UCLouvain
École polytechnique de Louvain

Concepts, paradigmes et sémantiques des langages en informatique

LINFO1104

Twit'Oz : le prédicateur de texte

Jadoul Nicolas NOMA = 2164-18-00

Nimbona Davy NOMA = 1013-19-01

Professeur : M. Peter Van Roy

Tuteurs : M. Antoine Vanderschueren

M. Hadrien Libioulle

Année académique 2019-2020

Session de juin

Introduction

Dans le cadre du projet du cours de concepts, paradigmes et sémantiques de langage de programmation il nous est demandé de concevoir une application de prédicateur de texte, à base de tweets déjà fournis, sur base de ce que l'utilisateur écrit. Le programme doit pouvoir prédire sur base de minimum 1 mot écrit et utiliser de la récursion terminale. Nous allons donc vous présenter comment nous avons implémenter ce code en OZ en 4 points : la structure du programme, son algorithme, les choix de conception des threads et leur synchronisation et enfin l'interface où l'utilisateur pourra écrire.

1 Structure

La structure du programme est ainsi implémentée : dans Reader.oz nous lisons tout les tweets issus des 208 fichiers, dans Parse.oz nous allons parser les lignes de tweets, dans Dict.oz nous créons les dictionnaires qui stockeront tous les mots et son mot suivant et enfin dans main.oz la synchronisation des threads et l'interface.

2 Algorithme

L'algorithme est en grande partie basée sur la récursion terminale, le programme est exécuté en appelant la fonction *ReadTweets*, celle-ci, à l'aide de *FullScan*, lit toutes les lignes de tweets se trouvant dans les fichiers ; *ParseTweets*, à l'aide de *TweetToListOfWord*, prend en argument les lignes de tweets et retourne une matrice contenant les mots de chaque ligne lue groupé par tweet ; *WordsLink*, à l'aide de *WordsLinkTweet* prend en argument la matrice des mots groupés par tweet et retourne une liste de tuples composé d'un ou deux mots et de son possible successeur ; *DicFreq* prend en argument un dictionnaire vide et la liste de tuple et retourne un dictionnaire D contenant comme clé tous les mots avec comme valeur un dictionnaire DNext. Ce dernier a comme clé les mots suivants possibles et la fréquence comme valeur ; *FinalDictionary* ; à l'aide de *FindMaxFreq*, trouve le successeur avec la plus grande fréquence pour chaque mot du dictionnaire D. Ce successeur deviendra la valeur de son prédécesseur dans le dictionnaire D ; *FindNext* prend en arguments les deux derniers mots écrits et retourne le mot suivant si correspondance dans un des deux dictionnaires (1-gramme et 2-gramme) ou "Aucune proposition" si ce n'est pas le cas.

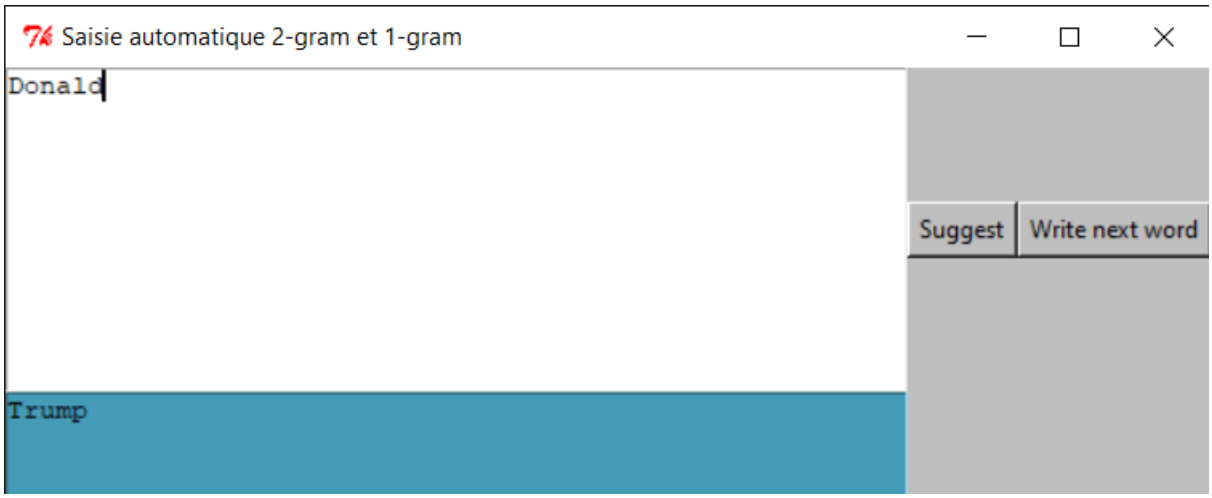
3 Threads

Exécutant notre programme sur des ordinateurs de maison contenant généralement 4 cœurs, nous avons opté pour que celui ci utilise constamment quatre threads, excepté pour la création des dictionnaires. Nous avons donc quatre threads de lecture, quatre premiers threads de parsing

suivis de quatre autres. Ces threads travaille sur des streams et peuvent donc s'enchaîner plus rapidement. Par exemple si un des threads de lecture se finit avant les autres, le thread de parsing lié à ce dernier peut commencer sans devoir attendre que les trois autres threads de lecture se terminent. Pour les dictionnaires il faut éviter que deux threads en modifient un en même temps sinon des erreurs peuvent se produire. Puisque nous n'avons que deux dictionnaires (1-gram et 2-gram), nous n'aurons que deux threads assignés à un dictionnaire chacun.

4 Interface

Notre interface est une interface avec un fond blanc pour la partie 'saisie texte' et bleue pour la partie 'mot suivant proposé. À droite nous avons deux boutons : *Suggest* qui propose un possible mot successeur à ce qui est écrit dans la partie 'saisie texte' et *Write next word* qui écrit dans la partie 'saisie texte' le mot proposé par *Suggest* et appelle sa procédure pour une proposition automatique.



Conclusion

Nous avons réussi à implémenter un code qui est exécuté avec plusieurs threads (quatre). Notre saisie automatique arrive à prédire un mot en prenant en compte deux mots qui le précèdent. Le langage OZ rend très facile et efficace l'utilisation de la concurrence. Malheureusement nous nous sommes pas attaqués au parsing des majuscules et de la ponctuation.