



# Université des Sciences et de la Technologie Houari Boumediene

Faculté d'Electronique et d'Informatique

Département d'Informatique

Spécialité:

Systèmes Informatiques Intelligents

## Rapport de TP RCR

---

# TP O3 Les Réseaux Sémantiques.

---

Binôme :

MORSLI Amira  
GANIBARDI Nawel

Professeur :

Mme. Khellaf

# Sommaire

<b>1</b>	<b>Traitement du fichier contenant le réseau sémantique</b>	<b>1</b>
<b>2</b>	<b>Première Étape. Implémenter l'algorithme de propagation de marqueurs dans les réseaux sémantiques.</b>	<b>4</b>
2.1	Principe et fonctionnement de l'algorithme . . . . .	4
2.2	Exemples d'exécution . . . . .	5
2.2.1	Avec saturation . . . . .	5
2.2.2	Sans saturation . . . . .	7
2.2.3	Cas de plusieurs buts . . . . .	7
<b>3</b>	<b>Seconde Étape. Implémenter l'algorithme d'héritage dans les réseaux sémantiques.</b>	<b>8</b>
3.1	Principe et fonctionnement de l'algorithme . . . . .	8
3.2	Exemples d'exécution . . . . .	9
<b>4</b>	<b>Troisième Étape. Génération de fichier contenant un réseau sémantique aléatoire et cohérent.</b>	<b>10</b>

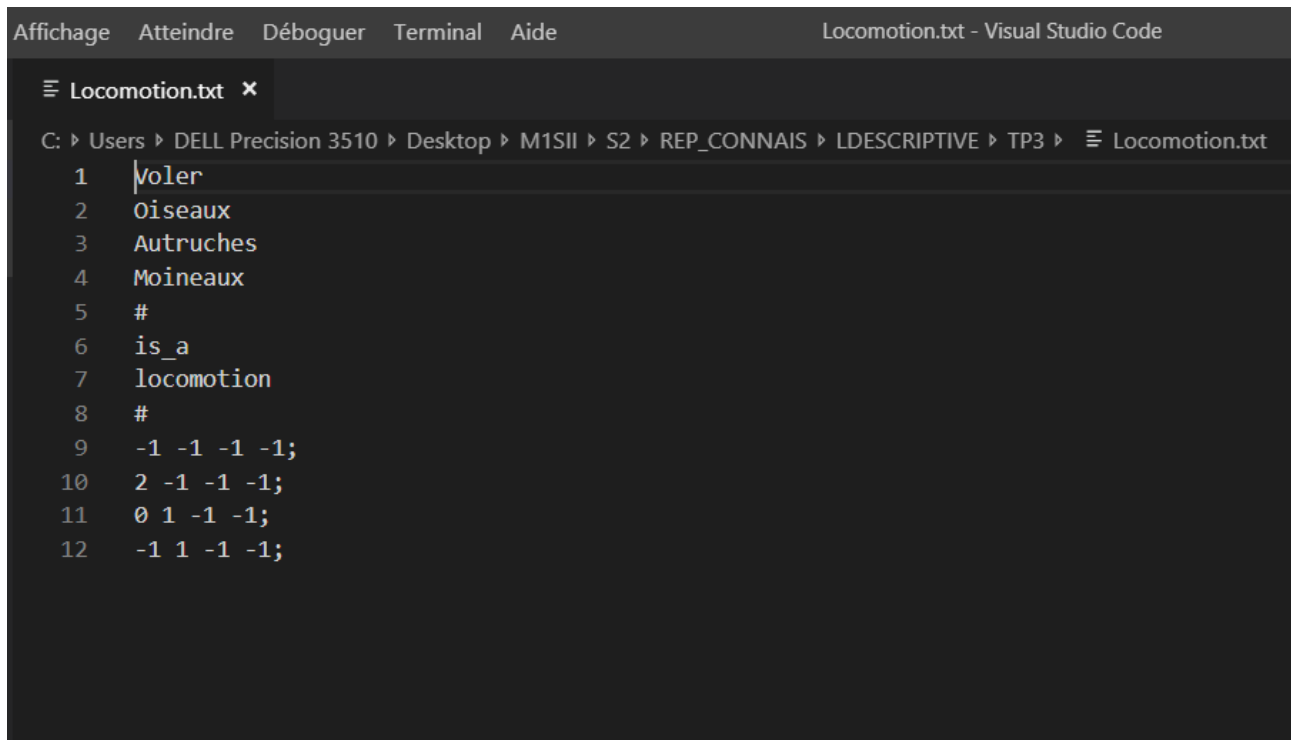
## Table des figures

1	Structure du fichier 'Logiques.txt' contenant le réseau sémantique. . . . .	1
2	Exemple contenant le réseau sémantique 'Elephants.txt'. . . . .	2
3	La partie du code qui traite le fichier contenant le réseau sémantique. . . . .	3
4	La partie du code qui génère la structure dictionnaire 'dict_nodes'. . . . .	3
5	Structure 'dict_node' extraite propre au réseau sémantique 'Elephant.txt'. . . . .	4
6	Propagation avec saturation de l'exemple 'Elephant.txt'. . . . .	5
7	représentation graphique de l'exemple 'Elephant.txt' avant le marquage. . . . .	6
8	représentation graphique de l'exemple 'Elephant.txt' après le marquage. . . . .	6
9	Propagation sans saturation de l'exemple 'Elephant.txt'. . . . .	7
10	Les résultats obtenus dans le cas de deux (2) buts. . . . .	7
11	La fonction de l'héritage qui gère les exceptions. . . . .	8
12	Exemple d'héritage sur l'exemple "Elephant.txt" au noeud 'RoyalElephant'. . . . .	9
13	La fonction de génération du fichier '.txt' contenant le réseau sémantique. . . . .	10
14	Affichage de la structure dictionnaire du fichier généré précédemment. . . . .	11
15	Affichage graphique du réseau sémantique généré précédemment. . . . .	11
16	Application de la fonction d'héritage sur l'un des noeuds du réseau pour démontrer la prise en compte des exceptions et la cohérence de la génération. . . . .	12

# 1 Traitement du fichier contenant le réseau sémantique

Nous avons choisi de représenter notre réseau sémantique à l'aide d'un fichier d'extension '.txt', en suivant le schéma décrit ultérieurement: Une partie listant l'ensemble des concepts. Suivie par la portion décrivant les relations reliant ces concepts (la relation exprimant l'exception étant implicite et n'étant pas comprise dans cette liste). Et enfin, une matrice faisant la relation entre concepts à l'aide d'indices, à savoir: La relation d'indice 1 représentée par un 1, celle d'indice 2, par un 2... La relation d'exception correspond à -1 et une relation inexistante à 0.

Nous illustrons notre explication par un exemple représentant le réseau sémantique 'Logiques' présent au niveau du support de cours fourni.



```
Affichage  Atteindre  Déboguier  Terminal  Aide  Locomotion.txt - Visual Studio Code

≡ Locomotion.txt ✕
C: ▸ Users ▸ DELL Precision 3510 ▸ Desktop ▸ M1SII ▸ S2 ▸ REP_CONNAIS ▸ LDESCRIPTIVE ▸ TP3 ▸ ≡ Locomotion.txt
1  Voler
2  oiseaux
3  Autruches
4  Moineaux
5  #
6  is_a
7  locomotion
8  #
9  -1 -1 -1 -1;
10 2 -1 -1 -1;
11 0 1 -1 -1;
12 -1 1 -1 -1;
```

Figure 1: Structure du fichier 'Logiques.txt' contenant le réseau sémantique.

Ci-dessous un second exemple utilisé pour tester nos implémentations des méthodes proposées: le réseau sémantique 'Elephants.txt'. Cet exemple a été trouvé sur internet, ainsi que sur l'une des séries de TD des années précédentes.

```

Affichage  Atteindre  Déboguier  Terminal  Aide  Elephants.txt - Visual Studio Code
≡ Elephants.txt ✕
C: > Users > DELL Precision 3510 > Desktop > M1SII > S2 > REP_CONNAIS > LDESCRIPTIVE > TP3 > ≡ Elephants.txt
1  AnimauxContemporains
2  MammifereHerbivore
3  Elephant
4  RoyalElephant
5  DustyRoyalElephant
6  Lapin
7  Mammouth
8  Legumes
9  Elephantidaes
10 Tubercules
11 Ecorces
12 Bois
13 Herbe
14 Grise
15 Blanche
16 Trompe
17 Defense
18 E
19 R
20 D
21 #
22 is_a
23 a_couleur
24 a_caracteristique
25 mange
26 #
27 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1;
28 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 4 -1 -1 -1 -1 -1 -1;
29 1 1 -1 -1 -1 -1 -1 4 -1 4 -1 -1 -1 2 -1 3 3 -1 -1;
30 -1 -1 1 -1 -1 -1 -1 -1 -1 -1 -1 -1 0 2 -1 -1 -1 -1;
31 -1 -1 -1 1 -1 -1 -1 -1 -1 -1 4 4 -1 2 0 -1 -1 -1 -1;
32 -1 1 -1 -1 -1 -1 -1 -1 -1 4 -1 -1 -1 -1 -1 -1 -1 -1 -1;
33 -1 -1 -1 -1 -1 -1 -1 4 1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1;

```

Figure 2: Exemple contenant le réseau sémantique 'Elephants.txt'.

Pour la partie traitement de fichier, à savoir conversion du '.txt' en un réseau sémantique, nous avons eu recours à une structure de stockage, qui extrait l'ensemble de noeuds et de relations, et qui convertit la matrice associée de chaîne de caractères en structure matricielle. Cette partie nous est utile pour le marquage, elle facilite le chaînage.

Par la suite, cette structure sert à remplir notre dictionnaire qui, à chaque noeud, fait correspondre une liste de relations. La structure dictionnaire par contre, servira à l'extraction de caractéristiques au moment de l'héritage.

```

class Traitement:
    def __init__(self):
        self.noeuds = None
        self.relations = None
        self.matrice = None

    def read_file(self, chemin):
        f = open(chemin, 'r')
        x = f.read()
        liste = x.split("#")
        self.noeuds = liste[0].splitlines()
        self.relations = remove_empty(liste[1].splitlines())
        tmp = remove_empty(liste[2].replace(";", "").splitlines())
        for i in range(len(tmp)):
            elt = tmp[i]
            elt = delete_empty(elt.split(' '))
            tmp[i] = to_integers(elt)
        self.matrice = np.array(tmp)

    def printer(self):
        print(self.noeuds)
        print(self.relations)
        print(self.matrice)

```

Figure 3: La partie du code qui traite le fichier contenant le réseau sémantique.

```

def create_nodes_relations(noeuds, relations, matrice):
    dict_nodes = {}
    for i in range(len(matrice)):
        svt = []
        rel = {}
        for j in range(len(matrice[i])):
            if matrice[i][j] != -1:
                if matrice[i][j] == 0:
                    if 'is_Exception' not in rel:
                        rel['is_Exception'] = []
                        key = 'is_Exception'
                    else:
                        if relations[matrice[i][j]-1] not in rel:
                            rel[relations[matrice[i][j]-1]] = []
                            key = relations[matrice[i][j]-1]
                rel[key].append(noeuds[j])

        dict_nodes[noeuds[i]] = rel
    return dict_nodes

def print_dict_nodes(dict_nodes):
    for x in dict_nodes:
        print("Noeud: "+x)
        for y in dict_nodes[x]:
            print("Relations: "+str(y)+" :"+str(dict_nodes[x].get(y)))
    print("\n")

```

Figure 4: La partie du code qui génère la structure dictionnaire 'dict\_nodes'.

**Remarque:** Nous remarquons que notre matrice symbolise les relations entre noeuds, et donc ne donne pas directement accès aux relations *d'exception* sur d'autres relations. Pour palier à ce problème, nous avons fait en sorte de préciser le lien d'exception directement avec l'individu qui porte la propriété à exclure, or ces spécificités sont propres à une seule et une unique relation, ce qui substitue donc le lien d'exception direct. Dans l'exemple qui suit, le concept 'DustyRoyalElephant' fait exception à la couleur 'Blanche', Or la couleur blanche est répertoriée comme étant une instance de la relation 'a\_couleur', ce qui revient à dire que le concept 'DustyRoyalElephant' fait exception à la relation 'a\_couleur' pointant sur le concept 'Blanche'.

```

Noeud: AnimauxContemporains

Noeud: MammifereHerbivore
Relations: mange :['Herbe']

Noeud: Elephant
Relations: is_a :['AnimauxContemporains', 'MammifereHerbivore']
Relations: mange :['Legumes', 'Tubercules']
Relations: a_couleur :['Grise']
Relations: a_caracteristique :['Trompe', 'Defense']

Noeud: RoyalElephant
Relations: is_a :['Elephant']
Relations: is_Exception :['Grise']
Relations: a_couleur :['Blanche']

Noeud: DustyRoyalElephant
Relations: is_a :['RoyalElephant']
Relations: mange :['Ecorces', 'Bois']
Relations: a_couleur :['Grise']
Relations: is_Exception :['Blanche']

Noeud: Lapin
Relations: is_a :['MammifereHerbivore']
Relations: mange :['Tubercules']

Noeud: Mammouth
Relations: mange :['Legumes']
Relations: is_a :['Elephantidaes']

```

Figure 5: Structure 'dict\_node' extraite propre au réseau sémantique 'Elephant.txt'.

## 2 Première Étape. Implémenter l'algorithme de propagation de marqueurs dans les réseaux sémantiques.

### 2.1 Principe et fonctionnement de l'algorithme

**En entrée**, l'algorithme dispose :

- D'un réseau sémantique défini par un ensemble de noeuds et de liens.
- De deux plusieurs noeuds marqués, un de départ et l'autre d'arrivée (but).
- D'une relation à démontrer.

**En sortie**, l'algorithme doit fournir, en tenant compte du cas où aucune réponse n'est fournie par manque de connaissances :

- Soit une réponse à la question posée dans le cas où il s'agit d'inférer une seule instance. ou toutes les réponses possibles.

- Les deux listes de marquage M1 et M2.

Cette méthode a pour objectif d'inférer une relation entre deux nœuds M1 et M2 en se basant sur les relations existantes entre ces nœuds et les autres nœuds du réseau. Les étapes suivies sont:

- Ajouter les nœuds de début et de fin (but) aux listes de marquage (respectivement M1 et M2).
- On parcourt chaque nœud  $x$  de la liste M1, en y ajoutant tous les nœuds reliés à  $x$  par la relation 'is\_a'. Nous traitons le nœud de départ en premier.
- On effectue le même traitement concernant la liste M2, mais cette fois-ci à partir du nœud but.
- Une fois l'ensemble des nœuds d'un même niveau parcourus, l'étape de marquage est incrémentée.
- Quand le marquage est terminé, nous parcourons les deux listes M1 et M2, ainsi que la matrice représentative, à la recherche d'un lien portant l'indice correspondant à la relation donnée sous forme de requête. Nous avons à cette étape là trois (3) cas de figure: Aucune solution trouvée, par manque de connaissance. Une seule solution est trouvée ou bien le type de marquage choisi est *sans saturation*, à savoir le traitement s'arrête quand une seule et unique correspondance est trouvée. Toutes les solutions potentielles sont trouvées, dans ce cas-là le type de marquage est dit *avec saturation*.
- La gestion d'exception se fait à ce niveau, en cherchant une correspondance dans la structure dictionnaire, entre les nœuds 'solutions' trouvés et la relation mentionnée dans la requête. Les nœuds faisant exception à la relation se verront retirés des solutions potentielles.
- Enfin, nous affichons la ou les solutions, si elles existent, ainsi que les deux (2) listes de marquage M1 et M2.

## 2.2 Exemples d'exécution

Ci-dessous un ensemble d'exemples illustrant le bon fonctionnement de notre méthode. Nous avons notamment pris en compte l'option de saturation ou non, ainsi que l'éventualité d'avoir plusieurs buts en entrée.

### 2.2.1 Avec saturation

La question posée est 'Quels éléphants sont de couleur grise?'. Donc le nœud 'Éléphant' comme début, le nœud 'Grise' comme but, la relation '*a\_couleur'commerequete,etl'option'saturation'active*'.

```
#Propagation sans saturation
x = Traitement()
path = 'C:\\Users\\TRETEC\\Desktop\\M1SII\\Elephants.txt'
x.read_file(path)
structure = create_nodes_relations(x.noeuds,x.relations,x.matrice)
solution1, relation1, m1, m2 = propagation(x.matrice,x.noeuds,x.relations,'Elephant','Grise','a_couleur')

Liste marquage 1: ['Elephant']
Liste marquage 2: ['Grise']
SANS SATURATION: Solution trouvée, le lien: [['Elephant', 'Grise']]
```

Figure 6: Propagation avec saturation de l'exemple 'Elephant.txt'.

Nous avons également implémenté une représentation graphique du processus de marquage à l'aide d'un outil graphique: la bibliothèque 'networkx'.



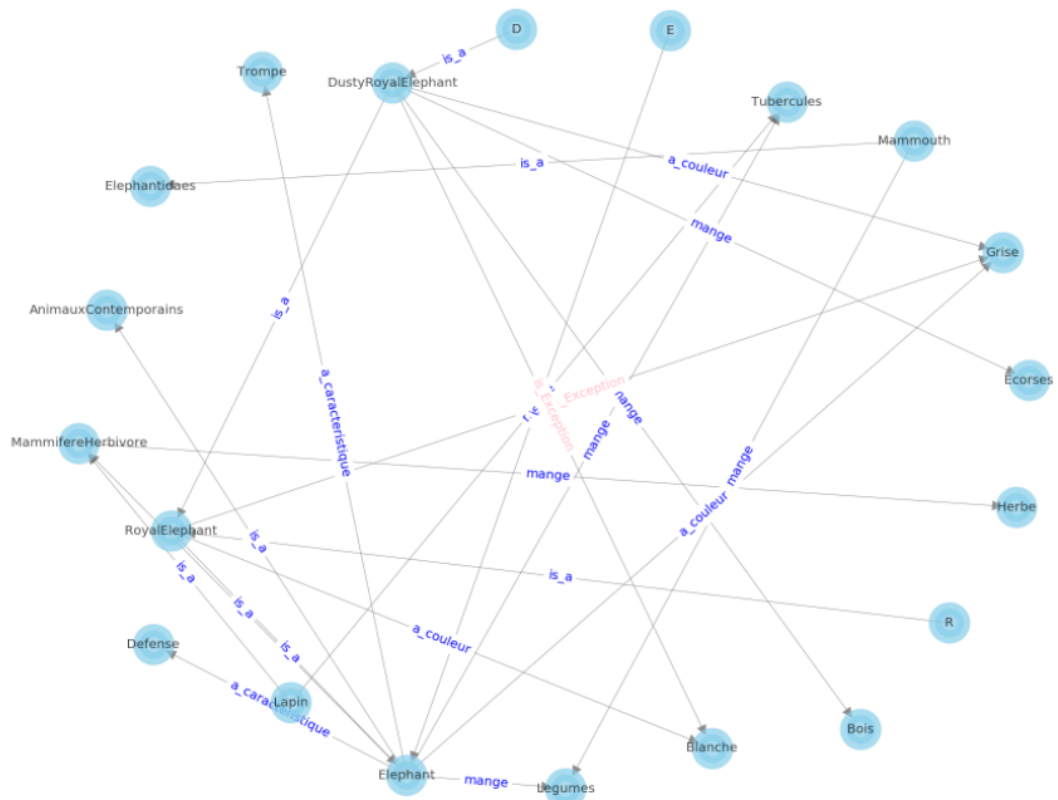


Figure 7: représentation graphique de l'exemple 'Elephant.txt' avant le marquage.

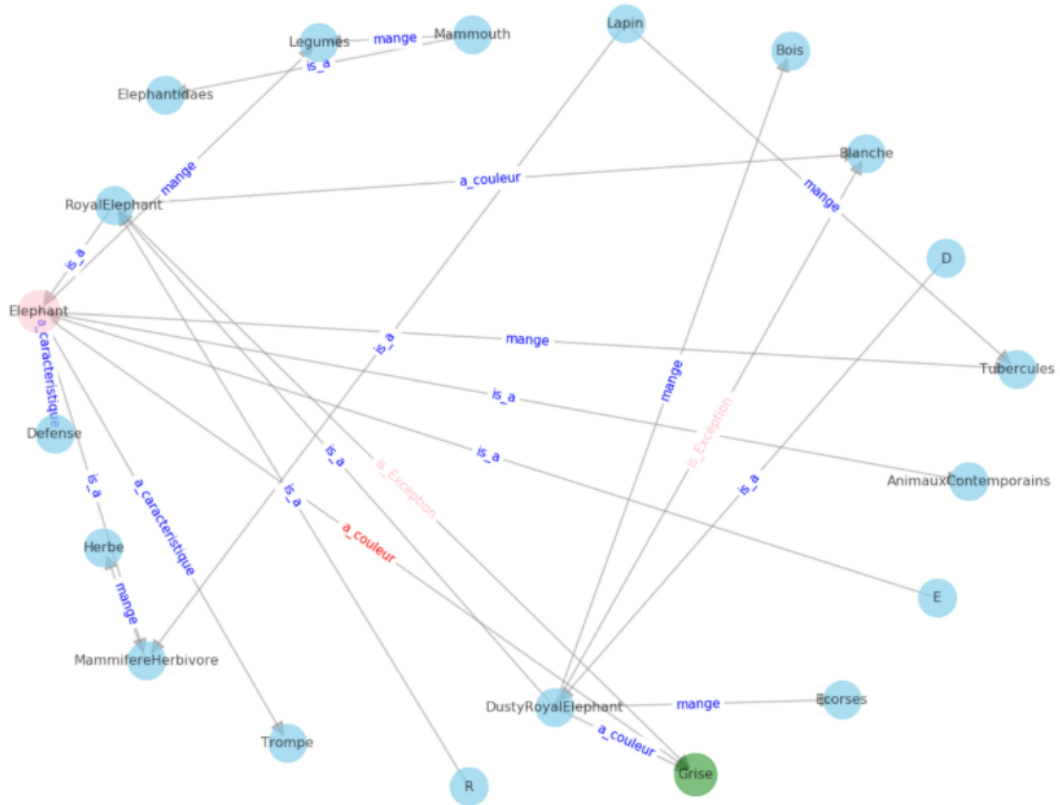


Figure 8: représentation graphique de l'exemple 'Elephant.txt' après le marquage.

## 2.2.2 Sans saturation

La question posée est 'Quel éléphant est de couleur grise?'. Donc le noeud 'Éléphant' comme début, le noeud 'Grise' comme but, la relation 'a\_couleur' comme requête, et l'option 'saturation' désactivée.

```
#Propagation avec saturation
solution_,relation_,m1_,m2_ = propagation_saturation(x.matrice,x.noeds,x.relations,'Elephant','Grise','a_couleur')
arcs,G = creategraph_bis_bis(x.matrice, x.noeds, x.relations, solution_, relation_, m1_, m2_)
#noeds marqués par M1 en rose
#noeds marqués par M2 en vert
#Les liens en rouge marquent une solution

Liste marquage 1: ['Elephant', 'RoyalElephant', 'E', 'DustyRoyalElephant', 'R', 'D']
Liste marquage 2: ['Grise']
AVEC SATURATION: Solution(s) trouvée(s), le(s) lien(s): [['DustyRoyalElephant', 'Grise'], ['Elephant', 'Grise']]
```

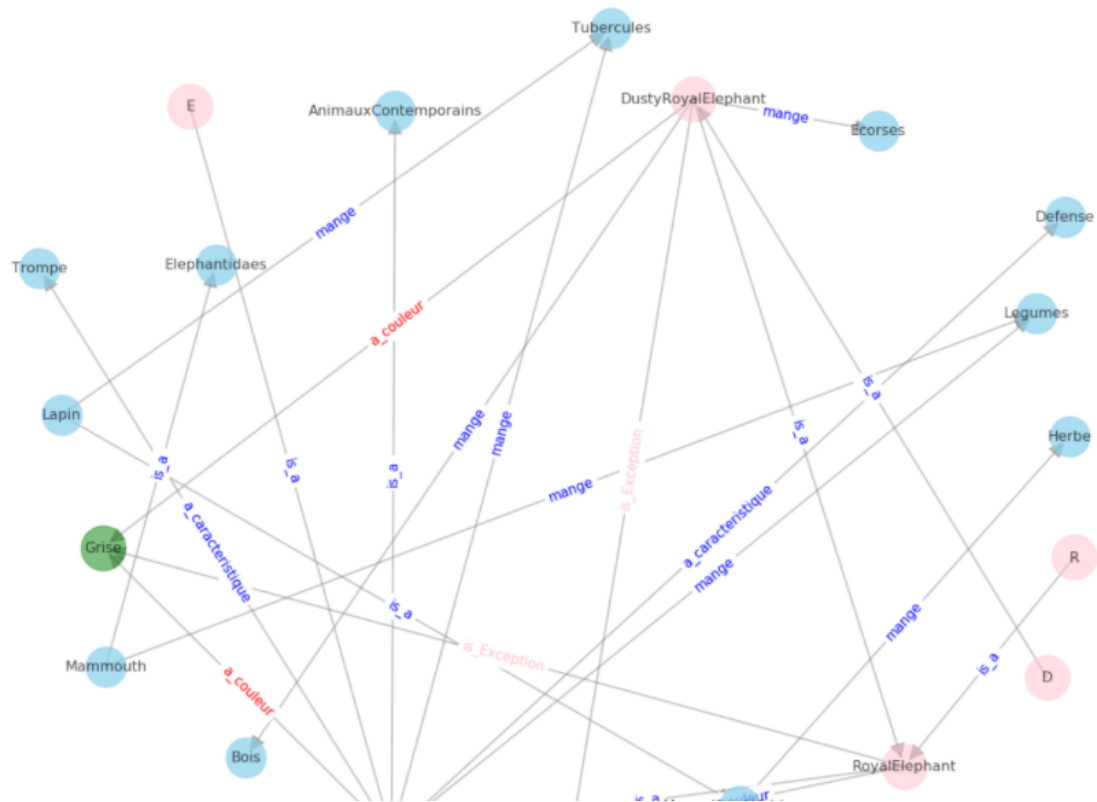


Figure 9: Propagation sans saturation de l'exemple 'Elephant.txt'.

## 2.2.3 Cas de plusieurs buts

Dans ce cas-là, le but est donné sous forme de liste contenant plusieurs informations. Ceci donne la possibilité d'exprimer la conjonction de plusieurs buts. La question posée dans ce notre exemple est 'Quels éléphants mangent des écorces et du bois?'.

```
Entrée [15]: #propagation à plusieurs buts
propagation_buts(x.matrice, x.noeds, x.relations, 'Elephant', 'mange')

Nombre de buts? 2
Votre but?Bois
Liste marquage 1: ['Elephant', 'RoyalElephant', 'E', 'DustyRoyalElephant', 'R']
Liste marquage 2: ['Bois']
SANS SATURATION: Solution trouvée, le lien: [['DustyRoyalElephant', 'Bois']]

Liste marquage 1: ['Elephant', 'RoyalElephant', 'E', 'DustyRoyalElephant', 'R', 'D']
Liste marquage 2: ['Bois']
AVEC SATURATION: Solution(s) trouvée(s), le(s) lien(s): [['DustyRoyalElephant', 'Bois']]

Votre but?Ecorces
Liste marquage 1: ['Elephant', 'RoyalElephant', 'E', 'DustyRoyalElephant', 'R']
Liste marquage 2: ['Ecorces']
SANS SATURATION: Solution trouvée, le lien: [['DustyRoyalElephant', 'Ecorces']]

Liste marquage 1: ['Elephant', 'RoyalElephant', 'E', 'DustyRoyalElephant', 'R', 'D']
Liste marquage 2: ['Ecorces']
AVEC SATURATION: Solution(s) trouvée(s), le(s) lien(s): [['DustyRoyalElephant', 'Ecorces']]
```

Figure 10: Les résultats obtenus dans le cas de deux (2) buts.

### 3 Seconde Étape. Implémenter l'algorithme d'héritage dans les réseaux sémantiques.

#### 3.1 Principe et fonctionnement de l'algorithme

**En entrée**, l'algorithme prend le réseau sémantique sous forme de structure dictionnaire, ainsi que le noeud à explorer.

**En sortie**, l'algorithme retourne l'ensemble des propriétés extraites relatives à ce noeud.

L'inférence par héritage sert à déduire des propriétés par transitivité. Il repose sur des liens de type 'est\_un' ou 'sorte\_de' reliant un concept à un autre concept plus élevé. Notre implémentation suit les étapes:

- Extraire la structure 'dictionnaire' décrite précédemment à partir du réseau sémantique.
- A partir du noeud en entrée, extraire les relations directes, tout en prenant en considération les exceptions, à savoir, élimination de toute propriétés notées 'is\_exception' au niveau du dictionnaire.
- Parcourir l'ensemble des noeuds présents dans les relations 'is\_a', et répéter le même traitement jusqu'à ne plus avoir de concept plus élevé.
- Enfin l'ensemble des propriétés extraites sont affichées.

Notre implémentation extrait les propriétés à tous les niveaux. Il nous est également possible d'avoir recours à une inférence avec saturation: extraire toutes les informations sur tous les noeuds, et ce en parcourant l'ensemble des clés de la structure dictionnaire.

```
def heritage_traitement(structure, noeud):
    rel_noeud = structure[noeud]
    exc = []
    isa = []
    isa_ = []

    isa = print_node(structure, noeud, exc)
    if len(isa) == 0:
        return

    if 'is_Exception' in rel_noeud:
        exc = rel_noeud['is_Exception']

    isa_ = isa
    while(len(isa_)>0):
        for x in isa_:
            if 'is_Exception' in structure[x]:
                for y in structure[x].get('is_Exception'):
                    if y not in exc:
                        exc.append(y)
            isa_.extend(print_node(structure, x, exc))
            isa_.remove(x)
        isa = isa_
    return isa
```

Figure 11: La fonction de l'héritage qui gère les exceptions.

## 3.2 Exemples d'exécution

Nous avons cherché à extraire les propriétés relatives au noeud 'RoyalElephant', nous remarquons clairement que les exceptions sont traitées avec succès.

```
#Heritage avec gestion d'exceptions
x = Traitement()
path = 'C:\\Users\\TRETEC\\Desktop\\M1SII\\Elephants.txt'
x.read_file(path)
structure = create_nodes_relations(x.noeds,x.relations,x.matrice)

noeud = 'RoyalElephant'
print('Ce que nous pouvons deduire concernant le noeud: '+noeud)
heritage_traitement(structure,noeud)
```

```
Ce que nous pouvons deduire concernant le noeud: RoyalElephant
is_a : Elephant
a_couleur : Blanche
is_a : AnimauxContemporains
is_a : MammifereHerbivore
mange : Legumes
mange : Tubercules
a_caracteristique : Trompe
a_caracteristique : Defense
mange : Herbe
```

Figure 12: Exemple d'héritage sur l'exemple "Elephant.txt" au noeud 'RoyalElephant'.

## 4 Troisième Étape. Génération de fichier contenant un réseau sémantique aléatoire et cohérent.

Le principe est de générer un fichier '.txt' respectant la structure décrite précédemment. Au tout début les nombres de concepts et de relations sont choisis aléatoirement, de telle façon à suivre une logique: pas trop de relations pour peu de noeuds. Nous avons également limité le nombre de noeuds pour pouvoir visualiser clairement notre réseau à la représentation.

Après la fixation de ces deux constantes, une matrice de relations est créée, ses éléments seront compris entre 0 et le nombre de relations total. A ce niveau les exceptions ne sont pas encore générées. Nous prenons bien le soin de vérifier qu'il n'y a pas de boucle directe (un noeud en relation avec lui-même), et qu'il n'y a pas non plus surcharge d'arcs, pour une lisibilité plus claire.

A ce niveau, la matrice est donnée en entrée à une fonction, qui se chargera d'y ajouter des exceptions entre deux (2) noeuds  $n1$  et  $n2$  choisis aléatoirement, en suivant la logique: si le noeud  $n2$  recevant l'exception n'a pas de prédécesseur, alors pas d'exception possible. S'il existe un chemin entre le noeud de départ  $n1$  et  $n2$ , alors l'exception peut-être ajoutée. Pour garder la clarté du réseau, nous limitons le nombre de fois par un nombre constant de chances.

```
def generation_file(chemin):
    f = open(chemin, 'w')
    for _ in range(10):
        value1 = randint(5, 8)
        noeuds = ''
        for i in range(value1):
            noeuds = noeuds+"noeud"+str(i+1)+"\n"
        noeuds = noeuds+"#\n"

    for _ in range(10):
        value2 = randint(5, 8)
        relations = 'is_a'\n'
        for i in range(value2):
            relations = relations+"relation_"+str(i+1)+"\n"
        relations = relations+"#\n"

    matrice=''
    m=generation_matrice(value1,value2)
    m=generer_exceptions(m)

    for i in range(len(m)):
        for j in range(len(m)):
            matrice = matrice+str(m[i][j])+" "
        matrice = matrice[:-1]
        matrice = matrice+";\n"

    contenu = noeuds+relations+matrice.rstrip()

    f.write(contenu)
    f.close()
```

Figure 13: La fonction de génération du fichier '.txt' contenant le réseau sémantique.

```
#génération d'un fichier aléatoirement
generation_file("C:\\Users\\DELL Precision 3510\\Desktop\\M1SII\\S2\\REP_CONNAIS\\LDESCRIPTIVE\\TP3\\myfile.txt")

y = Traitement()
y.read_file('C:\\Users\\DELL Precision 3510\\Desktop\\M1SII\\S2\\REP_CONNAIS\\LDESCRIPTIVE\\TP3\\myfile.txt')
structure_y = create_nodes_relations(y.noeds,y.relations,y.matrice)
#affichage de la structure générée
print_dict_nodes(structure_y)

Noeud: noeud1
Relations: relation_2 :['noeud3']
Relations: relation_1 :['noeud4', 'noeud6']
Relations: relation_3 :['noeud7']

Noeud: noeud2
Relations: relation_1 :['noeud1']
Relations: is_a :['noeud3']
Relations: is_Exception :['noeud4', 'noeud5']
Relations: relation_2 :['noeud6', 'noeud7']

Noeud: noeud3
Relations: relation_2 :['noeud4']
Relations: relation_1 :['noeud5']
Relations: relation_3 :['noeud7']

Noeud: noeud4
Relations: relation_1 :['noeud5', 'noeud6']

Noeud: noeud5
Relations: relation_1 :['noeud1']
Relations: is_a :['noeud7']

Noeud: noeud6
Relations: relation_3 :['noeud3']
Relations: relation_4 :['noeud5', 'noeud7']

Noeud: noeud7
Relations: is_a :['noeud4']
```

Figure 14: Affichage de la structure dictionnaire du fichier généré précédemment.

```
#affichage du graphe correspondant
arcs_y,G_y=creategraph_exception(y.matrice,y.noeds,y.relations)
```

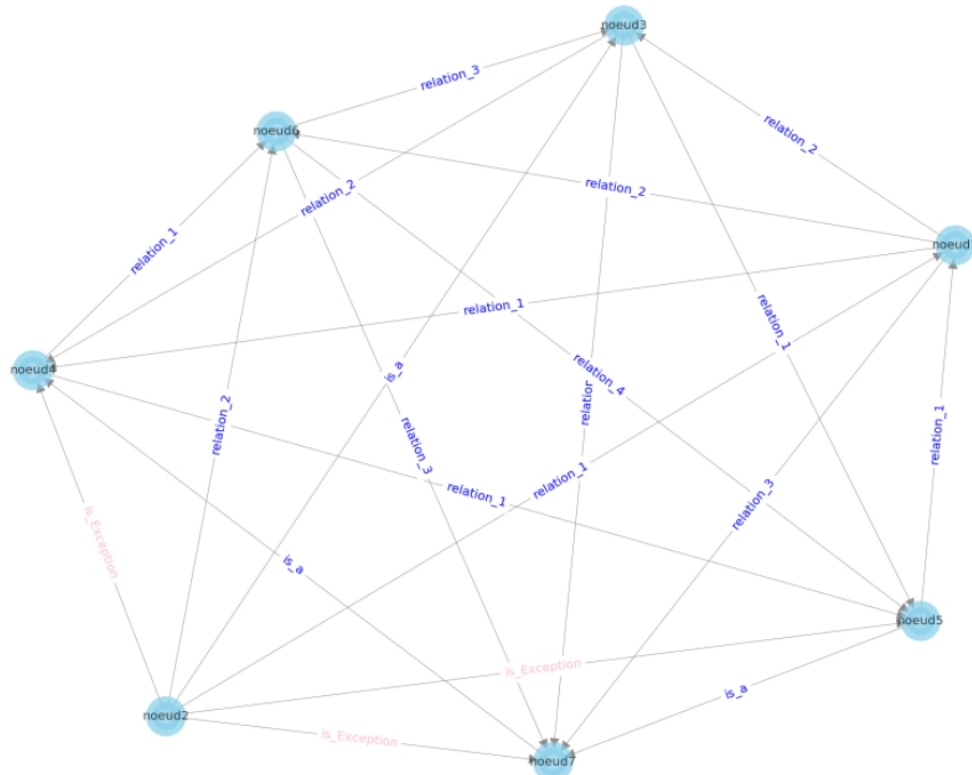


Figure 15: Affichage graphique du réseau sémantique généré précédemment.

```
noeud = 'noeud1'
print('Ce que nous pouvons deduire concernant le noeud: '+noeud)
heritage_traitement(structure_y,noeud)

Ce que nous pouvons deduire concernant le noeud: noeud1
relation_2 : noeud3
relation_1 : noeud4
relation_1 : noeud6
relation_3 : noeud7
```

---

Figure 16: Application de la fonction d'héritage sur l'un des noeuds du réseau pour démontrer la prise en compte des exceptions et la cohérence de la génération.