Génération d'objets combinatoires décrits par une grammaire

Djamal Abdelkader et Julien Guyot

1 Introduction

Ce rapport retrace ce qui a été fait durant le projet de cette UE. Nous présentons dans un premier temps le plan de ce rapport:

1. Résumé de ce qui a été mis en oeuvre durant ce projet.
2. Questions sur papier
3. Présentation et explication du code

2 Implémentation

Nous avons implémenté les fonctionnalités suivantes :

1. Calcul de la valuation d'une règle (méthode valuation)
2. Comptage du nombre d'objets de taille donnée dérivant d'une règle (count)
3. Calcul de la liste des objets
4. Méthode unrank
5. Méthode rank
6. Caching
7. Extension de l'expressivité des grammaires

Nous reviendront plus en détail sur les implémentations de ces fonctionnalités plus tard dans le rapport. La plupart des fonctionnalités sont testées, dans le fichier test_projet.py . Les fonctions sont commentées, expliquant plus ou moins leur utilité, leur paramètres ou retour, ainsi que leur fonctionnement.

3 A faire sur papier

Partie 2.2 :

1. Rendu des appels à la fonction count

taille de l'objet	Fib	Cas1	Cas2	CasAU	CasBAu
0	1	0	0	0	0
1	2	2	1	1	0
2	3	3	1	2	1
3	5	5	2	3	2
4	8	8	3	5	3
5	13	13	5	8	5
6	21	21	8	13	8
7	34	34	13	21	13
8	55	55	21	34	21
9	89	89	34	55	34
10	144	144	55	89	55

Figure 1: Nombre d'objets dérivant de chaque règle de la grammaire des mots de Fibonacci, en fonction de leur taille

Taille de l'objet	Tree	Node
0	0	0
1	1	0
2	1	1
3	2	2
4	5	5
5	14	14
6	42	42
7	132	132
8	429	429
9	1430	1430
10	4862	4862

Figure 2: Nombre d'objets dérivant de chaque règle de la grammaire des arbres binaires, en fonction de leur taille

2. Grammaire des mots sur $\{a, b\}$ [1]

$$G = \quad aG|bG|\epsilon$$

3. Grammaire des mots bien de Dyck :

$$G = \quad (G)|\{G\}|,$$

4. Grammaire des mots n'ayant pas trois lettres de suite

$$G = \text{ aA } |bB|aaA|bbB|\epsilon$$

[1] Les symboles non-terminaux sont notés en majuscules, tandis que les terminaux sont notés en minuscules. ϵ est le mot vide

$$A = \text{"b" CasB |"bb" CasB } |\epsilon$$

$$B = aA|aaA|\epsilon$$

5. Palindromes:
   - Deux lettres : $G = aGa|bGb|a|b|\epsilon$
   - Trois lettres : $G = aGa|bGb|cGc|a|b|c|\epsilon$
6. Grammaire contenant le même nombre de a que de b:

$$G = aGbG|bGaG|\epsilon$$

7. fonction vérifiant une grammaire : voir le fichier

3.1 Tests de cohérence générique

$\forall n \in \mathbb{N} \quad C \in \text{AbstractRule} \quad i \in \{0...C.count(n) - 1\}$:

- C.count(n) = len(C.list(n))
- C.list(n)[i] = C.unrank(n, i)
- C.rank(n, C.unrank(n, i)) = i
- C.unrank(n, C.rank(obj)) = obj

4 Présentation du code

4.1 Structure

Le code est divisé en deux fichiers : projet.py, qui contient le code de ce qui a été demandé, ainsi que test_projet.py, qui contient des tests associés aux fonctionnalités du premier fichier.

Les types des paramètres sont notés dans un style ressemblant à OCaml dans les commentaires.

4.2 Règles de grammaire "simples" (dérivant de AbstractClass)

Ces classes correspondent précisément à ce qui a été donné dans le sujet, plus le constructeur Bound(). Ces classes implémentent les méthodes suivantes

valuation

count(n) Avec un "cache" qui correspond à un dictionnaire python. Il existe probablement des caches plus optimisés, mais cela semblait plus intéressant qu'une liste dans la mesure où cette structure de donnée est relativement optimisée pour la recherche d'une valeur à partir d'une clef

list(n) Dans notre implémentation, il y a un cas où list() peut boucler indéfiniment. Cela s'observe dans le dernier bloc du fichier de test

unrank(n, r)

random(n)

rank(n, obj) Notre implémentation nécessite de donner la taille n de l'objet obj [2]. ainsi que de donner une fonction de comparaison en paramètre au constructeur de UnionRule, ainsi que deux fonctions (une fonction calculant la taille de l'objet, et une fonction "destructeur" ayant l'effet inverse de la fonction cons)
Le fonctionnement de ces fonctions est souvent expliqué en commentaire

4.3 Règles de grammaire expressives

Dans ce cas, les classes sont très basiques, et la traduction d'une grammaire expressive vers une grammaire simple se fait par la fonction simplify_grammar. La règle sequence y a été ajoutée.

[2]cela n'est pas strictement nécessaire, mais ça évite d'une part d'avoir à donner une fonction d'évaluation de la taille à chaque constructeur, ainsi que de recalculer cette dernière à chaque appel