Projet d'informatique théorique 2

Université Bordeaux 1 — 2012-2013
Licence Informatique 3ème année

L'objectif du projet est de réaliser un logiciel de reconnaissance de motifs dans un texte en utilisant des techniques vues en cours et TD sur les automates. Les fonctions à écrire sont détaillées ci-dessous. Une partie de ce projet présente un choix entre l'écriture de certaines fonctions et des questions de nature plus théorique.

Pour manipuler les automates, il est fourni une interface en Python. Elle permet d'accélérer le développement du projet, en ne se concentrant que sur les algorithmes. Il est permis d'utiliser un autre langage (C, Java par exemple). Dans ce cas, informez votre enseignant de TD du langage dans lequel vous souhaitez programmer.

1 Modalités de réalisation

Le projet est à réaliser par groupe de 2 personnes. Chacun des membres d'un groupe doit réaliser une partie significative du travail demandé. L'écriture seule du rapport, par exemple, est insuffisante. Les 2 membres d'un groupe doivent connaître les options générales choisies ainsi que les difficultés rencontrées. Chacun devra exposer individuellement sa contribution lors de la soutenance (semaines du 1er ou 8 avril). La notation pourra varier à l'intérieur d'un groupe en cas de travail trop inégal, et tout plagiat détecté sera sévèrement sanctionné.

Le projet sera présenté dans un court rapport (3 à 5 pages) illustré par des exemples, exposant les problèmes rencontrés et les choix effectués pour les résoudre.

Date limite de remise du source et du rapport par mail à it2@labri.fr : **28 mars à 20h**.

Le fichier transmis doit être un fichier de type `.tar.gz` contenant un seul répertoire à vos noms, dans lequel se trouvera :

- le fichier du rapport sous forme pdf : `rapport.pdf`.
- un répertoire `./src/` contenant les sources du projet.

En cas de difficultés ou de questions, n'hésitez pas à contacter l'équipe pédagogique it2@labri.fr.

2 La bibliothèque fournie

Une bibliothèque compatible Python 2 et 3 vous est fournie. Cette section la présente rapidement. Elle permet entre autres :

- de charger des automates prédéfinis dans un fichier de type XML. Un exemple de tel fichier est fourni : `http://www.labri.fr/~zeitoun/enseignement/12-13/AS+IT2/automata.xml` ;
- de créer des automates directement ;

– d'accéder aux différentes caractéristiques d'un automate, de récupérer les successeurs d'un état par une lettre, etc.
– d'afficher les automates.

La bibliothèque se trouve dans le fichier `automaton.py` fourni sur le site du cours à l'adresse :

http://www.labri.fr/∼zeitoun/enseignement/12-13/AS+IT2/automaton.py

Ne pas modifier cette bibliothèque. Cette bibliothèque est documentée : pour accéder à l'aide sous python, il suffit de taper `help()`, puis `automaton`. Les fonctions sont décrites et présentées avec des exemples. Il est aussi possible de consulter la documentation à l'aide d'une navigateur web en lisant le fichier `automaton.html` présent à l'adresse

http://www.labri.fr/∼zeitoun/enseignement/12-13/AS+IT2/automaton.html.

La plupart des fonctions sont faciles à comprendre. Un point particulier est la gestion des ε-transitions : pour les représenter, on décide d'une ou plusieurs lettres jouant le rôle d'ε. Par exemple, le code suivant permet de créer et d'afficher un automate dans lequel le caractère `'0'` est utilisé pour représenter ε.

```
import automaton
aut1 = automaton.automaton(
    epsilons = ['0'],
    states = [5], initials = [0,1], finals = [3,4],
    transitions=[
        (0,'a',1), (1,'b',2), (2,'b',2), (2,'0',3), (3,'a',4)
    ]
)
aut1.display()
```

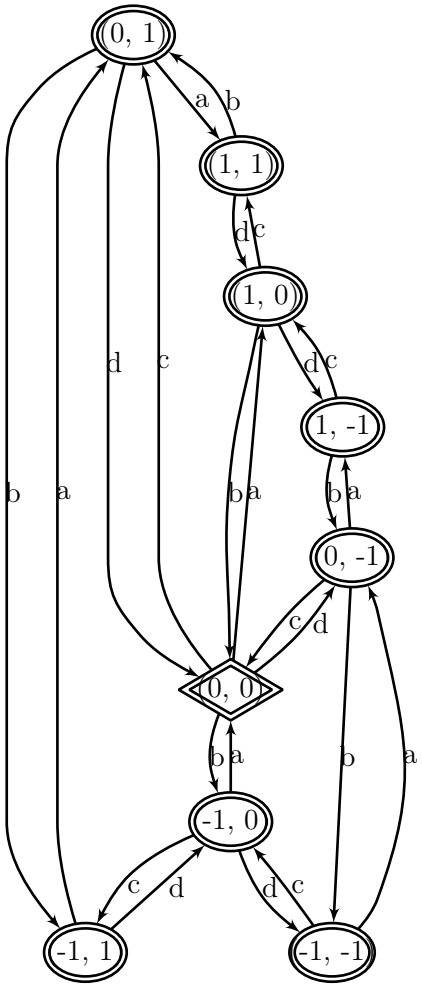
Plusieurs lettres peuvent être interprétées comme des ε-transitions dans un même automate. Les ensembles d'états ou de lettres retournés par les fonctions sont de type `automaton.pretty_set` qui hérite de `frozenset`, permettant les manipulations habituelles sur les ensembles.
Voir http://docs.python.org/2/library/stdtypes.html#frozenset.

Il est possible d'utiliser des tuples pour coder les états d'un automate. Par exemple, voici le programme qui construit un automate qui reconnaît tout les mots m de $\{a, b, c, d\}^*$ tel que pour tout préfixe w de m on ait $||w|_a - |w|_b| \leq 1$ et $||w|_c - |w|_d| \leq 1$.

```
import automaton
def automate():
    alphabet = [ 'a', 'b', 'c', 'd' ]
    epsilons = []
    initiaux = [ (0,0) ]
    etats = []
    finaux = []
    transitions = []
    for a_moins_b in [0,1,-1]:
        for c_moins_d in [0, 1, -1]:
            etats.append( (a_moins_b, c_moins_d) )
            finaux.append( (a_moins_b, c_moins_d) )
    for origine in etats:
        for lettre in alphabet:
```

```
            if lettre == 'a' :
                fin = ( origine[0]+1, origine[1] )
            elif lettre == 'b' :
                fin = ( origine[0]-1, origine[1] )
            elif lettre == 'c' :
                fin = ( origine[0], origine[1]+1 )
            else:
                fin = ( origine[0], origine[1]-1 )
            if abs(fin[0]) <= 1 and abs(fin[1]) <= 1 :
                transitions.append( (origine,lettre,fin) )
    return automaton.automaton(
        alphabet, epsilons, etats, initiaux, finaux, transitions
    )
B = automate( )
B.display()
```

Ce programme affiche l'automate suivant :

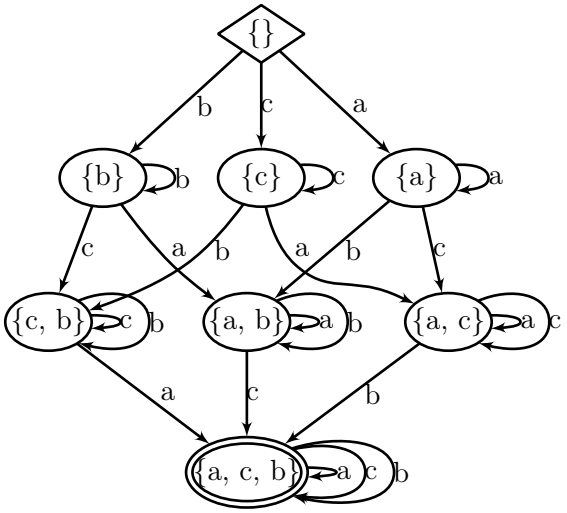


En utilisant `automaton.pretty_set`, il est possible d'utiliser des ensembles pour coder les états d'un automate. Par exemple, si l'on note par A l'alphabet, voici un programme qui construit

l'automate qui reconnaît l'ensemble des mots possédant toutes les lettres de A :

```
import automaton
def automate_des_mots_contenant_l_alphabet( alphabet ):
    initiaux = [ automaton.pretty_set() ]
    epsilons = []
    finaux =  [ automaton.pretty_set(alphabet) ]
    etats = set( automaton.pretty_set() )
    pile = [ automaton.pretty_set() ]
    transitions = []
    while len( pile ) > 0:
        origine = pile.pop()
        for lettre in alphabet:
            fin = origine.union( [lettre] )
            if not( fin in etats ):
                etats.add( fin )
                pile.append( fin )
            transitions.append( (origine, lettre, fin) )
    return automaton.automaton(
        alphabet, epsilons, etats, initiaux, finaux, transitions
    )
A = automate_des_mots_contenant_l_alphabet( ['a','b','c'] )
A.display()
```

Ce programme affiche l'automate suivant :



Quelques exemples de fonctions

Cette section décrit quelques fonctions disponibles dans la bibliothèque. Pour l'aide complète : `help()` puis `automaton`. Il est conseillé de lire le fichier `automaton.py` avant de commencer à travailler. Les fonctions ont en général des arguments optionnels qui changent leur comportement par défaut. Par exemple, on peut préciser pour la fonction `display()` qui affiche un automate un titre de figure, et un booléen permettant d'attendre ou non que l'utilisateur ferme la fenêtre graphique pour continuer. Consulter l'aide pour connaître les options éventuelles. Les premières fonctions à comprendre sont les suivantes :

- `Aut.has_epsilon_characters()` : teste si l'automate `Aut` a des ε-transitions.
- `Aut.get_alphabet()` : retourne l'alphabet complet de l'automate `Aut`.
- `Aut.get_epsilons()` : retourne l'ensemble des lettres représentant ε dans `Aut`.
- `Aut.get_states()`, `Aut.get_initial_states()` et `Aut.get_final_states()` retournent les ensembles d'états, d'états initiaux et d'états finaux de `Aut`.
- `Aut.display()` : dessine et affiche l'automate `Aut`.
- `Aut.delta()` et `Aut.delta_star()` : calculent les successeurs d'un d'état (ou plus généralement d'un ensemble d'états) par une lettre (pour `delta()`) ou un mot (pour `delta_star()`). La documentation explique comment sont traitées les ε-transitions.

3 Travail demandé

Il est demandé d'écrire les fonctions suivantes, en suivant les algorithmes du cours ou des TD. Les automates arguments peuvent être non déterministes et/ou incomplets. Pour pouvoir tester les programmes, il est demandé de respecter les noms de fonctions ci-dessous.

- Une fonction `completer( Aut )` qui et retourne l'automate complété de l'automate `Aut`.
- Des fonctions `union( Aut1, Aut2 )`, `intersection( Aut1, Aut2 )` calculant des automates pour l'union et l'intersection des langages reconnus par les automates de `Aut1` et `Aut2`.
- Une fonction `miroir(Aut)` qui calcule l'automate miroir d'un automate `Aut` : il est obtenu en retournant les flèches, en déclarant initiaux les états qui étaient finaux dans `Aut` et en déclarant finaux les états qui étaient initiaux dans `Aut`.
- Une fonction `determinisation( Aut )` qui et retourne l'automate déterminisé calculé à partir de l'automate `Aut`, en suivant l'algorithme vu en cours et TD.
- Une fonction `complement( Aut )` calculant le complément du langage reconnu par `Aut`.
- Une fonction `minimiser( Aut )` qui calcule l'automate minimal de l'automate `Aut`. On pourra utiliser soit l'algorithme de Moore, soit l'algorithme du double renversement décrit dans la feuille de TD n° 3.
- Une fonction `expression_vers_automate( E )` qui prend en argument une expression rationnelle et qui construit l'automate minimal équivalent à cette expression. On pourra utiliser l'algorithme de Thomson, plus rapide à écrire que celui de Glushkov.
  L'expression sera codée en format préfixe, sous forme de liste : par exemple, l'expression $(a+b^*a)^*$ sera représentée par la liste `E = ["*", ["+", ["a", [".", ["*","b"], ["a"]]]]]`.

Choisissez au moins une de ces questions pour terminer :

a) Montrer l'algorithme du double renversement, comme proposé dans la feuille de TD n° 3.

b) Écrire un analyseur d'expressions rationnelles, qui convertit les expressions rationnelles, données comme chaînes de caractères de la forme `(a + b * a)*`, en listes Python, de la forme `["*", ["+", ["a", [".", ["*","b"], ["a"]]]]]`.

A Installer la bibliothèque d'automate sur son ordinateur

Pour utiliser la bibliothèque sur votre ordinateur, vous devez :

- Installer Python (au moins la version 2.6)
- Installer GraphViz
- Télécharger la bibliothèque automaton.py à l'URL donnée plus haut.

A.1 Sous debian et ubuntu

Python est déjà installé sous Ubuntu et Debian.

Il ne vous reste plus qu'à installer graphviz en tapant la ligne de commande suivante :

```
sudo apt-get install graphviz
```

A.2 Sous MacOSX

Python est déjà installé sous Ubuntu et Debian.

Pour installer GraphViz, vous devez installer le paquet correspondant à votre version du système d'exploitation présent à l'adresse :

http://graphviz.org/Download_macos.php.

A.3 Sous Windows

Pour installer Python, vous devez installer le paquet d'installation (correspondant à votre version du système d'exploitation) présent à l'adresse :

http://www.python.org/download/.

Pour installer GraphViz, vous devez installer le paquet (correspondant à votre version du système d'exploitation) situé à l'adresse :

http://graphviz.org/Download_windows.php.

Ajouter dans la variable d'environement PATH de windows, les répertoires des éxecutables de python et de graphviz. Pour ce faire, vous devez éditer la valeur de la variable PATH qui se trouve à :

```
demarre->ordinateur(click droit)->propriétés->Modifier les paramètres->
Paramètres Système avancés ->Variables d'environnements->Variables systèmes->Path->
Modifier
```

en ajoutant à la fin de la ligne, le texte suivant :

```
;C:Program Files (x86)\Graphviz2.30\bin;C:\Python33
```

(si vous avez téléchargé la version 2.30 de GraphViz et la version 3.3 de Python).