Intelligence Artificielle

~ Représentation des connaissances ~

- Rapport TP -

MOUFFOK Tayeb Abderraouf

- Master Informatique Visuelle 1 -

Groupe 2

TP 1 - Solveur SAT

Etape 1 :

Le chemin du répertoire qui contient le solveur SAT ainsi que les deux fichiers CNF est le suivant : « C:\Users\MCA\Desktop\studies\IA\Représentation des Connaissances\TP\TP1\ubcsat-1-1-0 »

Etape 2 :

Exécution du solveur sur *test1.cnf* :

```
# Solution found for -target 0

 -1 2 -3 4 5
```

Exécution du solveur sur *test2.cnf* :

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

Etape 3 :

1/ La traduction de la base de connaissances relative aux connaissances zoologiques sous la forme cnf a été effectuée et est dans le fichier *« mollusques.cnf ».*

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

Test de la satisfiabilité de cette base :

```
# Solution found for -target 0

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

La base de connaissances est donc satisfiable.

2/ Fichiers Benchmark :

- *« benchmark_file_1.cnf »* n'est pas satisfiable.
- *« benchmark_file_2.cnf »* est satisfiable.

Etape 4 :

L'algorithme est dans le fichier « inference.py »

```
def clause_to_list(clause):
    '''
        Fonction qui transforme une clause en liste de littéraux
    '''
    clause_list = clause.replace("0","").split()
    clause_list = [int(i) for i in clause_list]
    return clause_list

def lister_clauses(file):
    '''
        Fonction qui transforme un fichier CNF en une liste de clauses
    '''
    cnf = file.strip().split("\n")
    nb_var = int(cnf[0].split()[-2])
    del cnf[0]
    nb_clauses = len(cnf)
    cnf_clauses = []
    for i in range(nb_clauses):
        cnf_clauses.append(clause_to_list(cnf[i]))
    return cnf_clauses


def solve_cnf(cnf_file):
    '''
        Fonction qui applique le solveur SAT sur le CNF donné en paramètre
    '''
    import os
    f = open('cnf_file_tmp.cnf', 'w')
    f.write(cnf_file)
    f.close()
    os.system('ubcsat -alg saps -i cnf_file_tmp.cnf -solve > tmp')
    output = open('tmp', 'r').read()
    os.remove('tmp')
    os.remove('cnf_file_tmp.cnf')

    if "# Solution found for -target 0" in output:
        return True
```

```
    elif "# No Solution found for -target 0" in output:
        return False
    else:
        print(output)
        return None


def count_variables(clauses):
    '''
        Fonction qui compte le nombre de variables présentes dans une liste de clauses
    '''
    variables = []
    for clause in clauses:
        for litteral in clause:
            if abs(litteral) not in variables:
                variables.append(abs(litteral))
    return len(variables)


def make_cnf_file(clauses):
    '''
        Fonction qui crée un fichier CNF à partir de clauses données en paramètres
    '''
    nb_variables = count_variables(clauses)
    nb_clauses = len(clauses)
    file = "p cnf "+str(nb_variables)+" "+str(nb_clauses)+"\n"
    for clause in clauses:
        for litteral in clause:
            file += str(litteral)+ " "
        file += "0\n"
    return file


def test_inference(bc_cnf_file, litteral):
    '''
        Fonction qui teste l'inférence d'une BC sur un littéral
    '''
    file = open(bc_cnf_file, 'r')
    data = file.read()
    file.close()

    clauses = lister_clauses(data)
    clauses.append(clause_to_list(litteral))

    file = make_cnf_file(clauses)

    solution = solve_cnf(file)

    if solution is not None:
        if solution == True:
            print("La BC \""+ bc_cnf_file +"\" n'infère pas le littéral "+litteral)
```

```
        else:
            print("La BC \""+ bc_cnf_file +"\" infère le littéral "+litteral)


if __name__ == "__main__":
    test_inference("test1.cnf", "-2 0")
```

- Algorithme d'inférence écrit en Python -

TP 2 - Logique des défauts

Pour ce TP, on va exploiter la Toolbox implémentée en Java nommée ***Orbital***, afin de faire l'exercice 5 du TD2, on va également voir un autre exemple par la suite.

Exercice 5 - TD2

Soit l'ensemble de défauts D={d1,d2} avec $d1 = A: B/C$ et $d2 = A:\neg C/D$.
Quelles sont les extensions des théories Δ_1, Δ_2, Δ_3 définies par :

1. Δ_1=<W_1,D> avec W_1={ $\neg A$ }
2. Δ_2=<W_2,D> avec W_2={A, $\neg$B }
3. Δ_3=<W_3,D> avec W_3={A, $\neg B \wedge C$ }

On commence tout d'abord par initialiser les 3 mondes W1, W2, W3 comme présenté ci-dessous.

```
// Initialisation des 3 Mondes

// Monde W1
WorldSet world1 = new WorldSet();
world1.addFormula(a.e.NOT+"A");

// Monde W2
WorldSet world2 = new WorldSet();
world2.addFormula("A");
world2.addFormula(a.e.NOT+"B");

// Monde W3
WorldSet world3 = new WorldSet();
world3.addFormula("A");
world3.addFormula(a.e.NOT+"B"+a.e.AND+"C");
```

Ensuite, on déclare l'ensemble des défauts comme suit :

```
// Déclaration des deux règles

// règle d1

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

```
// règle d2
DefaultRule rule2 = new DefaultRule();
rule2.setPrerequisite("A");
rule2.setJustificatoin("~C");
rule2.setConsequence("D");

// Création de l'ensemble de règles
RuleSet myRules = new RuleSet();

// Ajout des deux règles crées plus tôt à l'ensemble
myRules.addRule(rule1);
myRules.addRule(rule2);
```

Et là, on est prêt pour appeler le raisonneur et générer les extensions possibles pour chacune des théories.

```
// Appel des raisonneurs pour générer les extensions possibles pour chacune des
théories

DefaultReasoner loader1 = new DefaultReasoner(world1, myRules);
HashSet<String> extensions1 = loader1.getPossibleScenarios();
```

Ci-dessous les résultats pour les théories 1 et 2.

```
#######################  Théorie 1

Pour le monde suivant :
        ~A
Et pour cet ensemble de règles :
        [(A):(B) ==> (C)] , [(A):(~C) ==> (D)]

Il n'y a pas d'extension possible.



#######################  Théorie 2

Trying  A & ~B
Trying  A & ~B
Pour le monde suivant :
        A & ~B
Et pour cet ensemble de règles :
        [(A):(B) ==> (C)] , [(A):(~C) ==> (D)]

Les extensions possibles :
         Ext: Th(W U (D))
        = D & ~B &  A
```

Quant à la théorie 3, le raisonneur retourne une erreur lorsqu'il rencontre une règle de défaut utilisable mais pas applicable.

```
####################  Théorie 3

Trying  A & ~B&C
Trying  A & ~B&C
error
Exception in thread "main" java.lang.NullPointerException
        at be.fnord.util.logic.WFF.getClosure(WFF.java:231)
        at be.fnord.util.logic.defaultLogic.RuleSet.applyRules(RuleSet.java:116)
        at be.fnord.util.logic.DefaultReasoner.getPossibleScenarios(DefaultReasoner.java:114)
        at be.fnord.DefaultLogic.TP_LogiqueDesDefauts.exercice5(TP_LogiqueDesDefauts.java:115)
        at be.fnord.DefaultLogic.TP_LogiqueDesDefauts.main(TP_LogiqueDesDefauts.java:187)
```

Exemple supplémentaire

Dans cet exemple, on va voir quels sont les extensions de la théorie **<W, D>** tel que :

$W = \{ B \rightarrow \neg A \wedge \neg C \}$ et $D = \{ d1, d2, d3 \}$ avec $d1 = : A / A$, $d2 = : B / B$, $d3 = : C / C$.

En suivant les mêmes étapes précédentes :

```java
// Déclaration du monde W
WorldSet myWorld = new WorldSet();
myWorld.addFormula("(B -> (~A & ~C))");

// Déclaration de l'ensemble des défauts
DefaultRule rule1 = new DefaultRule();
rule1.setPrerequisite(a.e.EMPTY_FORMULA);
rule1.setJustificatoin("A");
rule1.setConsequence("A");

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

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

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

```
// Appel du raisonneur pour générer les extensions
DefaultReasoner loader = new DefaultReasoner(myWorld, myRules);
HashSet<String> extensions = loader.getPossibleScenarios();
```

Et voici donc ci-dessous les résultats obtenus.

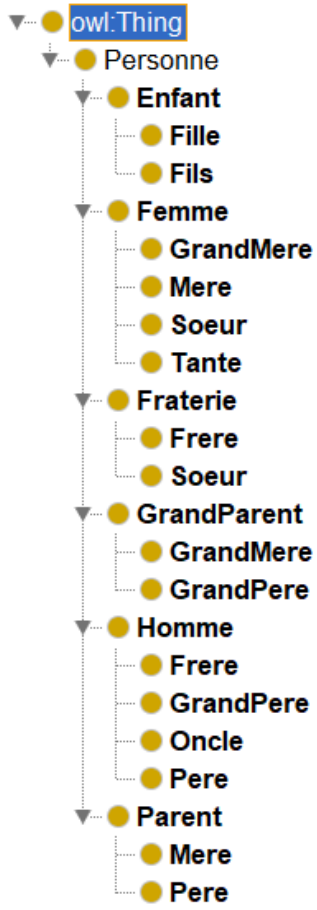
```
Exemple supplémentaire :

Trying  (B -> (~A & ~C))
Trying  (B -> (~A & ~C))
Trying  (B -> (~A & ~C))
Trying  (B -> (~A & ~C))
Trying  (B -> (~A & ~C))
Trying  (B -> (~A & ~C))
Trying  (B -> (~A & ~C))
Trying  (B -> (~A & ~C))
Trying  (B -> (~A & ~C))
Pour le monde suivant :
        (B -> (~A & ~C))
Et pour cet ensemble de règles :
        [([]):(A) ==> (A)] , [([]):(B) ==> (B)] , [([]):(C) ==> (C)]
Les extensions possibles :
         Ext: Th(W U (B))
        = (~C | ~B) &  B & ~A & ~C & (~A | ~B)
         Ext: Th(W U (C & A))
        = (~C | ~B) &  A & C & ~B & (~A | ~B)
fin
```

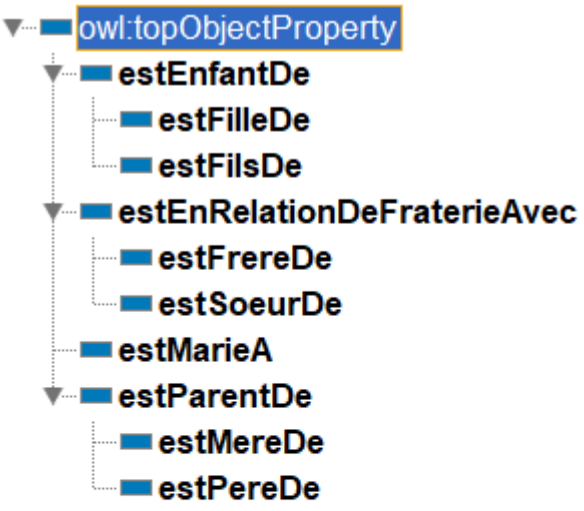
TP 3 - Logique des descriptions

Pour ce TP, on utilisera l'outil Protégé, qui permet de créer des ontologies. On utilisera Protégé en combinaison avec le moteur d'inférence Hermit, qui est pré-installé pour raisonner sur les ontologies (inférer des nouveaux faits), et vérifier la cohérence des modélisations réalisées.

On va implémenter un exemple similaire à celui présent dans la série de TP, il s'agit d'exprimer des connaissances relatives aux relations familiales.

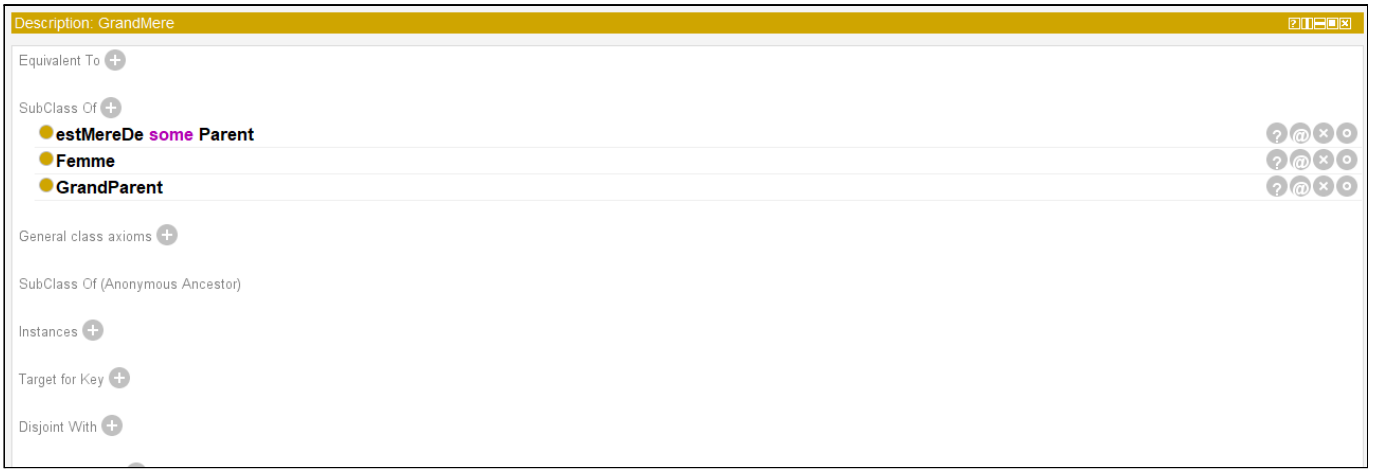
T-BOX





Les captures d'écran ci-dessus représentent la T-Box de notre modèle, à droite se trouvent les concepts atomiques - ou classes -, à gauche se trouvent les rôles atomiques - ou propriétés d'objets -.

Chaque concept et chaque rôle ont leur propre description, par exemple, on a décrit le concept Grand-mère, comme étant une subsomption du concept Femme, Grand-parent, et ayant au moins un enfant de classe Parent. *(voir sur Protégé pour explorer toutes les descriptions)*

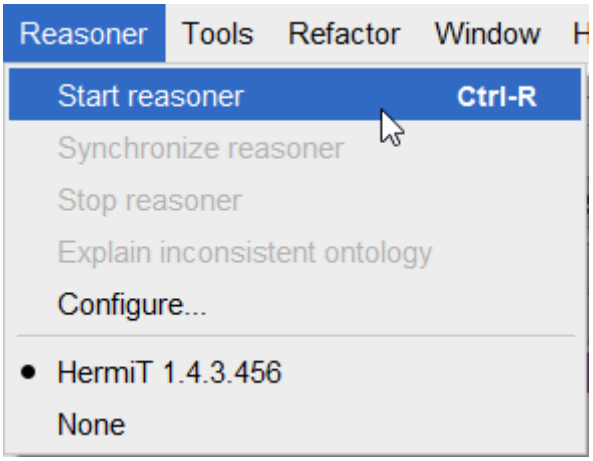


A-BOX

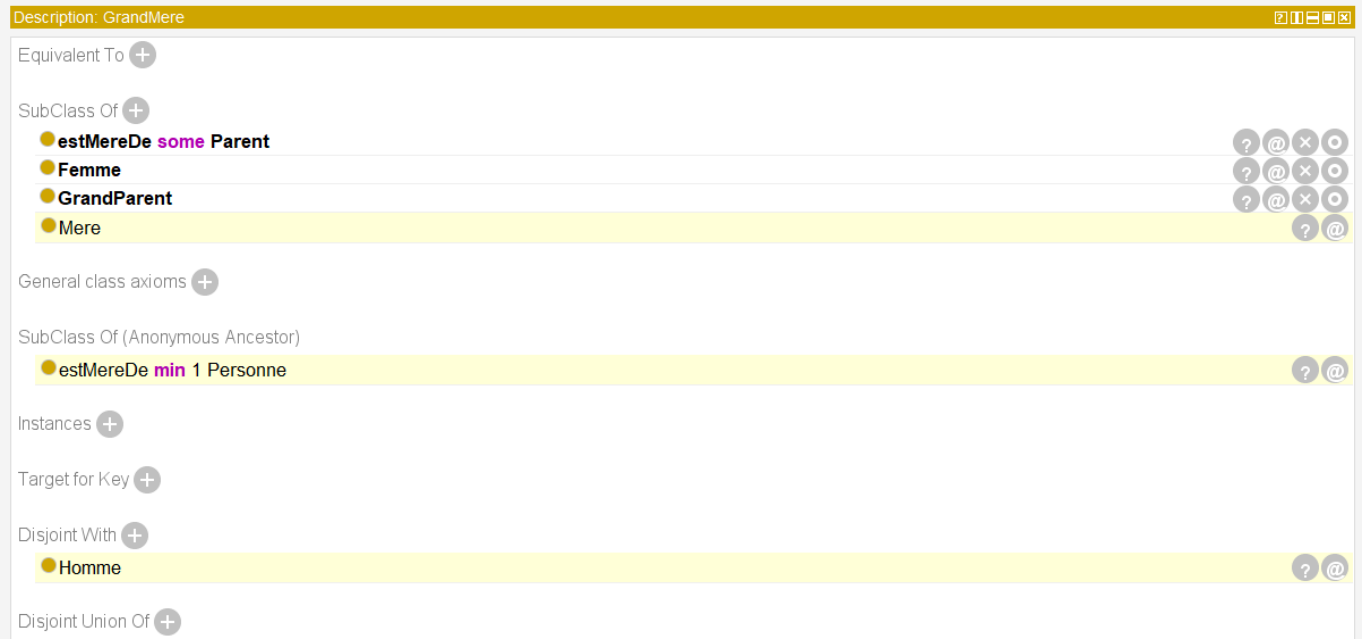
Ci-dessus la liste des individus qui forment la A-Box, pour chaque individu a été également modélisé sa propre description *(voir sur Protégé pour explorer la description de chaque individu).*

- Alex est une femme et est marié à Thomas
- Chloé est la fille de Marie et de Peter
- Claude est une femme et est la fille de Sylvie
- Jean est un homme et est le fils de John
- John est un homme et est marié à Sylvie
- Marie est une femme et est marié à Peter
- Michael est un homme et est le fils de Thomas et d'Alex
- Paul est un homme et est le fils de Peter
- Peter est un homme et est marié à Marie
- Sylvie est une femme, est marié à John et est la fille de Marie et de Peter
- Thomas est un homme, est marié à Alex et est le fils de Peter
- Tom est un homme et est le fils d'Alex et de Thomas

Une fois la T-Box et la A-Box modélisées et implémentées, on est prêt à faire appel au raisonneur HermiT pour simuler le raisonnement et avoir les déductions en exploitant les connaissances insérées.



Les déductions effectuées sont mises en évidence en jaune *(voir sur Protégé pour explorer toutes les déductions).*



TP 4 - Logique des croyances

Pour ce TP, on va utiliser la Toolbox implémentée sur Python nommée ***py_dempster_shafer*** pour faire l'exercice 1 du TD.

Exercice 1 :

Nous désirons développer un système afin de définir l'espèce d'une plante végétale "l'iris». Les trois espèces possibles sont: Setosa, Versicolor, Virginica. Trois experts en botanique évaluent l'appartenance d'un échantillon à une espèce comme suit:

- le premier expert appui l'appartenance de l'échantillon au type Versicolor avec un degré 0.6.
- le second expert estime l'appartenance de l'échantillon:
  - au type Setosa à 0.1,
  - et au type Setosa ou au type Virginica à 0.5.
- le troisième expert ne donne pas d'indice particulier.

1- Modélisez ces connaissances en utilisant la théorie des fonctions de croyance
2- Calculez les degrés de croyance et les degrés de plausibilité associés à la distribution du premier expert.
3- Quelle est l'hypothèse la plus soutenue?
4- Comment combiner les différentes hypothèses en utilisant la théorie des fonctions de croyance? Explicitez chaque étape.

On commence par modéliser les connaissances décrites dans l'énoncé comme suit :

```
''' OMEGA = {Setosa (S), Versicolor (V), Virginica (I) } '''

# 1/ Modélisation des connaissances
m1 = MassFunction({'V':0.6, 'SVI':0.4})
m2 = MassFunction({'S':0.1, 'SI':0.5, 'SVI':0.4})
m3 = MassFunction({'SVI':1})
```

Puis, pour calculer les degrés de croyances et les degrés de plausibilité associés à la distribution d'une certaine connaissance, on a implémenté la fonction suivante.

```
def calcul_degres_croyances_plausibilite(m):
    ''' Fonction qui calcule les degrés de croyances et de plausibilités '''
    hypotheses = ['V', 'S', 'I', 'VI', 'VS', 'SI', 'SVI', '']
    ms = []
    beliefs = []
    plausibilities = []
    for hypothese in hypotheses:
        ms.append(m[hypothese])
        beliefs.append(m.bel(hypothese))
        plausibilities.append(m.pl(hypothese))

    return ms, beliefs, plausibilities
```

Pour calculer les degrés associés à la distribution du premier expert, on appelle la fonction précédente en passant comme paramètre m1, ce qui donne le résultat ci-dessous.

```
2/ Calcul des degrés de croyance et des degrés de plausibilité du premier expert
```

	m	Bel	Pl
V	0.6	0.6	1
S	0	0	0.4
I	0	0	0.4
VI	0	0.6	1
VS	0	0.6	1
SI	0	0	0.4
SVI	0.4	1	1
	0	0	0

Enfin, pour combiner les différentes hypothèses, il suffit d'exécuter la ligne suivante.

```
# 4/ Combinaison de m1 avec m2 et m3
m123 = m1 & m2 & m3
```

On appelle la fonction de calcul précédente pour afficher les résultats de la combinaison des 3 hypothèses.

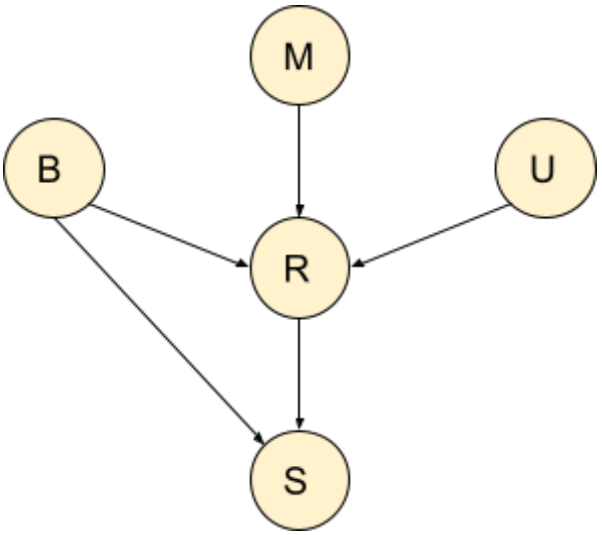
```
4/ Combinaison de m1 avec m2 et m3 :
```

	m	Bel	Pl
V	0.375	0.375	0.625
S	0.0625	0.0625	0.625
I	0	0	0.5625
VI	0	0.375	0.9375
VS	0	0.4375	1
SI	0.3125	0.375	0.625
SVI	0.25	1	1
	0	0	0

TP 5 - Réseaux Bayésiens

Pour ce TP, on va exploiter la librairie Python “***pgmpy”***, qui permet de modéliser et d

On va implémenter un exemple, qui représente un problème où un réseau social souhaite modérer les publications et les commentaires publiés sur la plateforme, et suspendre les comptes des mauvais utilisateurs. Supposons le réseau bayésien ci-dessous avec les informations suivantes.



M : Une prédiction d'un modèle de ML qui peut lire le contenu et donner un score (probabilité) que ce contenu soit signalé.

U : Un autre utilisateur signale le contenu.

B : Le compte a été suspendu auparavant pour tout contenu de mauvaise qualité.

R : Score (probabilité) que le contenu soit retiré de la plateforme.

S : Score (Probabilité) que le compte soit suspendu

Soit la distribution de probabilités suivante :

U=	T	F
P(U)	0.15	0.85

M=	T	F
P(M)	0.05	0.95

B=	T	F
P(B)	0.10	0.90

S=	T	F
P(S|RB)	0.40	0.60
P(S|R!B)	0.05	0.95
P(S|!RB)	0.12	0.88
P(S|!R!B)	0.02	0.98

R=	T	F
P(R|MBU)	0.95	0.05
P(R|MB!U)	0.90	0.10
P(R|M!BU)	0.85	0.15
P(R|M!B!U)	0.76	0.24
P(R|!MBU)	0.18	0.82
P(R|!MB!U)	0.06	0.94
P(R|!M!BU)	0.14	0.86
P(R|!M!B!U)	0.04	0.96

Pour modéliser ce réseau, on commence par initialiser le modèle, puis y ajouter les nœuds et les arcs du réseau comme suit.

```
bayesNet = BayesianModel() # Initialisation du modèle

### Ajout des noeuds
bayesNet.add_node("M")
bayesNet.add_node("U")
bayesNet.add_node("R")
bayesNet.add_node("B")
bayesNet.add_node("S")

### Ajout des arcs
bayesNet.add_edge("M", "R")
bayesNet.add_edge("U", "R")
bayesNet.add_edge("B", "R")
bayesNet.add_edge("B", "S")
bayesNet.add_edge("R", "S")
```

On ajoute ensuite les distributions des probabilités conditionnelles.

```
### Ajout des probabilités
cpd_A = TabularCPD('M', 2, values=[[.95], [.05]])
cpd_U = TabularCPD('U', 2, values=[[.85], [.15]])
cpd_H = TabularCPD('B', 2, values=[[.90], [.10]])

cpd_S = TabularCPD('S', 2, values=[[0.98, .88, .95, .6], [.02, .12, .05, .40]],
                   evidence=['R', 'B'], evidence_card=[2, 2])

cpd_R = TabularCPD(variable='R', variable_card=2,
                   values=[[0.96, .86, .94, .82, .24, .15, .10, .05], [.04, .14,
.06, .18, .76, .85, .90, .95]],
                   evidence=['M', 'B', 'U'], evidence_card=[2, 2, 2])
bayesNet.add_cpds(cpd_A, cpd_U, cpd_H, cpd_S, cpd_R)
```

Une fois que cela est fait, on crée un solveur qui utilise l'élimination des variables en interne pour pouvoir inférer notre modèle.

```
# Création du solveur qui permettra d'inférer notre réseau
solver = VariableElimination(bayesNet)
```

Arrivé ici, on peut effectuer des inférences sur notre modèle en utilisant le solver.

Par exemple, pour trouver la probabilité qu'un contenu doit être retiré de la plateforme, il suffit de calculer ***P(R)***.

```
# Calcul de P(R)
result = solver.query(variables=['R'])
print("P(R) = ", result.values[1])
```

```
P(R) =  0.09378000000000003
```

Autre exemple, on trouve la probabilité qu'un contenu doit être retiré sachant que le modèle de Machine Learning l'a signalé en calculant ***P(R|M)***.

```
# Calcul de P(R|M)
result = solver.query(variables=['R'], evidence={'M': 1})
print("P(R|M) = ", result.values[1])
```

```
P(R|M) =  0.7869
```

Il est aussi possible de faire des calculs un peu plus complexes avec *pgmpy*, par exemple, on peut calculer la probabilité qu'un compte soit suspendu sachant que le compte a déjà été suspendu auparavant et que son dernier contenu a été retiré de la plateforme, ce qui revient à calculer $P(S|B \wedge R)$.

```
# Calcul de P(S|B^R)
result = solver.query(variables=['S'], evidence={'B': 1, 'R': 1})
print("\n\n\n\nP(S|B^R) = ", result.values[1])
```

```
P(S|B^R) =  0.4
```

TP 6 - Logique Floue

Pour ce TP, on va utiliser une librairie Python qui s'appelle ***simpful***, qui permet de faire du raisonnement en logique floue. On va donc exploiter cette bibliothèque pour faire l'exercice 1 du TD, dont l'énoncé est ci-dessous.

Exercice 1 :

Considérons un système de contrôle des risques de la cybercriminalité (RC) en fonction de trois paramètres : la technologie de la cyber-sécurité (TC), les normes de la cyber-sécurité (NC) et la portée de l'information (PI). Ces différents paramètres sont spécifiés par les ensembles flous suivants:

- Paramètre d'entrée **TC** :

Avancée (AV)	Triangle (20,35,45)
Acceptable (AC)	Triangle (35,45,60)
Insuffisante (IN)	Triangle (45,60,80)

- Paramètre d'entrée **NC** :

Dans les normes(DN)	Trapèze (9,24,40,55)
Hors normes (HN)	Trapèze (40,55,60,70)

- Paramètre d'entrée **PI** :

Très grande (TG)	Trapèze (5,10,15,20)
Grande (GR)	Trapèze (15,20,25,30)
Moyenne (MO)	Trapèze (25,30,35,40)
Faible (FA)	Trapèze (35,40,45,50)

- Paramètre de sortie **RC** :

très fort (TF)	Triangle (-80,-50,-10)
Fort (FO)	Triangle (-50,-10,10)
Moyen (MO)	Triangle (-10,10,40)
Faible (FA)	Triangle (10,40,70)

La matrice d'inférence est la suivante:

NC	DN			HN		
TC / PI	AV	AC	IN	AV	AC	IN
TG	FA	FA	FA	MO	MO	MO
GR	FA	MO	MO	MO	FO	FO
MO	MO	MO	MO	FO	FO	TF
FA	FO	FO	FO	TF	TF	TF

a- Spécifiez les différentes étapes de la conception d'un contrôleur flou.

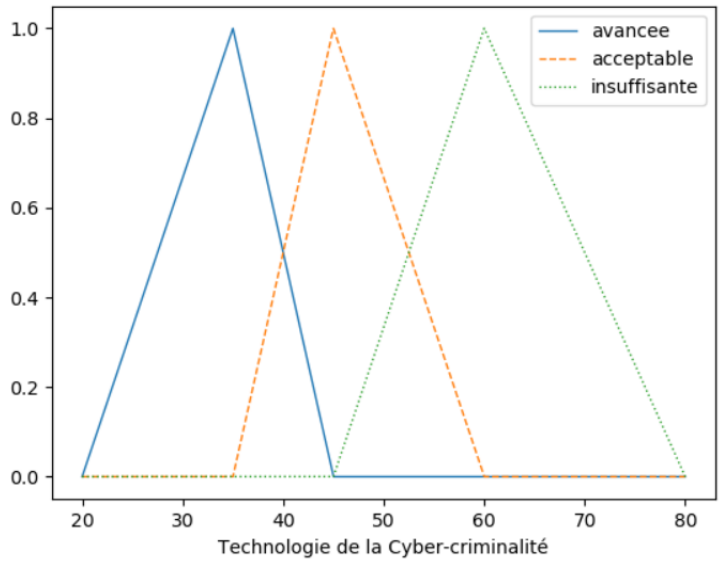
b- Appliquez chaque étape au problème donné en précisant les connaissances utilisées.

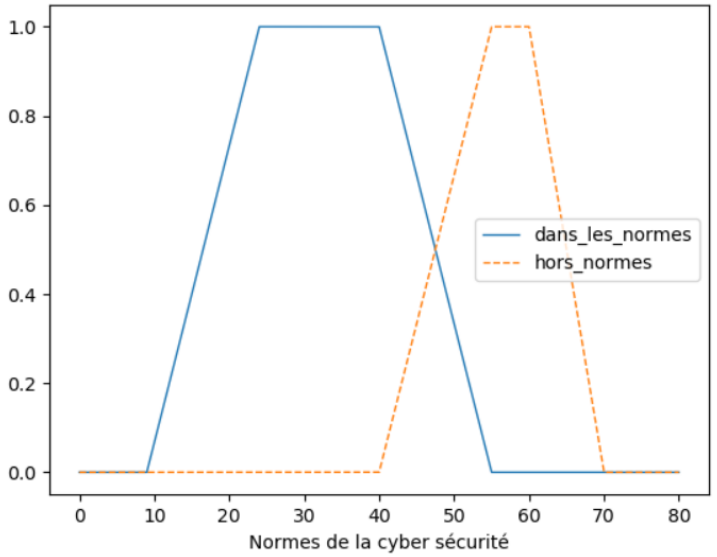
c- Simuler le fonctionnement du contrôleur avec les paramètres suivants :
TC=52 ; NC=42 ; PI=17

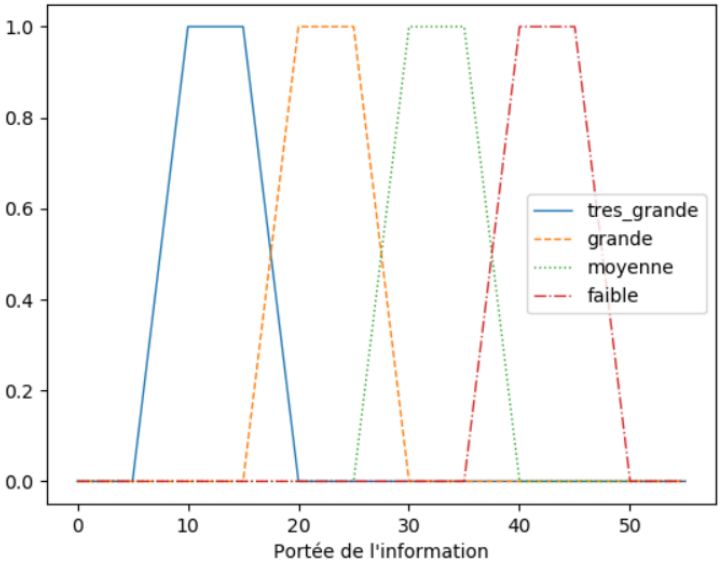
On commence par déclarer les fonctions d'appartenances des entrées et de la sortie comme suit.

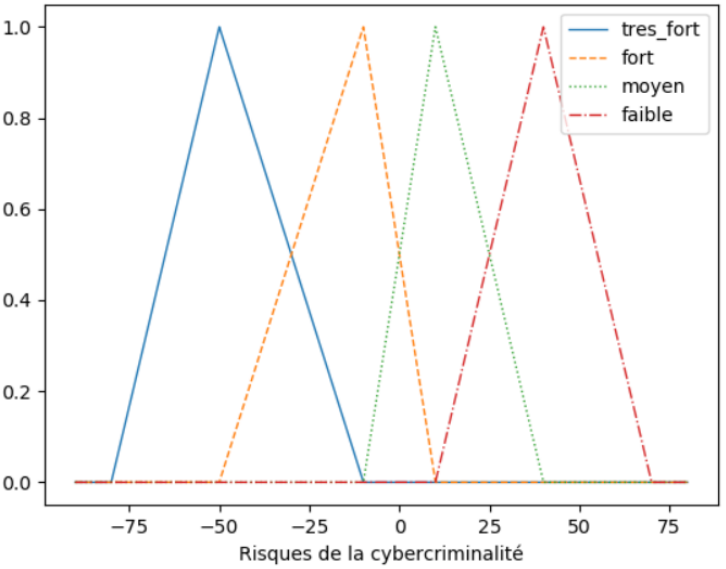
```
FS = FuzzySystem()

###### Technologie de la Cyber-criminalité (TC)

O_AV = TriangleFuzzySet(20, 35, 45, term="avancee")
O_AC = TriangleFuzzySet(35, 45, 60, term="acceptable")
O_IN = TriangleFuzzySet(45, 60, 80, term="insuffisante")

LV_TC = LinguisticVariable([O_AV, O_AC, O_IN], universe_of_discourse=[20,80])
FS.add_linguistic_variable("TC", LV_TC)


###### Normes de la cyber sécurité (NC)

O_DN = TrapezoidFuzzySet(9, 24, 40, 55, term="dans_les_normes")
O_HN = TrapezoidFuzzySet(40, 55, 60, 70, term="hors_normes")

LV_NC = LinguisticVariable([O_DN, O_HN], universe_of_discourse=[0, 80])

FS.add_linguistic_variable("NC", LV_NC)


###### Portée de l'information (PI)

O_TG = TrapezoidFuzzySet(5, 10, 15, 20, term="tres_grande")
O_GR = TrapezoidFuzzySet(15, 20, 25, 30, term="grande")
O_MO = TrapezoidFuzzySet(25, 30, 35, 40, term="moyenne")
O_FA = TrapezoidFuzzySet(35, 40, 45, 50, term="faible")

LV_PI = LinguisticVariable([O_TG, O_GR, O_MO, O_FA],universe_of_discourse=[0, 55])

FS.add_linguistic_variable("PI", LV_PI)


###### Risques de la cybercriminalité (RC)

O_TF = TriangleFuzzySet(-80, -50, -10,   term="tres_fort")
O_FO = TriangleFuzzySet(-50, -10, 10,  term="fort")
O_MOrc = TriangleFuzzySet(-10, 10, 40, term="moyen")
O_FArc = TriangleFuzzySet(10, 40, 70, term="faible")

LV_RC = LinguisticVariable([O_TF, O_FO, O_MOrc, O_FArc],
universe_of_discourse=[-90,80])
FS.add_linguistic_variable("RC", LV_RC)
```

On peut visualiser chacune des fonctions :









On rajoute par la suite les règles présentes dans la matrice d'inférence, comme fait ci-dessous.

```
###### Ajout des règles

FS.add_rules([
    "IF (NC IS dans_les_normes) AND (TC IS avancee) AND (PI IS tres_grande) THEN (RC IS
faible)",
    "IF (NC IS dans_les_normes) AND (TC IS avancee) AND (PI IS grande) THEN (RC IS
faible)",
    "IF (NC IS dans_les_normes) AND (TC IS avancee) AND (PI IS moyenne) THEN (RC IS
moyen)",
    "IF (NC IS dans_les_normes) AND (TC IS avancee) AND (PI IS faible) THEN (RC IS fort)",
```

```
    "IF (NC IS dans_les_normes) AND (TC IS acceptable) AND (PI IS tres_grande) THEN (RC IS
faible)",
    "IF (NC IS dans_les_normes) AND (TC IS acceptable) AND (PI IS grande) THEN (RC IS
faible)",
    "IF (NC IS dans_les_normes) AND (TC IS acceptable) AND (PI IS moyenne) THEN (RC IS
moyen)",
    "IF (NC IS dans_les_normes) AND (TC IS acceptable) AND (PI IS faible) THEN (RC IS
fort)",

    "IF (NC IS dans_les_normes) AND (TC IS insuffisante) AND (PI IS tres_grande) THEN (RC
IS faible)",
    "IF (NC IS dans_les_normes) AND (TC IS insuffisante) AND (PI IS grande) THEN (RC IS
faible)",
    "IF (NC IS dans_les_normes) AND (TC IS insuffisante) AND (PI IS moyenne) THEN (RC IS
moyen)",
    "IF (NC IS dans_les_normes) AND (TC IS insuffisante) AND (PI IS faible) THEN (RC IS
fort)",



    "IF (NC IS hors_normes) AND (TC IS avancee) AND (PI IS tres_grande) THEN (RC IS
moyen)",
    "IF (NC IS hors_normes) AND (TC IS avancee) AND (PI IS grande) THEN (RC IS moyen)",
    "IF (NC IS hors_normes) AND (TC IS avancee) AND (PI IS moyenne) THEN (RC IS fort)",
    "IF (NC IS hors_normes) AND (TC IS avancee) AND (PI IS faible) THEN (RC IS
tres_fort)",

    "IF (NC IS hors_normes) AND (TC IS acceptable) AND (PI IS tres_grande) THEN (RC IS
moyen)",
    "IF (NC IS hors_normes) AND (TC IS acceptable) AND (PI IS grande) THEN (RC IS fort)",
    "IF (NC IS hors_normes) AND (TC IS acceptable) AND (PI IS moyenne) THEN (RC IS fort)",
    "IF (NC IS hors_normes) AND (TC IS acceptable) AND (PI IS faible) THEN (RC IS
tres_fort)",

    "IF (NC IS hors_normes) AND (TC IS insuffisante) AND (PI IS tres_grande) THEN (RC IS
moyen)",
    "IF (NC IS hors_normes) AND (TC IS insuffisante) AND (PI IS grande) THEN (RC IS
fort)",
    "IF (NC IS hors_normes) AND (TC IS insuffisante) AND (PI IS moyenne) THEN (RC IS
tres_fort)",
    "IF (NC IS hors_normes) AND (TC IS insuffisante) AND (PI IS faible) THEN (RC IS
tres_fort)"
    ])
```

Après ça, notre contrôleur flou est prêt pour de l'inférence, on va simuler son fonctionnement avec les paramètres suivants : TC = 52 ; NC = 42 ; PI = 17.

```
FS.set_variable("TC", 52)
FS.set_variable("NC", 42)
FS.set_variable("PI", 17)

rc_inference_value = FS.inference()
```

```
Pour TC = 52, NC = 42 et PI = 17, RC est égal à 23.1702970562242
```