DEVOIR 4

EN ÉQUIPE

Conception d'une application de TALN

Auteurs

Alex Lebrun - alex.lebrun.1@ulaval.ca

Walter Bonetti - walter.bonetti.1@ulaval.ca

Isabelle Eysseric - isabelle.eysseric.1@ulaval.

Mathieu ALain - mathieu.alain.2@ulaval.ca

DEVOIR 4

EN ÉQUIPE

Sommaire

SYSTÈME À BASE DE CONNAISSANCES

Traitement Automatique du Langage Naturel

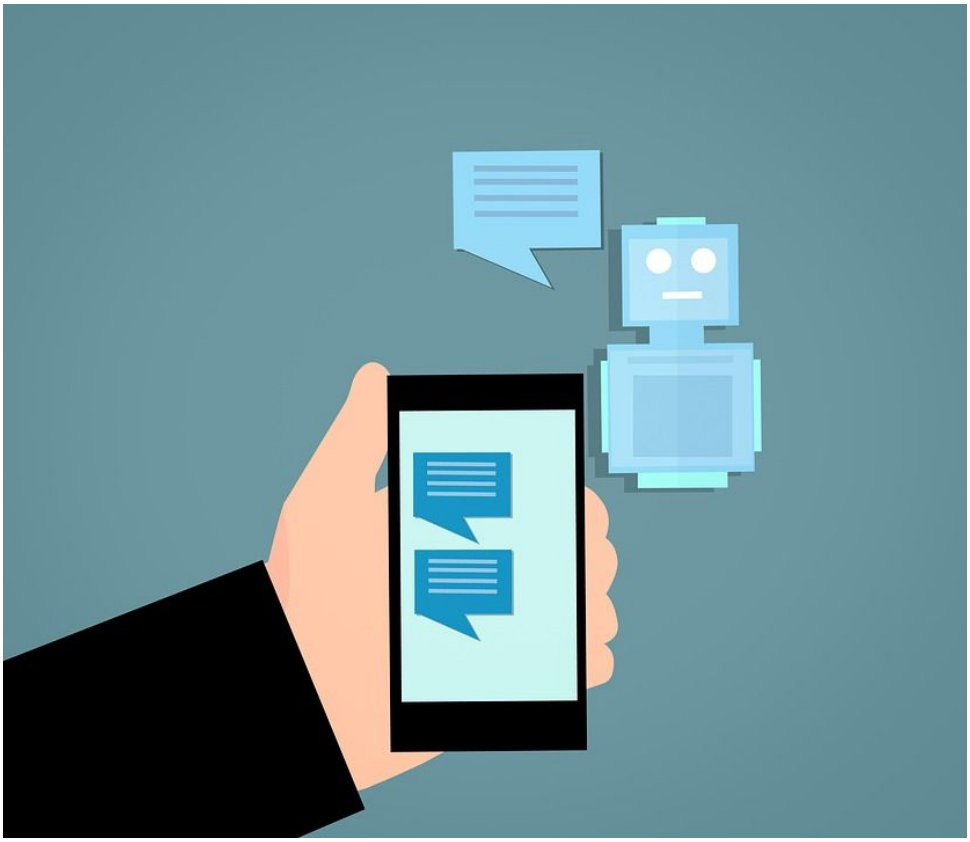

Figure 1 *: Dessin représentant une personne textant avec un système expert, ici un chatbot [1].*

1. Le TALN: Introduction

Nous avons conçu un système expert permettant le traitement automatique de la langue naturelle (TALN[2]) sous forme de questions - réponses[3]. Ce système utilise l'analyse syntaxique et l'interprétation sémantique pour répondre correctement aux questions de l'utilisateur.

Le programme a été conçu en langage de programmation Prolog[4] pour permettre la représentation symbolique des phrases et des règles de grammaires nécessaires.

La figure ci-dessous représente bien la structure de notre programme, excepté le web que nous n'utilisons pas dans la version initiale.

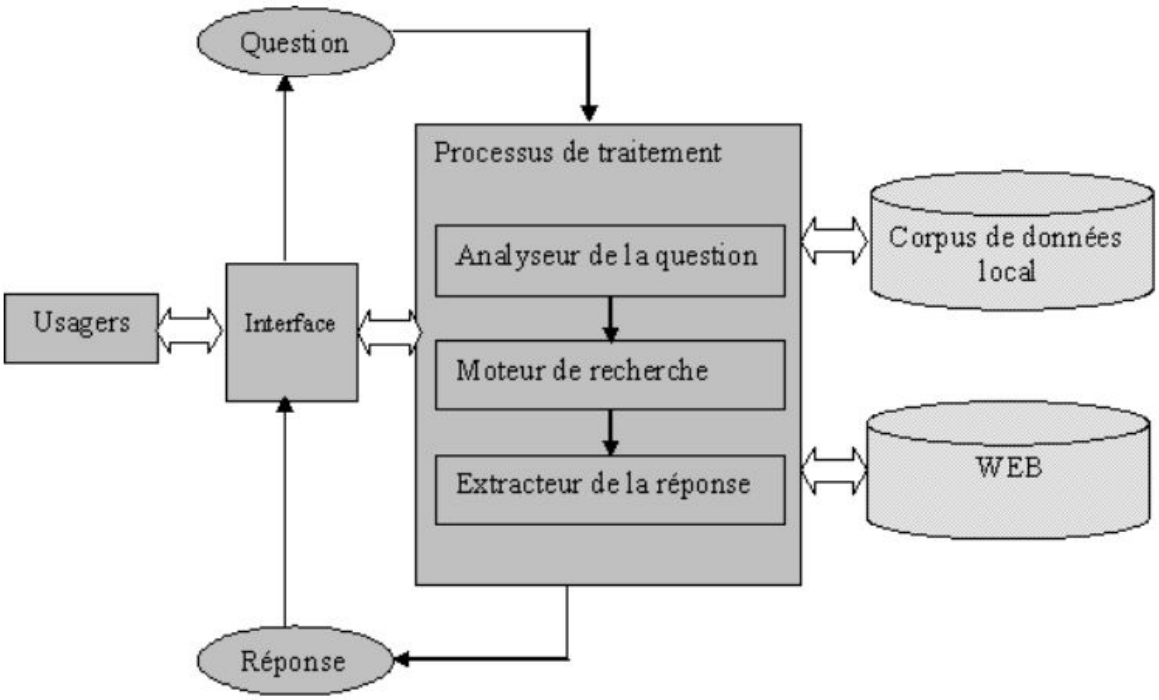

***Figure 2** : Architecture de notre système de question - réponse [5].*

2. La structure des phrases

2.1 Introduction

Dans cette section, nous identifierons les structures syntaxiques des phrases. Nous déterminerons les relations des groupes de mots ensembles comme sujet-verbe ou bien encore verbe-objet. Cette analyse nous servira de base pour l'interprétation sémantique.

L'analyse syntaxique[6] vérifie que les expressions soient bien formées et détermine une structure linguistique avec nos relations établies. Nous utiliserons une grammaire augmentée pour l'analyse syntaxique ainsi que l'interprétation sémantique[7].

2.2 Ensemble de phrases utilisées

Nous vous fournissons, dans cette section, l'ensemble de phrases utilisables dans ce programme.

Phrase 1 : `un chien est un mammifere`

Phrase 2 : `pégase est un mammifere`

Phrase 3 : `tout mammifere est un animal`

Phrase 4 : `est ce qu'un chien est un animal`

Phrase 5 : `est ce que pégase est un mammifere`

Il est clair qu'il est difficile de réellement tester notre système de TALN avec uniquement cinq phrases et les mots qu'elles contiennent. C'est pourquoi nous avons ajouté quelques mots pour bien tester le fonctionnement du système. Voici donc la liste complète des mots disponibles:

Article :	`un`
Adverbe :	`tout`
Noms Propres :	`basilic, marsupilami, pegase`
Noms Communs :	`chien, lezard, mammifere, reptile, animal`
Verbe:	`est`

Nous avons cependant limité la forme que peuvent prendre les phrases pour éviter de complexifier le système inutilement.

Nos phrases seront: `un` (nom commun) `est un` (nom commun)
(nom propre) `est un` (nom commun)
`tout` (nom commun) `est un` (nom commun)
`est ce que un` (nom commun) `est un` (nom commun)
`est ce que un` (nom propre) `est un` (nom commun)

3.3 L'analyse syntaxique

La grammaire augmentée est utilisée à l'aide d'une grammaire à clauses définies (DCG)[8] pour mieux traiter les problèmes d'accords entre le sujet et le verbe par exemple. Elle nous permet d'utiliser l'inférence logique pour l'analyse syntaxique mais aussi pour l'interprétation sémantique.

On peut voir ci-dessous l'arbre syntaxique avec la grammaire attribuée pour le traitement automatique d'une des phrases du programme.

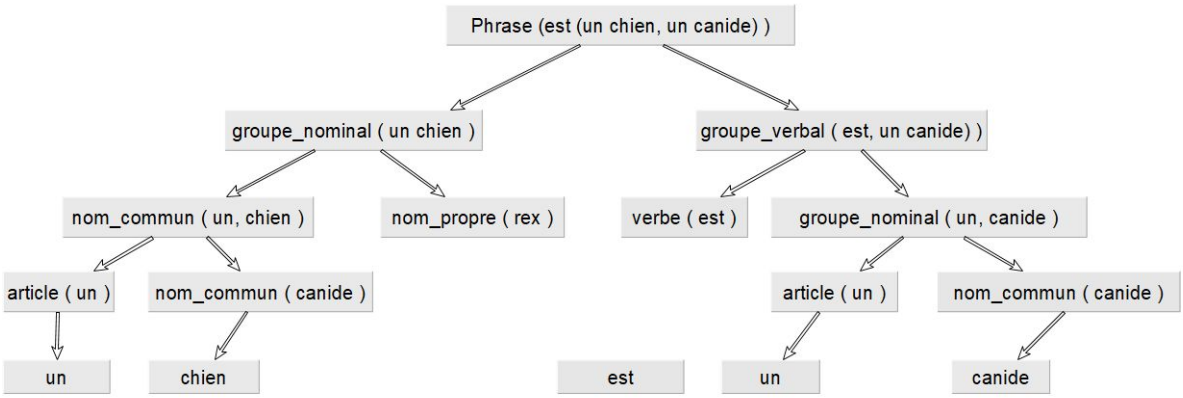

Figure 3 *: Arbre syntaxique de la phrase "un chien est un canide".*

Signification des attributs de cet arbre :

```
phrase( PREDICAT )
groupe _nominal ( SUJET )
groupe _verbal ( ACTION, OBJET )
nom_commun ( chien ) -->  [ chien ]
nom_commun ( canide ) -->  [ canide ]
nom_propre ( rex )  -->  [rex]
article ( un )  -->  [ un ]
verbe ( est )  -->  [ est ]
```

À noter *que le nom des attributs est un peu différent du code du programme. Dans le code, les noms ont été réduits par pure simplicité. Ici les noms sont dans leur forme la plus longue par soucie de compréhension.*

3.3 L'interprétation sémantique

Maintenant, regardons en détail, pour chaque phrase possible du programme, l'interprétation sémantique du programme gérée par notre grammaire augmentée.

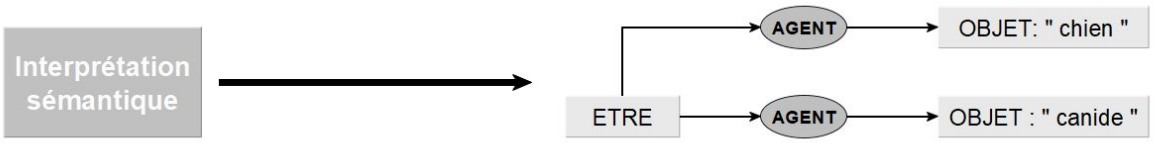

Figure 4 *: Interprétation sémantique de la phrase "un chien est un canide" représenté sous forme de graphe conceptuel.*

Phrase 1 : **`un chien est un mammifere`**

```
%Phrase de type:  un chien est un mammifere
pA(ENONCE)-->  gnA(OBJET),gv(_,AGENT),
               {ENONCE=..[AGENT,OBJET]}.

%Regles pour constituer une phrase
gnA(OBJET) --> article, nom_commun(OBJET).
gv(ACTION,OBJET) --> verbe(ACTION), gnA(OBJET).
```

Phrase 2 : **`pegase est un mammifere`**

```
%Phrase de type:  marsupilami est un mammifere
pA(ENONCE)-->  gnA(OBJET),gv(_,AGENT),
               {ENONCE=..[AGENT,OBJET]}.

%Regles pour constituer une phrase
gnA(OBJET) --> nom_propre(OBJET).
gnA(OBJET) --> article, nom_commun(OBJET).
gv(ACTION,OBJET) --> verbe(ACTION), gnA(OBJET).
```

Phrase 3 : **tout mammifere est un animal**

```
%Phrase de type:  tout mammifere est un mammifere
pB(ENONCE) --> gnB(OBJET),gv(ACTION,AGENT),
                {ENONCE=..[ACTION,AGENT,OBJET]}.

%Regles pour constituer une phrase
gnB(OBJET) --> adverbe, nom_commun(OBJET)
gnA(OBJET) --> article, nom_commun(OBJET).
gv(ACTION,OBJET) --> verbe(ACTION), gnA(OBJET).
```

Phrase 4 : **est ce que un chien est un animal**

```
%Phrase de type:  est ce que un mammifere est un animal
pC(ENONCE) --> gquestion, gnA(AGENT), gv(ACTION,OBJET),
                {ENONCE=..[ACTION,AGENT,OBJET]}.

%Regles pour constituer une phrase
gquestion --> [est,ce,que].
gnA(OBJET) --> article, nom_commun(OBJET).
gv(ACTION,OBJET) --> verbe(ACTION), gnA(OBJET).
```

Phrase 5 : **est ce que pegase est un mammifere**

```
%Phrase de type:  est ce que pegase est un mammifere
pC(ENONCE) --> gquestion, gnA(AGENT), gv(ACTION,OBJET),
                {ENONCE=..[ACTION,AGENT,OBJET]}.

%Regles pour constituer une phrase
gquestion --> [est,ce,que].
gnA(OBJET) --> nom_propre(OBJET).
gnA(OBJET) --> article, nom_commun(OBJET).
gv(ACTION,OBJET) --> verbe(ACTION), gnA(OBJET).
```

Ici, nous avons la liste des mots qui est reconnue par le systèmes à base de connaissances.

```
% Liste des mots reconnus
article --> [un].
adverbe --> [tout].
nom_propre(basilic) --> [basilic].
nom_propre(marsupilami) --> [marsupilami].
nom_propre(pegase) --> [pegase].
nom_commun(chien) --> [chien].
nom_commun(lezard) --> [lezard].
nom_commun(mammifere) --> [mammifere].
nom_commun(reptile) --> [reptile].
nom_commun(animal) --> [animal].
verbe(est) --> [est].
```

Et là, l'arbre syntaxique avec son interprétation sémantique sous forme de graphe conceptuel. L'utilisateur entre une phrase qui va être analysée puis interprétée par le programme.

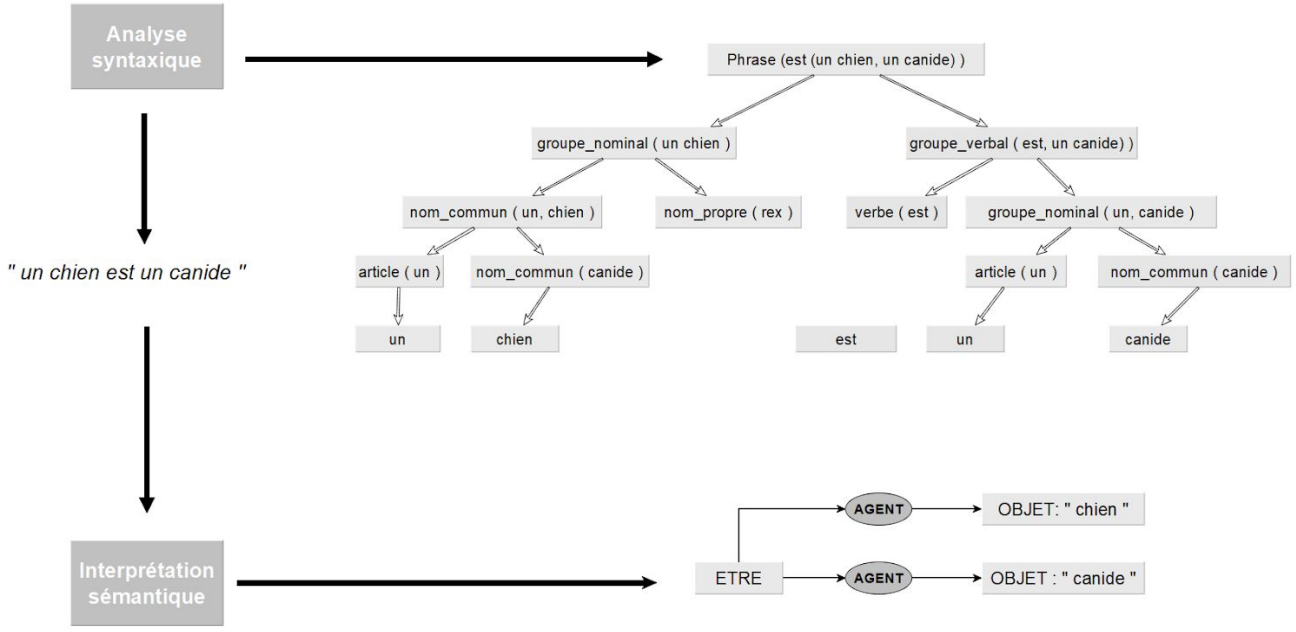

Figure 5 *: Analyse syntaxique et interprétation sémantique de la phrase "un chien est un canide".*

3. L’analyse

3.1 Introduction

Après l’analyse syntaxique et l’interprétation sémantique vues dans la partie précédente, nous avons besoin d’un analyseur .

L’analyse sémantique[9] nous permet de vérifier la sémantique de la phrase entrée par l’utilisateur, ajoute des faits à notre arbre syntaxique et construit également la table des symboles.

Voici l’analyse sémantique sous trois formes différentes dans notre programme:

a) `…. est un ….`
b) `tout …. est un ….`
c) `est ce que …. est un ….`

3.2 Analyse de la forme (a)

L'analyse de la forme " …. `est un` …. " analyse les phrases pour créer des faits qui sont insérés dans la base par des assertions avec le prédicat `assert/1`.

Par exemple, la phrase "`pegase est un mammifere`" donne `mammifere(chien).`

Interface :	Base de faits :
`rex est un chien`	`chien(rex)`

Code :

```
% Crée un element X(premier element) de la categorie Y(2e element)
% Equivalent de Y(X).
analyseA :-
    readln(P),
    (   pA(A,P,[])
           -> assert(A),writeln("ok")
           ; writeln("Phrase non reconnue")
    ),nl.
```

3.3 Analyse de la forme (b)

L'analyse de la forme " `tout …. est un ….` " analyse les phrases pour créer des règles. Elle les insère par des assertions.

Par exemple, la phrase "`tout mammifere est un animal`" donne la règle animal`(X) :- mammifere(X).`

Interface:	Base de faits:
`Tout mammifere est un animal`	animal`(X) :- mammifere(X).`

Code:

```
% Fait un lien est() entre:
% la sous-categorie Y(premier element) et la catégorie Z(2e element)
% Equivalent de Z(X):-Y(X).
analyseB :-
    readln(P),
    (   pB(A,P,[])
          -> fonctionVarB(A)
          ; writeln("Phrase non reconnue")
    ),nl.


% Fonction pour associer des variables à des items
% Fait partie de analyseB
fonctionVarB(est(Y,X)) :-
   concat(Y,'(X):-',V1),
   concat(V1,X,V2),
   concat(V2,'(X)',CommandeStr),
   term_to_atom(Commande, CommandeStr),
   assert(Commande),
   writeln("ok").
```

3.4 Analyse de la forme (c)

L'analyse de la forme " `est-ce que …. est un ….` " analyse les phrases comme des requêtes auxquelles le programme répondra `oui` ou `non` selon la base des faits.

Par exemple, la phrase "`est ce que un chien est un animal`" équivaut à la requête en Prolog `?- animal(chien).`

Interface:

```
est ce que pegase est
un animal
```

Requête:

```
?- animal(pegase).
```

Code:

```
% Répond "oui" si:
% l'element X(premier element) est dans la categorie Y(2e element)
% Equivalent de ?- Y(X).
analyseC :-
    readln(P),
    (   pC(A,P,[])
          -> (questionX(A)
                  -> writeln("oui")
                  ; writeln("non")
              )
          ; writeln("Phrase non reconnue")
    ),nl.

% Pour faire le test
% true si X est une sorte de Y, false sinon
questionX(est(X,Y)):-
   concat(Y,'(',V1),
   concat(V1,X,V2),
   concat(V2,')',CommandeStr),
   term_to_atom(Commande, CommandeStr),
   call(Commande).
```

4. Interface

4.1 Prédicat go/0

Le prédicat `go/0` fait appel à trois autres prédicats :

- `analyseA`,
- `analyseB`,
- `analyseC`.

Chacun des trois prédicats appelés dans le prédicat `go/0`, analyse une des trois formes données à la suite l'une de l'autre.

Code:

```
%Programme principal
go :-
  writeln("Ecrire votre phrase sans majuscule et sans ponctuation"),
  writeln("Sous la forme:  un chien est un mammifere"),
  analyseA,
  writeln("Sous la forme:  pegase est un mammifere"),
  analyseA,
  writeln("Sous la forme:  tout mammifere est un animal"),
  analyseB,
  writeln("Sous la forme:  est ce que un chien est un animal"),
  analyseC,
  writeln("Sous la forme:  est ce que pegase est un mammifere"),
  analyseC.
```

4.2 Prédicat analyseA/0, analyse/0 et analyseC/0

Ces trois prédicats analysent la phrase saisie et donne la réponse :

- "`Ok`" pour l'`analyseA` et l'`analyseB`

```
?- go.
Ecrire votre phrase sans majuscule et sans ponctuation
Sous la forme:  un chien est un mammifere
|: un chien est un mammifere
ok
```

***Figure 6** : Réponse "ok" pour les phrase de la forme A et B*

- La réponse à la requête pour l'`analyseC`, soit "`oui`" ou "`non`".

```
Sous la forme:  est ce que pegase est un mammifere
|: est ce que pegase est un mammifere
oui
```

***Figure 7** : Réponse "ok" pour les phrase de la forme A et B*

4.3 Prédicat readln/1

Pour saisir une phrase, on utilise le prédicat prédéfini `readln/1`.

Par exemple:

```
?- readln(P)
|: rex est un chien
P = [rex, est, un, chien]
```

5. Résultats

Nous allons vous présenter dans cette section les tests effectués pour vous montrer les résultats de notre analyse sémantique des phrases.

Ensuite nous discuterons des résultats obtenues à la suite de nos tests.

Et enfin, nous conclurons en vous donnant notre état de satisfaction relatif à cette application et d'éventuels pistes d'amélioration.

5.1 Présentation des tests

Dans cette section nous allons tester les différentes facettes de notre programme. Pour rappeler les éléments mentionnés dans les sections précédentes, voici les différents mots qui nous serviront à tester le système:

Article :	`un`
Adverbe :	`tout`
Noms Propres :	`basilic, marsupilami, pegase`
Noms Communs :	`chien,lezard,mammifere,reptile,animal`
Verbe:	`est`

Nos phrases seront: `un` (nom commun) `est un` (nom commun)
(nom propre) `est un` (nom commun)
`tout` (nom commun) `est un` (nom commun)
`est ce que un` (nom commun) `est un` (nom commun)
`est ce que` (nom propre) `est un` (nom commun)

%test de la fonction principale:

go.

```
?- go.
Ecrire votre phrase sans majuscule et sans ponctuation
Sous la forme:  un chien est un mammifere
|: un chien est un mammifere
ok

Sous la forme:  pegase est un mammifere
|: pegase est un mammifere
ok

Sous la forme:  tout mammifere est un animal
|: tout mammifere est un animal
ok

Sous la forme:  est ce que un chien est un animal
|: est ce que un chien est un animal
oui

Sous la forme:  est ce que pegase est un mammifere
|: est ce que pegase est un mammifere
oui

true.

?-
```

Figure 8 *: fonction principale dans la console*

```
%tests pour :  un (nom commun) est un (nom commun)

%test de la phrase choisie
analyseA.
|: un chien est un mammifere
ok
mammifere(chien).                          true

%test d'une phrase similaire
analyseA.
|: un lezard est un reptile
ok
reptile(lezard).                           true

%test d'une phrase incorrecte
analyseA.
|: un rat est petit                        Phrase non reconnue.
```

```
?- analyseA.
|: un chien est un mammifere
ok

true.

?- analyseA.
|: un lezard est un reptile
ok

true.

?- analyseA.
|: un rat est petit
Phrase non reconnue

true.

?- mammifere(chien).
true .

?- reptile(lezard).
true.

?-
```

Figure 9 *: tests pour: un (nom commun) est un (nom commun)*

```
%tests pour :  (nom propre) est un (nom commun)

%test de la phrase choisie
analyseA.
|: pegase est un mammifere
ok
mammifere(pegase).                          true

%test d'une phrase similaire
analyseA.
|: basilic est un reptile
ok
reptile(basilic)                            true

%test d'une phrase incorrecte
analyseA.
|: rat est un chat                          Phrase non reconnue.
```

```
?- analyseA.
|: pegase est un mammifere
ok

true.

?- analyseA.
|: basilic est un reptile
ok

true.

?- analyseA.
|: rat est un chat
Phrase non reconnue

true.

?- mammifere(pegase).
true .

?- reptile(basilic).
true.

?-
```

Figure 10 *: tests pour la phrase de la forme (nom propre) est un (nom commun)*

```
%tests pour :  tout (nom commun) est un (nom commun)

%test de la phrase choisie
analyseB.
|: tout mammifere est un animal
ok

%test d'une phrase similaire
analyseB.
|: tout reptile est un animal
ok

%test d'une phrase incorrecte
analyseB.
|: tout rat est chien                    Phrase non reconnue.

%test d'une phrase similaire
%(réponse si les tests A sont faits)
?- animal(X).                            X = chien
```

```
?- analyseB.
|: tout mammifere est un animal
ok

true.

?- analyseB.
|: tout reptile est un animal
ok

true.

?- analyseB.
|: tout rat est chien
Phrase non reconnue

true.

?- animal(X).
X = chien ;

?- 
true.

?- animal(X).
X = chien ;
```

Figure 11 *: tests pour: tout (nom commun) est un (nom commun)*

```
%tests pour : est ce que un (nom commun) est un (nom commun)

%test de la phrase choisie (deux niveaux de profondeur)
analyseC.
|: est ce que un chien est un animal          oui

%test d'une phrase incorrecte
analyseC.
|: est que reptile est un animal              Phrase non reconnue.
```

```
?- analyseC.
|: est ce que un chien est un animal
oui

true.

?- analyseC.
|: est ce que reptile est un animal
Phrase non reconnue

true.

?-
```

Figure 12 *: tests pour: est ce que un (nom commun) est un (nom commun)*

```
%tests pour :  est ce que (nom propre) est un (nom commun)

%test de la phrase choisie (deux niveaux de profondeur)
analyseC.
|: est ce que pegase est un animal          oui

%test d'une phrase similaire (direct)
analyseC.
|: est ce que basilic est un animal          oui

%test d'une phrase incorrecte
analyseC.
|: est que un pegase est un animal          Phrase non reconnue.
```

```
?- analyseC.
|: est ce que pegase est un animal
oui

true.

?- analyseC.
|: est ce que basilic est un animal
oui

true.

?- analyseC.
|: est ce que un pegase est un animal
Phrase non reconnue

true.

?-
```

Figure 13 *: tests pour: est ce que (nom propre) est un (nom commun)*

```
%Tests qui doivent retourner "false" ou "non"

%test de l'analyseA pour une relation inexistante
%(noms utilisés)
mammifere(lezard).                                    false

%test de l'analyseA pour une relation inexistante
%(nom jamais utilisé)
mammifere(marsupilami).                               false

%test de l'analyseB pour une relation inexistante
animal(marsupilami).                                  false

%test de l'analyseC pour une relation inexistante
analyseC.
|: est ce que marsupilami est un mammifere            non

%test de l'analyseC pour le sens des relation
analyseC.
|: est ce que un animal est un mammifere              non
```

```
?- mammifere(lezard).
false.

?- mammifere(marsupilami).
false.

?- animal(marsupilami).
false.

?- analyseC.
|: est ce que marsupilami est un mammifere
non

true.

?- analyseC.
|: est ce que un animal est un mammifere
non

true.

?-
```

***Figure 14** : tests pour les fonctions d'analyse*

5.2 Discussion de l'expérience

Nous sommes relativement satisfaits du résultat obtenu. Les phrases sont reconnues et les fonctions d'analyse effectuent les tâches demandées. Cependant il est clair que de multiples façons existent pour régler ce problème et nous n'avons pas nécessairement choisi la plus optimale.

Il aurait été intéressant de pouvoir expérimenter avec d'autres méthodes ou même approfondir le travail sur le DCG pour le rendre plus performant. Nous n'avons qu'effleurer la surface de ce que ce module peut réaliser.

5.3 Amélioration possibles

Pour plus de flexibilité, nous aurions aimé agrandir la base des mots reconnus. Le plus grand handicap de ce type de système est que chaque mot doit être reconnu pour être utilisable. Donc soit on code tout à la main, soit on trouve une façon d'intégrer un dictionnaire d'une source externe. Cette option aurait été intéressante, mais trop demandante pour un Travail Pratique en fin de session.

Notre système ajoute des relations facilement, mais il y a une faille importante. L'erreur n'est pas permise. D'aucune façon possible, on ne peut enlever une relation qui serait fausse. L'utilisateur ne peut simplement pas revenir en arrière et corriger son erreur. L'ajout d'une fonction d'effacement tel "`retract/1`" serait d'une importance capitale pour avoir un système complet.

Notre système est à une très petite portée pour le moment. Chaque relation est ajoutée une par une. Si jamais une base de classification complexe et diversifiée, comme le classement des insectes, devait être entrée par ce système, ce serait un travail long et fastidieux. L'ajout de conjonction tel "et" permettrait de sauver un temps considérable.

DEVOIR 4

Une fonction pour tester si un groupe est vide serait utile, de même qu'une fonction pour faire une liste des éléments qui ne sont pas classés.

En résumé, bien que particulièrement utile et beaucoup plus facile d'utilisation pour remplir une base de connaissances, un tel système peut rapidement devenir difficile à gérer. C'est pourquoi de bien faire les fondations d'un pareil système doit être fait minutieusement en prévision des ajouts à venir.

Ce TP nous a permis d'apprivoiser ce système sans devoir le pousser trop loin. Ceci nous a permis de voir les éventuelles limites d'un système TALN sans pour autant avoir le casse-tête de tout régler.

6. Bibliographie

[1] Image du site Netcore.in, « Chatbot A new language of Business »

[2] Site web Wikipedia, « Traitement automatique du langage naturel »

[3] Site web Wikipedia, « Système de questions-réponses »

[4] Site web Wikipedia, « Prolog »

[5] Image du site HAL archives-ouvertes.fr, «Systèmes de questions-réponses collaboratif et interactif»

[6] Site web Wikipedia, « Analyse syntaxique »

[7] Livre *Intelligence Artificielle 3e édition* de Stuart Russel et Peter Norvig, « Grammaires augmentées et interprétation sémantique » page [948-958].

[8] Site web Wikipedia, «definite clause grammar (DCG)»

[9] Site web Wikipedia, « Analyse sémantique »

7. Annexes

Vous trouverez en annexe, l'énoncé qui nous a guidés dans la conception de notre application de traitement automatique du langage, ainsi qu'un barème nous permettant de ne pas oublier d'éléments à intégrer.

6.1 Énoncé

Travail pratique : questionnements logiques en langage naturel

Ce travail compte pour 5% de la note finale.

Échéance : le dimanche 16 Décembre 2018 (23h59)

Objectifs d'apprentissage :

À l'issue de ce travail, un étudiant devrait être capable de :

- résoudre un problème concret de traitement automatique du langage naturel;
- identifier les enjeux du traitement du langage naturel sur un problème concret;
- analyser les limites d'une solution à un problème concret de traitement du langage naturel.

Description du contexte du travail pratique :

Écrire un programme Prolog qui devra comprendre (par une analyse sémantique) des phrases en français de la forme suivante et répondre correctement à la question logique :

a) ….. est un ….
b) tout …. est un ….
c) est ce que ….. est un ….

Les phrases de la forme (a) seront analysées pour créer des faits qui seront insérés dans la base par des assertions (en utilisant le prédicat assert/1). Par exemple, la phrase «rex est un chien » donnera le fait : chien(rex).

Les phrases de la forme (b) seront analysées pour créer des règles, elle aussi insérées par des assertions. Par exemple, la phrase « tout chien est un canidé » donnera la règle :

canide(X) :- chien(X).

Les phrases de la forme (c) seront analysées comme des requêtes auxquelles le programme répondra oui ou non selon l'état de la base de faits. Ainsi la phrase « Est-ce-que Rex est un canidé?» reviendra à poser la requête :

?- canide(rex).

Le programme répondra oui ou non selon si la réponse à cette requête.

Voici deux exemples de fonctionnement :

?- go. rex est un chien ok tout chien est un canidé ok est ce que rex est un canidé oui yes	?- go. rex est un chien ok tout chien est un canidé ok est ce que bill est un canidé non yes

Le programme à construire en langage Prolog consiste en un prédicat go/0 qui fait appel à 3 autres prédicats, chacun analysant une des 3 formes données à la suite l'une de l'autre : analyseA, analyseB, analyseC.

Ces 3 prédicats analysent la phrase saisie et donne la réponse :

- ok pour analyseA et analyseB,
- la réponse à la requête pour analyseC, soit oui ou non.

La saisie d'une phrase se fait à l'aide du prédicat prédéfini readln/1. Par exemple :

```
? - readln(P).
| : rex est un chien
P=[rex, est, un, chien]
```

Travail à réaliser :

1- **Ensemble de phrases utilisées (10 %)** : Constituer 5 (au minimum) ensembles de phrases contenant les 3 formes (a), (b) et (c). Il est possible de faire des variantes du type : « est une ».

2- **Analyse de la forme (a) (15 %) :** Programmer en Prolog analyseA en utilisant une grammaire attribuée et l'opérateur DCG.

3- **Analyse de la forme (b) (15 %) :** Programmer en Prolog analyseB en utilisant une grammaire attribuée et l'opérateur DCG.

4- **Analyse de la forme (c) (20 %) :** Programmer en Prolog analyseC en utilisant une grammaire attribuée et l'opérateur DCG.

5- **Interface (10 %) :** Programmer le prédicat go/0 en langage Prolog selon les directives données précédemment.

6- **Résultats (20%) :**

- Faire des tests pour montrer les résultats d'analyse de phrases (à insérer dans le rapport).
- Discuter les résultats obtenus : Êtes-vous satisfait de votre travail ? Comment pourriez-vous l'améliorer ? Est-ce que de nouvelles phrases pourraient facilement être interprétées ?

Les 10% restants sont dédiés à l'appréciation globale du travail remis (présentation et expression écrite).

Ne pas oublier de valider votre travail à l'aide de la grille d'autoévaluation fournie à l'annexe 1.

Modalités de remise de ce travail :

En plus du programme Prolog (fichier PL), vous devez remettre un rapport. Ce rapport doit avoir comme d'habitude une page couverture, une introduction et une conclusion. Le rapport contiendra 2 parties : les 5 ensembles de phrases utilisés et la partie Résultats.

L'ensemble du travail est à remettre en format électronique via le Portail des cours. Les travaux remis en retard ne seront pas corrigés.

7.2 Barème

Annexe 1. Grille d'autoévaluation

Conception (90%)				
Phrases (10%)	Un ensemble d'au moins 5 ensembles de phrases sont indiquées et utilisables dans le programme.	Un ensemble d'au moins 5 ensembles de phrases sont indiquées mais ne sont pas toutes utilisables dans le programme.	Moins de 5 ensembles de phrases sont indiquées et utilisables dans le programme.	On ne sait pas vraiment quelles sont les phrases utilisables par le programme.
AnalyseA (15%)	Le prédicat permet d'analyser les phrases de la forme (a) et faire l'assertion du fait correspond.	Le prédicat permet d'analyser les phrases de la forme (a) mais l'assertion du fait correspondant ne se fait pas.	Le prédicat ne permet d'analyser les phrases de la forme (a) au complet et ne fait pas l'assertion du fait correspondant.	Aucune analyse des phrases de la forme (a).
AnalyseB (15%)	Le prédicat permet d'analyser les phrases de la forme (b) et faire l'assertion du fait correspond.	Le prédicat permet d'analyser les phrases de la forme (b) mais l'assertion du fait correspondant ne se fait pas.	Le prédicat ne permet d'analyser les phrases de la forme (b) au complet et ne fait pas l'assertion du fait correspondant.	Aucune analyse des phrases de la forme (b).
AnalyseC (20%)	Le prédicat permet d'analyser les phrases de la forme (c) et répond à la question correctement.	Le prédicat permet d'analyser les phrases de la forme (c) mais ne répond pas à la question correctement.	Le prédicat permet d'analyser les phrases de la forme (c) mais ne donne pas de réponse	Le prédicat ne permet d'analyser les phrases de la forme (c) ou est absent.
Interface (10%)	Le prédicat go/0 est implanté selon les directives et fonctionne sans erreur.	Le prédicat est implanté selon les directives mais fonctionne avec erreur.	Le prédicat est implanté mais ne fonctionne pas du tout.	Le prédicat n'est pas implanté

Résultats (20%)	Les tests sont présentés ainsi qu'une discussion de l'expérience et des améliorations possibles.	Les tests sont présentés mais il manque une discussion.	Les tests ne sont pas présentés mais il y a une discussion	Il n'y a pas de tests, ni de discussion.

Appréciation globale (10%)				
Expression écrite (5%)	Le rapport ne contient aucune faute (vocabulaire, grammaire, syntaxe, etc.).	Le rapport ne contient pas plus d'une dizaine de fautes (vocabulaire, grammaire, syntaxe, etc.).	Le rapport ne contient pas plus de 5 fautes par page (vocabulaire, grammaire, syntaxe, etc.).	Le rapport contient plus de 5 fautes par page (vocabulaire, grammaire, syntaxe, etc.).
Présentation du rapport (5%)	Tous les éléments demandés sont présents et le rapport est paginé et contient une page de garde.	Tous les éléments demandés sont présents, mais le rapport n'est pas paginé ou ne contient pas de page de garde.	Il manque 1 élément demandé.	Il y a au moins 2 éléments demandés manquants.