ENST/TALN – Travaux Pratiques

Analyse syntaxique

Eric de la Clergerie
ATOLL/INRIA

6 Mars 2002

Introduction

Le but de ce TP est de compléter une petite grammaire d'arbres adjoints du français pour obtenir une plus grande couverture linguistique.

Ce TP s'effectue avec le système DyALog, développé par l'équipe ATOLL de l'INRIA et librement disponible (voir URL `http://atoll.inria.fr/` rubrique «Logiciels»).

TAG

DyALog permet la compilation d'analyseurs syntaxique pour différents formalismes linguistiques dont les Grammaires d'arbres adjoints (TAG). Nous illustrons le processus de compilation sur une grammaire des palindromes (Figure 1).

FIG. 1: TAG pour les palindromes

```
%> cat palin1.tag
tree     palin ([]).
tree     palin ([X]).
auxtree  palin ([X],*palin ,[X]).
?-recorded('N'(N)), tag_phrase( palin ,0, N ). %% Requête
```

La compilation s'effectue avec la commande `dyacc`

```
%> dyacc -parse -verbose_tag palin1.tag -o palin1
```

Pour pouvoir utiliser les parseurs construits avec DyALog, il faut configurer l'accès aux bibliothèques dynamiques de DyALog avec :

```
%> export LD_LIBRARY_PATH="/usr/local/lib/DyALog:$LD_LIBRARY_PATH"
```

Le parseur `palinq` peut alors être appelé pour analyser une chaîne particulière, donnée en argument de l'option `-slex` :

```
%>./palin1 -slex xyzyx
Answer:
    N = 5
```

Plus généralement, DyALog peut analyser un treilli de mots fourni en entrée. DyALog inclut un mini segmenteur de phrases produisant de tels treillis, à savoir `lex2db`. Cet utilitaire peut découper une entrée lettre par lettre (option `-letter`) ou mot par mot (option `-word`) :

```
%> lex2db -letter | ./palin1 -
xyzyx
^D
Answer:
    N = 5
```

```
%> lex2db -word | ./palin1 -
ferme la ferme
^D
Answer:
    N = 3
```

Une trace de l'analyse sous forme d'une forêt partagée de dérivation s'obtient avec l'option **-forest** :

```
%>./palin1 -forest -slex xyzyx
Answer:
    N = 5

----------------------------------------------------------------------
Shared Forest

*ANSWER*{answer=> [N = 5]}
        0 <-- [0]1
palin (0,5)
        1 <-- [0]2
palin (0,5) * palin (2,3)
        2 <-- [1]3
palin (0,5) * palin (1,4)
        3 <--
```

L'ajout d'arguments aux non-terminaux et d'échapements vers Prolog sont possibles pour les TAG.

```
%> cat palin2.tag
:- tag_features ( palin , palin (_), palin (_ )). % info attachée à palin
tree     bot=palin (0) at palin ([]).
tree     bot=palin (1) at palin ([X]).
auxtree  bot=palin(L) at palin ([X],top=palin(K) at * palin ,[X],{L is K+1}).
?-recorded('N'(N)), tag_phrase(top=palin(L) at palin ,0, N). %% Requête
%> dyacc -parse palin4.tag -o palin4
```

Structures de traits et domaines finis

DyALog facilite l'écriture de grammaires grâce à l'utilisation de structures de traits et de domaines finis. Ainsi, le fragment suivant définit quelques traits possibles pour un verbe, les ensembles finis de valeurs possibles pour certains de ses traits, et une entrée de lexique pour la forme «aime» du verbe «aimer» :

```
:- features ([v],[gen,mode,num,pers,tense]).
:- finite_set ( mode, [ind, subj, inf ] ).
:- finite_set ( tense , [ pres, fut ]).
:- finite_set ( num, [pl, sing ] ).
:- finite_set ( gender , [ fem,masc ] ).
```

```
:- finite_set ( pers , [1,2,3] ).
lexicon (aime,
      v{ pers => pers [1,3],
         num => sing,
         mode => mode[ind,subj],
         tense => pres }
      ).
```

Architecture XTAG

Les grammaires TAG à large couverture suivent l'architecture XTAG, qui identfie 3 grands niveaux :

Arbres élémentaires Ils sont regroupés en *famille* et peuvent être ancrables (noeud <>).

```
tag_tree { name=>n0vn1,        % Verbe transitif
           family=>n0vn1,
           tree => tree s(id=np0 at np,vp(<>v,id=np1 at np))
         }.
```

Lexique syntaxique Il regroupe les lemmes, indiquant pour chaque lemme les familles dont il ancre les arbres. Il est également possible de spécifier des contraintes de co-ancrages et d'équations portant sur les noeuds des arbres ancrés. Un lemme peut ancrer plusieurs familles.

```
tag_lemma(manger,v, [tag_anchor{ name => [n0vn1,n0v ] } ]).
tag_lemma(donner,v, tag_anchor{ name => n0vn1pn2, coanchors => [p=à ] }).
```

Lexique morphologique Il regroupe les formes fléchies, indiquant pour chacune les lemmes associés et les traits morphologiques spécifiques.

```
tag_lexicon (aime, aimer , v , v{ num => sing , pers => 3 }).
```

Exercice 1 : Prise en main

Récupérer la grammaire TAG respectant l'architecture XTAG disponible par FTP à `ftp://ftp.inria.fr/INRIA/Projects/Atoll/Eric.Clergerie/TAG/td_enst02.tar.gz` ou par WEB à partir de `http://atoll.inria.fr/~clerger` Rubrique «Teaching».

Compiler la grammaire, examiner son source et tester quelques phrases comme :

- *Jean mange*
- *Jean mange une pomme*
- *Jean mange une pomme verte*
- *Jean fume*
- ** Jean fume une pomme*

En pratique, commencer par faire (l'équivalent de)

```
> tar xzvf td_enst02.tar.gz
> cd td_enst02; make base
```

Les examples peuvent ensuite être exécutés avec

```
> ./lex2db -word | ./base - -forest
Jean mange
^D
```

ou avec

```
> ./run_parser -s "Jean mange"
```

Le petit utilitaire `run_parser` permet également de tester des phrases annotées prises dans un fichier, comme le fichier `examples.txt` :

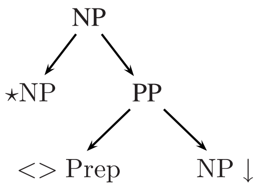
```
> ./run_parser examples.txt
```

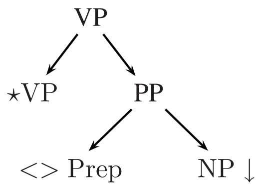
Exercice 2 : Gestion des accords

Elle n'est pas faite dans la grammaire de base sauf pour l'arbre `dn` qui gère les groupes nominaux. En vous en inspirant, mettre les conditions d'accord pour les autres arbres (adjectifs et verbes).

Exercice 3 : Groupes prépositionnels

Ajouter une famille `prep` avec les arbres permettant de gérer l'attachement des groupes prépositionnels (**PP**) sur les groupes nominaux ou verbaux.





Tester avec
- *Jean mange la pomme de Pierre*
- *Jean regarde une pomme avec un télescope*
- *Jean regarde une pomme avec le télescope de Pierre*

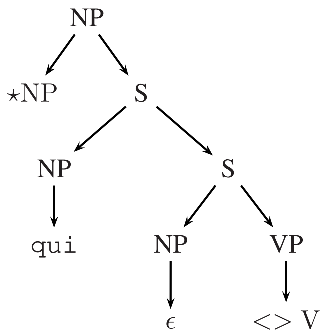
Exercice 4 : Catégorisation (1)

Ajouter un arbre `n0vn1pn2` et une famille `n0vn1pn2` pour les verbes ditransitifs, comme *donner* :
- *Jean donne une pomme à Pierre*

Exercice 5 : Gestion des relatives

Compléter les familles verbales présentes pour inclure la gestion des relatives :
- *Jean regarde Anne qui nage*
- *Jean regarde Anne qui fume un cigare*
- *Jean regarde le cigare que Anne fume*
- *Jean regarde le cigare que Anne donne à Paul*

Note : on pourra s'appuer sur le modèle suivant :



Extension1 : Gérer les accords entre un antécedent sujet et son verbe :

– **Jean regarde Anne qui nagent*

Extension2 : Avec des modèles similaires, on peut ajouter des arbres pour gérer les formes interrogatives, à condition d'ajouter aux lexiques des entrées pour des pronoms ou determinants interrogatifs. Ceci permet aussi de traiter les relatives suivantes :

– *Jean regarde Anne à qui Anne donne une pomme*

Exercice 6 : Catégorisation (2)

Ajouter un arbre et une famille `vvp` pour les verbes sous-catégorisant des infinitives, par exemple pour le verbe *devoir* :

– *Jean doit manger une pomme*
– *Jean peut manger une pomme*
– *Jean doit pouvoir nager*

Extension : Le traitement correct des infinitives oblige à raffiner les familles verbales précédentes pour l'accord de mode entre le verbe et son groupe verbal :

– **Jean doit nage*
– **Jean doit pouvoir nage*
– **Jean doit peut nager*

Exercice 7 : Catégorisation (3)

Ajouter un arbre et une famille `vvp` pour les verbes sous-catégorisant une complétive, par exemple pour le verbe *penser* :

– *Jean pense que Anne mange une pomme*
– *Jean pense que Anne doit manger une pomme avec Pierre*

Suivant l'exemple du verbe *manger*, assigner une sous-catégorisant multiple au verbe *vouloir* :

– *Jean veut manger une pomme*
– *Jean veut que Anne mange une pomme*
– *Jean doit vouloir manger une pomme*
– *Jean veut devoir nager avec Pierre*

Vérifier que le programme permet l'analyse de la phrase

– *Anne doit penser que Anne doit vouloir regarder la pomme que Paul veut donner à Pierre*

Exercice 8 : Négation

Ajouter les arbres pour traiter les formes simples de la négation :

– *Jean ne mange pas*
– *Jean ne veut pas manger*
– *Jean ne veut pas ne pas manger*
– *Jean ne ne mange pas pas*
– ** Jean veut ne manger pas*

Suggestion : ajouter un trait booleen sur les verbes pour éviter les doubles négations.

Note1 : Une grammaire jouet du français peut être testée en ligne à `http://atoll.inria.fr` Rubrique «Démos». Elle bénéficie d'un affichage graphique plus convivial !

Note2 : Les forêts de dérivation produites par votre analyseur peuvent être visualisées sous forme graphiue en les soumettant au script CGI `http://medoc.inria.fr/pub-cgi-bin/forest.cgi`.