KHOW Antoine
HAFID Yahya
AOUINE Katia

Rapport de projet de Linguistique

Problématique : Représentation de la langue

Vous pouvez trouver le dépôt final du projet sur le dépôt Gitlab suivant sous le tag ‘rendu_projet’ : https://gaufre.informatique.univ-paris-diderot.fr/Ant.oine/linguistique

Nous avons choisi de traiter le sujet de la représentation de la langue, c'est-à-dire la création d'un programme qui permet de distinguer le sujet du verbe et de détecter certains autres éléments dans une phrase donnée. La première étape consistant à choisir une langue à traiter, nous avions le choix entre la langue française et la langue anglaise. Nous avons finalement choisi de traiter le français, par souci de complexité de la seconde à cause des verbes irréguliers, et également car les verbes n’ont pas de terminaisons définies en anglais au présent.

Si en français la complexité de détection des verbes était beaucoup plus simple qu’en anglais, la détection des sujets elle n’en restait pas moins complexe, un sujet étant une terminologie décidée dans les règles de grammaire et pouvant prendre des formes très abstraites. Sans compter prendre en compte bien d’autres détections telles que les adverbes, adjectifs, etc sans oublier les divers temps de conjugaison.

Pour traiter ce problème, nous avions donc 2 possibilités, utiliser une librairie externe (Stanza) qui fait ce travail de manière décousue et donc ré-assembler et réinterpréter les résultats afin d’assembler une représentation correcte mais exigeant une grande précision sur de nombreux cas. Ou bien faire tout ce travail à partir de zéro afin d’avoir une meilleure compréhension du sujet, de travailler différentes méthodes de détection mais ayant une précision moindre.

Nous avons alors décidé de prendre la 2ème option et de partir sur des bases simples, détecter des phrases simples (contenant 1 verbe et 1 sujet grand max) et au présent puis augmenter le cran de difficulté petit à petit.

Hypothèses et solutions possibles

Avec le cadre fixé, il fallait donc commencer à réfléchir à comment faire. Pour les verbes, une piste de réflexion fut bien entendu les terminaisons des différents temps et groupes de verbe. Evidemment cela allait être plus compliqué au présent à cause de terminaisons en -e par exemple sans compter divers autres temps et également distinguer des terminaisons spécifiques/irrégulières telles que -ds, -is, etc. Il faudrait également penser à lever les ambiguïtés du verbe mais cela se ferait probablement à la fin.

Pour la détection du sujet en revanche, peu d’idées nous vinrent à l’esprit immédiatement mais nous avons pu isoler 3 cas courants :

- Sujet à côté du verbe
- Sujet séparé par CC du verbe
- Sujet après le verbe

Si bien sûr ces cas étaient faciles à mettre en place, une inquiétude subsistait car le programme n'allait sûrement pas comprendre ce qu'était la différence entre un sujet et un verbe mais nous décidions de voir tout ça plus tard en temps voulu.

Solution retenue

Après 2 semaines de travail, quelques problèmes commencèrent à arriver, non seulement il devenait compliqué de distinguer le sujet pour certaines personnes (ex : je perds, tu perds, même terminaison avec -ds) mais en plus les diverses terminaisons des verbes étaient souvent les mêmes que celles de noms communs :

- cahier -> terminaison en -er présent pour les verbes du 1er groupe
- personne -> terminaison en -e présent pour les verbes du 1er groupe avec je
- toutes -> terminaison en -es présent pour les verbes du 1er groupe avec tu
- etc...

Ainsi, de nombreux noms communs furent associés en tant que verbes donnant des sujets sans aucun sens (ex : la personne, "la" associé comme sujet du verbe "personne").

En fin de compte, pour palier à ce problème, il fut décidé d'utiliser un dictionnaire contenant tous les verbes dans leurs formes conjuguées et infinitif et de le parcourir pour vérifier si un mot était un verbe ou non (Le dictionnaire Lefff : https://www.labri.fr/perso/clement/lefff/).

Si l'on perdait en efficacité de temps, nous obtenions un grand gain de temps et d'efforts car ce dictionnaire très bien répertorié indiquait pour chaque verbe le temps et la personne. Nous pouvions donc utiliser cette information afin de déterminer des pistes du sujet, la plus grande difficulté étant d'adapter ce dictionnaire et comment l'utiliser.

Nous avancions donc progressivement sur le projet avec cette méthode qui permettait de ne plus sélectionner des verbes de manière "hasardeuse".

L'implémentation et sa description

Suite aux premières discussions nous avons pu établir très vite 3 parties distinctes du code :

- Parser : Traiter l'input histoire de mieux découper la phrase, potentiellement détecter tous les noms communs avant passage à l'interpréteur (données transmises probablement sous forme de tuple ou dictionnaire)
- Interpréteur : Devra trouver sujets et verbes en priorité, lever les ambiguïtés, possibilité de chercher d'autres parties de phrases (ex : COD, compléments, etc)
- Main : Fera la liaison entre Parser et Interpréteur

Nous devions programmer tout cela en Python, langage jugé le plus apte pour effectuer efficacement du traitement de phrases, il n'y a pas eu d'utilisation de librairies externes excepté `re` pour parser les phrases.

Au début, nous avons pensé à utiliser la structure de données `dict` en Python qui se constitue d'attributs (clé, valeur), la clé aurait été chaque mot de la phrase associée à une valeur pouvant être Verbe, Nom, etc.
Il fut également décidé de traiter des cas de phrases très simples avec 1 verbe et 1 sujet au max, mais avec l'utilisation du dictionnaire de verbe, il était devenu alors très facile de détecter plusieurs sujets et verbes dans une même phrase.

Cependant, nous avons réalisé bien tard que cette structure de données n'admettait qu'un seul exemplaire unique de chaque mot ce qui était bien problématique. Nous avons donc réadapté tout le code vers la fin pour remplacer les dictionnaires par des tuples (mot, fonction).

La structure finale est donc une liste de tuples (mot, fonction) avec fonction limitée à : Nom, Verbe, Verbe Infinitif, Déterminant, Ponctuation, Pronom, Nom Propre, Sujet et Nom Propre Sujet. Cette structure sera nommée dico_tuple en général dans le code, nous l'appellerons DT dans ce rapport.

Parser

Si le Parser était prévu au départ pour contenir pas mal de fonctions, avec l'utilisation du dictionnaire Lefff son rôle a très vite décliné. Il ne contient plus grand chose désormais, une fonction de parsing renvoyant une liste contenant les mots d'une phrase qui étaient séparés par de la ponctuation, des espaces etc ainsi qu'une fonction qui va initier le DT en plaçant un placeholder 'Nom' attribué pour chaque mot de la phrase.

Main

Comme son nom l'indique, le Main permet d'initialiser le programme et relie le Parser et l'Interpréteur. Il va appeler petit à petit chaque fonction du Parser et de l'Interpréteur pour obtenir à la fin une représentation de la phrase entrée mot par mot. Cette partie du programme s'occupe également de l'affichage de la phrase une fois l'interprétation terminée.

Interpréteur

Il s'agit de la partie la plus technique et dense du projet, elle permet de détecter les verbes, sujets et de lever une partie des ambiguïtés, pour cette partie nous allons détailler quelques fonctions qui la composent afin d'expliciter les méthodes mises en place pour traiter les différents cas possibles.

Recherche du verbe

Pour trouver le verbe (fonction `search_verbe()`), nous commençons par ouvrir le fichier contenant le dictionnaire de verbe et allons l'itérer ligne par ligne (chaque ligne contenant la description pour un verbe). Pour chaque ligne itérée, nous allons également itérer un mot de la phrase découpée, et pour chaque mot, nous allons transmettre ces 2 données à une autre fonction (`comp_dico()`) qui va vérifier si les 2 mots correspondent, si oui, elle renvoie un tuple contenant le mot de la phrase et sa fonction (Verbe) + sa référence Lefff (permettant

de connaître la personne de la conjugaison). Elle peut également cumuler les références Lefff s'il y en a plusieurs et les verbes à l'infinitif sont renvoyés sans référence, s'il ne s'agit pas d'un verbe, alors elle renvoie None prévenant la fonction appelante qu'il ne faut pas toucher à ce mot.

Etablissement de la piste du sujet

Une fois le verbe trouvé, il va falloir déterminer à quoi ressemblera le sujet, c'est là qu'interviennent les références Lefff, une fonction (`traitement_forme_verbale()`) va donc créer un tableau et y ajouter chaque sujet possible pour ce verbe selon la référence apportée. S'il s'agit d'un verbe à la 3ème personne du singulier, cette fonction va ajouter la balise '3s', cette balise indiquera qu'un nom propre est un sujet potentiel d'une phrase. Une fois la piste du sujet déterminée, la fonction va donc renvoyer un tableau contenant ces pistes.

Bien évidemment cette méthode a un défaut certain, elle est trop basique, si un verbe a une terminaison à la 2ème personne du singulier par exemple, la seule piste qui sera proposée pour le sujet sera 'tu' et rien d'autre.

Recherche du sujet

Tout est désormais en place pour la recherche du sujet, nous commençons en premier lieu par itérer chaque élément de la liste fournie par la recherche de la piste sujet puis par itérer chaque tuple du DT. En premier lieu, si la balise '3s' est retrouvée, l'on recherche alors les sujets nom propre, une fois cela fait on retourne sur les éléments de la piste du sujet.
A chaque fois, l'on compare si le mot de la phrase correspond à une des pistes du sujet, et si oui on change le tuple en conséquence.

A la base il était décidé qu'un mot était sujet si et seulement si, il était présent dans la piste du sujet et qu'il était suivi d'un verbe, l'implémentation de base arrêtait justement de chercher une fois un verbe atteint. Mais avec l'appui du dictionnaire et en supposant qu'une phrase était écrite dans un français suivant les règles de la grammaire, la 2ème condition a été abandonnée.

Cas à part, le traitement du sujet dans une phrase interrogative se fait différemment d'une phrase classique, l'on regardera plutôt les '-' et les '-t-'. A la base Yahya devait s'en occuper mais ayant une lourde charge de travail l'empêchant d'être dans les meilleures conditions, cette partie a été faite légèrement en retard et ainsi compliquée à réadapter avec les nouvelles structures du code.
Notre programme détectera si une phrase est interrogative via le '? ' final, si vous le spécifiez en fin de phrase, attendez vous à voir une dé-construction simpliste avec 1 verbe et 1 sujet max.

Dernières retouches

Désormais que la phrase a été parsée et organisée de manière correcte, il reste quelques derniers cas à régler. En premier lieu une détection simple des noms propres (attention un nom propre en début de phrase sera ignoré) puis une levée de certaines ambiguïtés. Nous avons décidé de lever en priorité les ambiguïtés les plus fréquentes, le cas des déterminants (ex : la porte, porte ici n'est pas un verbe mais sera détectée en premier lieu comme tel). Pour cela, nous avons créé un fichier déterminants.txt regroupant tous les déterminants de la langue française, cela fait, il suffit de comparer les mots de la phrase à chacun de ces déterminants et si un verbe est directement suivi du dit déterminant, alors il est considéré comme un nom permettant ainsi de lever certaines ambiguïtés.

Une fois ces 2 étapes faites, le main devrait procéder à l'affichage.

Évaluation quantitative ou qualitative ?

A la fin de ce projet, on peut constater que nous ne sommes allés vers aucun extrême, si à la base il était décidé davantage de s'orienter vers une évaluation qualitative, nous avons finalement effectué une implémentation entre les 2.

Le dictionnaire Lefff permettant une analyse profonde et quasiment parfaite des verbes et donc de détecter plusieurs verbes dans une même phrase sans grand problème.

En revanche ayant fourni beaucoup d'effort sur ce point, les phrases interrogatives sont susceptibles de moins marcher sans compter les '-' s'ils sont présents dans la phrase qui ne seront pas retirés détectant donc 'voulez-vous' par exemple comme une seule entité Nom au lieu d'un Verbe et d'un Sujet.

Il n'est pas exclu également qu'ayant un sujet éloigné d'un verbe d'un complément trop long ou autre empêche leur détection correcte. Sur une échelle de 1 à 5, je dirais que nous sommes à environ 4/5 de la quantitative et à environ 3,5/5 de la qualitative.

Perspectives d'amélioration

Un premier point important est l'évaluation de phrases interrogatives, il faudrait impérativement réadapter ce code pour mieux y détecter les sujets et verbes et également passer cette analyse à une recherche plus approfondie avec nos fonctions de recherche de nom propre etc.

Notre programme permet d'évaluer les phrases seulement 1 par 1, on pourrait ajouter à ce programme une fonctionnalité qui lui permettrait de gérer des paragraphes entiers plutôt.

Il serait également possible d'affiner davantage la recherche de verbes/sujets pour être sûr de tous les détecter et peut-être aussi d'y ajouter le temps de conjugaison.

Niveau ambiguïtés, nous n'avons traité qu'un seul cas celui des déterminants, l'on pourrait le faire pour bien d'autres cas tels que les adverbes ou autres mais nous n'avions pas trouvé

d’autre moyen mis à part créer un nouveau fichier dictionnaire avec tous les adverbes puis l’itérer ce qui aurait été un peu ennuyeux.

En termes de complexité, notre programme peut prendre de longues secondes à analyser une longue phrase, l’on pourrait également réduire ce temps en effectuant moins d’itération.

Notre programme ne comprend pas particulièrement ce qu’est la différence entre un nom propre et un mot en majuscule, pour lui un nom propre est un mot commençant par une majuscule et n’étant pas en début de phrase (ainsi si un prénom comme Julie était écrit ‘julie’ le programme le considérerait comme un Nom normal). Il serait probablement possible de remédier à cela avec un dictionnaire contenant des noms propres ou tous les mots communs afin qu’il puisse comparer.

A noter également que la phrase fournit au programme doit être écrite sans faute d’orthographe, une seule faute d’orthographe et un verbe pourrait être étiqueté comme un Nom.

Si notre programme détecte les déterminants et noms propres en plus, il ne distingue pas la différence entre adjectifs, COD, compléments, etc. L’on pourrait lui permettre de détecter tout cela mais n’ayant pas trouvé de méthodes efficaces pour y aboutir, ces options n’ont pas été traitées.

Exécution du programme

Le fichier main.py relie déjà chaque partie du programme, il suffit donc pour l’exécuter d’entrer à la racine du projet la commande : `python3 main.py`
Le programme devrait alors afficher ‘Veuillez entrer une phrase à parser : ’ il ne vous reste plus qu’à rentrer la phrase que vous voulez analyser puis le programme va s’en charger.

Fichiers complémentaires

En plus des fichiers sources en .py et des fichiers .txt utilisés (localisés dans le dossier ressources/), il y a 2 autres fichiers non utilisés :

- Le fichier tests.txt, il regroupe les quelques tests effectués sur le programme et fournit donc une certaine diversité supplémentaire de test
- Le fichier conjugaison.txt, il ne sert à rien mais contient les terminaisons des verbes au présent sur lesquelles nous avions travaillé au début puis abandonné pour l’usage du dictionnaire

Conclusion

Nous avons appris avec ce projet que l'analyse d'une phrase et sa représentation linguistique étaient des tâches bien ardues. En essayant de "réinventer la roue" nous nous sommes aperçu qu'il était très difficile de démarrer d'autant plus que nous manquions d'outils au début ce qui nous a beaucoup ralenti. Au final nous avons compris qu'une phrase avait une structure complexe et que son analyse est loin d'être achevable à 100% d'autant plus qu'une langue change avec le temps et qu'un tel programme se doit donc d'évoluer en permanence. Et pour ce faire, il faut absolument être assisté par d'autres personnes pour construire morceau par morceau un tel mécanisme, sans les bons outils et la bonne aide un tel programme serait impossible à créer.