Projet de Combinatoire

Rapport: Stéphane Romany
Binome: Thibault Martin

Abstract

Ce projet à pour but de générer une grammaire de manière automatique, en fournissant des règles de constructions au programme, au moyen d'un dictionnaire python. Ce projet est édité en python (à partir de la version Python 3.5).

1 Définitions formelles et Algorithmes

En Annexe: Tableaux (Dans l'ordre d'apparition des questions).

La fonction Init_grammar effectue trois étapes afin de préparer les grammaire pour leur futur utilisation.
Elle fourni à chaque règles la "connaissance" de grammaire toute entière via un dictionnaire interne à chaque règles via la méthode _set_grammar de chaque règle. Ensuite elle vérifie si la grammaires est valide à l'aide des méthodes check des grammaires dont le but est de vérifier que tous les Non-terminaux de la grammaires apparaissent dans le dictionnaire et qu'il n'y à pas de règles récursive (exemple : "Tree" : UnionRule("Tree" , "Tree")). Enfin on calcule la valuation pour toutes les règles jusqu'à stabilisation i.e que les valuations de toutes les règles à l'étape n soient égales au valuation à l'étape $n-1$ et différentes de l'infini.

2 Efficacité et usabilité

2.1 Sécifications

Toutes les méthodes et optimisations suivantes supposeront que la grammaire aura été préalablement initialisé via la fonction init_grammar.

- Count(n):
  Cette méthode doit renvoyer le nombre d'éléments que l'on peut construire avec N éléments de base de la grammaire concernée.

  Exemple:
  Grammaire des Arbres binaires: L'élément de base est la feuille. Donc la méthode count sur cette grammaire doit renvoyer le nombre d'arbres que l'on peut construire avec n feuille.

- List(n):
  Cette méthode doit renvoyer une liste contenant tous les éléments constructibles à partir de N éléments de base de la grammaire.

  Exemple:
  Grammaire des mots sur l'alphabet $\Sigma = A, B$: L'élément de base est la lettre. Donc la méthode list sur cette grammaire doit renvoyer le nombre la liste de tous les mots que l'on peut construire avec n lettres.

- Unrank(n, i):
  Cette méthode doit renvoyer l'élément de rang i se trouvant dans la liste de tous les éléments constructibles à partir de N éléments de base de la grammaire. Cette élément est généré sur le tas. Il ne s'agit pas de générer la list puis de choisir un indice aléatoire dans cette liste..

  Exemple:

Grammaire des mots de Dyck $\Sigma = \{(,)\}$:
L'élément de base est la parenthèse. Donc la méthode unrank sur sur cette grammaire doit renvoyer le mot de Dyck à la position i de l'ensemble des mots que l'on peut construire avec N parenthèses.

- Rank(E):
  Cette méthode doit renvoyer le rang de l'élément E se trouvant dans l'ensemble de tous les éléments constructibles à partir de N éléments de base de la grammaire. Autrement dit cette méthode renvoi l'indice de cet élément dans la liste — sans la parcourir — des éléments de taille N dans la grammaire concernée.

  Exemple:
  Grammaire des mots de Dyck $\Sigma = \{(,)\}$:
  Donc la méthode Rank sur cette grammaire doit renvoyer le rang du mot de Dyck passé en paramètre. Si le mot est de taille 10 alors le Rang de ce mot doit etre compris entre 0 et le résultat de la méthode Count(10).

- Random(n):
  Cette méthode doit renvoyer un élément arbitrairement choisi de manière aléatoire dans la liste de tous les éléments constructibles à partir de N éléments de base de la grammaire. Cette élément est généré sur le tas. Il ne s'agit pas de générer la list puis de choisir un indice aléatoire dans cette liste.

  Exemple:
  Grammaire des mots des Palindrome sur $\Sigma = \{(,)\}$:
  Donc la méthode Rank sur cette grammaire doit renvoyer le rang du mot de Dyck passé en paramètre. Si le mot est de taille 10 alors le Rang de ce mot doit etre compris entre 0 et le résultat de la méthode Count(10).

2.2 Optimisation et Mémoisation

Cette rubrique concerne les diverses optimisations.

- Caching:
  Nous avons utilisé un module du package functool de python3 pour bénéficier des décorateurs de mémoisation.
  Néamnoins il est possible de définir soit meme ses fonctions de mémoisation en utilisant des décorateurs. Voici la démarche:

- Grammaire condensée:
  L'implémentation donne la possibilité de compresser l'écriture d'une grammaire dans un dictionnaire python. Il est également fourni une fonction de convertion dont le but est de "décompresser" cette grammaire pour la rendre sous la forme des grammaires de base. Chaque nouvelle classe dispose d'une méthode 'conv' qui génère la règle descendant de la classe AbstracrRule.

  Exemple:
  Gzip = { "Tree" : Union (Singleton Leaf,
  Prod(NonTerm "Tree", NonTerm "Tree", "".join) }

  Gunzip = { "Tree" : UnionRule("Node", "Leaf"),
  "Node" : ProductRule("Tree", "Tree", lambda (a, b) : Node(a, b)),
  "Leaf" : SingletonRule(Leaf) }

- Bound(Gram, low, up):
  Cette implémentation permet de créer un objet capable de générer tous les résultats des méthode Count et List d'une grammaire "Gram" pour N éléments entre low à up, bornes inclus. Ces résultats sont stockés dans les attributs _list et _count de la classe Bound.

Exemple:
B = Bound(treeGramm["Tree"], 3 , 6)
B._list = [treeGramm["Tree"].list(i) for i in range(3, 7)]
B._count = [treeGramm["Tree"].count(i) for i in range(3, 7)]

- Sequence(Rule, Epsilon, cons):
  Cette implémentation permet un raccourci syntaxique pour composé une règle "sequence" UnionRule avec une règle EpsilonRule et un produit entre un non-terminal et cette meme règle "sequence".
  La Sequence prend en paramètre des règles condensées (Type Union, Prod, NonTerm, Singleton, Epsilon), et sa conversion via la fonction de conversion retourne la grammaire sous une forme décondensée.

  Exemple:
  "Sequence" : Sequence(Singleton("a"), Epsilon(""))
  Devient:
  "Sequence" : UnionRule ("Eps-1", "Prod-1")
  "Prod-1" : ProductRule ("Sequence" "Singl-1")
  "Singl-1" : SingletonRule ("a")
  "Eps-1" : EpsilonRule ("")

Pour aller plus loin

Dans l'état actuel de l'implémentation, la grammaire HTML représentant l'ensemble des pages complexe que l'on peut générer n'est pas totalement fonctionnel. Mais il y a un bon début.

3 Conclusion

Ce projet nous a permis de découvrir de nombreuses fonctionnalité de python3, ainsi que certaines différences entre les technologies python 2 et 3.
Nous avons également pu implémenté la plupart des fonctionnalités du projet, bien que nous ayons eu du mal au début à imaginer comment généraliser la génération de grammaire.
L'idée de pouvoir générer des pages XML ou HTML simplement en fournissant la bonne grammaire est plutôt intéressante, mettant a disposition un catalogue de pages toutes prêtes à l'emploi !

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: Tree Grammar 2.1

Fibonacci	Fib	Cas1	Cas2	Vide	CasAu	CasBau	AtomA	AtomB
n	—	—	—	—	—	—	—	—
0	1	0	0	1	0	0	0	0
1	2	2	1	0	1	0	1	1
2	3	3	1	0	2	1	0	0
3	5	5	2	0	3	2	0	0
4	8	8	3	0	5	3	0	0
5	13	13	5	0	8	5	0	0
6	21	21	8	0	13	8	0	0
7	34	34	13	0	21	13	0	0
8	55	55	21	0	34	21	0	0
9	89	89	34	0	55	34	0	0
10	144	144	55	0	89	55	0	0

Table 2: Fib Grammar 2.1

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

Table 3: Grammaire AB 2.2

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

Table 4: Grammaire de Dyck 2.3

AB2Max	UnionRule("Vide", "Start")
Start	UnionRule("CasA", "CasB")
CasA	ProductRule("AtomA", "StartedA")
CasB	ProductRule("AtomB", "StartedB")
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")
EndB	ProductRule("AtomB", "FollowedByA")
Vide	EpsilonRule("")
AtomA	SingletonRule("A")
AtomB	SingletonRule("B")

Table 5: Grammaire des mots de deux lettres identique consécutives maximum 2.4

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

Table 6: Grammaire des palindromes sur AB 2.5

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")
SymB1	ProductRule("AtomB", "SymB2")
SymC1	ProductRule("AtomC", "SymC2")
SymA2	ProductRule("PalABC", "AtomA")
SymB2	ProductRule("PalABC", "AtomB")
SymC2	ProductRule("PalABC", "AtomC")
Vide	EpsilonRule("")
AtomA	SingletonRule("A")
AtomB	SingletonRule("B")
AtomC	SingletonRule("C")

Table 7: Grammaire des palindromes sur ABC 2.5

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

Table 8: Grammaire Autant de A que de B 2.6

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: 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: 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: 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: 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: 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: 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