Analyse syntaxique

Cours d'initiation au TAL - M1 SCA & SDL

15 octobre 2013

1 Arbre syntaxique

A l'aide d'un navigateur, affichez le tutoriel NLTK (URL : *http ://nltk.org/book*).

Ouvrez le tutoriel à partir du chapitre 7 « Extracting Information from Text »et allez directement à la section 7.4 « Recursion in Linguistic Structure ». Lisez attentivement les explications en exécutant tous les exemples qui les accompagnent dans une fenêtre *Idle* ouverte en parallèle.

Cette section vient après une section sur le « chunking » qui consiste à découper un texte en groupes de mots ou « chunks ». Ces « chunks »ne sont pas eux-mêmes structurés et certains mots sont laissés hors d'un « chunk ».

Dans le paragraphe « Building Nested Structure with Cascaded Chunkers », on montre comment structurer des « chunks » en utilisant un « chunker » en plusieurs passes (à chaque passe, il essaie de découper les « chunks » de la passe précédente en « sous-chunks »). On montre aussi les limites de cette approche.

Le paragraphe « Trees »introduit la notion d'arbre. Le constructeur *nltk.Tree (etiquette, liste_fils)* permet de construire un arbre. Son premier argument *etiquette* est une chaîne de caractères qui étiquettera la racine de l'arbre. Le second argument est une liste des sous-arbres fils de l'arbre construit. Certains éléments de la liste peuvent être des chaînes de caractères. Ces chaînes représentent alors les étiquettes de feuilles de l'arbre.

Exercice 1.1 *Ecrivez un programme Python qui crée et affiche tous les différents arbres syntaxiques des phrases ci-dessous (toutes sont ambigües donc elles ont au moins deux arbres syntaxiques).*

Le joueur de football américain connaît Marie.
Marie emprunte un stylo à Jean.
Marie arrive et repart avec Jean.
Les filles de Marie et Jean arrivent.

Vous utiliserez les catégories grammaticales suivantes :

symbole	*nom en anglais*	*nom en français*
S	*sentence*	*proposition*
NP	*noun phrase*	*syntagme nominal*
VP	*verb phrase*	*syntagme verbal*
PP	*prepositional phrase*	*syntagme prépositionnel*
Adj	*adjective*	*adjectif*
Conj	*conjunction*	*conjonction*
Det	*determiner*	*déterminant*
N	*common noun*	*nom commun*
Prep	*preposition*	*préposition*
V	*verb*	*verbe*

Mettez le programme dans un fichier TP3exo1.1.py.

2 Grammaires algébriques et analyse syntaxique

Passez au chapitre 8 « Analyzing Sentence Structure ». La section 8.1 « Some Grammatical Dilemmas » aborde deux aspects de l'analyse syntaxique :

- Le paragraphe « Linguistic Data and Unlimited Possibilities » présente la notion de grammaire générative qui permet de produire une infinité de phrases dites grammaticales.
- Le paragraphe « Ubiquitous Ambiguity » aborde la question de l'ambiguïté syntaxique.

La section 8.2 « What's the Use of Syntax ? » introduit la notion de constituant syntaxique. Puis la section 8.3 « Context Free Grammar » présente le formalisme des grammaires algébriques, tel qu'il est appliqué à la modélisation de la syntaxe des langues.

Le paragraphe « A simple Grammar » illustre ce qu'est une grammaire algébrique à l'aide d'un exemple linguistique et une démonstration d'analyse descendante. L'exemple a été choisi pour montrer la notion d'ambiguïté syntaxique.

Dans le paragraphe « Writing Your Own Grammars », on montre comment utiliser sa propre grammaire pour faire une analyse. Cette grammaire doit avoir été écrite dans un fichier *.cfg* au prélable selon une syntaxe qui n'est pas précisée ici. Le paragraphe « Recursion in Syntactic Structure » présente enfin un exemple de grammaire récursive.

La section 8.4 « Parsing With Context Free Grammar » aborde l'analyse proprement dite à l'aide d'une grammaire algébrique. Différentes stratégies sont proposées. La première exposée dans le paragraphe « Recursive Descent Parsing » est la méthode descendante récursive. Elle consiste à construire l'arbre syntaxique à partir de la racine S, qui représente la phrase globale, en descendant jusqu'aux feuilles qui représentent les mots de la phrase. Une démonstration qui peut être lancée avec l'instruction *nltk.app.rdparser()* vous permet de mieux la comprendre. Etudiez-la attentivement !

La seconde méthode exposée dans le paragraphe « Shift-Reduce Parsing » est une méthode ascendante qui utilise une pile. Au fur et à mesure de la lecture des mots de la phrase, ceux-ci sont empilés (*shift*). Dès que sur la pile, apparaît une suite de symboles représentant la partie droite d'une règle, cette suite est remplacée par la partie gauche de

la règle (*reduce*). A la fin, il ne doit rester que le symbole de départ sur la pile. Etudiez avec soin la démonstration réalisée par l'instruction *nltk.app.srparser()*.

Exercice 2.1 *Voici une suite de phrases grammaticalement correctes :*

Le beau bébé dort dans le berceau.
Jean voit le bébé dans le berceau dans la chambre.
Jean pense que dans le berceau le bébé dort.

1. *Ecrivez un programme Python qui permette d'analyser (en mode trace) ces phrases avec une grammaire algébrique que vous concevrez, en utilisant la stratégie* shift-reduce*. Pour écrire cette grammaire, vous pourrez utiliser les non terminaux de l'exercice précédent mais vous avez la liberté de modifier et d'enrichir ces non terminaux. Vous mettrez le programme dans un fichier* TP3exo2.1.py*.*
2. *Exécutez le programme précédent et enregistrez la sortie du programme dans un fichier* TP3exo2.1.txt*. Si vous avez rencontré des problèmes dans l'analyse de certaines phrases, cherchez la cause de ces problèmes et notez vos conclusions dans le même fichier.*

3 Envoi des réponses aux exercices

Réunir les différents fichiers contenant les réponses aux exercices dans une archive nommée *<nom-étudiant>.<prénom-étudiant>.TP3.zip* et envoyez-le par mail en pièce jointe à l'adresse *perrier@loria.fr* en mettant dans l'objet du message *initiation au TAL : TP3.*