Génération d'objets combinatoires décrits par une grammaire

Rapport de projet par

Léo Andrès

19 novembre 2019

Master 2 Fondements de l'Informatique et Ingénierie du Logiciel

université PARIS-SACLAY

Dans le cadre du cours de

Florent Hivert

Résumé

Ce document est un rapport de mon projet *Génération d'objets combinatoires décrits par une grammaire*, effectué lors de ma formation en Master 2 Fondements de l'Informatique et Ingénierie du Logiciel à l'Université Paris-Saclay. Ce projet a eu lieu dans le cadre du cours de Florent Hivert.

Table des matières

1 Question 1

1.1 Cas des mots de Fibonnaci

$$
\begin{aligned}
\mathsf{Fib} &: [1, 2, 3, 5, 8, 13, 21, 34, 55, 89] \\
\mathsf{Cas1} &: [0, 2, 3, 5, 8, 13, 21, 34, 55, 89] \\
\mathsf{Cas2} &: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34] \\
\mathsf{Vide} &: [1, 0, 0, 0, 0, 0, 0, 0, 0, 0] \\
\mathsf{CasAu} &: [0, 1, 2, 3, 5, 8, 13, 21, 34, 55] \\
\mathsf{CasBAu} &: [0, 0, 1, 2, 3, 5, 8, 13, 21, 34] \\
\mathsf{AtomA} &: [0, 1, 0, 0, 0, 0, 0, 0, 0, 0] \\
\mathsf{AtomB} &: [0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
\end{aligned}
$$

1.2 Cas des arbres binaires complets

$$
\begin{aligned}
\mathsf{Tree} &: [0, 1, 1, 2, 5, 14, 42, 132, 429, 1430] \\
\mathsf{Node} &: [0, 0, 1, 2, 5, 14, 42, 132, 429, 1430] \\
\mathsf{Leaf} &: [0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
\end{aligned}
$$

2 Question 2

Cf. `ab.py`.

3 Question 3

Cf. `dick_word.py`.

4 Question 4

Cf. `ab_not_three.py`.

5 Question 5

5.1 Palindromes avec deux lettres

Cf. `pal.py`.

5.2 Palindromes avec trois lettres

Cf. `pal_three.py`

6 Question 6

Cf. `even.py`.

7 Question 7

Cf. la méthode `is_correct` dans `grammar.py`. On s'assure aussi qu'un identifiant n'est lié qu'une fois au plus.

8 Question 8

8.1 Mots de Fibonnaci

$$
\begin{aligned}
\mathrm{Fib} &: 0 \\
\mathrm{Cas1} &: 1 \\
\mathrm{Cas2} &: 1 \\
\mathrm{Vide} &: 0 \\
\mathrm{CasAu} &: 1 \\
\mathrm{CasBAu} &: 2 \\
\mathrm{AtomA} &: 1 \\
\mathrm{AtomB} &: 1
\end{aligned}
$$

8.2 Arbres binaires complets

Tree : 1
Node : 2
Leaf : 1

8.3 AB NOT THREE

NO3AB : 0
Q4_B : 1
Q3_A : 1
Q0_A : 1
Q0_B : 1
Q1_A : 1
Q1_B : 1
Q2_A : 1
Q2_B : 1
Q0_LETTER : 1
Q1_LETTER : 1
Q2_LETTER : 1
Q1 : 0
Q2 : 0
Q3 : 0
Q4 : 0
A : 1
B : 1
EMPTY : 0

8.4 AB

AB : 0
LETTER : 1
EMPTY : 0
A : 1
B : 1
CAS1 : 1

8.5 Dyck words

DYCK : 0
EMPTY : 0
LPAR : 1
RPAR : 1
START : 2
END : 1

8.6 even

EVEN : 0
ANY_CASE : 2
CASE1 : 2
CASE2 : 2
APLUS : 1
BPLUS : 1
A : 1
B : 1
EMPTY : 0

8.7 pal

PAL : 0
SINGLE_LETTER : 1
LETTER : 1
ANY_CASE : 2
A : 1
B : 1
EMPTY : 0
CAS1 : 1
CAS2 : 1
CAS3 : 2
CAS4 : 2

8.8 pal three

PAL : 0
SINGLE_LETTER_AUX : 1
SINGLE_LETTER : 1
LETTER : 1
ANY_CASE_AUX : 2
ANY_CASE : 2
A : 1
B : 1
C : 1
EMPTY : 0
CAS1 : 1
CAS2 : 1
CAS3 : 1
CAS4 : 2
CAS5 : 2
CAS6 : 2

9 Question 9

Cela est vérifié par un test, cf. la méthode `test` dans `grammar.py`.

10 Question 10

Cf. la méthode `init_grammar` dans `grammar.py`.

11 Question 11

L'ensemble des grammaires ont été testées et l'implémentation semble correcte. L'exécution du fichier `present.py` peut aider à s'en convaincre.

12 Question 12

Cf. les méthodes `count` dans les fichiers idoines.

13 Question 13

Soit $\mathbb{E}$ un ensemble à n éléments.

On veut que les fonctions `list`, `count`, `rank`, `unrank` et `random` respectent les propriétés suivantes :

$$\texttt{Card}(\texttt{list}(\mathbb{E})) = n \tag{1}$$

$$\forall e,\ e \in \texttt{list}(\mathbb{E}) \Leftrightarrow e \in \mathbb{E} \tag{2}$$

$$\forall e \in \texttt{list}(E),\ 0 \leq \texttt{rank}(e) \leq \texttt{count}(n) \tag{3}$$

$$\forall i \in \mathbb{N},\ 0 \leq i < n \Rightarrow\ \texttt{unrank}(i) \in \texttt{list}(E) \tag{4}$$

$$\forall e \in \mathbb{E},\ \texttt{unrank}(\texttt{rank}(e)) = e \tag{5}$$

$$\forall i \in \mathbb{N},\ 0 \leq i < n \Rightarrow \texttt{rank}(\texttt{unrank}(i)) = i \tag{6}$$

$$\forall k \in \mathbb{N},\ \texttt{random}() = e \Rightarrow e \in \mathbb{E} \tag{7}$$

$$\forall k \in \mathbb{N},\ 0 \leq k < n \Rightarrow \texttt{unrank}(i) \in \mathbb{E} \tag{8}$$

$$\forall e \in \mathbb{E}, \exists k \in \mathbb{N}, \text{l'appel numéro } k \text{ à la fonction } \texttt{random} \text{ produit le résultat } e \tag{9}$$

$$\forall n \in \mathbb{N}, \texttt{count}(n) = \texttt{len}(\texttt{list}(n)) \tag{10}$$

Il est à noter que l'on a fait abstraction de la taille des objets, normalement présente dans les différentes fonctions, afin de simplifier.

14 Question 14

Cf. la méthode `test` dans `grammar.py`.

15 Question 15

Cf. le module `test.py`, qui lance les tests sur toutes les grammaires. Les tests sont aussi lancés à chaque commit dans le dépôt git et disponibles sur builds.sr.ht/~zapashcanon/project-grammar-gen.

16 Question 16

TODO

17 Question 17

On a créé un module `memo`, cf. `memo.py`. Ce module contient une unique fonction `memo`. Elle peut être vue comme une fonction d'ordre supérieure, qui à une fonction associe sa version mémoïsée. On peut ensuite mémoïser n'importe quelle fonction au moyen d'un décorateur.

18 Question 18

TODO

19 Question 19

TODO

20 Question 20

Cf. la méthode `add_sequence` dans le module `grammar.py`. Un exemple est donné dans la grammaire définie dans le module `a_list.py`.

21 Question 21

TODO