Projet TP - Langages formels

Nathan Lhote, Guillaume Maurras, Marie-Hélène Stefanini

2022-2023

Présentation du projet

L'objectif de ce projet est d'implémenter des algorithmes étudiés en cours à l'aide d'une bibliothèque **Python** pour définir et manipuler des automates à pile et des grammaires. La bibliothèque `contextfree.py` fournie comporte les fonctionnalités de base dont vous aurez besoin. Chacune des trois parties devrait pouvoir être faite au cours d'une séance de TP. Vous pouvez bien entendu travailler entre les séances.

Rendu de TP

Le rendu sera fait par groupe de 2. Ce projet doit être complété et soumis sur Ametice le jour du dernier TP. Vous rendrez ce projet sous forme d'une archive `.zip`. L'archive devra contenir, en plus du fichier contenant votre code `tp-langages.py`, un dossier `tests` comportant plusieurs fichiers `.pa` et `.gr`. Vous devrez également ajouter un fichier readme contenant le nom des membres du binôme. L'archive devra obligatoirement être nommée:

`NOMDEFAMILLE1-Prenom1_NOMDEFAMILLE2-Prenom2.zip`

Attention: les noms des fichiers et des fonctions que vous ajouterez devront être **exactement** ceux donnés par le sujet pour pouvoir être évalués. De plus vous ne pouvez pas modifier les fonctions déjà fournies de la bibliothèque.

0 Présentation de la bibliothèque

0.1 Création d'un automate à pile

Un automate à pile est donné par cinq attributs: un nom au format `str`, un état initial (`str`), un symbole de pile initial (`str`), une liste d'états finaux (liste de `str`) et une liste de transitions: liste de quintuplets (état de départ, lettre, tête de pile, liste de symboles à empiler, état d'arrivée) où la lettre doit être un caractère (`chr`). Exécutez le main du fichier `tp-langages.py`. Vous pouvez voir dans la console un automate vide. Ajoutez les lignes suivantes avant le print:

```
a.add_transition("0","a","Z0",["A","Z0"],"1")
a.add_transition("1","b","A",[],"0")
a.set_initialstate("0")
a.set_initialstack("Z0")
a.make_final("1")
```

Identifiez dans la console, l'ensemble des états, l'alphabet, l'alphabet de pile, l'état initial, les états finaux et les transitions de l'automate `a`. Dessinez l'automate `a`.

0.2 Fichier automate

La bibliothèque vous fournit une méthode pour stocker un automate à pile dans un fichier texte avec l'extension `.pa`. Exécutez la ligne:

`a.to_txtfile("./tests/automaton0.pa")`

Vérifiez que le dossier `tests` contient bien un fichier `automaton0.pa`, et lisez son contenu: la première ligne indique l'état initial, la seconde ligne les états finaux et la troisième le symbole de pile initial. Les lignes suivantes décrivent les transitions de l'automate.

Exécutez:

```
a1=StackAutomaton("aut1")
a1.from_txtfile("./tests/automaton1.pa")
print(a1)
```

Ouvrez le fichier `automaton1.pa` et notez que cet automate comporte un symbole spécial qui n'est pas affiché dans l'alphabet. En effet le symbole `%` est réservé pour représenter le mot vide ε. Dessinez sur une feuille l'automate `a1`. Quel est le langage reconnu par cet automate ? Faites de même pour l'automate contenu dans le fichier `automaton2.pa`.

0.3 Autres fonctionnalités

Explorez les différentes méthodes de la bibliothèque en exécutant:

```
print(a1.get_alphabet())
print(a1.get_stackalphabet())
print(a1.get_states())
print(a1.get_transitions())
a1.make_copy(a)
print(a1)
```

Note: vous pouvez parcourir l'ensemble des transitions d'un automate à pile `a` de la façon suivante:

```
for (source, letter, head, push, target) in a.get_transitions():
```

0.4 Grammaires

La bibliothèque `contextfree.py` permet également de manipuler des grammaires.

Exécutez:

```
g=Grammar("gr")
g.add_rule("X",["a","X","a"])
g.add_rule("X",["a","a"])
g.set_axiom("X")
print(g)
```

De même que pour les automates à piles, vous pouvez générer un fichier à partir d'une grammaire:

`g.to_txtfile("./tests/grammar0.gr")`

ou bien importer une grammaire à partir d'un fichier. Exécutez:

```
g1=Grammar("gr1")
g1.from_txtfile("./tests/grammar1.gr")
print(g1)
```

Quels sont les langages générés par `g, g1` ? Et pour la grammaire correspondant au fichier `grammar2.gr` ?

Explorez les différentes méthodes de la bibliothèque en exécutant:

```
print(g1.get_alphabet())
print(g1.get_symbolalphabet())
print(g1.get_rules())
print(g1)
```

Note: vous pouvez parcourir l'ensemble des règles d'une grammaire `g` de la façon suivante:

```
for (symbol,repˆlace) in gr.get_rules():
```

0.5 Création de fichiers test

Créez les 4 fichiers suivants dans le dossier tests:

- `automaton_anbn.ap` un automate à pile déterministe reconnaissant $\{a^n b^n \mid n > 0\}$.
- `automaton_palindromes.ap` un automate à pile reconnaissant les palindromes sur l'alphabet $\{a, b\}$.
- `grammar_anbn.gr` une grammaire générant $\{a^n b^n \mid n > 0\}$.
- `grammar_palindromes.ap` une grammaire générant les palindromes sur l'alphabet $\{a, b\}$.

Il est fortement encouragé de créer d'autres fichiers `.ap` et `.gr` et de vous aider de ces fichiers pour tester votre code régulièrement. Par ailleurs ces fichiers seront également utilisés pour vous évaluer.

Le projet est découpé en trois parties. **Pensez à tester votre code régulièrement**.

1 Exécution d'un automate

Le but de cette partie est de définir une fonction qui exécute un automate à pile déterministe sur un mot.

1.1 Automate déterministe

Définir une fonction `is_deterministic` telle que `is_deterministic(a)` renvoie `True` si l'automate a est déterministe et `False` sinon.

1.2 Exécution

Définissez une fonction `execute` telle que `execute(a,s)` renvoie la chaîne de caractères `"ERROR"` si l'automate `a` est non-déterministe. Dans le cas contraire la fonction doit renvoyer `True` si la chaîne de caractères `s` est acceptée par l'automate et `False` sinon.

2 Forme normale de Chomsky

Le but de cette partie est de définir une fonction `cnf` telle que `cnf(g)` transforme une grammaire `g` en grammaire sous forme normale de Chomsky.

2.1 Test de forme normale

Premièrement vous devez définir une fonction `is_cnf` pour tester si une grammaire est sous forme normale de Chomsky.

2.2 Forme normale

Deuxièmement, vous devez définir cinq fonctions `step_1,step_2,step_3,step_4,step_5` une pour chaque étape de la mise sous forme normale de Chomsky.

- étape 1: Suppression de l'axiome dans le membre droit des règles.
- étape 2: Suppression des terminaux dans les membres droits des règles de longueur au moins 2.
- étape 3: Réduction de la longueur des membres droits des règles à 2 maximum.
- étape 4: Élimination des règles ε.
- étape 5: Suppression des règles unitaires (par ex: $X \to Y$).

3 Algorithme CYK

Vous devez définir une fonction `cyk` implémentant l'algorithme. Exécuter `cyk(g,s)` doit 1) mettre `g` sous forme normale de Chomsky si ce n'est déjà le cas et 2) renvoyer `True` ou `False` suivant si la grammaire génère ou non la chaîne de caractère `s`.

Vous pouvez créer un tableau de $n \times n$ listes vides en exécutant:
`[[[] for i in range(n)] for i in range(n)]`