République Algérienne Démocratique et Populaire
nistère de l'Enseignement Supérieur et de la Recherche Scientifiqu
Université des Sciences et de la Technologie Houari Boumediene

Faculté d'informatique

Module : Métaheuristique et algorithmes évolutionnaires

TP
Représentation de connaissances

Binôme

LOUNAS Katia *191931012680*

SADI OUFELLA Ouiza *191931012441*

31/05/2023
Groupe 04

Table des matières

Introduction

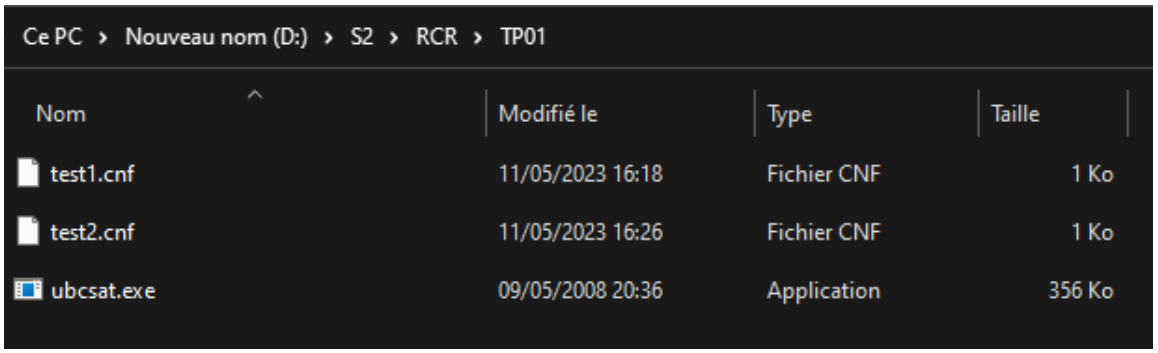
La question de la représentation des connaissances constitue un enjeu capital dans le domaine de l'intelligence artificielle, étant même considérée comme le pilier central de cette discipline. En effet, la manière dont les connaissances liées à un problème donné sont représentées peut grandement influencer la facilité ou la difficulté de sa résolution. D'une manière générale, l'utilisation de multiples modes de représentation des connaissances permet une mise en œuvre plus efficace des outils de traitement de ces connaissances lorsqu'il s'agit de résoudre des problèmes spécifiques.

L'objectif de ce document est d'explorer les différents formalismes utilisés en intelligence artificielle pour la représentation des connaissances et le raisonnement, tout en évaluant l'efficacité de ces outils.

TP01 : Logique des propositions d'ordre 1

Etape 01:

- Création du répertoire TP01 contenant le SAT UBCSAT avec les deux fichiers test1.cnf et test2.cnf



Etape 02:

- Exécution du fichier test1.cnf

L'exécution se fait à traver le cmd en effectuant cette commande:

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

Le résultat comprend deux parties, la première représente les informations de UPCAST (paramètre, version, commandes…)

Ici dans cette partie, on note le rapport du résultat de l'exécution.

```
D:\S2\RCR\TP01>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 1924115617
#  -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          7          7
```

par la suite, on trouve l'affichage du résultat , ici on remarque que le solveur a pu trouver une solution donc le fichier "test.cnf" est satisfiable pour ces valeur affichée qui représente la proposition suivante:

$$a \vee b \vee \neg c \vee \neg d \vee e$$

```
# Solution found for -target 0

 1 2 -3 -4 5


Variables = 5
Clauses = 11
TotalLiterals = 27
TotalCPUTimeElapsed = 0.001
FlipsPerSecond = 7001
RunsExecuted = 1
SuccessfulRuns = 1
PercentSuccess = 100.00
Steps_Mean = 7
Steps_CoeffVariance = 0
Steps_Median = 7
CPUTime_Mean = 0.000999927520752
CPUTime_CoeffVariance = 0
CPUTime_Median = 0.000999927520752
```

La dernière partie, c'est l'affichage de l'ensemble des statistiques , tel que le nombre de variables et de clauses, le temps d'exécution, le pourcentage de réussite ….

- Exécution du fichier test2.cnf

L'exécution s'effectue par le biais de la ligne de commande en utilisant la même commande suivante :

ubcsat -alg saps -i test2.cnf –solve

Il s'agit d'une procédure similaire à la première exécution, où la première partie fournit les informations relatives à UPCAST, telles que les paramètres, la version et les commandes utilisées.

```
D:\S2\RCR\TP01>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 1924692622
#  -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
#
```

Par la suite, le message "no solution found" apparaît, ce qui indique que le fichier nommé "test1.cnf" n'est pas satisfiable. Par conséquent, le taux de réussite obtenu est de 0%, tel qu'affiché dans la figure suivante.

```
# No Solution found for -target 0


Variables = 5
Clauses = 13
TotalLiterals = 29
TotalCPUTimeElapsed = 0.007
FlipsPerSecond = 14285777
RunsExecuted = 1
SuccessfulRuns = 0
PercentSuccess = 0.00
Steps_Mean = 100000
Steps_CoeffVariance = 0
Steps_Median = 100000
CPUTime_Mean = 0.00699996948242
CPUTime_CoeffVariance = 0
CPUTime_Median = 0.00699996948242
```

Etape 03:

- Traduction de la base de connaissances relative aux connaissances zoologiques (céphalopodes) sous forme CNF

1 ère traduction : le terme «généralement» est ignoré

1- (Na ⊃ CEa) ; (Nb ⊃ CEb) ; (Nc ⊃ CEc);
3- (Ma ⊃ COa); (Mb ⊃ COb) ; (Mc ⊃ COc);
4- (CEa ⊃ ¬ COa) ; (CEb ⊃ ¬COb) ; (CEc ⊃ ¬COc);
2- (CEa ⊃ Ma); (CEa ⊃ Mb) ; (CEa ⊃ Mc);
5- (Na ⊃ COa); (Nb ⊃ COb); (Nc ⊃ COc);
6- Na ; CEb ; Mc

1- (¬Na ∨ CEa) ; (¬Nb ∨ CEb) ; (¬Nc ∨ CEc)
3- (¬Ma ∨ COa) ; (¬Mb ∨ COb) ; (¬Mc ∨ COc)
4- (¬CEa ∨ ¬ COa) ; (¬CEb ∨ ¬COb) ; (¬CEc ∨ ¬COc)
2- (¬CEa ∨ Ma) ; (¬CEa ∨ Ma) ; (¬CEa ∨ Ma)
5- (¬Na ∨ COa) ; (¬Nb ∨ COb) ; (¬Nc ∨ COc)
6- Na; CEb; Mc

-FNC:

soit les numéros de variables comme suit:

Na=1,	CEa=2,	Nb=3,	Nc=4
CEb=5,	CEc=6,	Ma=7,	COa=8
Mb=9,	COb=10,	Mc=11,	COc=12

-Fichier cnf :

On tout on obtient 12 variables et 18 clause

Etape03.cnf

```
p cnf 12 18
-1 2 0
-3 5 0
-4 6 0
-7 8 0
-9 10 0
-11 12 0
-2 -8 0
-2 -7 0
-6 -12 0
-2 7 0
-2 9 0
-2 11 0
-1 8 0
-3 10 0
-4 12 0
1 0
5 0
11 0
```

-Résultat :

Le solveur a abouti à la conclusion qu'aucune solution n'est possible pour l'ensemble des formules, ce qui démontre leur non-satisfiabilité.

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


Variables = 12
Clauses = 18
TotalLiterals = 33
TotalCPUTimeElapsed = 0.015
FlipsPerSecond = 6666620
RunsExecuted = 1
SuccessfulRuns = 0
PercentSuccess = 0.00
Steps_Mean = 100000
Steps_CoeffVariance = 0
Steps_Median = 100000
CPUTime_Mean = 0.0150001049042
CPUTime_CoeffVariance = 0
CPUTime_Median = 0.0150001049042
```

2 ème traduction :

1- ((Céa Na) Coa);
2- ((Céb Nb) Cob);
3- ((Céc Nc) Coc);
4- ((Ma (Céa Na)) Coa);
5- ((Mb (Céb Nb)) Cob);
6- ((Mc (Céc Nc)) Coc).

-FNC:

1-(¬Na ∨ Cea); (¬Nb ∨ Ceb); (¬Nc ∨ Cec)
2-(¬Cea ∨ M a); (¬Ceb ∨ M b); (¬Cec ∨ M c)
3-(¬N a ∨ Coa); (¬N b ∨ Cob); (¬N c ∨ Coc)
4-(¬M a ∨ Cea ∨ Coa); (¬M a ∨ ¬N a ∨ Coa)
5-(¬M b ∨ Ceb ∨ Cob); (¬M b ∨ ¬N b ∨ Cob)
6-(¬M c ∨ Cec ∨ Coc); (¬M c ∨ ¬N c ∨ Coc)
7-(¬Cea ∨ Na ∨ ¬Coa);(¬Ceb ∨ Nb ∨ ¬Cob) 8-(¬Cec ∨ Nc ∨ ¬Coc)

soit les numéros de variables comme suit:

Cea=1	Na=2	Coa=3	Ceb=4
Nb=5	Cob=6	Cec=7	Nc=8
Coc=9	Ma=10	Mb=11	Mc=12

-Fichier cnf :

On tout on obtient 12 variables et 21 clause

Etape3.cnf

```
p cnf 12 18
-2 1 0
-5 1 0
-8 7 0
-1 10 0
-4 11 0
-7 12 0
-2 3 0
-5 6 0
-8 9 0
-10 1 6 0
-10 2 3 0
-11 4 6 0
-11 5 6 0
-12 7 9 0
-12 8 9 0
-1 2 -3 0
-4 5 -6 0
-7 8 -9 0
2 0
4 0
12 0
```

-Résultat :

Le solveur a abouti à une solution pour toutes les formules, ce qui démontre leur satisfiabilité.

```
# 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 6 -7 -8 9 -10
 11 12


Variables = 12
Clauses = 18
TotalLiterals = 45
TotalCPUTimeElapsed = 0.001
FlipsPerSecond = 3000
RunsExecuted = 1
SuccessfulRuns = 1
PercentSuccess = 100.00
Steps_Mean = 3
Steps_CoeffVariance = 0
Steps_Median = 3
CPUTime_Mean = 0.000999927520752
CPUTime_CoeffVariance = 0
CPUTime_Median = 0.000999927520752
```

Test des benchmarks:

On a testé avec le premier fichier “uf20-01.cnf” on a obtenu une solution ce qui signifie que le solveur a su résoudre le système, donc le fichier est satisfiable.

```
D:\S2\RCR\TP01>ubcsat -alg saps -i uf20-02.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 1930962369
#  -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
#
```

voici l’affichage de la solution.

```
# 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


Variables = 20
Clauses = 91
TotalLiterals = 273
TotalCPUTimeElapsed = 0.000
FlipsPerSecond = 1
RunsExecuted = 1
SuccessfulRuns = 1
PercentSuccess = 100.00
Steps_Mean = 7
Steps_CoeffVariance = 0
Steps_Median = 7
CPUTime_Mean = 0
CPUTime_CoeffVariance = 0
CPUTime_Median = 0
```

Puis on a testé avec le fichier “uf20-02.cnf”, en revanche ici le solveur UBCAST n’a pas trouvé de solution, par conséquent le fichier n’est pas satisfiable.

```
D:\S2\RCR\TP01>ubcsat -alg saps -i uf20-02.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 2007597542
#  -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        140     100000
# No Solution found for -target 0


Variables = 75
Clauses = 325
TotalLiterals = 975
TotalCPUTimeElapsed = 0.039
FlipsPerSecond = 2564100
RunsExecuted = 1
SuccessfulRuns = 0
PercentSuccess = 0.00
Steps_Mean = 100000
Steps_CoeffVariance = 0
Steps_Median = 100000
CPUTime_Mean = 0.0390000343323
CPUTime_CoeffVariance = 0
CPUTime_Median = 0.0390000343323
```

Etape 04:

Dans cette étape nous avons implémenté un algorithme en Python pour simuler l'inférence d'une base de connaissances en utilisant le solveur SAT et le raisonnement par l'absurde.

Cet algorithme utilise la bibliothèque **pysat** pour le solveur SAT (ici, **Glucose3**). La base de connaissances est représentée sous forme de liste de clauses CNF, où chaque clause est une liste de littéraux. Le littéral phi est également passé en paramètre.

L'algorithme crée une instance du solveur SAT et ajoute les clauses de la base de connaissances ainsi que la négation de phi à la formule. Ensuite, il vérifie la satisfiabilité de la formule à l'aide de la méthode solve() du solveur. Si la formule est satisfiable, cela signifie que la base de connaissances n'infère pas phi. Sinon, la base de connaissances infère phi.

-Code :

```
from pysat.solvers import Glucose3

def inference_BC(BC, phi):
    solver = Glucose3()  # Créer un solveur SAT

    # Ajouter BC à la base de connaissances
    for clause in BC:
        solver.add_clause(clause)

    # Ajouter la négation de phi au solveur SAT
    solver.add_clause([-phi])

    # Vérifier la satisfiabilité de la base de connaissances étendue
    if solver.solve():
        # BC n'infère pas phi
        return False
    else:
        # BC infère phi
        return True

# Exemple d'utilisation
BC = [[1, -2], [-2, -3], [3]]  # Exemple de base de connaissances (BC)
phi = -3  # Exemple de formule (phi)

result = inference_BC(BC, -phi)  # Négation de phi
if result:
    print("BC infère phi.")
else:
    print("BC n'infère pas phi.")
```

-Résultat :

```
PS D:\S2\RCR\TP01> & C:/Users/HP.LAPTOP-ESRMF9MQ/AppData/Local/Programs/Python/Python310/python.exe d:/S2/RCR/TP01/Etape.py
la base de fait est la suivante :
[[1, -2], [-2, -3], [3]]
le litéral est:
-3
BC infère phi.
```

Conclusion

A travers ce TP 01 on conclut que l'utilisation d'**UBCSAT** pour vérifier la satisfiabilité des formules de premier ordre présente plusieurs avantages. UBCSAT est un solveur SAT efficace et largement utilisé, qui offre une variété d'algorithmes et de paramètres pour trouver des solutions satisfaisantes. Il permet de traiter des formules de premier ordre complexes et de les résoudre de manière efficace. De plus, **UBCSAT** fournit des informations détaillées sur le processus d'exécution, telles que les statistiques, les résultats et les temps d'exécution, ce qui facilite l'analyse et l'interprétation des résultats. En utilisant UBCSAT, il est possible de déterminer rapidement si une formule de premier ordre est satisfiable ou non, ce qui est essentiel dans de nombreux domaines tels que la vérification formelle, l'intelligence artificielle et la recherche opérationnelle.

TP02 : Logique des propositions d'ordre 1

Installation

Pour installer la bibliothèque Tweety Projet avec Maven, vous devez ajouter la dépendance correspondante à votre fichier pom.xml. Voici comment procéder :

1- Ouvert le fichier pom.xml dans notre projet Maven.
2- Ajout de la dépendance Tweety Project à la section <dependencies> du fichier pom.xml :

```
<dependencies>
   <!-- Autres dépendances -->
   <dependency>
      <groupId>de.tweetyproject</groupId>
      <artifactId>tweety</artifactId>
      <version>1.6.0</version>
   </dependency>
</dependencies>
```

3- Enregistrement du fichier pom.xml. Maven télécharge automatiquement la bibliothèque Tweety Project lors de la compilation du projet.

4- Maintenant utiliser Tweety Project dans notre code Java en important les classes nécessaires.

Exploration de la librairie

Activation de la logique du premier ordre :

Une instance de FolSignature est créée pour activer la logique du premier ordre. La logique du premier ordre est activée en passant true comme argument au constructeur de FolSignature.

```
FolSignature signature = new FolSignature(true);
```

Création des types et des constantes :

Les types sportifSort et sportSort sont créés à l'aide de la classe Sort pour représenter les types "sportif" et "sport" respectivement.
Des constantes telles que katia, lyse et C sont créées à l'aide de la classe Constant pour représenter des individus dans le domaine.

```
//Creation des types
        Sort sportifSort = new Sort("sportif");
        Sort sportSort = new Sort("sport");


//Creation des constantes
        Constant lyse = new Constant("lyse", Sort.ANY);
        Constant ballet = new Constant("ballet", Sort.ANY);
        signature.add(lyse, ballet);
```

Création des prédicats :

Les prédicats sportif, sport et pratique sont créés à l'aide de la classe Predicate pour représenter les relations entre les individus.
Les listes de types sont utilisées pour spécifier les arguments des prédicats.
Les prédicats sont ajoutés à la signature en utilisant la méthode add de FolSignature.

```
//le predicat unaire sportif
        List<Sort> predicateList1 = new ArrayList<Sort>();
        predicateList1.add(sportifSort);
        Predicate sportif = new Predicate("sportif", predicateList1);

//le predicat unaire sport
        List<Sort> predicateList2 = new ArrayList<Sort>();
        predicateList2.add(sportSort);
        Predicate sport = new Predicate("sport", predicateList2);

 //le predicat binaire pratique
        List<Sort> predicateList3 = new ArrayList<Sort>();
        predicateList3.add(sportifSort);
        predicateList3.add(sportSort);
        Predicate pratique = new Predicate("pratique", predicateList3);


        signature.add(sport, sportif, pratique);
        System.out.println("les predicats sont : \n" +signature);
```

Création des propositions :

Un objet FolParser est créé pour analyser les formules dans la logique du premier ordre.
La signature est configurée pour le FolParser en utilisant la méthode setSignature.
Un objet FolBeliefSet appelé BSet est créé pour contenir les formules.

```
//Creation des propositions
      FolParser parser = new FolParser();
      parser.setSignature(signature);
      FolBeliefSet BSet = new FolBeliefSet();
```

Création de formules :

Des exemples de formules sont créées à l'aide de la méthode parseFormula de FolParser.
Les formules créées représentent des assertions logiques dans la logique du premier ordre.
Les formules sont ajoutées à BSet à l'aide de la méthode add.
Les formules sont affichées à l'écran à l'aide de System.out.println.

```
 // Exemple de création de formules
FolFormula formula1 = (FolFormula) parser.parseFormula("forall Y:( sportif(Y) &&
exists X: ( pratique(X,Y) => (sport(X) ) ))");
 FolFormula formula2 = (FolFormula) parser.parseFormula("pratique(ballet, lyse)");
  FolFormula formula3 = (FolFormula)parser.parseFormula("sportif(lyse)");
```

Évaluation des formules :

Un raisonneur (reasoner) est configuré en utilisant la classe SimpleFolReasoner.
Une instance de FolBeliefSet appelée BSet1 est créée pour contenir les formules à évaluer.
Des formules supplémentaires sont créées et ajoutées à BSet1.
La formule à inférer est évaluée en utilisant la méthode query du raisonneur.

```
// Vérifier si la formule est inférée
      isEntailed = prover.query(BSet1, formula4);
```

Exemple de connaissance

Prédicats :

- Le prédicat unaire "sportif" est défini avec le sort "sportifSort" et représente une caractéristique des individus qui sont des sportifs.
- Le prédicat unaire "sport" est défini avec le sort "sportSort" et représente un sport.
- Le prédicat binaire "pratique" est défini avec les sorts "sportifSort" et "sportSort" et représente la relation entre un individu sportif et un sport qu'il pratique.

Formules :

- "formula1": pratique(ballet,lyse)
- "formula2" sportif(lyse)
- "formula3": ∀ (x)∀ (y) pratique(x,y) ⊃ sportif(y) ∧ sport(x)

A inférer :

- sport(ballet)

Résultats

```
Formule 1 : forall Y: (sportif(Y)&&exists X: ((pratique(X,Y)=>sport(X))))
Formule 2 : pratique(ballet,lyse)
Formule 3 : sportif(lyse)
Propositions :
forall Y: (sportif(Y)&&exists X: ((pratique(X,Y)=>sport(X))))
pratique(ballet,lyse)
sportif(lyse)
------------------------------
Formule a infrer : sport(ballet)
La formule est inferee.
```

TP03 : Logique Modale

Premier exemple :

Etape 01:

Enoncé de l'exercice :

Exercice 2:

Soit M le modèle modal défini par <W, R, V> avec

$W = \{w_1, w_2, w_3, w_4, w_5, w_6\}$

R est telle que $w_1 R w_2$, $w_1 R w_3$, $w_3 R w_3$, $w_3 R w_4$, $w_3 R w_5$, $w_3 R w_6$, $w_4 R w_4$, $w_4 R w_5$ $w_6 R w_5$

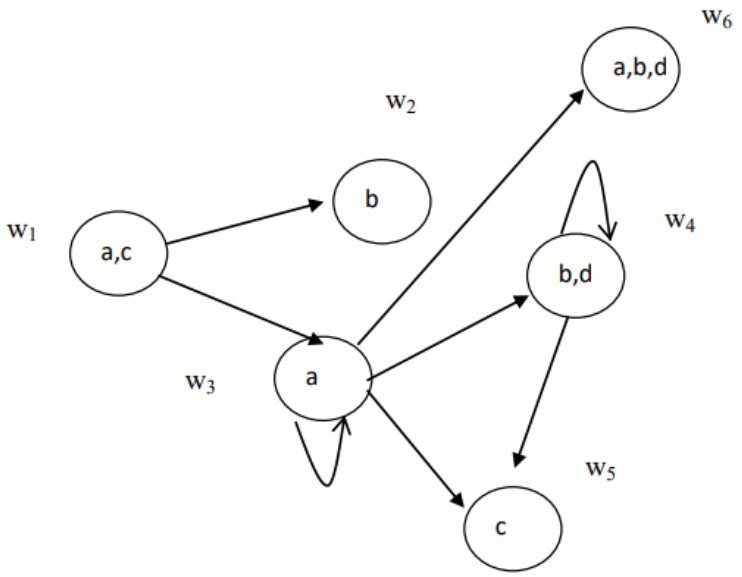
V est telle que $V(a) = \{w_1, w_3, w_6\}$, $V(b) = \{w_2, w_4, w_6\}$, $V(c) = \{w_1, w_5\}$, $V(d) = \{w_4, w_6\}$.

1- Représentez ce modèle modal.
2- Quelle est la spécificité de ce modèle ?
3- Spécifiez, en justifiant, les assertions vraies dans ce modèle modal avec la spécificité que $M,x \models \neg B$ ssi non $(M,x \models B)$.
   a- $M,w_1 \models \Box(\Diamond a \vee \neg b)$
   b- $M,w_2 \models \neg(\Diamond((c \supset \neg b) \vee \Diamond a))$
   c- $M,w_3 \models \Diamond\Diamond(a \supset \neg c)$
   d- $M,w_4 \models \Box(c \vee d)$

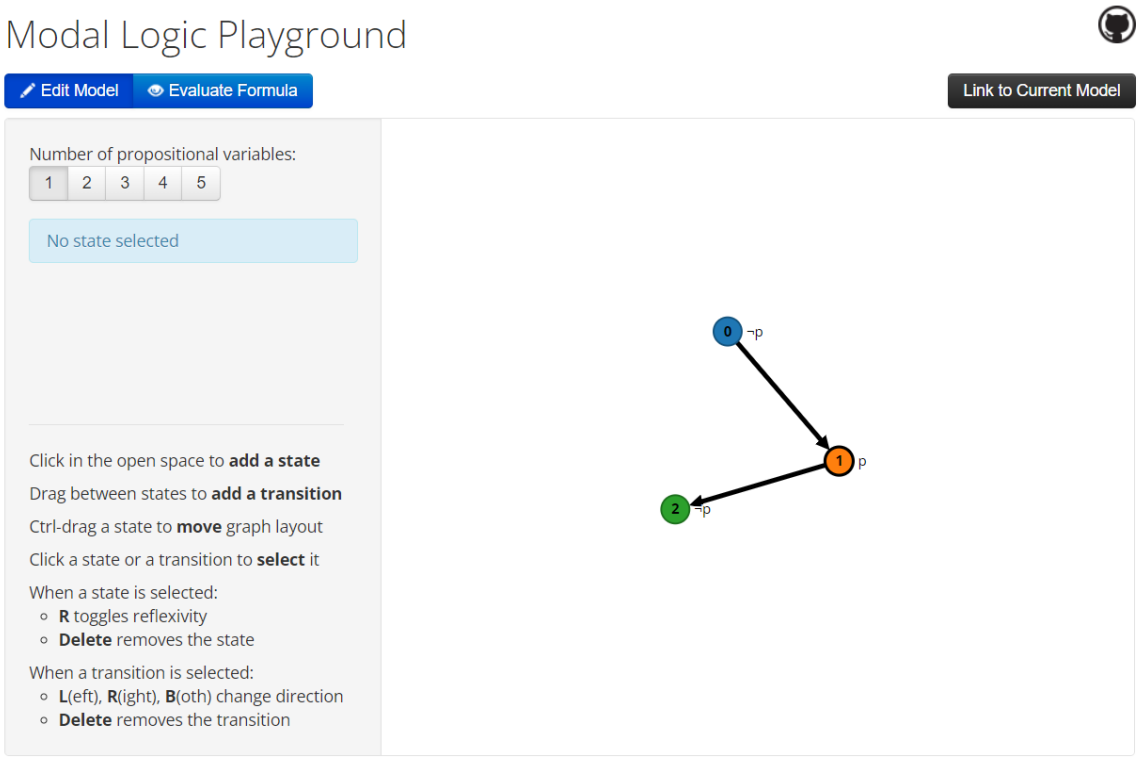
Représentation du modèle modal :

Exercice 2 :

1- Représentez ce modèle modal.

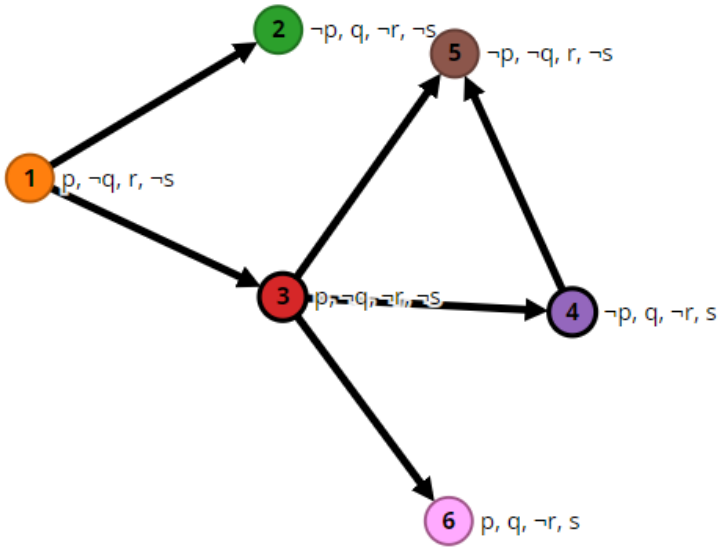


L’interface de Modal Logic Playground :



Représentation du modèle modal sur l’interface :

Pour cet exemple on prend : p=a, q=b, r=c et s=d.

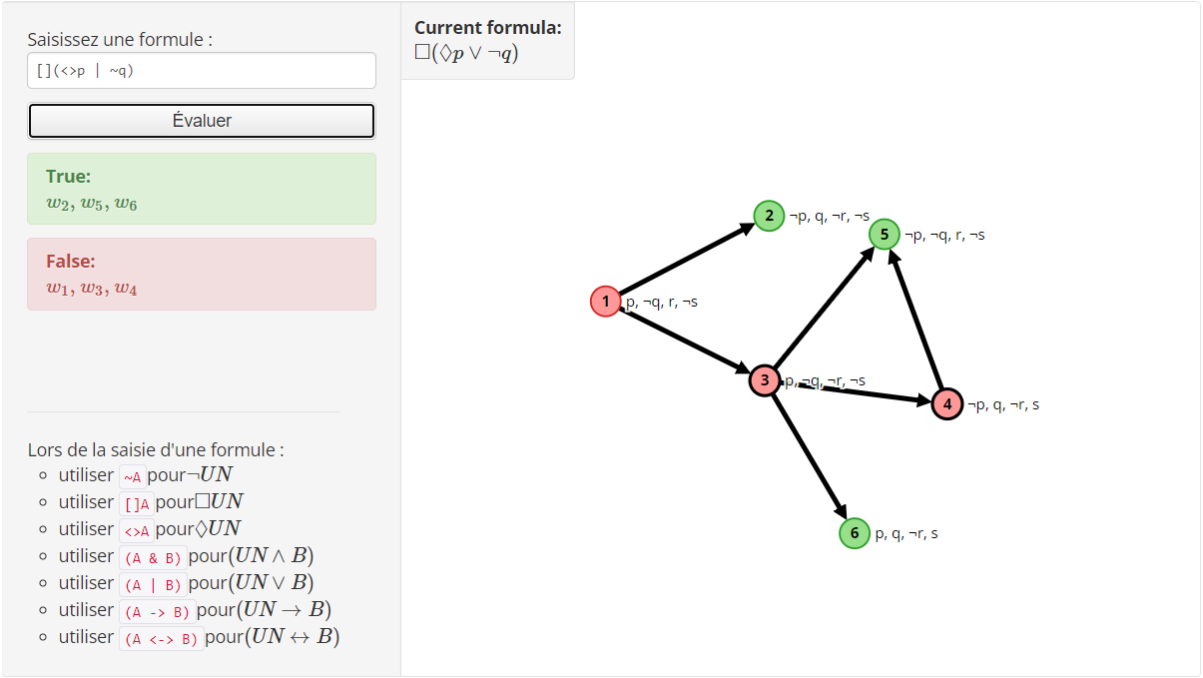


Etape 02:

Evaluation des formules :

- **Formule 01 :** M,w1 |== □(◇a ∨ ¬ b)

  Résultat : M,w1 |== □(◇p ∨ ¬ q)

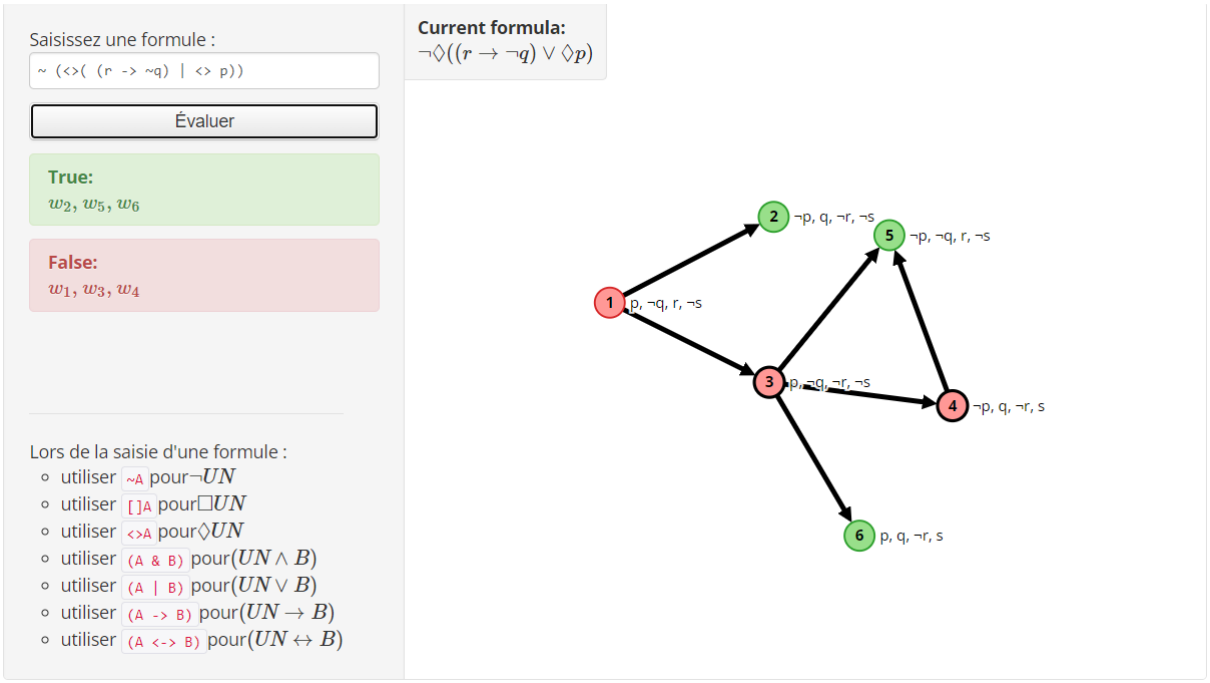


Explication : On a les mondes accessibles par w1 : w2 et w3.

La formule (◇p ∨ ¬ q) est fausse en w3 et w1 R w3 donc □(◇p ∨ ¬ q) est fausse en w1.

- **Formule 02 :** M,w2 |== ¬ (◇((c ⊃ ¬ b) ∨ ◇a))

  Résultat : M,w2 |== ¬ (◇((r ⊃ ¬ q) ∨ ◇p))

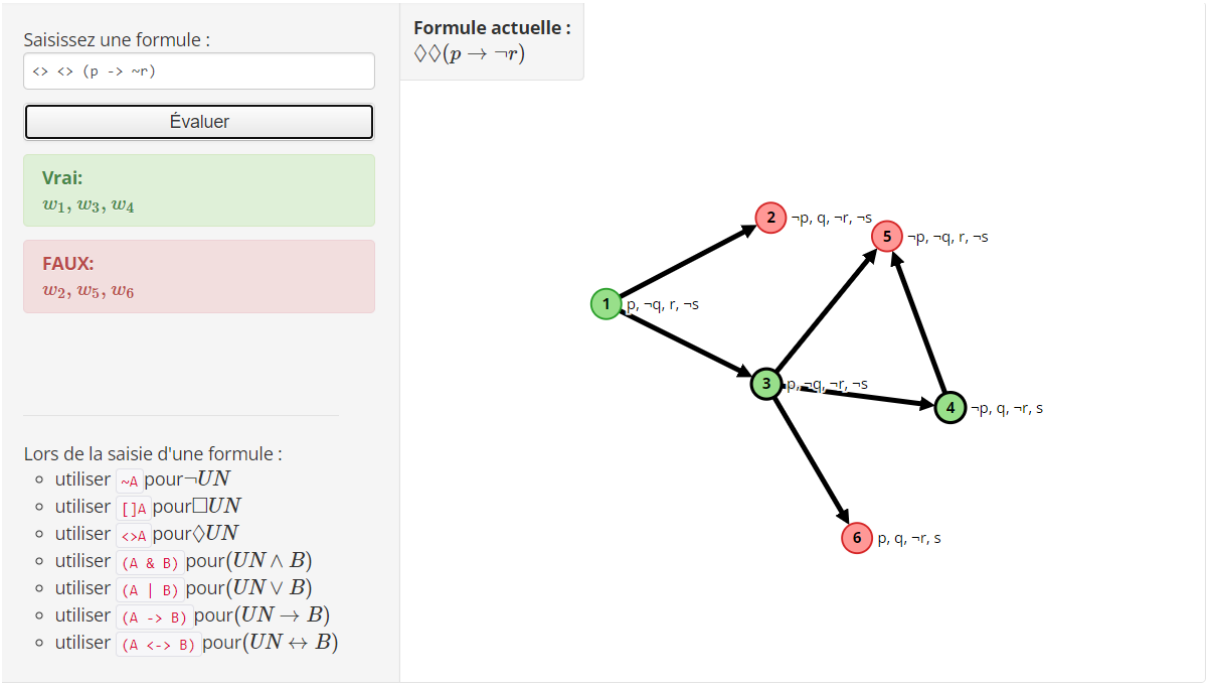


Explication : On a les mondes accessibles par w2 : aucun monde.

La formule est vraie en w2 car ◇((c ⊃ ┐b) ∨ ◇a)) est faux en w2 puisqu'il n'y a aucun monde accessible.

- **Formule 03 :** M,w3 |== ◇◇(a ⊃ ┐c)

  Résultat : M,w3 |== ◇◇(p ⊃ ┐r)

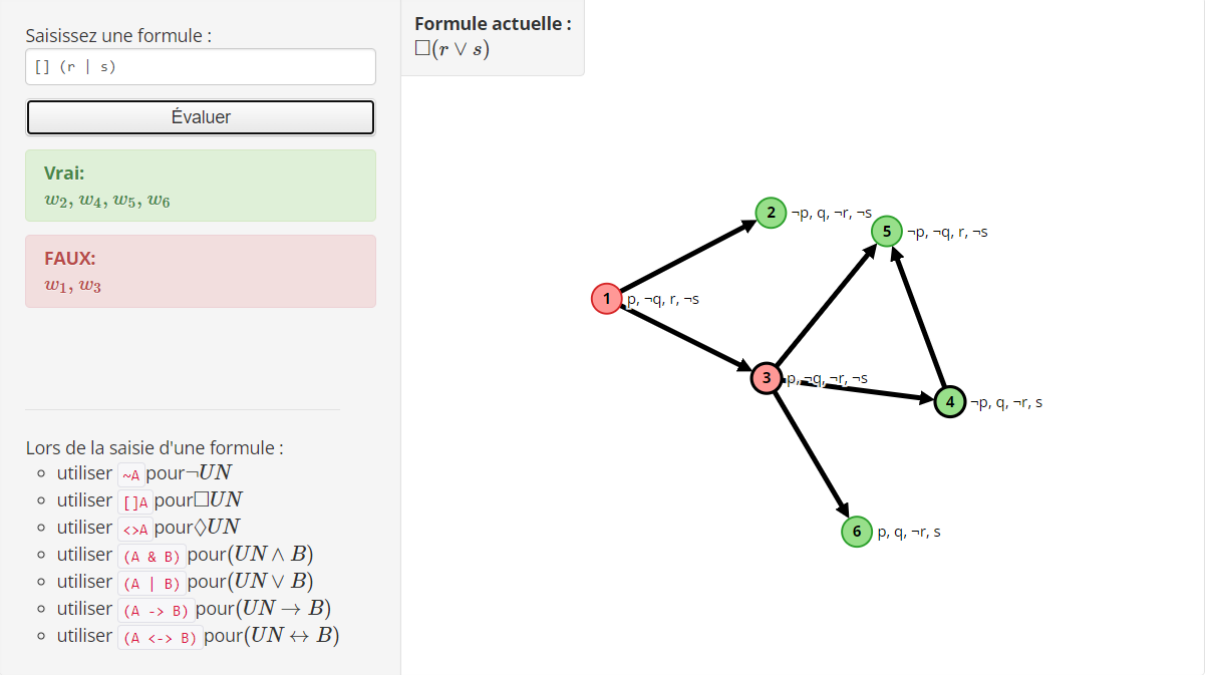


Explication : On a les mondes accessibles par w3 :w4,w5,w6.

La formule ◇◇(p ⊃ ┐r) est vraie en w3 (w3 R w5) et (p ⊃ ┐r) est vrai en w5 et ◇(p ⊃ ┐r) est vrai en w4 (w4 R w5).

Formule 04 : M,w4 |== □ (c ∨ d)

Résultat : M,w4 |== □ (r ∨ s)



Explication : On a les mondes accessibles par w4 : w4,w5.

La formule (r ∨ s) est vraie en w4 et en w5 donc □ (r ∨ s) est vraie en w4.

Deuxième exemple :

Etape 01:

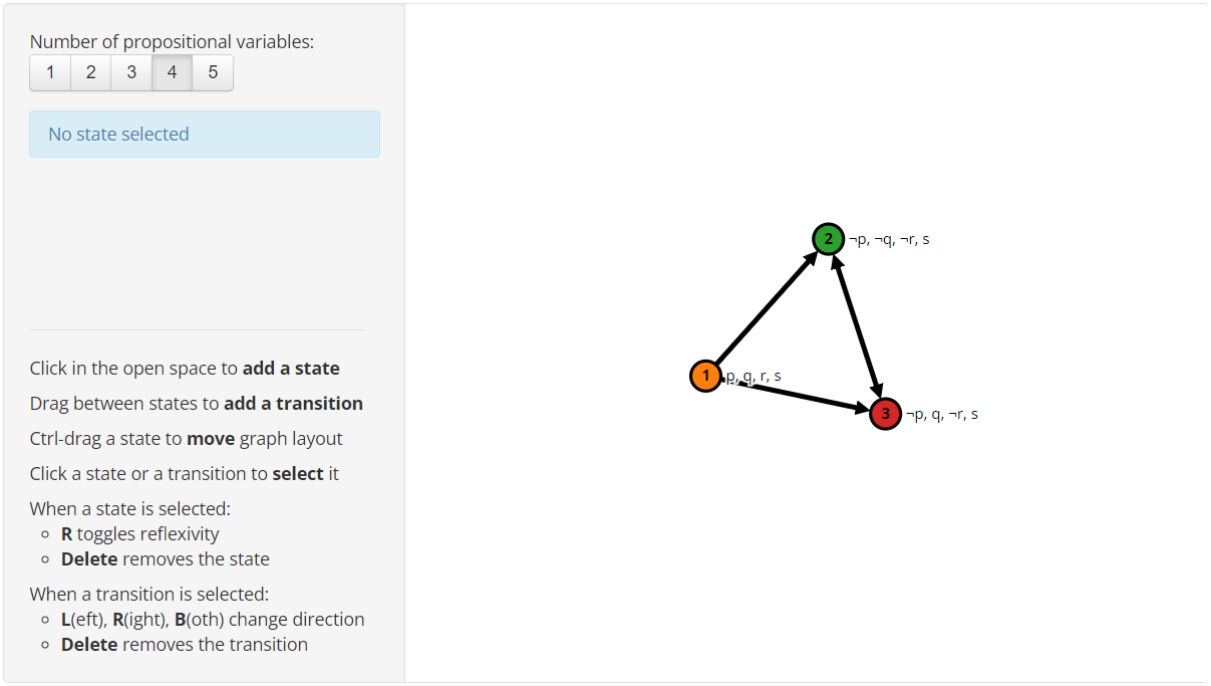
Enoncé de l'exercice :

Pour prendre en compte trois mondes possibles dans notre modèle modal sur les planètes vitales et non vitales, ainsi que sur la présence d'eau, d'oxygène et de gravité :
Ensemble des mondes possibles (W) : {w1, w2, w3}

Relation d'accessibilité (R) : w1 R w1,w1 R w2, w1 R w3 ,w2 R w2,w2 R w3,w3 R w3, w3 R w2
Fonction de valuation (V) : V(vitale) = {w1}, V(eau) = {w1, w3} ,V(oxygène) = {w1} ,V(gravite) = {w1, w2}.
w1 = terre
w2 = Jupiter , Saturne
w3 = Terre , Mars

Représentation du modèle modal sur l'interface :

Pour cet exemple on prend : p=vitale, q=eau, r=oxygène et s=gravité.

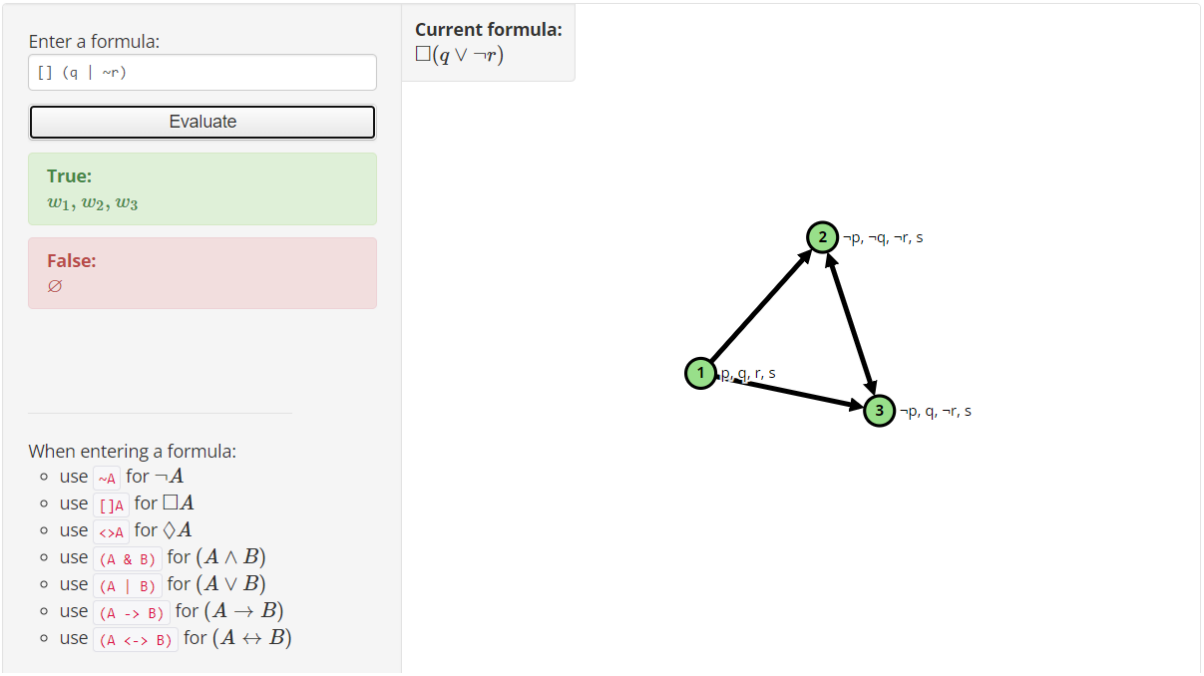


Etape 02:

Evaluation des formules :

- **Formule 01 :** M,w1 |== □(q ∨ ┐r)

Résultat :

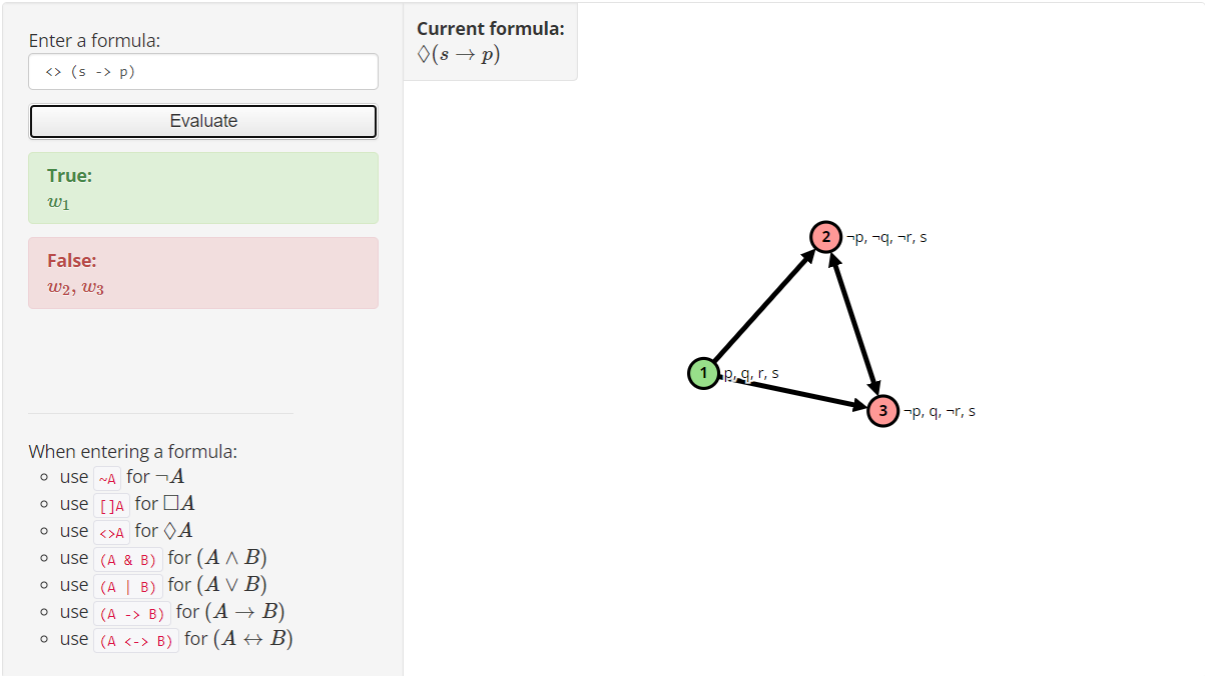


Explication : On a les mondes accessibles par w1 :w2,w3,w1.

La formule (q ∨ ┐r) est vraie en w2,w3 donc □(q ∨ ┐r) est vrai en w1 .

- **Formule 02 :** M,w2 |== ◇(s ⊃ p)

Résultat :

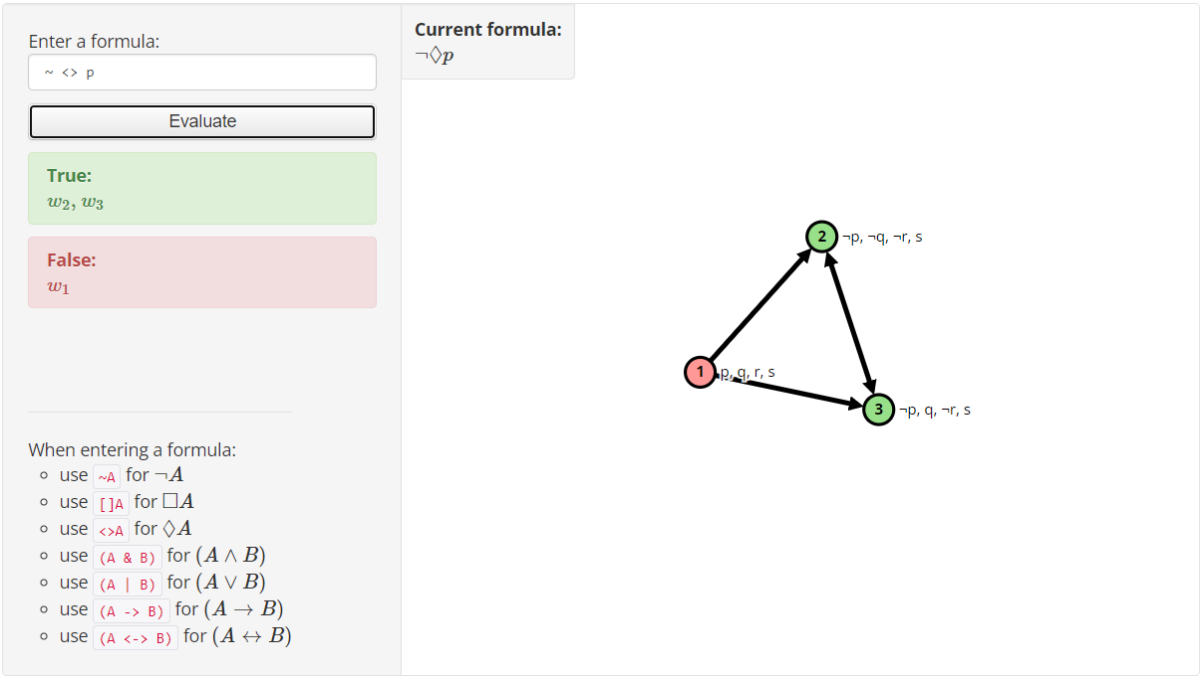


Explication : On a les mondes accessibles par w2 :w2,w3.

La formule (s ⊃ p) est fausse en w2,w3 donc ◇(s ⊃ p) est fausse en w2 .

- **Formule 03 :** M,w3 |== ¬(◇p)

  Résultat :



Explication : On a les mondes accessibles par w3 :w2,w3.

La formule (p) est fausse en w2,w3 donc (◇p) est fausse en w3 , ce qui nous donne : ¬(◇p) Vraie .

Etape 03:

La librairie Java tweety : exemple 2.

Afin d'utiliser la bibliothèque Java tweety Modal Logic, il est nécessaire d'avoir un modèle logique contenant une base à partir de laquelle nous pouvons faire des déductions pour déterminer la véracité ou la fausseté des formules.
Nous allons utiliser les formules déduites vraies dans l'exemple 2 et étudier la véracité des autres formules avec des raisonneurs.
Nous avons donc : { Formules 1 ; □(q ¬r) }

Code source :

```
import java.io.IOException;
import org.tweetyproject.commons.ParserException;
import org.tweetyproject.logics.commons.syntax.Predicate;
import org.tweetyproject.logics.commons.syntax.RelationalFormula;
import org.tweetyproject.logics.fol.syntax.FolFormula;
import org.tweetyproject.logics.fol.syntax.FolSignature;
import org.tweetyproject.logics.ml.parser.MlParser;
import org.tweetyproject.logics.ml.reasoner.AbstractMlReasoner;
import org.tweetyproject.logics.ml.reasoner.MleanCoPReasoner;
import org.tweetyproject.logics.ml.reasoner.SPASSMlReasoner;
import org.tweetyproject.logics.ml.reasoner.SimpleMlReasoner;
import org.tweetyproject.logics.ml.syntax.MlBeliefSet;


public class TP03 {

  public static void main(String[] args) throws ParserException, IOException {
    // Create a new modal belief set
    MlBeliefSet base = new MlBeliefSet();

    // Create a parser to parse formulas
    MlParser parser = new MlParser();

    // Define the signature with predicates
    FolSignature signature = new FolSignature();
    signature.add(new Predicate("p", 0));
    signature.add(new Predicate("q", 0));

    parser.setSignature(signature);

    //formule de l'ensemble
    base.add((RelationalFormula) parser.parseFormula("[](!q || p)"));
    base.add((RelationalFormula) parser.parseFormula("<>(p || q)"));
    base.add((RelationalFormula) parser.parseFormula("<>(p)"));


    //  SimpleMlReasoner
    SimpleMlReasoner reasoner = new SimpleMlReasoner();

    //valeur de verite
    boolean PTrue = reasoner.query(base, (FolFormula) parser.parseFormula("<>(p)"));
    boolean NotQOrPTrue = reasoner.query(base, (FolFormula) parser.parseFormula("[](!q || p)"));
    boolean NotPTrue = reasoner.query(base, (FolFormula) parser.parseFormula("[](!p)"));

    // Print the results
    System.out.println("ensemble : " + base);
    System.out.println("[](!q || p) is " + NotQOrPTrue);
    System.out.println("[](!p) is " + NotPTrue);
  }

}
```

Résultat :

```
"C:\Program Files\Java\jdk-16.0.1\bin\java.exe" ...
ensemble : { <>(p||q), <>(p), [](!q||p) }
[](!q || p) is true
[](!p) is false

Process finished with exit code 0
```

TP04 : Logique des défauts

Dans ce TP, nous allons implémenter des exemples de la logique des défauts en utilisant la toolbox “deafultlogic” qui est développée par Evan Morrison en java.

Evan Morrison a proposé un formalisme appelé "defaultlogic" (ou "logique par défaut") pour représenter et raisonner sur de telles connaissances.

C’est une approche formelle pour modéliser la logique des défauts en utilisant des règles par défaut et des règles de réfutation. Il fournit un cadre de raisonnement pour inférer des conclusions à partir de connaissances incertaines ou incomplètes, en tenant compte des exceptions ou des réfutations potentielles.

Exploration de l’outil

Ici on détaillera la syntaxe sur l’un des exemples qu’on a implémentés.

Définition du monde initial :

- La variable **myWorld** est créée en tant qu'ensemble de mondes.
- Une formule est ajoutée au monde initial : "((Symptom1) & (Symptom1 -> Disease))". Cette formule indique que le symptôme 1 est présent dans le monde initial, et que s'il y a le symptôme 1, cela implique la présence de la maladie.

```
WorldSet myWorld = new WorldSet();
    myWorld.addFormula("((Symptom1) & (Symptom1 -> Disease))");
```

Définition des règles de défauts :

- Deux objets **DefaultRule** sont créés pour représenter les règles de défauts.
- Pour la règle 1, le symptôme 1 est défini comme prérequis, la maladie comme justification et la non-présence du symptôme 1 comme conséquence.
- Pour la règle 2, la maladie est définie comme prérequis, le traitement comme justification et la non-présence de la maladie comme conséquence.

- Les règles sont ajoutées à l'ensemble de règles **myRules**.

```
// Définition des règles de défauts
    DefaultRule rule1 = new DefaultRule();
    rule1.setPrerequisite("Symptom1");
    rule1.setJustificatoin("Disease");
    rule1.setConsequence("~Symptom1");

    DefaultRule rule2 = new DefaultRule();
    rule2.setPrerequisite("Disease");
    rule2.setJustificatoin("Treatment");
    rule2.setConsequence("~Disease");

// Ajout des règles à l'ensemble myRules
    RuleSet myRules = new RuleSet();
    myRules.addRule(rule1);
    myRules.addRule(rule2);
```

Création du moteur de raisonnement et obtention des scénarios possibles :

- Un objet **DefaultReasoner** est créé en utilisant le monde initial myWorld et l'ensemble de règles **myRules**.
- La méthode **getPossibleScenarios()** est appelée pour obtenir les scénarios possibles (ensembles de mondes) qui satisfont les règles de défauts.

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

Affichage des résultats :

- Les informations sur le monde initial et les règles sont affichées.
- Les extensions possibles (scénarios) sont affichées.
- Pour chaque extension, le monde initial étendu avec l'extension est affiché en utilisant la notation "**Th(W U (extension))**".
- La fermeture de chaque monde étendu est affichée en utilisant la méthode **getClosure()** de l'objet WFF.

```
    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();

     }
```

Résultats

Test des exemple de la classe DefaultLogic:

Voici le résultat du premier exemple, ici on pense que x est un oiseau et que x peut voler sachant que x est un penguin.

```
------------------------------------------------------------
Trying  bird_x & (penguin_x -> ~flies_x) & penguin_x
Trying  bird_x & (penguin_x -> ~flies_x) & penguin_x
Given the world:
    bird_x & (penguin_x -> ~flies_x) & penguin_x
 And the rules
    [(bird_x):(flies_x) ==> (flies_x)] , [(bird_x):(penguin_x) ==> (~flies_x)]
Possible Extensions
     Ext: Th(W U (~flies_x))
    =  (~flies_x | ~penguin_x) & penguin_x & ~flies_x & bird_x
------------------------------------------------------------
```

Ici l'exemple est plutôt théorique, dans le but de montrer le bon fonctionnement du raisonnement en logique des défauts.

```
Given the world:

 And the rules
    [([]):(A & ~B) ==> (A)] , [([]):(~A & B) ==> (~A)]
Possible Extensions
     Ext: Th(W U (A))
    =  A
     Ext: Th(W U (~A))
    =  ~A
------------------------------------------------------------
Execution time was 1055 ms.
```

Exemple de connaissance

Enoncé

Kamel est un agriculteur
Généralement tous les agriculteur utilise des pesticides
Un jour on a su que Kamel est agriculteur Bio donc Kamel n'utilise pas des pesticides

Formulation

$$\frac{\text{Agriculteur : non Agriculteur\_Bio}}{\text{utilise\_pesticides}}$$

Résultats

```
Trying  Kamel_est_agriculteur
Given the world:
    Kamel_est_agriculteur
 And the rules
    [(Kamel_est_agriculteur):(utilise_pesticides) ==> (utilise_pesticides)]
Possible Extensions
     Ext: Th(W U (utilise_pesticides))
    =  utilise_pesticides & Kamel_est_agriculteur
We one day learn that Kamel_est_agriculteur_bio
Trying  Kamel_est_agriculteur & Kamel_est_agriculteur_bio & (Kamel_est_agriculteur-> ~utilise_pesticides)
Trying  Kamel_est_agriculteur & Kamel_est_agriculteur_bio & (Kamel_est_agriculteur-> ~utilise_pesticides)
Given the world:
    Kamel_est_agriculteur & Kamel_est_agriculteur_bio & (Kamel_est_agriculteur-> ~utilise_pesticides)
 And the rules
    [(Kamel_est_agriculteur):(utilise_pesticides) ==> (utilise_pesticides)] , [(Kamel_est_agriculteur):(Kamel_est_agriculteur_bi
Possible Extensions
     Ext: Th(W U (~utilise_pesticides))
    = Kamel_est_agriculteur_bio & ~utilise_pesticides &  (~utilise_pesticides | ~Kamel_est_agriculteur) & Kamel_est_agriculteur
```

TP05 : Les réseaux sémantiques

Il convient de souligner que les connaissances ne sont pas toujours exprimées à travers des mots et des phrases. Les systèmes informatiques peuvent également utiliser des schémas et des dessins, autrement dit, des représentations graphiques, parmi lesquelles nous retrouvons les réseaux sémantiques. Ces derniers offrent un moyen de représenter les relations entre différentes entités de manière visuelle, ce qui facilite l'organisation et la compréhension des connaissances complexes.
De nos jours, les réseaux sémantiques sont reconnus non seulement pour leur capacité à représenter des informations, mais aussi pour leur utilité graphique dans la représentation de données. Ils offrent également des algorithmes efficaces pour inférer des propriétés d'un objet en se basant sur son appartenance à une catégorie, tels que l'héritage, la propagation, et bien d'autres.

À Travers ce rapport nous allons tester et détailler 3 algorithmes; la propagation de marque, algorithme d'héritage et par la suite un algorithme qui permet d'inhiber la propagation dans le cas des liens d'exception.

1- Propagation de marque

L'algorithme de propagation de marque, également appelé algorithme de marquage, est une technique utilisée pour propager une marque ou un état à travers un réseau ou un graphe.

L'idée de base de cet algorithme est de marquer initialement certains nœuds du réseau, puis de propager cette marque le long des liens ou des arêtes du réseau, en suivant certaines règles ou conditions définies. L'objectif est de déterminer quels autres nœuds du réseau peuvent être atteints ou influencés à partir des nœuds initialement marqués.

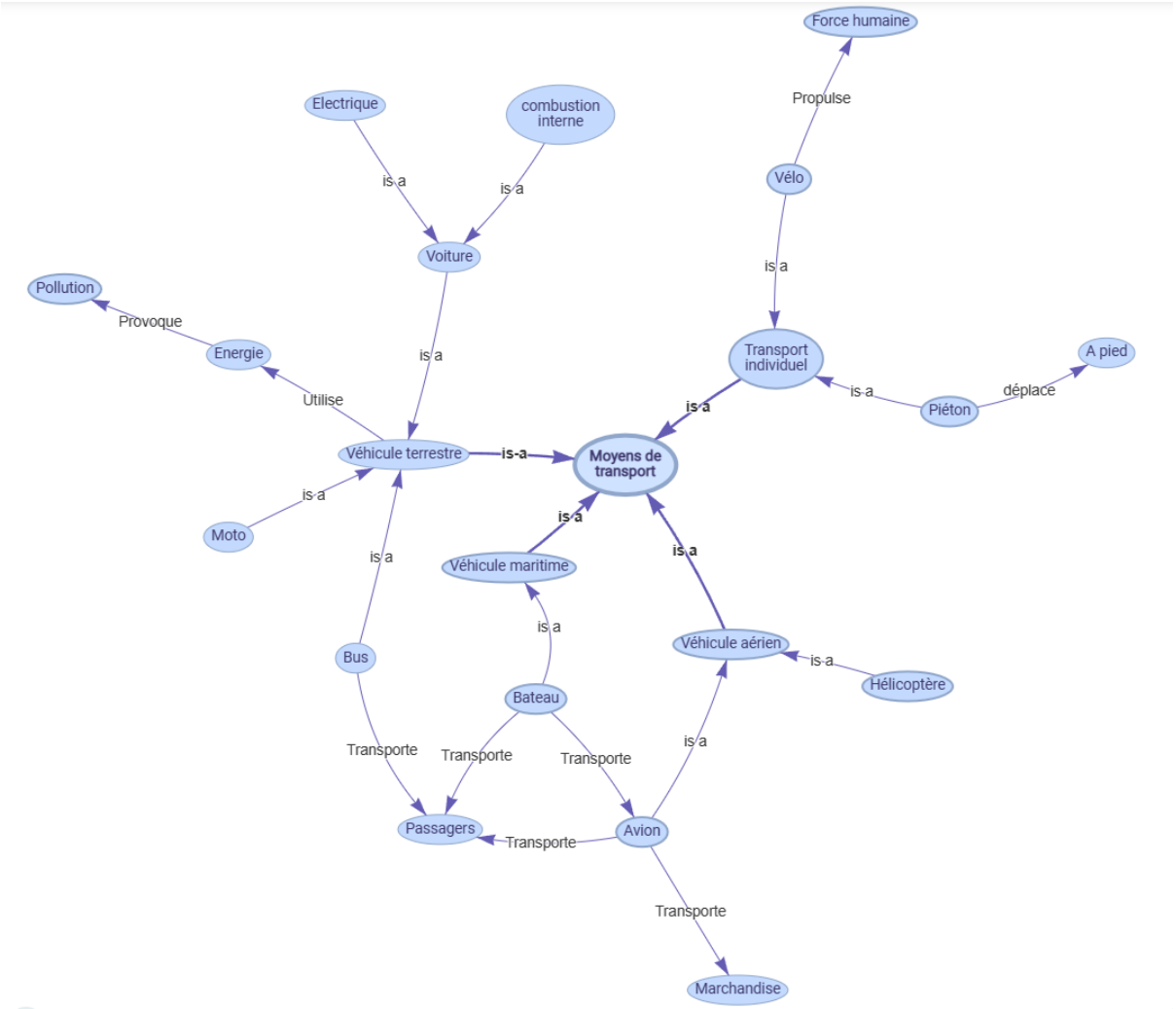
Implementation

Nous allons d'abord construire notre reseau semantique sur l'outil "Plateforme khezour" afin de récupérer le fichier json qu'on va pas la suite traiter avec des algorithmes écrits en python.

Réseaux:

Notre réseau modélise les moyens de transport avec les différentes relations et classes qui vont avec.

le voici dans la figure qui suit.



Réseaux sémantique des moyens de transport

Code:

1- La première partie du code charge les données du réseau sémantique à partir d'un fichier JSON. Le fichier JSON doit contenir les informations sur les nœuds (nodes) et les liens (edges) du réseau.

```
import json

# Charger le fichier JSON
with open('semantic_network.json', 'r') as file:
    data = json.load(file)
    nodes = data['nodes']
    edges = data['edges']
```

2- Ensuite, La fonction, "propagation_de_marqueurs", prend en entrée le réseau sémantique, deux listes de nœuds (node1 et node2) et une relation, et renvoie une liste de solutions trouvées par propagation de marqueurs. Elle effectue les étapes suivantes :

- Elle récupère tous les nœuds du réseau sémantique.
- Elle initialise une liste vide pour stocker les solutions trouvées.
- Elle parcourt les listes de nœuds node1 et node2 en utilisant une boucle for.
- Pour chaque index i dans la boucle, elle essaie d'obtenir les nœuds correspondants dans les listes node1 et node2 en utilisant l'étiquette du nœud.
- Elle récupère toutes les arêtes du réseau sémantique.
- Elle récupère les arêtes de propagation pour le nœud M1, qui ont pour destination le nœud M1 et dont l'étiquette est "is a".
- Tant que la liste des arêtes de propagation n'est pas vide et qu'aucune solution n'a été trouvée, elle effectue les étapes suivantes :
  a. Elle extrait un nœud temporaire de la liste des arêtes de propagation.
  b. Elle récupère les arêtes de contenu du nœud temporaire, qui ont pour source le même nœud que le nœud temporaire et dont l'étiquette est égale à la relation spécifiée.
  c. Elle vérifie si l'un des nœuds de destination des arêtes de contenu est égal au nœud M2. Si c'est le cas, elle a trouvé une solution, sinon, elle poursuit la propagation.
  d. Si aucune solution n'a été trouvée, elle récupère les arêtes "is a" qui ont pour destination le nœud source du nœud temporaire et les ajoute à la liste des arêtes de propagation.
- Elle ajoute à la liste des solutions trouvées le résultat de la fonction "get_label" appliquée au réseau sémantique, au nœud M2 et à la relation spécifiée si une solution a été trouvée, sinon elle ajoute la chaîne "il n'y a pas de lien entre les 2 nœuds".

```
import json

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

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

    solutions_found = []

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

        try:
```

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

            edges = reseau_semantique["edges"]
            propagation_edges = [edge for edge in edges if (edge["to"] == M1["id"] and edge["label"] == "is
a")]
            while len(propagation_edges) != 0 and not solution_found:
                temp_node = propagation_edges.pop()
                temp_node_contient_edges = [edge for edge in edges if (edge["from"] == temp_node["from"]
and edge["label"] == relation)]
                solution_found = any(d['to'] == M2["id"] for d in temp_node_contient_edges)
                if not solution_found:
                    temp_node_is_a_edges = [edge for edge in edges if (edge["to"] == temp_node["from"] and
edge["label"] == "is a")]
                    propagation_edges.extend(temp_node_is_a_edges)

            solutions_found.append(get_label(reseau_semantique, M2, relation) if solution_found else "il n'y
a pas un lien entre les 2 noeuds")
        except IndexError:
            solutions_found.append("Aucune reponse n'est fournie par manque de connaissances.")

    return(solutions_found)

print("Partie 1: l'algorithm de propagation de marqueurs")
f = open('semantic_network.json')

reseau_semantique = json.load(f)

M1_node = [
            "Moyens de transport",
            "Moyens de transport"]

M2_node = ["Passagers", "Marchandise"]

relation = "Transporte"
solutions = propagation_de_marqueurs(reseau_semantique,
                        M1_node,
                        M2_node,
                        relation)

i = 0
#Affichage du resultat
for solution in solutions:
        print(" ".join([M1_node[i], relation, M2_node[i]]))
        i += 1
        print(solution)
```

Résultats

Question:

Quels sont les moyens de transport qui transportent de la marchandise ou des passagères?

On voit bien que l'algorithme résolu la question en utilisant la méthode de propagation de marques

```
PS D:\S2\RCR\TP01> & C:/Users/HP.LAPTOP-ESRMF9MQ/AppData/Local/Programs/Python/Python310/python.exe
Partie 1: l'algorithm de propagation de marqueurs
Moyens de transport Transporte Passagers
il y a un lien entre les 2 noeuds : Bus, Avion, Bateau
Moyens de transport Transporte Marchandise
il y a un lien entre les 2 noeuds : Avion
```

2- Héritage

Code:

Définition de la fonction heritage :

Cette fonction effectue l'inférence sur le réseau sémantique en recherchant les héritages d'un nœud spécifié par son étiquette.

- Elle utilise les nœuds et les arêtes du réseau sémantique fourni.
- La fonction recherche le nœud correspondant à l'étiquette donnée (name) dans la liste des nœuds.
- Ensuite, elle recherche les arêtes sortantes de ce nœud avec l'étiquette "is_a" et enregistre les nœuds de destination de ces arêtes dans une liste appelée legacy_edges.
- La boucle principale de la fonction parcourt les arêtes "is_a" en commençant par le dernier élément de legacy_edges. Elle ajoute l'étiquette correspondante du nœud de destination dans la liste legacy et étend la liste legacy_edges avec les nœuds de destination des nouvelles arêtes "is_a" trouvées.
- De plus, la fonction récupère les arêtes autres que "is_a" reliant le nœud en cours à d'autres nœuds, enregistre ces propriétés dans la liste properties, sous la forme d'une chaîne de caractères contenant l'étiquette de l'arête et l'étiquette du nœud de destination.
- La boucle principale se répète jusqu'à ce que la liste legacy_edges soit vide, ce qui indique que tous les héritages ont été explorés.
- Finalement, la fonction renvoie les listes legacy (contenant les héritages) et properties (contenant les propriétés).

```
def heritage(reseau_semantique, name):
    the_end = False

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

    node = [node for node in nodes if node["label"] == name][0]
    legacy_edges = [edge["to"] for edge in edges if (edge["from"] == node["id"]
and edge["label"] == "is a")]
    legacy = []
    properties = []
```

```
    while not the_end:
        n = legacy_edges.pop()
        legacy.append(get_label(reseau_semantique, n))
        legacy_edges.extend([edge["to"] for edge in edges if (edge["from"] == n
and edge["label"] == "is a")])
        properties_nodes = [edge for edge in edges if (edge["from"] == n and
edge["label"] != "is a")]
        for pn in properties_nodes:
            properties.append(": ".join([pn["label"],
get_label(reseau_semantique, pn["to"])]))
        if len(legacy_edges) == 0:
            the_end = True

    return legacy, properties
```

Résultats

Question:

Utilisez le noeud ‘Avion’ :

```
PS D:\S2\RCR\TP01> & C:/Users/HP.LAPTOP-ESRMF9MQ/AppData/Local/Programs/Python/Python310/python.exe
Partie 2: l'algorithm d'heritage
Resultat de l'inference utiliser:
Avion
VÃ©hicule aÃ©rien
Moyens de transport
Deduction des priorites:
```

2- Exception

Code:

La fonction propagation_de_marqueurs effectue la propagation de marqueurs à travers le réseau sémantique. Elle prend en paramètre le réseau sémantique, deux nœuds initiaux (node1 et node2), et une relation spécifique.

- La fonction récupère la liste des nœuds du réseau.
- Elle initialise une liste vide solutions_found pour stocker les solutions trouvées.
- À chaque itération de la boucle, la fonction vérifie s'il reste des nœuds à traiter (node1 et node2 ont la même longueur) et si une solution a déjà été trouvée.

- Pour chaque paire de nœuds à traiter, la fonction tente de récupérer les nœuds correspondants (M1 et M2) dans le réseau sémantique.
- Ensuite, elle récupère les arêtes de propagation à partir du nœud M1, en ignorant les arêtes de type "exception".
- La fonction continue la propagation tant qu'il y a des arêtes de propagation et qu'aucune solution n'a été trouvée.
- Lors de chaque itération, elle vérifie si le nœud M2 est atteint par une arête de la relation spécifiée. Si c'est le cas, elle marque la solution comme trouvée.
- Si aucune solution n'est trouvée, elle récupère également les arêtes "is a" du nœud temporaire et les ajoute à la liste des arêtes de propagation.
- Enfin, elle ajoute la solution trouvée ou un message indiquant l'absence de lien entre les nœuds dans la liste solutions_found.

```
def propagation_de_marqueurs(reseau_semantique, node1, node2, relation):
    nodes = reseau_semantique["nodes"]
    solutions_found = []
    for i in range(min(len(node1),len(node2))):
        solution_found = False
        try:
            M1 = [node for node in nodes if node["label"] == node1[i]][0]
            M2 = [node for node in nodes if node["label"] == node2[i]][0]

            edges = reseau_semantique["edges"]
            propagation_edges = [edge for edge in edges if (edge["to"] ==
M1["id"] and edge["label"] == "is a" and edge["edge_type"] != "exception")]
            while len(propagation_edges) != 0 and not solution_found:
                temp_node = propagation_edges.pop()
                temp_node_contient_edges = [edge for edge in edges if
(edge["from"] == temp_node["from"] and edge["label"] == relation and
edge["edge_type"] != "exception")]
                solution_found = any(d['to'] == M2["id"] for d in
temp_node_contient_edges)
                if not solution_found:
                    temp_node_is_a_edges = [edge for edge in edges if (edge["to"]
== temp_node["from"] and edge["label"] == "is a" and edge["edge_type"] !=
"exception")]
                    propagation_edges.extend(temp_node_is_a_edges)

            solutions_found.append(get_label(reseau_semantique, M2, relation) if
solution_found else "il n'y a pas un lien entre les 2 noeuds")
        except IndexError:
            solutions_found.append("Aucune reponse n’est fournie par manque de
connaissances.")

    return(solutions_found)
```

Résultats

Question:

Quels sont les moyens de transport qui transportent des passagers?

```
PS D:\S2\RCR\TP01> & C:/Users/HP.LAPTOP-ESRMF9MQ/AppData/Local/Programs/Python/Python310/python.exe
Partie 3: l'algorithm de propagation de marqueurs avec exception
Moyens de transport Transporte Passagers
il y a un lien entre les 2 noeuds : Avion
```

Conclusion

Au travers de cette tâche pratique, nous avons constaté que l'utilisation d'algorithmes simples nous permet de traiter des questions et des problèmes relativement complexes en analysant des fichiers représentant des réseaux sémantiques. De plus, cette analyse est réalisée dans un laps de temps raisonnable et produit des résultats satisfaisants.

TP06 : Les Logiques de description

Enoncé de l'exercice :

Raisonnement logique de description en exploitant les TBOX et les ABOX, supposons que nous ayons les concepts suivants définis dans notre TBOX :

TBOX :

1.Entreprise est : Startup, Grande entreprise, Petite entreprise.
- Concept : Entreprise
- Sous-concepts : Startup, Grande entreprise, Petite entreprise

2.Un employé appartient à une seule entreprise
-Concept : Employé

3.Un produit appartient à une seule entreprise
- Concept : Produit
4.Une entreprise a au moins deux employés.

ABOX :

1. Assertion : Apple est une grande entreprise.

- Concept : Entreprise
- Individu : Apple
- Propriété : EstDeType
- Valeur : Grande entreprise

2. Assertion : Google est une entreprise.
- Concept : Entreprise
- Individu : Google

3. Assertion : OpenAI est une startup.
- Concept : Startup
- Individu : OpenAI

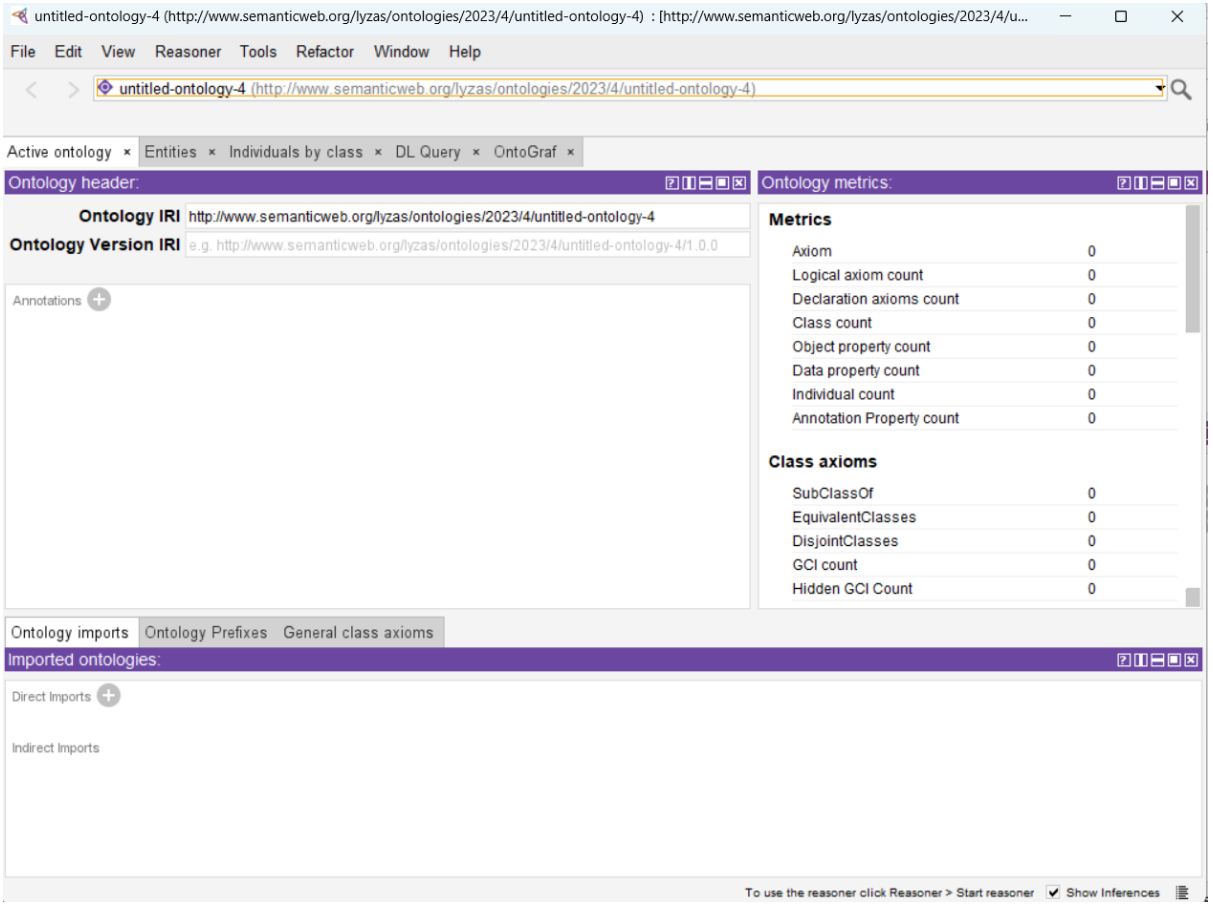
4. Assertion : John Doe travaille chez Apple en tant que développeur.
- Concept : Employé
- Individu : John Doe
- Propriété : TravailleChez
- Valeur : Apple

5. Assertion : L'iPhone est un produit d'Apple.
- Concept : Produit
- Individu : iPhone
- Propriété : EstDeType
- Valeur : Produit
- Propriété : AppartientÀ
- Valeur : Apple

5. Assertion : Katia travaille chez OpenAI.
- Concept : Employé
- Individu : Katia.
- Propriété : TravailleChez
- Valeur : OpenAI

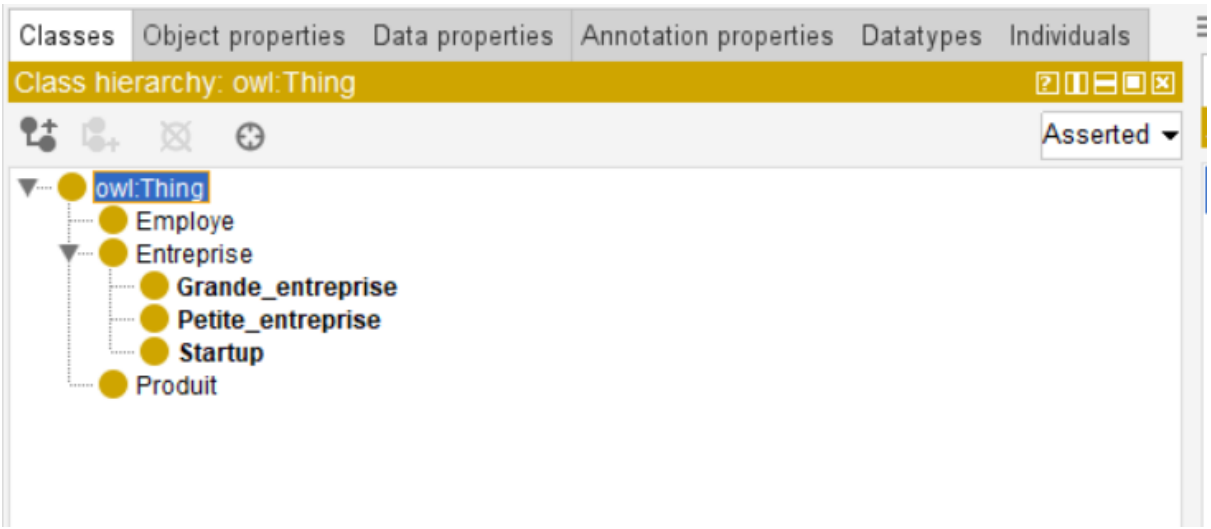
Outil utilisé :

Protégé est un éditeur d'ontologie gratuit et open source et un cadre pour la construction de systèmes intelligents.

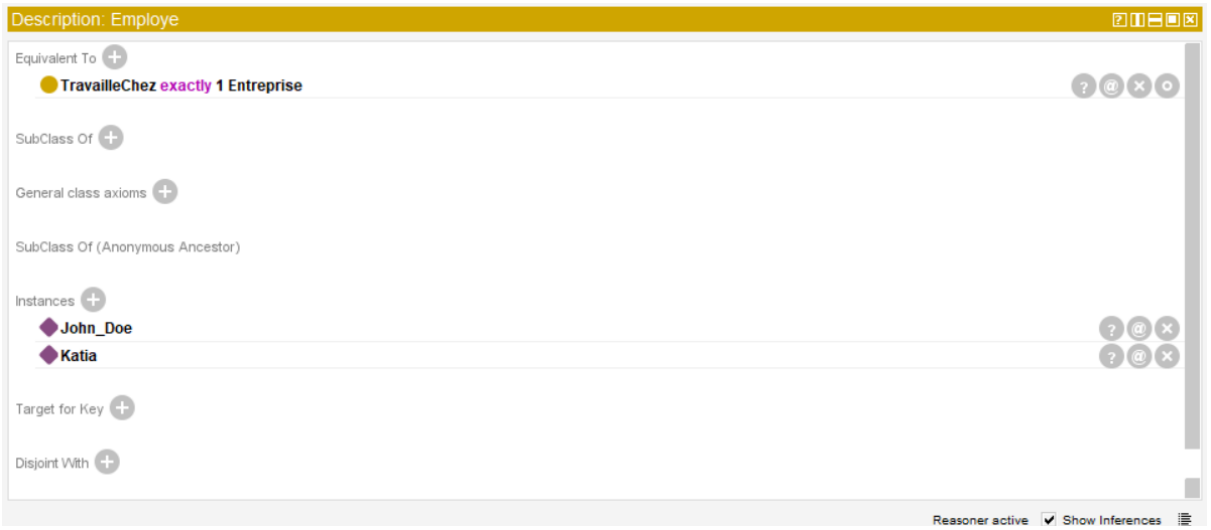


Représentation du modèle modal :

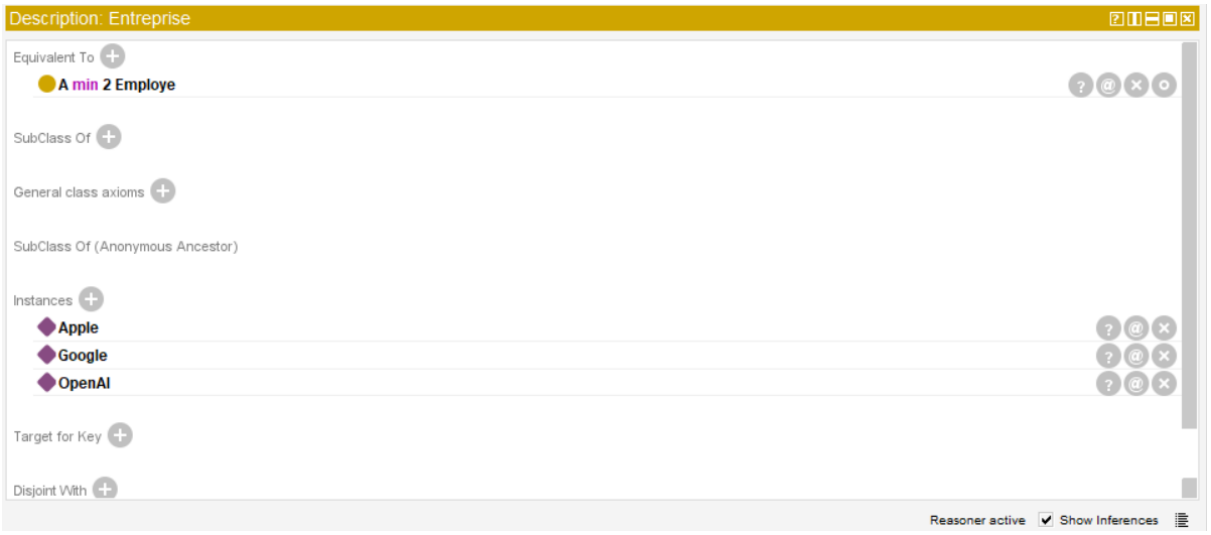
Création des classes et sous-classes :



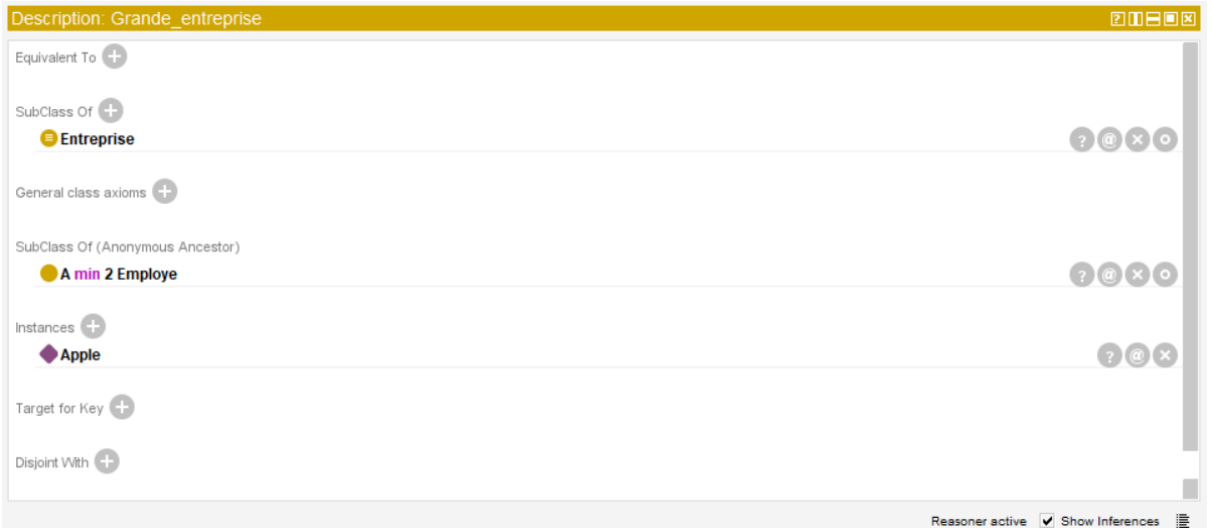
Description de la classe employe :



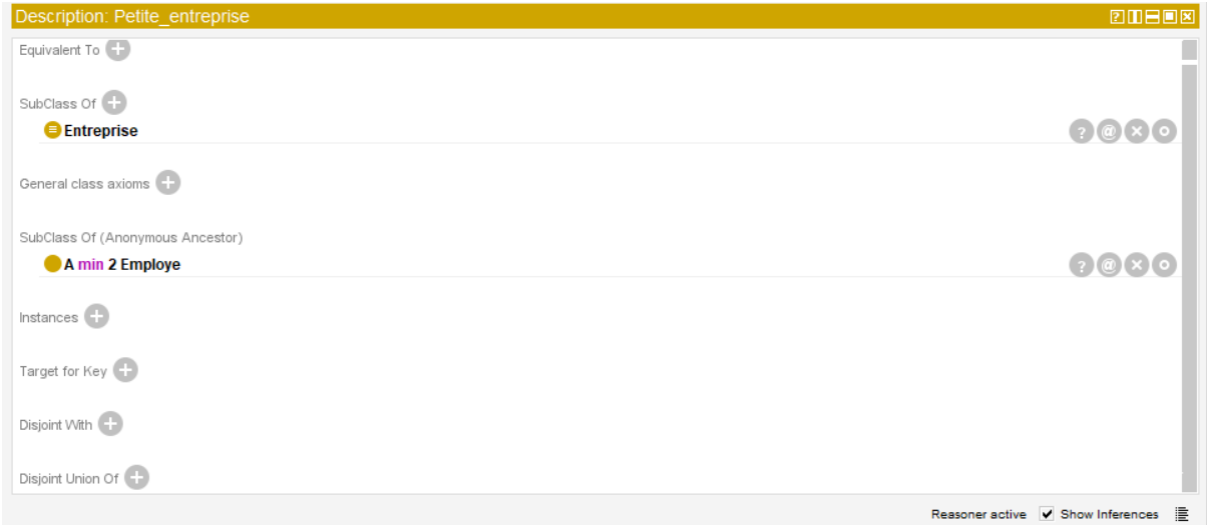
Description de la classe Entreprise :



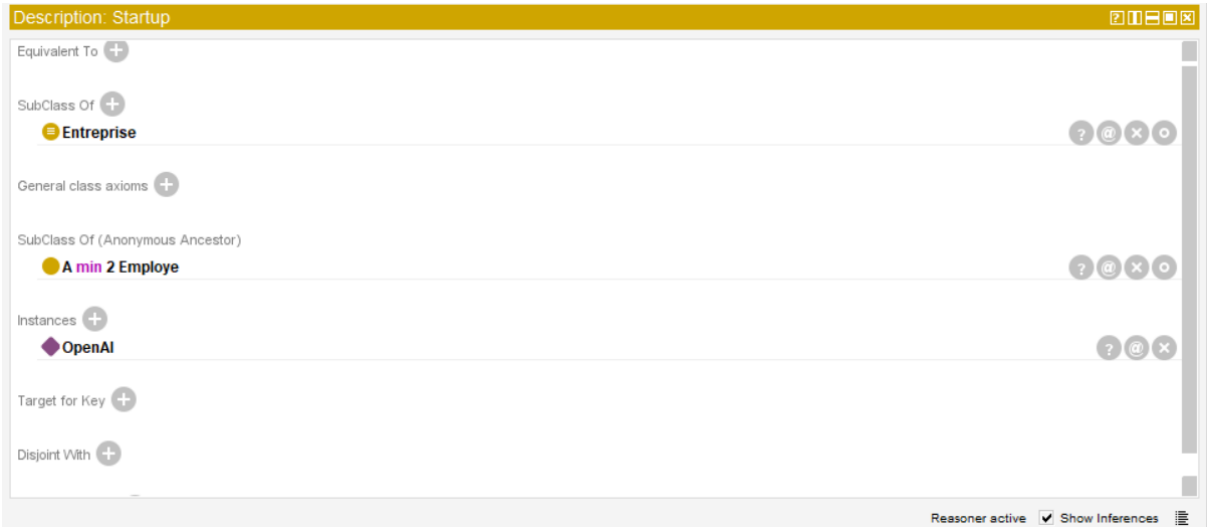
Sous-classe Grande entreprise :



Sous-classe petite entreprise :



Sous-classe startup:

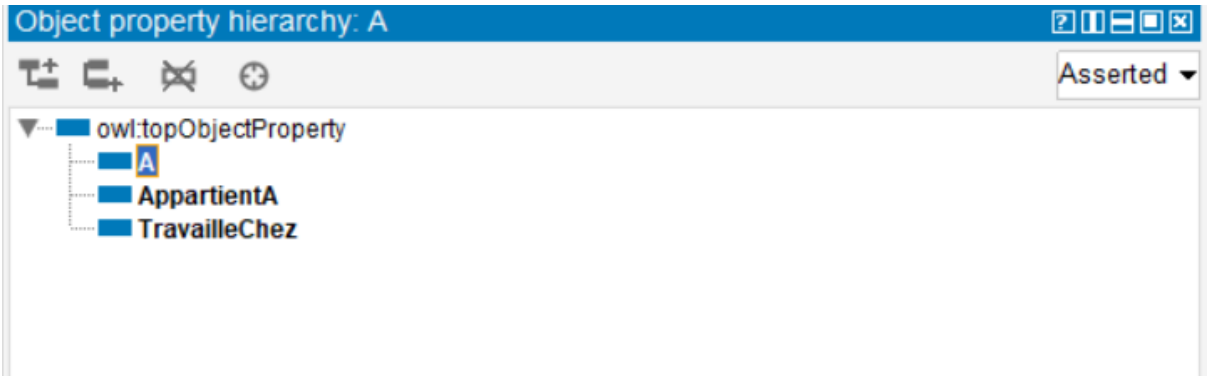


Création des propriétés :

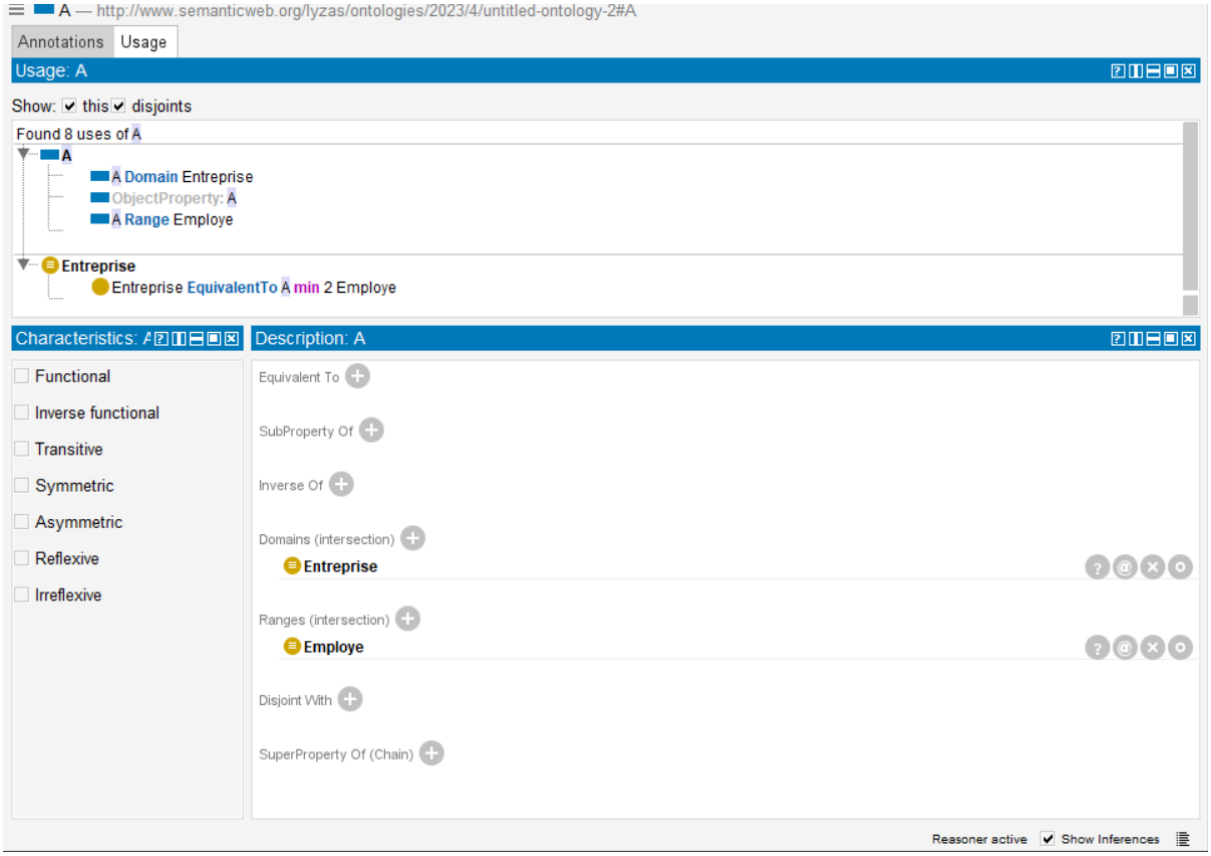
En utilisant le raisonnement logique de description, nous pouvons faire des déductions basées sur ces informations ,qui seront affichés dans la fenêtre **Usage** de chaque captures d'écran :

1. Compte tenu de l'assertion 1, nous pouvons déduire que Apple est une entreprise.

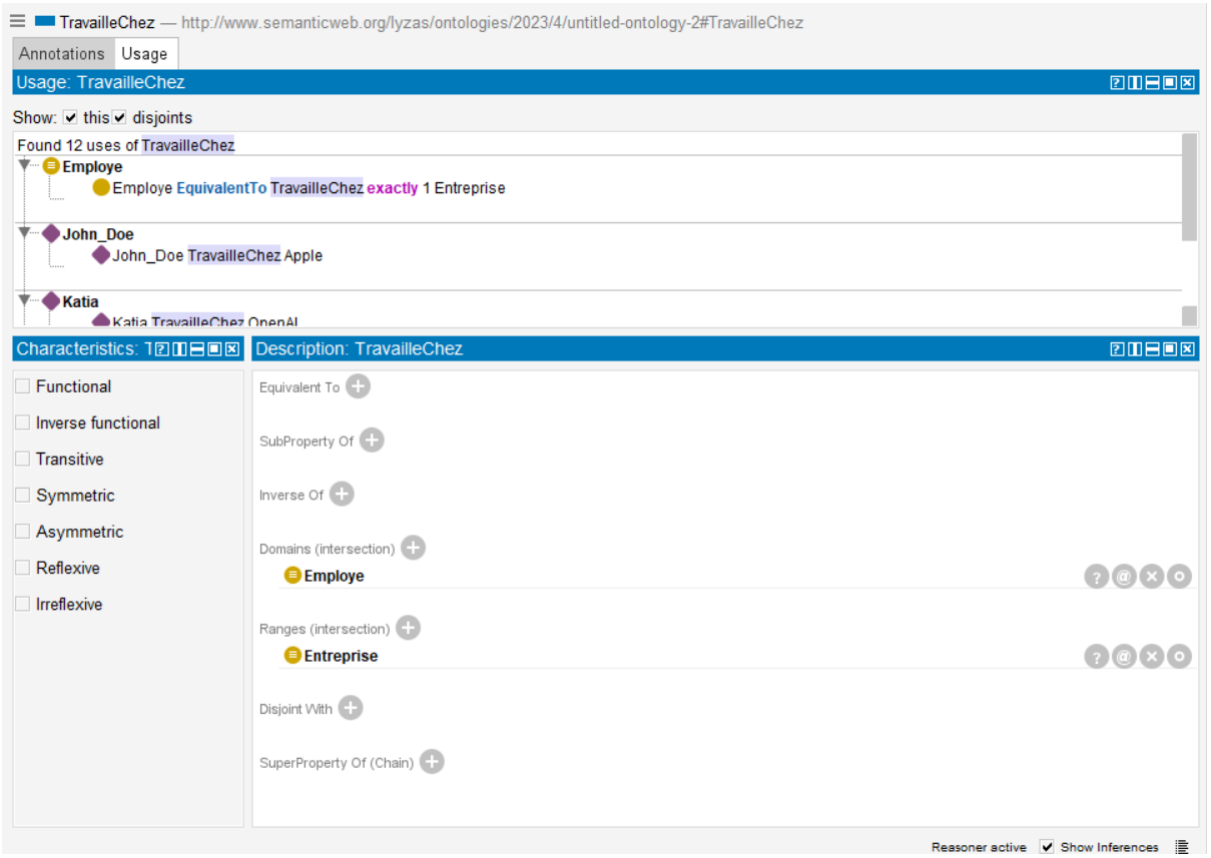
2. Compte tenu de l'assertion 5, nous pouvons déduire que l'iPhone est un produit d'Apple qui est une entreprise.



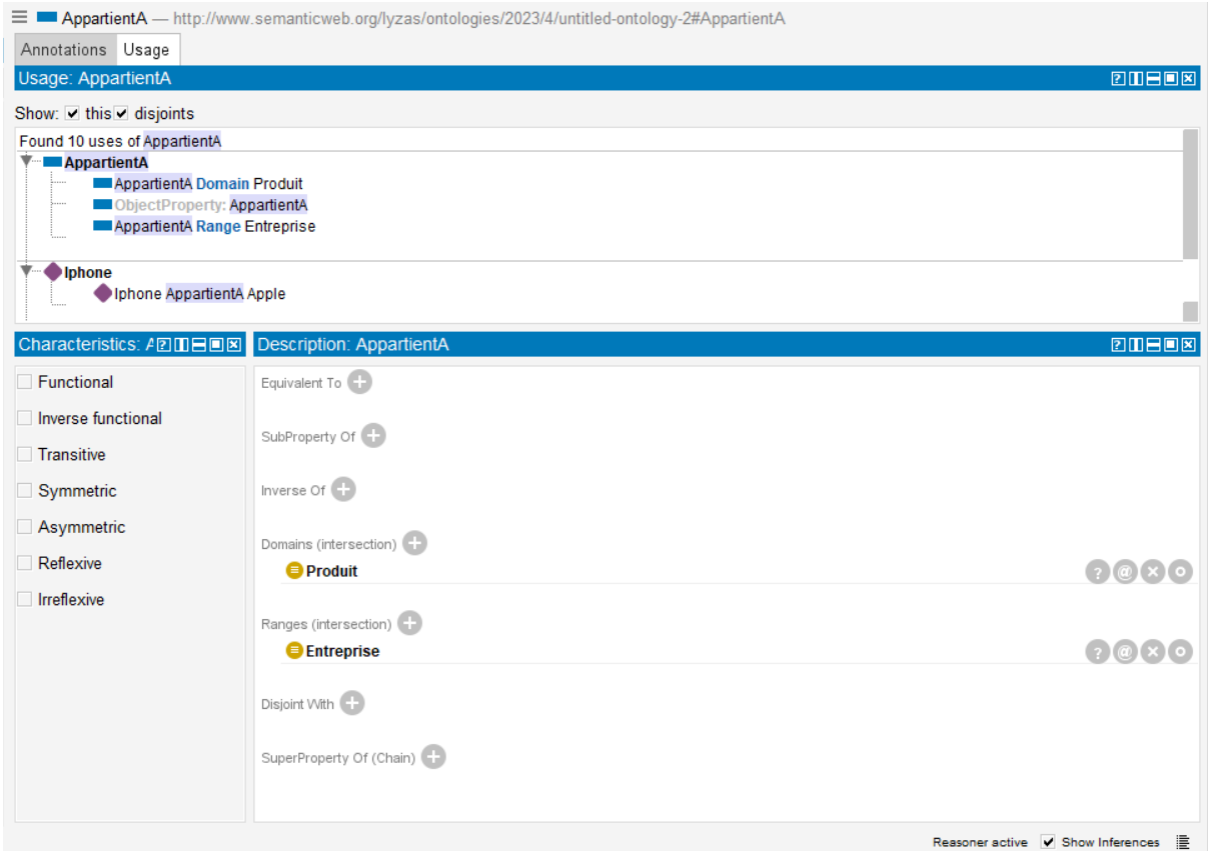
propriété A :



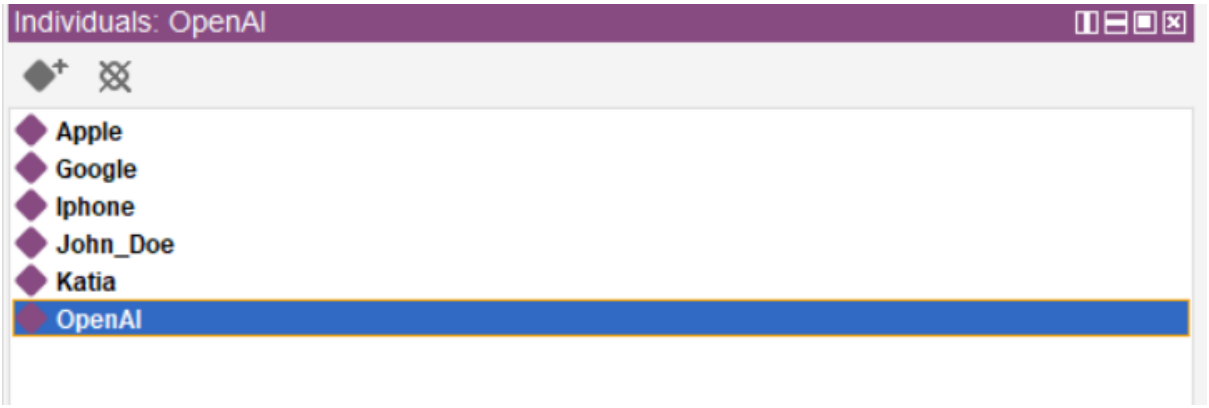
Propriété travailleChez :



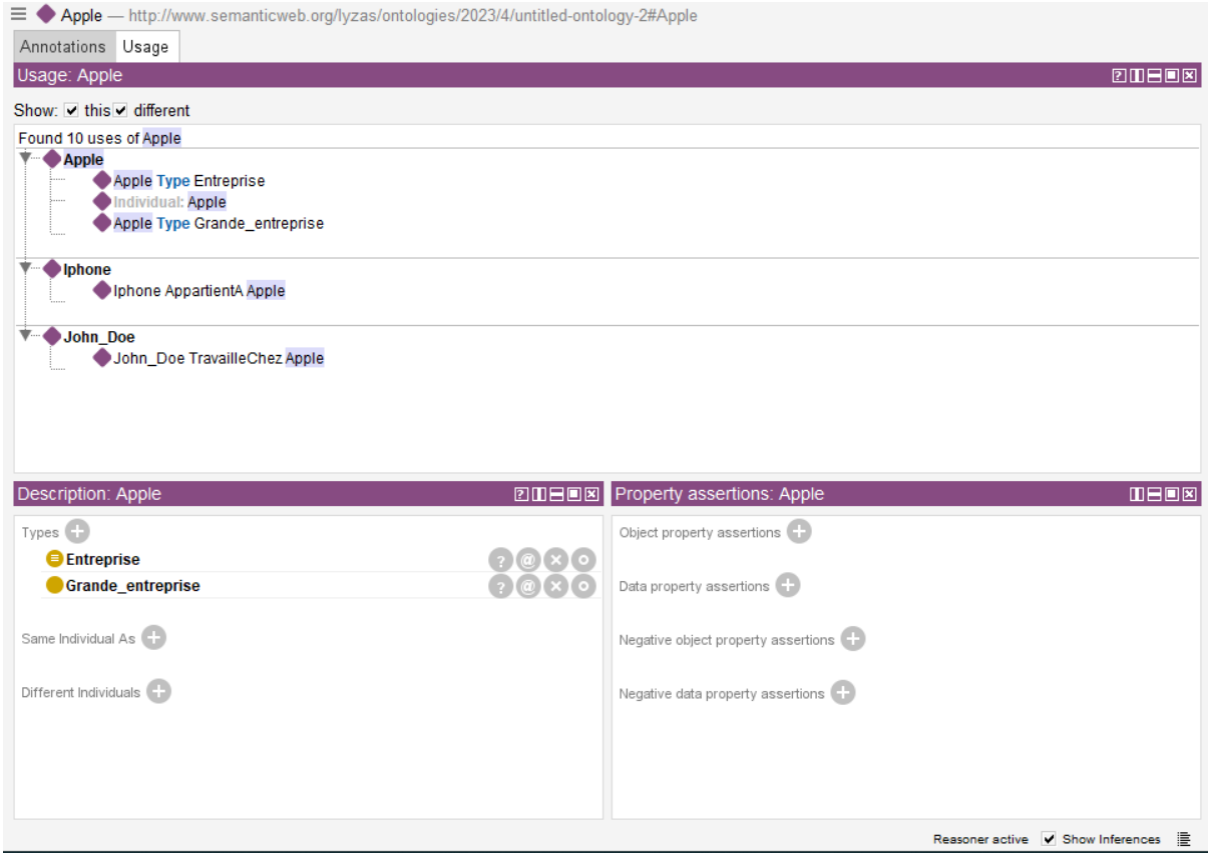
Propriété appartientA :



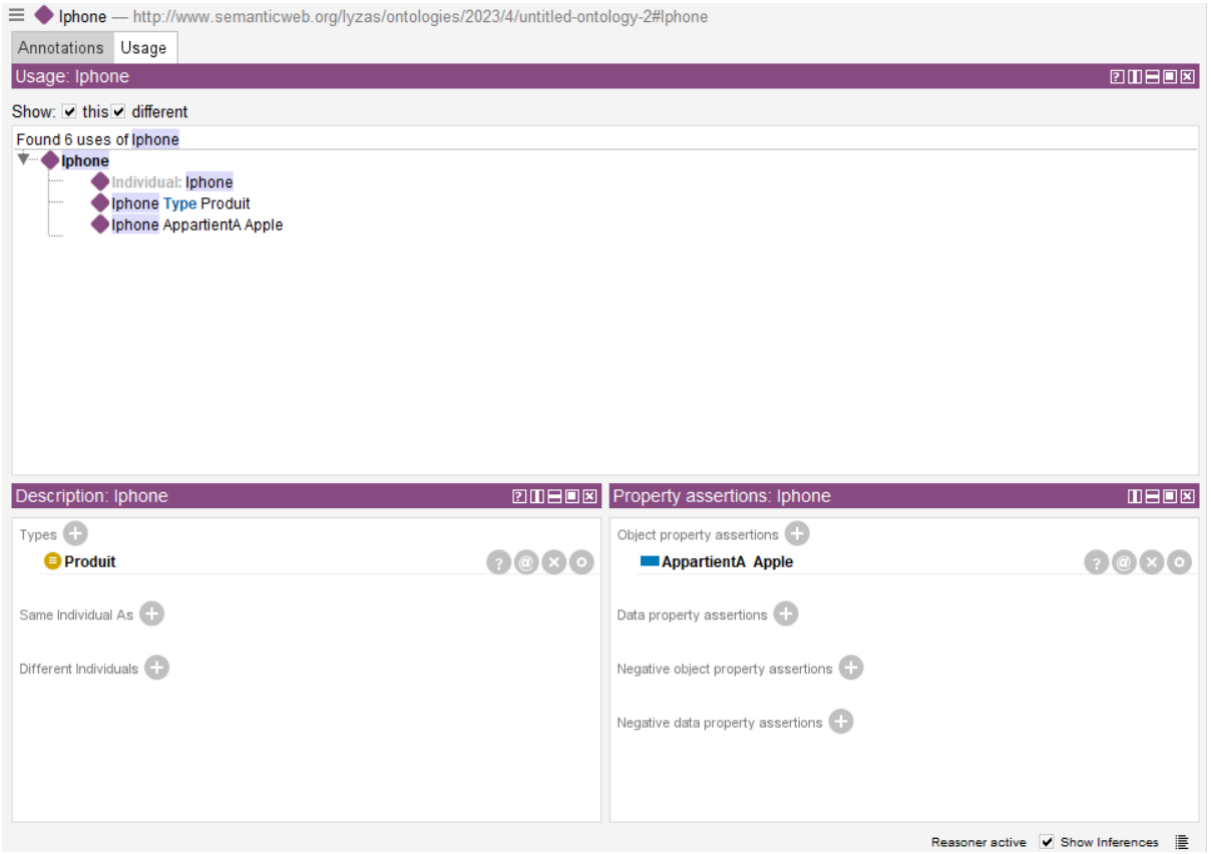
Création des individus :



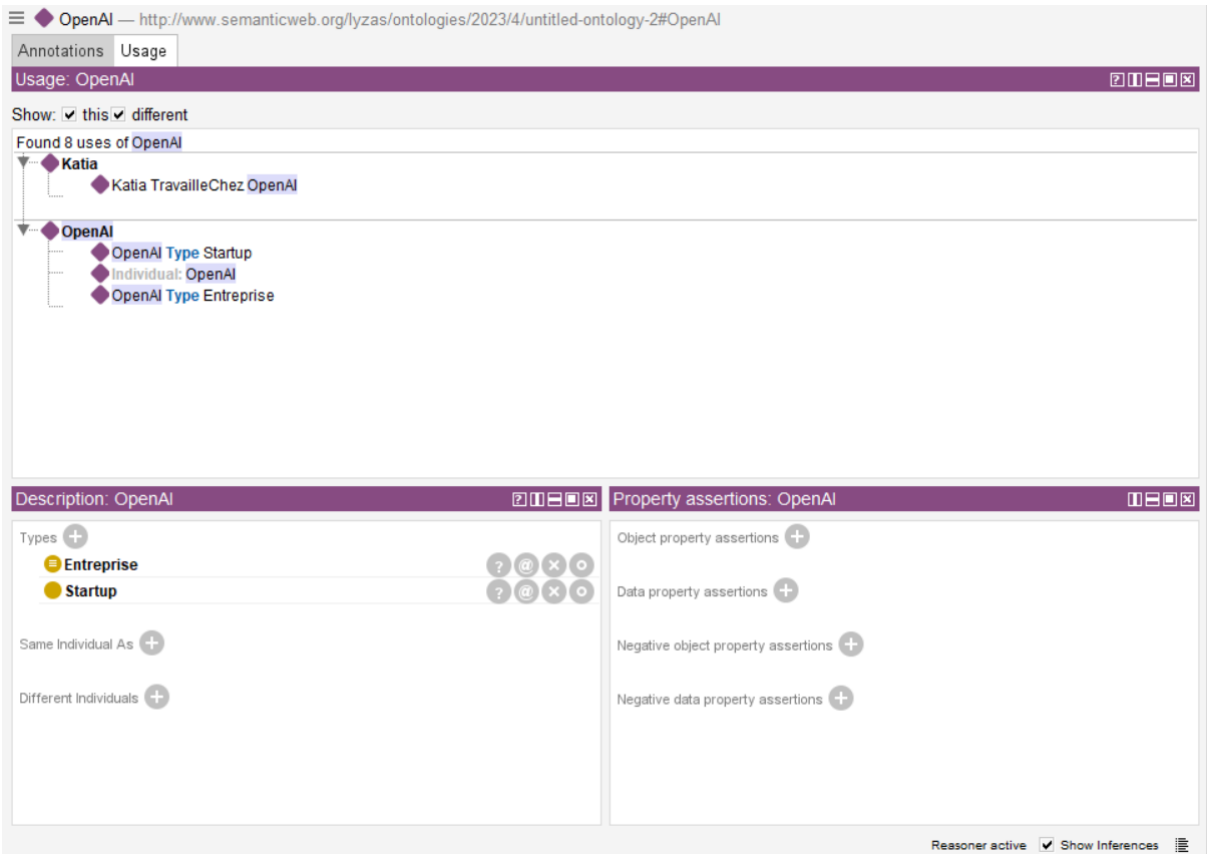
Apple :



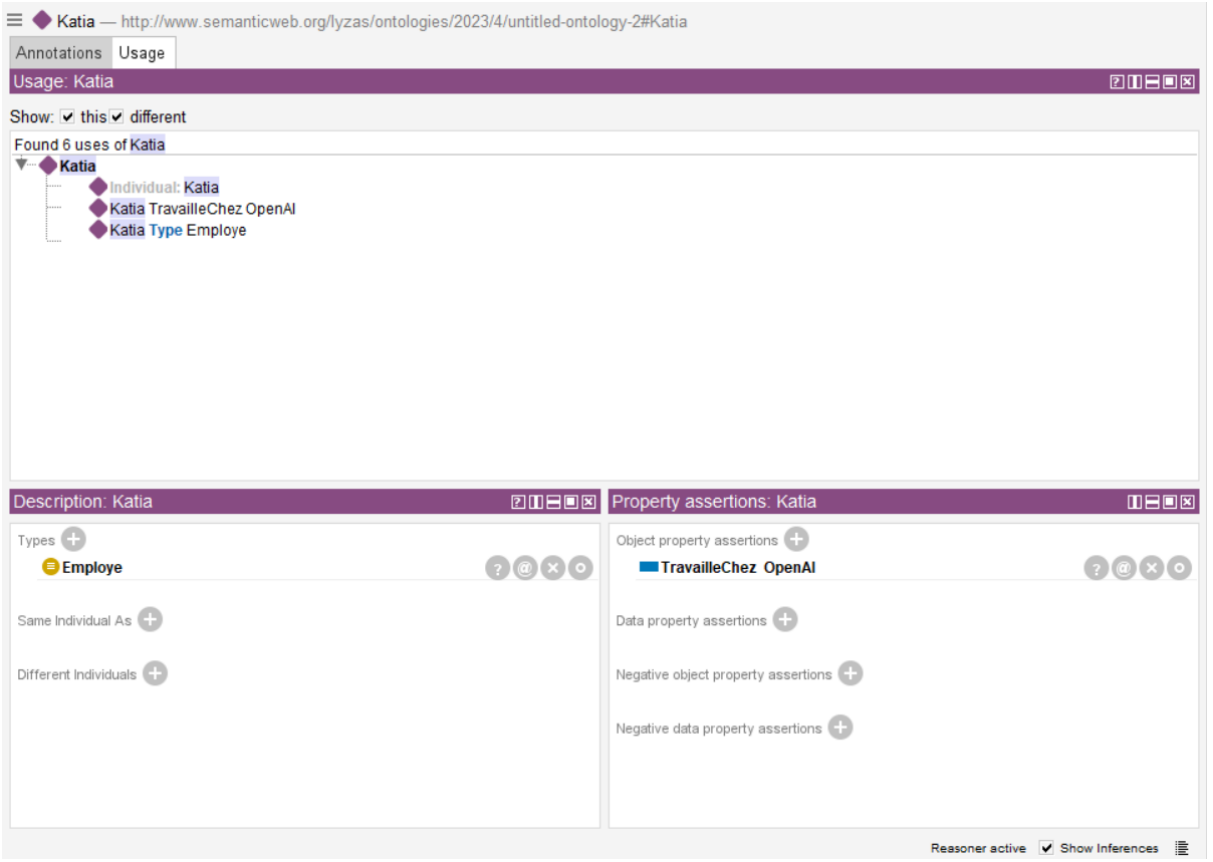
Iphone :


Iphone — http://www.semanticweb.org/lyzas/ontologies/2023/4/untitled-ontology-2#Iphone
Annotations
Usage
Usage: Iphone
Show: this different
Found 6 uses of Iphone
Iphone
Individual: Iphone
Iphone Type Produit
Iphone AppartientA Apple
Description: Iphone
Types
Produit
Same Individual As
Different Individuals
Property assertions: Iphone
Object property assertions
AppartientA Apple
Data property assertions
Negative object property assertions
Negative data property assertions
Reasoner active
Show Inferences

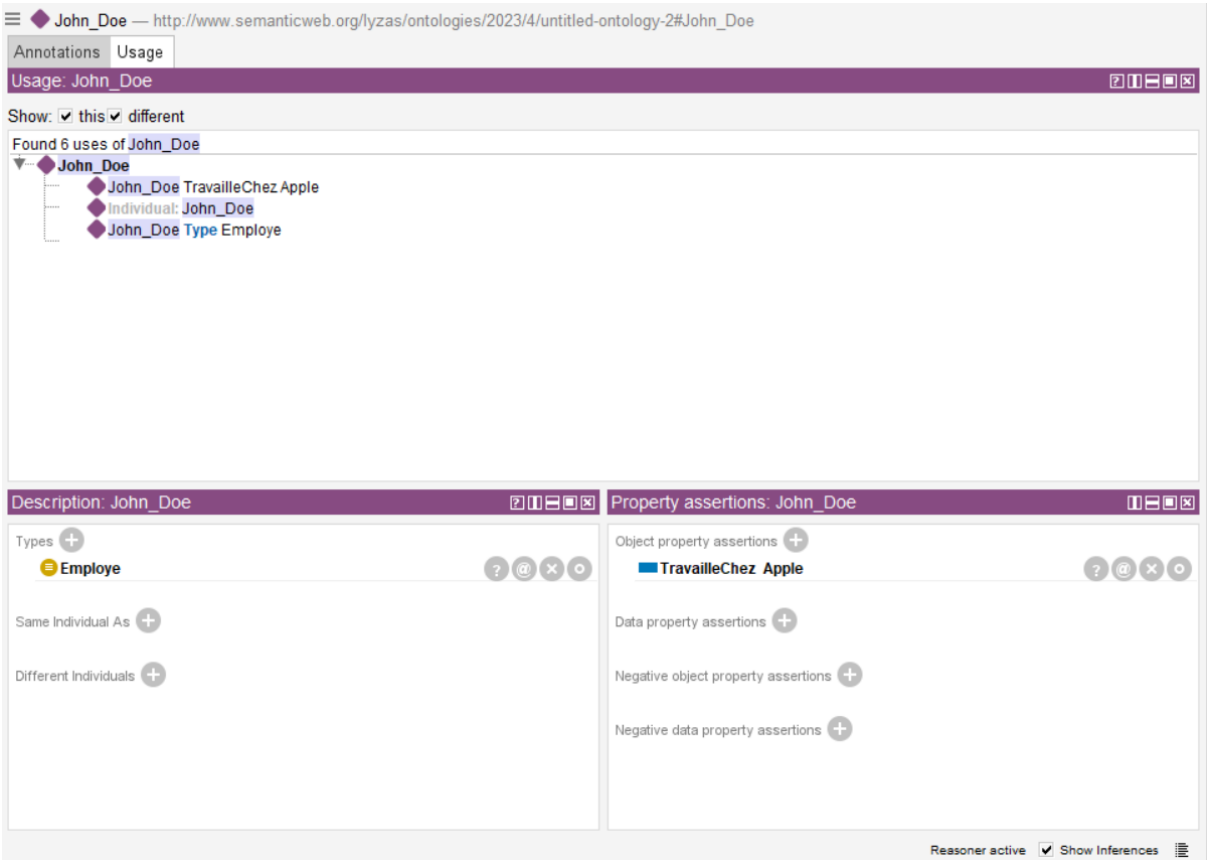
OpenAI :


OpenAI — http://www.semanticweb.org/lyzas/ontologies/2023/4/untitled-ontology-2#OpenAI
Annotations
Usage
Usage: OpenAI
Show: this different
Found 8 uses of OpenAI
Katia
Katia TravailleChez OpenAI
OpenAI
OpenAI Type Startup
Individual: OpenAI
OpenAI Type Entreprise
Description: OpenAI
Types
Entreprise
Startup
Same Individual As
Different Individuals
Property assertions: OpenAI
Object property assertions
Data property assertions
Negative object property assertions
Negative data property assertions
Reasoner active
Show Inferences

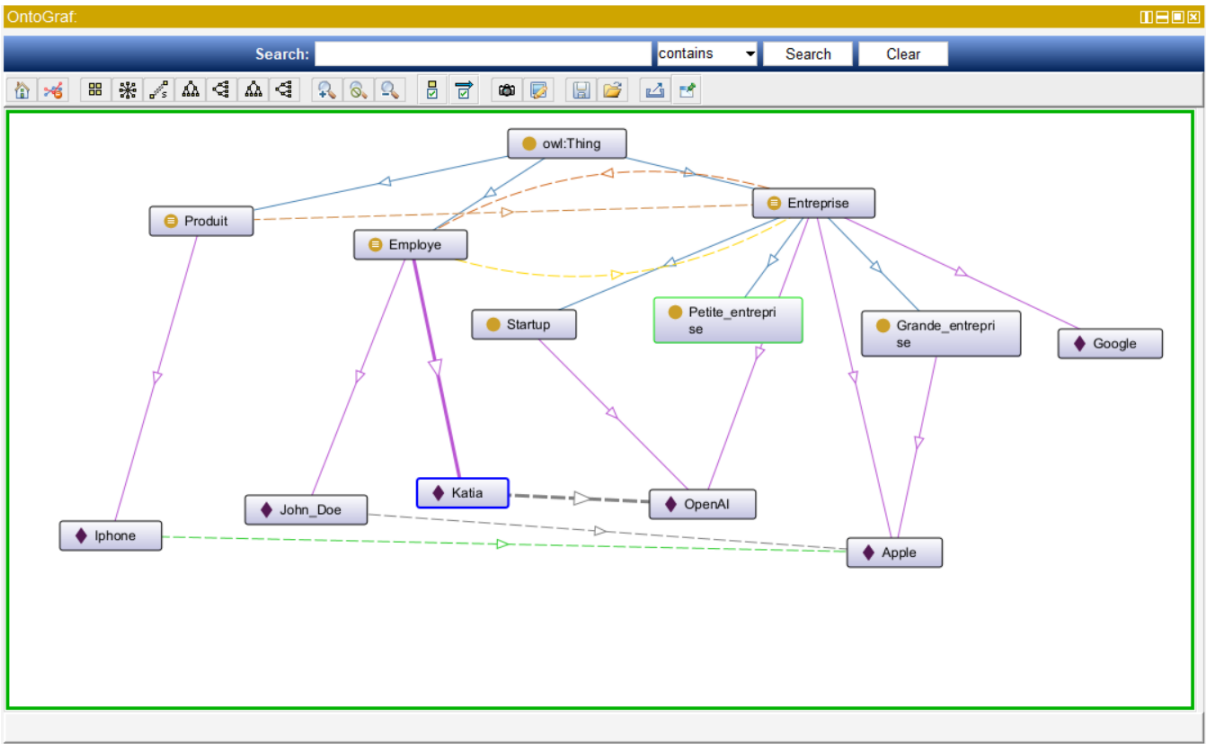
Katia :



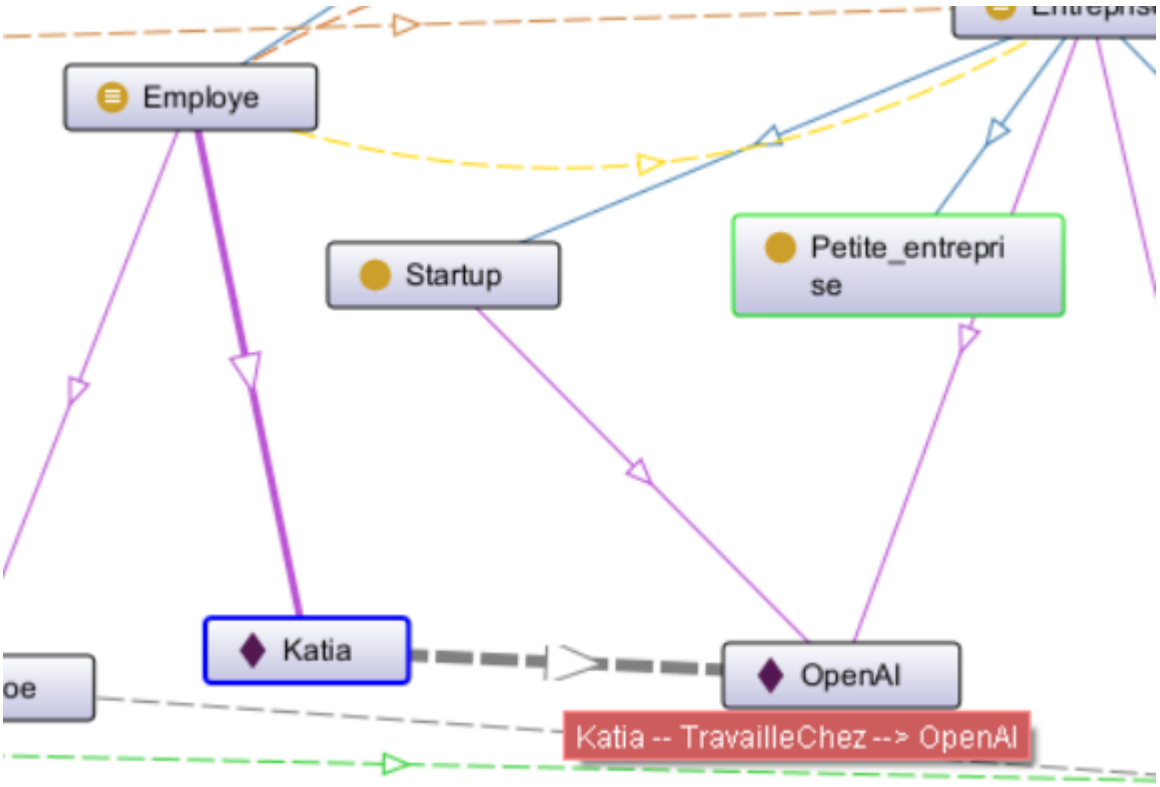
John Doe :



Visualisation sur OntoGraf :



L'arc entre Katia et OpenAI : fait reference à la ABOX n° 5 :



Conclusion

Grâce à l'utilisation de divers outils, nous avons pu approfondir notre compréhension des concepts étudiés et explorer les différents formalismes utilisés en intelligence artificielle pour la représentation des connaissances et le raisonnement. Cela nous a permis de tirer des conclusions générales plus éclairées et de mieux appréhender l'application de ces concepts dans le domaine de l'intelligence artificielle.