25 AVRIL 2017

PROJET LANGAGE NATUREL

LE DIALECTE DE YODA

ALEXIS COUVREUR

AIX-MARSEILLE UNIVERSITE

TABLE DES MATIERES

DESCRIPTION DU PROBLEME, EXEMPLES D'ENTREE/SORTIE

DESCRIPTION DU PROBLEME

Le but est de fournir par écrit une phrase normalement construite en Français au programme, celui-ci devra être en mesure de pouvoir transformer/traduire cette phrase dans le dialecte de Yoda (cf. Wikipedia). Sa façon de parler est devenue mythique : elle utilise la syntaxe objet, sujet, verbe, totalement à l'inverse de la plupart des langues indo-européennes, qui suivent la syntaxe sujet, verbe, objet.

Cependant, pour que certaines phrases restent compréhensibles, ce ne sont souvent que les auxiliaires qui sont déplacés. La formation des phrases est une résultante d'une figure stylistique appelée anastrophe.

EXEMPLES D'ENTREES ET DE SORTIES

Voici des exemples d'entrées et de sorties :

- « J'aimerais aller à la piscine. » « Aller à la piscine, j'aimerais. »
- « Il faut faire attention aux orages. » « Faire attention aux orages, il faut. »
- « Bonjour. » « Bonjour. »
- « Je suis ton père. » « Ton père, je suis. »
- « Un petit ballon de foot est plus rapide. » « Plus rapide, un petit ballon de foot est. »

RECHERCHE BIBLIOGRAPHIQUE DES SOLUTIONS EXISTANTES

L'ANASTROPHE

L'anastrophe est une figure de style, dite de « construction », qui consiste en une inversion de l'ordre habituel des mots d'un énoncé pour créer un effet de langue raffiné.

Dans la langue Française, qui est qualifiée de langue SVO (sujet-verbe-objet) on transforme donc une phrase normale en anastrophe tel que :

Sujet – Verbe – Objet *devient* Objet – Sujet – Verbe

ANALYSEURS SYNTAXIQUES LIBRES

J'ai choisi d'étudier les analyseurs libres suivants : **TreeTagger**, **Berkeley Parser**, **Stanford Parser** et **UDpipe**.

- **TreeTagger** est un outil permettant d'annoter un texte avec sa partie du discours et des informations sur le lemme. **TreeTagger** est utilisée pour l'Allemand, l'Anglais, le Français, l'Italien, etc. ce qui en fait un point fort étant donné qu'il sera possible d'utiliser la partie du discours pour identifier correctement le pattern des phrases et ainsi pouvoir agir en conséquence.

- **Berkeley Parser** est un outil qui permet d'écrire et de trier des arbres résultant d'une entrée tokenizée. Beaucoup d'options supplémentaires sont présentes mais il ne contient pas de grammaire française par défaut (seulement Anglais, Allemand et Chinois). Il est possible de faire apprendre une grammaire avec une « treebank ». Etant donnée l'entrée de mon programme (phrases courte) cet outil fournit beaucoup plus que ce qui est nécessaire et semble plus complexe à mettre en place.

- **Stanford parser** est un outil d'analyse de langage naturel qui travaille sur la structure grammaticale dans des phrases et fournit des informations s'il s'agit du sujet, de l'objet ou alors du verbe. **Stanford parser** est un analyste probabiliste, il va se baser sur la connaissance sur la probabilité du langage pour sortir des situations ambigües.

- **UDPipe** est un outil de tokenization, qui tag et affiche les lemmes. Il fournit le résultat sous plusieurs formes tel que le texte tokenisé, dans un tableau ou encore dans un arbre.

TESTS DE LOGICIELS LIBRES EXISTANTS

Un site Anglophone proposant le service de transformer des phrases Anglophones dans le dialecte de Yoda existe à l'adresse suivante : http://www.yodaspeak.co.uk/ ; l'output est souvent accompagné d'interjections donnant un rendu plus semblable à celui de Yoda.

L'utilisation de **TreeTagger** est réalisable très rapidement à travers Java, il suffit d'abord d'installer **TreeTagger** et d'ensuite utiliser un binder tel que **TT4J**, qui dispose d'un répertoire maven ce qui rend l'implémentation plus simple.

Voici un exemple d'entrée/sortie lors de l'utilisation de l'analyseur **TreeTagger** à travers Java :

Il	PRO:PER	il
est	VER:pres	Ãªtre
ton	DET:POS	ton
père	NOM	père
.	SENT	.

En négligeant les problèmes d'encodage en sortie, on peut facilement constater comment transformer cette phrase en un anastrophe grâce aux informations données. Le pronom « il » est le sujet, le verbe « est » est le verbe et l'objet est « ton père » ; ainsi s'il on transforme la phrase SVO en OSV on obtient : ton père il est. Il suffit de mettre les majuscules au bon endroit et de séparer la phrase par des virgules pour qu'elle soit correcte.

En revanche, un inconvénient et le fait que la librairie TT4J n'as pas de Tokenizer intégré, alors qu'il s'agit là d'une des parties les plus importantes du procéder de transformation de phrase, une analyse morpho-syntaxique doit être réalisé sans soucis.

C'est donc vers le **Stanford Parser** que je me tourne, celui-ci proposant un Tokenizer (paramétrable) ainsi que de nombreuses fonctionnalités tel que :

- L'étiquetage morpho-syntaxique
- Le parcours syntaxique
- Le parcours sémantique

Ce sont les principales fonctionnalités de la librairie.

CAHIER DES CHARGES POUR L'IMPLEMENTATION FINALE

TECHNOLOGIES UTILISEES

Le but est de réaliser un programme permettant de transformer un texte dans le dialecte de Yoda.

Le langage de programmation sera le **Java**, en utilisant l'analyseur syntaxique **Stanford Parser** (celui-ci se portant bien en Java). C'est cet analyseur que j'ai choisi car il est très complet et bien documenté, mais il aussi l'un des seul à intégrer un *Tokenizer* dans son package, ce qui évite de faire appel à des ressources externes et/ou autres librairies. *Et le programme résultant sera un appel par ligne de commande, il n'y aura pas d'interface graphique.*

ENTREES/SORTIE ATTENDUES

Le texte à traduire/transformer aura certaines conditions à respecter, il ne devra pas excéder 80 caractères, il s'agit là de la moyenne du nombre de caractère pour une phrase Française de longueur moyenne (12 à 18 mots, c'est-à-dire 66 à 99 mots).

Le texte devra être exclusivement en **Français** et toutes fautes de frappe et mots non reconnus ne seront pas traités. Le texte devra avoir une forme spécifique tel qu'il commence par un **sujet**, suivi d'un **verbe** et enfin d'un **objet** (SVO).

Le texte traduit/transformé sera donc formé tel qu'il commence par un **objet**, suivi du **sujet** et enfin d'un **verbe**. L'anastrophe proposant une tournure de phrase nécessitant l'ajout de ponctuation au sein de la phrase pour *casser* le style classique, ceux-ci seront donc ajoutées au texte résultant, exemple : *« Il est ton père. »* devient *« Ton père**,** il est. »*

MISE EN ŒUVRE

INTEGRATION DE L'ANALYSEUR

L'analyseur **Stanford Parser** est intégré via **Maven** en revanche il nécessite d'ajouter les modèles Français, dans lequel est fournis un modèle pour créer une instance d'un **Analyseur Lexicalisée**, il s'agit d'un modèle dit *Probabilistic Context-Free Grammar* (**PCFG**) ce qui se traduit par *Grammaire Hors-Contexte Probabiliste*, celui va donc permettre de tokenizer le texte de manière intelligente en levant les ambiguïtés.

TOKENIZATION

Lorsque l'on effectue la tokenization, on obtient le résultat sous forme d'un arbre. Nous allons, pour la suite de la présentation de la mise en œuvre, utiliser la phrase suivante comme exemple : « Je suis ton père. »

Voici donc le résultat de la tokenization sous forme d'arbre :

```
(ROOT
  (SENT
    (VN (CLS Je) (V suis))
    (NP (DET ton) (NC père))
    (PUNC .)))
```

INTERPRETATION

Interpréter ce résultat est à première vue très simple et le travail est fait, il suffit, pour arriver au résultat escompté d'inverser les nœuds VN et NP, mais il s'agit d'un exemple parmi tant d'autre. Car prenons une autre phrase tel que : « Il dort pendant les cours. »

```
(ROOT
  (SENT
    (VN (CLS Il) (V dort))
    (PP (P pendant)
      (NP (DET les) (NC cours)))
    (PUNC .)))
```

On remarque tout de suite les problèmes de l'utilisation d'un arbre. Je me suis alors tourné vers une relation de dépendance entre les mots, ceci étant aussi inclut dans le **Stanford Parser**.

ARBRE DE DEPENDANCES

On instancie un arbre de dépendance, auquel on donne comme paramètre un arbre comme vu précédemment. Voici un exemple de sortie :

Universal dependencies :

```
nmod:poss(dog-2, My-1)
nsubj(likes-4, dog-2)
advmod(likes-4, also-3)
root(ROOT-0, likes-4)
xcomp(likes-4, eating-5)
dobj(eating-5, sausage-6)
```

Exemple depuis le site officiel de l'analyseur **Stanford Parser**, on peut donc tout de suite se rendre compte qu'il possède des informations cruciales pour notre problème, à savoir une indentification du sujet, du verbe et de l'objet.

Malheureusement, les structures grammaticales ne sont pas supportées dans le modèle Français !

```
System.out.println(
  LexicalizedParser
    .loadModel("frenchFactored.ser.gz")
    .treebankLanguagePack()
    .supportsGrammaticalStructures()
);
```

En effet, ce code visant à savoir si l'étude des structures grammaticales est supportée, et par conséquent l'analyse des dépendances, retourne faux.

Pour remédier à ce problème, j'ai décidé d'opter pour une manière presque similaire, je vais utiliser une autre libraire en plus de l'analyseur **Stanford Parser**. Cet autre analyseur syntaxique est le **MaltParser**, il permettra de proposer un résultat tel que l'exemple produit le résultat suivant :

1	Je	Je	C	CLS	null	2	suj	_	_
2	suis	suis	V	V	null	0	root	_	_
3	ton	ton	D	DET	null	4	det	_	_
4	père	père	N	NC	null	2	obj	_	_
5	.	.	P	PUNC	null	2	mod	_	_

UTILISATION DU MALTPARSER

Parmi les deux familles principales d'analyseurs, **MaltParser** fait partie des analyseurs à transitions, il peut être caractérisé comme un générateur d'analyse par analyseur de données. Alors qu'un générateur d'analyseur traditionnel construit un analyseur à l'aide d'une grammaire, un générateur d'analyse par analyseur de données construit un analyseur à l'aide d'une banque d'arbres (TreeBank). **MaltParser** est une implémentation de l'analyse par dépendance inductive, où l'analyse syntaxique d'une phrase équivaut à la dérivation d'une structure de dépendance, et où l'apprentissage par machine inductive est utilisé pour guider l'analyseur à des points de choix non déterminés. La méthodologie d'analyse repose sur trois composantes essentielles :

- Des algorithmes d'analyse déterministes pour la construction de graphiques de dépendance marqués
- Modèles basés sur l'histoire pour prédire la prochaine action de l'analyseur aux points de choix non déterminés
- L'apprentissage discriminatif pour cartographier les histoires aux actions d'analyseur

Ce nouveau **Parser** qui vient s'ajouter au traitement de l'entrée nous donne un graph que l'on peut voir comme précédemment et plusieurs informations utiles, à savoir un étiquetage morpho-syntaxique sur les lemmes (basique et amélioré), l'index de la dépendance et enfin la détection de coréférences et résolution d'anaphores ce qui permet donc de dire que tel ou tel lemme est le sujet, l'objet, le verbe (ici appelé root).

TRAITEMENT DE L'ENTREE

Ainsi, tous nos outils sont prêts à l'emploi et nous pouvons procéder au traitement de l'entrée. On va donc procéder de la manière suivante, tout d'abord on va détecter si dans le texte il existe plusieurs phrases, dans tous les cas on applique le traitement phrase après phrase. On la tokenize en utilisant **Stanford Parser** qui nous produit un arbre, à partir de cet arbre on va établir un graphe des dépendances avec le **MaltParser**.

On va parcourir le graphe produit du début à la fin en isolant les séquences concernant le sujet, le verbe et l'objet.

GRAMMAIRE HORS CONTEXTE

Pour vérifier qu'il s'agit d'un SVO on va créer une grammaire hors-contexte tel que :

Phrase -> GN GV

GN -> D N GN'

GN' -> &

GN' -> A

GN' -> P N

GV -> V ~~GN~~

D -> Determinants…

A -> Adjectifs…

P -> Pronoms…

N -> Noms…

V -> Verbes…

Cette grammaire permet de constituer une phrase de type SVO et donc de vérifier si une phrase en entrée est correcte.

GN est barré de GV car sinon on aurait des phrases plus conséquentes.

De ce fait on établit une liste des premiers et suivants, on peut donc faire le parcours et ainsi en cas de problème renvoyer la phrase originale.

TRAITEMENT FINAL

Une fois ces séquences isolées, appelées respectivement S, V et O, on change l'ordre de manière à former une séquence OSV. Suite à ça on enlève les espaces produits par la tokenization, on met les majuscules où il faut et on retourne la phrase dans le style d'une anastrophe.

RESULTATS ATTENDU/OBTENUS

Nous allons effectuer une série de test, ces tests porteront sur des phrases ayant les tournures suivantes :

1. Sujet – Verbe (SV)
2. Sujet – Verbe – Objet (SVO)
3. Sujet – Verbe – Objet indirect – Objet direct (SVOIOD)
4. Sujet – Verbe – Complément sujet (SVCS)
5. Sujet – Verbe – Objet – Complément d'objet (SVOCO)

Ces tests devraient être fonctionnels car même s'il s'agissait d'une structure SVO à en devenir une structure OSV, il s'agit en fait d'une même et unique base commune à chaque phrase, ici SV. Ainsi toutes les phrases commençantes avec une structure SV fonctionneront.

SUJET – VERBE

Entrée	Obtenu	Attendu
Les garçons ont sauté au-dessus du mur.	Au-dessus du mur, les garçons ont sauté.	Au-dessus du mur, les garçons ont sauté.
Il dort pendant les cours.	Pendant les cours, il dort.	Pendant les cours, il dort.
Les oiseaux de la même plume volent ensemble.	Ensemble, les oiseaux de la même plume volent.	Ensemble, les oiseaux de la même plume volent.

493ms

SUJET – VERBE – OBJET

Entrée	Obtenu	Attendu
Elle enseigne l'Anglais à l'université.	À l'université, elle enseigne l'anglais.	À l'université, elle enseigne l'Anglais.
Sa voiture a percuté un arbre ce matin.	Un arbre ce matin, sa voiture a percuté.	Un arbre ce matin, sa voiture a percuté.
Ils vont rencontrer le patron Lundi prochain.	Rencontrer le patron lundi prochain, ils vont.	Rencontrer le patron Lundi prochain, ils vont.

1s 534m

SUJET – VERBE – OBJET INDIRECT – OBJET DIRECT

Entrée	Obtenu	Attendu
La femme a donnée à sa fille un cadeau pour son anniversaire.	À sa fille un cadeau pour son anniversaire, la femme a donnée.	À sa fille un cadeau pour son anniversaire, la femme a donnée.
Le gestionnaire de la banque a accordé un prêt pour le pauvre agriculteur ce matin.	Un prêt pour le pauvre agriculteur ce matin, le gestionnaire de la banque a accordé.	Un prêt pour le pauvre agriculteur ce matin, le gestionnaire de la banque a accordé.
M. Mendoza nous a appris le grec à l'époque.	Le grec à l'époque, m. mendoza nous a appris.	Le grec à l'époque, M. Mendoza nous a appris.

1s 95ms

SUJET – VERBE – COMPLEMENT SUJET

Entrée	Obtenu	Attendu
Hannah était un professeur à Delhi.	Un professeur à delhi, hannah était.	Un professeur à Delhi, Hannah était.
Le vieux monsieur a l'air heureux aujourd'hui.	L'air heureux aujourd'hui, le vieux monsieur a.	L'air heureux aujourd'hui, le vieux monsieur a.
Ces jeunes gens vont devenir des experts bientôt.	Devenir des experts bientôt, ces jeunes gens vont.	Devenir des experts bientôt, ces jeunes gens vont.

521ms

SUJET – VERBE – OBJET – COMPLEMENT D'OBJET

Entrée	Obtenu	Attendu
Le Premier ministre a nommé M. X un ministre en 2004.	M. x un ministre en 2004, le premier ministre a nommé.	M. X un ministre en 2004, le premier ministre a nommé.
L'inspecteur a trouvé l'homme innocent.	L'homme innocent, l'inspecteur a trouvé.	L'homme innocent, l'inspecteur a trouvé.
Le locateur a appelé le nouveau locataire un escroc.	Le nouveau locataire un escroc, le locateur a appelé.	Le nouveau locataire un escroc, le locateur a appelé.

2s 213ms

INTERPRETATION DES RESULTATS ET CONCLUSION

On peut voir que nos résultats sont satisfaisants, malgré un temps d'exécution assez long pour une machine, le texte en retour est acceptable. Il comporte cependant des erreurs suite au traitement du texte, on perd certaines majuscules par exemple.

On peut voir que cela couvre une grande partie des phrases d'un longueur de 80 caractères et que l'objectif a bien été atteint.

Ce projet a donc atteint son but, il est capable de transformer une phrase simple dans le dialecte de Yoda, si celle-ci est interprétable. Nous avons pu utiliser les différentes ressources étudiées à travers les TD/TP et cours, à savoir les grammaires hors contextes, les arbres de dérivations, l'étude morpho-syntaxique, l'utilisation des expressions régulières mais aussi l'intégration d'analyseurs syntaxiques dans le langage Java.

Il pourrait être amélioré en ajoutant des onomatopées et en évitant la casse, devenir une API ou utilisable à travers une interface graphique.

C'est un projet que j'ai réalisé avec joie et curiosité, qui n'a pas cessé de m'apporter de nouvelles connaissances.