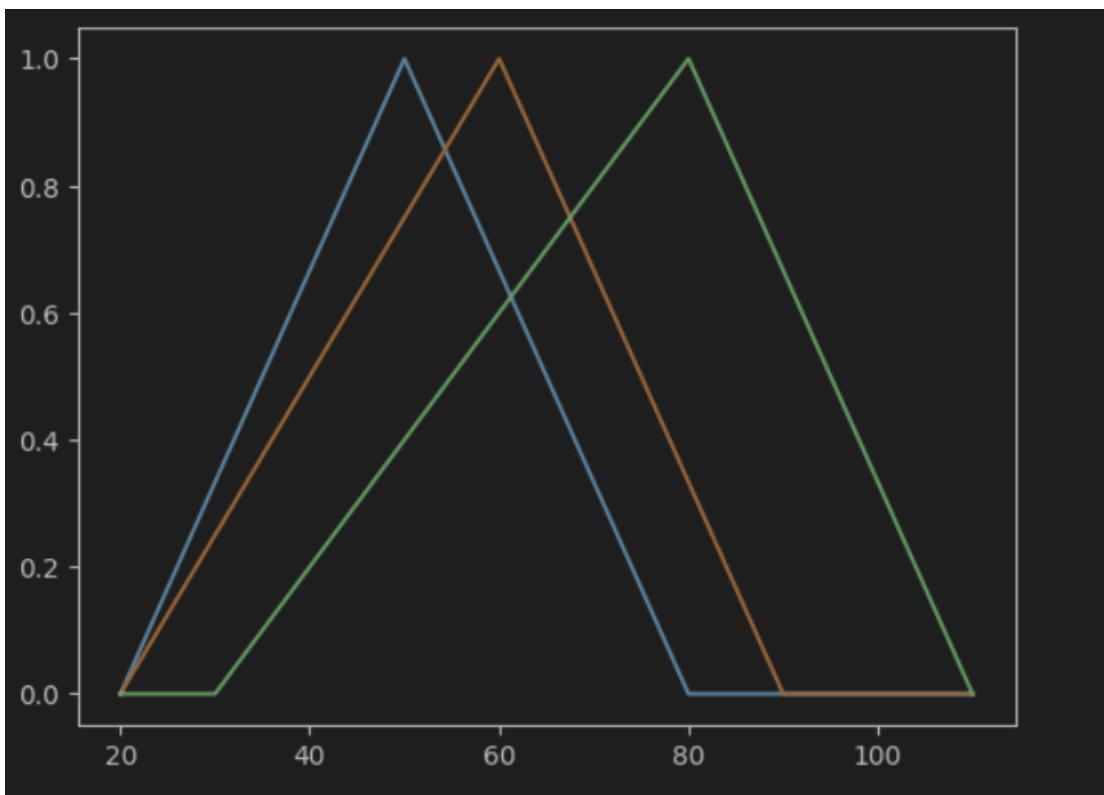
# Initialisation du deuxième contrôleur d'entrée pour la forme de l'objet
F0 = Domain('Forme de l objet', 2, 16)

# Initialisation du contrôleur de sortie pour la détection de l'objet
D0 = Domain('Détection de l objet', 10, 150)
```


Etape 2 subdivision des E/S en sous ensembles flou (Fuzzification)

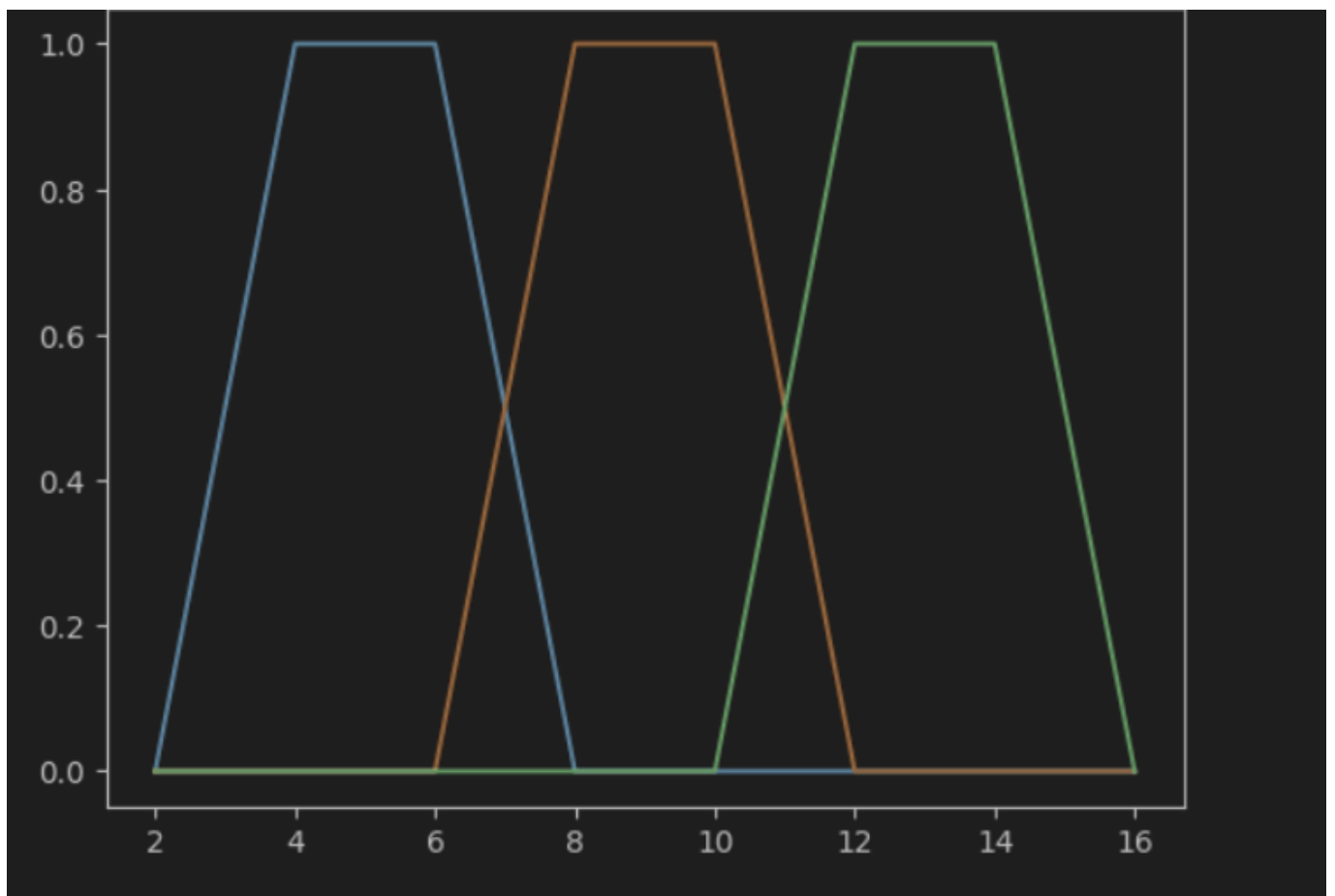
```
C0.R = trapezoid(20, 50, 50, 80)
C0.V = trapezoid(20, 60, 60, 90)
C0.B = trapezoid(30, 80, 80, 110)
C0.R.plot()
C0.V.plot()
C0.B.plot()
```

Une fois compilé, nous obtenons les résultats suivants :



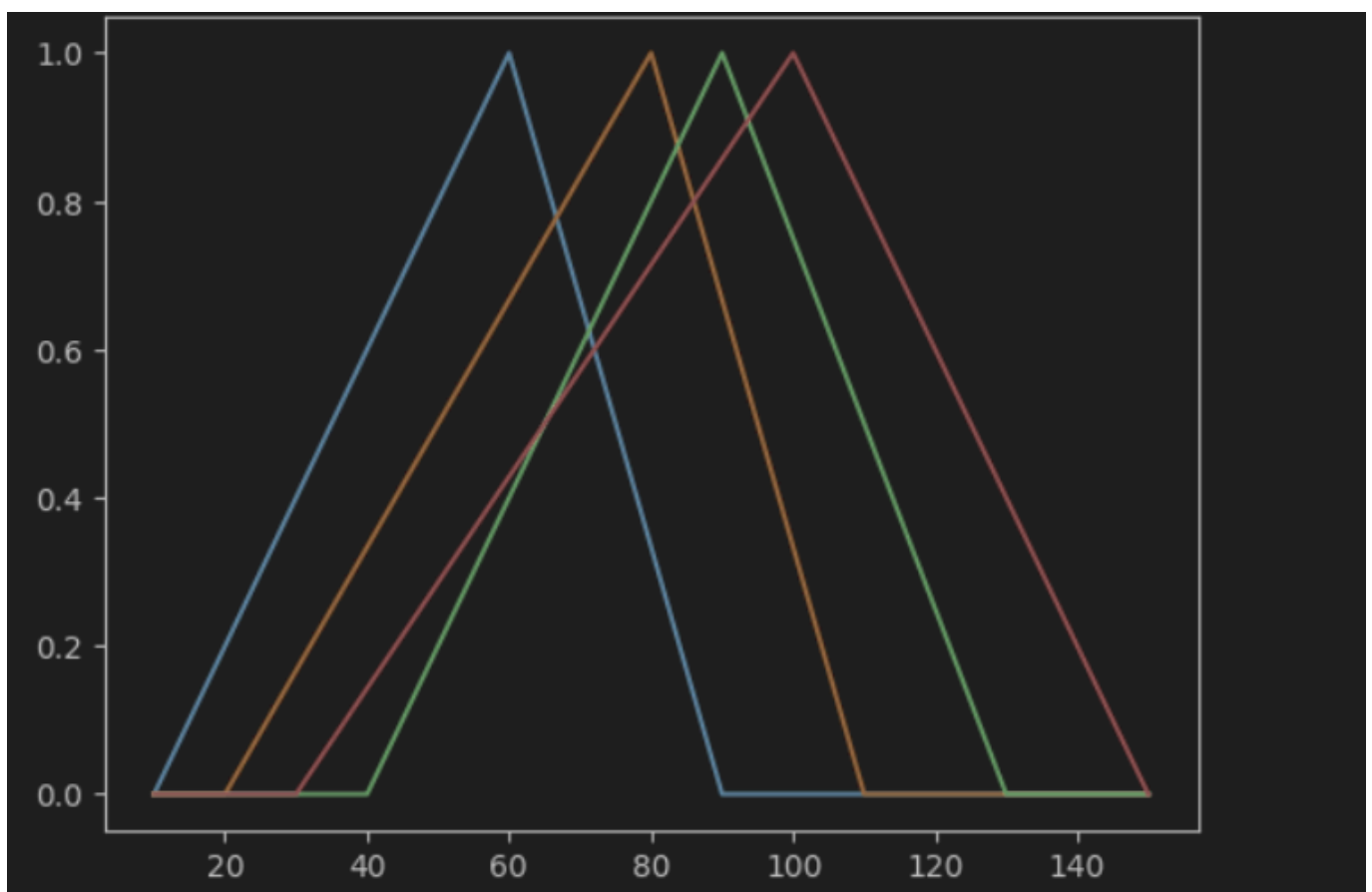
```
F0.CA = trapezoid(2, 4, 6, 8)
F0.CE = trapezoid(6, 8, 10, 12)
F0.TR = trapezoid(10, 12, 14, 16)
F0.CA.plot()
F0.CE.plot()
F0.TR.plot()
```

Une fois compilé, nous obtenons les résultats suivants :



```
D0.ND = trapezoid(10, 60, 60, 90)
D0.FD = trapezoid(20, 80, 80, 110)
D0.MD = trapezoid(40, 90, 90, 130)
D0.SD = trapezoid(30, 100, 100, 150)
D0.ND.plot()
D0.FD.plot()
D0.MD.plot()
D0.SD.plot()
```

Une fois compilé, nous obtenons les résultats suivants :



Etape 3 Définition de la base de règle floue

```
# Initialisation des règles de notre système pour la vision par ordinateur
rules = Rule({
    (CO.R, FO.CA): DO.ND,    # R1
    (CO.R, FO.CE): DO.FD,    # R2
    (CO.R, FO.TR): DO.FD,    # R3
    (CO.V, FO.CA): DO.ND,    # R4
    (CO.V, FO.CE): DO.FD,    # R5
    (CO.V, FO.TR): DO.MD,    # R6
    (CO.B, FO.CA): DO.ND,    # R7
    (CO.B, FO.CE): DO.MD,    # R8
    (CO.B, FO.TR): DO.SD     # R9
})
```

Etape 4 Application de la méthode d'inférence

```
# On récupère les pourcentages d'appartenances de nos contrôleurs d'entrée selon leur valeur
co_output = list(C0(50).values())
co_output = [float(x) for x in co_output]

fo_output = list(F0(10).values())
fo_output = [float(x) for x in fo_output]

# On applique les règles d'inférence de Mandani
do_nd = max(
    min(co_output[0], fo_output[0]), # R1
    min(co_output[0], fo_output[1]), # R2
    min(co_output[0], fo_output[2])  # R3
)

do_fd = max(
    min(co_output[1], fo_output[0]), # R4
    min(co_output[1], fo_output[1]), # R5
    min(co_output[1], fo_output[2])  # R6
)

do_md = max(
    min(co_output[2], fo_output[0]), # R7
    min(co_output[2], fo_output[1])  # R8
)

do_sd = (
    min(co_output[2], fo_output[0])  # R9
)

# Combine the results using max-min method
all_values = [do_nd, do_fd, do_md, do_sd]

print(f"Détection de l'objet non détecté {do_nd}\nDétection de l'objet faiblement détecté {do_fd}\nDétection de l'objet moyennement détecté {do_md}\nDétection de l'objet fortement détecté {do_sd}")
```

Une fois compilé, nous obtenons les résultats suivants :

```
Détection de l'objet non détecté 1.0
Détection de l'objet faiblement détecté 0.75
Détection de l'objet moyennement détecté 0.4
Détection de l'objet fortement détecté 0.0
```

Etape 5 Application de la défuzzification

```
# Défuzzification (Calcule du centre de gravité)
fig, axis = plt.subplots(figsize=(7, 5))

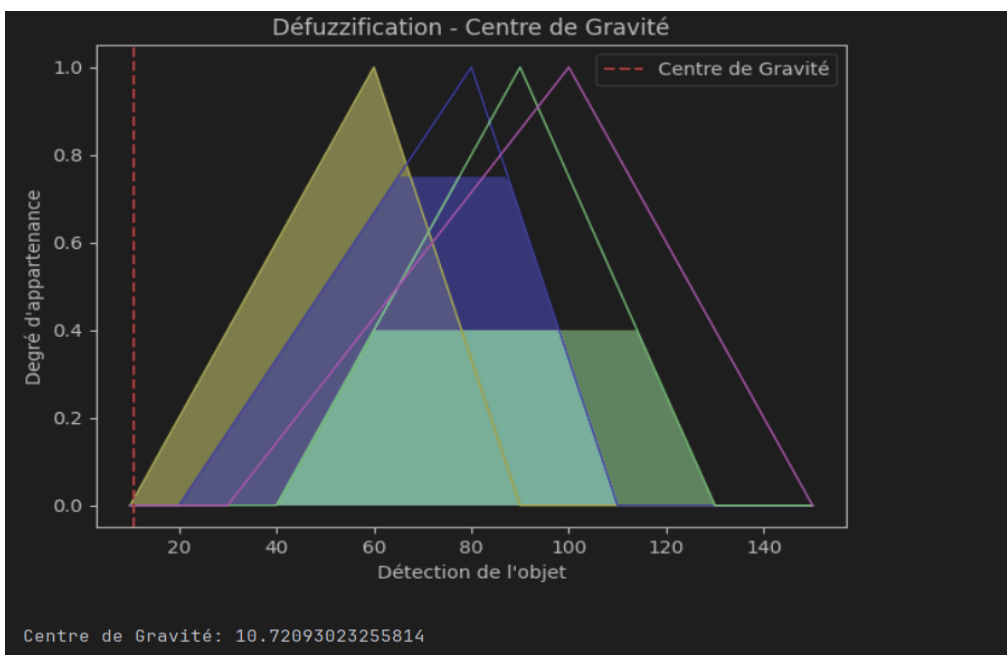
x_D0 = D0.range

D0_0 = np.zeros_like(x_D0)
parametres_D0 = [D0.ND, D0.FD, D0.MD, D0.SD]
j = 0
colors = ['y', 'b', 'g', 'm']
for each in parametres_D0:
    axis.plot(x_D0, each.array(), colors[j], linewidth=1)
    axis.fill_between(x_D0, D0_0, [min(all_values[j], x) for x in each.array()], facecolor=colors[j],
                     alpha=0.7)
    j += 1

# Mark the center of gravity on the plot
center_of_gravity = sum(x * y for x, y in zip(x_D0, all_values)) / sum(all_values)
axis.axvline(x=center_of_gravity, color='r', linestyle='--', label='Centre de Gravité')
axis.legend()

plt.title('Défuzzification - Centre de Gravité')
plt.xlabel('Détection de l\'objet')
plt.ylabel('Degré d\'appartenance')
plt.show()

print(f'Centre de Gravité: {center_of_gravity}')
```



The End