Outils de combinatoire analytique en Sage

Matthieu Dien, Marguerite Zamansky
encadrés par Antoine Genitrini et Frederic Peschanski

8 mai 2013

1 Introduction

Là, je raconte le projet, j'explique en gros ce qu'on fait.

1.1 Quelques bases de combinatoire analytique

La combinatoire est une branche des mathématiques qui s'intéresse à l'étude des structures finies ou dénombrables : des objets qui peuvent être créés par un nombre fini de règles. On peut s'intéresser aux propriétés de ces objets combinatoires, mais souvent on voudra les compter. Pour ça la méthode classique utilise des raisonnements par récurrence, souvent compliqués, parfois inefficaces. La combinatoire analytique a pour but de décrire de manière quantitatives ces structures combinatoires en utilisant des outils analytiques. La pierre d'angle de théorie est la *fonction génératrice*. Nous commencerons par les définitions de quelques notions de base de la combinatoire analytique.

Définition *Une <u>classe combinatoire</u> est un ensemble fini ou dénombrable sur lequel est défini une fonction taille qui vérifie les conditions suivantes :*

- *la taille d'un élément est un entier positif,*
- *il y a un nombre fini d'élément de chaque taille.*

Définition *La <u>suite de comptage</u> d'une classe combinatoire $\mathcal{A}$ est la suite $(A_n)_{n\geqslant 0}$ où A_n est le nombre d'objet de taille n dans $\mathcal{A}$.*

Définition *La <u>série génératrice ordinaire</u> d'une suite (A_n) est la série entière*

$$A(z) = \sum_{n=0}^{\infty} A_n z^n.$$

La série génératrice ordinaire d'une classe combinatoire est la série génératrice ordinaire de sa suite de comptage. De manière équivalente, le série génératrice d'une classe combinatoire $\mathcal{A}$ peut s'écrire sous la forme

$$A(z) = \sum_{a\in\mathcal{A}} z^{|a|}.$$

Nous attirons l'attention sur le fait qu'une série génératrice ordinaire est vue en tant que série formelle, et non comme une série entière.

Souvent, pour dénombrer les objets, on les décompose en éléments plus simples, mais du même type, pour obtenir une équation de récurrence vérifiée par les A_n. Cette équation peut-être plus ou moins simple à résoudre. L'approche proposée par la combinatoire analytique, repose sur la vision d'une classe combinatoire comme une construction à partir d'autres classes combinatoires. On définit les constructions admissibles, qui produisent une classe combinatoire.

Définition *Soient, $\mathcal{A}$, $\mathcal{B}^1$,...,$\mathcal{B}^m$ des classes combinatoires, et Φ une construction,*

$$\mathcal{A} = \Phi(\mathcal{B}^1, \ldots, \mathcal{B}^m).$$

La construction Φ est admissible si et seulement si la suite de comptage (A_n) de $\mathcal{A}$ ne dépend que des suites de comptages $(B_n^1), \ldots, (B_n^m)$ de $\mathcal{B}^1$,...,$\mathcal{B}^m$.

Pour chaque construction admissible Φ, il existe un opérateur Ψ, agissant sur les génératrices ordinaires : si

$$\mathcal{A} = \Phi(\mathcal{B}^1, \ldots, \mathcal{B}^m),$$

alors,

$$A(z) = \Psi(B^1(z), \ldots, B^m(z)).$$

L'existence de cet opérateur va nous permettre d'utiliser la décomposition en classes combinatoires élémentaires pour obtenir les séries génératrices ordinaires de classes combinatoires plus compliquées.

1.2 Constructions de base

Pour définir les constructions de base, on se donne deux classes combinatoires particulières, la classe neutre $\mathcal{E}$, constituée d'un unique objet de taille nulle et la classe atomique $\mathcal{Z}$, constituée d'un unique objet de taille 1. Leurs séries génératrices sont donc respectivement $E(z) = 1$ et $Z(z) = z$.

Produit cartésien

La construction produit cartésien appliqué à deux classes combinatoires $\mathcal{B}$ et C donne la classe combinatoire

$$\mathcal{A} = \{\alpha = (\beta, \gamma) \mid \beta \in \mathcal{B}, \gamma \in C\}$$

munie de la fonction taille définie par

$$|\alpha|_{\mathcal{A}} = |\beta|_{\mathcal{B}} + |\gamma|_C.$$

On écrit alors $\mathcal{A} = \mathcal{B} \times C$. En regardant toutes les possibilités de paires, on trouve que la suite de comptage de A est donnée par le produit de convolution des suites de comptage de B et C :

$$A_n = \sum_{k=0}^{n} B_k C_{n-k}.$$

On remarque bien sûr que le produit cartésien est une construction admissible, et on reconnaît dans cette égalité le produit de deux séries entières. Ainsi la série génératrice de $\mathcal{A}$ est

$$A(z) = B(z) \cdot C(z).$$

Somme combinatoire

La somme combinatoire est définie de manière à avoir les mêmes propriétés que la somme disjointe, mais sans avoir à imposer la disjonction des ensembles : on se donne deux objets neutres distincts, i.e. de taille nulle, $\square$ et $\diamond$ et on pose

$$\mathcal{B} + \mathcal{C} := (\{\square\} \times \mathcal{B}) \cup (\{\diamond\} \times \mathcal{C}),$$

comme $\square$ et $\diamond$ sont distincts, le membre de droite est une union disjointe, pour toute classes combinatoires $\mathcal{B}$ et $\mathcal{C}$. Il y a autant d'objets de taille n dans $\{\square\} \times \mathcal{B})$ et $(\{\diamond\} \times \mathcal{C})$ que dans $\mathcal{B}$ et $\mathcal{C}$, et par conséquent, si $\mathcal{A} = \mathcal{B} + \mathcal{C}$, on a

$$A_n = B_n + C_n,$$

et ce quel que soit n, la somme combinatoire est donc une construction admissible. La série génératrice de $\mathcal{A}$ est

$$A(z) = B(z) + C(z).$$

Séquence

Soit $\mathcal{B}$ une classe combinatoire, on définit $Seq(\mathcal{B})$ comme la somme combinatoire infinie

$$Seq(\mathcal{B}) := \{\epsilon\} + \mathcal{B} + (\mathcal{B} \times \mathcal{B}) + (\mathcal{B} \times \mathcal{B} \times \mathcal{B}) + \dots$$

Ce qui est équivalent à voir la séquence comme l'ensemble des suites d'objets de $\mathcal{B}$:

$$Seq(\mathcal{B}) = \{(\beta_1, \dots, \beta_l) \mid \beta_i \in \mathcal{B},\ l \geqslant 0\}.$$

En combinant les fonctions tailles associées au produit cartésien et à la somme combinatoire, on définit la taille dans $Seq(\mathcal{B})$:

$$|(\beta_1, \dots, \beta_l)|_{Seq(\mathcal{B})} = |\beta_1|_{\mathcal{B}} + \dots + |\beta_l|_{\mathcal{B}}$$

$Seq(\mathcal{B})$ est une classe combinatoire si et seulement si, $\mathcal{B}$ ne contient pas d'objet de taille nulle. En effet, s'il existe $\beta_0 \in \mathcal{B}$ de taille nulle, on peut créer une infinité de suites de β_0, toutes de taille nulle, $Seq(\mathcal{B})$ n'est donc pas une classe combinatoire.
Réciproquement, s'il n'existe pas d'objet de taille nulle dans $\mathcal{B}$, pour tout $n \geqslant 1$, il existe un nombre fini de suites d'entiers dont la somme est n, donc un nombre fini d'antécédents de n par la fonction taille de $Seq(\mathcal{B})$. Et la suite vide est le seul antécédent de 0, donc $Seq(\mathcal{B})$ est une classe combinatoire, et la séquence est une construction admissible.

Pour calculer la série génératrice de $\mathcal{A} = Seq(\mathcal{B})$, on utilise les propriétés vues avec le produit cartésien et la somme combinatoire :

$$A(z) = 1 + B(z) + B(z) \cdot B(z) + B(z) \cdot B(z) \cdot B(z) \dots$$

$$A(z) = 1 + B(z) + B(z)^2 + B(z)^3 + \dots$$

$$A(z) = \frac{1}{1 - B(z)}$$

Il existe d'autres constructions admissibles,

1.3 Spécification

Définition *Une spécification pour un n−uplet de classes combinatoires* $(\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_n)$*, est un système de n équations*

$$\begin{cases} \mathcal{A}_1 = & \Phi_1(\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_n) \\ \mathcal{A}_2 = & \Phi_2(\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_n) \\ \vdots & \qquad \vdots \\ \mathcal{A}_n = & \Phi_n(\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_n) \end{cases}$$

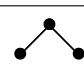
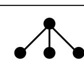
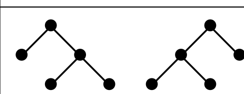
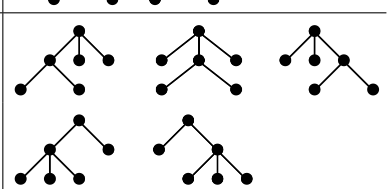
où Φ_i *est une construction admissible, produite avec les construction de produit cartésien, de la somme combinatoire, ou de la séquence sur les classes* $\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_n$*,* $\mathcal{E}$ *et* $\mathcal{Z}$*.*

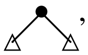
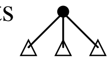
1.4 Fonctions génératrices multivariées

Jusque là, on ne considérait que des

exemple : les arbres binaires-ternaires

Un arbre binaire-ternaire est un arbre dont les noeuds internes ont deux ou trois fils :

taille 1	•
taille 2	$\varnothing$
taille 3	
taille 4	
taille 5	
taille 6	

Un arbre binaire ternaire est soit une feuille $\bullet$, soit un nœud binaire avec deux arbres pendants , soit un nœud ternaire avec trois arbres pendants  Si on prend en compte les paramètres suivants :

$$\begin{cases} z : \text{le nombre de feuilles} \\ u : \text{le nombre de nœuds binaires} \\ v : \text{le nombre de nœuds ternaires,} \end{cases}$$

on obtient la spécification suivante

$$\mathcal{BT} = \{z\} + \{u\} \cdot \mathcal{BT}^2 + \{v\} \cdot \mathcal{BT}^3,$$

ce qui en terme de série génératrice donne

$$BT(z, u, v) = z + uBT(z, u, v)^2 + vBT(z, u, v)^3$$

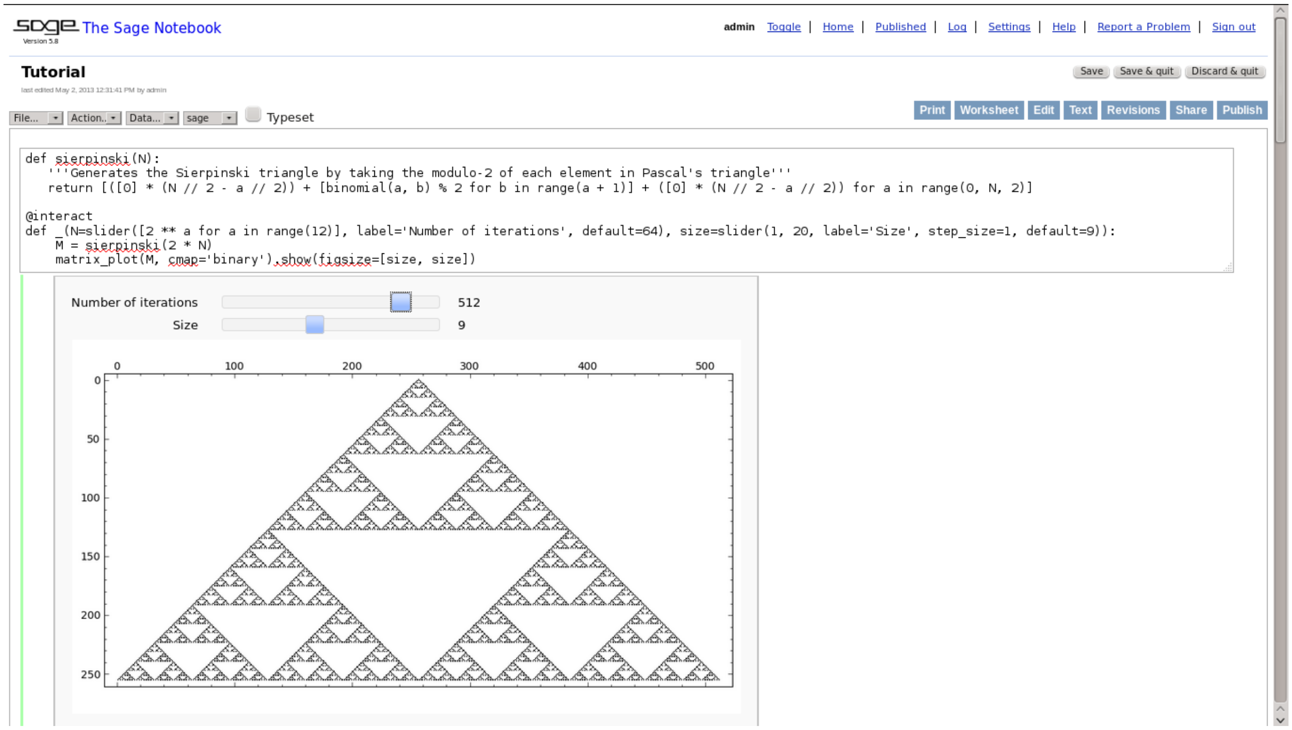
2 Sage

2.1 Introduction

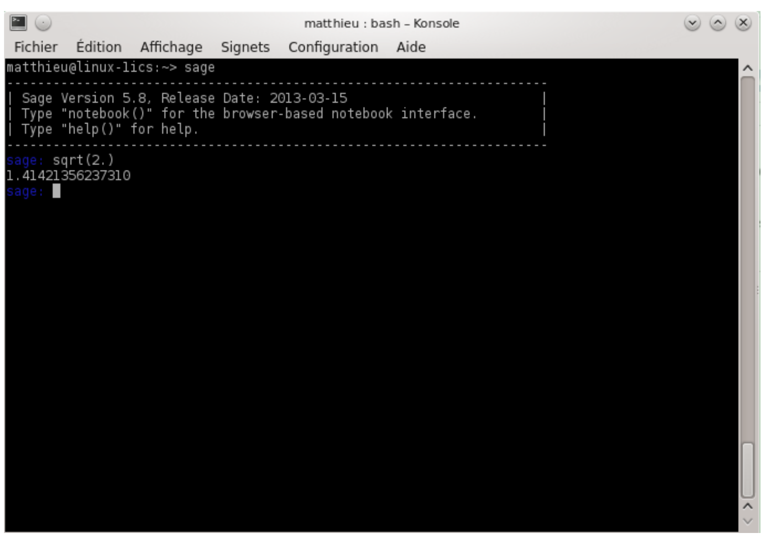
2.1.1 Présentation

Sage est un logiciel libre (sous licence GPL) de calcul formel, symbolique et numérique [S$^+$13]. Il est principalement écrit en C/C++ et Python/Cython/Pyrex. Le but du projet Sage est de fournir une alternative aux solutions de calculs propriétaires comme Mapple, Mathematica, Matlab ...
Historiquement Sage est un regroupement de différents projets libres autour d'une interface Python unifiée, pour faciliter les traitements entre ces *briques de base* (Singular, Maxima, GP/PARI, GAP). Mais de plus en plus le projet se tourne vers le développement de ses propres paquetages pour acquérir son indépendance, donc améliorée la cohérence de l'architecture, du le calendrier de release et faciliter ainsi le développement de nouvelles fonctionnalités et la maintenance d'ancienne.
Dans Sage, deux interfaces sont disponibles :

– L'interface bloc-note (ou notebook). Elle se présente sous la forme d'une interface embarquée dans un navigateur web, comme ceci :



– L'interface ligne de commande. Elle se présente comme un top-level python classique :



Le langage permettant de manipuler les outils Sage est un Python avec quelques sucres syntaxiques supplémentaires permettant une manipulation plus aisée des concepts mathématiques utilisés. Par exemple, pour déclarer l'anneau des polynômes de variables x, y et z, à coefficients dans

$\mathbb{Q}$:
```
R.<x,y,z> = PolynomialRing(QQ)
```

2.1.2 Architecture

Sage ayant pour objectif de couvrir l'ensemble des besoins en calcul pour tous les domaines des Mathématiques, la taille du code est donc assez conséquente (un peu plus de 11 milliards de lignes de code sans compter les paquets supplémentaires). Cela a été une première difficulté dans la réalisation de ce projet.
Le code de Sage s'organise autour d'une hiérarchie de dossiers représentant les différents domaines mathématiques : *combinat* pour la combinatoire, *algebras* pour l'algèbre, *games* pour la théorie des jeux, *probability*, *graphs*, etc.
De plus, la fonctionnalité d'aide de Sage permet d'afficher le fichier source d'un objet Sage et son emplacement :

```
PolynomialRing? #affichera la documentation
PolynomialRing?? #affichera le fichier source et son emplacement
```

Enfin, toute la documentation et les tests de Sage sont contenus directement dans le code source grâce au mécanisme des *docstring* Python : la documentation d'une fonction est écrite après son prototype dans le fichier source, par exemple

```python
def une_fonction():
    """
    Ceci est la docstring de une_fonction
    une_fonction renvoie 1

    Test test directement dans la fonction comme dans Sage::

        sage : une_fonction()
        1
    """

    return 1
```

Cela permet une plus grande facilité pour la compréhension du code et le maintien à jour du code comme de la documentation et des tests.

2.2 Implémentation

2.2.1 LazyPowerSeries

Sage contenait déjà une implémentation des séries génératrices monovariées (basée sur le travail fait sur Aldor [HR06]) sous la forme d'une classe Python. L'implémentation se présente donc sous la forme de deux classes héritant de `LazyPowerSeriesRing` et `LazyPowerSeries`. Les opérations supportées pour ces séries sont l'addition, la multiplication, la composition, la dérivation

et l'intégration (la primitive ne fait). Nous supportons l'ensemble de ces opérations (exceptée la primitivation) et nous y avons ajouté la séquence.
De plus, comme son nom l'indique, l'implémentation existante était paresseuse, ce qui est nécessaire car les objets manipulés sont des séries donc potentiellement non finies (et c'est généralement le cas). Nous avons évidemment repris ce style de programmation. Cette *paresse* est obtenu en utilisant le concept de générateur python, que nous présenterons avant de présenter les opérations implémentées.

2.2.2 Générateurs

Les générateurs Python [SPH01], sont un mécanisme permettant de créer des objets itérables. Ils sont déclarés sous forme de fonction classique contenant le mot clé `yield`. A chaque appel de la méthode `next` associé à ce générateur, le corps de la *fonction* est éxécuté jusqu'à rencontrer un `yield`, et l'argument donné à `yield` est retourné par `next` et l'éxécution du corps ce stoppe. Au prochain appel de `next`, l'éxécution du corps reprend jusqu'au prochain `yield`.
Par exemple, si nous voulions itérer sur l'ensemble des entiers naturels :

```
def integers_definition():
    i = 0
    while True :
        yield i
        i += 1

integers = integers_definition()

while True :
    n = integers.next()
    if n % 2 == 0:
        print("%d est pair"%n)
    else :
        print("%d est impair"%n)

# ou

for n in integers_definition():
    n = integers.next()
    if n % 2 == 0:
        print("%d est pair"%n)
    else :
        print("%d est impair"%n)
```

2.2.3 Représentation des séries

Lazy, Stream et tout ça, représentation...

2.3 Opérateurs

2.3.1 Addition

2.3.2 Multiplication

2.3.3 Séquence

Références

[HR06] Ralf Hemmecke and Martin Rubey. Aldor-Combinat : An Implementation of Combinatorial Species, 2006. Available at http://www.risc.uni-linz.ac.at/people/hemmecke/aldor/combinat/.

[S+13] W. A. Stein et al. *Sage Mathematics Software (Version 5.8)*. The Sage Development Team, 2013. http://www.sagemath.org.

[SPH01] Neil Schemenauer, Tim Peters, and Lie Hetland. Simple Generators, 2001. http ://www.python.org/dev/peps/pep-0255/.