Being lazy

Martin Rubey

5.8.2025

Being lazy with SageMath

Martin Rubey, Travis Scrimshaw, Mainak Roy, Tejasvi Chebrolu

5.8.2025

Computing with formal power series

```
sage: S.<x> = PowerSeriesRing(QQ)
sage: f = exp(x)
sage: f[4]
1/24
sage: f[42]
```

Computing with formal power series

```
sage: S.<x> = PowerSeriesRing(QQ)
sage: f = exp(x)
sage: f[4]
1/24
sage: f[42]
---------------------------------------------------------------------------
IndexError Traceback (most recent call last)
Cell In[151], line 1
----> 1 f[Integer(42)]

File ~/sage-trac/src/sage/rings/power_series_poly.pyx:462, in sage.rings.power_series_poly.PowerSeries_poly.__getitem__()
    460 return self.base_ring().zero()
    461 else:
--> 462 raise IndexError("coefficient not known")
    463 return self.__f[n]
    464

IndexError: coefficient not known
```

Computing with formal power series

```
sage: S.<x> = PowerSeriesRing(QQ)
sage: f = exp(x)
sage: f[4]
1/24
sage: f[42]
---------------------------------------------------------------------------
IndexError Traceback (most recent call last)
Cell In[151], line 1
----> 1 f[Integer(42)]

File ~/sage-trac/src/sage/rings/power_series_poly.pyx:462, in sage.rings.power_series_poly.PowerSeries_poly.__getitem__()
    460 return self.base_ring().zero()
    461 else:
--> 462 raise IndexError("coefficient not known")
    463 return self.__f[n]
    464

IndexError: coefficient not known
```

Relax!

Computing with formal power series

```
sage: S.<x> = PowerSeriesRing(QQ)
sage: f = exp(x)
sage: f[4]
1/24
sage: f[42]
---------------------------------------------------------------------------
IndexError Traceback (most recent call last)
Cell In[151], line 1
----> 1 f[Integer(42)]

File ~/sage-trac/src/sage/rings/power_series_poly.pyx:462, in sage.rings.power_series_poly.PowerSeries_poly.__getitem__()
    460 return self.base_ring().zero()
    461 else:
--> 462 raise IndexError("coefficient not known")
    463 return self.__f[n]
    464

IndexError: coefficient not known
```

Relax!

```
sage: S.<x> = LazyPowerSeriesRing(QQ)
sage: f = exp(x)
sage: f[42]
```

Computing with formal power series

```
sage: S.<x> = PowerSeriesRing(QQ)
sage: f = exp(x)
sage: f[4]
1/24
sage: f[42]
---------------------------------------------------------------------------
IndexError Traceback (most recent call last)
Cell In[151], line 1
----> 1 f[Integer(42)]

File ~/sage-trac/src/sage/rings/power_series_poly.pyx:462, in sage.rings.power_series_poly.PowerSeries_poly.__getitem__()
    460 return self.base_ring().zero()
    461 else:
--> 462 raise IndexError("coefficient not known")
    463 return self.__f[n]
    464

IndexError: coefficient not known
```

Relax!

```
sage: S.<x> = LazyPowerSeriesRing(QQ)
sage: f = exp(x)
sage: f[42]
1/1405006117752879898543142606244511569936384000000000
```

Computing with lazy formal power series

- operations are easier to implement

Computing with lazy formal power series

- operations are easier to implement (just 17 000 lines of code)

Computing with lazy formal power series

- operations are easier to implement (just 17 000 lines of code)
- getting all of the first few coefficients takes longer

Computing with lazy formal power series

- operations are easier to implement (just 17 000 lines of code)
- getting all of the first few coefficients takes longer
- more convenient to use

Computing with lazy formal power series

- operations are easier to implement (just 17 000 lines of code)
- getting all of the first few coefficients takes longer
- more convenient to use
- more powerful: implicit definitions are easy to resolve

Implicit definitions, part 1

A binary tree is empty, or a root with two binary trees attached:

```
sage: S.<x> = LazyPowerSeriesRing(QQ)
sage: b = S.undefined()
sage: b.define(1 + x*b^2)
```

Implicit definitions, part 1

A binary tree is empty, or a root with two binary trees attached:

```
sage: S.<x> = LazyPowerSeriesRing(QQ)
sage: b = S.undefined()
sage: b.define(1 + x*b^2)
sage: b
1 + x + 2*x^2 + 5*x^3 + 14*x^4 + 42*x^5 + 132*x^6 + O(x^7)
```

Implicit definitions, part 1

A binary tree is empty, or a root with two binary trees attached:

```
sage: S.<x> = LazyPowerSeriesRing(QQ)
sage: b = S.undefined()
sage: b.define(1 + x*b^2)
sage: b
1 + x + 2*x^2 + 5*x^3 + 14*x^4 + 42*x^5 + 132*x^6 + O(x^7)
```

Trees with two or three children depending on the parity of the level:

```
sage: S.<x> = LazyPowerSeriesRing(QQ)
sage: b = S.undefined()
sage: c = S.undefined()
sage: b.define(1 + x*c^3)
sage: c.define(1 + x*b^2)
```

Implicit definitions, part 1

A binary tree is empty, or a root with two binary trees attached:

```
sage: S.<x> = LazyPowerSeriesRing(QQ)
sage: b = S.undefined()
sage: b.define(1 + x*b^2)
sage: b
1 + x + 2*x^2 + 5*x^3 + 14*x^4 + 42*x^5 + 132*x^6 + O(x^7)
```

Trees with two or three children depending on the parity of the level:

```
sage: S.<x> = LazyPowerSeriesRing(QQ)
sage: b = S.undefined()
sage: c = S.undefined()
sage: b.define(1 + x*c^3)
sage: c.define(1 + x*b^2)
sage: b
1 + x + 3*x^2 + 9*x^3 + 34*x^4 + 132*x^5 + 546*x^6 + O(x^7)
```

Implicit definitions, part 2

Solve the differential equation

$$f''(x) + f(x) = 0$$

with initial conditions $f(0) = 1$, $f'(0) = 0$:

```
sage: S.<x> = LazyPowerSeriesRing(QQ)
sage: f = S.undefined()
sage: S.define_implicitly([(f, [1, 0])], [diff(f, 2) + f])
```

Implicit definitions, part 2

Solve the differential equation

$$f''(x) + f(x) = 0$$

with initial conditions $f(0) = 1$, $f'(0) = 0$:

```
sage: S.<x> = LazyPowerSeriesRing(QQ)
sage: f = S.undefined()
sage: S.define_implicitly([(f, [1, 0])], [diff(f, 2) + f])
sage: f
1 - 1/2*x^2 + 1/24*x^4 - 1/720*x^6 + O(x^7)
```

Implicit definitions, part 3

A "Lambertization-like" operator on functions

Asked 10 days ago Modified 6 days ago Viewed 266 times

7

Define an operator L on, say, formal series $f(x)$ with $f(0) = 1$ by requiring that $L(f) = F$ is the solution of the functional equation

$$F(xf(x)) = f(x).$$

Some examples:

```
sage: R.<c> = InfinitePolynomialRing(QQ)
sage: S.<x> = LazyPowerSeriesRing(R.fraction_field())
sage: f = S(lambda n: c[n] if n else 1); f
1 + c_1*x + c_2*x^2 + c_3*x^3 + c_4*x^4 + c_5*x^5 + O(x^6)
sage: F = S.undefined()
sage: S.define_implicitly([F], [F(x*f) - f])
```

Implicit definitions, part 3

mathoverflow

A "Lambertization-like" operator on functions

Asked 10 days ago Modified 6 days ago Viewed 266 times

7

Define an operator L on, say, formal series $f(x)$ with $f(0) = 1$ by requiring that $L(f) = F$ is the solution of the functional equation

$$F(xf(x)) = f(x).$$

Some examples:

```
sage: R.<c> = InfinitePolynomialRing(QQ)
sage: S.<x> = LazyPowerSeriesRing(R.fraction_field())
sage: f = S(lambda n: c[n] if n else 1); f
1 + c_1*x + c_2*x^2 + c_3*x^3 + c_4*x^4 + c_5*x^5 + O(x^6)
sage: F = S.undefined()
sage: S.define_implicitly([F], [F(x*f) - f])
sage: F[4]
c_4 - 4*c_3*c_1 - 2*c_2^2 + 10*c_2*c_1^2 - 5*c_1^4
```

Formal series

The same code handles

- Laurent series

Formal series

The same code handles

- Laurent series
- multivariate power series

Formal series

The same code handles

- Laurent series
- multivariate power series
- Dirichlet series

Formal series

The same code handles

- Laurent series
- multivariate power series
- Dirichlet series
- symmetric functions in any number of alphabets

Formal series

The same code handles

- Laurent series
- multivariate power series
- Dirichlet series
- symmetric functions in any number of alphabets
- combinatorial species in any number of sorts

Formal series

The same code handles

- Laurent series
- multivariate power series
- Dirichlet series
- symmetric functions in any number of alphabets
- combinatorial species in any number of sorts
- the completion of any graded algebra in `SageMath`

Multivariate power series

A path with unit diagonal steps above the x-axis either ends with an up step or with a down step.

$$f(z, x) = \sum_{p} x^{\text{length of } p} y^{\text{final height of } p}$$

satisfies

$$f(x, y) = 1 + xyf(x, y) + \frac{x}{y}\big(f(x, y) - f(x, 0)\big)$$

Multivariate power series

A path with unit diagonal steps above the x-axis either ends with an up step or with a down step.

$$f(z, x) = \sum_p x^{\text{length of } p} y^{\text{final height of } p}$$

satisfies

$$f(x, y) = 1 + xyf(x, y) + \frac{x}{y}\big(f(x, y) - f(x, 0)\big)$$

```
sage: S.<x, y> = LazyPowerSeriesRing(QQ)
sage: f = S.undefined()
sage: eq = y + x*y^2*f + x*(f - f(x, 0)) - y*f
sage: S.define_implicitly([f], [eq])
```

Multivariate power series

A path with unit diagonal steps above the x-axis either ends with an up step or with a down step.

$$f(z, x) = \sum_{p} x^{\text{length of } p} y^{\text{final height of } p}$$

satisfies

$$f(x, y) = 1 + xyf(x, y) + \frac{x}{y}\big(f(x, y) - f(x, 0)\big)$$

```
sage: S.<x, y> = LazyPowerSeriesRing(QQ)
sage: f = S.undefined()
sage: eq = y + x*y^2*f + x*(f - f(x, 0)) - y*f
sage: S.define_implicitly([f], [eq])
sage: f
1 + (x^2+x*y) + (2*x^4+2*x^3*y+x^2*y^2)
+ (5*x^6+5*x^5*y+3*x^4*y^2+x^3*y^3) + O(x,y)^7
```

Dirichlet series

Let a_n be the number of ordered factorization of the integer n into parts strictly larger than 1. $a_8 = 4$ because

$$8 = 4 \cdot 2 = 2 \cdot 4 = 2 \cdot 2 \cdot 2.$$

Let $f(s) = \sum_n a_n/n^s$, then $f(s) = 1 + (\sum_{n>1} 1/n^s)f(s)$:

```
sage: S = LazyDirichletSeriesRing(ZZ, "s")
sage: g = S(constant=1, valuation=2)
sage: g
1/(2^z) + 1/(3^z) + 1/(4^z) + O(1/(5^z))
sage: f = S.undefined()
sage: f.define(1 + g*f)
```

Dirichlet series

Let a_n be the number of ordered factorization of the integer n into parts strictly larger than 1. $a_8 = 4$ because

$$8 = 4 \cdot 2 = 2 \cdot 4 = 2 \cdot 2 \cdot 2.$$

Let $f(s) = \sum_n a_n/n^s$, then $f(s) = 1 + (\sum_{n>1} 1/n^s)f(s)$:

```
sage: S = LazyDirichletSeriesRing(ZZ, "s")
sage: g = S(constant=1, valuation=2)
sage: g
1/(2^z) + 1/(3^z) + 1/(4^z) + O(1/(5^z))
sage: f = S.undefined()
sage: f.define(1 + g*f)
sage: f
1 + 1/(2^s) + 1/(3^s) + 2/4^s + 1/(5^s) + 3/6^s + 1/(7^s)
+ 4/8^s + O(1/(9^s))
sage: oeis(f[:10])
...
1: A074206: Kalmar's problem: number of ordered factorizations
            of n.
```

Free algebra

Create all words in X and Y such that there are no two consecutive Y's.

```
sage: A.<X,Y> = FreeAlgebra(QQ, degrees=[1,1])
sage: L = A.completion()
sage: F = L.undefined()
sage: G = L.undefined()
sage: F.define(1 + X*F + Y*G)
sage: G.define(1 + X*F)
```

Free algebra

Create all words in X and Y such that there are no two consecutive Y's.

```
sage: A.<X,Y> = FreeAlgebra(QQ, degrees=[1,1])
sage: L = A.completion()
sage: F = L.undefined()
sage: G = L.undefined()
sage: F.define(1 + X*F + Y*G)
sage: G.define(1 + X*F)
sage: F[5]
X^5 + X^4*Y + X^3*Y*X + X^2*Y*X^2 + X^2*Y*X*Y + X*Y*X^3
+ X*Y*X^2*Y + X*Y*X*Y*X + Y*X^4 + Y*X^3*Y + Y*X^2*Y*X
+ Y*X*Y*X^2 + Y*X*Y*X*Y
```

Symmetric functions

Let us verify

$$h_n[XY] = \sum_{\lambda \vdash n} h_\lambda[X] m_\lambda[Y].$$

```
sage: m = SymmetricFunctions(QQ).m()
sage: h = SymmetricFunctions(QQ).h()
sage: L = LazySymmetricFunctions(h)
sage: X = tensor([h[1], m[[]]])
sage: Y = tensor([h[[]], m[1]])
sage: H = L(lambda n: h[n])
```

Symmetric functions

Let us verify

$$h_n[XY] = \sum_{\lambda \vdash n} h_\lambda[X] m_\lambda[Y].$$

```
sage: m = SymmetricFunctions(QQ).m()
sage: h = SymmetricFunctions(QQ).h()
sage: L = LazySymmetricFunctions(h)
sage: X = tensor([h[1], m[[]]])
sage: Y = tensor([h[[]], m[1]])
sage: H = L(lambda n: h[n])
sage: H(X*Y)
(h[]#m[]) + (h[1]#m[1]) + (h[1,1]#m[1,1]+h[2]#m[2])
+ (h[1,1,1]#m[1,1,1]+h[2,1]#m[2,1]+h[3]#m[3])
+ O^8
```

Combinatorial species

A rooted tree is a root together with a set of trees:

$$\mathcal{A} = X\mathcal{E}(\mathcal{A})$$

```
sage: S.<X> = LazyCombinatorialSpecies(QQ)
sage: E = S.Sets()
sage: A = S.undefined()
sage: A.define(X*E(A))
```

Combinatorial species

A rooted tree is a root together with a set of trees:

$$\mathcal{A} = X\mathcal{E}(\mathcal{A})$$

```
sage: S.<X> = LazyCombinatorialSpecies(QQ)
sage: E = S.Sets()
sage: A = S.undefined()
sage: A.define(X*E(A))
sage: A
X + X^2 + (X*E_2+X^3) + (X*E_3+X^2*E_2+2*X^4)
+ (X*E_4+X^2*E_3+3*X^3*E_2+X*E_2(X^2)+3*X^5) + O^6
```

Combinatorial species

```
sage: A[5]
X*E_4(X) + 3*X^3*E_2(X) + X*E_2(X^2) + X^2*E_3(X) + 3*X^5
sage: ascii_art(list(RootedTrees(5)))

[ o,   o ,   o ,    o  ,   o_,   _o__ ,   o_,   _o__,   __o___ ]
[ |    |     |      |     / /   /   /    / /   / / /   / / / / ]
[ o    o     o_    _o__  o o   o   o_   o o   o o o   o o o o  ]
[ |    |    / /   / / /    |      / /   | |       |            ]
[ o    o_  o o   o o o     o     o o    o o       o            ]
[ |   / /    |             |                                   ]
[ o  o o     o             o                                   ]
[ |                                                            ]
[ o                                                            ]
```

Weighted multisort species

Rooted trees with internal vertices of sort X and leaves of sort Y satisfy

$$\mathcal{A} = X\mathcal{E}_+(\mathcal{A}) + Y$$

```
sage: S.<X, Y> = LazyCombinatorialSpecies(QQ)
sage: A = S.undefined()
sage: A.define(X*(E-1)(A) + Y)
```

Weighted multisort species

Rooted trees with internal vertices of sort X and leaves of sort Y satisfy

$$\mathcal{A} = X\mathcal{E}_+(\mathcal{A}) + Y$$

```
sage: S.<X, Y> = LazyCombinatorialSpecies(QQ)
sage: A = S.undefined()
sage: A.define(X*(E-1)(A) + Y)
sage: A[5]
X^4*Y + X^3*E_2(Y) + X*E_2(X*Y) + 2*X^3*Y^2 + X^2*E_3(Y)
+ 2*X^2*Y*E_2(Y) + X*E_4(Y)
```

Weighted multisort species

Rooted trees with internal vertices of sort X and leaves of sort Y satisfy

$$\mathcal{A} = X\mathcal{E}_+(\mathcal{A}) + Y$$

```
sage: S.<X, Y> = LazyCombinatorialSpecies(QQ)
sage: A = S.undefined()
sage: A.define(X*(E-1)(A) + Y)
sage: A[5]
X^4*Y + X^3*E_2(Y) + X*E_2(X*Y) + 2*X^3*Y^2 + X^2*E_3(Y)
+ 2*X^2*Y*E_2(Y) + X*E_4(Y)
```

Let us now compute $\mathcal{A}(X, tX)$:

```
sage: R.<t> = QQ[]
sage: S.<X, Y> = LazyCombinatorialSpecies(R)
sage: E = LazyCombinatorialSpecies(R, "X").Sets()
sage: A = S.undefined()
sage: A.define(X*(E-1)(A) + Y)
sage: A(X, t*X)[5]
t^4*X*E_4(X) + (2*t^3+t^2)*X^3*E_2(X) + t^2*X*E_2(X^2)
+ t^3*X^2*E_3(X) + (2*t^2+t)*X^5
```

Weighted multisort species

```
sage: A(X, t*X)[5]
t^4*X*E_4(X) + (2*t^3+t^2)*X^3*E_2(X) + t^2*X*E_2(X^2)
+ t^3*X^2*E_3(X) + (2*t^2+t)*X^5
```

```
sage: ascii_art(list(RootedTrees(5)))
[ o,   o ,   o ,    o  ,   o_,   _o__ ,   o_,   _o__,   __o___ ]
[ |    |     |      |     / /   /   /    / /   / / /   / / / / ]
[ o    o     o_    _o__  o o   o   o_   o o   o o o   o o o o  ]
[ |    |    / /   / / /    |      / /   | |       |            ]
[ o    o_  o o   o o o     o     o o   o o       o            ]
[ |   / /    |             |                                   ]
[ o  o o     o             o                                   ]
[ |                                                            ]
[ o                                                            ]
```

Open problems, part 1

Generalize `define_implicitly` to composition of weighted `LazySymmetricFunctions` and `LazyCombinatorialSpecies`.

Open problems, part 1

Generalize `define_implicitly` to composition of weighted `LazySymmetricFunctions` and `LazyCombinatorialSpecies`.
What is the problem?

```
sage: S.<x> = LazyPowerSeriesRing(QQ)
sage: f = S.undefined()
sage: S.define_implicitly([f], [eq])
```

Open problems, part 1

Generalize `define_implicitly` to composition of weighted `LazySymmetricFunctions` and `LazyCombinatorialSpecies`. What is the problem?

```
sage: S.<x> = LazyPowerSeriesRing(QQ)
sage: f = S.undefined()
sage: S.define_implicitly([f], [eq])
```

1. defines $f = \sum_n f_n x^n$,
2. compute the first non-zero coefficient of the power series eq,
3. if this is a linear equation in f_n, solve it, otherwise fail.

Step 2 fails for $F \circ G$ of `LazyCombinatorialSpecies`, if G is the unknown species.

For example, for an indeterminate a,

$$\mathcal{E}(aX) = 1 + aX + \left(a\mathcal{E}_2 + \binom{a}{2}X^2\right) + \left(a\mathcal{E}_3 + a(a-1)X\mathcal{E}_2 + \binom{a}{3}X^3\right) + \dots$$

Open problems, part 1

Generalize `define_implicitly` to composition of weighted `LazySymmetricFunctions` and `LazyCombinatorialSpecies`.
What is the problem?

```
sage: S.<x> = LazyPowerSeriesRing(QQ)
sage: f = S.undefined()
sage: S.define_implicitly([f], [eq])
```

1. defines $f = \sum_n f_n x^n$,
2. compute the first non-zero coefficient of the power series eq,
3. if this is a linear equation in f_n, solve it, otherwise fail.

Step 2 fails for $F \circ G$ of `LazyCombinatorialSpecies`, if G is the unknown species.
We would need to be able to compose with

$$a + bX + cX^2 + d\mathcal{E}_2 + eX^3 + fX\mathcal{E}_2 + g\mathcal{C}_3 + h\mathcal{E}_4 + \dots$$

Open problems, part 1

Generalize `define_implicitly` to composition of weighted `LazySymmetricFunctions` and `LazyCombinatorialSpecies`.
What is the problem?

```
sage: S.<x> = LazyPowerSeriesRing(QQ)
sage: f = S.undefined()
sage: S.define_implicitly([f], [eq])
```

1. defines $f = \sum_n f_n x^n$,
2. compute the first non-zero coefficient of the power series eq,
3. if this is a linear equation in f_n, solve it, otherwise fail.

Step 2 fails for $F \circ G$ of `LazyCombinatorialSpecies`, if G is the unknown species.
but for a weight t,

$$\mathcal{E}(tX) = 1 + tX + t^2\mathcal{E}_2 + t^3\mathcal{E}_3 + \dots,$$

composition behaves differently.

Open problems, part 2

Find a better algorithm for functorial composition.
In terms of representations and group actions:

- let $\mathfrak{S}_n$ be the symmetric group
- let $g : \mathfrak{S}_n \to \mathfrak{S}_m$ be a representation, given as a list of stabilizer subgroups of the corresponding (right) action $\{1, \dots, m\} \times \mathfrak{S}_n \to \{1, \dots, m\}$
- let $f : X \times \mathfrak{S}_m \to X$ be a transitive action, given by its stabilizer subgroup

Compute the stabilizer subgroups for each orbit of the action

$$\begin{aligned} fg : X \times \mathfrak{S}_n &\to X \\ (x, \pi) &\mapsto f\big(x, g(\pi)\big) \end{aligned}$$