Travaux pratiques 2 de programmation logique

Philippe Morignot, sur une base de François Fages et al. – Mai 2023.

Finir en SWI-Prolog les exercices du TP. 1 et en particulier les exercices 6, 8 et 9, qui sont ré-utilisés dans le présent T.P.

Interrogation d'une base de données en anglais

Cette partie concerne le traitement du langage naturel en Prolog. Le but est d'interroger la base de données familiale de l'exercice 6 du T.P.1 de la façon suivante :

?- answer("who is the father of alexandre ?", Answer).

Query = father(_27880,alexandre)

Answer = father(philippe, alexandre) ;

false.

?- answer("is philippe the father of alexandre ?", Answer).

Query = father(philippe,alexandre)

Answer = father(philippe, alexandre) ;

false.

?- answer("is paul the father of alexandre ?", Answer).

Query = father(paul,alexandre)

false.

?- answer("philippe is the father of alexandre", Answer).

Query = true,father(philippe,alexandre)

Answer = (true, father(philippe, alexandre)) ;

false.

?- answer("paul is the father of alexandre", Answer).

Query = true,father(paul,alexandre)

false.

?- answer("is paul a brother ?", Answer).

Query = brother(paul,_25642)

Answer = brother(paul, daniel) ;

Answer = brother(paul, serge) ;

Answer = brother(paul, genevieve1) ;

false.

?- answer("the brother of daniel is the father of philippe", Answer).

Query = brother(_21858,daniel),father(_21858,philippe)

Answer = (brother(paul, daniel), father(paul, philippe)) ;

false.

?- answer("the father of philippe is the father of dominique", Answer).

Query = father(_13642,philippe),father(_13642,dominique)

Answer = (father(paul, philippe), father(paul, dominique)) ;

false.

?- answer("who is the ancestor of romain ?", Answer).

Query = ancestor(_1996,romain)

Answer = ancestor(frederic1, romain) ;

Answer = ancestor(fabienne, romain) ;

Answer = ancestor(paul, romain) ;

Answer = ancestor(andre, romain) ;

Answer = ancestor(rene, romain) ;

Answer = ancestor(odile, romain) ;

Answer = ancestor(suzanne, romain) ;

Answer = ancestor(genevieve2, romain) ;

false.

On donnera ci-dessous une grammaire formelle simple de l'anglais, et on représentera la structure syntaxique d'une phrase dans cette langue par un terme Prolog, appelée arbre syntaxique abstrait. Cet arbre donne les règles grammaticales à appliquer pour dériver une phrase.

Un élément clé du parsing est le calcul de la liste des feuilles d'un arbre syntaxique abstrait, puisqu'elles doivent correspondre à la liste des mots dans la phrase. Aussi on se servira des prédicats **feuilles**, **feuilles_lin** et **feuilles_freeze** des exercices 8 et 9 du T.P. 1.

On va réutiliser la base de données familiale de l'exercice 6 du T.P.1 précédent.

On considère une grammaire simple de l'anglais pour interroger cette base de données. Le verbe « is » est utilisé au singulier uniquement. La grammaire est formellement définie ci-dessous, elle est définie en Prolog par un prédicat pour chaque unité grammaticale composée d'un ou de plusieurs mots. Chaque prédicat de phrase possède un argument représentant l'arbre syntaxique abstrait pour cette phrase.

La phrase elle-même est donnée par la liste des feuilles du terme Prolog de l'arbre syntaxique abstrait.

Pour suivre la grammaire ci-dessous, on notera dans le terme Prolog de l'arbre syntaxique abstrait, « s » pour sentence/phrase, « np » pour noun phrase/groupe nominal, « npo » pour noun phrase of/groupe nominal of, « n » pour name / nom propre, « v » pour verb / verbe, « vp » pour verb phrase / groupe verbal, etc.

<phrase> ::= <groupeNominal> <groupeVerbal>

<groupeNominal> ::= <nomPropre> (homme ou femme)

<groupeNominal> ::= <articleIndéfini> <nom>

<groupeNominal> ::= <articleDéfini> <nom> <of> <nomPropre>

<articleIndéfini> ::= a

<articleDéfini> ::= the

<of> ::= of

<groupeVerbal> ::= <verbe> <groupeNominal>

<verbe> ::= is

<nom> ::= brother | sister | mother | father | grandmother | grandfather | aunt | uncle | ancestor

Exercice 10 (grammaire formelle) : écrire le prédicat **sentence(Syntax_Tree)** qui réussit si **Syntax_Tree** est un arbre syntaxique abstrait selon la grammaire formelle simple précédente de l'anglais.

On écrira ainsi des prédicats de grammaire de la forme suivante, par exemple pour la première règle grammaticale :

```
sentence(s(N, V)):-
   nounphrase(N),
   verbphrase(V).
```

Tester sur un arbre syntaxique abstrait simple construit à la main.

Vérifier qu'on peut générer tous les arbres syntaxiques abstraits possibles selon la grammaire formelle précédente.

```
?- sentence(A), feuilles(A,P).
A = s(n(philippe), vp(is, n(philippe))),
P = [philippe, is, philippe] ;
A = s(n(philippe), vp(is, n(alexandre))),
P = [philippe, is, alexandre] ;
A = s(n(philippe), vp(is, n(leonard))),
P = [philippe, is, leonard] ;
A = s(n(philippe), vp(is, n(franck))),
P = [philippe, is, franck] ;
A = s(n(philippe), vp(is, n(xavier1))),
P = [philippe, is, xavier1] ;
A = s(n(philippe), vp(is, n(frederic1))),
P = [philippe, is, frederic1] ;
A = s(n(philippe), vp(is, n(romain))),
…
A = s(n(philippe), vp(is, npo(the, brother, of, philippe))),
P = [philippe, is, the, brother, of, philippe] ;
A = s(n(philippe), vp(is, npo(the, brother, of, alexandre))),
P = [philippe, is, the, brother, of, alexandre] ;
A = s(n(philippe), vp(is, npo(the, brother, of, leonard))),
```

P = [philippe, is, the, brother, of, leonard] ;

A = s(n(philippe), vp(is, npo(the, brother, of, franck))),

P = [philippe, is, the, brother, of, franck] ;

A = s(n(philippe), vp(is, npo(the, brother, of, xavier1))),

P = [philippe, is, the, brother, of, xavier1] ;

A = s(n(philippe), vp(is, npo(the, brother, of, frederic1))),

P = [philippe, is, the, brother, of, frederic1] ;

A = s(n(philippe), vp(is, npo(the, brother, of, romain))),

P = [philippe, is, the, brother, of, romain] ;

A = s(n(philippe), vp(is, npo(the, brother, of, paul))),

P = [philippe, is, the, brother, of, paul] ;

…

P = [philippe, is, the, sister, of, philippe] ;

A = s(n(philippe), vp(is, npo(the, sister, of, alexandre))),

P = [philippe, is, the, sister, of, alexandre] ;

A = s(n(philippe), vp(is, npo(the, sister, of, leonard))),

P = [philippe, is, the, sister, of, leonard] ;

A = s(n(philippe), vp(is, npo(the, sister, of, franck))),

P = [philippe, is, the, sister, of, franck] ;

A = s(n(philippe), vp(is, npo(the, sister, of, xavier1))),

P = [philippe, is, the, sister, of, xavier1] ;

A = s(n(philippe), vp(is, npo(the, sister, of, frederic1))),

P = [philippe, is, the, sister, of, frederic1] ;

A = s(n(philippe), vp(is, npo(the, sister, of, romain))),

P = [philippe, is, the, sister, of, romain] ;

A = s(n(philippe), vp(is, npo(the, sister, of, paul))),

P = [philippe, is, the, sister, of, paul] ;

A = s(n(philippe), vp(is, npo(the, sister, of, martin))),

P = [philippe, is, the, sister, of, martin] ;

…

A = s(n(philippe), vp(is, npo(the, mother, of, alexandre))),

P = [philippe, is, the, mother, of, alexandre] ;

A = s(n(philippe), vp(is, npo(the, mother, of, leonard))),

P = [philippe, is, the, mother, of, leonard] ;

A = s(n(philippe), vp(is, npo(the, mother, of, franck))),

P = [philippe, is, the, mother, of, franck] ;

A = s(n(philippe), vp(is, npo(the, mother, of, xavier1))),

P = [philippe, is, the, mother, of, xavier1] ;

A = s(n(philippe), vp(is, npo(the, mother, of, frederic1))),

P = [philippe, is, the, mother, of, frederic1] ;

A = s(n(philippe), vp(is, npo(the, mother, of, romain))),

P = [philippe, is, the, mother, of, romain] ;

A = s(n(philippe), vp(is, npo(the, mother, of, paul))),

P = [philippe, is, the, mother, of, paul] ;

A = s(n(philippe), vp(is, npo(the, mother, of, martin))),

P = [philippe, is, the, mother, of, martin] ;

…

Que faudrait-il ajouter à cette grammaire formelle simple de l'anglais, pour représenter une grammaire formelle simple du français ?

Exercice 11 (liste de mots) : écrire le prédicat **string_to_word_list(String, Word_List)** qui réussit si **Word_List** est la liste des mots contenus dans la phrase **String** . **String** est une chaîne de caractère où les mots sont séparés par des espaces (utiliser les prédicats **split_string/4** et **maplist/3**).

?- string_to_word_list("un deux trois quatre",L).

L = [un, deux, trois, quatre].

Les prédicats prédicats **feuilles_lin** et **feuilles_freeze** renvoient l'image renversée de la liste des feuilles. Aussi écrire le prédicat **string_to_reversed_word_list(String, Reversed_Word_List)** qui est l'analogue du prédicat **string_to_word_list(String, Word_List)** mais dans lequel deuxième argument **Reversed_Word_List** est l'image renversée de la liste **Word_List**.

?- string_to_reversed_word_list("un deux trois quatre",L).

L = [quatre, trois, deux, un].

La prédicat **parse(String, Syntax_Tree)** suivant calcule alors l'arbre syntaxique abstrait d'une phrase :

```
parse(String, Syntax_Tree):-
   string_to_reversed_word_list(String, Reversed_Word_List),
   sentence(Syntax_Tree),
   feuilles_lin(Syntax_Tree, Reversed_Word_List).
```

?- parse("philippe is the brother of frederic1", L).

L = s(n(philippe), vp(is, npo(the, brother, of, frederic1))) ;

false.

?- parse("dominique is a sister", L).

L = s(n(dominique), vp(is, np(a, sister))) ;

false.

?- parse("a brother is paul",L).

L = s(np(a, brother), vp(is, n(paul))) ;

false.

Exercice 12 (efficacité du parsing) : utiliser le prédicat **feuilles_freeze/2** de l'exercice 9 du T.P.1 pour remplacer la version précédente **parse** du parsing d'une phrase, selon le principe « générer-et-tester », par un parser plus efficace, selon le principe « contraindre-et-génèrer ». Ce prédicat **parse_freeze(String, Syntax_Tree)** poste le prédicat **feuilles_freeze** avant de générer l'arbre syntaxique abstrait.

?- parse_freeze("dominique is the sister of philippe",L).

L = s(n(dominique), vp(is, npo(the, sister, of, philippe))) ;

false.

Comparer sur une même phrase les performances du prédicat **parse_freeze** et celles du prédicat **parse**. Pourquoi **parse_freeze** est-il plus efficace que **parse** ?

Exercice 13 (phrase interrogative): ajouter des règles grammaticales aux prédicats représentant des phrases, comme **sentence/1**, pour accepter des phrases interrogatives des deux formes suivantes :

- Questions fermées, comme par exemple « is paul the father of philippe ? ».
- Questions ouvertes, comme par exemple « who is the father of philippe ? ».

On devra vraisemblablement ajouter au code source la directive « **:- discontiguous sentence/1.** » pour indiquer à Prolog que dans le fichier source le prédicat **sentence** est en fait défini sur plusieurs groupes de clauses et non un seul.

?- parse_freeze("who is the father of philippe ?",L).

L = open_question(who, is, npo(the, father, of, philippe), ?) ;

false.

?- parse_freeze("is paul the father of philippe ?",L).

L = closed_question(is, paul, npo(the, father, of, philippe), ?) ;

false.

Exercice 14 (sémantique) : Ajouter aux prédicats de grammaire de l'exercice 10 précédent, la construction du sens de chaque phrase, sous forme d'un terme Prolog, qui sera pour une phrase le but Prolog à appeler pour interroger la base de données familiale précédente et obtenir la ou les réponses. Typiquement, il s'agit d'un deuxième argument aux prédicats de phrase comme **sentence/1** de l'exercice 10 (Prolog reconnait que un prédicat avec un argument n'est pas le même que ce même prédicat avec deux arguments, aussi on n'aura pas à modifier les prédicats de l'exercice 10 mais à en ajouter avec deux arguments et non un comme dans l'exercice 10).

Le prédicat **answer/2**, donné ci-dessous, vous permettra ensuite d'interroger cette base de données en langage naturel, par l'exécution du but Prolog représentant la sémantique attachée à la phrase.

semantics(String, Goal):-

```
   string_to_reversed_word_list(String, Reversed_Word_List),
   leaves_freeze(Syntax_Tree, Reversed_Word_List),
   sentence(Syntax_Tree, Goal).

answer(String, Goal):-
   semantics(String, Goal),
   write('Query = '), writeln(Goal),
   Goal.
```

Pour cela, on ne peut pas utiliser des termes Prolog de but comme par exemple **brother(X,Y)** comme deuxième argument sémantique dans les prédicats de grammaire :

noun(brother, brother(X,Y)).

parce que un prédicat appelant, avec un deuxième argument comme **Goal**, à unifier à **brother(X,Y)**, ne pourrait pas accéder aux variables **X** et **Y** du prédicat **brother**. Et trouver les bons arguments de **brother(X,Y)** est précisément une part importante de la construction du but, pour interroger la base de données familiale.

Aussi on utilisera la construction **exists(X, Goal)** qui sort du but **Goal** le premier argument sous forme de variable **X**, et ainsi permettre d'y accéder dans les prédicats de grammaire. Ce qui donne, pour reprendre le même exemple :

noun(brother, exists(X, exists(Y, brother(X,Y)))).

Ou encore, avec **Goal** ou **(Goal1, Goal2)**, comme par exemple dans le prédicat :

```
sentence(closed_question(V,N,P,I), Goal) :-
  verb(V),
  name(N),
  nounphrase(P, exists(N, Goal)),
  interrogation(I).
```

```
?- answer("who is the father of alexandre ?", Answer).
Query = father(_27880,alexandre)
```

Answer = father(philippe, alexandre) ;

false.

?- answer("is philippe the father of alexandre ?", Answer).

Query = father(philippe,alexandre)

Answer = father(philippe, alexandre) ;

false.

?- answer("is paul the father of alexandre ?", Answer).

Query = father(paul,alexandre)

false.

?- answer("philippe is the father of alexandre", Answer).

Query = true,father(philippe,alexandre)

Answer = (true, father(philippe, alexandre)) ;

false.

?- answer("paul is the father of alexandre", Answer).

Query = true,father(paul,alexandre)

false.

?- answer("is paul a brother ?", Answer).

Query = brother(paul,_25642)

Answer = brother(paul, daniel) ;

Answer = brother(paul, serge) ;

Answer = brother(paul, genevieve1) ;

false.

?- answer("the father of philippe is the father of dominique", Answer).

Query = father(_13642,philippe),father(_13642,dominique)

Answer = (father(paul, philippe), father(paul, dominique)) ;

false.

?- answer("the brother of daniel is the father of philippe", Answer).

Query = brother(_21858,daniel),father(_21858,philippe)

Answer = (brother(paul, daniel), father(paul, philippe)) ;

false.

?- answer("who is the ancestor of romain ?", Answer).

Query = ancestor(_1996,romain)

Answer = ancestor(frederic1, romain) ;

Answer = ancestor(fabienne, romain) ;

Answer = ancestor(paul, romain) ;

Answer = ancestor(andre, romain) ;

Answer = ancestor(rene, romain) ;

Answer = ancestor(odile, romain) ;

Answer = ancestor(suzanne, romain) ;

Answer = ancestor(genevieve2, romain) ;

false.

Exercice 15 (questions ouvertes, non notées) :

Que faudrait-il ajouter à la grammaire puis à ce programme pour répondre à des questions comme « who are the ancestors or romain ? » ou « who are the brothers of dominique ? » ?

La question posée à ce programme est en langage naturel. Que faudrait-il ajouter à ce programme pour que la réponse soit aussi en langage naturel, et non sous forme de terme Prolog vrai ou faux ?

ChatGPT est un programme en apprentissage automatique, avec une interface en langage naturel, qui permet d'interroger une grosse base de données représentant l'image de tout le web en 2022. Pourriez-vous donner des pistes sur ce qu'il faudrait à votre avis ajouter à notre programme pour faire la même chose (sans apprentissage automatique) ?