M2 P7 Prolog-TAL – Travaux Dirigés 2

Analyser le français, en toute modestie !

Eric de la Clergerie et André Bittar

27 Novembre 2008

Introduction

Le but de ce TD est d'étoffer une grammaire jouet du français, exprimée au début en DCG et éventuellement ensuite en BMG.

Le matériel pour ce TD peut être récupéré sous forme d'archive `.tgz` à `http://alpage.inria.fr/~clerger/PrologTAL08/td2.tgz`. Il comprend :

- une ébauche de grammaire DCG pour le français `french1.dcg`
- une mini-segmenteur `lex2db` permettant de construire un treilli de mots à partir d'une chaîne.
- un fichier `Makefile`

Exercice 1 : Premières phrases

Compiler `french1.dcg` avec l'option '`-dyacc`' (éventuellement utiliser la commande `make french1`). Examiner le fichier `french1.dcg` et tester quelques phrases,

- soit en mode lecture sur liste avec ./ french1 −− Paul dort
- soit en mode lecture sur treillis de mots avec echo "Paul␣dort" | ./ lex2db −word | ./ french1 −
  Examiner la sortie en treillis de `lex2db`

Exercice 2 : Ajouter quelques constructions

Construire une grammaire `french2.dcg` en ajoutant les constructions syntaxiques suivantes :

- groupes prépositionnels `gp` sur les phrases
- `gp` sur les groupes nominaux
- adjectifs `adj` antéposés et postposés dans les groupes nominaux

Tester des phrases comme :

- Paul mange une jolie petite pomme avec Marie
- les filles de Marie aiment Paul

Note : la gestion des adjectifs antéposés peut éventuellement effectuée avec l'opérateur de Kleene $@*$.

Exercice 3 : Gérer les accords

Construire une grammaire `french3.dcg` en ajoutant des arguments aux non-terminaux pour gérer les accords en nombre et genre au sein des groupes nominaux et entre le verbe et son sujet.

Tester quelques phrases dont :

- Paul mange une petite pomme
- * Paul mangent une petite pomme
- * Paul mange un petite pomme
- Paul mange une petit pomme

Exercice 4 : Construire un lexique

Construire une grammaire `french4.dcg` séparant plus clairement grammaire et lexique. Le lexique s'appuiera sur la mise en place de paradigmes morphologiques pour engendrer les formes associées à un lemme. Pour construire les formes à partir d'une racine et de flexion, utiliser le prédicat name_builder/3

```
%% exemple: name_builder('~w~w',[mang,e],mange)
:-std_prolog name_builder/3.
name_builder(Format,Args,Name) :-
        string_stream(_,S),
        format(S,Format,Args),
        flush_string_stream(S,Name),
        close(S).
```

Compléter la grammaire avec la gestion des clitiques nominatifs et ajouter les flexions correspondantes pour quelques verbes (au présent).

Regarder comment un paradigme `v1` pour les verbes du 1er groupe peut être compléter avec une notion de zone frontière pour pouvoir gérer un verbe comme *manger* (avec *mangeons*).

Exercice 5 : Un soupçon de sous-catégorisation

Construire une grammaire `french5.dcg` pouvant exploiter des éléments de sous-catégorisation fournis par le lexique, essentiellement sur la présence ou absence d'un objet.

Compléter le lexique pour pouvoir analyser ou non les phrases suivantes :

- Paul dort
- * Paul dort une pomme
- Paul aime Marie
- * Paul aime
- Paul mange
- Paul mange une pomme

Exercice 6 : Construire des arbres d'analyse

Construire une grammaire `french6.dcg` en ajoutant un argument supplémentaire permettant la construction d'une arbre syntaxique.

Ainsi, l'exécution de ./ french6 -- Paul mange une jolie pomme retournerait :

```
Answer:
    L = [Paul,mange,une,pomme]
    T = s(gn(Paul), gv(mange, gn(une, n(jolie, pomme))))
```

Note : Comparer l'arbre syntaxique avec la forêt de dérivation retournée par l'analyseur avec l'option `-forest` :

```
./french6 -forest -- Paul mange une jolie pomme
```

ou

```
echo "Paul␣mange␣une␣jolie␣pomme" | ./lex2db -word |./french6 -forest -
```

Exercice 7 : Construire des formes sémantiques

Construire une grammaire `french7.dcg` en ajoutant un argument supplémentaire permettant la construction de formes sémantiques. Les formes sémantiques sont construire à partir :

- de termes prédicats comme manger(X,Y),
- de l'opérateur de conjonction (Sem1 & Sem2)

– et de l'opérateur λ dans X^dormir(X).

Ainsi, l'exécution de ./ french7 -- Paul mange une jolie pomme retournerait :

```
Answer:
    L = [Paul,mange,une,jolie,pomme]
    T = s(gn(Paul), gv(mange, gn(une, n(jolie, pomme))))
    Sem = ((manger(Paul, F__0) & (undef(F__0) & pomme(F__0)) & joli(F__0)) &
    true)
```

Note : Laisser de coté la construction de formes sémantiques pour les GP, au moins pour ceux attachés à un groupe verbal GP. Pour ceux qui veulent être complets, réifier les prédicats des verbes avec des formules de la forme Evt:manger(X,Y). Ainsi, l'exécution de ./ french7 -- Paul mange une pomme pour Marie retournerait :

```
Answer:
    L = [Paul,mange,une,pomme,pour,Marie]
    T = s(gn(Paul), gv(gv(mange, gn(une, pomme)), gp(pour, gn(Marie))))
    Sem = (((G__1 : manger(Paul, F__0) & undef(F__0) & pomme(F__0)) & pour(
    G__1, Marie)) & true)
Answer:
    L = [Paul,mange,une,pomme,pour,Marie]
    T = s(gn(Paul), gv(mange, gn(gn(une, pomme), gp(pour, gn(Marie)))))
    Sem = ((_ : manger(Paul, D__0) & (undef(D__0) & pomme(D__0)) & pour(D__0,
    Marie)) & true)
```

Il est à noter que cette réification serait aussi nécessaire pour traiter des adverbes.

Exercice 8 : Gérer des extractions

Construire une grammaire `french8.dcg` permettant l'analyse de phrase interrogatives introduites par un pronom interrogative (*que* ou *qui*). Utiliser soit un argument supplémentaire, soit l'utilisation d'une pile BMG quest (plus simple !).

Réfléchir à l'impact de ces extractions/déplacement sur les arbres d'analyse et forme sémantiques.

L'exécution de ./ french8 -- que Paul mange retournerait une réponse du genre :

```
Answer:
    L = [que,Paul,mange]
    T = s(que, s(gn(Paul), gv(mange, trace)))
    Sem = ((manger(Paul, J__0) & wh(J__0)) & true)
```

Si utilisation d'une pile BMG, bloquer avec l'opérateur isl (ou isl_quest) le déchargement d'un pronom interrogative sous certains constituants, par exemple sous les GP.

Compléter la grammaire avec la gestion des relatives introduites par *que* et *qui*.

```
./french8 -- il mange la pomme que Marie aime
Answer:
    L = [il,mange,la,pomme,que,Marie,aime]
    T = s(il, gv(mange, gn(gn(la, pomme), srel(que, s(gn(Marie), gv(aime,
    trace))))))
    Sem = (_ : manger(il, K__0) & (def(K__0) & pomme(K__0)) & (_ : aimer(Marie
    , K__0) & rel(K__0)) & true)
```