Projet de Combinatoire

Rapport: Thibault Martin
Binome: Stéphane Romany

1 Introduction

L'objectif de ce projet était de produire un code permettant, à partir de grammaires définies par l'utilisateur sous une forme précise, de récupérer le nombre d'éléments générés par cette grammaire (en fonction de la taille des éléments), d'en faire une liste et également permettre de récupérer le ième élément ou de trouver le rang d'un élément (sans parcourir tous les éléments). Ce projet est édité en python (Python 3.5).

2 Définitions formelles

Une grammaire est définie par l'utilisateur sous une forme précise permettant ensuite d'utiliser les fonctionnalités de notre programme.

- Singleton :
  Un singleton représente le plus petit objet d'une grammaire
  Exemple:
  Singleton("a"), Singleton("("), Singleton(Leaf), ...
- Epsilon :
  Epsilon désigne un objet vide, dans certains cas on veut la possibilité de ne rien écrire.
  Exemple:
  Epsilon("") (la chaine vide)
- Union :
  Union permet de faire un "choix" entre 2 possibilités et représente donc l'ensemble des éléments du choix 1 et 2.
  Exemple:
  Vide : Epsilon("")
  A : Singleton("a")
  Union(Vide,A) est l'ensemble des éléments de Vide et A, c'est à dire {"","a"}
- Produit :
  Un produit permet de faire un produit cartésien entre 2 ensembles.
  Exemple:
  A : Singleton("a")
  B : Singleton("b")
  U : Union(Vide,A) Produit(U,B) est l'ensemble des éléments résultant du produit cartésien entre U et B, c'est à dire {"b","ab"}

Concernant la structure, nous avons implémenté strictement celle présentée dans le sujet en respectant la hiérarchie de classe etc.

Les réponses au questions 2.2.1 - 2.2.6 sont disponibles en annexes à la fin du rapport

3 Algorithmes

Cf. tableaux valuations Q. 8
Avec la valuagtion de fait, nous avons implémenté une fonction init_grammar qui :

- Donne la grammaire complète à chaque règle.
- Vérifie que chaque règle est valide (pas de récursion infini, ou de règle inconnue).
- Calcule la valuation de chaque règle jusqu'à atteindre un point fix (Les constantes sont initialisées avec une valuation fix (0 ou 1), mais les Produits/Unions sont initialisés avec l'infini. Si une valuation est toujours infini c'est qu'il y a une erreur dans la grammaire).

4 Efficacité et usabilité

- Count(n):
  Cette méthode renvoie le nombre d'éléments de la grammaire que l'on peut construire de taille N.

  Exemple :
  Sur la grammaire des arbres, N indiques le nombre de feuille des arbres, Count(4) renvoie donc 5. Sur les autres, N indique le nombre de caractères des mots.

- List(n):
  Cette méthode renvoie une liste contenant tous les éléments de taille N de la grammaire (la taille de la liste doit être le résultat du Count(N).

  Exemple :
  List(4) sur les arbres (Count(4) = 5):
  Node(Leaf, Node(Leaf, Node(Leaf, Leaf)))
  Node(Leaf, Node(Node(Leaf, Leaf), Leaf))
  Node(Node(Leaf, Leaf), Node(Leaf, Leaf))
  Node(Node(Leaf, Node(Leaf, Leaf)), Leaf)
  Node(Node(Node(Leaf, Leaf), Leaf), Leaf)

- Unrank(n, i):
  Cette méthode renvoie l'élément de rang i parmis les éléments de taille N. Le résultat doit être le même que si on récupère l'élément i dans la liste des éléments de même taille.

  La méthode est trivial pour singleton et epsilon (on retourne l'objet si certaines conditions sont respectées).

  Pour union, on cherche l'objet dans le membre de gauche si le rang est inférieur au nombre d'objets à gauche, sinon on cherche l'objet à droite (on modifie le rang cherché en fonction du nombre d'objets à gauche).

  Pour le produit c'est plus compliqué. On compte le nombre d'éléments en faisant varier les tailles des membres de gauches et droites pour trouver un offset indiquant la taille du membre de gauche et un nouveau rang, puis on calcul avec ça des valeurs pour déterminer le rang du membre de gauche, et celui du membre de droite (on recommence récursivement).
  Exemple:
  Unrank(3) sur la grammaire des arbres retourne donc :
  Node(Node(Leaf, Node(Leaf, Leaf)), Leaf)
  (Cf. résultat de l'exemple sur List(4))

- Rank(E):
  Cette méthode renvoie le rang de l'élément donné dans la liste des éléments de même taille. On utilise des fonctions qui sont données avec la grammaire pour déterminer d'où l'objet vient. Si ces fonctions ne sont pas fournies, le Rank ne peut pas fonctionner (par exemple pour AutantAB, nous n'avons pas réussi à désambiguiser la grammaire, et certaines fonctions nécéssaires au rank ne sont pas implémentables).

  Pour l'union, on a une fonction qui nous dit si l'objet appartient à l'ensemble de gauche ou de droite.

  Pour le produit, on a une fonction qui sépare l'objet en 2 objets distincts, et à partir de ces 2 objets on calcul leur rang, on détermine le nombre d'objets ce trouvant avant dans l'ordre lexicographique et on obtient donc le rang de l'objet.
  Exemple:
  Rank(Node(Node(Leaf, Node(Leaf, Leaf)), Leaf)) sur la grammaire des arbres renvoie 3 car 3 arbres ce trouvent avant lexicographiquement.

- Random(n):
  Cette méthode renvoie un élément de taille n aléatoire. Pour ce faire on appel la fonction unrank sur les objets de taille n avec un rank tiré aléatoirement entre 0 et le résultat de Count(n).

5 Optimisation et Mémoisation

- Caching:
  Nous avons utilisé une librairie (functool) pour ajouter à certaines fonctions une indication indiquant au programme qu'il devait mémoriser les x derniers résultats de cette fonction (x = 32 dans notre implémentation).
  Exemple :
  @lru_cache(maxsize=32)
  Cette indication est l'équivalent de @cached_function en sage.

- Grammaire condensée:
  La grammaire condensé permet d'écrire une grammaire de manière imbriquée sans utiliser une règle par élément. Une fonction va permettre de générer une grammaire à partir de sa forme condensée en générant dynamiquement des règles et en remplaçant la règle par son nom dans la grammaire

  Exemple:
  treeGramCond = {"Tree" : Union(Prod(NonTerm("Tree"), NonTerm("Tree")), Singleton(Leaf))}
  Donnera :
  treeGramCond = {"Tree" : UnionRule("Prod-x","Sing-y"),
  "Prod-x" : ProductRule("Tree","Tree")
  "Sing-y" : SingletonRule(Leaf)}
  Avec x et y des nombre représentant la taille de la grammaire au moment de la création de la règle, pour que tous les noms soient différents

- Bound :
  Bound est simplement un raccourci permettant de générer toutes les valeurs de Count et List pour des objets de taille entre les bornes passées en paramètres.

- Sequence :
  Une séquence est un raccourci syntaxique pour désigner une règle de type A. Nous avons décidé d'implémenter la séquence en tant que grammaire condensée.
  Exemple:
  gramSeq = {"SeqA" : Sequence(Singleton("a"), Epsilon(""), "".join, unpack2, isEmpty, size)}
  Cette grammaire génère les mots contenant seulement "a" (ou le mot vide)
  Devient :
  gramSeq = {'SeqA' : UnionRule("Eps-1", "Prod-3")
  'Prod-3': ProductRule("SeqA", "Sing-2")
  'Sign-2' : SingletonRule("a")
  'Eps-1' : EpsilonRule("")}

- HTML :
  Nous avons commencé à créer une grammaire pour représenter un document HTLM complexe, mais ce n'est pas encore fonctionnel. Nous allons essayer de terminer cette grammaire pour la présentation.

6 Conclusion

Il nous a fallu du temps pour comprendre comment mettre en place ce projet (comprendre l'architecture et ce qui était demandé) car il semblait très abstrait. Nous avons réussi à implémenter la totalité du projet en apprenant aussi au passage (dans mon cas) à développer des choses plus avancées en python. Je ne savais pas qu'il était possible, en commençant ce projet, de généraliser autant un code pour permettre de générer n'importe quelle grammaire... et ça a été une bonne expérience !

Tree	Tree	Node	Leaf
n	—	—	—
0	0	0	0
1	1	0	1
2	1	1	0
3	2	2	0
4	5	5	0
5	14	14	0
6	42	42	0
7	132	132	0
8	426	426	0
9	1430	1430	0
10	4862	4862	0

Table 1: Q. 2.1.1 - Grammaire des arbres

Fibonacci	Fib	Cas1	Cas2	Vide	CasAu	CasBau	AtomA	AtomB
n	—	—	—	—	—	—	—	—
0	1	0	0	1	0	0	0	0
1	1	1	1	0	0	0	1	1
2	1	1	0	0	1	0	0	0
3	2	2	1	0	1	1	0	0
4	3	3	1	0	2	1	0	0
5	5	5	2	0	3	2	0	0
6	8	8	3	0	5	3	0	0
7	13	13	5	0	8	5	0	0
8	21	21	8	0	13	8	0	0
9	34	34	13	0	21	13	0	0
10	55	55	21	0	55	21	0	0

Table 2: Q. 2.1.1 - Grammaire des mots de fibonnacci

AB	UnionRule("Vide","StartAB")
StartAB	UnionRule("CasA","CasB")
CasA	ProductRule("AtomA","ABword", "".join)
CasB	ProductRule("AtomB","ABword", "".join)
Vide	EpsilonRule("")
AtomA	SingletonRule("A")
AtomB	SingletonRule("B")

Table 3: Q. 2.1.2 - Grammaire des mots AB

DyckWord	UnionRule("Vide","CasStart")
CasStart	ProductRule("AtomL","CasMid", "".join)
CasMid	ProductRule("DyckWord","CasEnd", "".join)
CasEnd	SingletonRule(")")
Vide	EpsilonRule("")
AtomL	SingletonRule("(")
AtomR	SingletonRule(")")

Table 4: Q. 2.1.3 - Grammaire des mots de Dyck

AB2Max	UnionRule("Vide","Start")
Start	UnionRule("CasA","CasB")
CasA	ProductRule("AtomA","StartedA", "".join)
CasB	ProductRule("AtomB","StartedB", "".join)
StartedA	UnionRule("Vide","NextA")
StartedB	UnionRule("Vide","NextB")
NextA	UnionRule("CasB","EndA")
NextB	UnionRule("CasA","EndB")
FollowedByA	UnionRule("CasA","Vide")
FollowedByB	UnionRule("CasB","Vide")
EndA	ProductRule("AtomA","FollowedByB", "".join)
EndB	ProductRule("AtomB","FollowedByA", "".join)
Vide	EpsilonRule("")
AtomA	SingletonRule("A")
AtomB	SingletonRule("B")

Table 5: Q. 2.1.4 - Grammaire des mots de 2 lettres identiques consécutives maximum

PalAB	UnionRule("Vide","StartAB")
StartAB	UnionRule("Single","Sym")
Single	UnionRule("AtomA","AtomB")
Sym	UnionRule("SymA","SymB")
SymA	ProductRule("AtomA","SymA2", "".join)
SymB	ProductRule("AtomB","SymB2", "".join)
SymA2	ProductRule("PalAB","AtomA", "".join)
SymB2	ProductRule("PalAB","AtomB", "".join)
Vide	EpsilonRule("")
AtomA	SingletonRule("A")
AtomB	SingletonRule("B")

Table 6: Q. 2.1.5 - Grammaire des palindromes sur AB

PalABC	UnionRule("Vide","StartABC")
StartABC	UnionRule("Single1","Sym1")
Single1	UnionRule("AtomA","Single2")
Single2	UnionRule("AtomB","AtomC")
Sym1	UnionRule("SymA1","Sym2")
Sym2	UnionRule("SymB1","SymC1")
SymA1	ProductRule("AtomA","SymA2", "".join)
SymB1	ProductRule("AtomB","SymB2", "".join)
SymC1	ProductRule("AtomC","SymC2", "".join)
SymA2	ProductRule("PalABC","AtomA", "".join)
SymB2	ProductRule("PalABC","AtomB", "".join)
SymC2	ProductRule("PalABC","AtomC", "".join)
Vide	EpsilonRule("")
AtomA	SingletonRule("A")
AtomB	SingletonRule("B")
AtomC	SingletonRule("C")

Table 7: Q. 2.1.5 - Grammaire des palindromes sur ABC

AutantAB	UnionRule("Vide","StartAB")
StartAB	UnionRule("StartWithA","StartWithB")
StartWithA	ProductRule("AtomA","B1", "".join)
StartWithB	ProductRule("AtomB","A1", "".join)
A1	UnionRule("A2","BDoubleA")
A2	ProductRule("AtomA","AutantAB", "".join)
B1	UnionRule("B2","ADoubleB")
B2	ProductRule("AtomB","AutantAB", "".join)
BDoubleA	ProductRule("AtomB","DoubleA", "".join)
ADoubleB	ProductRule ("AtomA","DoubleB", "".join)
DoubleA	ProductRule ("A1","A1", "".join)
DoubleB	ProductRule ("B1","B1", "".join)
Vide	EpsilonRule("")
AtomA	SingletonRule("A")
AtomB	SingletonRule("B")

Table 8: Q. 2.1.6 - Grammaire avec autant de A que de B

n	Fib	Cas1	Cas2	CasAu	CasBau	Vide	AtomA	AtomB
règle	Union	Union	Union	Product	Product	Epsilon	Singleton	Singleton
0	∞	∞	∞	∞	∞	∞	∞	∞
1	∞	∞	∞	∞	∞	0	1	1
2	0	∞	1	∞	∞	0	1	1
3	0	1	1	1	∞	0	1	1
4	0	1	1	1	2	0	1	1
5	0	1	1	1	2	0	1	1
final	0	1	1	1	2	0	1	1

Table 9: Q. 8 - Valuation de Fibonacci

n	ABWord	StartAB	CasA	CasB	Vide	AtomA	AtomB
règle	Union	Union	Product	Product	Epsilon	Singleton	Singleton
0	∞	∞	∞	∞	∞	∞	∞
1	∞	∞	∞	∞	0	1	1
2	0	∞	∞	∞	0	1	1
3	0	∞	1	1	0	1	1
4	0	1	1	1	0	1	1
5	0	1	1	1	0	1	1
final	0	1	1	1	0	1	1

Table 10: Q. 8 - Valuation de la grammaire AB

n	DyckWord	CasStart	CasMid	CasEnd	Vide	AtomL	AtomR
règle	Union	Product	Product	Product	Epsilon	Singleton	Singleton
0	∞	∞	∞	∞	∞	∞	∞
1	∞	∞	∞	∞	0	1	1
2	0	∞	∞	∞	0	1	1
3	0	∞	∞	1	0	1	1
4	0	∞	1	1	0	1	1
5	0	2	1	1	0	1	1
6	0	2	1	1	0	1	1
final	0	2	1	1	0	1	1

Table 11: Q. 8 - Valuation de la grammaire des mots de Dyck

n	AB2Max	Start	CasA	CasB	StartedA	StartedB	NextA	NextB	FollowedByA	FollowedByB	EndA	EndB	Vide	AtomA	AtomB
règle	Union	Union	Product	Product	Union	Union	Union	Union	Union	Union	Product	Product	Epsilon	Singleton	Singleton
0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	0	1	1
2	0	∞	∞	∞	0	0	∞	∞	0	0	∞	∞	0	1	1
3	0	∞	1	1	0	0	∞	∞	0	0	1	1	0	1	1
4	0	1	1	1	0	0	1	1	0	0	1	1	0	1	1
5	0	1	1	1	0	0	1	1	0	0	1	1	0	1	1
final	0	1	1	1	0	0	1	1	0	0	1	1	0	1	1

Table 12: Q. 8 - Valuation de la grammaire des mots de AB2Max

n	PalAB	StartAB	Single	Sym	SymA	SymB	SymA2	SymB2	Vide	AtomA	AtomB
règle	Union	Union	Union	Union	Product	Product	Product	Product	Epsilon	Singleton	Singleton
0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	∞	∞	∞	∞	∞	∞	∞	∞	0	1	1
2	0	∞	1	∞	∞	∞	∞	∞	0	1	1
3	0	1	1	∞	∞	∞	1	1	0	1	1
4	0	1	1	∞	2	2	1	1	0	1	1
5	0	1	1	2	2	2	1	1	0	1	1
6	0	1	1	2	2	2	1	1	0	1	1
final	0	1	1	2	2	2	1	1	0	1	1

Table 13: Q. 8 - Valuation de la grammaire des Palindromes AB

n	PalABC	StartABC	Single1	Single2	Sym1	Sym2	SymA1	SymB1	SymC1	SymA2	SymB2	SymC2	Vide	AtomA	AtomB	AtomC
règle	Union	Union	Union	Union	Union	Union	Product	Product	Product	Product	Product	Product	Epsilon	Singleton	Singleton	Singleton
0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	0	1	1	1
2	0	∞	1	1	∞	∞	∞	∞	∞	∞	∞	∞	0	1	1	1
3	0	1	1	1	∞	∞	∞	∞	∞	1	1	1	0	1	1	1
4	0	1	1	1	∞	∞	2	2	2	1	1	1	0	1	1	1
5	0	1	1	1	2	2	2	2	2	1	1	1	0	1	1	1
6	0	1	1	1	2	2	2	2	2	1	1	1	0	1	1	1
final	0	1	1	1	2	2	2	2	2	1	1	1	0	1	1	1

Table 14: Q. 8 - Valuation de la grammaire des Palindromes ABC

n	AutantAB	StartAB	StartWithA	StartWithB	A1	A2	B1	B2	BDoubleA	ADoubleB	DoubleA	DoubleB	Vide	AtomA	AtomB
règle	Union	Union	Product	Product	Union	Product	Union	Product	Product	Product	Product	Product	Epsilon	Singleton	Singleton
0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	0	1	1
2	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	0	1	1
3	0	∞	∞	∞	∞	1	∞	1	∞	∞	∞	∞	0	1	1
4	0	∞	∞	∞	1	1	1	1	∞	∞	∞	∞	0	1	1
5	0	∞	2	2	1	1	1	1	∞	∞	2	2	0	1	1
6	0	2	2	2	1	1	1	1	3	3	2	2	0	1	1
7	0	2	2	2	1	1	1	1	3	3	2	2	0	1	1
final	0	2	2	2	1	1	1	1	3	3	2	2	0	1	1

Table 15: Valuation de la grammaire Autant de A que de B