Université de Provence - Aix-Marseille 1
Licence d'informatique

Automates et Langages : analyseur lexical Python

Victor Martin
<victormartinfr@gmail.com>

Marseille, le 28 avril 2011

Table des matières

Chapitre

1 Présentation du projet

Le but de ce projet est de construire un analyseur lexical en Python, pour du code Python. Ce projet etant effectué dans le cadre du cours d'automates et langages, il fallait bien evidemment modéliser des automates en Python pour sa réussite. J'ai effectué la partie concernant les automates finis déterminites. J'ai utilisé les modules copy, time et sys.

Le programme s'utilise comme suit :

1. On créé un fichier *in*.
2. On lance dans une console : *python projet.py in out*

Chapitre 2

Présentation des automates

Voici les différents automates. Dans les graphes, V signifie tout l'alphabet, soit les 256 caractères et V \ {0, 1} signifie tout l'alphabet privé du 0 et du 1.

2.1 L'automate des nombres

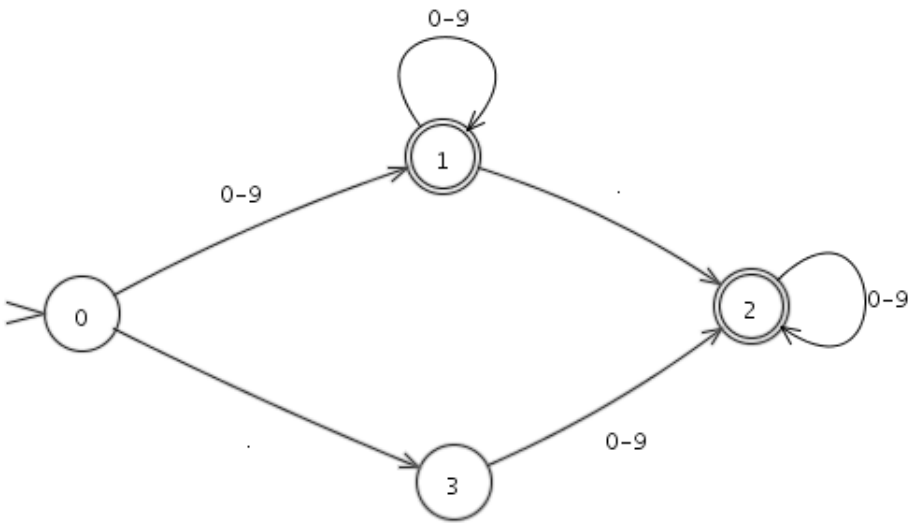

Figure 2.1 – Automate qui reconnait les nombres (AutNum)

2.2 L'automate des chaines de caractères

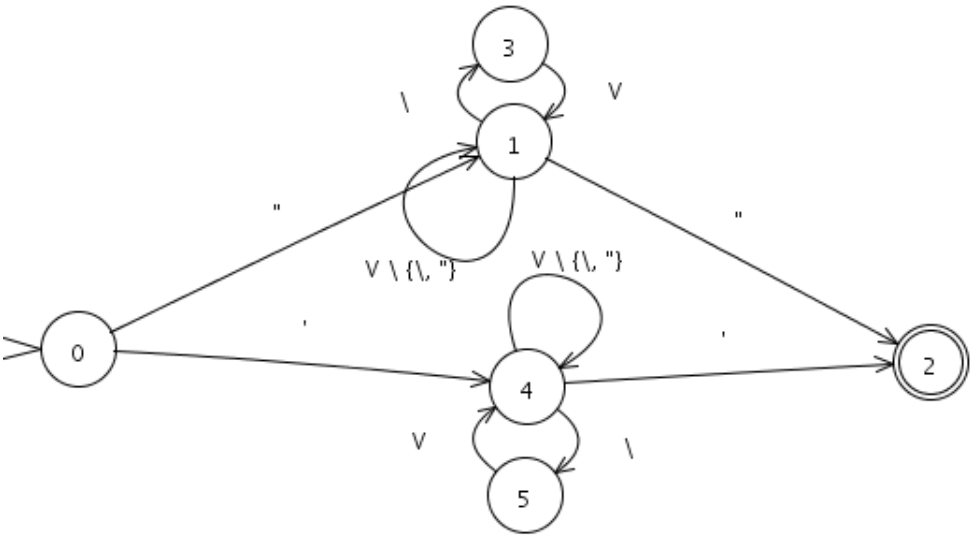

FIGURE 2.2 – Automate qui reconnait les chaines de caractères (AutCar)

2.3 L'automate des mots clefs et l'automate des symboles

Concernant l'automate des mots clefs et celui des symboles , il a fallu développer un outil permettant de créer un automate qui reconnaisse les mots d'une liste.

```
1  def initTab(Auto, listeMot):
2
3    """ Permet d'initialiser un automate de telle sorte qu'il ne reconnaisse que les mots dans le
         tableau donne en parametre """
4
5    nbEtat = 0 #Nombre d'etats total
6    for mot in listeMot:
7      i=0
8      etatActuel = 0
9      while i < len(mot): #On parcoure caractere apres caractere
10       etat = Auto.delta[etatActuel][ord(mot[i])]  #Etat de l'automate apres la lecture du
             caractere.
11       if etat != -1:  #Si la transition a deja ete definie pour ce caractere, on va a l'etat deja
             existant
12         etatActuel = etat
13       else:          #Sinon on fais une transition vers un etat encore inutilise, et on repars de
             ce nouvel etat.
14         nbEtat += 1
15         Auto.delta[etatActuel][ord(mot[i])] = nbEtat
16         etatActuel = nbEtat
17       i+=1
18     Auto.F += [etatActuel]
```

Ici, on va prendre un automate en argument et une liste de mots, et modifier l'automate de telle sorte qu'il reconnaisse les mots de la liste.

Pour chaque mot, on part de l'état initial. Pour chaque lettre du mot, si la transition est déjà défini on va à l'état correspondant, sinon on créé la transition vers un état non utilisé. Quand on a fini de lire le mot, on peut rajouter à la liste des états finaux l'état dans lequel on se trouve (l'état après avoir lu la dernière lettre du mot) (ligne 18). Et on recommence ensuite avec le mot suivant.

2.4 L'automate des identificateurs

L'automate des identificateurs est très proche de l'automate des mots-clefs. Voici comment se déroule l'algorithme effectuant son initialisation :

1. On copie la table de transition de AutCl dans AutId. On rajoute un état acceptant λ dans AutId.
2. Tous les états non-acceptant dans AutCl sont acceptant dans AutId (et inversement).
3. Pour chaque état de AutId :
   (a) Si la transition après un caractère possible dans une variable (a-zA-Z0-9_) mène vers un état rebus (-1) :
      i. On redéfinit cette transition pour qu'elle mène vers l'état λ, et non plus l'état rebus.
4. Étant donné qu'une variable ne peut pas commencer par un chiffre, on définit toutes les transitions de l'état initial après lecture d'un chiffre, menant vers l'état rebus.

Chapitre 3

Présentation de l'analyseur lexical

Après initialisation de tous les automates et de toutes les variables globales, il faut passer à l'analyse. La fonction qui fait l'analyse lexical recoit en argument le texte pré-formaté (chaque ligne du texte est rangée dans un tableau et les \n sont ignorés). Voici une ébauche de la fonction d'analyse lexicale.
Pour chaque ligne :

1. On compte le retrait.
2. Si le retrait n'est pas suivi d'un commentaire ou d'un retour à la ligne :
   (a) Si le retrait est correct (s'il y a moins de deux débuts de procédure) alors pour la suite de la ligne :
      i. Si on ne détecte pas de séparateur, de commentaire ou chaine multilignes :
         A. On applique tous les automates, on ajoute la plus grande unité lexicale trouvée et on repart de la fin de cette unité.
         B. Un erreur si aucun automate ne reconnait.
      ii. Sinon s'il s'agit d'une chaine multiligne, on l'ajoute à la liste des unités lexicales.
      iii. Sinon on passe à la suite.
   (b) Sinon on retourne une erreur d'indentation.
3. Sinon on passe à la suite.

On remarque dans ce pseudo-code que l'on ajoute une unité lexicale invalidée par les automates : la chaine de caractères multilignes.
Effectivement, dans l'énoncé il est précisé que les blocs de la forme *""" ceci est un test """* étaient des commentaires multilignes. Or, ces blocs sont en faites des chaines multilignes. Ici, mon code ne tient pas compte des chaines multilignes après un retrait (considérées donc comme des commentaires multilignes), mais en tient compte lorsqu'il y a par exemple :
varStr1 = """ceci est
une
chaine """
Ainsi, on ne perd pas d'information en ajoutant ce type de chaines à la listes des unités lexicales, et on facilite le travail des analyseurs suivants en ne retenant pas celles présentent en début de ligne.

Dans le fichier constantes.py, certaines erreurs sont définies, ainsi l'analyseur retourne la position et le type de l'erreur, ce qui permet d'avoir un affichage de l'erreur et de sa position si erreur il y a.