Being lazy

Martin Rubey

17.9.2024

Being lazy with SageMath

Martin Rubey, Travis Scrimshaw, Mainak Roy, Tejasvi Chebrolu

17.9.2024

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 17 340 lines of code)

Computing with lazy formal power series

- operations are easier to implement (just 17 340 lines of code)
- getting all of the first few coefficients takes longer

Computing with lazy formal power series

- operations are easier to implement (just 17 340 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 17 340 lines of code)
- getting all of the first few coefficients takes longer
- more convenient to use
- more powerful: implicit definitions are easy to resolve

Inplicit 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)
```

Inplicit 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)
```

Inplicit 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)
```

Inplicit 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)
```

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)
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.
```

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 = LazySpecies(QQ, "X")
sage: X = S(SymmetricGroup(1))
sage: E = S(lambda n: SymmetricGroup(n))
```

Combinatorial species

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

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

```
sage: S = LazySpecies(QQ, "X")
sage: X = S(SymmetricGroup(1))
sage: E = S(lambda n: SymmetricGroup(n))
sage: E
1 + X + E_2 + E_3 + E_4 + E_5 + E_6 + O^7
sage: A = S.undefined()
sage: A.define(X*E(A))
sage: A
```

Combinatorial species

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

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

```
sage: S = LazySpecies(QQ, "X")
sage: X = S(SymmetricGroup(1))
sage: E = S(lambda n: SymmetricGroup(n))
sage: E
1 + X + E_2 + E_3 + E_4 + E_5 + E_6 + O^7
sage: A = S.undefined()
sage: A.define(X*E(A))
sage: A
X + X^2 + (X^3+X*E_2) + (X^2*E_2+2*X^4+X*E_3)
+ (X^2*E_3+3*X^5+3*X^3*E_2+X*{((1,2)(3,4),)}+X*E_4)
+ O^6
```

Combinatorial species

```
sage: A[5]
X^2*E_3 + 3*X^5 + 3*X^3*E_2 + X*{((1,2)(3,4),)} + X*E_4
```

```
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: S2 = LazySpecies(QQ, "X, Y")
sage: X = S2((SymmetricGroup(1), {0: [1]}))
sage: Y = S2((SymmetricGroup(1), {1: [1]}))
sage: A = S2.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: S2 = LazySpecies(QQ, "X, Y")
sage: X = S2((SymmetricGroup(1), {0: [1]}))
sage: Y = S2((SymmetricGroup(1), {1: [1]}))
sage: A = S2.undefined()
sage: A.define(X*(E-1)(A) + Y)
sage: A[5]
X^2*E_3(Y) + 2*Y^2*X^3 + X^4*Y + X^3*E_2(Y) + X*E_4(Y)
+ X*{((1,2)(3,4),): ({1, 2}, {3, 4})} + 2*Y*X^2*E_2(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: S2 = LazySpecies(QQ, "X, Y")
sage: X = S2((SymmetricGroup(1), {0: [1]}))
sage: Y = S2((SymmetricGroup(1), {1: [1]}))
sage: A = S2.undefined()
sage: A.define(X*(E-1)(A) + Y)
sage: A[5]
X^2*E_3(Y) + 2*Y^2*X^3 + X^4*Y + X^3*E_2(Y) + X*E_4(Y)
+ X*{((1,2)(3,4),): ({1, 2}, {3, 4})} + 2*Y*X^2*E_2(Y)
```

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

```
sage: R.<t> = QQ[]
sage: St = LazySpecies(R, "X")
sage: X = St(SymmetricGroup(1))
sage: A(X, t*X)[5]
(2*t^2+t)*X^5 + (2*t^3+t^2)*E_2*X^3 + t^2*X*{((1,2)(3,4),)}
+ t^3*E_3*X^2 + t^4*E_4*X
```

Weighted multisort species

```
sage: A(X, t*X)[5]
(2*t^2+t)*X^5 + (2*t^3+t^2)*E_2*X^3 + t^2*X*{((1,2)(3,4),)}
+ t^3*E_3*X^2 + t^4*E_4*X
```

```
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                                                            ]
```