Sage Reference Manual: C/C++ Library Interfaces

Release 7.6

The Sage Development Team

CONTENTS

1	Sage interface to Cremona's eclib library (also known as mwrank)	3
2	Cython interface to Cremona's eclib library (also known as mwrank)	19
3	Cremona matrices	21
4	Modular symbols using eclib newforms	23
5	Cremona modular symbols	25
6	Cremona modular symbols	29
7	Rubinstein's lcalc library	31
8	Interface between Sage and PARI 8.1 Guide to real precision in the PARI interface	39 39
9	Convert PARI objects to Sage types	43
10	Ring of pari objects	47
11	Hyperelliptic Curve Point Finding, via ratpoints.	49
12	libSingular: Functions	51
13	libSingular: Function Factory	61
14	libSingular: Conversion Routines and Initialisation	63
15	Wrapper for Singular's Polynomial Arithmetic	65
16	libSingular: Options	67
17	Wrapper for Singular's Rings	73
18	Singular's Groebner Strategy Objects	75
19	Cython wrapper for the Parma Polyhedra Library (PPL)	79
20	Linbox interface	139
21	Flint imports	141
22	FLINT fmpz_poly class wrapper	143

23	FLINT Arithmetic Functions	147
24	Symmetrica library	149
25	Utilities for Sage-mpmath interaction	157
26	Victor Shoup's NTL C++ Library	161
27	The Elliptic Curve Method for Integer Factorization (ECM)	163
28	An interface to Anders Buch's Littlewood-Richardson Calculator lrcalc	165
29	Readline	173
30	Context Managers for LibGAP	177
31	Gap functions	179
32	Long tests for libGAP	181
33	Utility functions for libGAP	183
34	libGAP shared library Interface to GAP 34.1 Using the libGAP C library from Cython	185 187
35	Short tests for libGAP	193
36	libGAP element wrapper	195
37	LibGAP Workspace Support	209
38	Library interface to Embeddable Common Lisp (ECL)	211
39	GSL arrays	219
40	Indices and Tables	221

An underlying philosophy in the development of Sage is that it should provide unified library-level access to the some of the best GPL'd C/C++ libraries. Sage provides access to many libraries which are included with Sage.

The interfaces are implemented via shared libraries and data is moved between systems purely in memory. In particular, there is no interprocess interpreter parsing (e.g., pexpect), since everything is linked together and run as a single process. This is much more robust and efficient than using pexpect.

Each of these interfaces is used by other parts of Sage. For example, eclib is used by the elliptic curves module to compute ranks of elliptic curves and PARI is used for computation of class groups. It is thus probably not necessary for a casual user of Sage to be aware of the modules described in this chapter.

CONTENTS 1

2 CONTENTS

CHAPTER

ONE

SAGE INTERFACE TO CREMONA'S ECLIB LIBRARY (ALSO KNOWN AS MWRANK)

This is the Sage interface to John Cremona's eclib C++ library for arithmetic on elliptic curves. The classes defined in this module give Sage interpreter-level access to some of the functionality of eclib. For most purposes, it is not necessary to directly use these classes. Instead, one can create an EllipticCurve and call methods that are implemented using this module.

Note: This interface is a direct library-level interface to eclib, including the 2-descent program mwrank.

```
sage.libs.eclib.interface. get_precision ()
    Return the global NTL real number precision.
```

See also set_precision().

Warning: The internal precision is binary. This function multiplies the binary precision by $0.3 (= \log_2(10)$ approximately) and truncates.

OUTPUT:

(int) The current decimal precision.

EXAMPLES:

```
sage: mwrank_get_precision()
50
```

class sage.libs.eclib.interface. mwrank_EllipticCurve (ainvs, verbose=False)

Bases: sage.structure.sage_object.SageObject

The mwrank_EllipticCurve class represents an elliptic curve using the Curvedata class from eclib, called here an 'mwrank elliptic curve'.

Create the mwrank elliptic curve with invariants ainvs, which is a list of 5 or less integers a_1 , a_2 , a_3 , a_4 , and a_5 .

If strictly less than 5 invariants are given, then the *first* ones are set to 0, so, e.g., [3, 4] means $a_1 = a_2 = a_3 = 0$ and $a_4 = 3$, $a_5 = 4$.

INPUT:

- •ainvs (list or tuple) a list of 5 or less integers, the coefficients of a nonsingular Weierstrass equation.
- •verbose (bool, default False) verbosity flag. If True, then all Selmer group computations will be verbose.

EXAMPLES:

We create the elliptic curve $y^2 + y = x^3 + x^2 - 2x$:

```
sage: e = mwrank_EllipticCurve([0, 1, 1, -2, 0])
sage: e.ainvs()
[0, 1, 1, -2, 0]
```

This example illustrates that omitted a-invariants default to 0:

```
sage: e = mwrank_EllipticCurve([3, -4])
sage: e
y^2 = x^3 + 3*x - 4
sage: e.ainvs()
[0, 0, 0, 3, -4]
```

The entries of the input list are coerced to int. If this is impossible, then an error is raised:

```
sage: e = mwrank_EllipticCurve([3, -4.8]); e
Traceback (most recent call last):
...
TypeError: ainvs must be a list or tuple of integers.
```

When you enter a singular model you get an exception:

```
sage: e = mwrank_EllipticCurve([0, 0])
Traceback (most recent call last):
...
ArithmeticError: Invariants (= 0,0,0,0,0) do not describe an elliptic curve.
```

CPS_height_bound ()

Return the Cremona-Prickett-Siksek height bound. This is a floating point number B such that if P is a point on the curve, then the naive logarithmic height h(P) is less than $B + \hat{h}(P)$, where $\hat{h}(P)$ is the canonical height of P.

```
Warning: We assume the model is minimal!
```

EXAMPLES:

```
sage: E = mwrank_EllipticCurve([0, 0, 0, -1002231243161, 0])
sage: E.CPS_height_bound()
14.163198527061496
sage: E = mwrank_EllipticCurve([0,0,1,-7,6])
sage: E.CPS_height_bound()
0.0
```

ainvs ()

Returns the a-invariants of this mwrank elliptic curve.

EXAMPLES:

```
sage: E = mwrank_EllipticCurve([0,0,1,-1,0])
sage: E.ainvs()
[0, 0, 1, -1, 0]
```

certain ()

Returns True if the last two_descent() call provably correctly computed the rank. If

two_descent() hasn't been called, then it is first called by certain() using the default parameters.

The result is True if and only if the results of the methods rank () and rank_bound () are equal.

EXAMPLES:

A 2-descent does not determine $E(\mathbf{Q})$ with certainty for the curve $y^2 + y = x^3 - x^2 - 120x - 2183$:

```
sage: E = mwrank_EllipticCurve([0, -1, 1, -120, -2183])
sage: E.two_descent(False)
...
sage: E.certain()
False
sage: E.rank()
0
```

The previous value is only a lower bound; the upper bound is greater:

```
sage: E.rank_bound()
2
```

In fact the rank of E is actually 0 (as one could see by computing the L-function), but Sha has order 4 and the 2-torsion is trivial, so mwrank cannot conclusively determine the rank in this case.

conductor ()

Return the conductor of this curve, computed using Cremona's implementation of Tate's algorithm.

Note: This is independent of PARI's.

EXAMPLES:

```
sage: E = mwrank_EllipticCurve([1, 1, 0, -6958, -224588])
sage: E.conductor()
2310
```

gens ()

Return a list of the generators for the Mordell-Weil group.

EXAMPLES:

```
sage: E = mwrank_EllipticCurve([0, 0, 1, -1, 0])
sage: E.gens()
[[0, -1, 1]]
```

isogeny_class (verbose=False)

Returns the isogeny class of this mwrank elliptic curve.

EXAMPLES:

rank ()

Returns the rank of this curve, computed using two descent().

In general this may only be a lower bound for the rank; an upper bound may be obtained using the function <code>rank_bound()</code> . To test whether the value has been proved to be correct, use the method <code>certain()</code>

EXAMPLES:

```
sage: E = mwrank_EllipticCurve([0, -1, 0, -900, -10098])
sage: E.rank()
0
sage: E.certain()
True
```

```
sage: E = mwrank_EllipticCurve([0, -1, 1, -929, -10595])
sage: E.rank()
0
sage: E.certain()
False
```

rank bound ()

Returns an upper bound for the rank of this curve, computed using two_descent().

If the curve has no 2-torsion, this is equal to the 2-Selmer rank. If the curve has 2-torsion, the upper bound may be smaller than the bound obtained from the 2-Selmer rank minus the 2-rank of the torsion, since more information is gained from the 2-isogenous curve or curves.

EXAMPLES:

The following is the curve 960D1, which has rank 0, but Sha of order 4:

```
sage: E = mwrank_EllipticCurve([0, -1, 0, -900, -10098])
sage: E.rank_bound()
0
sage: E.rank()
```

In this case the rank was computed using a second descent, which is able to determine (by considering a 2-isogenous curve) that Sha is nontrivial. If we deliberately stop the second descent, the rank bound is larger:

```
sage: E = mwrank_EllipticCurve([0, -1, 0, -900, -10098])
sage: E.two_descent(second_descent = False, verbose=False)
sage: E.rank_bound()
2
```

In contrast, for the curve 571A, also with rank 0 and Sha of order 4, we only obtain an upper bound of 2:

```
sage: E = mwrank_EllipticCurve([0, -1, 1, -929, -10595])
sage: E.rank_bound()
2
```

In this case the value returned by rank () is only a lower bound in general (though this is correct):

```
sage: E.rank()
0
sage: E.certain()
False
```

regulator ()

Return the regulator of the saturated Mordell-Weil group.

EXAMPLES:

```
sage: E = mwrank_EllipticCurve([0, 0, 1, -1, 0])
sage: E.regulator()
0.05111140823996884
```

saturate (bound=-1)

Compute the saturation of the Mordell-Weil group at all primes up to bound.

INPUT:

•bound (int, default -1) – Use -1 (the default) to saturate at *all* primes, 0 for no saturation, or n (a positive integer) to saturate at all primes up to n.

EXAMPLES:

Since the 2-descent automatically saturates at primes up to 20, it is not easy to come up with an example where saturation has any effect:

```
sage: E = mwrank_EllipticCurve([0, 0, 0, -1002231243161, 0])
sage: E.gens()
[[-1001107, -4004428, 1]]
sage: E.saturate()
sage: E.gens()
[[-1001107, -4004428, 1]]
```

Check that trac ticket #18031 is fixed:

```
sage: E = EllipticCurve([0,-1,1,-266,968])
sage: Q1 = E([-1995,3674,125])
sage: Q2 = E([157,1950,1])
sage: E.saturation([Q1,Q2])
([(1 : -27 : 1), (157 : 1950 : 1)], 3, 0.801588644684981)
```

selmer rank ()

Returns the rank of the 2-Selmer group of the curve.

EXAMPLES:

The following is the curve 960D1, which has rank 0, but Sha of order 4. The 2-torsion has rank 2, and the Selmer rank is 3:

```
sage: E = mwrank_EllipticCurve([0, -1, 0, -900, -10098])
sage: E.selmer_rank()
3
```

Nevertheless, we can obtain a tight upper bound on the rank since a second descent is performed which establishes the 2-rank of Sha:

```
sage: E.rank_bound()
0
```

To show that this was resolved using a second descent, we do the computation again but turn off second_descent:

```
sage: E = mwrank_EllipticCurve([0, -1, 0, -900, -10098])
sage: E.two_descent(second_descent = False, verbose=False)
sage: E.rank_bound()
2
```

For the curve 571A, also with rank 0 and Sha of order 4, but with no 2-torsion, the Selmer rank is strictly greater than the rank:

```
sage: E = mwrank_EllipticCurve([0, -1, 1, -929, -10595])
sage: E.selmer_rank()
2
sage: E.rank_bound()
2
```

In cases like this with no 2-torsion, the rank upper bound is always equal to the 2-Selmer rank. If we ask for the rank, all we get is a lower bound:

```
sage: E.rank()
0
sage: E.certain()
False
```

set_verbose (verbose)

Set the verbosity of printing of output by the two_descent () and other functions.

INPUT:

•verbose (int) – if positive, print lots of output when doing 2-descent.

```
sage: E = mwrank_EllipticCurve([0, 0, 1, -1, 0])
sage: E.saturate() # no output
sage: E.gens()
[[0, -1, 1]]
sage: E = mwrank_EllipticCurve([0, 0, 1, -1, 0])
sage: E.set_verbose(1)
sage: E.saturate() # tol 1e-14
Basic pair: I=48, J=-432
disc=255744
2-adic index bound = 2
By Lemma 5.1(a), 2-adic index = 1
2-adic index = 1
One (I, J) pair
Looking for quartics with I = 48, J = -432
Looking for Type 2 quartics:
Trying positive a from 1 up to 1 (square a first...)
(1,0,-6,4,1)
                    --trivial
Trying positive a from 1 up to 1 (...then non-square a)
Finished looking for Type 2 quartics.
Looking for Type 1 quartics:
Trying positive a from 1 up to 2 (square a first...)
(1,0,0,4,4) --nontrivial...(x:y:z) = (1:1:0)
Point = [0:0:1]
   height = 0.0511114082399688402358
Rank of B=im(eps) increases to 1 (The previous point is on the egg)
Exiting search for Type 1 quartics after finding one which is globally...
⇔soluble.
Mordell rank contribution from B=im(eps) = 1
Selmer rank contribution from B=im(eps) = 1
      rank contribution from B=im(eps) = 0
Mordell rank contribution from A=ker(eps) = 0
Selmer rank contribution from A=\ker(eps)=0
```

```
Sha rank contribution from A=ker(eps) = 0

Searching for points (bound = 8)...done:
found points which generate a subgroup of rank 1
and regulator 0.0511114082399688402358

Processing points found during 2-descent...done:
now regulator = 0.0511114082399688402358

Saturating (with bound = -1)...done:
points were already saturated.
```

silverman bound ()

Return the Silverman height bound. This is a floating point number B such that if P is a point on the curve, then the naive logarithmic height h(P) is less than $B + \hat{h}(P)$, where $\hat{h}(P)$ is the canonical height of P.

Warning: We assume the model is minimal!

EXAMPLES:

```
sage: E = mwrank_EllipticCurve([0, 0, 0, -1002231243161, 0])
sage: E.silverman_bound()
18.29545210468247
sage: E = mwrank_EllipticCurve([0,0,1,-7,6])
sage: E.silverman_bound()
6.284833369972403
```

INPUT:

- •verbose (bool, default True) print what mwrank is doing.
- •selmer_only (bool, default False) selmer_only switch.
- •first_limit (int, default 20) bound on |x| + |z| in quartic point search.
- •second_limit (int, default 8) bound on $\log \max(|x|, |z|)$, i.e. logarithmic.
- •n_aux (int, default -1) (only relevant for general 2-descent when 2-torsion trivial) number of primes used for quartic search. n_aux=-1 causes default (8) to be used. Increase for curves of higher rank.
- •second_descent (bool, default True) (only relevant for curves with 2-torsion, where mwrank uses descent via 2-isogeny) flag determining whether or not to do second descent. *Default strongly recommended*.

OUTPUT:

Nothing – nothing is returned.

```
 \begin{array}{lll} \textbf{class} \text{ sage.libs.eclib.interface.} & \textbf{mwrank\_MordellWeil} \ (\textit{curve}, & \textit{verbose=True}, & \textit{pp=1}, \\ & \textit{maxr=999}) \\ & \textbf{Bases:} \text{ sage.structure.sage\_object.SageObject} \end{array}
```

The <code>mwrank_MordellWeil</code> class represents a subgroup of a Mordell-Weil group. Use this class to saturate a specified list of points on an <code>mwrank_EllipticCurve</code>, or to search for points up to some bound.

INPUT:

- •curve (mwrank_EllipticCurve) the underlying elliptic curve.
- •verbose (bool, default False) verbosity flag (controls amount of output produced in point searches).
- •pp (int, default 1) process points flag (if nonzero, the points found are processed, so that at all times only a **Z**-basis for the subgroup generated by the points found so far is stored; if zero, no processing is done and all points found are stored).

•maxr (int, default 999) – maximum rank (quit point searching once the points found generate a subgroup of this rank; useful if an upper bound for the rank is already known).

EXAMPLES:

```
sage: E = mwrank_EllipticCurve([1,0,1,4,-6])
sage: EQ = mwrank_MordellWeil(E)
sage: EQ
Subgroup of Mordell-Weil group: []
sage: EQ.search(2)
P1 = [0:1:0]
                is torsion point, order 1
P1 = [1:-1:1] is torsion point, order 2
P1 = [2:2:1]
                is torsion point, order 3
P1 = [9:23:1] is torsion point, order 6
sage: E = mwrank\_EllipticCurve([0,0,1,-7,6])
sage: EQ = mwrank_MordellWeil(E)
sage: EQ.search(2)
P1 = [0:1:0] is torsion point, order 1

P1 = [-3:0:1] is generator number 1
P4 = [-91:804:343]
                         = -2*P1 + 2*P2 + 1*P3 \pmod{torsion}
sage: EQ
Subgroup of Mordell-Weil group: [[1:-1:1], [-2:3:1], [-14:25:8]]
```

Example to illustrate the verbose parameter:

```
sage: E = mwrank_EllipticCurve([0,0,1,-7,6])
sage: EQ = mwrank_MordellWeil(E, verbose=False)
sage: EQ.search(1)
sage: EO
Subgroup of Mordell-Weil group: [[1:-1:1], [-2:3:1], [-14:25:8]]
sage: EQ = mwrank_MordellWeil(E, verbose=True)
sage: EQ.search(1)
P1 = [0:1:0]
               is torsion point, order 1
P1 = [-3:0:1]
                is generator number 1
saturating up to 20...Checking 2-saturation
Points have successfully been 2-saturated (max q used = 7)
Checking 3-saturation
Points have successfully been 3-saturated (max q used = 7)
Checking 5-saturation
Points have successfully been 5-saturated (max q used = 23)
Checking 7-saturation
Points have successfully been 7-saturated (max q used = 41)
Checking 11-saturation
Points have successfully been 11-saturated (max q used = 17)
Checking 13-saturation
Points have successfully been 13-saturated (max q used = 43)
Checking 17-saturation
Points have successfully been 17-saturated (max q used = 31)
Checking 19-saturation
```

```
Points have successfully been 19-saturated (max q used = 37)
P2 = [-2:3:1]
                 is generator number 2
saturating up to 20...Checking 2-saturation
possible kernel vector = [1,1]
This point may be in 2E(Q): [14:-52:1]
...and it is!
Replacing old generator #1 with new generator [1:-1:1]
Points have successfully been 2-saturated (max q used = 7)
Index gain = 2^1
Checking 3-saturation
Points have successfully been 3-saturated (max q used = 13)
Checking 5-saturation
Points have successfully been 5-saturated (max q used = 67)
Checking 7-saturation
Points have successfully been 7-saturated (max q used = 53)
Checking 11-saturation
Points have successfully been 11-saturated (max q used = 73)
Checking 13-saturation
Points have successfully been 13-saturated (max q used = 103)
Checking 17-saturation
Points have successfully been 17-saturated (max q used = 113)
Checking 19-saturation
Points have successfully been 19-saturated (max q used = 47)
done (index = 2).
Gained index 2, new generators = [1:-1:1] [-2:3:1]
P3 = [-14:25:8] is generator number 3
saturating up to 20...Checking 2-saturation
Points have successfully been 2-saturated (max q used = 11)
Checking 3-saturation
Points have successfully been 3-saturated (max q used = 13)
Checking 5-saturation
Points have successfully been 5-saturated (max q used = 71)
Checking 7-saturation
Points have successfully been 7-saturated (max q used = 101)
Checking 11-saturation
Points have successfully been 11-saturated (max q used = 127)
Checking 13-saturation
Points have successfully been 13-saturated (max q used = 151)
Checking 17-saturation
Points have successfully been 17-saturated (max q used = 139)
Checking 19-saturation
Points have successfully been 19-saturated (max q used = 179)
done (index = 1).
P4 = [-1:3:1] = -1*P1 + -1*P2 + -1*P3 \pmod{torsion}
               = 2*P1 + 0*P2 + 1*P3 \pmod{torsion}
P4 = [0:2:1]
P4 = [2:13:8] = -3*P1 + 1*P2 + -1*P3 \pmod{torsion}
P4 = [1:0:1]
               = -1*P1 + 0*P2 + 0*P3 \pmod{torsion}
P4 = [2:0:1]
                = -1*P1 + 1*P2 + 0*P3 \pmod{torsion}
P4 = [18:7:8] = -2*P1 + -1*P2 + -1*P3 \pmod{torsion}
               = 1*P1 + 0*P2 + 1*P3 \pmod{torsion}
P4 = [3:3:1]
P4 = [4:6:1]
                = 0*P1 + -1*P2 + -1*P3 \pmod{torsion}
P4 = [36:69:64] = 1*P1 + -2*P2 + 0*P3 \pmod{torsion}
P4 = [68:-25:64]
                        = -2*P1 + -1*P2 + -2*P3 \pmod{torsion}
P4 = [12:35:27] = 1*P1 + -1*P2 + -1*P3 \pmod{torsion}
sage: EO
Subgroup of Mordell-Weil group: [[1:-1:1], [-2:3:1], [-14:25:8]]
```

Example to illustrate the process points (pp) parameter:

points ()

Return a list of the generating points in this Mordell-Weil group.

OUTPUT:

(list) A list of lists of length 3, each holding the primitive integer coordinates [x, y, z] of a generating point.

EXAMPLES:

process (v, sat=0)

This function allows one to add points to a mwrank_MordellWeil object.

Process points in the list v , with saturation at primes up to sat . If sat is zero (the default), do no saturation.

INPUT:

- •v (list of 3-tuples or lists of ints or Integers) a list of triples of integers, which define points on the curve.
- •sat (int, default 0) saturate at primes up to sat, or at *all* primes if sat is zero.

OUTPUT:

None. But note that if the verbose flag is set, then there will be some output as a side-effect.

```
sage: EQ.points()
[[1, -1, 1], [-2, 3, 1], [-14, 25, 8]]
```

Example to illustrate the saturation parameter sat:

Here the processing was followed by saturation at primes up to 20. Now we prevent this initial saturation:

```
sage: E = mwrank_EllipticCurve([0,0,1,-7,6])
sage: EQ = mwrank_MordellWeil(E)
sage: EQ.process([[1547, -2967, 343], [2707496766203306, 864581029138191,_
\rightarrow2969715140223272], [-13422227300, -49322830557, 12167000000]], sat=0)
P1 = [1547:-2967:343]
                              is generator number 1
P2 = [2707496766203306:864581029138191:2969715140223272]
                                                               is generator.
→number 2
P3 = [-13422227300:-49322830557:12167000000]
                                                       is generator number 3
sage: EQ.points()
[[1547, -2967, 343], [2707496766203306, 864581029138191, 2969715140223272], [-
\hookrightarrow13422227300, -49322830557, 12167000000]]
sage: EQ.regulator()
375.42919921875
sage: EQ.saturate(2) # points were not 2-saturated
saturating basis...Saturation index bound = 93
WARNING: saturation at primes p > 2 will not be done;
Gained index 2
New regulator = 93.857300720636393209
(False, 2, '[]')
sage: EQ.points()
[[-2, 3, 1], [2707496766203306, 864581029138191, 2969715140223272], [-
\hookrightarrow13422227300, -49322830557, 12167000000]]
sage: EQ.regulator()
93.8572998046875
sage: EQ.saturate(3) # points were not 3-saturated
saturating basis...Saturation index bound = 46
WARNING: saturation at primes p > 3 will not be done;
Gained index 3
New regulator = 10.4285889689595992455
(False, 3, '[]')
sage: EO.points()
[[-2, 3, 1], [-14, 25, 8], [-13422227300, -49322830557, 121670000001]
sage: EQ.regulator()
10.4285888671875
sage: EQ.saturate(5) # points were not 5-saturated
saturating basis...Saturation index bound = 15
WARNING: saturation at primes p > 5 will not be done;
Gained index 5
New regulator = 0.417143558758383969818
(False, 5, '[]')
sage: EQ.points()
```

```
[[-2, 3, 1], [-14, 25, 8], [1, -1, 1]]
sage: EQ.regulator()
0.4171435534954071
sage: EQ.saturate()  # points are now saturated
saturating basis...Saturation index bound = 3
Checking saturation at [ 2 3 ]
Checking 2-saturation
Points were proved 2-saturated (max q used = 11)
Checking 3-saturation
Points were proved 3-saturated (max q used = 13)
done
(True, 1, '[]')
```

rank ()

Return the rank of this subgroup of the Mordell-Weil group.

OUTPUT:

(int) The rank of this subgroup of the Mordell-Weil group.

EXAMPLES:

```
sage: E = mwrank_EllipticCurve([0,-1,1,0,0])
sage: E.rank()
0
```

A rank 3 example:

```
sage: E = mwrank_EllipticCurve([0,0,1,-7,6])
sage: EQ = mwrank_MordellWeil(E)
sage: EQ.rank()
0
sage: EQ.regulator()
1.0
```

The preceding output is correct, since we have not yet tried to find any points on the curve either by searching or 2-descent:

```
sage: EQ
Subgroup of Mordell-Weil group: []
```

Now we do a very small search:

We do in fact now have a full Mordell-Weil basis.

regulator ()

Return the regulator of the points in this subgroup of the Mordell-Weil group.

Note: eclib can compute the regulator to arbitrary precision, but the interface currently returns the output as a float.

OUTPUT:

(float) The regulator of the points in this subgroup.

EXAMPLES:

```
sage: E = mwrank_EllipticCurve([0,-1,1,0,0])
sage: E.regulator()
1.0

sage: E = mwrank_EllipticCurve([0,0,1,-7,6])
sage: E.regulator()
0.417143558758384
```

saturate (max_prime=-1, odd_primes_only=False)

Saturate this subgroup of the Mordell-Weil group.

INPUT:

- •max_prime (int, default -1) saturation is performed for all primes up to max_prime . If -1 (the default), an upper bound is computed for the primes at which the subgroup may not be saturated, and this is used; however, if the computed bound is greater than a value set by the eclib library (currently 97) then no saturation will be attempted at primes above this.
- •odd_primes_only (bool, default False) only do saturation at odd primes. (If the points have been found via :meth:two_descent() they should already be 2-saturated.)

OUTPUT:

(3-tuple) (ok, index, unsatlist) where:

- •ok (bool) True if and only if the saturation was provably successful at all primes attempted. If the default was used for max_prime and no warning was output about the computed saturation bound being too high, then True indicates that the subgroup is saturated at *all* primes.
- •index (int) the index of the group generated by the original points in their saturation.
- •unsatlist (list of ints) list of primes at which saturation could not be proved or achieved. Increasing the decimal precision should correct this, since it happens when a linear combination of the points appears to be a multiple of p but cannot be divided by p. (Note that eclib uses floating point methods based on elliptic logarithms to divide points.)

Note: We emphasize that if this function returns True as the first return argument (ok), and if the default was used for the parameter max_prime, then the points in the basis after calling this function are saturated at *all* primes, i.e., saturating at the primes up to max_prime are sufficient to saturate at all primes. Note that the function might not have needed to saturate at all primes up to max_prime. It has worked out what prime you need to saturate up to, and that prime might be smaller than max_prime.

Note: Currently (May 2010), this does not remember the result of calling <code>search()</code>. So calling <code>search()</code> up to height 20 then calling <code>saturate()</code> results in another search up to height 18.

EXAMPLES:

```
sage: E = mwrank_EllipticCurve([0,0,1,-7,6])
sage: EQ = mwrank_MordellWeil(E)
```

We initialise with three points which happen to be 2, 3 and 5 times the generators of this rank 3 curve. To prevent automatic saturation at this stage we set the parameter sat to 0 (which is in fact the default):

Now we saturate at p = 2, and gain index 2:

```
sage: EQ.saturate(2) # points were not 2-saturated
saturating basis...Saturation index bound = 93
WARNING: saturation at primes p > 2 will not be done;
...
Gained index 2
New regulator = 93.857300720636393209
(False, 2, '[]')
sage: EQ
Subgroup of Mordell-Weil group: [[-2:3:1], [2707496766203306:864581029138191:
→2969715140223272], [-13422227300:-49322830557:12167000000]]
sage: EQ.regulator()
93.8572998046875
```

Now we saturate at p = 3, and gain index 3:

Now we saturate at p = 5, and gain index 5:

```
sage: EQ.saturate(5) # points were not 5-saturated
saturating basis...Saturation index bound = 15
WARNING: saturation at primes p > 5 will not be done;
...
Gained index 5
New regulator = 0.417143558758383969818
```

```
(False, 5, '[]')

sage: EQ

Subgroup of Mordell-Weil group: [[-2:3:1], [-14:25:8], [1:-1:1]]

sage: EQ.regulator()

0.4171435534954071
```

Finally we finish the saturation. The output here shows that the points are now provably saturated at all primes:

```
sage: EQ.saturate() # points are now saturated
saturating basis...Saturation index bound = 3
Checking saturation at [ 2 3 ]
Checking 2-saturation
Points were proved 2-saturated (max q used = 11)
Checking 3-saturation
Points were proved 3-saturated (max q used = 13)
done
(True, 1, '[]')
```

Of course, the process () function would have done all this automatically for us:

But we would still need to use the saturate () function to verify that full saturation has been done:

```
sage: EQ.saturate()
saturating basis...Saturation index bound = 3
Checking saturation at [ 2 3 ]
Checking 2-saturation
Points were proved 2-saturated (max q used = 11)
Checking 3-saturation
Points were proved 3-saturated (max q used = 13)
done
(True, 1, '[]')
```

Note the output of the preceding command: it proves that the index of the points in their saturation is at most 3, then proves saturation at 2 and at 3, by reducing the points modulo all primes of good reduction up to 11, respectively 13.

```
search (height_limit=18, verbose=False)
```

Search for new points, and add them to this subgroup of the Mordell-Weil group.

INPUT:

•height limit (float, default: 18) – search up to this logarithmic height.

Note: On 32-bit machines, this *must* be < 21.48 else $\exp(h_{\text{lim}}) > 2^{31}$ and overflows. On 64-bit machines, it must be *at most* 43.668. However, this bound is a logarithmic bound and increasing it by just 1 increases

the running time by (roughly) $\exp(1.5) = 4.5$, so searching up to even 20 takes a very long time.

Note: The search is carried out with a quadratic sieve, using code adapted from a version of Michael Stoll's ratpoints program. It would be preferable to use a newer version of ratpoints.

•verbose (bool, default False) – turn verbose operation on or off.

EXAMPLES:

A rank 3 example, where a very small search is sufficient to find a Mordell-Weil basis:

In the next example, a search bound of 12 is needed to find a non-torsion point:

sage.libs.eclib.interface. set_precision (n)

Set the global NTL real number precision. This has a massive effect on the speed of mwrank calculations. The default (used if this function is not called) is n=50, but it might have to be increased if a computation fails. See also $get_precision()$.

INPUT:

•n (long) – real precision used for floating point computations in the library, in decimal digits.

Warning: This change is global and affects *all* future calls of eclib functions by Sage.

```
sage: mwrank_set_precision(20)
```

CYTHON INTERFACE TO CREMONA'S ECLIB LIBRARY (ALSO KNOWN AS MWRANK)

EXAMPLES:

sage.libs.eclib.mwrank.get_precision()

Returns the working floating point precision of mwrank.

OUTPUT:

(int) The current precision in decimal digits.

EXAMPLES:

```
sage: from sage.libs.eclib.mwrank import get_precision
sage: get_precision()
50
```

sage.libs.eclib.mwrank.initprimes (filename, verb=False)

Initialises mwrank/eclib's internal prime list.

INPUT:

- •filename (string) the name of a file of primes.
- •verb (bool: default False) verbose or not?

```
sage: file = os.path.join(SAGE_TMP, 'PRIMES')
sage: open(file,'w').write(' '.join([str(p) for p in prime_range(10^7,10^7+20)]))
sage: mwrank_initprimes(file, verb=True)
Computed 78519 primes, largest is 1000253
```

```
reading primes from file ...
read extra prime 10000019
finished reading primes from file ...
Extra primes in list: 10000019

sage: mwrank_initprimes("x" + file, True)
Traceback (most recent call last):
...
IOError: No such file or directory: ...
```

sage.libs.eclib.mwrank. $set_precision$ (n)

Sets the working floating point precision of mwrank.

INPUT:

•n (int) – a positive integer: the number of decimal digits.

OUTPUT:

None.

```
sage: from sage.libs.eclib.mwrank import set_precision
sage: set_precision(50)
```

THREE

CREMONA MATRICES

```
{f class} sage.libs.eclib.mat. {f Matrix}
```

Bases: object

A Cremona Matrix.

EXAMPLES:

```
sage: M = CremonaModularSymbols(225)
sage: t = M.hecke_matrix(2)
sage: type(t)
<type 'sage.libs.eclib.mat.Matrix'>
sage: t
61 x 61 Cremona matrix over Rational Field
```

add_scalar (s)

Return new matrix obtained by adding s to each diagonal entry of self.

EXAMPLES:

```
sage: M = CremonaModularSymbols(23, cuspidal=True, sign=1)
sage: t = M.hecke_matrix(2); print(t.str())
[ 0   1]
[ 1 -1]
sage: w = t.add_scalar(3); print(w.str())
[ 3   1]
[ 1   2]
```

charpoly (var='x')

Return the characteristic polynomial of this matrix, viewed as as a matrix over the integers.

ALGORITHM:

Note that currently, this function converts this matrix into a dense matrix over the integers, then calls the charpoly algorithm on that, which I think is LinBox's.

EXAMPLES:

```
sage: M = CremonaModularSymbols(33, cuspidal=True, sign=1)
sage: t = M.hecke_matrix(2)
sage: t.charpoly()
x^3 + 3*x^2 - 4
sage: t.charpoly().factor()
(x - 1) * (x + 2)^2
```

ncols ()

Return the number of columns of this matrix.

EXAMPLES:

```
sage: M = CremonaModularSymbols(1234, sign=1)
sage: t = M.hecke_matrix(3); t.ncols()
156
sage: M.dimension()
156
```

nrows ()

Return the number of rows of this matrix.

EXAMPLES:

```
sage: M = CremonaModularSymbols(19, sign=1)
sage: t = M.hecke_matrix(13); t
2 x 2 Cremona matrix over Rational Field
sage: t.nrows()
2
```

sage_matrix_over_ZZ (sparse=True)

Return corresponding Sage matrix over the integers.

INPUT:

•sparse – (default: True) whether the return matrix has a sparse representation

EXAMPLES

```
sage: M = CremonaModularSymbols(23, cuspidal=True, sign=1)
sage: t = M.hecke_matrix(2)
sage: s = t.sage_matrix_over_ZZ(); s
[ 0  1]
[ 1 -1]
sage: type(s)
<type 'sage.matrix.matrix_integer_sparse.Matrix_integer_sparse'>
sage: s = t.sage_matrix_over_ZZ(sparse=False); s
[ 0  1]
[ 1 -1]
sage: type(s)
<type 'sage.matrix.matrix_integer_dense.Matrix_integer_dense'>
```

str ()

Return full string representation of this matrix, never in compact form.

EXAMPLES:

```
sage: M = CremonaModularSymbols(22, sign=1)
sage: t = M.hecke_matrix(13)
sage: t.str()
'[14  0  0  0  0]\n[-4 12  0  8  4]\n[ 0 -6  4 -6  0]\n[ 4  2  0  6 -4]\n[ 0  __
→0  0  0  14]'
```

```
class sage.libs.eclib.mat. MatrixFactory
```

Bases: object

CHAPTER

FOUR

MODULAR SYMBOLS USING ECLIB NEWFORMS

```
class sage.libs.eclib.newforms. ECModularSymbol
    Bases: object
```

Modular symbol associated with an elliptic curve, using John Cremona's newforms class.

EXAMPLES:

By default, symbols are based at the cusp ∞ , i.e. we evaluate $\{\infty, r\}$:

```
sage: [M(1/i) for i in range(1,11)]
[2/5, -8/5, -3/5, 7/5, 12/5, 12/5, 7/5, -3/5, -8/5, 2/5]
```

We can also switch the base point to the cusp 0:

```
sage: [M(1/i, base_at_infinity=False) for i in range(1,11)]
[0, -2, -1, 1, 2, 2, 1, -1, -2, 0]
```

For the minus symbols this makes no difference since $\{0,\infty\}$ is in the plus space. Note that to evaluate minus symbols the space must be defined with sign 0, which makes both signs available:

If the ECModularSymbol is created with sign 0 then as well as asking for both + and - symbols, we can also obtain both (as a tuple). However it is more work to create the full modular symbol space:

```
[12/5, 0],

[12/5, 0],

[7/5, -1],

[-3/5, -1],

[-8/5, 0],

[2/5, 0]]
```

The curve is automatically converted to its minimal model:

Non-optimal curves are handled correctly in eclib, by comparing the ratios of real and/or imaginary periods:

```
sage: from sage.libs.eclib.newforms import ECModularSymbol
sage: E1 = EllipticCurve('11a1') # optimal
sage: E1.period_lattice().basis()
(1.26920930427955, 0.634604652139777 + 1.45881661693850*I)
sage: M1 = ECModularSymbol(E1,0)
sage: M1(0)
[2/5, 0]
sage: M1(1/3)
[-3/5, 1]
```

One non-optimal curve has real period 1/5 that of the optimal one, so plus symbols scale up by a factor of 5 while minus symbols are unchanged:

```
sage: E2 = EllipticCurve('11a2') # not optimal
sage: E2.period_lattice().basis()
(0.253841860855911, 0.126920930427955 + 1.45881661693850*I)
sage: M2 = ECModularSymbol(E2,0)
sage: M2(0)
[2, 0]
sage: M2(1/3)
[-3, 1]
sage: all((M2(r,1)==5*M1(r,1)) for r in QQ.range_by_height(10))
True
sage: all((M2(r,-1)==M1(r,-1)) for r in QQ.range_by_height(10))
True
```

The other non-optimal curve has real period 5 times that of the optimal one, so plus symbols scale down by a factor of 5; again, minus symbols are unchanged:

```
sage: E3 = EllipticCurve('11a3') # not optimal
sage: E3.period_lattice().basis()
(6.34604652139777, 3.17302326069888 + 1.45881661693850*I)
sage: M3 = ECModularSymbol(E3,0)
sage: M3(0)
[2/25, 0]
sage: M3(1/3)
[-3/25, 1]
sage: all((5*M3(r,1)==M1(r,1)) for r in QQ.range_by_height(10))
True
sage: all((M3(r,-1)==M1(r,-1)) for r in QQ.range_by_height(10))
True
```

CREMONA MODULAR SYMBOLS

```
{\bf class} \; {\tt sage.libs.eclib.homspace.} \; {\tt ModularSymbols}
```

Bases: object

Class of Cremona Modular Symbols of given level and sign (and weight 2).

EXAMPLES:

```
sage: M = CremonaModularSymbols(225)
sage: type(M)
<type 'sage.libs.eclib.homspace.ModularSymbols'>
```

dimension ()

Return the dimension of this modular symbols space.

EXAMPLES:

```
sage: M = CremonaModularSymbols(1234, sign=1)
sage: M.dimension()
156
```

hecke_matrix (p, dual=False, verbose=False)

Return the matrix of the p -th Hecke operator acting on this space of modular symbols.

The result of this command is not cached.

INPUT:

•p – a prime number

•dual – (default: False) whether to compute the Hecke operator acting on the dual space, i.e., the transpose of the Hecke operator

•verbose - (default: False) print verbose output

OUTPUT:

(matrix) If p divides the level, the matrix of the Atkin-Lehner involution W_p at p; otherwise the matrix of the Hecke operator T_p ,

```
sage: M = CremonaModularSymbols(37)
sage: t = M.hecke_matrix(2); t
5 x 5 Cremona matrix over Rational Field
sage: print(t.str())
[ 3 0 0 0 0 0]
[-1 -1 1 1 0]
[ 0 0 -1 0 1]
```

```
[-1 \quad 1 \quad 0 \quad -1 \quad -1]
[ 0 0 1 0 -1 ]
sage: t.charpoly().factor()
(x - 3) * x^2 * (x + 2)^2
sage: print (M.hecke_matrix(2, dual=True).str())
[ 3 -1 0 -1 0]
[ 0 -1 0 1 0]
[ 0 1 -1 0 1 ]
[ 0 1 0 -1 0]
[ 0 0 1 -1 -1 ]
sage: w = M.hecke_matrix(37); w
5 x 5 Cremona matrix over Rational Field
sage: w.charpoly().factor()
(x - 1)^2 * (x + 1)^3
sage: sw = w.sage_matrix_over_ZZ()
sage: st = t.sage_matrix_over_ZZ()
sage: sw^2 == sw.parent()(1)
sage: st*sw == sw*st
True
```

is_cuspidal ()

Return whether or not this space is cuspidal.

EXAMPLES:

```
sage: M = CremonaModularSymbols(1122); M.is_cuspidal()
0
sage: M = CremonaModularSymbols(1122, cuspidal=True); M.is_cuspidal()
1
```

level ()

Return the level of this modular symbols space.

EXAMPLES:

```
sage: M = CremonaModularSymbols(1234, sign=1)
sage: M.level()
1234
```

number_of_cusps ()

Return the number of cusps for $\Gamma_0(N)$, where N is the level.

EXAMPLES:

```
sage: M = CremonaModularSymbols(225)
sage: M.number_of_cusps()
24
```

sign ()

Return the sign of this Cremona modular symbols space. The sign is either 0, +1 or -1.

```
sage: M = CremonaModularSymbols(1122); M
Cremona Modular Symbols space of dimension 433 for Gamma_0(1122) of weight 2_
    with sign 0
sage: M.sign()
0
sage: M = CremonaModularSymbols(1122, sign=-1); M
Cremona Modular Symbols space of dimension 209 for Gamma_0(1122) of weight 2_
    with sign -1
sage: M.sign()
-1
```

Sage Reference Manual: C/C++ Library Interfaces, Release 7.6							

CREMONA MODULAR SYMBOLS

```
sage.libs.eclib.constructor. CremonaModularSymbols ( level, sign=0, cuspidal=False, verbose=0)
```

Return the space of Cremona modular symbols with given level, sign, etc.

INPUT:

- •level an integer >= 2 (at least 2, not just positive!)
- •sign an integer either 0 (the default) or 1 or -1.
- •cuspidal (default: False); if True, compute only the cuspidal subspace
- •verbose (default: False): if True, print verbose information while creating space

EXAMPLES:

When run interactively, the following command will display verbose output:

```
sage: M = CremonaModularSymbols(43, verbose=1)
After 2-term relations, ngens = 22
        = 22
ngens
maxnumrel = 32
relation matrix has = 704 entries...
Finished 3-term relations: numrel = 16 ( maxnumrel = 32)
relmat has 42 nonzero entries (density = 0.0596591)
Computing kernel...
time to compute kernel = (... seconds)
rk = 7
Number of cusps is 2
ncusps = 2
About to compute matrix of delta
delta matrix done: size 2x7.
About to compute kernel of delta
done
```

The input must be valid or a ValueError is raised:

```
sage: M = CremonaModularSymbols(-1)
Traceback (most recent call last):
...
ValueError: the level (= -1) must be at least 2
sage: M = CremonaModularSymbols(0)
Traceback (most recent call last):
...
ValueError: the level (= 0) must be at least 2
```

The sign can only be 0 or 1 or -1:

```
sage: M = CremonaModularSymbols(10, sign = -2)
Traceback (most recent call last):
...
ValueError: sign (= -2) is not supported; use 0, +1 or -1
```

We do allow -1 as a sign (see trac ticket #9476):

RUBINSTEIN'S LCALC LIBRARY

This is a wrapper around Michael Rubinstein's lcalc. See http://oto.math.uwaterloo.ca/~mrubinst/L_function_public/CODE/.

AUTHORS:

- Rishikesh (2010): added compute_rank() and hardy_z_function()
- Yann Laigle-Chapuy (2009): refactored
- Rishikesh (2009): initial version

```
class sage.libs.lcalc.lcalc_Lfunction. Lfunction
     Bases: object
```

Initialization of L-function objects. See derived class for details, this class is not supposed to be instantiated directly.

EXAMPLES:

```
sage: from sage.libs.lcalc.lcalc_Lfunction import *
sage: Lfunction_from_character(DirichletGroup(5)[1])
L-function with complex Dirichlet coefficients
```

compute rank ()

Computes the analytic rank (the order of vanishing at the center) of of the L-function

EXAMPLES:

```
sage: chi=DirichletGroup(5)[2] #This is a quadratic character
sage: from sage.libs.lcalc.lcalc_Lfunction import *
sage: L=Lfunction_from_character(chi, type="int")
sage: L.compute_rank()
0
sage: E=EllipticCurve([-82,0])
sage: L=Lfunction_from_elliptic_curve(E, number_of_coeffs=40000)
sage: L.compute_rank()
3
```

find_zeros (T1, T2, stepsize)

Finds zeros on critical line between T1 and T2 using step size of stepsize. This function might miss zeros if step size is too large. This function computes the zeros of the L-function by using change in signs of areal valued function whose zeros coincide with the zeros of L-function.

Use find_zeros_via_N() for slower but more rigorous computation.

INPUT:

•T1 – a real number giving the lower bound

- •T2 a real number giving the upper bound
- •stepsize step size to be used for the zero search

OUTPUT:

list – A list of the imaginary parts of the zeros which were found.

EXAMPLES:

```
sage: from sage.libs.lcalc.lcalc_Lfunction import *
sage: chi=DirichletGroup(5)[2] #This is a quadratic character
sage: L=Lfunction_from_character(chi, type="int")
sage: L.find_zeros(5,15,.1)
[6.64845334472..., 9.83144443288..., 11.9588456260...]

sage: L=Lfunction_from_character(chi, type="double")
sage: L.find_zeros(1,15,.1)
[6.64845334472..., 9.83144443288..., 11.9588456260...]

sage: chi=DirichletGroup(5)[1]
sage: L=Lfunction_from_character(chi, type="complex")
sage: L.find_zeros(-8,8,.1)
[-4.13290370521..., 6.18357819545...]

sage: L=Lfunction_Zeta()
sage: L.find_zeros(10,29.1,.1)
[14.1347251417..., 21.0220396387..., 25.0108575801...]
```

Finds count number of zeros with positive imaginary part starting at real axis. This function also verifies that all the zeros have been found.

INPUT:

- •count number of zeros to be found
- •do negative (default: False) False to ignore zeros below the real axis.
- •max_refine when some zeros are found to be missing, the step size used to find zeros is refined. max_refine gives an upper limit on when lcalc should give up. Use default value unless you know what you are doing.
- •rank integer (default: -1) analytic rank of the L-function. If -1 is passed, then we attempt to compute it. (Use default if in doubt)
- •test_explicit_formula integer (default: 0) If nonzero, test the explicit fomula for additional confidence that all the zeros have been found and are accurate. This is still being tested, so using the default is recommended.

OUTPUT:

list – A list of the imaginary parts of the zeros that have been found

```
sage: from sage.libs.lcalc.lcalc_Lfunction import *
sage: chi=DirichletGroup(5)[2] #This is a quadratic character
sage: L=Lfunction_from_character(chi, type="int")
sage: L.find_zeros_via_N(3)
[6.64845334472..., 9.83144443288..., 11.9588456260...]
```

```
sage: L=Lfunction_from_character(chi, type="double")
sage: L.find_zeros_via_N(3)
[6.64845334472..., 9.83144443288..., 11.9588456260...]

sage: chi=DirichletGroup(5)[1]
sage: L=Lfunction_from_character(chi, type="complex")
sage: L.find_zeros_via_N(3)
[6.18357819545..., 8.45722917442..., 12.6749464170...]

sage: L=Lfunction_Zeta()
sage: L.find_zeros_via_N(3)
[14.1347251417..., 21.0220396387..., 25.0108575801...]
```

hardy_z_function (s)

Computes the Hardy Z-function of the L-function at s

INPUT:

•s - a complex number with imaginary part between -0.5 and 0.5

EXAMPLES:

```
sage: chi=DirichletGroup(5)[2] #This is a quadratic character
sage: from sage.libs.lcalc.lcalc_Lfunction import >
sage: L=Lfunction_from_character(chi, type="int")
sage: L.hardy_z_function(0)
0.231750947504...
sage: L.hardy_z_function(.5).imag().abs() < 1.0e-16</pre>
True
sage: L.hardy_z_function(.4+.3*I)
0.2166144222685... - 0.00408187127850...*I
sage: chi=DirichletGroup(5)[1]
sage: L=Lfunction_from_character(chi,type="complex")
sage: L.hardy_z_function(0)
0.7939675904771...
sage: L.hardy_z_function(.5).imag().abs() < 1.0e-16</pre>
True
sage: E=EllipticCurve([-82,0])
sage: L=Lfunction_from_elliptic_curve(E, number_of_coeffs=40000)
sage: L.hardy_z_function(2.1)
-0.00643179176869...
sage: L.hardy_z_function(2.1).imag().abs() < 1.0e-16</pre>
True
```

value (s, derivative=0)

Computes the value of the L-function at s

INPUT:

- •s a complex number
- •derivative integer (default: 0) the derivative to be evaluated
- •rotate (default: False) If True, this returns the value of the Hardy Z-function (sometimes called the Riemann-Siegel Z-function or the Siegel Z-function).

```
sage: chi=DirichletGroup(5)[2] #This is a quadratic character
sage: from sage.libs.lcalc.lcalc_Lfunction import *
sage: L=Lfunction_from_character(chi, type="int")
```

```
sage: L.value(.5) # abs tol 3e-15
0.231750947504016 + 5.75329642226136e-18*I
sage: L.value(.2+.4*I)
0.102558603193... + 0.190840777924...*I
sage: L=Lfunction_from_character(chi, type="double")
sage: L.value(.6) # abs tol 3e-15
0.274633355856345 + 6.59869267328199e-18*I
sage: L.value(.6+I)
0.362258705721... + 0.433888250620...*I
sage: chi=DirichletGroup(5)[1]
sage: L=Lfunction_from_character(chi, type="complex")
sage: L.value(.5)
0.763747880117... + 0.216964767518...*I
sage: L.value(.6+5*I)
0.702723260619... - 1.10178575243...*I
sage: L=Lfunction_Zeta()
sage: L.value(.5)
-1.46035450880...
sage: L.value(.4+.5*I)
-0.450728958517... - 0.780511403019...*I
```

class sage.libs.lcalc.lcalc_Lfunction. Lfunction_C
 Bases: sage.libs.lcalc.lcalc_Lfunction.Lfunction

The Lfunction_C class is used to represent L-functions with complex Dirichlet Coefficients. We assume that L-functions satisfy the following functional equation.

$$\Lambda(s) = \omega Q^s \overline{\Lambda(1-\bar{s})}$$

where

$$\Lambda(s) = Q^s \left(\prod_{j=1}^a \Gamma(\kappa_j s + \gamma_j) \right) L(s)$$

See (23) in http://arxiv.org/abs/math/0412181

INPUT:

- •what_type_L integer, this should be set to 1 if the coefficients are periodic and 0 otherwise.
- •dirichlet_coefficient List of dirichlet coefficients of the L-function. Only first M coefficients are needed if they are periodic.
- •period If the coefficients are periodic, this should be the period of the coefficients.
- •Q See above
- •OMEGA See above
- •kappa List of the values of κ_i in the functional equation
- ulletgamma List of the values of γ_j in the functional equation
- •pole List of the poles of L-function
- •residue List of the residues of the L-function

NOTES:

If an L-function satisfies $\Lambda(s) = \omega Q^s \Lambda(k-s)$, by replacing s by s + (k-1)/2, one can get it in the form we need.

class sage.libs.lcalc.lcalc_Lfunction. Lfunction_D

Bases: sage.libs.lcalc.lcalc_Lfunction.Lfunction

The Lfunction_D class is used to represent L-functions with real Dirichlet coefficients. We assume that L-functions satisfy the following functional equation.

$$\Lambda(s) = \omega Q^s \overline{\Lambda(1-\bar{s})}$$

where

$$\Lambda(s) = Q^s \left(\prod_{j=1}^a \Gamma(\kappa_j s + \gamma_j) \right) L(s)$$

See (23) in http://arxiv.org/abs/math/0412181

INPUT:

- •what_type_L integer, this should be set to 1 if the coefficients are periodic and 0 otherwise.
- •dirichlet_coefficient List of dirichlet coefficients of the L-function. Only first M coefficients are needed if they are periodic.
- •period If the coefficients are periodic, this should be the period of the coefficients.
- •Q See above
- •OMEGA See above
- •kappa List of the values of κ_j in the functional equation
- ulletgamma List of the values of γ_j in the functional equation
- •pole List of the poles of L-function
- •residue List of the residues of the L-function

NOTES:

If an L-function satisfies $\Lambda(s) = \omega Q^s \Lambda(k-s)$, by replacing s by s+(k-1)/2, one can get it in the form we need.

class sage.libs.lcalc.lcalc_Lfunction. Lfunction_I

Bases: sage.libs.lcalc.lcalc_Lfunction.Lfunction

The Lfunction_I class is used to represent L-functions with integer Dirichlet Coefficients. We assume that L-functions satisfy the following functional equation.

$$\Lambda(s) = \omega Q^s \overline{\Lambda(1-\bar{s})}$$

where

$$\Lambda(s) = Q^s \left(\prod_{j=1}^a \Gamma(\kappa_j s + \gamma_j) \right) L(s)$$

See (23) in http://arxiv.org/abs/math/0412181

INPUT:

•what_type_L - integer, this should be set to 1 if the coefficients are periodic and 0 otherwise.

- \bullet dirichlet_coefficient List of dirichlet coefficients of the L-function. Only first M coefficients are needed if they are periodic.
- •period If the coefficients are periodic, this should be the period of the coefficients.
- •Q See above
- •OMEGA See above
- •kappa List of the values of κ_i in the functional equation
- ulletgamma List of the values of γ_j in the functional equation
- •pole List of the poles of L-function
- •residue List of the residues of the L-function

NOTES:

If an L-function satisfies $\Lambda(s) = \omega Q^s \Lambda(k-s)$, by replacing s by s+(k-1)/2, one can get it in the form we need.

```
class sage.libs.lcalc.lcalc_Lfunction. Lfunction_Zeta
    Bases: sage.libs.lcalc.lcalc_Lfunction.Lfunction
```

The Lfunction_Zeta class is used to generate the Riemann zeta function.

sage.libs.lcalc.lcalc_Lfunction. **Lfunction_from_character** (*chi*, *type='complex'*) Given a primitive Dirichlet character, this function returns an lcalc L-function object for the L-function of the character.

INPUT:

- •chi A Dirichlet character
- •use_type string (default: "complex") type used for the Dirichlet coefficients. This can be "int", "double" or "complex".

OUTPUT:

L-function object for chi.

EXAMPLES:

```
sage: from sage.libs.lcalc.lcalc_Lfunction import Lfunction_from_character
sage: Lfunction_from_character(DirichletGroup(5)[1])
L-function with complex Dirichlet coefficients
sage: Lfunction_from_character(DirichletGroup(5)[2], type="int")
L-function with integer Dirichlet coefficients
sage: Lfunction_from_character(DirichletGroup(5)[2], type="double")
L-function with real Dirichlet coefficients
sage: Lfunction_from_character(DirichletGroup(5)[1], type="int")
Traceback (most recent call last):
...
ValueError: For non quadratic characters you must use type="complex"
```

```
sage.libs.lcalc.lcalc_Lfunction. Lfunction_from_elliptic_curve (E, num-ber_of_coeffs=10000)
```

Given an elliptic curve E, return an L-function object for the function L(s, E).

INPUT:

•E - An elliptic curve

•number_of_coeffs - integer (default: 10000) The number of coefficients to be used when constructing the L-function object. Right now this is fixed at object creation time, and is not automatically set intelligently.

OUTPUT:

L-function object for L(s, E).

```
sage: from sage.libs.lcalc.lcalc_Lfunction import Lfunction_from_elliptic_curve
sage: L = Lfunction_from_elliptic_curve(EllipticCurve('37'))
sage: L
L-function with real Dirichlet coefficients
sage: L.value(0.5).abs() < 1e-15 # "noisy" zero on some platforms (see #9615)
True
sage: L.value(0.5, derivative=1)
0.305999...</pre>
```

CHAPTER

EIGHT

INTERFACE BETWEEN SAGE AND PARI

8.1 Guide to real precision in the PARI interface

In the PARI interface, "real precision" refers to the precision of real numbers, so it is the floating-point precision. This is a non-trivial issue, since there are various interfaces for different things.

8.1.1 Internal representation and conversion between Sage and PARI

Real numbers in PARI have a precision associated to them, which is always a multiple of the CPU wordsize. So, it is a multiple of 32 of 64 bits. When converting from Sage to PARI, the precision is rounded up to the nearest multiple of the wordsize:

```
sage: x = 1.0
sage: x.precision()
53
sage: pari(x)
1.00000000000000
sage: pari(x).bitprecision()
64
```

With a higher precision:

```
sage: x = RealField(100).pi()
sage: x.precision()
100
sage: pari(x).bitprecision()
128
```

When converting back to Sage, the precision from PARI is taken:

```
sage: x = RealField(100).pi()
sage: y = pari(x).sage()
sage: y
3.1415926535897932384626433832793333156
sage: parent(y)
Real Field with 128 bits of precision
```

So pari(x).sage() is definitely not equal to x since it has 28 bogus bits.

Therefore, some care must be taken when juggling reals back and forth between Sage and PARI. The correct way of avoiding this is to convert pari (x) .sage() back into a domain with the right precision. This has to be done by the user (or by Sage functions that use PARI library functions). For instance, if we want to use the PARI library to compute sgrt (pi) with a precision of 100 bits:

```
sage: R = RealField(100)
sage: s = R(pi); s
3.1415926535897932384626433833
sage: p = pari(s).sqrt()
sage: x = p.sage(); x  # wow, more digits than I expected!
1.7724538509055160272981674833410973484
sage: x.prec()  # has precision 'improved' from 100 to 128?
128
sage: x == RealField(128)(pi).sqrt() # sadly, no!
False
sage: R(x)  # x should be brought back to precision 100
1.7724538509055160272981674833
sage: R(x)  == s.sqrt()
True
```

8.1.2 Output precision for printing

Even though PARI reals have a precision, not all significant bits are printed by default. The maximum number of digits when printing a PARI real can be set using the methods Pari.set_real_precision_bits() or Pari.set_real_precision().

We create a very precise approximation of pi and see how it is printed in PARI:

```
sage: pi = pari(RealField(1000).pi())
```

The default precision is 15 digits:

```
sage: pi
3.14159265358979
```

With a different precision:

```
sage: _ = pari.set_real_precision(50)
sage: pi
3.1415926535897932384626433832795028841971693993751
```

Back to the default:

```
sage: _ = pari.set_real_precision(15)
sage: pi
3.14159265358979
```

8.1.3 Input precision for function calls

When we talk about precision for PARI functions, we need to distinguish three kinds of calls:

- 1. Using the string interface, for example pari ("sin(1)").
- 2. Using the library interface with exact inputs, for example pari(1).sin().
- 3. Using the library interface with inexact inputs, for example pari(1.0).sin().

In the first case, the relevant precision is the one set by the methods Pari.set_real_precision_bits() or Pari.set_real_precision():

```
sage: pari.set_real_precision_bits(150)
sage: pari("sin(1)")
0.841470984807896506652502321630298999622563061
sage: pari.set_real_precision_bits(53)
sage: pari("sin(1)")
0.841470984807897
```

In the second case, the precision can be given as the argument precision in the function call, with a default of 53 bits. The real precision set by Pari.set_real_precision_bits() or Pari.set_real_precision() is irrelevant.

In these examples, we convert to Sage to ensure that PARI's real precision is not used when printing the numbers. As explained before, this artificially increases the precision to a multiple of the wordsize.

```
sage: s = pari(1).sin(precision=180).sage(); print(s); print(parent(s))
0.841470984807896506652502321630298999622563060798371065673
Real Field with 192 bits of precision
sage: s = pari(1).sin(precision=40).sage(); print(s); print(parent(s))
0.841470984807896507
Real Field with 64 bits of precision
sage: s = pari(1).sin().sage(); print(s); print(parent(s))
0.841470984807896507
Real Field with 64 bits of precision
```

In the third case, the precision is determined only by the inexact inputs and the precision argument is ignored:

```
sage: pari(1.0).sin(precision=180).sage()
0.841470984807896507
sage: pari(1.0).sin(precision=40).sage()
0.841470984807896507
sage: pari(RealField(100).one()).sin().sage()
0.84147098480789650665250232163029899962
```

8.1.4 Elliptic curve functions

An elliptic curve given with exact a-invariants is considered an exact object. Therefore, you should set the precision for each method call individually:

```
sage: e = pari([0,0,0,-82,0]).ellinit()
sage: etal = e.elleta(precision=100)[0]
sage: etal.sage()
3.6054636014326520859158205642077267748
sage: etal = e.elleta(precision=180)[0]
sage: etal.sage()
3.60546360143265208591582056420772677481026899659802474544
```

CONVERT PARI OBJECTS TO SAGE TYPES

```
sage.libs.pari.convert_sage.gen_to_sage (z, locals=None)
     Convert a PARI gen to a Sage/Python object.
     INPUT:
         •z - PARI gen
         •locals - optional dictionary used in fallback cases that involve sage_eval()
     OUTPUT:
     One of the following depending on the PARI type of z
         •a Integer if z is an integer (type t_INT)
         •a Rational if z is a rational (type t_FRAC)
         •a RealNumber if z is a real number (type t_REAL). The precision will be equivalent.
         •a NumberFieldElement_quadratic or a ComplexNumber if z is a complex number (type
          t_COMPLEX). The former is used when the real and imaginary parts are integers or rationals and the
          latter when they are floation point numbers. In that case The precision will be the maximal precision of
          the real and imaginary parts.
         •a Python list if z is a vector or a list (type t_VEC, t_COL)
         •a Python string if z is a string (type t_STR)
         •a Python list of Python integers if z is a small vector (type t VECSMALL)
         •a matrix if z is a matrix (type t_MAT)
         •a padic element (type t_PADIC)
         •a Infinity if z is an infinity (type t_INF)
     EXAMPLES:
     sage: from sage.libs.pari.convert_sage import gen_to_sage
```

Converting an integer:

```
sage: z = pari('12'); z

12
sage: z.type()
't_INT'
sage: a = gen_to_sage(z); a

12
sage: a.parent()
Integer Ring
```

```
sage: gen_to_sage(pari('7^42'))
311973482284542371301330321821976049
```

Converting a rational number:

```
sage: z = pari('389/17'); z
389/17
sage: z.type()
't_FRAC'
sage: a = gen_to_sage(z); a
389/17
sage: a.parent()
Rational Field

sage: gen_to_sage(pari('5^30 / 3^50'))
931322574615478515625/717897987691852588770249
```

Converting a real number:

For complex numbers, the parent depends on the PARI type:

```
sage: z = pari('(3+I)'); z
3 + I
sage: z.type()
't_COMPLEX'
sage: a = gen_to_sage(z); a
i + 3
sage: a.parent()
Number Field in i with defining polynomial x^2 + 1
sage: z = pari('(3+I)/2'); z
3/2 + 1/2 * I
sage: a = gen_to_sage(z); a
1/2 * i + 3/2
sage: a.parent()
Number Field in i with defining polynomial x^2 + 1
sage: z = pari('1.0 + 2.0*I'); z
1.00000000000000 + 2.0000000000000*I
sage: a = gen_to_sage(z); a
sage: a.parent()
```

```
Complex Field with 64 bits of precision
```

Converting polynomials:

```
sage: f = pari('(2/3) *x^3 + x - 5/7 + y')
sage: f.type()
't POL'
sage: R.<x,y> = QQ[]
sage: gen_to_sage(f, {'x': x, 'y': y})
2/3*x^3 + x + y - 5/7
sage: parent(gen_to_sage(f, {'x': x, 'y': y}))
Multivariate Polynomial Ring in x, y over Rational Field
sage: x,y = SR.var('x,y')
sage: gen_to_sage(f, {'x': x, 'y': y})
2/3*x^3 + x + y - 5/7
sage: parent(gen_to_sage(f, {'x': x, 'y': y}))
Symbolic Ring
sage: gen_to_sage(f)
Traceback (most recent call last):
NameError: name 'x' is not defined
```

Converting vectors:

```
sage: z1 = pari('[-3, 2.1, 1+I]'); z1
[-3, 2.1000000000000, 1 + I]
sage: z2 = pari('[1.0*I, [1,2]]~'); z2
[1.00000000000000*I, [1, 2]]~
sage: z1.type(), z2.type()
('t_VEC', 't_COL')
sage: a1 = gen_to_sage(z1)
sage: a2 = gen_to_sage(z2)
sage: type(a1), type(a2)
(<... 'list'>, <... 'list'>)
sage: [parent(b) for b in al]
[Integer Ring,
Real Field with 64 bits of precision,
Number Field in i with defining polynomial x^2 + 1
sage: [parent(b) for b in a2]
[Complex Field with 64 bits of precision, <... 'list'>]
sage: z = pari('Vecsmall([1,2,3,4])')
sage: z.type()
't_VECSMALL'
sage: a = gen_to_sage(z); a
[1, 2, 3, 4]
sage: type(a)
<... 'list'>
sage: [parent(b) for b in a]
[<... 'int'>, <... 'int'>, <... 'int'>]
```

Matrices:

```
sage: z = pari('[1,2;3,4]')
sage: z.type()
```

```
't_MAT'
sage: a = gen_to_sage(z); a
[1 2]
[3 4]
sage: a.parent()
Full MatrixSpace of 2 by 2 dense matrices over Integer Ring
```

Conversion of p-adics:

```
sage: z = pari('569 + O(7^8)'); z
2 + 4*7 + 4*7^2 + 7^3 + O(7^8)
sage: a = gen_to_sage(z); a
2 + 4*7 + 4*7^2 + 7^3 + O(7^8)
sage: a.parent()
7-adic Field with capped relative precision 8
```

Conversion of infinities:

```
sage: gen_to_sage(pari('oo'))
+Infinity
sage: gen_to_sage(pari('-oo'))
-Infinity
```

RING OF PARI OBJECTS

AUTHORS:

- William Stein (2004): Initial version.
- Simon King (2011-08-24): Use UniqueRepresentation, element_class and proper initialisation of elements.

```
class sage.rings.pari_ring. Pari (x, parent=None)
    Bases: sage.structure.element.RingElement
    Element of Pari pseudo-ring.

class sage.rings.pari_ring. PariRing
    Bases: sage.misc.fast_methods.Singleton, sage.rings.ring.Ring

EXAMPLES: sage: R = PariRing(); R Pseudoring of all PARI objects. sage: loads(R.dumps()) is R True

Element
    alias of Pari
    characteristic ()
    is_field (proof=True)
    random_element (x=None, y=None, distribution=None)
        Return a random integer in Pari.
        NOTE:
```

The given arguments are passed to ${\tt ZZ.random_element}$ (...) .

INPUT:

- •x, y optional integers, that are lower and upper bound for the result. If only x is provided, then the result is between 0 and x 1, inclusive. If both are provided, then the result is between x and y 1, inclusive.
- •distribution optional string, so that ZZ can make sense of it as a probability distribution.

```
sage: R = PariRing()
sage: R.random_element()
-8
sage: R.random_element(5,13)
12
sage: [R.random_element(distribution="1/n") for _ in range(10)]
[0, 1, -1, 2, 1, -95, -1, -2, -12, 0]
```

```
zeta ()
```

Return -1.

```
sage: R = PariRing()
sage: R.zeta()
-1
```

HYPERELLIPTIC CURVE POINT FINDING, VIA RATPOINTS.

Access the ratpoints library to find points on the hyperelliptic curve:

$$y^2 = a_n x^n + \dots + a_1 x + a_0.$$

INPUT:

- •coeffs list of integer coefficients a_0 , a_1 , ..., a_n
- •H the bound for the denominator and the absolute value of the numerator of the x-coordinate
- •verbose if True, ratpoints will print comments about its progress
- •max maximum number of points to find (if 0, find all of them)

OUTPUT:

The points output by this program are points in (1, ceil(n/2), 1)-weighted projective space. If n is even, then the associated homogeneous equation is $y^2 = a_n x^n + \cdots + a_1 x z^{n-1} + a_0 z^n$ while if n is odd, it is $y^2 = a_n x^n z + \cdots + a_1 x z^n + a_0 z^{n+1}$.

```
sage: from sage.libs.ratpoints import ratpoints
sage: for x,y,z in ratpoints([1..6], 200):
          print(-1*y^2 + 1*z^6 + 2*x*z^5 + 3*x^2*z^4 + 4*x^3*z^3 + 5*x^4*z^2 + ...
. . . . :
\leftrightarrow 6*x<sup>5</sup>*z)
0
0
0
0
0
sage: for x,y,z in ratpoints([1..5], 200):
         print (-1*y^2 + 1*z^4 + 2*x*z^3 + 3*x^2*z^2 + 4*x^3*z + 5*x^4)
0
0
0
0
0
0
0
sage: for x,y,z in ratpoints([1..200], 1000):
         print("{} {} {} ".format(x,y,z))
```

The denominator of x can be restricted, for example to find integral points:

```
sage: from sage.libs.ratpoints import ratpoints
sage: coeffs = [400, -112, 0, 1]
sage: ratpoints(coeffs, 10^6, max_x_denom=1, intervals=[[-10,0],[1000,2000]])
[(1, 0, 0), (-8, 28, 1), (-8, -28, 1), (-7, 29, 1), (-7, -29, 1),
(-4, 28, 1), (-4, -28, 1), (0, 20, 1), (0, -20, 1), (1368, 50596, 1),
(1368, -50596, 1), (1624, 65444, 1), (1624, -65444, 1)]
sage: ratpoints(coeffs, 1000, min_x_denom=100, max_x_denom=200)
[(1, 0, 0),
(-656, 426316, 121),
(-656, -426316, 121),
(452, 85052, 121),
(452, -85052, 121),
(988, 80036, 121),
(988, -80036, 121),
(-556, 773188, 169),
(-556, -773188, 169),
(264, 432068, 169),
(264, -432068, 169)
```

Finding the integral points on the compact component of an elliptic curve:

```
sage: E = EllipticCurve([0,1,0,-35220,-1346400])
sage: e1, e2, e3 = E.division_polynomial(2).roots(multiplicities=False)
sage: coeffs = [E.a6(),E.a4(),E.a2(),1]
sage: ratpoints(coeffs, 1000, max_x_denom=1, intervals=[[e3,e2]])
[(1, 0, 0),
(-165, 0, 1),
(-162, 366, 1),
(-162, -366, 1),
(-120, 1080, 1),
(-120, -1080, 1),
(-90, 1050, 1),
(-90, -1050, 1),
(-85, 1020, 1),
(-85, -1020, 1),
(-42, 246, 1),
(-42, -246, 1),
(-40, 0, 1)
```

CHAPTER

TWELVE

LIBSINGULAR: FUNCTIONS

Sage implements a C wrapper around the Singular interpreter which allows to call any function directly from Sage without string parsing or interprocess communication overhead. Users who do not want to call Singular functions directly, usually do not have to worry about this interface, since it is handled by higher level functions in Sage.

AUTHORS:

- Michael Brickenstein (2009-07): initial implementation, overall design
- Martin Albrecht (2009-07): clean up, enhancements, etc.
- Michael Brickenstein (2009-10): extension to more Singular types
- Martin Albrecht (2010-01): clean up, support for attributes
- Simon King (2011-04): include the documentation provided by Singular as a code block.
- Burcin Erocal, Michael Brickenstein, Oleksandr Motsak, Alexander Dreyer, Simon King (2011-09) plural support

EXAMPLES:

The direct approach for loading a Singular function is to call the function $singular_function()$ with the function name as parameter:

```
sage: from sage.libs.singular.function import singular_function
sage: P.<a,b,c,d> = PolynomialRing(GF(7))
sage: std = singular_function('std')
sage: I = sage.rings.ideal.Cyclic(P)
sage: std(I)
[a + b + c + d,
b^2 + 2*b*d + d^2,
b*c^2 + c^2*d - b*d^2 - d^3,
b*c*d^2 + c^2*d^2 - b*d^3 + c*d^3 - d^4 - 1,
b*d^4 + d^5 - b - d,
c^3*d^2 + c^2*d^3 - c - d,
c^2*d^4 + b*c - b*d + c*d - 2*d^2]
```

If a Singular library needs to be loaded before a certain function is available, use the <code>lib()</code> function as shown below:

```
sage: from sage.libs.singular.function import singular_function, lib as singular_lib
sage: primdecSY = singular_function('primdecSY')
Traceback (most recent call last):
...
NameError: Singular library function 'primdecSY' is not defined
sage: singular_lib('primdec.lib')
sage: primdecSY = singular_function('primdecSY')
```

There is also a short-hand notation for the above:

```
sage: import sage.libs.singular.function_factory
sage: primdecSY = sage.libs.singular.function_factory.ff.primdec__lib.primdecSY
```

The above line will load "primdec.lib" first and then load the function primdecSY.

A call handler is an abstraction which hides the details of the implementation differences between kernel and library functions.

```
class sage.libs.singular.function. Converter
    Bases: sage.structure.sage_object.SageObject
```

A Converter interfaces between Sage objects and Singular interpreter objects.

```
ring()
```

Return the ring in which the arguments of this list live.

EXAMPLES:

```
sage: from sage.libs.singular.function import Converter
sage: P.<a,b,c> = PolynomialRing(GF(127))
sage: Converter([a,b,c],ring=P).ring()
Multivariate Polynomial Ring in a, b, c over Finite Field of size 127
```

```
class sage.libs.singular.function. KernelCallHandler
```

 $Bases: \ \textit{sage.libs.singular.function.BaseCallHandler}$

A call handler is an abstraction which hides the details of the implementation differences between kernel and library functions.

This class implements calling a kernel function.

Note: Do not construct this class directly, use <code>singular_function()</code> instead.

```
class sage.libs.singular.function. LibraryCallHandler
    Bases: sage.libs.singular.function.BaseCallHandler
```

A call handler is an abstraction which hides the details of the implementation differences between kernel and library functions.

This class implements calling a library function.

Note: Do not construct this class directly, use <code>singular_function()</code> instead.

```
class sage.libs.singular.function. Resolution
    Bases: object
```

A simple wrapper around Singular's resolutions.

```
class sage.libs.singular.function. RingWrap
    Bases: object
```

A simple wrapper around Singular's rings.

```
characteristic ()
```

Get characteristic.

EXAMPLES:

```
sage: from sage.libs.singular.function import singular_function
sage: P.<x,y,z> = PolynomialRing(QQ)
sage: ringlist = singular_function("ringlist")
sage: l = ringlist(P)
sage: ring = singular_function("ring")
sage: ring(l, ring=P).characteristic()
0
```

is commutative ()

Determine whether a given ring is commutative.

EXAMPLES:

```
sage: from sage.libs.singular.function import singular_function
sage: P.<x,y,z> = PolynomialRing(QQ)
sage: ringlist = singular_function("ringlist")
sage: l = ringlist(P)
sage: ring = singular_function("ring")
sage: ring(l, ring=P).is_commutative()
True
```

ngens ()

Get number of generators.

EXAMPLES:

```
sage: from sage.libs.singular.function import singular_function
sage: P.<x,y,z> = PolynomialRing(QQ)
sage: ringlist = singular_function("ringlist")
sage: l = ringlist(P)
sage: ring = singular_function("ring")
sage: ring(l, ring=P).ngens()
3
```

npars ()

Get number of parameters.

EXAMPLES:

```
sage: from sage.libs.singular.function import singular_function
sage: P.<x,y,z> = PolynomialRing(QQ)
sage: ringlist = singular_function("ringlist")
sage: l = ringlist(P)
sage: ring = singular_function("ring")
sage: ring(l, ring=P).npars()
0
```

ordering_string()

Get Singular string defining monomial ordering.

```
sage: from sage.libs.singular.function import singular_function
sage: P.<x,y,z> = PolynomialRing(QQ)
sage: ringlist = singular_function("ringlist")
sage: l = ringlist(P)
sage: ring = singular_function("ring")
```

```
sage: ring(1, ring=P).ordering_string()
'dp(3),C'
```

par_names ()

Get parameter names.

EXAMPLES:

```
sage: from sage.libs.singular.function import singular_function
sage: P.<x,y,z> = PolynomialRing(QQ)
sage: ringlist = singular_function("ringlist")
sage: l = ringlist(P)
sage: ring = singular_function("ring")
sage: ring(l, ring=P).par_names()
[]
```

var_names ()

Get names of variables.

EXAMPLES:

```
sage: from sage.libs.singular.function import singular_function
sage: P.<x,y,z> = PolynomialRing(QQ)
sage: ringlist = singular_function("ringlist")
sage: l = ringlist(P)
sage: ring = singular_function("ring")
sage: ring(l, ring=P).var_names()
['x', 'y', 'z']
```

class sage.libs.singular.function. SingularFunction

Bases: sage.structure.sage_object.SageObject

The base class for Singular functions either from the kernel or from the library.

class sage.libs.singular.function. SingularKernelFunction

Bases: sage.libs.singular.function.SingularFunction

EXAMPLES:

```
sage: from sage.libs.singular.function import SingularKernelFunction
sage: R.<x,y> = PolynomialRing(QQ, order='lex')
sage: I = R.ideal(x, x+1)
sage: f = SingularKernelFunction("std")
sage: f(I)
[1]
```

class sage.libs.singular.function. SingularLibraryFunction

Bases: sage.libs.singular.function.SingularFunction

EXAMPLES:

```
sage: from sage.libs.singular.function import SingularLibraryFunction
sage: R.<x,y> = PolynomialRing(QQ, order='lex')
sage: I = R.ideal(x, x+1)
sage: f = SingularLibraryFunction("groebner")
sage: f(I)
[1]
```

sage.libs.singular.function.all_singular_poly_wrapper (s)

Tests for a sequence s, whether it consists of singular polynomials.

EXAMPLES:

```
sage: from sage.libs.singular.function import all_singular_poly_wrapper
sage: P.<x,y,z> = QQ[]
sage: all_singular_poly_wrapper([x+1, y])
True
sage: all_singular_poly_wrapper([x+1, y, 1])
False
```

sage.libs.singular.function.all_vectors (s)

Checks if a sequence s consists of free module elements over a singular ring.

EXAMPLES:

```
sage: from sage.libs.singular.function import all_vectors
sage: P.<x,y,z> = QQ[]
sage: M = P**2
sage: all_vectors([x])
False
sage: all_vectors([(x,y)])
False
sage: all_vectors([M(0), M((x,y))])
True
sage: all_vectors([M(0), M((x,y)), (0,0)])
False
```

sage.libs.singular.function.is_sage_wrapper_for_singular_ring (ring)

Check whether wrapped ring arises from Singular or Singular/Plural.

EXAMPLES:

```
sage: from sage.libs.singular.function import is_sage_wrapper_for_singular_ring
sage: P.<x,y,z> = QQ[]
sage: is_sage_wrapper_for_singular_ring(P)
True
```

```
sage: A.<x,y,z> = FreeAlgebra(QQ, 3)
sage: P = A.g_algebra(relations={y*x:-x*y}, order = 'lex')
sage: is_sage_wrapper_for_singular_ring(P)
True
```

sage.libs.singular.function.is_singular_poly_wrapper (p)

Checks if p is some data type corresponding to some singular poly.

EXAMPLES:

```
sage: from sage.libs.singular.function import is_singular_poly_wrapper
sage: A.<x,y,z> = FreeAlgebra(QQ, 3)
sage: H.<x,y,z> = A.g_algebra({z*x:x*z+2*x, z*y:y*z-2*y})
sage: is_singular_poly_wrapper(x+y)
True
```

sage.libs.singular.function. lib (name)

Load the Singular library name.

INPUT:

•name - a Singular library name

```
sage: from sage.libs.singular.function import singular_function
sage: from sage.libs.singular.function import lib as singular_lib
sage: singular_lib('general.lib')
sage: primes = singular_function('primes')
sage: primes(2,10, ring=GF(127)['x,y,z'])
(2, 3, 5, 7)
```

sage.libs.singular.function. list_of_functions (packages=False)

Return a list of all function names currently available.

INPUT:

•packages – include local functions in packages.

EXAMPLES:

```
sage: from sage.libs.singular.function import list_of_functions
sage: 'groebner' in list_of_functions()
True
```

sage.libs.singular.function. singular_function (name)

Construct a new libSingular function object for the given name.

This function works both for interpreter and built-in functions.

INPUT:

•name – the name of the function

EXAMPLES:

```
sage: P.<x,y,z> = PolynomialRing(QQ)
sage: f = 3*x*y + 2*z + 1
sage: g = 2*x + 1/2
sage: I = Ideal([f,g])
```

```
sage: from sage.libs.singular.function import singular_function
sage: std = singular_function("std")
sage: std(I)
[3*y - 8*z - 4, 4*x + 1]
sage: size = singular_function("size")
sage: size([2, 3, 3])
3
sage: size("sage")
4
sage: size(["hello", "sage"])
2
sage: factorize = singular_function("factorize")
sage: factorize(f)
[[1, 3*x*y + 2*z + 1], (1, 1)]
sage: factorize(f, 1)
[3*x*y + 2*z + 1]
```

We give a wrong number of arguments:

```
sage: factorize()
Traceback (most recent call last):
...
RuntimeError: error in Singular function call 'factorize':
Wrong number of arguments (got 0 arguments, arity code is 303)
```

```
sage: factorize(f, 1, 2)
Traceback (most recent call last):
...
RuntimeError: error in Singular function call 'factorize':
Wrong number of arguments (got 3 arguments, arity code is 303)
sage: factorize(f, 1, 2, 3)
Traceback (most recent call last):
...
RuntimeError: error in Singular function call 'factorize':
Wrong number of arguments (got 4 arguments, arity code is 303)
```

The Singular function list can be called with any number of arguments:

```
sage: singular_list = singular_function("list")
sage: singular_list(2, 3, 6)
[2, 3, 6]
sage: singular_list()
[]
sage: singular_list(1)
[1]
sage: singular_list(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

We try to define a non-existing function:

```
sage: number_foobar = singular_function('number_foobar')
Traceback (most recent call last):
...
NameError: Singular library function 'number_foobar' is not defined
```

```
sage: from sage.libs.singular.function import lib as singular_lib
sage: singular_lib('general.lib')
sage: number_e = singular_function('number_e')
sage: number_e(10r)
67957045707/25000000000
sage: RR(number_e(10r))
2.71828182828000
```

```
sage: singular_lib('primdec.lib')
sage: primdecGTZ = singular_function("primdecGTZ")
sage: primdecGTZ(I)
[[[y - 8/3*z - 4/3, x + 1/4], [y - 8/3*z - 4/3, x + 1/4]]]
sage: singular_list((1,2,3),3,[1,2,3], ring=P)
[(1, 2, 3), 3, [1, 2, 3]]
sage: ringlist=singular_function("ringlist")
sage: l = ringlist(P)
sage: 1[3].__class__
<class 'sage.rings.polynomial.multi_polynomial_sequence.PolynomialSequence_generic
' >
sage: 1
[0, ['x', 'y', 'z'], [['dp', (1, 1, 1)], ['C', (0,)]], [0]]
sage: ring=singular_function("ring")
sage: ring(l)
<RingWrap>
sage: matrix = Matrix(P, 2, 2)
sage: matrix.randomize(terms=1)
sage: det = singular_function("det")
```

```
sage: det(matrix)
-3/5*x*y*z
sage: coeffs = singular_function("coeffs")
sage: coeffs(x*y+y+1,y)
[ 1]
[x + 1]
sage: intmat = Matrix(ZZ, 2,2, [100,2,3,4])
sage: det(intmat)
sage: random = singular_function("random")
sage: A = random(10,2,3); A.nrows(), max(A.list()) <= 10</pre>
sage: P.<x,y,z> = PolynomialRing(QQ)
sage: M=P**3
sage: leadcoef = singular_function("leadcoef")
sage: v=M((100*x, 5*y, 10*z*x*y))
sage: leadcoef(v)
1.0
sage: v = M([x+y, x*y+y**3, z])
sage: lead = singular_function("lead")
sage: lead(v)
(0, y^3)
sage: jet = singular_function("jet")
sage: jet(v, 2)
(x + y, x*y, z)
sage: syz = singular_function("syz")
sage: I = P.ideal([x+y, x*y-y, y*2, x**2+1])
sage: M = syz(I)
sage: M
[(-2*y, 2, y + 1, 0), (0, -2, x - 1, 0), (x*y - y, -y + 1, 1, -y), (x^2 + 1, -x - y)]
\hookrightarrow 1, -1, -x)
sage: singular_lib("mprimdec.lib")
sage: syz(M)
[(-x - 1, y - 1, 2*x, -2*y)]
sage: GTZmod = singular_function("GTZmod")
sage: GTZmod(M)
[[[(-2*y, 2, y + 1, 0), (0, x + 1, 1, -y), (0, -2, x - 1, 0), (x*y - y, -y + 1, ]
\rightarrow 1, -y), (x^2 + 1, 0, 0, -x - y)], [0]]]
sage: mres = singular_function("mres")
sage: resolution = mres(M, 0)
sage: resolution
<Resolution>
sage: singular_list(resolution)
[[(-2*y, 2, y + 1, 0), (0, -2, x - 1, 0), (x*y - y, -y + 1, 1, -y), (x^2 + 1, -x - y)]
\rightarrow 1, -1, -x)], [(-x - 1, y - 1, 2*x, -2*y)], [(0)]]
sage: A.<x,y> = FreeAlgebra(QQ, 2)
sage: P.\langle x,y \rangle = A.g_algebra(\{y*x:-x*y\})
sage: I= Sequence([x*y,x+y], check=False, immutable=True)
sage: twostd = singular_function("twostd")
sage: twostd(I)
[x + y, y^2]
sage: M=syz(I)
doctest...
sage: M
[(x + y, x*y)]
sage: syz(M)
[(0)]
```

```
sage: mres(I, 0)
<Resolution>
sage: M=P**3
sage: v=M((100*x, 5*y, 10*y*x*y))
sage: leadcoef(v)
-10
sage: v = M([x+y,x*y+y**3,x])
sage: lead(v)
(0, y^3)
sage: jet(v, 2)
(x + y, x*y, x)
sage: l = ringlist(P)
sage: len(1)
sage: ring(l)
<noncommutative RingWrap>
sage: I=twostd(I)
sage: 1[3]=I
sage: ring(l)
<noncommutative RingWrap>
```

CHAPTER

THIRTEEN

LIBSINGULAR: FUNCTION FACTORY

AUTHORS:

• Martin Albrecht (2010-01): initial version

```
{\bf class} \ {\tt sage.libs.singular.function\_factory}. \ {\bf SingularFunctionFactory} \\ {\bf Bases:} \ {\tt object}
```

A convenient interface to libsingular functions.

```
trait_names ()
    EXAMPLES:
```

```
sage: import sage.libs.singular.function_factory
sage: "groebner" in sage.libs.singular.function_factory.ff.trait_names()
True
```

H /	D	T	F	
- T	۱г			г

FOURTEEN

LIBSINGULAR: CONVERSION ROUTINES AND INITIALISATION

AUTHOR:

• Martin Albrecht <malb@informatik.uni-bremen.de>



CHA	PT	ER
-----	----	----

FIFTEEN

WRAPPER FOR SINGULAR'S POLYNOMIAL ARITHMETIC

AUTHOR:

• Martin Albrecht (2009-07): refactoring



CHAPTER

SIXTEEN

LIBSINGULAR: OPTIONS

Singular uses a set of global options to determine verbosity and the behavior of certain algorithms. We provide an interface to these options in the most 'natural' python-ic way. Users who do not wish to deal with Singular functions directly usually do not have to worry about this interface or Singular options in general since this is taken care of by higher level functions.

We compute a Groebner basis for Cyclic-5 in two different contexts:

```
sage: P.<a,b,c,d,e> = PolynomialRing(GF(127))
sage: I = sage.rings.ideal.Cyclic(P)
sage: import sage.libs.singular.function_factory
sage: std = sage.libs.singular.function_factory.ff.std
```

By default, tail reductions are performed:

```
sage: from sage.libs.singular.option import opt, opt_ctx
sage: opt['red_tail']
True
sage: std(I)[-1]
d^2*e^6 + 28*b*c*d + ...
```

If we don't want this, we can create an option context, which disables this:

```
sage: with opt_ctx(red_tail=False, red_sb=False):
....:    std(I)[-1]
d^2*e^6 + 8*c^3 + ...
```

However, this does not affect the global state:

```
sage: opt['red_tail']
True
```

On the other hand, any assignment to an option object will immediately change the global state:

```
sage: opt['red_tail'] = False
sage: opt['red_tail']
False
sage: opt['red_tail'] = True
sage: opt['red_tail']
True
```

Assigning values within an option context, only affects this context:

```
sage: with opt_ctx:
....: opt['red_tail'] = False
```

```
sage: opt['red_tail']
True
```

Option contexts can also be safely stacked:

```
sage: with opt_ctx:
       opt['red_tail'] = False
. . . . :
. . . . :
        print(opt)
. . . . :
         with opt_ctx:
             opt['red_through'] = False
. . . . :
             print(opt)
. . . . :
. . .
general options for libSingular (current value 0x00000082)
general options for libSingular (current value 0x00000002)
sage: print(opt)
general options for libSingular (current value 0x02000082)
```

Furthermore, the integer valued options deg_bound and mult_bound can be used:

```
sage: R.<x,y> = QQ[]
sage: I = R*[x^3+y^2,x^2*y+1]
sage: opt['deg_bound'] = 2
sage: std(I)
[x^2*y + 1, x^3 + y^2]
sage: opt['deg_bound'] = 0
sage: std(I)
[y^3 - x, x^2*y + 1, x^3 + y^2]
```

The same interface is available for verbosity options:

```
sage: from sage.libs.singular.option import opt_verb
sage: opt_verb['not_warn_sb']
False
sage: opt.reset_default() # needed to avoid side effects
sage: opt_verb.reset_default() # needed to avoid side effects
```

AUTHOR:

- Martin Albrecht (2009-08): initial implementation
- Martin Albrecht (2010-01): better interface, verbosity options
- Simon King (2010-07): Python-ic option names; deg bound and mult bound

```
class sage.libs.singular.option. LibSingularOptions
```

Bases: sage.libs.singular.option.LibSingularOptions_abstract

Pythonic Interface to libSingular's options.

Supported options are:

- •return_sb or returnSB the functions syz , intersect , quotient , modulo return a standard base instead of a generating set if return_sb is set. This option should not be used for lift .
- •fast_hc or fastHC tries to find the highest corner of the staircase (HC) as fast as possible during a standard basis computation (only used for local orderings).

- •int_strategy or intStrategy avoids division of coefficients during standard basis computations. This option is ring dependent. By default, it is set for rings with characteristic 0 and not set for all other rings.
- •lazy uses a more lazy approach in std computations, which was used in SINGULAR version before 2-0 (and which may lead to faster or slower computations, depending on the example).
- •length select shorter reducers in std computations.
- •not_regularity or notRegularity disables the regularity bound for res and mres.
- •not_sugar or notSugar disables the sugar strategy during standard basis computation.
- •not_buckets or notBuckets disables the bucket representation of polynomials during standard basis computations. This option usually decreases the memory usage but increases the computation time. It should only be set for memory-critical standard basis computations.
- •old_std or oldStd uses a more lazy approach in std computations, which was used in SINGULAR version before 2-0 (and which may lead to faster or slower computations, depending on the example).
- •prot shows protocol information indicating the progress during the following computations: facstd , fglm , groebner , lres , mres , minres , mstd , res , slimgb , sres , std , stdfglm , stdhilb , syz .
- $ullet red_sb'$ or redSB computes a reduced standard basis in any standard basis computation.
- •red_tail or redTail reduction of the tails of polynomials during standard basis computations. This option is ring dependent. By default, it is set for rings with global degree orderings and not set for all other rings.
- •red_through or redThrough for inhomogenous input, polynomial reductions during standard basis computations are never postponed, but always finished through. This option is ring dependent. By default, it is set for rings with global degree orderings and not set for all other rings.
- •sugar_crit or sugarCrit uses criteria similar to the homogeneous case to keep more useless pairs.
- •weight_m or weightM automatically computes suitable weights for the weighted ecart and the weighted sugar method.

In addition, two integer valued parameters are supported, namely:

- •deg_bound or degBound The standard basis computation is stopped if the total (weighted) degree exceeds deg_bound. deg_bound should not be used for a global ordering with inhomogeneous input. Reset this bound by setting deg_bound to 0. The exact meaning of "degree" depends on the ring odering and the command: slimgb uses always the total degree with weights 1, std does so for block orderings, only.
- •mult_bound or multBound The standard basis computation is stopped if the ideal is zero-dimensional in a ring with local ordering and its multiplicity is lower than $mult_bound$. Reset this bound by setting $mult_bound$ to 0.

EXAMPLES:

```
sage: from sage.libs.singular.option import LibSingularOptions
sage: libsingular_options = LibSingularOptions()
sage: libsingular_options
general options for libSingular (current value 0x06000082)
```

Here we demonstrate the intended way of using libSingular options:

```
sage: R.<x,y> = QQ[]
sage: I = R*[x^3+y^2,x^2*y+1]
sage: I.groebner_basis(deg_bound=2)
[x^3 + y^2, x^2*y + 1]
sage: I.groebner_basis()
[x^3 + y^2, x^2*y + 1, y^3 - x]
```

The option mult bound is only relevant in the local case:

reset default ()

Reset libSingular's default options.

EXAMPLES:

```
sage: from sage.libs.singular.option import opt
sage: opt['red_tail']
True
sage: opt['red_tail'] = False
sage: opt['red_tail']
False
sage: opt['deg_bound']
0
sage: opt['deg_bound'] = 2
sage: opt['deg_bound']
2
sage: opt.reset_default()
sage: opt['red_tail']
True
sage: opt['deg_bound']
0
```

class sage.libs.singular.option. LibSingularOptionsContext

Bases: object

Option context

This object localizes changes to options.

```
sage: from sage.libs.singular.option import opt, opt_ctx
sage: opt
general options for libSingular (current value 0x06000082)
```

```
sage: with opt_ctx(redTail=False):
....:     print(opt)
....:     with opt_ctx(redThrough=False):
....:     print(opt)
general options for libSingular (current value 0x04000082)
general options for libSingular (current value 0x04000002)

sage: print(opt)
general options for libSingular (current value 0x06000082)
```

opt

 ${\bf class} \; {\tt sage.libs.singular.option.} \; {\bf LibSingularOptions_abstract}$

Bases: object

Abstract Base Class for libSingular options.

load (value=None)

EXAMPLES:

```
sage: from sage.libs.singular.option import opt as sopt
sage: bck = sopt.save(); hex(bck[0]), bck[1], bck[2]
('0x6000082', 0, 0)
sage: sopt['redTail'] = False
sage: hex(int(sopt))
'0x4000082'
sage: sopt.load(bck)
sage: sopt['redTail']
True
```

save ()

Return a triple of integers that allow reconstruction of the options.

EXAMPLES:

```
sage: from sage.libs.singular.option import opt
sage: opt['deg_bound']
0
sage: opt['red_tail']
True
sage: s = opt.save()
sage: opt['deg_bound'] = 2
sage: opt['red_tail'] = False
sage: opt['deg_bound']
2
sage: opt['red_tail']
False
sage: opt.load(s)
sage: opt['deg_bound']
0
sage: opt['red_tail']
True
sage: opt.reset_default() # needed to avoid side effects
```

class sage.libs.singular.option. LibSingularVerboseOptions

Bases: sage.libs.singular.option.LibSingularOptions_abstract

Pythonic Interface to libSingular's verbosity options.

Supported options are:

- •mem shows memory usage in square brackets.
- •yacc Only available in debug version.
- •redefine warns about variable redefinitions.
- •reading shows the number of characters read from a file.
- •loadLib or load lib shows loading of libraries.
- •debugLib or debug lib warns about syntax errors when loading a library.
- •loadProc or load_proc shows loading of procedures from libraries.
- •defRes or def_res shows the names of the syzygy modules while converting resolution to list.
- •usage shows correct usage in error messages.
- •Imap or imap shows the mapping of variables with the fetch and imap commands.
- •notWarnSB or not_warn_sb do not warn if a basis is not a standard basis
- •contentSB or content_sb avoids to divide by the content of a polynomial in std and related algorithms. Should usually not be used.
- •cancelunit avoids to divide polynomials by non-constant units in std in the local case. Should usually not be used.

EXAMPLES:

```
sage: from sage.libs.singular.option import LibSingularVerboseOptions
sage: libsingular_verbose = LibSingularVerboseOptions()
sage: libsingular_verbose
verbosity options for libSingular (current value 0x00002851)
```

reset_default ()

Return to libSingular's default verbosity options

```
sage: from sage.libs.singular.option import opt_verb
sage: opt_verb['not_warn_sb']
False
sage: opt_verb['not_warn_sb'] = True
sage: opt_verb['not_warn_sb']
True
sage: opt_verb.reset_default()
sage: opt_verb['not_warn_sb']
False
```

CHAPTER

SEVENTEEN

WRAPPER FOR SINGULAR'S RINGS

AUTHORS:

- Martin Albrecht (2009-07): initial implementation
- Kwankyu Lee (2010-06): added matrix term order support

```
sage.libs.singular.ring.currRing_wrapper()
```

Returns a wrapper for the current ring, for use in debugging ring_refcount_dict.

EXAMPLES:

```
sage: from sage.libs.singular.ring import currRing_wrapper
sage: currRing_wrapper()
The ring pointer ...
```

```
\verb|sage.libs.singular.ring.poison_currRing| (\textit{frame}, \textit{event}, \textit{arg})
```

Poison the currRing pointer.

This function sets the currRing to an illegal value. By setting it as the python debug hook, you can poison the currRing before every evaluated Python command (but not within Cython code).

INPUT:

•frame, event, arg - the standard arguments for the CPython debugger hook. They are not used.

OUTPUT:

Returns itself, which ensures that poison_currRing() will stay in the debugger hook.

EXAMPLES:

```
sage: previous_trace_func = sys.gettrace() # None if no debugger running
sage: from sage.libs.singular.ring import poison_currRing
sage: sys.settrace(poison_currRing)
sage: sys.gettrace()
<built-in function poison_currRing>
sage: sys.settrace(previous_trace_func) # switch it off again
```

```
sage.libs.singular.ring.print_currRing()
```

Print the currRing pointer.

```
sage: from sage.libs.singular.ring import print_currRing
sage: print_currRing() # random output
DEBUG: currRing == 0x7fc6fa6ec480
sage: from sage.libs.singular.ring import poison_currRing
```

```
sage: _ = poison_currRing(None, None, None)
sage: print_currRing()
DEBUG: currRing == 0x0
```

```
class sage.libs.singular.ring.ring_wrapper_Py
```

Bases: object

Python object wrapping the ring pointer.

This is useful to store ring pointers in Python containers.

You must not construct instances of this class yourself, use wrap_ring() instead.

```
sage: from sage.libs.singular.ring import ring_wrapper_Py
sage: ring_wrapper_Py
<type 'sage.libs.singular.ring_wrapper_Py'>
```

CHAPTER

EIGHTEEN

SINGULAR'S GROEBNER STRATEGY OBJECTS

AUTHORS:

- Martin Albrecht (2009-07): initial implementation
- Michael Brickenstein (2009-07): initial implementation
- Hans Schoenemann (2009-07): initial implementation

```
{\bf class} \ {\bf sage.libs.singular.groebner\_strategy}. \ {\bf GroebnerStrategy} \\ {\bf Bases:} \ {\bf sage.structure.sage\_object.SageObject}
```

A Wrapper for Singular's Groebner Strategy Object.

This object provides functions for normal form computations and other functions for Groebner basis computation.

ALGORITHM:

Uses Singular via libSINGULAR

ideal ()

Return the ideal this strategy object is defined for.

EXAMPLES:

```
sage: from sage.libs.singular.groebner_strategy import GroebnerStrategy
sage: P.<x,y,z> = PolynomialRing(GF(32003))
sage: I = Ideal([x + z, y + z])
sage: strat = GroebnerStrategy(I)
sage: strat.ideal()
Ideal (x + z, y + z) of Multivariate Polynomial Ring in x, y, z over Finite
→Field of size 32003
```

normal_form (p)

Compute the normal form of p with respect to the generators of this object.

```
sage: from sage.libs.singular.groebner_strategy import GroebnerStrategy
sage: P.<x,y,z> = PolynomialRing(QQ)
sage: I = Ideal([x + z, y + z])
sage: strat = GroebnerStrategy(I)
sage: strat.normal_form(x*y) # indirect doctest
z^2
sage: strat.normal_form(x + 1)
-z + 1
```

ring()

Return the ring this strategy object is defined over.

EXAMPLES:

```
sage: from sage.libs.singular.groebner_strategy import GroebnerStrategy
sage: P.<x,y,z> = PolynomialRing(GF(32003))
sage: I = Ideal([x + z, y + z])
sage: strat = GroebnerStrategy(I)
sage: strat.ring()
Multivariate Polynomial Ring in x, y, z over Finite Field of size 32003
```

class sage.libs.singular.groebner_strategy. NCGroebnerStrategy

Bases: sage.structure.sage_object.SageObject

A Wrapper for Singular's Groebner Strategy Object.

This object provides functions for normal form computations and other functions for Groebner basis computation.

ALGORITHM:

Uses Singular via libSINGULAR

ideal ()

Return the ideal this strategy object is defined for.

EXAMPLES:

```
sage: from sage.libs.singular.groebner_strategy import NCGroebnerStrategy
sage: A.<x,y,z> = FreeAlgebra(QQ, 3)
sage: H.<x,y,z> = A.g_algebra({y*x:x*y-z, z*x:x*z+2*x, z*y:y*z-2*y})
sage: I = H.ideal([y^2, x^2, z^2-H.one()])
sage: strat = NCGroebnerStrategy(I)
sage: strat.ideal() == I
True
```

normal form (p)

Compute the normal form of p with respect to the generators of this object.

EXAMPLES:

```
sage: A.<x,y,z> = FreeAlgebra(QQ, 3)
sage: H.<x,y,z> = A.g_algebra({y*x:x*y-z, z*x:x*z+2*x, z*y:y*z-2*y})
sage: JL = H.ideal([x^3, y^3, z^3 - 4*z])
sage: JT = H.ideal([x^3, y^3, z^3 - 4*z], side='twosided')
sage: from sage.libs.singular.groebner_strategy import NCGroebnerStrategy
sage: SL = NCGroebnerStrategy(JL.std())
sage: ST = NCGroebnerStrategy(JT.std())
sage: SL.normal_form(x*y^2)
x*y^2
sage: ST.normal_form(x*y^2)
y*z
```

ring()

Return the ring this strategy object is defined over.

```
sage: from sage.libs.singular.groebner_strategy import NCGroebnerStrategy
sage: A.<x,y,z> = FreeAlgebra(QQ, 3)
```

```
sage: H.<x,y,z> = A.g_algebra({y*x:x*y-z, z*x:x*z+2*x, z*y:y*z-2*y})
sage: I = H.ideal([y^2, x^2, z^2-H.one()])
sage: strat = NCGroebnerStrategy(I)
sage: strat.ring() is H
True
```

sage.libs.singular.groebner_strategy. $unpickle_GroebnerStrategy0$ (I) EXAMPLES:

```
sage: from sage.libs.singular.groebner_strategy import GroebnerStrategy
sage: P.<x,y,z> = PolynomialRing(GF(32003))
sage: I = Ideal([x + z, y + z])
sage: strat = GroebnerStrategy(I)
sage: loads(dumps(strat)) == strat # indirect doctest
True
```

sage.libs.singular.groebner_strategy. $unpickle_NCGroebnerStrategy0$ (I) EXAMPLES:

```
sage: from sage.libs.singular.groebner_strategy import NCGroebnerStrategy
sage: A.<x,y,z> = FreeAlgebra(QQ, 3)
sage: H.<x,y,z> = A.g_algebra({y*x:x*y-z, z*x:x*z+2*x, z*y:y*z-2*y})
sage: I = H.ideal([y^2, x^2, z^2-H.one()])
sage: strat = NCGroebnerStrategy(I)
sage: loads(dumps(strat)) == strat # indirect doctest
True
```

Sage Reference Manual: C/C++ Library Interfaces, Release 7.6	

CHAPTER

NINETEEN

CYTHON WRAPPER FOR THE PARMA POLYHEDRA LIBRARY (PPL)

The Parma Polyhedra Library (PPL) is a library for polyhedral computations over \mathbf{Q} . This interface tries to reproduce the C++ API as faithfully as possible in Cython/Sage. For example, the following C++ excerpt:

```
Variable x(0);
Variable y(1);
Constraint_System cs;
cs.insert(x >= 0);
cs.insert(x <= 3);
cs.insert(y >= 0);
cs.insert(y <= 3);
C_Polyhedron poly_from_constraints(cs);</pre>
```

translates into:

```
sage: from sage.libs.ppl import Variable, Constraint_System, C_Polyhedron
sage: x = Variable(0)
sage: y = Variable(1)
sage: cs = Constraint_System()
sage: cs.insert(x >= 0)
sage: cs.insert(x <= 3)
sage: cs.insert(y >= 0)
sage: cs.insert(y <= 3)
sage: poly_from_constraints = C_Polyhedron(cs)</pre>
```

The same polyhedron constructed from generators:

```
sage: from sage.libs.ppl import Variable, Generator_System, C_Polyhedron, point
sage: gs = Generator_System()
sage: gs.insert(point(0*x + 0*y))
sage: gs.insert(point(0*x + 3*y))
sage: gs.insert(point(3*x + 0*y))
sage: gs.insert(point(3*x + 3*y))
sage: poly_from_generators = C_Polyhedron(gs)
```

Rich comparisons test equality/inequality and strict/non-strict containment:

```
sage: poly_from_generators == poly_from_constraints
True
sage: poly_from_generators >= poly_from_constraints
True
sage: poly_from_generators < poly_from_constraints
False
sage: poly_from_constraints.minimized_generators()
Generator_System {point(0/1, 0/1), point(0/1, 3/1), point(3/1, 0/1), point(3/1, 3/1)}</pre>
```

```
sage: poly_from_constraints.minimized_constraints()
Constraint_System {-x0+3>=0, -x1+3>=0, x0>=0, x1>=0}
```

As we see above, the library is generally easy to use. There are a few pitfalls that are not entirely obvious without consulting the documentation, in particular:

- There are no vectors used to describe <code>Generator</code> (points, closure points, rays, lines) or <code>Constraint</code> (strict inequalities, non-strict inequalities, or equations). Coordinates are always specified via linear polynomials in <code>Variable</code>
- All coordinates of rays and lines as well as all coefficients of constraint relations are (arbitrary precision) integers. Only the generators point () and closure_point () allow one to specify an overall divisor of the otherwise integral coordinates. For example:

```
sage: from sage.libs.ppl import Variable, point
sage: x = Variable(0); y = Variable(1)
sage: p = point( 2*x+3*y, 5 ); p
point(2/5, 3/5)
sage: p.coefficient(x)
2
sage: p.coefficient(y)
3
sage: p.divisor()
```

• PPL supports (topologically) closed polyhedra (*C_Polyhedron*) as well as not neccesarily closed polyhedra (*NNC_Polyhedron*). Only the latter allows closure points (=points of the closure but not of the actual polyhedron) and strict inequalities (> and <)

The naming convention for the C++ classes is that they start with PPL_ , for example, the original Linear_Expression becomes PPL_Linear_Expression . The Python wrapper has the same name as the original library class, that is, just Linear_Expression . In short:

- If you are using the Python wrapper (if in doubt: thats you), then you use the same names as the PPL C++ class library.
- If you are writing your own Cython code, you can access the underlying C++ classes by adding the prefix PPL_

Finally, PPL is fast. For example, here is the permutahedron of 5 basis vectors:

```
sage: from sage.libs.ppl import Variable, Generator_System, point, C_Polyhedron
sage: basis = list(range(5))
sage: x = [ Variable(i) for i in basis ]
sage: gs = Generator_System();
sage: for coeff in Permutations(basis):
....: gs.insert(point( sum( (coeff[i]+1)*x[i] for i in basis ) ))
sage: C_Polyhedron(gs)
A 4-dimensional polyhedron in QQ^5 defined as the convex hull of 120 points
```

The above computation (using PPL) finishes without noticeable delay (timeit measures it to be 90 microseconds on sage.math). Below we do the same computation with cddlib, which needs more than 3 seconds on the same hardware:

```
sage: basis = list(range(5))
sage: gs = [ tuple(coeff) for coeff in Permutations(basis) ]
sage: Polyhedron(vertices=gs, backend='cdd') # long time (3s on sage.math, 2011)
A 4-dimensional polyhedron in QQ^5 defined as the convex hull of 120 vertices
```

DIFFERENCES VS. C++

Since Python and C++ syntax are not always compatible, there are necessarily some differences. The main ones are:

- The Linear_Expression also accepts an iterable as input for the homogeneous coefficients.
- Polyhedron and its subclasses as well as Generator_System and Constraint_System can be set immutable via a set_immutable() method. This is the analog of declaring a C++ instance const. All other classes are immutable by themselves.

AUTHORS:

- Volker Braun (2010-10-08): initial version.
- Risan (2012-02-19): extension for MIP_Problem class

```
class sage.libs.ppl. C_Polyhedron
    Bases: sage.libs.ppl.Polyhedron
```

Wrapper for PPL's $C_Polyhedron\ class.$

An object of the class *C_Polyhedron* represents a topologically closed convex polyhedron in the vector space. See *NNC_Polyhedron* for more general (not necessarily closed) polyhedra.

When building a closed polyhedron starting from a system of constraints, an exception is thrown if the system contains a strict inequality constraint. Similarly, an exception is thrown when building a closed polyhedron starting from a system of generators containing a closure point.

INPUT:

- •arg the defining data of the polyhedron. Any one of the following is accepted:
 - -A non-negative integer. Depending on degenerate_element, either the space-filling or the empty polytope in the given dimension arg is constructed.

```
-A Constraint_System.
```

-A Generator_System.

-A single Constraint.

-A single Generator.

-A C_Polyhedron.

•degenerate_element - string, either 'universe' or 'empty'. Only used if arg is an integer.

OUTPUT:

A $C_Polyhedron$.

```
sage: gs.insert( point(-x-y) )
sage: gs.insert( ray(x) )
sage: C_Polyhedron(gs)
A 1-dimensional polyhedron in QQ^2 defined as the convex hull of 1 point, 1 ray
```

The empty and universe polyhedra are constructed like this:

```
sage: C_Polyhedron(3, 'empty')
The empty polyhedron in QQ^3
sage: C_Polyhedron(3, 'empty').constraints()
Constraint_System {-1==0}
sage: C_Polyhedron(3, 'universe')
The space-filling polyhedron in QQ^3
sage: C_Polyhedron(3, 'universe').constraints()
Constraint_System {}
```

Note that, by convention, the generator system of a polyhedron is either empty or contains at least one point. In particular, if you define a polyhedron via a non-empty <code>Generator_System</code> it must contain a point (at any position). If you start with a single generator, this generator must be a point:

```
sage: C_Polyhedron( ray(x) )
Traceback (most recent call last):
...
ValueError: PPL::C_Polyhedron::C_Polyhedron(gs):
*this is an empty polyhedron and
the non-empty generator system gs contains no points.
```

class sage.libs.ppl. Constraint

Bases: object

Wrapper for PPL's Constraint class.

An object of the class Constraint is either:

```
•an equality \sum_{i=0}^{n-1} a_i x_i + b = 0
```

- •a non-strict inequality $\sum_{i=0}^{n-1} a_i x_i + b \geq 0$
- •a strict inequality $\sum_{i=0}^{n-1} a_i x_i + b > 0$

where n is the dimension of the space, a_i is the integer coefficient of variable x_i , and b_i is the integer inhomogeneous term.

INPUT/OUTPUT:

You construct constraints by writing inequalities in Linear_Expression. Do not attempt to manually construct constraints.

```
sage: from sage.libs.ppl import Constraint, Variable, Linear_Expression
sage: x = Variable(0)
sage: y = Variable(1)
sage: 5*x-2*y > x+y-1
4*x0-3*x1+1>0
sage: 5*x-2*y >= x+y-1
4*x0-3*x1+1>=0
sage: 5*x-2*y == x+y-1
4*x0-3*x1+1==0
sage: 5*x-2*y <= x+y-1</pre>
```

```
-4*x0+3*x1-1>=0
sage: 5*x-2*y < x+y-1
-4*x0+3*x1-1>0
sage: x > 0
x0>0
```

Special care is needed if the left hand side is a constant:

```
sage: 0 == 1  # watch out!
False
sage: Linear_Expression(0) == 1
-1==0
```

OK ()

Check if all the invariants are satisfied.

EXAMPLES:

```
sage: from sage.libs.ppl import Linear_Expression, Variable
sage: x = Variable(0)
sage: y = Variable(1)
sage: ineq = (3*x+2*y+1>=0)
sage: ineq.OK()
True
```

ascii dump ()

Write an ASCII dump to stderr.

EXAMPLES:

coefficient (v)

Return the coefficient of the variable $\,\triangledown\,$.

INPUT:

```
\bulletv -a Variable.
```

OUTPUT:

An integer.

```
sage: from sage.libs.ppl import Variable
sage: x = Variable(0)
sage: ineq = (3*x+1 > 0)
sage: ineq.coefficient(x)
3
```

coefficients ()

Return the coefficients of the constraint.

See also coefficient ().

OUTPUT:

A tuple of integers of length space_dimension().

EXAMPLES:

```
sage: from sage.libs.ppl import Variable
sage: x = Variable(0); y = Variable(1)
sage: ineq = ( 3*x+5*y+1 == 2); ineq
3*x0+5*x1-1==0
sage: ineq.coefficients()
(3, 5)
```

inhomogeneous_term ()

Return the inhomogeneous term of the constraint.

OUTPUT:

Integer.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable
sage: y = Variable(1)
sage: ineq = ( 10+y > 9 )
sage: ineq
x1+1>0
sage: ineq.inhomogeneous_term()
1
```

is_equality()

Test whether self is an equality.

OUTPUT:

Boolean. Returns True if and only if self is an equality constraint.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable
sage: x = Variable(0)
sage: (x==0).is_equality()
True
sage: (x>=0).is_equality()
False
sage: (x>0).is_equality()
```

is_equivalent_to (c)

Test whether self and c are equivalent.

INPUT:

```
•c -a Constraint.
```

OUTPUT:

Boolean. Returns True if and only if self and c are equivalent constraints.

Note that constraints having different space dimensions are not equivalent. However, constraints having different types may nonetheless be equivalent, if they both are tautologies or inconsistent.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, Linear_Expression
sage: x = Variable(0)
sage: y = Variable(1)
sage: ( x>0 ).is_equivalent_to( Linear_Expression(0) < x )
True
sage: ( x>0 ).is_equivalent_to( 0*y<x )
False
sage: ( 0*x>1 ).is_equivalent_to( 0*x==-2 )
True
```

is_inconsistent()

Test whether self is an inconsistent constraint, that is, always false.

An inconsistent constraint can have either one of the following forms:

```
•an equality: \sum 0x_i + b = 0 with b \neq 0,
```

- •a non-strict inequality: $\sum 0x_i + b \ge 0$ with b < 0, or
- •a strict inequality: $\sum 0x_i + b > 0$ with $b \le 0$.

OUTPUT:

Boolean. Returns True if and only if self is an inconsistent constraint.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable
sage: x = Variable(0)
sage: (x==1).is_inconsistent()
False
sage: (0*x>=1).is_inconsistent()
True
```

is_inequality()

Test whether self is an inequality.

OUTPUT:

Boolean. Returns True if and only if self is an inequality constraint, either strict or non-strict.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable
sage: x = Variable(0)
sage: (x==0).is_inequality()
False
sage: (x>=0).is_inequality()
True
sage: (x>0).is_inequality()
True
```

is_nonstrict_inequality()

Test whether self is a non-strict inequality.

OUTPUT:

Boolean. Returns True if and only if self is an non-strict inequality constraint.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable
sage: x = Variable(0)
sage: (x==0).is_nonstrict_inequality()
False
sage: (x>=0).is_nonstrict_inequality()
True
sage: (x>0).is_nonstrict_inequality()
False
```

is strict inequality()

Test whether self is a strict inequality.

OUTPUT

Boolean. Returns True if and only if self is an strict inequality constraint.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable
sage: x = Variable(0)
sage: (x==0).is_strict_inequality()
False
sage: (x>=0).is_strict_inequality()
False
sage: (x>0).is_strict_inequality()
True
```

is_tautological ()

Test whether self is a tautological constraint.

A tautology can have either one of the following forms:

```
•an equality: \sum 0x_i + 0 = 0,
```

- •a non-strict inequality: $\sum 0x_i + b \ge 0$ with $b \ge 0$, or
- •a strict inequality: $\sum 0x_i + b > 0$ with b > 0.

OUTPUT:

Boolean. Returns True if and only if self is a tautological constraint.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable
sage: x = Variable(0)
sage: (x==0).is_tautological()
False
sage: (0*x>=0).is_tautological()
True
```

space_dimension ()

Return the dimension of the vector space enclosing self.

OUTPUT:

Integer.

```
sage: from sage.libs.ppl import Variable
sage: x = Variable(0)
sage: y = Variable(1)
sage: (x>=0).space_dimension()
1
sage: (y==1).space_dimension()
2
```

type ()

Return the constraint type of self.

OUTPUT:

String. One of 'equality', 'nonstrict_inequality', or 'strict_inequality'.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable
sage: x = Variable(0)
sage: (x==0).type()
'equality'
sage: (x>=0).type()
'nonstrict_inequality'
sage: (x>0).type()
'strict_inequality'
```

class sage.libs.ppl. Constraint_System

Bases: sage.libs.ppl._mutable_or_immutable

Wrapper for PPL's Constraint_System class.

An object of the class Constraint_System is a system of constraints, i.e., a multiset of objects of the class Constraint. When inserting constraints in a system, space dimensions are automatically adjusted so that all the constraints in the system are defined on the same vector space.

EXAMPLES:

```
sage: from sage.libs.ppl import Constraint_System, Variable
sage: x = Variable(0)
sage: y = Variable(1)
sage: cs = Constraint_System( 5*x-2*y > 0 )
sage: cs.insert( 6*x<3*y )
sage: cs.insert( x >= 2*x-7*y )
sage: cs.Constraint_System {5*x0-2*x1>0, -2*x0+x1>0, -x0+7*x1>=0}
```

OK ()

Check if all the invariants are satisfied.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, Constraint_System
sage: x = Variable(0)
sage: y = Variable(1)
sage: cs = Constraint_System( 3*x+2*y+1 <= 10 )
sage: cs.OK()
True</pre>
```

ascii_dump ()

Write an ASCII dump to stderr.

EXAMPLES:

clear ()

Removes all constraints from the constraint system and sets its space dimension to 0.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, Constraint_System
sage: x = Variable(0)
sage: cs = Constraint_System(x>0)
sage: cs
Constraint_System {x0>0}
sage: cs.clear()
sage: cs
Constraint_System {}
```

empty ()

Return True if and only if self has no constraints.

OUTPUT:

Boolean.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, Constraint_System, point
sage: x = Variable(0)
sage: cs = Constraint_System()
sage: cs.empty()
True
sage: cs.insert(x>0)
sage: cs.empty()
False
```

has_equalities ()

Tests whether self contains one or more equality constraints.

OUTPUT:

Boolean.

```
sage: from sage.libs.ppl import Variable, Constraint_System
sage: x = Variable(0)
sage: cs = Constraint_System()
sage: cs.insert(x>0)
```

```
sage: cs.insert( x<0 )
sage: cs.has_equalities()
False
sage: cs.insert( x==0 )
sage: cs.has_equalities()
True</pre>
```

has_strict_inequalities ()

Tests whether self contains one or more strict inequality constraints.

OUTPUT:

Boolean.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, Constraint_System
sage: x = Variable(0)
sage: cs = Constraint_System()
sage: cs.insert( x>=0 )
sage: cs.insert( x==-1 )
sage: cs.has_strict_inequalities()
False
sage: cs.insert( x>0 )
sage: cs.has_strict_inequalities()
True
```

insert (c)

Insert c into the constraint system.

INPUT:

 \bullet c -a Constraint.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, Constraint_System
sage: x = Variable(0)
sage: cs = Constraint_System()
sage: cs.insert( x>0 )
sage: cs
Constraint_System {x0>0}
```

space_dimension ()

Return the dimension of the vector space enclosing self.

OUTPUT:

Integer.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, Constraint_System
sage: x = Variable(0)
sage: cs = Constraint_System( x>0 )
sage: cs.space_dimension()
1
```

```
class sage.libs.ppl. Constraint_System_iterator
```

Bases: object

Wrapper for PPL's Constraint_System::const_iterator class.

EXAMPLES:

next ()

x.next() -> the next value, or raise StopIteration

```
class sage.libs.ppl. Generator
```

Bases: object

Wrapper for PPL's Generator class.

An object of the class Generator is one of the following:

```
•a line \ell=(a_0,\ldots,a_{n-1})^T

•a ray r=(a_0,\ldots,a_{n-1})^T

•a point p=(\frac{a_0}{d},\ldots,\frac{a_{n-1}}{d})^T

•a closure point c=(\frac{a_0}{d},\ldots,\frac{a_{n-1}}{d})^T
```

where n is the dimension of the space and, for points and closure points, d is the divisor.

INPUT/OUTPUT:

Use the helper functions line(), ray(), point(), and $closure_point()$ to construct generators. Analogous class methods are also available, see Generator.line(), Generator.ray(), Generator.point(), $Generator.closure_point()$. Do not attempt to construct generators manually.

Note: The generators are constructed from linear expressions. The inhomogeneous term is always silently discarded.

```
sage: from sage.libs.ppl import Generator, Variable
sage: x = Variable(0)
sage: y = Variable(1)
sage: Generator.line(5*x-2*y)
line(5, -2)
sage: Generator.ray(5*x-2*y)
ray(5, -2)
sage: Generator.point(5*x-2*y, 7)
point(5/7, -2/7)
sage: Generator.closure_point(5*x-2*y, 7)
closure_point(5/7, -2/7)
```

OK ()

Check if all the invariants are satisfied.

EXAMPLES:

```
sage: from sage.libs.ppl import Linear_Expression, Variable
sage: x = Variable(0)
sage: y = Variable(1)
sage: e = 3*x+2*y+1
sage: e.OK()
True
```

ascii_dump ()

Write an ASCII dump to stderr.

EXAMPLES:

static closure_point (expression=0, divisor=1)

Construct a closure point.

A closure point is a point of the topological closure of a polyhedron that is not a point of the polyhedron itself.

INPUT:

- •expression a Linear_Expression or something convertible to it (Variable or integer).
- •divisor an integer.

OUTPUT:

A new Generator representing the point.

Raises a ValueError if ``divisor==0.

```
sage: from sage.libs.ppl import Generator, Variable
sage: y = Variable(1)
sage: Generator.closure_point(2*y+7, 3)
closure_point(0/3, 2/3)
sage: Generator.closure_point(y+7, 3)
closure_point(0/3, 1/3)
sage: Generator.closure_point(7, 3)
closure_point()
sage: Generator.closure_point(0, 0)
Traceback (most recent call last):
...
ValueError: PPL::closure_point(e, d):
d == 0.
```

```
coefficient ( v)
    Return the coefficient of the variable v.
    INPUT:
       •v -a Variable.
    OUTPUT:
    An integer.
    EXAMPLES:
    sage: from sage.libs.ppl import Variable, line
    sage: x = Variable(0)
    sage: line = line(3*x+1)
    sage: line
    line(1)
    sage: line.coefficient(x)
coefficients ()
    Return the coefficients of the generator.
    See also coefficient().
    OUTPUT:
    A tuple of integers of length space_dimension().
    EXAMPLES:
    sage: from sage.libs.ppl import Variable, point
    sage: x = Variable(0); y = Variable(1)
    sage: p = point(3*x+5*y+1, 2); p
    point(3/2, 5/2)
    sage: p.coefficients()
    (3, 5)
divisor ()
    If self is either a point or a closure point, return its divisor.
    OUTPUT:
    An integer. If self is a ray or a line, raises ValueError.
    EXAMPLES:
    sage: from sage.libs.ppl import Generator, Variable
    sage: x = Variable(0)
    sage: y = Variable(1)
    sage: point = Generator.point(2*x-y+5)
    sage: point.divisor()
    sage: line = Generator.line(2 * x - y + 5)
    sage: line.divisor()
    Traceback (most recent call last):
```

is_closure_point ()

Test whether self is a closure point.

ValueError: PPL::Generator::divisor():

*this is neither a point nor a closure point.

OUTPUT:

Boolean.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, point, closure_point, ray, line
sage: x = Variable(0)
sage: line(x).is_closure_point()
False
sage: ray(x).is_closure_point()
False
sage: point(x,2).is_closure_point()
False
sage: closure_point(x,2).is_closure_point()
True
```

is_equivalent_to (g)

Test whether self and g are equivalent.

INPUT:

```
\bulletq -a Generator.
```

OUTPUT:

Boolean. Returns True if and only if self and g are equivalent generators.

Note that generators having different space dimensions are not equivalent.

EXAMPLES:

is_line()

Test whether self is a line.

OUTPUT:

Boolean.

```
sage: from sage.libs.ppl import Variable, point, closure_point, ray, line
sage: x = Variable(0)
sage: line(x).is_line()
True
sage: ray(x).is_line()
False
sage: point(x,2).is_line()
False
sage: closure_point(x,2).is_line()
```

is_line_or_ray()

Test whether self is a line or a ray.

OUTPUT:

Boolean.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, point, closure_point, ray, line
sage: x = Variable(0)
sage: line(x).is_line_or_ray()
True
sage: ray(x).is_line_or_ray()
True
sage: point(x,2).is_line_or_ray()
False
sage: closure_point(x,2).is_line_or_ray()
False
```

is_point()

Test whether self is a point.

OUTPUT:

Boolean.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, point, closure_point, ray, line
sage: x = Variable(0)
sage: line(x).is_point()
False
sage: ray(x).is_point()
False
sage: point(x,2).is_point()
True
sage: closure_point(x,2).is_point()
False
```

is_ray ()

Test whether self is a ray.

OUTPUT:

Boolean.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, point, closure_point, ray, line
sage: x = Variable(0)
sage: line(x).is_ray()
False
sage: ray(x).is_ray()
True
sage: point(x,2).is_ray()
False
sage: closure_point(x,2).is_ray()
False
```

static line (expression)

Construct a line.

INPUT:

•expression - a Linear_Expression or something convertible to it (Variable or integer).

OUTPUT:

A new Generator representing the line.

Raises a ValueError` if the homogeneous part of ``expression represents the origin of the vector space.

EXAMPLES:

```
sage: from sage.libs.ppl import Generator, Variable
sage: y = Variable(1)
sage: Generator.line(2*y)
line(0, 1)
sage: Generator.line(y)
line(0, 1)
sage: Generator.line(1)
Traceback (most recent call last):
...
ValueError: PPL::line(e):
e == 0, but the origin cannot be a line.
```

static point (expression=0, divisor=1)

Construct a point.

INPUT:

•expression - a Linear_Expression or something convertible to it (Variable or integer).

•divisor - an integer.

OUTPUT:

A new Generator representing the point.

Raises a ValueError` if ``divisor==0.

EXAMPLES:

```
sage: from sage.libs.ppl import Generator, Variable
sage: y = Variable(1)
sage: Generator.point(2*y+7, 3)
point(0/3, 2/3)
sage: Generator.point(y+7, 3)
point(0/3, 1/3)
sage: Generator.point(7, 3)
point()
sage: Generator.point(0, 0)
Traceback (most recent call last):
...
ValueError: PPL::point(e, d):
d == 0.
```

static ray (expression)

Construct a ray.

INPUT:

•expression - a Linear_Expression or something convertible to it (Variable or integer).

OUTPUT:

A new Generator representing the ray.

Raises a ValueError` if the homogeneous part of ``expression represents the origin of the vector space.

EXAMPLES:

```
sage: from sage.libs.ppl import Generator, Variable
sage: y = Variable(1)
sage: Generator.ray(2*y)
ray(0, 1)
sage: Generator.ray(y)
ray(0, 1)
sage: Generator.ray(1)
Traceback (most recent call last):
...
ValueError: PPL::ray(e):
e == 0, but the origin cannot be a ray.
```

space_dimension ()

Return the dimension of the vector space enclosing self.

OUTPUT:

Integer.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, point
sage: x = Variable(0)
sage: y = Variable(1)
sage: point(x).space_dimension()
1
sage: point(y).space_dimension()
2
```

type ()

Return the generator type of self.

OUTPUT:

String. One of 'line', 'ray', 'point', or 'closure_point'.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, point, closure_point, ray, line
sage: x = Variable(0)
sage: line(x).type()
'line'
sage: ray(x).type()
'ray'
sage: point(x,2).type()
'point'
sage: closure_point(x,2).type()
'closure_point'
```

```
class sage.libs.ppl. Generator_System
```

Bases: sage.libs.ppl._mutable_or_immutable

Wrapper for PPL's Generator_System class.

An object of the class Generator_System is a system of generators, i.e., a multiset of objects of the class Generator (lines, rays, points and closure points). When inserting generators in a system, space dimensions are automatically adjusted so that all the generators in the system are defined on the same vector space. A system of generators which is meant to define a non-empty polyhedron must include at least one point: the reason is that lines, rays and closure points need a supporting point (lines and rays only specify directions while closure points only specify points in the topological closure of the NNC polyhedron).

EXAMPLES:

OK ()

Check if all the invariants are satisfied.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, Generator_System, point
sage: x = Variable(0)
sage: y = Variable(1)
sage: gs = Generator_System( point(3*x+2*y+1) )
sage: gs.OK()
True
```

ascii_dump ()

Write an ASCII dump to stderr.

EXAMPLES:

clear ()

Removes all generators from the generator system and sets its space dimension to 0.

```
sage: from sage.libs.ppl import Variable, Generator_System, point
sage: x = Variable(0)
sage: gs = Generator_System( point(3*x) ); gs
Generator_System {point(3/1)}
sage: gs.clear()
sage: gs
Generator_System {}
```

empty ()

Return True if and only if self has no generators.

OUTPUT:

Boolean.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, Generator_System, point
sage: x = Variable(0)
sage: gs = Generator_System()
sage: gs.empty()
True
sage: gs.insert( point(-3*x) )
sage: gs.empty()
False
```

insert (g)

Insert g into the generator system.

The number of space dimensions of self is increased, if needed.

INPUT:

 \bullet q - a Generator.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, Generator_System, point
sage: x = Variable(0)
sage: gs = Generator_System( point(3*x) )
sage: gs.insert( point(-3*x) )
sage: gs
Generator_System {point(3/1), point(-3/1)}
```

space_dimension ()

Return the dimension of the vector space enclosing self.

OUTPUT:

Integer.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, Generator_System, point
sage: x = Variable(0)
sage: gs = Generator_System( point(3*x) )
sage: gs.space_dimension()
1
```

class sage.libs.ppl. Generator_System_iterator

Bases: object

Wrapper for PPL's Generator_System::const_iterator class.

EXAMPLES:

```
sage: from sage.libs.ppl import Generator_System, Variable, line, ray, point,

→closure_point, Generator_System_iterator
sage: x = Variable(0)
sage: y = Variable(1)
sage: gs = Generator_System( line(5*x-2*y) )
sage: gs.insert( ray(6*x-3*y) )
sage: gs.insert( point(2*x-7*y, 5) )
sage: gs.insert( closure_point(9*x-1*y, 2) )
sage: next(Generator_System_iterator(gs))
line(5, -2)
sage: list(gs)
[line(5, -2), ray(2, -1), point(2/5, -7/5), closure_point(9/2, -1/2)]
```

next ()

x.next() -> the next value, or raise StopIteration

```
class sage.libs.ppl. Linear_Expression
```

Bases: object

Wrapper for PPL's PPL_Linear_Expression class.

INPUT:

The constructor accepts zero, one, or two arguments.

If there are two arguments Linear_Expression(a,b), they are interpreted as

- •a an iterable of integer coefficients, for example a list.
- •b an integer. The inhomogeneous term.

A single argument Linear_Expression (arg) is interpreted as

- •arg something that determines a linear expression. Possibilities are:
 - -a Variable: The linear expression given by that variable.
 - -a Linear_Expression: The copy constructor.
 - -an integer: Constructs the constant linear expression.

No argument is the default constructor and returns the zero linear expression.

OUTPUT:

A Linear Expression

```
sage: from sage.libs.ppl import Variable, Linear_Expression
sage: Linear_Expression([1,2,3,4],5)
x0+2*x1+3*x2+4*x3+5
sage: Linear_Expression(10)
10
sage: Linear_Expression()
0
sage: Linear_Expression(10).inhomogeneous_term()
10
sage: x = Variable(123)
sage: expr = x+1; expr
x123+1
```

```
sage: expr.OK()
True
sage: expr.coefficient(x)
1
sage: expr.coefficient( Variable(124) )
0
```

OK ()

Check if all the invariants are satisfied.

EXAMPLES:

```
sage: from sage.libs.ppl import Linear_Expression, Variable
sage: x = Variable(0)
sage: y = Variable(1)
sage: e = 3*x+2*y+1
sage: e.OK()
True
```

all_homogeneous_terms_are_zero ()

Test if self is a constant linear expression.

OUTPUT:

Boolean.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, Linear_Expression
sage: Linear_Expression(10).all_homogeneous_terms_are_zero()
True
```

ascii dump ()

Write an ASCII dump to stderr.

EXAMPLES:

```
sage: sage_cmd = 'from sage.libs.ppl import Linear_Expression, Variable\n'
sage: sage_cmd += 'x = Variable(0)\n'
sage: sage_cmd += 'y = Variable(1)\n'
sage: sage_cmd += 'e = 3*x+2*y+1\n'
sage: sage_cmd += 'e.ascii_dump()\n'
sage: from sage.tests.cmdline import test_executable
sage: (out, err, ret) = test_executable(['sage', '-c', sage_cmd],__
timeout=100)  # long time, indirect doctest
sage: print(err)  # long time
size 3 1 3 2
```

coefficient(v)

Return the coefficient of the variable v.

INPUT:

```
•v -a Variable.
```

OUTPUT:

An integer.

```
sage: from sage.libs.ppl import Variable
sage: x = Variable(0)
sage: e = 3*x+1
sage: e.coefficient(x)
3
```

coefficients ()

Return the coefficients of the linear expression.

OUTPUT:

A tuple of integers of length space dimension().

EXAMPLES:

```
sage: from sage.libs.ppl import Variable
sage: x = Variable(0); y = Variable(1)
sage: e = 3*x+5*y+1
sage: e.coefficients()
(3, 5)
```

inhomogeneous_term ()

Return the inhomogeneous term of the linear expression.

OUTPUT:

Integer.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, Linear_Expression
sage: Linear_Expression(10).inhomogeneous_term()
10
```

is_zero ()

Test if self is the zero linear expression.

OUTPUT:

Boolean.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, Linear_Expression
sage: Linear_Expression(0).is_zero()
True
sage: Linear_Expression(10).is_zero()
False
```

space_dimension ()

Return the dimension of the vector space necessary for the linear expression.

OUTPUT:

Integer.

```
sage: from sage.libs.ppl import Variable
sage: x = Variable(0)
sage: y = Variable(1)
sage: ( x+y+1 ).space_dimension()
```

```
2
sage: ( x+y ).space_dimension()
2
sage: ( y+1 ).space_dimension()
2
sage: ( x +1 ).space_dimension()
1
sage: ( y+1-y ).space_dimension()
2
```

class sage.libs.ppl. MIP_Problem

Bases: sage.libs.ppl._mutable_or_immutable

wrapper for PPL's MIP_Problem class

An object of the class MIP_Problem represents a Mixed Integer (Linear) Program problem.

INPUT:

- •dim integer
- •args an array of the defining data of the MIP_Problem. For each element, any one of the following is accepted:
 - -A Constraint System.
 - -A Linear Expression.

OUTPUT:

A MIP Problem.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, Constraint_System, MIP_Problem
sage: x = Variable(0)
sage: y = Variable(1)
sage: cs = Constraint_System()
sage: cs.insert( x >= 0)
sage: cs.insert( y >= 0 )
sage: cs.insert( 3 * x + 5 * y <= 10 )
sage: m = MIP_Problem(2, cs, x + y)
sage: m.optimal_value()
10/3
sage: m.optimizing_point()
point(10/3, 0/3)</pre>
```

OK ()

Check if all the invariants are satisfied.

OUTPUT:

True if and only if self satisfies all the invariants.

```
sage: from sage.libs.ppl import Variable, Constraint_System, MIP_Problem
sage: x = Variable(0)
sage: y = Variable(1)
sage: m = MIP_Problem()
sage: m.add_space_dimensions_and_embed(2)
sage: m.add_constraint(x >= 0)
```

```
sage: m.OK()
True
```

add_constraint (c)

Adds a copy of constraint c to the MIP problem.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, Constraint_System, MIP_Problem
sage: x = Variable(0)
sage: y = Variable(1)
sage: m = MIP_Problem()
sage: m.add_space_dimensions_and_embed(2)
sage: m.add_constraint(x >= 0)
sage: m.add_constraint(y >= 0)
sage: m.add_constraint(3 * x + 5 * y <= 10)
sage: m.set_objective_function(x + y)
sage: m.optimal_value()
10/3</pre>
```

add constraints (cs)

Adds a copy of the constraints in cs to the MIP problem.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, Constraint_System, MIP_Problem
sage: x = Variable(0)
sage: y = Variable(1)
sage: cs = Constraint_System()
sage: cs.insert( x >= 0)
sage: cs.insert( y >= 0 )
sage: cs.insert( 3 * x + 5 * y <= 10 )
sage: m = MIP_Problem(2)
sage: m.set_objective_function(x + y)
sage: m.add_constraints(cs)
sage: m.optimal_value()
10/3</pre>
```

add space dimensions and embed (m)

Adds m new space dimensions and embeds the old MIP problem in the new vector space.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, Constraint_System, MIP_Problem
sage: x = Variable(0)
sage: y = Variable(1)
sage: cs = Constraint_System()
sage: cs.insert( x >= 0)
sage: cs.insert( y >= 0 )
sage: cs.insert( 3 * x + 5 * y <= 10 )
sage: m = MIP_Problem(2, cs, x + y)
sage: m.add_space_dimensions_and_embed(5)
sage: m.space_dimension()</pre>
```

add_to_integer_space_dimensions (i_vars)

Sets the variables whose indexes are in set $i_v ars$ to be integer space dimensions.

```
sage: from sage.libs.ppl import Variable, Variables_Set, Constraint_System,

→MIP_Problem
sage: x = Variable(0)
sage: y = Variable(1)
sage: cs = Constraint_System()
sage: cs.insert( x >= 0)
sage: cs.insert( y >= 0 )
sage: cs.insert( 3 * x + 5 * y <= 10 )
sage: m = MIP_Problem(2)
sage: m.set_objective_function(x + y)
sage: m.add_constraints(cs)
sage: i_vars = Variables_Set(x, y)
sage: m.add_to_integer_space_dimensions(i_vars)
sage: m.optimal_value()
</pre>
```

clear ()

Reset the MIP_Problem to be equal to the trivial MIP_Problem.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, Constraint_System, MIP_Problem
sage: x = Variable(0)
sage: y = Variable(1)
sage: cs = Constraint_System()
sage: cs.insert( x >= 0)
sage: cs.insert( y >= 0 )
sage: cs.insert( 3 * x + 5 * y <= 10 )
sage: m = MIP_Problem(2, cs, x + y)
sage: m.objective_function()
x0+x1
sage: m.clear()
sage: m.objective_function()
0</pre>
```

evaluate_objective_function (evaluating_point)

Return the result of evaluating the objective function on evaluating_point. ValueError thrown if self and evaluating_point are dimension-incompatible or if the generator evaluating_point is not a point.

```
sage: from sage.libs.ppl import Variable, Constraint_System, MIP_Problem,...
→Generator
sage: x = Variable(0)
sage: y = Variable(1)
sage: m = MIP_Problem()
sage: m.add_space_dimensions_and_embed(2)
sage: m.add_constraint(x >= 0)
sage: m.add_constraint(y >= 0)
sage: m.add constraint(3 * x + 5 * v \le 10)
sage: m.set_objective_function(x + y)
sage: g = Generator.point(5 * x - 2 * y, 7)
sage: m.evaluate_objective_function(g)
3/7
sage: z = Variable(2)
sage: q = Generator.point(5 * x - 2 * z, 7)
sage: m.evaluate_objective_function(g)
Traceback (most recent call last):
```

```
ValueError: PPL::MIP_Problem::evaluate_objective_function(p, n, d):
*this and p are dimension incompatible.
```

is_satisfiable ()

Check if the MIP_Problem is satisfiable

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, Constraint_System, MIP_Problem
sage: x = Variable(0)
sage: y = Variable(1)
sage: m = MIP_Problem()
sage: m.add_space_dimensions_and_embed(2)
sage: m.add_constraint(x >= 0)
sage: m.add_constraint(y >= 0)
sage: m.add_constraint(3 * x + 5 * y <= 10)
sage: m.is_satisfiable()
True</pre>
```

objective_function ()

Return the optimal value of the MIP_Problem.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, Constraint_System, MIP_Problem
sage: x = Variable(0)
sage: y = Variable(1)
sage: cs = Constraint_System()
sage: cs.insert( x >= 0)
sage: cs.insert( y >= 0 )
sage: cs.insert( 3 * x + 5 * y <= 10 )
sage: m = MIP_Problem(2, cs, x + y)
sage: m.objective_function()
x0+x1</pre>
```

optimal value ()

Return the optimal value of the MIP_Problem. ValueError thrown if self does not have an optimizing point, i.e., if the MIP problem is unbounded or not satisfiable.

```
sage: from sage.libs.ppl import Variable, Constraint_System, MIP_Problem
sage: x = Variable(0)
sage: y = Variable(1)
sage: cs = Constraint_System()
sage: cs.insert( x >= 0 )
sage: cs.insert( y >= 0 )
sage: cs.insert( 3 * x + 5 * y <= 10 )
sage: m = MIP_Problem(2, cs, x + y)
sage: m.optimal_value()
10/3
sage: cs = Constraint_System()
sage: cs.insert( x >= 0 )
sage: m = MIP_Problem(1, cs, x + x )
sage: m.optimal_value()
Traceback (most recent call last):
...
```

```
ValueError: PPL::MIP_Problem::optimizing_point():
*this does not have an optimizing point.
```

optimization_mode ()

Return the optimization mode used in the MIP_Problem.

It will return "maximization" if the MIP_Problem was set to MAXIMIZATION mode, and "minimization" otherwise.

EXAMPLES:

```
sage: from sage.libs.ppl import MIP_Problem
sage: m = MIP_Problem()
sage: m.optimization_mode()
'maximization'
```

optimizing_point ()

Returns an optimal point for the MIP_Problem, if it exists.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, Constraint_System, MIP_Problem
sage: x = Variable(0)
sage: y = Variable(1)
sage: m = MIP_Problem()
sage: m.add_space_dimensions_and_embed(2)
sage: m.add_constraint(x >= 0)
sage: m.add_constraint(y >= 0)
sage: m.add_constraint(3 * x + 5 * y <= 10)
sage: m.set_objective_function(x + y)
sage: m.optimizing_point()
point(10/3, 0/3)</pre>
```

set_objective_function (obj)

Sets the objective function to obj.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, Constraint_System, MIP_Problem
sage: x = Variable(0)
sage: y = Variable(1)
sage: m = MIP_Problem()
sage: m.add_space_dimensions_and_embed(2)
sage: m.add_constraint(x >= 0)
sage: m.add_constraint(y >= 0)
sage: m.add_constraint(3 * x + 5 * y <= 10)
sage: m.set_objective_function(x + y)
sage: m.optimal_value()</pre>
```

set_optimization_mode (mode)

Sets the optimization mode to mode.

```
sage: from sage.libs.ppl import MIP_Problem
sage: m = MIP_Problem()
sage: m.optimization_mode()
'maximization'
```

```
sage: m.set_optimization_mode('minimization')
sage: m.optimization_mode()
'minimization'
```

solve ()

Optimizes the MIP_Problem

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, Constraint_System, MIP_Problem
sage: x = Variable(0)
sage: y = Variable(1)
sage: m = MIP_Problem()
sage: m.add_space_dimensions_and_embed(2)
sage: m.add_constraint(x >= 0)
sage: m.add_constraint(y >= 0)
sage: m.add_constraint(3 * x + 5 * y <= 10)
sage: m.set_objective_function(x + y)
sage: m.solve()
{'status': 'optimized'}</pre>
```

space_dimension ()

Return the space dimension of the MIP_Problem.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, Constraint_System, MIP_Problem
sage: x = Variable(0)
sage: y = Variable(1)
sage: cs = Constraint_System()
sage: cs.insert( x >= 0)
sage: cs.insert( y >= 0 )
sage: cs.insert( 3 * x + 5 * y <= 10 )
sage: m = MIP_Problem(2, cs, x + y)
sage: m.space_dimension()
2</pre>
```

class sage.libs.ppl. NNC_Polyhedron

Bases: sage.libs.ppl.Polyhedron

Wrapper for PPL's NNC_Polyhedron class.

An object of the class NNC_Polyhedron represents a not necessarily closed (NNC) convex polyhedron in the vector space.

Note: Since NNC polyhedra are a generalization of closed polyhedra, any object of the class $C_Polyhedron$ can be (explicitly) converted into an object of the class $NNC_Polyhedron$. The reason for defining two different classes is that objects of the class $C_Polyhedron$ are characterized by a more efficient implementation, requiring less time and memory resources.

INPUT:

- •arg the defining data of the polyhedron. Any one of the following is accepted:
 - -An non-negative integer. Depending on degenerate_element , either the space-filling or the empty polytope in the given dimension arg is constructed.

```
-A Constraint_System.
```

-A Generator_System.

```
-A single Constraint.-A single Generator.-A NNC_Polyhedron.-A C_Polyhedron.
```

•degenerate element - string, either 'universe' or 'empty'. Only used if arg is an integer.

OUTPUT:

A C_Polyhedron.

EXAMPLES:

```
sage: from sage.libs.ppl import Constraint, Constraint_System, Generator,...
→Generator_System, Variable, NNC_Polyhedron, point, ray, closure_point
sage: x = Variable(0)
sage: y = Variable(1)
sage: NNC_Polyhedron( 5*x-2*y > x+y-1)
A 2-dimensional polyhedron in QQ^2 defined as the convex hull of 1 point, 1...
→closure_point, 1 ray, 1 line
sage: cs = Constraint_System()
sage: cs.insert( x > 0 )
sage: cs.insert(y > 0)
sage: NNC_Polyhedron(cs)
A 2-dimensional polyhedron in QQ^2 defined as the convex hull of 1 point, 1,
→closure_point, 2 rays
sage: NNC_Polyhedron( point(x+y) )
A 0-dimensional polyhedron in QQ^2 defined as the convex hull of 1 point
sage: gs = Generator_System()
sage: gs.insert( point(-y) )
sage: gs.insert( closure_point(-x-y) )
sage: gs.insert( ray(x) )
sage: p = NNC_Polyhedron(gs); p
A 1-dimensional polyhedron in QQ^2 defined as the convex hull of 1 point, 1...
⇔closure_point, 1 ray
sage: p.minimized_constraints()
Constraint_System \{x1+1==0, x0+1>0\}
```

Note that, by convention, every polyhedron must contain a point:

```
sage: NNC_Polyhedron( closure_point(x+y) )
Traceback (most recent call last):
...
ValueError: PPL::NNC_Polyhedron::NNC_Polyhedron(gs):
*this is an empty polyhedron and
the non-empty generator system gs contains no points.
```

```
class sage.libs.ppl. Poly_Con_Relation
```

Bases: object

Wrapper for PPL's Poly_Con_Relation class.

INPUT/OUTPUT:

You must not construct $Poly_Con_Relation$ objects manually. You will usually get them from $relation_with()$. You can also get pre-defined relations from the class methods nothing(), $is_disjoint()$, $strictly_intersects()$, $is_included()$, and saturates().

```
sage: from sage.libs.ppl import Poly_Con_Relation
                   = Poly_Con_Relation.saturates(); saturates
sage: saturates
saturates
sage: is_included = Poly_Con_Relation.is_included(); is_included
is_included
sage: is_included.implies(saturates)
False
sage: saturates.implies(is_included)
False
sage: rels = []
sage: rels.append( Poly_Con_Relation.nothing() )
sage: rels.append( Poly_Con_Relation.is_disjoint() )
sage: rels.append( Poly_Con_Relation.strictly_intersects() )
sage: rels.append( Poly_Con_Relation.is_included() )
sage: rels.append( Poly_Con_Relation.saturates() )
sage: rels
[nothing, is_disjoint, strictly_intersects, is_included, saturates]
sage: from sage.matrix.constructor import matrix
sage: m = matrix(5, 5)
sage: for i, rel_i in enumerate(rels):
        for j, rel_j in enumerate(rels):
. . . . :
              m[i,j] = rel_i.implies(rel_j)
sage: m
[1 0 0 0 0]
[1 1 0 0 0]
[1 0 1 0 0]
[1 0 0 1 0]
[1 0 0 0 1]
```

OK (check_non_empty=False)

Check if all the invariants are satisfied.

EXAMPLES:

```
sage: from sage.libs.ppl import Poly_Con_Relation
sage: Poly_Con_Relation.nothing().OK()
True
```

ascii_dump ()

Write an ASCII dump to stderr.

EXAMPLES:

implies (y)

Test whether self implies y.

INPUT:

```
\bullety -a Poly\_Con\_Relation.
```

OUTPUT:

Boolean. True if and only if self implies y.

EXAMPLES:

```
sage: from sage.libs.ppl import Poly_Con_Relation
sage: nothing = Poly_Con_Relation.nothing()
sage: nothing.implies( nothing )
True
```

static is_disjoint ()

Return the assertion "The polyhedron and the set of points satisfying the constraint are disjoint".

OUTPUT:

A Poly_Con_Relation.

EXAMPLES:

```
sage: from sage.libs.ppl import Poly_Con_Relation
sage: Poly_Con_Relation.is_disjoint()
is_disjoint
```

static is included ()

Return the assertion "The polyhedron is included in the set of points satisfying the constraint".

OUTPUT:

A Poly_Con_Relation.

EXAMPLES:

```
sage: from sage.libs.ppl import Poly_Con_Relation
sage: Poly_Con_Relation.is_included()
is_included
```

static nothing ()

Return the assertion that says nothing.

OUTPUT:

A Poly_Con_Relation.

EXAMPLES:

```
sage: from sage.libs.ppl import Poly_Con_Relation
sage: Poly_Con_Relation.nothing()
nothing
```

static saturates ()

Return the assertion "".

OUTPUT:

A Poly_Con_Relation.

```
sage: from sage.libs.ppl import Poly_Con_Relation
sage: Poly_Con_Relation.saturates()
saturates
```

static strictly_intersects ()

Return the assertion "The polyhedron intersects the set of points satisfying the constraint, but it is not included in it".

OUTPUT:

A Poly_Con_Relation.

EXAMPLES:

```
sage: from sage.libs.ppl import Poly_Con_Relation
sage: Poly_Con_Relation.strictly_intersects()
strictly_intersects
```

class sage.libs.ppl. Poly_Gen_Relation

Bases: object

Wrapper for PPL's Poly_Con_Relation class.

INPUT/OUTPUT:

You must not construct $Poly_Gen_Relation$ objects manually. You will usually get them from $relation_with()$. You can also get pre-defined relations from the class methods nothing() and subsumes().

EXAMPLES:

```
sage: from sage.libs.ppl import Poly_Gen_Relation
sage: nothing = Poly_Gen_Relation.nothing(); nothing
nothing
sage: subsumes = Poly_Gen_Relation.subsumes(); subsumes
subsumes
sage: nothing.implies( subsumes )
False
sage: subsumes.implies( nothing )
True
```

OK (check_non_empty=False)

Check if all the invariants are satisfied.

EXAMPLES:

```
sage: from sage.libs.ppl import Poly_Gen_Relation
sage: Poly_Gen_Relation.nothing().OK()
True
```

ascii_dump ()

Write an ASCII dump to stderr.

EXAMPLES:

implies (y)

Test whether self implies y.

INPUT:

```
•y -a Poly_Gen_Relation.
```

OUTPUT:

Boolean. True if and only if self implies y.

EXAMPLES:

```
sage: from sage.libs.ppl import Poly_Gen_Relation
sage: nothing = Poly_Gen_Relation.nothing()
sage: nothing.implies( nothing )
True
```

static nothing ()

Return the assertion that says nothing.

OUTPUT:

A Poly_Gen_Relation.

EXAMPLES:

```
sage: from sage.libs.ppl import Poly_Gen_Relation
sage: Poly_Gen_Relation.nothing()
nothing
```

static subsumes ()

Return the assertion "Adding the generator would not change the polyhedron".

OUTPUT:

A Poly_Gen_Relation.

EXAMPLES:

```
sage: from sage.libs.ppl import Poly_Gen_Relation
sage: Poly_Gen_Relation.subsumes()
subsumes
```

class sage.libs.ppl. Polyhedron

Bases: sage.libs.ppl._mutable_or_immutable

Wrapper for PPL's Polyhedron class.

An object of the class Polyhedron represents a convex polyhedron in the vector space.

A polyhedron can be specified as either a finite system of constraints or a finite system of generators (see Section Representations of Convex Polyhedra) and it is always possible to obtain either representation. That is, if we know the system of constraints, we can obtain from this the system of generators that define the same polyhedron and vice versa. These systems can contain redundant members: in this case we say that they are not in the minimal form.

INPUT/OUTPUT:

This is an abstract base for $C_Polyhedron$ and $NNC_Polyhedron$. You cannot instantiate this class.

```
OK ( check non empty=False)
```

Check if all the invariants are satisfied.

The check is performed so as to intrude as little as possible. If the library has been compiled with run-time assertions enabled, error messages are written on std::cerr in case invariants are violated. This is useful for the purpose of debugging the library.

INPUT:

•check_not_empty - boolean. True if and only if, in addition to checking the invariants, self must be checked to be not empty.

OUTPUT:

True if and only if self satisfies all the invariants and either check_not_empty is False or self is not empty.

EXAMPLES:

```
sage: from sage.libs.ppl import Linear_Expression, Variable
sage: x = Variable(0)
sage: y = Variable(1)
sage: e = 3*x+2*y+1
sage: e.OK()
True
```

add_constraint (c)

Add a constraint to the polyhedron.

Adds a copy of constraint c to the system of constraints of self, without minimizing the result.

See alse add_constraints().

INPUT:

•c - the Constraint that will be added to the system of constraints of self.

OUTPUT:

This method modifies the polyhedron self and does not return anything.

Raises a ValueError if self and the constraint c are topology-incompatible or dimension-incompatible.

```
sage: from sage.libs.ppl import Variable, C_Polyhedron
  sage: x = Variable(0)
  sage: y = Variable(1)
  sage: p = C_Polyhedron(y>=0)
  sage: p.add_constraint( x>=0 )
We just added a 1-d constraint to a 2-d polyhedron, this is
fine. The other way is not::
  sage: p = C_Polyhedron(x>=0)
  sage: p.add_constraint( y>=0 )
  Traceback (most recent call last):
  ValueError: PPL::C_Polyhedron::add_constraint(c):
  this->space_dimension() == 1, c.space_dimension() == 2.
The constraint must also be topology-compatible, that is,
:class:`C_Polyhedron` only allows non-strict inequalities::
  sage: p = C_Polyhedron(x>=0)
  sage: p.add_constraint( x< 1 )</pre>
  Traceback (most recent call last):
```

```
ValueError: PPL::C_Polyhedron::add_constraint(c):
    c is a strict inequality.
```

${\tt add_constraints}$ (cs)

Add constraints to the polyhedron.

Adds a copy of constraints in cs to the system of constraints of self, without minimizing the result.

See alse add constraint().

INPUT:

•cs - the Constraint_System that will be added to the system of constraints of self.

OUTPUT:

This method modifies the polyhedron self and does not return anything.

Raises a ValueError if self and the constraints in cs are topology-incompatible or dimension-incompatible.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, C_Polyhedron, Constraint_System
  sage: x = Variable(0)
  sage: y = Variable(1)
  sage: cs = Constraint_System()
  sage: cs.insert(x>=0)
  sage: cs.insert(y>=0)
  sage: p = C_Polyhedron(y <= 1)
  sage: p.add_constraints(cs)
We just added a 1-d constraint to a 2-d polyhedron, this is
fine. The other way is not::
  sage: p = C_Polyhedron(x<=1)
  sage: p.add_constraints(cs)
  Traceback (most recent call last):
  ValueError: PPL::C_Polyhedron::add_recycled_constraints(cs):
  this->space_dimension() == 1, cs.space_dimension() == 2.
The constraints must also be topology-compatible, that is,
:class:`C_Polyhedron` only allows non-strict inequalities::
  sage: p = C_Polyhedron(x>=0)
  sage: p.add_constraints( Constraint_System(x<0) )</pre>
  Traceback (most recent call last):
  ValueError: PPL::C_Polyhedron::add_recycled_constraints(cs):
  cs contains strict inequalities.
```

add_generator (g)

Add a generator to the polyhedron.

Adds a copy of constraint c to the system of generators of self, without minimizing the result.

INPUT:

 \bullet g - the Generator that will be added to the system of Generators of self.

OUTPUT:

This method modifies the polyhedron self and does not return anything.

Raises a ValueError if self and the generator g are topology-incompatible or dimension-incompatible, or if self is an empty polyhedron and g is not a point.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, C_Polyhedron, point, closure_
→point, ray
  sage: x = Variable(0)
  sage: y = Variable(1)
  sage: p = C_Polyhedron(1, 'empty')
  sage: p.add_generator( point(0*x) )
We just added a 1-d generator to a 2-d polyhedron, this is
fine. The other way is not::
  sage: p = C_Polyhedron(1, 'empty')
  sage: p.add_generator( point(0*y) )
  Traceback (most recent call last):
  ValueError: PPL::C_Polyhedron::add_generator(g):
  this->space_dimension() == 1, q.space_dimension() == 2.
The constraint must also be topology-compatible, that is,
:class:`C_Polyhedron` does not allow :func:`closure_point`
generators::
  sage: p = C_Polyhedron(point(0*x+0*y))
  sage: p.add_generator( closure_point(0*x) )
  Traceback (most recent call last):
  ValueError: PPL::C_Polyhedron::add_generator(g):
  g is a closure point.
```

Finally, ever non-empty polyhedron must have at least one point generator:

```
sage: p = C_Polyhedron(3, 'empty')
sage: p.add_generator( ray(x) )
Traceback (most recent call last):
...
ValueError: PPL::C_Polyhedron::add_generator(g):
*this is an empty polyhedron and g is not a point.
```

add_generators (gs)

Add generators to the polyhedron.

Adds a copy of the generators in qs to the system of generators of self, without minimizing the result.

```
See alse add_generator().
```

INPUT:

 \bullet gs - the ${\it Generator_System}$ that will be added to the system of constraints of ${\it self}$.

OUTPUT:

This method modifies the polyhedron self and does not return anything.

Raises a ValueError if self and one of the generators in gs are topology-incompatible or dimension-incompatible, or if self is an empty polyhedron and gs does not contain a point.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, C_Polyhedron, Generator_System,...
→point, ray, closure_point
  sage: x = Variable(0)
  sage: y = Variable(1)
  sage: gs = Generator_System()
  sage: gs.insert(point(0*x+0*y))
  sage: gs.insert(point(1*x+1*y))
  sage: p = C_Polyhedron(2, 'empty')
  sage: p.add_generators(gs)
We just added a 1-d constraint to a 2-d polyhedron, this is
fine. The other way is not::
  sage: p = C_Polyhedron(1, 'empty')
  sage: p.add_generators(gs)
  Traceback (most recent call last):
  ValueError: PPL::C_Polyhedron::add_recycled_generators(gs):
  this->space_dimension() == 1, gs.space_dimension() == 2.
The constraints must also be topology-compatible, that is,
:class:`C_Polyhedron` does not allow :func:`closure_point`
generators::
  sage: p = C_Polyhedron(point(0*x+0*y))
  sage: p.add_generators( Generator_System(closure_point(x) ))
  Traceback (most recent call last):
  ValueError: PPL::C_Polyhedron::add_recycled_generators(gs):
  gs contains closure points.
```

$add_space_dimensions_and_embed$ (m)

Add m new space dimensions and embed self in the new vector space.

The new space dimensions will be those having the highest indexes in the new polyhedron, which is characterized by a system of constraints in which the variables running through the new dimensions are not constrained. For instance, when starting from the polyhedron P and adding a third space dimension, the result will be the polyhedron

$$\left\{ (x, y, z)^T \in \mathbf{R}^3 \middle| (x, y)^T \in P \right\}$$

INPUT:

•m - integer.

OUTPUT:

This method assigns the embedded polyhedron to self and does not return anything.

Raises a ValueError if adding m new space dimensions would cause the vector space to exceed dimension $self.max_space_dimension()$.

```
sage: from sage.libs.ppl import Variable, C_Polyhedron, point
sage: x = Variable(0)
sage: p = C_Polyhedron( point(3*x) )
sage: p.add_space_dimensions_and_embed(1)
```

```
sage: p.minimized_generators()
Generator_System {line(0, 1), point(3/1, 0/1)}
sage: p.add_space_dimensions_and_embed( p.max_space_dimension() )
Traceback (most recent call last):
...
ValueError: PPL::C_Polyhedron::add_space_dimensions_and_embed(m):
adding m new space dimensions exceeds the maximum allowed space dimension.
```

add_space_dimensions_and_project (m)

Add m new space dimensions and embed self in the new vector space.

The new space dimensions will be those having the highest indexes in the new polyhedron, which is characterized by a system of constraints in which the variables running through the new dimensions are all constrained to be equal to 0. For instance, when starting from the polyhedron P and adding a third space dimension, the result will be the polyhedron

$$\left\{ (x, y, 0)^T \in \mathbf{R}^3 \middle| (x, y)^T \in P \right\}$$

INPUT:

•m - integer.

OUTPUT:

This method assigns the projected polyhedron to self and does not return anything.

Raises a ValueError if adding m new space dimensions would cause the vector space to exceed dimension self.max_space_dimension().

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, C_Polyhedron, point
sage: x = Variable(0)
sage: p = C_Polyhedron( point(3*x) )
sage: p.add_space_dimensions_and_project(1)
sage: p.minimized_generators()
Generator_System {point(3/1, 0/1)}
sage: p.add_space_dimensions_and_project( p.max_space_dimension() )
Traceback (most recent call last):
...
ValueError: PPL::C_Polyhedron::add_space_dimensions_and_project(m):
adding m new space dimensions exceeds the maximum allowed space dimension.
```

affine_dimension ()

Return the affine dimension of self.

OUTPUT:

An integer. Returns 0 if self is empty. Otherwise, returns the affine dimension of self.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, C_Polyhedron
sage: x = Variable(0)
sage: y = Variable(1)
sage: p = C_Polyhedron( 5*x-2*y == x+y-1 )
sage: p.affine_dimension()
1
```

ascii_dump ()

Write an ASCII dump to stderr.

EXAMPLES:

```
sage: sage_cmd = 'from sage.libs.ppl import C_Polyhedron, Variable\n'
sage: sage_cmd += 'x = Variable(0) \n'
sage: sage_cmd += 'y = Variable(1) \n'
sage: sage_cmd += 'p = C_Polyhedron(3*x+2*y==1)\n'
sage: sage_cmd += 'p.minimized_generators() \n'
sage: sage_cmd += 'p.ascii_dump() \n'
sage: from sage.tests.cmdline import test_executable
sage: (out, err, ret) = test_executable(['sage', '-c', sage_cmd],_
→timeout=100) # long time, indirect doctest
sage: print(err) # long time
space_dim 2
-ZE -EM +CM +GM +CS +GS -CP -GP -SC +SG
con_sys (up-to-date)
topology NECESSARILY_CLOSED
2 x 2 SPARSE (sorted)
index_first_pending 2
size 3 - 1 \ 3 \ 2 = (C)
size 3 1 0 0 >= (C)
gen_sys (up-to-date)
topology NECESSARILY_CLOSED
2 x 2 DENSE (not_sorted)
index_first_pending 2
size 3 \ 0 \ 2 \ -3 \ L \ (C)
size 3 2 0 1 P (C)
sat_c
0 x 0
sat_g
2 x 2
0 0
0 1
```

bounds_from_above (expr)

Test whether the expr is bounded from above.

INPUT:

```
\bulletexpr -a Linear_Expression
```

OUTPUT:

Boolean. Returns True if and only if expr is bounded from above in self.

Raises a ValueError if expr and this are dimension-incompatible.

```
sage: from sage.libs.ppl import Variable, C_Polyhedron, Linear_Expression
sage: x = Variable(0); y = Variable(1)
sage: p = C_Polyhedron(y<=0)
sage: p.bounds_from_above(x+1)
False
sage: p.bounds_from_above(Linear_Expression(y))
True
sage: p = C_Polyhedron(x<=0)
sage: p.bounds_from_above(y+1)
Traceback (most recent call last):</pre>
```

```
valueError: PPL::C_Polyhedron::bounds_from_above(e):
this->space_dimension() == 1, e.space_dimension() == 2.
```

bounds_from_below (expr)

Test whether the expr is bounded from above.

INPUT:

```
\operatorname{	ext{-expr}} - \operatorname{a} \operatorname{Linear\_Expression}
```

OUTPUT:

Boolean. Returns True if and only if expr is bounded from above in self.

Raises a ValueError if expr and this are dimension-incompatible.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, C_Polyhedron, Linear_Expression
sage: x = Variable(0); y = Variable(1)
sage: p = C_Polyhedron(y>=0)
sage: p.bounds_from_below(x+1)
False
sage: p.bounds_from_below(Linear_Expression(y))
True
sage: p = C_Polyhedron(x<=0)
sage: p.bounds_from_below(y+1)
Traceback (most recent call last):
...
ValueError: PPL::C_Polyhedron::bounds_from_below(e):
this->space_dimension() == 1, e.space_dimension() == 2.
```

$concatenate_assign(y)$

Assign to self the concatenation of self and y.

This function returns the Cartesian product of self and y.

Viewing a polyhedron as a set of tuples (its points), it is sometimes useful to consider the set of tuples obtained by concatenating an ordered pair of polyhedra. Formally, the concatenation of the polyhedra P and Q (taken in this order) is the polyhedron such that

$$R = \left\{ (x_0, \dots, x_{n-1}, y_0, \dots, y_{m-1})^T \in \mathbf{R}^{n+m} \middle| (x_0, \dots, x_{n-1})^T \in P, (y_0, \dots, y_{m-1})^T \in Q \right\}$$

Another way of seeing it is as follows: first embed polyhedron P into a vector space of dimension n+m and then add a suitably renamed-apart version of the constraints defining Q.

INPUT:

```
•m - integer.
```

OUTPUT:

This method assigns the concatenated polyhedron to self and does not return anything.

Raises a ValueError if self and y are topology-incompatible or if adding y.space_dimension() new space dimensions would cause the vector space to exceed dimension self.max_space_dimension().

```
sage: from sage.libs.ppl import Variable, C_Polyhedron, NNC_Polyhedron, point
sage: x = Variable(0)
sage: p1 = C_Polyhedron( point(1*x) )
sage: p2 = C_Polyhedron( point(2*x) )
sage: p1.concatenate_assign(p2)
sage: p1.minimized_generators()
Generator_System {point(1/1, 2/1)}
```

The polyhedra must be topology-compatible and not exceed the maximum space dimension:

```
sage: p1.concatenate_assign( NNC_Polyhedron(1, 'universe') )
Traceback (most recent call last):
...
ValueError: PPL::C_Polyhedron::concatenate_assign(y):
y is a NNC_Polyhedron.
sage: p1.concatenate_assign( C_Polyhedron(p1.max_space_dimension(), 'empty') )
Traceback (most recent call last):
...
ValueError: PPL::C_Polyhedron::concatenate_assign(y):
concatenation exceeds the maximum allowed space dimension.
```

constrains (var)

Test whether var is constrained in self.

INPUT:

```
•var -a Variable.
```

OUTPUT:

Boolean. Returns True if and only if var is constrained in self.

Raises a ValueError if var is not a space dimension of self.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, C_Polyhedron
sage: x = Variable(0)
sage: p = C_Polyhedron(1, 'universe')
sage: p.constrains(x)
False
sage: p = C_Polyhedron(x>=0)
sage: p.constrains(x)
True
sage: y = Variable(1)
sage: p.constrains(y)
Traceback (most recent call last):
...
ValueError: PPL::C_Polyhedron::constrains(v):
this->space_dimension() == 1, v.space_dimension() == 2.
```

constraints ()

Returns the system of constraints.

See also $minimized_constraints()$.

OUTPUT:

A Constraint_System.

```
sage: from sage.libs.ppl import Variable, C_Polyhedron
sage: x = Variable(0)
sage: y = Variable(1)
sage: p = C_Polyhedron( y>=0 )
sage: p.add_constraint( x>=0 )
sage: p.add_constraint( x+y>=0 )
sage: p.constraints()
Constraint_System {x1>=0, x0>=0, x0+x1>=0}
sage: p.minimized_constraints()
Constraint_System {x1>=0, x0>=0}
```

contains (y)

Test whether self contains y.

INPUT:

```
•y - a Polyhedron.
```

OUTPUT:

Boolean. Returns True if and only if self contains y.

Raises a ValueError if self and y are topology-incompatible or dimension-incompatible.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, C_Polyhedron, NNC_Polyhedron
sage: x = Variable(0)
sage: y = Variable(1)
sage: p0 = C_Polyhedron( x>=0 )
sage: p1 = C_Polyhedron( x>=1 )
sage: p0.contains(p1)
True
sage: p1.contains(p0)
False
```

Errors are raised if the dimension or topology is not compatible:

```
sage: p0.contains(C_Polyhedron(y>=0))
Traceback (most recent call last):
...
ValueError: PPL::C_Polyhedron::contains(y):
this->space_dimension() == 1, y.space_dimension() == 2.
sage: p0.contains(NNC_Polyhedron(x>0))
Traceback (most recent call last):
...
ValueError: PPL::C_Polyhedron::contains(y):
y is a NNC_Polyhedron.
```

contains_integer_point ()

Test whether self contains an integer point.

OUTPUT:

Boolean. Returns True if and only if self contains an integer point.

```
sage: from sage.libs.ppl import Variable, NNC_Polyhedron
sage: x = Variable(0)
sage: p = NNC_Polyhedron(x>0)
```

```
sage: p.add_constraint(x<1)
sage: p.contains_integer_point()
False
sage: p.topological_closure_assign()
sage: p.contains_integer_point()
True</pre>
```

difference assign (y)

Assign to self the poly-difference of self and y.

For any pair of NNC polyhedra P_1 and P_2 the convex polyhedral difference (or poly-difference) of P_1 and P_2 is defined as the smallest convex polyhedron containing the set-theoretic difference $P_1 \setminus P_2$ of P_1 and P_2 .

In general, even if P_1 and P_2 are topologically closed polyhedra, their poly-difference may be a convex polyhedron that is not topologically closed. For this reason, when computing the poly-difference of two $C_Polyhedron$, the library will enforce the topological closure of the result.

INPUT:

```
•y -a Polyhedron
```

OUTPUT:

This method assigns the poly-difference to self and does not return anything.

Raises a ValueError if self and y are topology-incompatible or dimension-incompatible.

EXAMPLES:

The poly-difference of C_polyhedron is really its closure:

```
sage: p = C_Polyhedron( point(0*x) )
sage: p.add_generator( point(1*x) )
sage: p.poly_difference_assign(C_Polyhedron( point(0*x) ))
sage: p.minimized_constraints()
Constraint_System {x0>=0, -x0+1>=0}
```

 ${\tt self}$ and ${\tt y}$ must be dimension- and topology-compatible, or an exception is raised:

```
sage: y = Variable(1)
sage: p.poly_difference_assign( C_Polyhedron(y>=0) )
Traceback (most recent call last):
...
ValueError: PPL::C_Polyhedron::poly_difference_assign(y):
this->space_dimension() == 1, y.space_dimension() == 2.
sage: p.poly_difference_assign( NNC_Polyhedron(x+y<1) )
Traceback (most recent call last):
...
ValueError: PPL::C_Polyhedron::poly_difference_assign(y):
y is a NNC_Polyhedron.</pre>
```

drop_some_non_integer_points ()

Possibly tighten self by dropping some points with non-integer coordinates.

The modified polyhedron satisfies:

- •it is (not necessarily strictly) contained in the original polyhedron.
- •integral vertices (generating points with integer coordinates) of the original polyhedron are not removed.

Note: The modified polyhedron is not neccessarily a lattice polyhedron; Some vertices will, in general, still be rational. Lattice points interior to the polyhedron may be lost in the process.

EXAMPLES:

generators ()

Returns the system of generators.

See also minimized_generators().

OUTPUT:

A Generator_System.

EXAMPLES:

$intersection_assign(y)$

Assign to self the intersection of self and y.

INPUT:

```
•y - a Polyhedron
```

OUTPUT:

This method assigns the intersection to self and does not return anything.

Raises a ValueError if self and y are topology-incompatible or dimension-incompatible.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, C_Polyhedron, NNC_Polyhedron
sage: x = Variable(0)
sage: y = Variable(1)
sage: p = C_Polyhedron(1*x+0*y >= 0)
sage: p.intersection_assign( C_Polyhedron(y>=0) )
sage: p.constraints()
Constraint_System \{x0>=0, x1>=0\}
sage: z = Variable(2)
sage: p.intersection_assign( C_Polyhedron(z>=0) )
Traceback (most recent call last):
ValueError: PPL::C_Polyhedron::intersection_assign(y):
this->space_dimension() == 2, y.space_dimension() == 3.
sage: p.intersection_assign( NNC_Polyhedron(x+y<1) )</pre>
Traceback (most recent call last):
ValueError: PPL::C_Polyhedron::intersection_assign(y):
y is a NNC_Polyhedron.
```

is_bounded ()

Test whether self is bounded.

OUTPUT:

Boolean. Returns True if and only if self is a bounded polyhedron.

EXAMPLES:

is_discrete ()

Test whether self is discrete.

OUTPUT:

Boolean. Returns True if and only if self is discrete.

```
sage: from sage.libs.ppl import Variable, C_Polyhedron, point, ray
sage: x = Variable(0); y = Variable(1)
sage: p = C_Polyhedron( point(1*x+2*y) )
sage: p.is_discrete()
True
sage: p.add_generator( point(x) )
```

```
sage: p.is_discrete()
False
```

is_disjoint_from (y)

Tests whether self and y are disjoint.

INPUT:

```
•y - a Polyhedron.
```

OUTPUT:

Boolean. Returns True if and only if self and y are disjoint.

Rayises a ValueError if self and y are topology-incompatible or dimension-incompatible.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, C_Polyhedron, NNC_Polyhedron
sage: x = Variable(0); y = Variable(1)
sage: C_Polyhedron(x<=0).is_disjoint_from( C_Polyhedron(x>=1) )
True
```

This is not allowed:

```
sage: x = Variable(0); y = Variable(1)
sage: poly_1d = C_Polyhedron(x<=0)
sage: poly_2d = C_Polyhedron(x+0*y>=1)
sage: poly_1d.is_disjoint_from(poly_2d)
Traceback (most recent call last):
...
ValueError: PPL::C_Polyhedron::intersection_assign(y):
this->space_dimension() == 1, y.space_dimension() == 2.
```

Nor is this:

```
sage: x = Variable(0); y = Variable(1)
sage: c_poly = C_Polyhedron( x <= 0 )
sage: nnc_poly = NNC_Polyhedron( x > 0 )
sage: c_poly.is_disjoint_from(nnc_poly)
Traceback (most recent call last):
...
ValueError: PPL::C_Polyhedron::intersection_assign(y):
y is a NNC_Polyhedron.
sage: NNC_Polyhedron(c_poly).is_disjoint_from(nnc_poly)
True
```

is_empty()

Test if self is an empty polyhedron.

OUTPUT:

Boolean.

```
sage: from sage.libs.ppl import C_Polyhedron
sage: C_Polyhedron(3, 'empty').is_empty()
True
sage: C_Polyhedron(3, 'universe').is_empty()
False
```

is_topologically_closed ()

Tests if self is topologically closed.

OUTPUT:

Returns True if and only if self is a topologically closed subset of the ambient vector space.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, C_Polyhedron, NNC_Polyhedron
sage: x = Variable(0); y = Variable(1)
sage: C_Polyhedron(3, 'universe').is_topologically_closed()
True
sage: C_Polyhedron( x>=1 ).is_topologically_closed()
True
sage: NNC_Polyhedron( x>1 ).is_topologically_closed()
False
```

is universe()

Test if self is a universe (space-filling) polyhedron.

OUTPUT:

Boolean.

EXAMPLES:

```
sage: from sage.libs.ppl import C_Polyhedron
sage: C_Polyhedron(3, 'empty').is_universe()
False
sage: C_Polyhedron(3, 'universe').is_universe()
True
```

max_space_dimension()

Return the maximum space dimension all kinds of Polyhedron can handle.

OUTPUT:

Integer.

EXAMPLES:

```
sage: from sage.libs.ppl import C_Polyhedron
sage: C_Polyhedron(1, 'empty').max_space_dimension() # random output
1152921504606846974
sage: C_Polyhedron(1, 'empty').max_space_dimension()
357913940 # 32-bit
1152921504606846974 # 64-bit
```

maximize (expr)

Maximize expr.

INPUT:

```
•expr -a Linear_Expression.
```

OUTPUT:

A dictionary with the following keyword:value pair:

•'bounded': Boolean. Whether the linear expression expr is bounded from above on self.

If expr is bounded from above, the following additional keyword:value pairs are set to provide information about the supremum:

- 'sup_n': Integer. The numerator of the supremum value.
- •'sup_d': Non-zero integer. The denominator of the supremum value.
- •'maximum': Boolean. True if and only if the supremum is also the maximum value.
- 'generator': a Generator. A point or closure point where expr reaches its supremum value.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, C_Polyhedron, NNC_Polyhedron,

→ Constraint_System, Linear_Expression
sage: x = Variable(0); y = Variable(1)
sage: cs = Constraint_System()
sage: cs.insert( x>=0 )
sage: cs.insert( y>=0 )
sage: cs.insert( 3*x+5*y<=10 )
sage: p = C_Polyhedron(cs)
sage: p.maximize( x+y )
{'bounded': True,
  'generator': point(10/3, 0/3),
  'maximum': True,
  'sup_d': 3,
  'sup_n': 10}</pre>
```

Unbounded case:

```
sage: cs = Constraint_System()
sage: cs.insert( x>0 )
sage: p = NNC_Polyhedron(cs)
sage: p.maximize( +x )
{'bounded': False}
sage: p.maximize( -x )
{'bounded': True,
   'generator': closure_point(0/1),
   'maximum': False,
   'sup_d': 1,
   'sup_n': 0}
```

minimize (expr)

Minimize expr.

INPUT:

```
\bulletexpr -a Linear_Expression.
```

OUTPUT:

A dictionary with the following keyword:value pair:

•'bounded': Boolean. Whether the linear expression expr is bounded from below on self.

If expr is bounded from below, the following additional keyword:value pairs are set to provide information about the infimum:

- •'inf_n': Integer. The numerator of the infimum value.
- •'inf_d': Non-zero integer. The denominator of the infimum value.
- •'minimum': Boolean. True if and only if the infimum is also the minimum value.
- 'generator': a Generator. A point or closure point where expr reaches its infimum value.

Unbounded case:

```
sage: cs = Constraint_System()
sage: cs.insert( x>0 )
sage: p = NNC_Polyhedron(cs)
sage: p.minimize( +x )
{'bounded': True,
   'generator': closure_point(0/1),
   'inf_d': 1,
   'inf_n': 0,
   'minimum': False}
sage: p.minimize( -x )
{'bounded': False}
```

minimized_constraints ()

Returns the minimized system of constraints.

See also constraints ().

OUTPUT:

A Constraint_System.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, C_Polyhedron
sage: x = Variable(0)
sage: y = Variable(1)
sage: p = C_Polyhedron( y>=0 )
sage: p.add_constraint( x>=0 )
sage: p.add_constraint( x+y>=0 )
sage: p.constraints()
Constraint_System {x1>=0, x0>=0, x0+x1>=0}
sage: p.minimized_constraints()
Constraint_System {x1>=0, x0>=0}
```

minimized_generators ()

Returns the minimized system of generators.

See also generators () .

OUTPUT:

A Generator System.

```
sage: from sage.libs.ppl import Variable, C_Polyhedron, point
sage: x = Variable(0)
sage: y = Variable(1)
sage: p = C_Polyhedron(3,'empty')
sage: p.add_generator( point(-x-y) )
sage: p.add_generator( point(0) )
sage: p.add_generator( point(+x+y) )
sage: p.generators()
Generator_System {point(-1/1, -1/1, 0/1), point(0/1, 0/1, 0/1), point(1/1, 1/-1, 0/1)}
sage: p.minimized_generators()
Generator_System {point(-1/1, -1/1, 0/1), point(1/1, 1/1, 0/1)}
```

${\tt poly_difference_assign}~(~y)$

Assign to self the poly-difference of self and y.

For any pair of NNC polyhedra P_1 and P_2 the convex polyhedral difference (or poly-difference) of P_1 and P_2 is defined as the smallest convex polyhedron containing the set-theoretic difference $P_1 \setminus P_2$ of P_1 and P_2 .

In general, even if P_1 and P_2 are topologically closed polyhedra, their poly-difference may be a convex polyhedron that is not topologically closed. For this reason, when computing the poly-difference of two $C_Polyhedron$, the library will enforce the topological closure of the result.

INPUT:

```
•y -a Polyhedron
```

OUTPUT:

This method assigns the poly-difference to self and does not return anything.

Raises a ValueError if self and y are topology-incompatible or dimension-incompatible.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, C_Polyhedron, point, closure_point,_
→NNC_Polyhedron
sage: x = Variable(0)
sage: p = NNC_Polyhedron( point(0*x) )
sage: p.add_generator( point(1*x) )
sage: p.poly_difference_assign(NNC_Polyhedron( point(0*x) ))
sage: p.minimized_constraints()
Constraint_System {-x0+1>=0, x0>0}
```

The poly-difference of C_polyhedron is really its closure:

```
sage: p = C_Polyhedron( point(0*x) )
sage: p.add_generator( point(1*x) )
sage: p.poly_difference_assign(C_Polyhedron( point(0*x) ))
sage: p.minimized_constraints()
Constraint_System {x0>=0, -x0+1>=0}
```

self and y must be dimension- and topology-compatible, or an exception is raised:

```
sage: y = Variable(1)
sage: p.poly_difference_assign( C_Polyhedron(y>=0) )
Traceback (most recent call last):
...
ValueError: PPL::C_Polyhedron::poly_difference_assign(y):
```

```
this->space_dimension() == 1, y.space_dimension() == 2.
sage: p.poly_difference_assign( NNC_Polyhedron(x+y<1) )
Traceback (most recent call last):
...
ValueError: PPL::C_Polyhedron::poly_difference_assign(y):
y is a NNC_Polyhedron.</pre>
```

poly_hull_assign (y)

Assign to self the poly-hull of self and y.

For any pair of NNC polyhedra P_1 and P_2 , the convex polyhedral hull (or poly-hull) of is the smallest NNC polyhedron that includes both P_1 and P_2 . The poly-hull of any pair of closed polyhedra in is also closed.

INPUT:

```
•y -a Polyhedron
```

OUTPUT:

This method assigns the poly-hull to self and does not return anything.

Raises a ValueError if self and y are topology-incompatible or dimension-incompatible.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, C_Polyhedron, point, NNC_Polyhedron
sage: x = Variable(0)
sage: y = Variable(1)
sage: p = C_Polyhedron( point(1*x+0*y) )
sage: p.poly_hull_assign(C_Polyhedron( point(0*x+1*y) ))
sage: p.generators()
Generator_System {point(0/1, 1/1), point(1/1, 0/1)}
```

self and y must be dimension- and topology-compatible, or an exception is raised:

```
sage: z = Variable(2)
sage: p.poly_hull_assign( C_Polyhedron(z>=0) )
Traceback (most recent call last):
...
ValueError: PPL::C_Polyhedron::poly_hull_assign(y):
this->space_dimension() == 2, y.space_dimension() == 3.
sage: p.poly_hull_assign( NNC_Polyhedron(x+y<1) )
Traceback (most recent call last):
...
ValueError: PPL::C_Polyhedron::poly_hull_assign(y):
y is a NNC_Polyhedron.</pre>
```

relation_with (arg)

Return the relations holding between the polyhedron self and the generator or constraint arg.

INPUT:

```
•arg - a Generator or a Constraint.
```

OUTPUT:

A Poly_Gen_Relation or a Poly_Con_Relation according to the type of the input.

Raises ValueError if self and the generator/constraint arg are dimension-incompatible.

```
sage: from sage.libs.ppl import Variable, C_Polyhedron, point, ray, Poly_Con_
→Relation
sage: x = Variable(0); y = Variable(1)
sage: p = C_Polyhedron(2, 'empty')
sage: p.add_generator( point(1*x+0*y) )
sage: p.add_generator( point(0*x+1*y) )
sage: p.minimized_constraints()
Constraint_System \{x0+x1-1==0, -x1+1>=0, x1>=0\}
sage: p.relation_with( point(1*x+1*y) )
nothing
sage: p.relation_with( point(1*x+1*y, 2) )
subsumes
sage: p.relation_with( x+y==-1 )
is_disjoint
sage: p.relation_with( x==y )
strictly_intersects
sage: p.relation_with( x+y<=1 )</pre>
is_included, saturates
sage: p.relation_with( x+y<1 )</pre>
is_disjoint, saturates
```

In a Sage program you will usually use relation_with() together with implies() or implies(), for example:

You can only get relations with dimension-compatible generators or constraints:

```
sage: z = Variable(2)
sage: p.relation_with( point(x+y+z) )
Traceback (most recent call last):
...
ValueError: PPL::C_Polyhedron::relation_with(g):
this->space_dimension() == 2, g.space_dimension() == 3.
sage: p.relation_with( z>0 )
Traceback (most recent call last):
...
ValueError: PPL::C_Polyhedron::relation_with(c):
this->space_dimension() == 2, c.space_dimension() == 3.
```

remove_higher_space_dimensions (new_dimension)

Remove the higher dimensions of the vector space so that the resulting space will have dimension new_dimension.

OUTPUT:

This method modifies self and does not return anything.

Raises a ValueError if new_dimensions is greater than the space dimension of self.

```
sage: from sage.libs.ppl import C_Polyhedron, Variable
sage: x = Variable(0)
sage: y = Variable(1)
sage: p = C_Polyhedron(3*x+0*y==2)
sage: p.remove_higher_space_dimensions(1)
sage: p.minimized_constraints()
```

```
Constraint_System {3*x0-2==0}
sage: p.remove_higher_space_dimensions(2)
Traceback (most recent call last):
...
ValueError: PPL::C_Polyhedron::remove_higher_space_dimensions(nd):
this->space_dimension() == 1, required space dimension == 2.
```

space_dimension ()

Return the dimension of the vector space enclosing self.

OUTPUT:

Integer.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, C_Polyhedron
sage: x = Variable(0)
sage: y = Variable(1)
sage: p = C_Polyhedron( 5*x-2*y >= x+y-1 )
sage: p.space_dimension()
2
```

strictly_contains (y)

Test whether self strictly contains y.

INPUT:

```
•y - a Polyhedron.
```

OUTPUT:

Boolean. Returns True if and only if self contains y and self does not equal y.

Raises a ValueError if self and y are topology-incompatible or dimension-incompatible.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, C_Polyhedron, NNC_Polyhedron
sage: x = Variable(0)
sage: y = Variable(1)
sage: p0 = C_Polyhedron( x>=0 )
sage: p1 = C_Polyhedron( x>=1 )
sage: p0.strictly_contains(p1)
True
sage: p1.strictly_contains(p0)
False
```

Errors are raised if the dimension or topology is not compatible:

```
sage: p0.strictly_contains(C_Polyhedron(y>=0))
Traceback (most recent call last):
...
ValueError: PPL::C_Polyhedron::contains(y):
this->space_dimension() == 1, y.space_dimension() == 2.
sage: p0.strictly_contains(NNC_Polyhedron(x>0))
Traceback (most recent call last):
...
ValueError: PPL::C_Polyhedron::contains(y):
y is a NNC_Polyhedron.
```

topological_closure_assign ()

Assign to self its topological closure.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, NNC_Polyhedron
sage: x = Variable(0)
sage: p = NNC_Polyhedron(x>0)
sage: p.is_topologically_closed()
False
sage: p.topological_closure_assign()
sage: p.is_topologically_closed()
True
sage: p.minimized_constraints()
Constraint_System {x0>=0}
```

unconstrain (var)

Compute the cylindrification of self with respect to space dimension var.

INPUT:

•var - a Variable. The space dimension that will be unconstrained. Exceptions:

OUTPUT:

This method assigns the cylindrification to self and does not return anything.

Raises a ValueError if var is not a space dimension of self.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, C_Polyhedron, point
sage: x = Variable(0)
sage: y = Variable(1)
sage: p = C_Polyhedron( point(x+y) ); p
A 0-dimensional polyhedron in QQ^2 defined as the convex hull of 1 point
sage: p.unconstrain(x); p
A 1-dimensional polyhedron in QQ^2 defined as the convex hull of 1 point, 1
ine
sage: z = Variable(2)
sage: p.unconstrain(z)
Traceback (most recent call last):
...
ValueError: PPL::C_Polyhedron::unconstrain(var):
this->space_dimension() == 2, required space dimension == 3.
```

$upper_bound_assign(y)$

Assign to self the poly-hull of self and y.

For any pair of NNC polyhedra P_1 and P_2 , the convex polyhedral hull (or poly-hull) of is the smallest NNC polyhedron that includes both P_1 and P_2 . The poly-hull of any pair of closed polyhedra in is also closed.

INPUT:

```
•y -a Polyhedron
```

OUTPUT:

This method assigns the poly-hull to self and does not return anything.

 $Raises\ a\ {\tt ValueError}\ if\ self\ and\ y\ are\ topology-incompatible\ or\ dimension-incompatible.$

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, C_Polyhedron, point, NNC_Polyhedron
sage: x = Variable(0)
sage: y = Variable(1)
sage: p = C_Polyhedron( point(1*x+0*y) )
sage: p.poly_hull_assign(C_Polyhedron( point(0*x+1*y) ))
sage: p.generators()
Generator_System {point(0/1, 1/1), point(1/1, 0/1)}
```

self and y must be dimension- and topology-compatible, or an exception is raised:

```
sage: z = Variable(2)
sage: p.poly_hull_assign( C_Polyhedron(z>=0) )
Traceback (most recent call last):
...
ValueError: PPL::C_Polyhedron::poly_hull_assign(y):
this->space_dimension() == 2, y.space_dimension() == 3.
sage: p.poly_hull_assign( NNC_Polyhedron(x+y<1) )
Traceback (most recent call last):
...
ValueError: PPL::C_Polyhedron::poly_hull_assign(y):
y is a NNC_Polyhedron.</pre>
```

class sage.libs.ppl. Variable

Bases: object

Wrapper for PPL's Variable class.

A dimension of the vector space.

An object of the class Variable represents a dimension of the space, that is one of the Cartesian axes. Variables are used as basic blocks in order to build more complex linear expressions. Each variable is identified by a non-negative integer, representing the index of the corresponding Cartesian axis (the first axis has index 0). The space dimension of a variable is the dimension of the vector space made by all the Cartesian axes having an index less than or equal to that of the considered variable; thus, if a variable has index i, its space dimension is i+1.

INPUT:

•i - integer. The index of the axis.

OUTPUT:

A Variable

EXAMPLES:

```
sage: from sage.libs.ppl import Variable
sage: x = Variable(123)
sage: x.id()
123
sage: x
x123
```

Note that the "meaning" of an object of the class Variable is completely specified by the integer index provided to its constructor: be careful not to be mislead by C++ language variable names. For instance, in the following example the linear expressions e1 and e2 are equivalent, since the two variables x and z denote the same Cartesian axis:

```
sage: x = Variable(0)
sage: y = Variable(1)
sage: z = Variable(0)
sage: e1 = x + y; e1
x0+x1
sage: e2 = y + z; e2
x0+x1
sage: e1 - e2
0
```

OK ()

Checks if all the invariants are satisfied.

OUTPUT:

Boolean.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable
sage: x = Variable(0)
sage: x.OK()
True
```

id()

Return the index of the Cartesian axis associated to the variable.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable
sage: x = Variable(123)
sage: x.id()
123
```

space_dimension ()

Return the dimension of the vector space enclosing self.

OUTPUT:

Integer. The returned value is self.id()+1.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable
sage: x = Variable(0)
sage: x.space_dimension()
1
```

```
class sage.libs.ppl. Variables_Set
```

Bases: object

Wrapper for PPL's Variables_Set class.

A set of variables' indexes.

EXAMPLES:

Build the empty set of variable indexes:

```
sage: from sage.libs.ppl import Variable, Variables_Set
sage: Variables_Set()
Variables_Set of cardinality 0
```

Build the singleton set of indexes containing the index of the variable:

```
sage: v123 = Variable(123)
sage: Variables_Set(v123)
Variables_Set of cardinality 1
```

Build the set of variables' indexes in the range from one variable to another variable:

```
sage: v127 = Variable(127)
sage: Variables_Set(v123,v127)
Variables_Set of cardinality 5
```

OK ()

Checks if all the invariants are satisfied.

OUTPUT:

Boolean.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, Variables_Set
sage: v123 = Variable(123)
sage: S = Variables_Set(v123)
sage: S.OK()
True
```

ascii_dump()

Write an ASCII dump to stderr.

EXAMPLES:

insert (v)

Inserts the index of variable v into the set.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, Variables_Set
sage: S = Variables_Set()
sage: v123 = Variable(123)
sage: S.insert(v123)
sage: S.space_dimension()
124
```

space_dimension ()

Returns the dimension of the smallest vector space enclosing all the variables whose indexes are in the set.

OUTPUT:

Integer.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, Variables_Set
sage: v123 = Variable(123)
sage: S = Variables_Set(v123)
sage: S.space_dimension()
124
```

sage.libs.ppl. closure_point (expression=0, divisor=1)

Constuct a closure point.

See Generator.closure_point() for documentation.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, closure_point
sage: y = Variable(1)
sage: closure_point(2*y, 5)
closure_point(0/5, 2/5)
```

sage.libs.ppl. equation (expression)

Constuct an equation.

INPUT:

•expression - a Linear_Expression.

OUTPUT:

The equation expression == 0.

EXAMPLES:

```
sage: from sage.libs.ppl import Variable, equation
sage: y = Variable(1)
sage: 2*y+1 == 0
2*x1+1==0
sage: equation(2*y+1)
2*x1+1==0
```

sage.libs.ppl. inequality (expression)

Constuct an inequality.

INPUT:

 \bullet expression -a Linear_Expression.

OUTPUT:

The inequality expression >= 0.

```
sage: from sage.libs.ppl import Variable, inequality
sage: y = Variable(1)
sage: 2*y+1 >= 0
2*x1+1>=0
sage: inequality(2*y+1)
2*x1+1>=0
```

```
sage.libs.ppl. line (expression)
    Constuct a line.
    See Generator.line() for documentation.
    EXAMPLES:
    sage: from sage.libs.ppl import Variable, line
     sage: y = Variable(1)
     sage: line(2*y)
    line(0, 1)
sage.libs.ppl. point (expression=0, divisor=1)
    Constuct a point.
    See Generator.point() for documentation.
    EXAMPLES:
    sage: from sage.libs.ppl import Variable, point
    sage: y = Variable(1)
    sage: point (2*y, 5)
    point (0/5, 2/5)
sage.libs.ppl. ray (expression)
    Constuct a ray.
    See Generator.ray() for documentation.
    EXAMPLES:
    sage: from sage.libs.ppl import Variable, ray
    sage: y = Variable(1)
    sage: ray(2*y)
    ray(0, 1)
sage.libs.ppl. strict_inequality (expression)
    Constuct a strict inequality.
    INPUT:
        •expression - a Linear_Expression.
    OUTPUT:
    The inequality expression > 0.
    EXAMPLES:
     sage: from sage.libs.ppl import Variable, strict_inequality
    sage: y = Variable(1)
    sage: 2*y+1 > 0
    2*x1+1>0
```

sage: strict_inequality(2*y+1)

 $2 \times x1 + 1 > 0$

CHAPTER

TWENTY

LINBOX INTERFACE

```
class sage.libs.linbox.linbox.Linbox_integer_dense
    Bases: object

charpoly ()
    OUTPUT:
    coefficients of charpoly or minpoly as a Python list

det ()
    OUTPUT:
    determinant as a sage Integer

minpoly ()
    OUTPUT:
    coefficients of minpoly as a Python list

smithform ()

class sage.libs.linbox.linbox.Linbox_modn_sparse
    Bases: object
```

Sage Reference Manual: C/C++ Library Interfaces, Release 7.6

TWENTYONE

FLINT IMPORTS

sage.libs.flint.flint.free_flint_stack()

TWENTYTWO

FLINT FMPZ POLY CLASS WRAPPER

AUTHORS:

- Robert Bradshaw (2007-09-15) Initial version.
- William Stein (2007-10-02) update for new flint; add arithmetic and creation of coefficients of arbitrary size.

```
class sage.libs.flint.fmpz_poly. Fmpz_poly
    Bases: sage.structure.sage_object.SageObject
```

Construct a new fmpz_poly from a sequence, constant coefficient, or string (in the same format as it prints).

EXAMPLES:

```
sage: from sage.libs.flint.fmpz_poly import Fmpz_poly
sage: Fmpz_poly([1,2,3])
3  1  2  3
sage: Fmpz_poly(5)
1  5
sage: Fmpz_poly(str(Fmpz_poly([3,5,7])))
3  3  5  7
```

degree ()

The degree of self.

EXAMPLES:

```
sage: from sage.libs.flint.fmpz_poly import Fmpz_poly
sage: f = Fmpz_poly([1,2,3]); f
3  1 2 3
sage: f.degree()
2
sage: Fmpz_poly(range(1000)).degree()
999
sage: Fmpz_poly([2,0]).degree()
0
```

derivative ()

Return the derivative of self.

```
sage: from sage.libs.flint.fmpz_poly import Fmpz_poly
sage: f = Fmpz_poly([1,2,6])
sage: f.derivative().list() == [2, 12]
True
```

```
div rem ( other)
```

Return self / other, self, % other.

EXAMPLES:

```
sage: from sage.libs.flint.fmpz_poly import Fmpz_poly
sage: f = Fmpz_poly([1,3,4,5])
sage: g = f^23
sage: g.div_rem(f)[1]
0
sage: g.div_rem(f)[0] - f^22
0
sage: f = Fmpz_poly([1..10])
sage: g = Fmpz_poly([1,3,5])
sage: q, r = f.div_rem(g)
sage: q*f+r
17  1 2 3 4 4 4 10 11 17 18 22 26 30 23 26 18 20
sage: g
3  1 3 5
sage: q*g+r
10  1 2 3 4 5 6 7 8 9 10
```

left_shift (n)

Left shift self by n.

EXAMPLES:

```
sage: from sage.libs.flint.fmpz_poly import Fmpz_poly
sage: f = Fmpz_poly([1,2])
sage: f.left_shift(1).list() == [0,1,2]
True
```

list ()

Return self as a list of coefficients, lowest terms first.

EXAMPLES:

```
sage: from sage.libs.flint.fmpz_poly import Fmpz_poly
sage: f = Fmpz_poly([2,1,0,-1])
sage: f.list()
[2, 1, 0, -1]
```

pow_truncate (exp, n)

Return self raised to the power of exp mod x^n.

```
sage: from sage.libs.flint.fmpz_poly import Fmpz_poly
sage: f = Fmpz_poly([1,2])
sage: f.pow_truncate(10,3)
3  1 20 180
sage: f.pow_truncate(1000,3)
3  1 2000 1998000
```

```
pseudo_div ( other)
pseudo_div_rem ( other)
right_shift ( n)
    Right shift self by n.
```

EXAMPLES:

```
sage: from sage.libs.flint.fmpz_poly import Fmpz_poly
sage: f = Fmpz_poly([1,2])
sage: f.right_shift(1).list() == [2]
True
```

truncate (n)

Return the truncation of self at degree n.

```
sage: from sage.libs.flint.fmpz_poly import Fmpz_poly
sage: f = Fmpz_poly([1,1])
sage: g = f**10; g
11  1 10 45 120 210 252 210 120 45 10 1
sage: g.truncate(5)
5  1 10 45 120 210
```

Sage Reference Manual: C/C++ Library Interfaces, Release 7.6		

TWENTYTHREE

FLINT ARITHMETIC FUNCTIONS

```
sage.libs.flint.arith.bell_number (n)
```

Returns the n th Bell number.

EXAMPLES:

```
sage: from sage.libs.flint.arith import bell_number
sage: [bell_number(i) for i in range(10)]
[1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147]
sage: bell_number(10)
115975
sage: bell_number(40)
157450588391204931289324344702531067
sage: bell_number(100)
475853912767648336587907688413872078263636696868256114666163346375591144978924426226727240442177
```

sage.libs.flint.arith.bernoulli_number (n)

Return the n-th Bernoulli number.

EXAMPLES:

sage.libs.flint.arith. $\operatorname{dedekind_sum}$ (p,q)

Return the Dedekind sum s(p, q) where p and q are arbitrary integers.

EXAMPLES:

```
sage: from sage.libs.flint.arith import dedekind_sum
sage: dedekind_sum(4, 5)
-1/5
```

sage.libs.flint.arith.euler_number (n)

Return the Euler number of index n.

```
sage: from sage.libs.flint.arith import euler_number
sage: [euler_number(i) for i in range(8)]
[1, 0, -1, 0, 5, 0, -61, 0]
```

sage.libs.flint.arith.harmonic_number (n)

Returns the harmonic number H n.

EXAMPLES:

```
sage: from sage.libs.flint.arith import harmonic_number
sage: n = 500 + randint(0,500)
sage: bool( sum(1/k for k in range(1,n+1)) == harmonic_number(n) )
True
```

sage.libs.flint.arith.number_of_partitions (n)

Returns the number of partitions of the integer n .

```
sage: from sage.libs.flint.arith import number_of_partitions
sage: number_of_partitions(3)
3
sage: number_of_partitions(10)
42
sage: number_of_partitions(40)
37338
sage: number_of_partitions(100)
190569292
sage: number_of_partitions(100000)
274935105697756965126775163209863526881734293159800547582031259843021473281149641730550507416607
```

TWENTYFOUR

SYMMETRICA LIBRARY

```
sage.libs.symmetrica.symmetrica.bdg_symmetrica ( part, perm)
```

Calculates the irreduzible matrix representation D^part(perm), whose entries are of integral numbers.

REFERENCE: H. Boerner: Darstellungen von Gruppen, Springer 1955. pp. 104-107.

```
sage.libs.symmetrica.symmetrica.chartafel_symmetrica (n)
```

you enter the degree of the symmetric group, as INTEGER object and the result is a MATRIX object: the charactertable of the symmetric group of the given degree.

EXAMPLES:

sage.libs.symmetrica.symmetrica.charvalue_symmetrica (irred, cls, table=None)

you enter a PARTITION object part, labelling the irreducible character, you enter a PARTITION object class, labeling the class or class may be a PERMUTATION object, then result becomes the value of that character on that class or permutation. Note that the table may be NULL, in which case the value is computed, or it may be taken from a precalculated charactertable.

FIXME: add table parameter

```
sage.libs.symmetrica.symmetrica.compute elmsym with alphabet symmetrica (n.
                                                                                                length,
                                                                                                al-
                                                                                               pha-
                                                                                                bet='x'
     computes the expansion of a elementary symmetric function labeled by a INTEGER number as a POLYNOM
     erg. The object number may also be a PARTITION or a ELM SYM object. The INTEGER length specifies the
     length of the alphabet. Both routines are the same.
     EXAMPLES: sage:
                                                symmetrica.compute_elmsym_with_alphabet(2,2);
                                                                                                     a
          x0*x1
                                                      Polynomial
                                                                    Ring in
                                                                               x0,
                   sage:
                             a.parent()
                                         Multivariate
                                                                                                   In-
          teger
                  Ring
                           sage:
                                                 symmetrica.compute_elmsym_with_alphabet([2],2);
                                                                                                     а
          x0*x1
                   sage:
                              symmetrica.compute_elmsym_with_alphabet(3,2)
                                                                                              symmet-
          rica.compute_elmsym_with_alphabet([3,2,1],2) 0
sage.libs.symmetrica.symmetrica.compute_homsym_with_alphabet_symmetrica (n,
                                                                                                length,
                                                                                                al-
                                                                                               pha-
                                                                                                bet='x'
     computes the expansion of a homogenous (=complete) symmetric function labeled by a INTEGER number as a
     POLYNOM erg. The object number may also be a PARTITION or a HOM SYM object. The INTEGER laenge
     specifies the length of the alphabet. Both routines are the same.
     EXAMPLES: sage:
                                                                                        x^3
                                   symmetrica.compute_homsym_with_alphabet(3,1,'x')
                                                                                                 sage:
          symmetrica.compute homsym with alphabet([2,1],1,x')
                                                                                              symmet-
          rica.compute_homsym_with_alphabet([2,1],2,'x') x0^3 + 2*x0^2*x1 + 2*x0*x1^2 + x1^3 sage:
          symmetrica.compute_homsym_with_alphabet([2,1],2,a,b) a^3 + 2*a^2*b + 2*a*b^2 + b^3 sage:
          symmetrica.compute_homsym_with_alphabet([2,1],2,'x').parent() Multivariate Polynomial Ring in x0,
          x1 over Integer Ring
sage.libs.symmetrica.symmetrica.compute_monomial_with_alphabet_symmetrica (n,
                                                                                                  length,
                                                                                                  al-
                                                                                                  pha-
     computes the expansion of a monomial symmetric function labeled by a PARTITION number as a POLYNOM
     erg. The INTEGER laenge specifies the length of the alphabet.
     EXAMPLES: sage:
                                symmetrica.compute_monomial_with_alphabet([2,1],2,'x')
                                                                                         x0^2*x1
          x0*x1^2
                     sage:
                                  symmetrica.compute_monomial_with_alphabet([1,1,1],2,'x')
                                                                                                 sage:
                                                                             x1^2
                                                               x0^2
          symmetrica.compute_monomial_with_alphabet(2,2,'x')
                                                                                     sage:
                                                                                                  sym-
          metrica.compute_monomial_with_alphabet(2,2,'a,b')
                                                             a^2
                                                                          b^2
                                                                    +
                                                                                  sage:
                                                                                              symmet-
          rica.compute_monomial_with_alphabet(2,2,'x').parent() Multivariate Polynomial Ring in x0, x1
          over Integer Ring
sage.libs.symmetrica.symmetrica.compute_powsym_with_alphabet_symmetrica (n,
                                                                                                length,
                                                                                                al-
                                                                                               pha-
                                                                                                bet='x'
     computes the expansion of a power symmetric function labeled by a INTEGER label or by a PARTITION label
     or a POW SYM label as a POLYNOM erg. The INTEGER laenge specifies the length of the alphabet.
                          symmetrica.compute_powsym_with_alphabet(2,2,'x') x0^2 + x1^2 sage:
                                                                                                  sym-
          metrica.compute_powsym_with_alphabet(2,2,'x').parent() Multivariate Polynomial Ring
                                                                                               in x0,
          x1 over Integer Ring sage:
                                           symmetrica.compute_powsym_with_alphabet([2],2,'x')
                                                                                              x0^2 +
                       symmetrica.compute_powsym_with_alphabet([2],2,'a,b') a^2 + b^2 sage:
          x1<sup>2</sup> sage:
                                                                                              symmet-
```

```
rica.compute powsym with alphabet([2,1],2,'a,b') a^3 + a^2 + b^3 + a^5 + a^5
sage.libs.symmetrica.symmetrica.compute_schur_with_alphabet_det_symmetrica ( part,
                                                                                                                                                                         length,
                                                                                                                                                                         al-
                                                                                                                                                                         pha-
                                                                                                                                                                         bet='x'
         EXAMPLES: sage: symmetrica.compute_schur_with_alphabet_det(2,2) \times 0^2 + \times 0^2 + \times 1^2 sage:
                 symmetrica.compute schur with alphabet det([2],2) x0^2 + x0^*x1 + x1^2 sage:
                                                                                                                                                                symmet-
                 rica.compute_schur_with_alphabet_det(Partition([2]),2) \times 0^2 + \times 0^* \times 1 + \times 1^2 = 0 sage:
                                                                                                                                                                 symmet-
                 rica.compute_schur_with_alphabet_det(Partition([2]),2,'y') y0^2 + y0*y1 + y1^2 sage:
                                                                                                                                                                symmet-
                 rica.compute_schur_with_alphabet_det(Partition([2]),2,'a,b') a^2 + a^b + b^2
sage.libs.symmetrica.symmetrica.compute_schur_with_alphabet_symmetrica (part,
                                                                                                                                                                length,
                                                                                                                                                                al-
                                                                                                                                                                pha-
                                                                                                                                                                bet='x'
         Computes the expansion of a schurfunction labeled by a partition PART as a POLYNOM erg. The INTEGER
         length specifies the length of the alphabet.
         EXAMPLES: sage:
                                              symmetrica.compute schur with alphabet(2,2) x0^2 + x0^*x^1 + x^2 sage:
                 symmetrica.compute_schur_with_alphabet([2],2) \times 0^2 + \times 0^* \times 1
                                                                                                                            + x1^2
                                                                                                                                                                symmet-
                 rica.compute_schur_with_alphabet(Partition([2]),2) x0^2 + x0^*x1 + x1^2 sage:
                                                                                                                                                                 symmet-
                 rica.compute schur with alphabet(Partition([2]),2,'y') y0^2 + y0^*y1 + y1^2 sage:
                                                                                                                                                                symmet-
                 rica.compute_schur_with_alphabet(Partition([2]),2,'a,b') a^2 + a*b + b^2 sage:
                                                                                                                                                                 symmet-
                 rica.compute_schur_with_alphabet([2,1],1,'x') 0
sage.libs.symmetrica.symmetrica.dimension_schur_symmetrica (s)
         you enter a SCHUR object a, and the result is the dimension of the corresponding representation of the symmet-
         ric group sn.
sage.libs.symmetrica.symmetrica.dimension_symmetrization_symmetrica (n,
         computes the dimension of the degree of a irreducible representation of the GL_n, n is a INTÉGER object,
         labeled by the PARTITION object a.
sage.libs.symmetrica.symmetrica.divdiff perm schubert symmetrica (perm, a)
         Returns the result of applying the divided difference operator \delta_i to a where a is either a permutation or a Schubert
         polynomial over QQ.
                                              symmetrica.divdiff perm schubert([2,3,1], [3,2,1]) X[2, 1] sage:
         EXAMPLES: sage:
                 rica.divdiff_perm_schubert([3,1,2], [3,2,1]) X[1, 3, 2] sage: symmetrica.divdiff_perm_schubert([3,2,4,1],
                 [3,2,1]) Traceback (most recent call last): ... ValueError: cannot apply delta {[3, 2, 4, 1]} to a (= [3, 2, 1])
sage.libs.symmetrica.symmetrica.divdiff_schubert_symmetrica(i, a)
         Returns the result of applying the divided difference operator \delta_i to a where a is either a permutation or a Schubert
         polynomial over QQ.
         EXAMPLES: sage: symmetrica.divdiff_schubert(1, [3,2,1]) X[2, 3, 1] sage: symmetrica.divdiff_schubert(2,
                 [3,2,1]) X[3, 1, 2] sage: symmetrica.divdiff_schubert(3, [3,2,1]) Traceback (most recent call last): ...
                 ValueError: cannot apply delta_{3} to a (= [3, 2, 1])
sage.libs.symmetrica.symmetrica.end ()
sage.libs.symmetrica.symmetrica.qupta_nm_symmetrica(n,m)
         this routine computes the number of partitions of n with maximal part m. The result is erg. The input n,m must
```

be INTEGER objects. The result is freed first to an empty object. The result must be a different from m and n.

- sage.libs.symmetrica.symmetrica.gupta_tafel_symmetrica (max) it computes the table of the above values. The entry n,m is the result of gupta_nm. mat is freed first. max must be an INTEGER object, it is the maximum weight for the partitions. max must be different from result.
- sage.libs.symmetrica.symmetrica.hall_littlewood_symmetrica (part) computes the so called Hall Littlewood Polynomials, i.e. a SCHUR object, whose coefficient are polynomials in one variable. The method, which is used for the computation is described in the paper: A.O. Morris The Characters of the group GL(n,q) Math Zeitschr 81, 112-123 (1963)
- sage.libs.symmetrica.symmetrica.kostka_number_symmetrica (shape, content) computes the kostkanumber, i.e. the number of tableaux of given shape, which is a PARTITION object, and of given content, which also is a PARTITION object, or a VECTOR object with INTEGER entries. The result is an INTEGER object, which is freed to an empty object at the beginning. The shape could also be a SKEWPARTITION object, then we compute the number of skewtableaux of the given shape.

EXAMPLES:

```
sage: symmetrica.kostka_number([2,1],[1,1,1])
2
sage: symmetrica.kostka_number([1,1,1],[1,1,1])
1
sage: symmetrica.kostka_number([3],[1,1,1])
1
```

sage.libs.symmetrica.symmetrica.kostka_tab_symmetrica (shape, content) computes the list of tableaux of given shape and content. shape is a PARTITION object or a SKEWPARTITION object and content is a PARTITION object or a VECTOR object with INTEGER entries, the result becomes a LIST object whose entries are the computed TABLEAUX object.

EXAMPLES:

```
sage: symmetrica.kostka_tab([3],[1,1,1])
[[[1, 2, 3]]]
sage: symmetrica.kostka_tab([2,1],[1,1,1])
[[[1, 2], [3]], [[1, 3], [2]]]
sage: symmetrica.kostka_tab([1,1,1],[1,1,1])
[[[1], [2], [3]]]
sage: symmetrica.kostka_tab([[2,2,1],[1,1]],[1,1,1])
[[[None, 1], [None, 2], [3]],
    [[None, 1], [None, 3], [2]],
    [[None, 2], [None, 3], [1]]]
sage: symmetrica.kostka_tab([[2,2],[1]],[1,1,1])
[[[None, 1], [2, 3]], [[None, 2], [1, 3]]]
```

sage.libs.symmetrica.symmetrica. $kostka_tafel_symmetrica$ (n) Returns the table of Kostka numbers of weight n.

```
sage: symmetrica.kostka_tafel(1)
[1]

sage: symmetrica.kostka_tafel(2)
[1 0]
[1 1]

sage: symmetrica.kostka_tafel(3)
[1 0 0]
[1 1 0]
```

```
sage: symmetrica.kostka_tafel(4)
[1 0 0 0 0]
[1 1 0 0 0]
[1 1 1 0 0]
[1 2 1 1 0]
[1 3 2 3 1]

sage: symmetrica.kostka_tafel(5)
[1 0 0 0 0 0 0 0]
[1 1 0 0 0 0 0 0]
[1 1 1 0 0 0 0 0]
[1 2 1 1 0 0 0]
[1 2 1 1 0 0 0]
[1 2 1 1 0 0 0]
[1 3 3 3 2 1 0]
[1 4 5 6 5 4 1]
```

sage.libs.symmetrica.symmetrica.kranztafel_symmetrica (a, b) you enter the INTEGER objects, say a and b, and res becomes a MATRIX object, the charactertable of S_b wr S_a, co becomes a VECTOR object of classorders and cl becomes a VECTOR object of the classlabels.

EXAMPLES:

```
sage: (a,b,c) = symmetrica.kranztafel(2,2)
sage: a
[ 1 -1 1 -1 1]
[ 1 1 1 1 1]
[-1 \ 1 \ 1 \ -1 \ 1]
[ 0 0 2 0 -2]
[-1 \ -1 \ 1 \ 1 \ 1]
sage: b
[2, 2, 1, 2, 1]
sage: for m in c: print(m)
[0 0]
[0 1]
[0 0]
[1 0]
[0 2]
[0 0]
[1 1]
[0 0]
[2 0]
[0 0]
```

```
sage.libs.symmetrica.symmetrica.mult_monomial_monomial_symmetrica ( m1, m2)
sage.libs.symmetrica.symmetrica.mult_schubert_schubert_symmetrica ( a, b)
Multiplies the Schubert polynomials a and b.

EXAMPLES: sage: symmetrica.mult_schubert([3,2,1], [3,2,1]) X[5, 3, 1, 2, 4]
sage.libs.symmetrica.symmetrica.mult_schubert_variable_symmetrica ( a, i)
Returns the product of a and x i. Note that indexing with i starts at 1.
```

EXAMPLES: sage: symmetrica.mult_schubert_variable([3,2,1], 2) X[3, 2, 4, 1] sage: symmetrica.mult_schubert_variable([3,2,1], 4) X[3, 2, 1, 4, 6, 5] - X[3, 2, 1, 5, 4]

sage.libs.symmetrica.symmetrica. $mult_schur_schur_symmetrica$ (s1, s2)

```
sage.libs.symmetrica.symmetrica.ndg_symmetrica (part, perm)
sage.libs.symmetrica.symmetrica.newtrans_symmetrica (perm)
        computes the decomposition of a schubertpolynomial labeled by the permutation perm, as a sum of Schurfunc-
        tion. FIXME!
sage.libs.symmetrica.symmetrica.odd_to_strict_part_symmetrica (part)
        implements the bijection between partitions with odd parts and strict partitions. input is a VECTOR type parti-
        tion, the result is a partition of the same weight with different parts.
sage.libs.symmetrica.symmetrica.odg_symmetrica (part, perm)
        Calculates the irreduzible matrix representation D^part(perm), which consists of real numbers.
        REFERENCE: G. James/ A. Kerber: Representation Theory of the Symmetric Group. Addison/Wesley
                1981. pp. 127-129.
sage.libs.symmetrica.symmetrica.outerproduct_schur_symmetrica (parta, partb)
        you enter two PARTITION objects, and the result is a SCHUR object, which is the expansion of the product
        of the two schurfunctions, labbeled by the two PARTITION objects parta and partb. Of course this can also be
        interpreted as the decomposition of the outer tensor product of two irreducibe representations of the symmetric
        group.
        EXAMPLES: sage: symmetrica.outerproduct schur([2],[2]) s[2,2] + s[3,1] + s[4]
sage.libs.symmetrica.symmetrica.part_part_skewschur_symmetrica (outer, inner)
        Returns the skew schur function s_{outer/inner}
        EXAMPLES: sage: symmetrica.part part skewschur([3,2,1],[2,1]) s[1,1,1] + 2*s[2,1] + s[3]
sage.libs.symmetrica.symmetrica.plethysm symmetrica (outer, inner)
sage.libs.symmetrica.symmetrica.q_core_symmetrica ( part, d)
        computes the q-core of a PARTITION object part. This is the remaining partition (=res) after removing of all
        hooks of length d (= INTEGER object). The result may be an empty object, if the whole partition disappears.
sage.libs.symmetrica.symmetrica.random_partition_symmetrica (n)
        returns a random partition p of the entered weight w. w must be an INTEGER object, p becomes a PARTITION
        object. Type of partition is VECTOR. Its the algorithm of Nijnhuis Wilf p.76
sage.libs.symmetrica.symmetrica.scalarproduct_schubert_symmetrica (a,b)
        EXAMPLES: sage: symmetrica.scalarproduct schubert([3,2,1], [3,2,1]) X[1, 3, 5, 2, 4] sage: symmetrica.scalarproduct schubert([3,2,1], [3,2,1]) X[1, 3, 5, 2, 4] sage: symmetrica.scalarproduct schubert([3,2,1], [3,2,1]) X[1, 3, 5, 2, 4] sage: symmetrica.scalarproduct schubert([3,2,1], [3,2,1]) X[1, 3, 5, 2, 4] sage: symmetrica.scalarproduct schubert([3,2,1], [3,2,1]) X[1, 3, 5, 2, 4] sage: symmetrica.scalarproduct schubert([3,2,1], [3,2,1]) X[1, 3, 5, 2, 4] sage: symmetrica.scalarproduct schubert([3,2,1], [3,2,1]) X[1, 3, 5, 2, 4] sage: symmetrica.scalarproduct schubert([3,2,1], [3,2,1]) X[1, 3, 5, 2, 4] sage: symmetrica.scalarproduct schubert([3,2,1], [3,2,1]) X[1, 3, 5, 2, 4] sage: symmetrica.scalarproduct schubert([3,2,1], [3,2,1]) X[1, 3, 5, 2, 4] sage: symmetrica.scalarproduct schubert([3,2,1], [3,2,1]) X[1, 3, 5, 2, 4] sage: symmetrica.scalarproduct schubert([3,2,1], [3,2,1]) X[1, 3, 5, 2, 4] sage: symmetrica.scalarproduct schubert([3,2,1], [3,2,1]) X[1, 3, 5, 2, 4] sage: symmetrica.scalarproduct schubert([3,2,1], [3,2,1]) X[1, 3, 5, 2, 4] sage: symmetrica.scalarproduct schubert([3,2,1], [3,2,1]) X[1, 3, 5, 2, 4] sage: symmetrica.scalarproduct schubert([3,2,1], [3,2,1]) X[1, 3, 5, 2, 4] sage: symmetrica.scalarproduct schubert([3,2,1], [3,2,1]) X[1, 3, 5, 2, 4] sage: symmetrica.scalarproduct schubert([3,2,1], [3,2,1]) X[1, 3, 5, 2, 4] sage: symmetrica.scalarproduct schubert([3,2,1], [3,2,1]) X[1, 3, 5, 2, 4] sage: symmetrica.scalarproduct schubert([3,2,1], [3,2,1]) X[1, 3, 5, 2, 4] sage: symmetrica.scalarproduct schubert([3,2,1], [3,2,1]) X[1, 3,2,1] sage: symmetrica.scalarproduct schubert([3,2,1], [3,2,1]) sage: symmetrica.scalarproduct schubert([3,2,1], [3,2,1]) sage: symmetrica.scalarproduct schubert([3,2,1], [3,2,1]) sage: symm
                rica.scalarproduct_schubert([3,2,1], [2,1,3]) X[1, 2, 4, 3]
sage.libs.symmetrica.symmetrica.scalarproduct_schur_symmetrica (s1, s2)
sage.libs.symmetrica.symmetrica.schur_schur_plet_symmetrica (outer, inner)
sage.libs.symmetrica.symmetrica.sdg_symmetrica (part, perm)
        Calculates the irreduzible matrix representation D^part(perm), which consists of rational numbers.
        REFERENCE: G. James/ A. Kerber: Representation Theory of the Symmetric Group. Addison/Wesley
                1981. pp. 124-126.
sage.libs.symmetrica.symmetrica.specht_dg_symmetrica (part, perm)
sage.libs.symmetrica.symmetrica.start()
sage.libs.symmetrica.symmetrica.strict_to_odd_part_symmetrica (part)
        implements the bijection between strict partitions and partitions with odd parts. input is a VECTOR type parti-
        tion, the result is a partition of the same weight with only odd parts.
sage.libs.symmetrica.symmetrica.t ELMSYM HOMSYM symmetrica (elmsym)
sage.libs.symmetrica.symmetrica.t ELMSYM MONOMIAL symmetrica (elmsym)
```

```
sage.libs.symmetrica.symmetrica.t_ELMSYM_POWSYM_symmetrica (elmsym)
sage.libs.symmetrica.symmetrica.t_ELMSYM_SCHUR_symmetrica (elmsym)
sage.libs.symmetrica.symmetrica.t_HOMSYM_ELMSYM_symmetrica ( homsym)
sage.libs.symmetrica.symmetrica.t_HOMSYM_MONOMIAL_symmetrica ( homsym)
sage.libs.symmetrica.symmetrica.t HOMSYM POWSYM symmetrica (homsym)
sage.libs.symmetrica.symmetrica.t HOMSYM SCHUR symmetrica (homsym)
sage.libs.symmetrica.symmetrica.t_MONOMIAL_ELMSYM_symmetrica ( monomial)
sage.libs.symmetrica.symmetrica.t_MONOMIAL_HOMSYM_symmetrica ( monomial)
sage.libs.symmetrica.symmetrica.t_MONOMIAL_POWSYM_symmetrica ( monomial)
sage.libs.symmetrica.symmetrica.t_MONOMIAL_SCHUR_symmetrica (monomial)
sage.libs.symmetrica.symmetrica.t_POLYNOM_ELMSYM_symmetrica(p)
    Converts a symmetric polynomial with base ring QQ or ZZ into a symmetric function in the elementary basis.
sage.libs.symmetrica.symmetrica.t POLYNOM MONOMIAL symmetrica (p)
    Converts a symmetric polynomial with base ring QQ or ZZ into a symmetric function in the monomial basis.
sage.libs.symmetrica.symmetrica.t_POLYNOM_POWER_symmetrica (p)
    Converts a symmetric polynomial with base ring QQ or ZZ into a symmetric function in the power sum basis.
sage.libs.symmetrica.symmetrica.t POLYNOM SCHUBERT symmetrica (a)
    Converts a multivariate polynomial a to a Schubert polynomial.
    EXAMPLES: sage:
                         R.\langle x1,x2,x3\rangle = QQ[]
                                              sage:
                                                       w0 =
                                                                x1^2*x2
                                                                         sage:
                                                                                 symmet-
        rica.t_POLYNOM_SCHUBERT(w0) X[3, 2, 1]
sage.libs.symmetrica.symmetrica.t_POLYNOM_SCHUR_symmetrica (p)
    Converts a symmetric polynomial with base ring QQ or ZZ into a symmetric function in the Schur basis.
sage.libs.symmetrica.symmetrica.t_POWSYM_ELMSYM_symmetrica (powsym)
sage.libs.symmetrica.symmetrica.t_POWSYM_HOMSYM_symmetrica (powsym)
sage.libs.symmetrica.symmetrica.t_POWSYM_MONOMIAL_symmetrica (powsym)
sage.libs.symmetrica.symmetrica.t POWSYM SCHUR symmetrica (powsym)
sage.libs.symmetrica.symmetrica.t SCHUBERT POLYNOM symmetrica (a)
    Converts a Schubert polynomial to a 'regular' multivariate polynomial.
    EXAMPLES: sage: symmetrica.t_SCHUBERT_POLYNOM([3,2,1]) x0^2*x1
sage.libs.symmetrica.symmetrica.t SCHUR ELMSYM symmetrica (schur)
sage.libs.symmetrica.symmetrica.t SCHUR HOMSYM symmetrica (schur)
\verb|sage.libs.symmetrica.t_SCHUR_MONOMIAL_symmetrica| (|\mathit{schur}|)
sage.libs.symmetrica.symmetrica.t_SCHUR_POWSYM_symmetrica (schur)
sage.libs.symmetrica.symmetrica.test_integer (x)
    Tests functionality for converting between Sage's integers and symmetrica's integers.
    EXAMPLES:
    sage: from sage.libs.symmetrica.symmetrica import test_integer
    sage: test_integer(1)
    sage: test_integer(-1)
```

```
-1
sage: test_integer(2^33)
8589934592
sage: test_integer(-2^33)
-8589934592
sage: test_integer(2^100)
1267650600228229401496703205376
sage: test_integer(-2^100)
-1267650600228229401496703205376
sage: for i in range(100):
...: if test_integer(2^i) != 2^i:
...: print("Failure at {}".format(i))
```

TWENTYFIVE

UTILITIES FOR SAGE-MPMATH INTERACTION

Also patches some mpmath functions for speed

```
sage.libs.mpmath.utils.bitcount (n)
Bitcount of a Sage Integer or Python int/long.
```

EXAMPLES:

```
sage: from mpmath.libmp import bitcount
sage: bitcount(0)
0
sage: bitcount(1)
1
sage: bitcount(100)
7
sage: bitcount(-100)
7
sage: bitcount(2r)
2
sage: bitcount(2L)
```

```
sage.libs.mpmath.utils.call (func, *args, **kwargs)
```

Call an impmath function with Sage objects as inputs and convert the result back to a Sage real or complex number.

By default, a RealNumber or ComplexNumber with the current working precision of mpmath (mpmath.mp.prec) will be returned.

If prec=n is passed among the keyword arguments, the temporary working precision will be set to n and the result will also have this precision.

If parent=P is passed, P.prec() will be used as working precision and the result will be coerced to P (or the corresponding complex field if necessary).

Arguments should be Sage objects that can be coerced into RealField or ComplexField elements. Arguments may also be tuples, lists or dicts (which are converted recursively), or any type that mpmath understands natively (e.g. Python floats, strings for options).

```
sage: import sage.libs.mpmath.all as a
sage: a.mp.prec = 53
sage: a.call(a.erf, 3+4*I)
-120.186991395079 - 27.7503372936239*I
sage: a.call(a.polylog, 2, 1/3+4/5*I)
0.153548951541433 + 0.875114412499637*I
```

```
sage: a.call(a.barnesg, 3+4*I)
-0.000676375932234244 - 0.0000442236140124728*I
sage: a.call(a.barnesg, -4)
0.000000000000000
sage: a.call(a.hyper, [2,3], [4,5], 1/3)
1.10703578162508
sage: a.call(a.hyper, [2,3], [4,(2,3)], 1/3)
1.95762943509305
sage: a.call(a.guad, a.erf, [0,1])
0.486064958112256
sage: a.call(a.gammainc, 3+4*I, 2/3, 1-pi*I, prec=100)
-274.18871130777160922270612331 + 101.59521032382593402947725236*I
sage: x = (3+4*I).n(100)
sage: y = (2/3).n(100)
sage: z = (1-pi*I).n(100)
sage: a.call(a.gammainc, x, y, z, prec=100)
-274.18871130777160922270612331 + 101.59521032382593402947725236*I
sage: a.call(a.erf, infinity)
1.000000000000000
sage: a.call(a.erf, -infinity)
-1.000000000000000
sage: a.call(a.gamma, infinity)
+infinity
sage: a.call(a.polylog, 2, 1/2, parent=RR)
0.582240526465012
sage: a.call(a.polylog, 2, 2, parent=RR)
2.46740110027234 - 2.17758609030360*I
sage: a.call(a.polylog, 2, 1/2, parent=RealField(100))
0.58224052646501250590265632016
sage: a.call(a.polylog, 2, 2, parent=RealField(100))
2.4674011002723396547086227500 - 2.1775860903036021305006888982*I
sage: a.call(a.polylog, 2, 1/2, parent=CC)
0.582240526465012
sage: type(_)
<type 'sage.rings.complex_number.ComplexNumber'>
sage: a.call(a.polylog, 2, 1/2, parent=RDF)
0.5822405264650125
sage: type(_)
<type 'sage.rings.real_double.RealDoubleElement'>
```

Check that trac ticket #11885 is fixed:

```
sage: a.call(a.ei, 1.0r, parent=float)
1.8951178163559366
```

Check that trac ticket #14984 is fixed:

```
sage: a.call(a.log, -1.0r, parent=float)
3.141592653589793j
```

sage.libs.mpmath.utils.from_man_exp (man, exp, prec=0, rnd='d')

Create normalized mpf value tuple from mantissa and exponent.

With prec > 0, rounds the result in the desired direction if necessary.

```
sage: from mpmath.libmp import from_man_exp
sage: from_man_exp(-6, -1)
```

```
(1, 3, 0, 2)

sage: from_man_exp(-6, -1, 1, 'd')

(1, 1, 1, 1)

sage: from_man_exp(-6, -1, 1, 'u')

(1, 1, 2, 1)
```

sage.libs.mpmath.utils.isqrt (n)

Square root (rounded to floor) of a Sage Integer or Python int/long. The result is a Sage Integer.

EXAMPLES:

```
sage: from mpmath.libmp import isqrt
sage: isqrt(0)
0
sage: isqrt(100)
10
sage: isqrt(10)
3
sage: isqrt(10r)
3
sage: isqrt(10L)
```

sage.libs.mpmath.utils.mpmath_to_sage (x, prec)

Convert any mpmath number (mpf or mpc) to a Sage RealNumber or ComplexNumber of the given precision.

EXAMPLES:

A real example:

```
sage: RealField(100) (pi)
3.1415926535897932384626433833
sage: t = RealField(100) (pi) ._mpmath_(); t
mpf('3.1415926535897932')
sage: a.mpmath_to_sage(t, 100)
3.1415926535897932384626433833
```

We can ask for more precision, but the result is undefined:

```
sage: a.mpmath_to_sage(t, 140) # random
3.1415926535897932384626433832793333156440
sage: ComplexField(140)(pi)
3.1415926535897932384626433832795028841972
```

A complex example:

```
sage: ComplexField(100)([0, pi])
3.1415926535897932384626433833*I
sage: t = ComplexField(100)([0, pi])._mpmath_(); t
mpc(real='0.0', imag='3.1415926535897932')
sage: sage.libs.mpmath.all.mpmath_to_sage(t, 100)
3.1415926535897932384626433833*I
```

Again, we can ask for more precision, but the result is undefined:

```
sage: sage.libs.mpmath.all.mpmath_to_sage(t, 140) # random
3.1415926535897932384626433832793333156440*I
sage: ComplexField(140)([0, pi])
3.1415926535897932384626433832795028841972*I
```

sage.libs.mpmath.utils. normalize (sign, man, exp, bc, prec, rnd)

Create normalized mpf value tuple from full list of components.

EXAMPLES:

```
sage: from mpmath.libmp import normalize
sage: normalize(0, 4, 5, 3, 53, 'n')
(0, 1, 7, 1)
```

```
sage.libs.mpmath.utils. sage to mpmath (x, prec)
```

Convert any Sage number that can be coerced into a RealNumber or ComplexNumber of the given precision into an mpmath mpf or mpc. Integers are currently converted to int.

Lists, tuples and dicts passed as input are converted recursively.

```
sage: import sage.libs.mpmath.all as a
sage: a.mp.dps = 15
sage: print(a.sage_to_mpmath(2/3, 53))
0.666666666666667
sage: print(a.sage_to_mpmath(2./3, 53))
0.666666666666667
sage: print(a.sage_to_mpmath(3+4*I, 53))
(3.0 + 4.0i)
sage: print(a.sage_to_mpmath(1+pi, 53))
4.14159265358979
sage: a.sage_to_mpmath(infinity, 53)
mpf('+inf')
sage: a.sage_to_mpmath(-infinity, 53)
mpf('-inf')
sage: a.sage_to_mpmath(NaN, 53)
mpf('nan')
sage: a.sage_to_mpmath(0, 53)
sage: a.sage_to_mpmath([0.5, 1.5], 53)
[mpf('0.5'), mpf('1.5')]
sage: a.sage_to_mpmath((0.5, 1.5), 53)
(mpf('0.5'), mpf('1.5'))
sage: a.sage_to_mpmath({'n':0.5}, 53)
{'n': mpf('0.5')}
```

TWENTYSIX

VICTOR SHOUP'S NTL C++ LIBRARY

Sage provides an interface to Victor Shoup's C++ library NTL. Features of this library include *incredibly fast* arithmetic with polynomials and asymptotically fast factorization of polynomials.

Sage Reference Manual: C/C++ Library Interfaces, Release 7.6		

THE ELLIPTIC CURVE METHOD FOR INTEGER FACTORIZATION (ECM)

Sage includes GMP-ECM, which is a highly optimized implementation of Lenstra's elliptic curve factorization method. See http://ecm.gforge.inria.fr/ for more about GMP-ECM. This file provides a Cython interface to the GMP-ECM library.

AUTHORS:

- Robert L Miller (2008-01-21): library interface (clone of ecmfactor.c)
- Jeroen Demeyer (2012-03-29): signal handling, documentation
- Paul Zimmermann (2011-05-22) added input/output of sigma

EXAMPLES:

```
sage: from sage.libs.libecm import ecmfactor
sage: result = ecmfactor(999, 0.00)
sage: result[0] and (result[1] in [27, 37, 999])
True
sage: result = ecmfactor(999, 0.00, verbose=True)
Performing one curve with B1=0
Found factor in step 1: ...
sage: result[0] and (result[1] in [27, 37, 999])
True
sage: ecmfactor(2^128+1,1000,sigma=227140902)
(True, 5704689200685129054721, 227140902)
```

sage.libs.libecm.ecmfactor (number, B1, verbose=False, sigma=0)

Try to find a factor of a positive integer using ECM (Elliptic Curve Method). This function tries one elliptic curve.

INPUT:

- •number positive integer to be factored
- •B1 bound for step 1 of ECM
- •verbose (default: False) print some debugging information

OUTPUT:

Either (False, None) if no factor was found, or (True, f) if the factor f was found.

EXAMPLES:

```
sage: from sage.libs.libecm import ecmfactor
```

This number has a small factor which is easy to find for ECM:

```
sage: N = 2^167 - 1
sage: factor(N)
2349023 * 79638304766856507377778616296087448490695649
sage: ecmfactor(N, 2e5)
(True, 2349023, ...)
```

If a factor was found, we can reproduce the factorization with the same sigma value:

```
sage: N = 2^167 - 1
sage: ecmfactor(N, 2e5, sigma=1473308225)
(True, 2349023, 1473308225)
```

With a smaller B1 bound, we may or may not succeed:

```
sage: ecmfactor(N, 1e2) # random
(False, None)
```

The following number is a Mersenne prime, so we don't expect to find any factors (there is an extremely small chance that we get the input number back as factorization):

```
sage: N = 2^127 - 1
sage: N.is_prime()
True
sage: ecmfactor(N, 1e3)
(False, None)
```

If we have several small prime factors, it is possible to find a product of primes as factor:

```
sage: N = 2^179 - 1
sage: factor(N)
359 * 1433 * 1489459109360039866456940197095433721664951999121
sage: ecmfactor(N, 1e3) # random
(True, 514447, 3475102204)
```

We can ask for verbose output:

AN INTERFACE TO ANDERS BUCH'S LITTLEWOOD-RICHARDSON CALCULATOR LRCALC

The "Littlewood-Richardson Calculator" is a C library for fast computation of Littlewood-Richardson (LR) coefficients and products of Schubert polynomials. It handles single LR coefficients, products of and coproducts of Schur functions, skew Schur functions, and fusion products. All of the above are achieved by counting LR (skew)-tableaux (also called Yamanouchi (skew)-tableaux) of appropriate shape and content by iterating through them. Additionally, lrcalc handles products of Schubert polynomials.

The web page of lrcalc is http://math.rutgers.edu/~asbuch/lrcalc/.

The following describes the Sage interface to this library.

EXAMPLES:

```
sage: import sage.libs.lrcalc.lrcalc as lrcalc
```

Compute a single Littlewood-Richardson coefficient:

```
sage: lrcalc.lrcoef([3,2,1],[2,1],[2,1])
2
```

Compute a product of Schur functions; return the coefficients in the Schur expansion:

```
sage: lrcalc.mult([2,1], [2,1])
{[2, 2, 1, 1]: 1,
    [2, 2, 2]: 1,
    [3, 1, 1, 1]: 1,
    [3, 2, 1]: 2,
    [3, 3]: 1,
    [4, 1, 1]: 1,
    [4, 2]: 1}
```

Same product, but include only partitions with at most 3 rows. This corresponds to computing in the representation ring of gl(3):

```
sage: lrcalc.mult([2,1], [2,1], 3)
{[2, 2, 2]: 1, [3, 2, 1]: 2, [3, 3]: 1, [4, 1, 1]: 1, [4, 2]: 1}
```

We can also compute the fusion product, here for sl(3) and level 2:

```
sage: lrcalc.mult([3,2,1], [3,2,1], 3,2)
{[4, 4, 4]: 1, [5, 4, 3]: 1}
```

Compute the expansion of a skew Schur function:

```
sage: lrcalc.skew([3,2,1],[2,1])
{[1, 1, 1]: 1, [2, 1]: 2, [3]: 1}
```

Compute the coproduct of a Schur function:

```
sage: lrcalc.coprod([3,2,1])
{([1, 1, 1], [2, 1]): 1,
  ([2, 1], [2]): 2,
  ([2, 1], [3]): 1,
  ([2, 1, 1], [1, 1]): 1,
  ([2, 1, 1], [2]): 1,
  ([2, 2], [1, 1]): 1,
  ([2, 2], [2]): 1,
  ([2, 2], [2]): 1,
  ([3, 1], [1, 1]): 1,
  ([3, 1], [1]): 1,
  ([3, 1], [2]): 1,
  ([3, 2], [1]): 1,
  ([3, 2], [1]): 1,
```

Multiply two Schubert polynomials:

```
sage: lrcalc.mult_schubert([4,2,1,3], [1,4,2,5,3])
{[4, 5, 1, 3, 2]: 1,
    [5, 3, 1, 4, 2]: 1,
    [5, 4, 1, 2, 3]: 1,
    [6, 2, 1, 4, 3, 5]: 1}
```

Same product, but include only permutations of 5 elements in the result. This corresponds to computing in the cohomology ring of Fl(5):

```
sage: lrcalc.mult_schubert([4,2,1,3], [1,4,2,5,3], 5)
{[4, 5, 1, 3, 2]: 1, [5, 3, 1, 4, 2]: 1, [5, 4, 1, 2, 3]: 1}
```

List all Littlewood-Richardson tableaux of skew shape μ/ν ; in this example $\mu=[3,2,1]$ and $\nu=[2,1]$. Specifying a third entry maxrows restricts the alphabet to $\{1,2,\ldots,maxrows\}$:

```
sage: list(lrcalc.lrskew([3,2,1],[2,1]))
[[None, None, 1], [None, 1], [1]], [[None, None, 1], [None, 1], [2]],
[[None, None, 1], [None, 2], [1]], [[None, None, 1], [None, 2], [3]]]

sage: list(lrcalc.lrskew([3,2,1],[2,1],maxrows=2))
[[[None, None, 1], [None, 1], [1]], [[None, None, 1], [None, 1], [2]], [[None, None, 1], [None, 2], [1]]]
```

Todo

use this library in the SymmetricFunctions code, to make it easy to apply it to linear combinations of Schur functions.

See also:

- lrcoef()
- mult()
- coprod()

- skew()
- lrskew()
- mult_schubert()

Underlying algorithmic in Ircalc

Here is some additional information regarding the main low-level C-functions in lrcalc. Given two partitions outer and inner with inner contained in outer, the function:

```
skewtab *st_new(vector *outer, vector *inner, vector *conts, int maxrows)
```

constructs and returns the (lexicographically) first LR skew tableau of shape outer / inner. Further restrictions can be imposed using conts and maxrows.

Namely, the integer maxrows is a bound on the integers that can be put in the tableau. The name is chosen because this will limit the partitions in the output of skew() or mult() to partitions with at most this number of rows.

The vector conts is the content of an empty tableau(!!). More precisely, this vector is added to the usual content of a tableau whenever the content is needed. This affects which tableaux are considered LR tableaux (see mult () below). conts may also be the NULL pointer, in which case nothing is added.

The other function:

```
int *st_next(skewtab *st)
```

computes in place the (lexicographically) next skew tableau with the same constraints, or returns 0 if st is the last one.

For a first example, see the skew() function code in the lrcalc source code. We want to compute a skew schur function, so create a skew LR tableau of the appropriate shape with $st_new($ (with conts = NULL), then iterate through all the LR tableaux with $st_next()$. For each skew tableau, we use that st->conts is the content of the skew tableau, find this shape in the res hash table and add one to the value.

For a second example, see mult(vector *sh1, vector *sh2, maxrows). Here we call $st_new()$ with the shape sh1 / (0) and use sh2 as the conts argument. The effect of using sh2 in this way is that st_next will iterate through semistandard tableaux T of shape sh1 such that the following tableau:

```
111111
22222 <--- minimal tableau of shape sh2
333

****
**T**

****

****
```

is a LR skew tableau, and st->conts contains the content of the combined tableaux.

More generally, st_new(outer,inner,conts,maxrows) and st_next can be used to compute the Schur expansion of the product S_{outer/inner} * S_conts, restricted to partitions with at most maxrows rows.

AUTHORS:

- Mike Hansen (2010): core of the interface
- Anne Schilling, Nicolas M. Thiéry, and Anders Buch (2011): fusion product, iterating through LR tableaux, finalization, documentation

```
sage.libs.lrcalc.lrcalc.coprod ( part, all=0)
```

Compute the coproduct of a Schur function.

Return a linear combination of pairs of partitions representing the coproduct of the Schur function given by the partition part .

INPUT:

- •part a partition.
- •all an integer.

If all is non-zero then all terms are included in the result. If all is zero, then only pairs of partitions (part1, part2) for which the weight of part1 is greater than or equal to the weight of part2 are included; the rest of the coefficients are redundant because Littlewood-Richardson coefficients are symmetric.

EXAMPLES:

```
sage: from sage.libs.lrcalc.lrcalc import coprod
sage: sorted(coprod([2,1]).items())
[(([1, 1], [1]), 1), (([2], [1]), 1), (([2, 1], []), 1)]
```

```
sage.libs.lrcalc.lrcalc.lrcoef (outer, inner1, inner2)
```

Compute a single Littlewood-Richardson coefficient.

Return the coefficient of outer in the product of the Schur functions indexed by inner1 and inner2.

INPUT:

- •outer a partition (weakly decreasing list of non-negative integers).
- •inner1 a partition.
- •inner2 a partition.

Note: This function converts its inputs into Partition () 's. If you don't need these checks and your inputs are valid, then you can use <code>lrcoef_unsafe()</code>.

EXAMPLES:

```
sage: from sage.libs.lrcalc.lrcalc import lrcoef
sage: lrcoef([3,2,1], [2,1], [2,1])
2
sage: lrcoef([3,3], [2,1], [2,1])
1
sage: lrcoef([2,1,1,1,1], [2,1], [2,1])
0
```

sage.libs.lrcalc.lrcalc.lrcoef_unsafe (outer, inner1, inner2)

Compute a single Littlewood-Richardson coefficient.

Return the coefficient of outer in the product of the Schur functions indexed by inner1 and inner2.

INPUT:

- •outer a partition (weakly decreasing list of non-negative integers).
- •inner1 a partition.
- •inner2 a partition.

Warning: This function does not do any check on its input. If you want to use a safer version, use lrcoef().

EXAMPLES:

```
sage: from sage.libs.lrcalc.lrcalc import lrcoef_unsafe
sage: lrcoef_unsafe([3,2,1], [2,1], [2,1])
2
sage: lrcoef_unsafe([3,3], [2,1], [2,1])
1
sage: lrcoef_unsafe([2,1,1,1,1], [2,1], [2,1])
0
```

sage.libs.lrcalc.lrcalc.lrskew (outer, inner, weight=None, maxrows=0)
Return the skew LR tableaux of shape outer / inner.

INPUT:

- •outer a partition.
- •inner a partition.
- •weight a partition (optional).
- •maxrows an integer (optional).

OUTPUT: a list of SkewTableau`x. This will change to an iterator over such skew tableaux once Cython will support the ``yield` statement. Specifying a third entry maxrows restricts the alphabet to $\{1,2,\ldots,maxrows\}$. Specifying weight returns only those tableaux of given content/weight.

```
sage: from sage.libs.lrcalc.lrcalc import lrskew
sage: for st in lrskew([3,2,1],[2]):
          st.pp()
. . 1
1 1
2
. . 1
1
  2.
2
     1
   2
3
sage: for st in 1rskew([3,2,1],[2], maxrows=2):
. . . . :
          st.pp()
. . 1
1 1
2
      1
2.
sage: lrskew([3,2,1],[2], weight=[3,1])
[[[None, None, 1], [1, 1], [2]]]
```

sage.libs.lrcalc.lrcalc.mult (part1, part2, maxrows=None, level=None, quantum=None)
Compute a product of two Schur functions.

Return the product of the Schur functions indexed by the partitions part1 and part2.

INPUT:

```
part1 - a partition
part2 - a partition
maxrows - (optional) an integer
level - (optional) an integer
quantum - (optional) an element of a ring
```

If maxrows is specified, then only partitions with at most this number of rows are included in the result.

If both maxrows and level are specified, then the function calculates the fusion product for $\mathfrak{sl}(\max s)$ of the given level.

If quantum is set, then this returns the product in the quantum cohomology ring of the Grassmannian. In particular, both maxrows and level need to be specified.

EXAMPLES:

```
sage: from sage.libs.lrcalc.lrcalc import mult
   sage: mult([2],[])
   {[2]: 1}
   sage: sorted(mult([2],[2]).items())
   [([2, 2], 1), ([3, 1], 1), ([4], 1)]
   sage: sorted(mult([2,1],[2,1]).items())
   [([2, 2, 1, 1], 1), ([2, 2, 2], 1), ([3, 1, 1, 1], 1), ([3, 2, 1], 2), ([3, 2, 1], 2)]
\rightarrow 3], 1), ([4, 1, 1], 1), ([4, 2], 1)]
   sage: sorted(mult([2,1],[2,1],maxrows=2).items())
   [([3, 3], 1), ([4, 2], 1)]
   sage: mult([2,1],[3,2,1],3)
   {[3, 3, 3]: 1, [4, 3, 2]: 2, [4, 4, 1]: 1, [5, 2, 2]: 1, [5, 3, 1]: 1}
   sage: mult([2,1],[2,1],3,3)
   \{[2, 2, 2]: 1, [3, 2, 1]: 2, [3, 3]: 1, [4, 1, 1]: 1\}
   sage: mult([2,1],[2,1],None,3)
   Traceback (most recent call last):
   \label{thm:problem} \textit{ValueError: maxrows needs to be specified } \textbf{if} \textit{ you specify the level}
The quantum product::
   sage: q = polygen(QQ, 'q')
   sage: sorted(mult([1],[2,1], 2, 2, quantum=q).items())
   [([], q), ([2, 2], 1)]
   sage: sorted(mult([2,1],[2,1], 2, 2, quantum=q).items())
   [([1, 1], q), ([2], q)]
   sage: mult([2,1],[2,1], quantum=q)
   Traceback (most recent call last):
   ValueError: missing parameters maxrows or level
```

sage.libs.lrcalc.lrcalc.mult_schubert (w1, w2, rank=0)

Compute a product of two Schubert polynomials.

Return a linear combination of permutations representing the product of the Schubert polynomials indexed by the permutations w1 and w2.

INPUT:

- •w1 a permutation.
- •w2 a permutation.
- •rank an integer.

If rank is non-zero, then only permutations from the symmetric group S(rank) are included in the result.

EXAMPLES:

```
sage: from sage.libs.lrcalc.lrcalc import mult_schubert
sage: result = mult_schubert([3, 1, 5, 2, 4], [3, 5, 2, 1, 4])
sage: sorted(result.items())
[([5, 4, 6, 1, 2, 3], 1), ([5, 6, 3, 1, 2, 4], 1),
  ([5, 7, 2, 1, 3, 4, 6], 1), ([6, 3, 5, 1, 2, 4], 1),
  ([6, 4, 3, 1, 2, 5], 1), ([6, 5, 2, 1, 3, 4], 1),
  ([7, 3, 4, 1, 2, 5, 6], 1), ([7, 4, 2, 1, 3, 5, 6], 1)]
```

sage.libs.lrcalc.lrcalc. skew (outer, inner, maxrows=0)

Compute the Schur expansion of a skew Schur function.

Return a linear combination of partitions representing the Schur function of the skew Young diagram outer / inner, consisting of boxes in the partition outer that are not in inner.

INPUT:

- •outer a partition.
- •inner a partition.
- •maxrows an integer or None.

If maxrows is specified, then only partitions with at most this number of rows are included in the result.

EXAMPLES:

```
sage: from sage.libs.lrcalc.lrcalc import skew
sage: sorted(skew([2,1],[1]).items())
[([1, 1], 1), ([2], 1)]
```

```
sage.libs.lrcalc.lrcalc.test iterable to vector (it)
```

A wrapper function for the cdef function iterable_to_vector and vector_to_list, to test that they are working correctly.

EXAMPLES:

```
sage: from sage.libs.lrcalc.lrcalc import test_iterable_to_vector
sage: x = test_iterable_to_vector([3,2,1]); x
[3, 2, 1]
```

```
sage.libs.lrcalc.lrcalc.test_skewtab_to_SkewTableau (outer, inner)
```

A wrapper function for the cdef function skewtab_to_SkewTableau for testing purposes.

It constructs the first LR skew tableau of shape outer/inner as an lrcalc skewtab, and converts it to a SkewTableau.

```
sage: from sage.libs.lrcalc.lrcalc import test_skewtab_to_SkewTableau
sage: test_skewtab_to_SkewTableau([3,2,1],[])
[[1, 1, 1], [2, 2], [3]]
sage: test_skewtab_to_SkewTableau([4,3,2,1],[1,1]).pp()
. 1 1 1
. 2 2
1 3
2
```

TWENTYNINE

READLINE

This is the library behind the command line input, it takes keypresses until you hit Enter and then returns it as a string to Python. We hook into it so we can make it redraw the input area.

EXAMPLES:

```
sage: from sage.libs.readline import *
sage: replace_line('foobar', 0)
sage: set_point(3)
sage: print('current line: ' + repr(copy_text(0, get_end())))
current line: 'foobar'
sage: print('cursor position: {}'.format(get_point()))
cursor position: 3
```

When printing with <code>interleaved_output</code> the prompt and current line is removed:

```
sage: with interleaved_output():
...:    print('output')
...:    print('current line: ' + repr(copy_text(0, get_end())))
...:    print('cursor position: {}'.format(get_point()))
output
current line: ''
cursor position: 0
```

After the interleaved output, the line and cursor is restored to the old value:

```
sage: print('current line: ' + repr(copy_text(0, get_end())))
current line: 'foobar'
sage: print('cursor position: {}'.format(get_point()))
cursor position: 3
```

Finally, clear the current line for the remaining doctests:

```
sage: replace_line('', 1)
```

sage.libs.readline. ${\tt clear_signals}$ ()

Remove the readline signal handlers

Remove all of the Readline signal handlers installed by set_signals()

```
sage: from sage.libs.readline import clear_signals
sage: clear_signals()
0
```

```
sage.libs.readline.copy_text (pos_start, pos_end)
```

Return a copy of the text between start and end in the current line.

INPUT:

•pos_start, pos_end - integer. Start and end position.

OUTPUT:

String.

EXAMPLES:

```
sage: from sage.libs.readline import copy_text, replace_line
sage: replace_line('foobar', 0)
sage: copy_text(1, 5)
'ooba'
```

```
sage.libs.readline. forced_update_display ()
```

Force the line to be updated and redisplayed, whether or not Readline thinks the screen display is correct.

EXAMPLES:

```
sage: from sage.libs.readline import forced_update_display
sage: forced_update_display()
0
```

sage.libs.readline.get_end()

Get the end position of the current input

OUTPUT:

Integer

EXAMPLES:

```
sage: from sage.libs.readline import get_end
sage: get_end()
0
```

sage.libs.readline.get_point()

Get the cursor position

OUTPUT:

Integer

EXAMPLES:

```
sage: from sage.libs.readline import get_point, set_point
sage: get_point()
0
sage: set_point(5)
sage: get_point()
5
sage: set_point(0)
```

```
sage.libs.readline. initialize ()
```

Initialize or re-initialize Readline's internal state. It's not strictly necessary to call this; readline() calls it before reading any input.

```
sage: from sage.libs.readline import initialize
sage: initialize()
0
```

class sage.libs.readline.interleaved_output

Context manager for asynchronous output

This allows you to show output while at the readline prompt. When the block is left, the prompt is restored even if it was clobbered by the output.

EXAMPLES:

```
sage: from sage.libs.readline import interleaved_output
sage: with interleaved_output():
....: print('output')
output
```

sage.libs.readline.print_status()

Print readline status for debug purposes

EXAMPLES:

```
sage: from sage.libs.readline import print_status
sage: print_status()
catch_signals: 1
catch_sigwinch: 1
```

sage.libs.readline.replace_line (text, clear_undo)

Replace the contents of rl_line_buffer with text.

The point and mark are preserved, if possible.

INPUT:

- •text the new content of the line.
- •clear undo integer. If non-zero, the undo list associated with the current line is cleared.

EXAMPLES:

```
sage: from sage.libs.readline import copy_text, replace_line
sage: replace_line('foobar', 0)
sage: copy_text(1, 5)
'ooba'
```

```
sage.libs.readline. set_point ( point)
```

Set the cursor position

INPUT:

•point - integer. The new cursor position.

```
sage: from sage.libs.readline import get_point, set_point
sage: get_point()
0
sage: set_point(5)
sage: get_point()
5
sage: set_point(0)
```

```
sage.libs.readline.set_signals ()
```

Install the readline signal handlers

Install Readline's signal handler for SIGINT, SIGQUIT, SIGTERM, SIGALRM, SIGTSTP, SIGTTIN, SIGTTOU, and SIGWINCH, depending on the values of rl_catch_signals and rl_catch_sigwinch.

```
sage: from sage.libs.readline import set_signals
sage: set_signals()
0
```

CONTEXT MANAGERS FOR LIBGAP

This module implements a context manager for global variables. This is useful since the behavior of GAP is sometimes controlled by global variables, which you might want to switch to a different value for a computation. Here is an example how you are suppose to use it from your code. First, let us set a dummy global variable for our example:

```
sage: libgap.set_global('FooBar', 123)
```

Then, if you want to switch the value momentarily you can write:

```
sage: with libgap.global_context('FooBar', 'test'):
...: print(libgap.get_global('FooBar'))
test
```

Afterward, the global variable reverts to the previous value:

```
sage: print(libgap.get_global('FooBar'))
123
```

The value is reset even if exceptions occur:

```
sage: with libgap.global_context('FooBar', 'test'):
....:    print(libgap.get_global('FooBar'))
....:    raise ValueError(libgap.get_global('FooBar'))
Traceback (most recent call last):
...
ValueError: test
sage: print(libgap.get_global('FooBar'))
123
```

It is recommended that you use the $sage.libs.gap.libgap.Gap.global_context()$ method and not construct objects of this class manually.

INPUT:

- •variable string. The variable name.
- •value anything that defines a GAP object.

```
sage: libgap.get_global('FooBar')
1
```

CHAPTER	CHAPTER
THIRTYONE	RTYONE

GAP FUNCTIONS

CHAPTER

THIRTYTWO

LONG TESTS FOR LIBGAP

These stress test the garbage collection inside GAP

```
sage.libs.gap.test_long.test_loop_1 ()
    EXAMPLES:
```

```
sage: from sage.libs.gap.test_long import test_loop_1
sage: test_loop_1() # long time (up to 25s on sage.math, 2013)
```

sage.libs.gap.test_long.test_loop_2 ()

```
EXAMPLES:
```

```
sage: from sage.libs.gap.test_long import test_loop_2
sage: test_loop_2() # long time (10s on sage.math, 2013)
```

sage.libs.gap.test_long. ${\tt test_loop_3}$ ()

```
sage: from sage.libs.gap.test_long import test_loop_3
sage: test_loop_3() # long time (31s on sage.math, 2013)
```

CHAPTER

THIRTYTHREE

UTILITY FUNCTIONS FOR LIBGAP

```
class sage.libs.gap.util. ObjWrapper
    Bases: object
```

Wrapper for GAP master pointers

EXAMPLES:

```
sage: from sage.libs.gap.util import ObjWrapper
sage: x = ObjWrapper()
sage: y = ObjWrapper()
sage: x == y
True
```

```
sage.libs.gap.util. command ( command_string)
```

Playground for accessing Gap via libGap.

You should not use this function in your own programs. This is just here for convenience if you want to play with the libgap libray code.

EXAMPLES:

```
sage: from sage.libs.gap.util import command
sage: command('1')
Output follows...
1

sage: command('1/0')
Traceback (most recent call last):
...
ValueError: libGAP: Error, Rational operations: <divisor> must not be zero

sage: command('NormalSubgroups')
Output follows...
<Attribute "NormalSubgroups">

sage: command('rec(a:=1, b:=2)')
Output follows...
rec( a := 1, b := 2 )
```

```
sage.libs.gap.util.error_enter_libgap_block_twice()
```

Demonstrate that we catch errors from entering a block twice.

```
sage: from sage.libs.gap.util import error_enter_libgap_block_twice
sage: error_enter_libgap_block_twice()
```

```
Traceback (most recent call last):
...
RuntimeError: Entered a critical block twice
```

sage.libs.gap.util.error_exit_libgap_block_without_enter()

Demonstrate that we catch errors from omitting libgap_enter.

EXAMPLES:

```
sage: from sage.libs.gap.util import error_exit_libgap_block_without_enter
sage: error_exit_libgap_block_without_enter()
Traceback (most recent call last):
...
RuntimeError: Called libgap_exit without previous libgap_enter
```

```
sage.libs.gap.util.gap_root()
```

Find the location of the GAP root install which is stored in the gap startup script.

EXAMPLES:

```
sage: from sage.libs.gap.util import gap_root
sage: gap_root() # random output
'/home/vbraun/opt/sage-5.3.rc0/local/gap/latest'
```

```
sage.libs.gap.util. get_owned_objects ()
```

Helper to access the refcount dictionary from Python code

```
sage.libs.gap.util. memory_usage ()
```

Return information about the memory useage.

See mem () for details.

CHAPTER

THIRTYFOUR

LIBGAP SHARED LIBRARY INTERFACE TO GAP

This module implements a fast C library interface to GAP. To use libGAP you simply call libgap (the parent of all GapElement instances) and use it to convert Sage objects into GAP objects.

EXAMPLES:

```
sage: a = libgap(10)
sage: a
10
sage: type(a)
<type 'sage.libs.gap.element.GapElement_Integer'>
sage: a*a
100
sage: timeit('a*a') # random output
625 loops, best of 3: 898 ns per loop
```

Compared to the expect interface this is >1000 times faster:

```
sage: b = gap('10')
sage: timeit('b*b') # random output; long time
125 loops, best of 3: 2.05 ms per loop
```

If you want to evaluate GAP commands, use the Gap.eval () method:

```
sage: libgap.eval('List([1..10], i->i^2)')
[ 1, 4, 9, 16, 25, 36, 49, 64, 81, 100 ]
```

not to be confused with the libgap call, which converts Sage objects to GAP objects, for example strings to strings:

```
sage: libgap('List([1..10], i->i^2)')
"List([1..10], i->i^2)"
sage: type(_)
<type 'sage.libs.gap.element.GapElement_String'>
```

You can usually use the <code>sage()</code> method to convert the resulting GAP element back to its Sage equivalent:

```
sage: a.sage()
10
sage: type(_)
<type 'sage.rings.integer.Integer'>
sage: libgap.eval('5/3 + 7*E(3)').sage()
7*zeta3 + 5/3
sage: generators = libgap.AlternatingGroup(4).GeneratorsOfGroup().sage()
sage: generators # a Sage list of Sage permutations!
```

```
[(1,2,3), (2,3,4)]
sage: PermutationGroup(generators).cardinality() # computed in Sage
12
sage: libgap.AlternatingGroup(4).Size() # computed in GAP
12
```

So far, the following GAP data types can be directly converted to the corresponding Sage datatype:

- GAP booleans true / false to Sage booleans True / False. The third GAP boolean value fail raises a ValueError.
- 2. GAP integers to Sage integers.
- 3. GAP rational numbers to Sage rational numbers.
- 4. GAP cyclotomic numbers to Sage cyclotomic numbers.
- 5. GAP permutations to Sage permutations.
- 6. The GAP containers List and rec are converted to Sage containers list and dict. Furthermore, the sage () method is applied recursively to the entries.

Special support is available for the GAP container classes. GAP lists can be used as follows:

```
sage: lst = libgap([1,5,7]); lst
[ 1, 5, 7 ]
sage: type(lst)
<type 'sage.libs.gap.element.GapElement_List'>
sage: len(lst)
3
sage: lst[0]
1
sage: [ x^2 for x in lst ]
[1, 25, 49]
sage: type(_[0])
<type 'sage.libs.gap.element.GapElement_Integer'>
```

Note that you can access the elements of GAP List objects as you would expect from Python (with indexing starting at 0), but the elements are still of type <code>GapElement</code>. The other GAP container type are records, which are similar to Python dictionaries. You can construct them directly from Python dictionaries:

```
sage: libgap({'a':123, 'b':456})
rec( a := 123, b := 456 )
```

Or get them as results of computations:

```
sage: rec = libgap.eval('rec(a:=123, b:=456, Sym3:=SymmetricGroup(3))')
sage: rec['Sym3']
Sym([1..3])
sage: dict(rec)
{'Sym3': Sym([1..3]), 'a': 123, 'b': 456}
```

The output is a Sage dictionary whose keys are Sage strings and whose Values are instances of <code>GapElement()</code> . So, for example, <code>rec['a']</code> is not a Sage integer. To recursively convert the entries into Sage objects, you should use the <code>sage()</code> method:

```
sage: rec.sage()
{'Sym3': NotImplementedError('cannot construct equivalent Sage object',),
   'a': 123,
   'b': 456}
```

Now rec['a'] is a Sage integer. We have not implemented the conversion of the GAP symmetric group to the Sage symmetric group yet, so you end up with a NotImplementedError exception object. The exception is returned and not raised so that you can work with the partial result.

While we don't directly support matrices yet, you can convert them to Gap List of Lists. These lists are then easily converted into Sage using the recursive expansion of the sage () method:

34.1 Using the libGAP C library from Cython

The lower-case <code>libgap_foobar</code> functions are ones that we added to make the <code>libGAPC</code> shared library. The <code>libGAP_foobar</code> methods are the original GAP methods simply prefixed with the string <code>libGAP_</code>. The latter were originally not designed to be in a library, so some care needs to be taken to call them.

In particular, you must call <code>libgap_mark_stack_bottom()</code> in every function that calls into the libGAP C functions. The reason is that the GAP memory manager will automatically keep objects alive that are referenced in local (stack-allocated) variables. While convenient, this requires to look through the stack to find anything that looks like an address to a memory bag. But this requires vigilance against the following pattern:

```
cdef f()
  libgap_mark_stack_bottom()
  libGAP_function()

cdef g()
  libgap_mark_stack_bottom();
  f()  # f() changed the stack bottom marker
  libGAP_function() # boom
```

The solution is to re-order g() to first call f(). In order to catch this error, it is recommended that you wrap calls into libGAP in libgap_enter / libgap_exit blocks and not call libgap_mark_stack_bottom manually. So instead, always write

```
cdef f() libgap_enter() libGAP_function() libgap_exit()
cdef g() f() libgap_enter() libGAP_function() libgap_exit()
```

If you accidentally call libgap_enter() twice then an error message is printed to help you debug this:

```
sage: from sage.libs.gap.util import error_enter_libgap_block_twice
sage: error_enter_libgap_block_twice()
Traceback (most recent call last):
...
RuntimeError: Entered a critical block twice
```

AUTHORS:

- William Stein, Robert Miller (2009-06-23): first version
- Volker Braun, Dmitrii Pasechnik, Ivan Andrus (2011-03-25, Sage Days 29): almost complete rewrite; first usable version.
- Volker Braun (2012-08-28, GAP/Singular workshop): update to gap-4.5.5, make it ready for public consumption

```
class sage.libs.gap.libgap. Gap
    Bases: sage.structure.parent.Parent
```

The libgap interpreter object.

Note: This object must be instantiated exactly once by the libgap. Always use the provided libgap instance, and never instantiate *Gap* manually.

EXAMPLES:

```
sage: libgap.eval('SymmetricGroup(4)')
Sym([1..4])
```

Element

alias of GapElement

collect ()

Manually run the garbage collector

EXAMPLES:

```
sage: a = libgap(123)
sage: del a
sage: libgap.collect()
```

count_GAP_objects ()

Return the number of GAP objects that are being tracked by libGAP

OUTPUT:

An integer

EXAMPLES:

```
sage: libgap.count_GAP_objects() # random output
5
```

eval (gap_command)

Evaluate a gap command and wrap the result.

INPUT:

•gap_command – a string containing a valid gap command without the trailing semicolon.

OUTPUT:

A GapElement.

```
sage: libgap.eval('0')
0
sage: libgap.eval('"string"')
"string"
```

function_factory (function_name)

Return a GAP function wrapper

This is almost the same as calling libgap.eval (function_name), but faster and makes it obvious in your code that you are wrapping a function.

INPUT:

•function_name - string. The name of a GAP function.

OUTPUT:

A function wrapper <code>GapElement_Function</code> for the GAP function. Calling it from Sage is equivalent to calling the wrapped function from GAP.

EXAMPLES:

```
sage: libgap.function_factory('Print')
<Gap function "Print">
```

get_global (variable)

Get a GAP global variable

INPUT:

•variable – string. The variable name.

OUTPUT:

A GapElement wrapping the GAP output. A ValueError is raised if there is no such variable in GAP.

EXAMPLES:

```
sage: libgap.set_global('FooBar', 1)
sage: libgap.get_global('FooBar')
1
sage: libgap.unset_global('FooBar')
sage: libgap.get_global('FooBar')
Traceback (most recent call last):
...
ValueError: libGAP: Error, VAL_GVAR: No value bound to FooBar
```

global_context (variable, value)

Temporarily change a global variable

INPUT:

- •variable string. The variable name.
- •value anything that defines a GAP object.

OUTPUT:

A context manager that sets/reverts the given global variable.

```
sage: libgap.set_global('FooBar', 1)
sage: with libgap.global_context('FooBar', 2):
....: print(libgap.get_global('FooBar'))
2
sage: libgap.get_global('FooBar')
1
```

mem ()

Return information about libGAP memory usage

The GAP workspace is partitioned into 5 pieces (see gasman.c in the GAP sources for more details):

- •The masterpointer area contains all the masterpointers of the bags.
- •The **old bags area** contains the bodies of all the bags that survived at least one garbage collection. This area is only scanned for dead bags during a full garbage collection.
- •The **young bags area** contains the bodies of all the bags that have been allocated since the last garbage collection. This area is scanned for dead bags during each garbage collection.
- •The **allocation area** is the storage that is available for allocation of new bags. When a new bag is allocated the storage for the body is taken from the beginning of this area, and this area is correspondingly reduced. If the body does not fit in the allocation area a garbage collection is performed.
- •The **unavailable area** is the free storage that is not available for allocation.

OUTPUT:

This function returns a tuple containing 5 integers. Each is the size (in bytes) of the five partitions of the workspace. This will potentially change after each GAP garbage collection.

EXAMPLES:

```
sage: libgap.collect()
sage: libgap.mem()  # random output
(1048576, 6706782, 0, 960930, 0)

sage: libgap.FreeGroup(3)
<free group on the generators [ f1, f2, f3 ]>
sage: libgap.mem()  # random output
(1048576, 6706782, 47571, 913359, 0)

sage: libgap.collect()
sage: libgap.mem()  # random output
(1048576, 6734785, 0, 998463, 0)
```

one ()

Return (integer) one in GAP.

EXAMPLES:

```
sage: libgap.one()
1
sage: parent(_)
C library interface to GAP
```

set_global (variable, value)

Set a GAP global variable

INPUT:

•variable – string. The variable name.

•value – anything that defines a GAP object.

EXAMPLES:

```
sage: libgap.set_global('FooBar', 1)
sage: libgap.get_global('FooBar')
1
sage: libgap.unset_global('FooBar')
sage: libgap.get_global('FooBar')
Traceback (most recent call last):
...
ValueError: libGAP: Error, VAL_GVAR: No value bound to FooBar
```

show ()

Print statistics about the GAP owned object list

Slight complication is that we want to do it without accessing libgap objects, so we don't create new GapElements as a side effect.

EXAMPLES:

```
sage: a = libgap(123)
sage: b = libgap(456)
sage: c = libgap(789)
sage: del b
sage: libgap.show() # random output
11 LibGAP elements currently alive
rec(full := rec(cumulative := 122, deadbags := 9,
deadkb := 0, freekb := 7785, livebags := 304915,
livekb := 47367, time := 33, totalkb := 68608),
nfull := 3, npartial := 14)
```

unset_global (variable)

Remove a GAP global variable

INPUT:

•variable – string. The variable name.

EXAMPLES:

```
sage: libgap.set_global('FooBar', 1)
sage: libgap.get_global('FooBar')
1
sage: libgap.unset_global('FooBar')
sage: libgap.get_global('FooBar')
Traceback (most recent call last):
...
ValueError: libGAP: Error, VAL_GVAR: No value bound to FooBar
```

zero ()

Return (integer) zero in GAP.

OUTPUT:

A GapElement.

```
sage: libgap.zero()
0
```

zero_element (*args, **kwds)
Deprecated: Use zero() instead. See trac ticket #17694 for details.

CHAPTER

THIRTYFIVE

SHORT TESTS FOR LIBGAP

```
sage.libs.gap.test.test_write_to_file ()
```

Test that libgap can write to files

See trac ticket #16502, trac ticket #15833.

```
sage: from sage.libs.gap.test import test_write_to_file
sage: test_write_to_file()
```

Sage Reference Manual: C/C++ Library Interfaces, Rel	laasa 7 6
ouge receive manual. 0/0++ Library interfaces, rec	icase 7.0

LIBGAP ELEMENT WRAPPER

This document describes the individual wrappers for various GAP elements. For general information about libGAP, you should read the <code>libgap</code> module documentation.

```
class sage.libs.gap.element. GapElement
```

Bases: sage.structure.element.RingElement

Wrapper for all Gap objects.

Note: In order to create GapElements you should use the libgap instance (the parent of all Gap elements) to convert things into GapElement . You must not create GapElement instances manually.

EXAMPLES:

```
sage: libgap(0)
0
```

If Gap finds an error while evaluating, a corresponding assertion is raised:

```
sage: libgap.eval('1/0')
Traceback (most recent call last):
...
ValueError: libGAP: Error, Rational operations: <divisor> must not be zero
```

Also, a ValueError is raised if the input is not a simple expression:

```
sage: libgap.eval('1; 2; 3')
Traceback (most recent call last):
...
ValueError: can only evaluate a single statement
```

is_bool ()

Return whether the wrapped GAP object is a GAP boolean.

OUTPUT:

Boolean.

EXAMPLES:

```
sage: libgap(True).is_bool()
True
```

is function ()

Return whether the wrapped GAP object is a function.

OUTPUT:

Boolean.

EXAMPLES:

```
sage: a = libgap.eval("NormalSubgroups")
sage: a.is_function()
True
sage: a = libgap(2/3)
sage: a.is_function()
False
```

is_list()

Return whether the wrapped GAP object is a GAP List.

OUTPUT:

Boolean.

EXAMPLES:

```
sage: libgap.eval('[1, 2,,,, 5]').is_list()
True
sage: libgap.eval('3/2').is_list()
False
```

is_permutation ()

Return whether the wrapped GAP object is a GAP permutation.

OUTPUT:

Boolean.

EXAMPLES:

```
sage: perm = libgap.PermList( libgap([1,5,2,3,4]) ); perm
(2,5,4,3)
sage: perm.is_permutation()
True
sage: libgap('this is a string').is_permutation()
False
```

is_record ()

Return whether the wrapped GAP object is a GAP record.

OUTPUT:

Boolean.

EXAMPLES:

```
sage: libgap.eval('[1, 2,,,, 5]').is_record()
False
sage: libgap.eval('rec(a:=1, b:=3)').is_record()
True
```

is_string()

Return whether the wrapped GAP object is a GAP string.

OUTPUT:

Boolean.

EXAMPLES:

```
sage: libgap('this is a string').is_string()
True
```

sage ()

Return the Sage equivalent of the GapElement

EXAMPLES:

```
sage: libgap(1).sage()
sage: type(_)
<type 'sage.rings.integer.Integer'>
sage: libgap(3/7).sage()
3/7
sage: type(_)
<type 'sage.rings.rational.Rational'>
sage: libgap.eval('5 + 7 \times E(3)').sage()
7*zeta3 + 5
sage: libgap(Infinity).sage()
+Infinity
sage: libgap(-Infinity).sage()
-Infinity
sage: libgap(True).sage()
sage: libgap(False).sage()
False
sage: type(_)
<... 'bool'>
sage: libgap('this is a string').sage()
'this is a string'
sage: type(_)
<... 'str'>
```

class sage.libs.gap.element.GapElement_Boolean

Bases: sage.libs.gap.element.GapElement

Derived class of GapElement for GAP boolean values.

EXAMPLES:

```
sage: b = libgap(True)
sage: type(b)
<type 'sage.libs.gap.element.GapElement_Boolean'>
```

sage (

Return the Sage equivalent of the GapElement

OUTPUT:

A Python boolean if the values is either true or false. GAP booleans can have the third value Fail, in which case a ValueError is raised.

```
sage: b = libgap.eval('true'); b
true
sage: type(_)
<type 'sage.libs.gap.element.GapElement_Boolean'>
sage: b.sage()
True
sage: type(_)
<... 'bool'>

sage: libgap.eval('fail')
fail
sage: _.sage()
Traceback (most recent call last):
...
ValueError: the GAP boolean value "fail" cannot be represented in Sage
```

class sage.libs.gap.element. GapElement_Cyclotomic

Bases: sage.libs.gap.element.GapElement

Derived class of GapElement for GAP universal cyclotomics.

EXAMPLES:

```
sage: libgap.eval('E(3)')
E(3)
sage: type(_)
<type 'sage.libs.gap.element.GapElement_Cyclotomic'>
```

sage (ring=None)

Return the Sage equivalent of the <code>GapElement_Cyclotomic</code> .

INPLIT

•ring – a Sage cyclotomic field or None (default). If not specified, a suitable minimal cyclotomic field will be constructed.

OUTPUT:

A Sage cyclotomic field element.

EXAMPLES:

```
sage: n = libgap.eval('E(3)')
sage: n.sage()
zeta3
sage: parent(_)
Cyclotomic Field of order 3 and degree 2

sage: n.sage(ring=CyclotomicField(6))
zeta6 - 1

sage: libgap.E(3).sage(ring=CyclotomicField(3))
zeta3
sage: libgap.E(3).sage(ring=CyclotomicField(6))
zeta6 - 1
```

class sage.libs.gap.element. GapElement_FiniteField

Bases: sage.libs.gap.element.GapElement

Derived class of GapElement for GAP finite field elements.

EXAMPLES:

```
sage: libgap.eval('Z(5)^2')
Z(5)^2
sage: type(_)
<type 'sage.libs.gap.element_GapElement_FiniteField'>
```

lift()

Return an integer lift.

OUTPUT:

The smallest positive <code>GapElement_Integer</code> that equals self in the prime finite field.

EXAMPLES:

```
sage: n = libgap.eval('Z(5)^2')
sage: n.lift()
4
sage: type(_)
<type 'sage.libs.gap.element.GapElement_Integer'>
sage: n = libgap.eval('Z(25)')
sage: n.lift()
Traceback (most recent call last):
TypeError: not in prime subfield
```

sage (ring=None, var='a')

Return the Sage equivalent of the <code>GapElement_FiniteField</code>.

INPUT:

•ring -a Sage finite field or None (default). The field to return self in. If not specified, a suitable finite field will be constructed.

OUTPUT:

An Sage finite field element. The isomorphism is chosen such that the Gap PrimitiveRoot() maps to the Sage multiplicative_generator().

EXAMPLES:

```
sage: n = libgap.eval('Z(25)^2')
sage: n.sage()
a + 3
sage: parent(_)
Finite Field in a of size 5^2

sage: n.sage(ring=GF(5))
Traceback (most recent call last):
...
ValueError: the given finite field has incompatible size
```

```
class sage.libs.gap.element.GapElement_Function
```

Bases: sage.libs.gap.element.GapElement

Derived class of GapElement for GAP functions.

```
sage: f = libgap.Cycles
sage: type(f)
<type 'sage.libs.gap.element.GapElement_Function'>
```

class sage.libs.gap.element. GapElement_Integer

Bases: sage.libs.gap.element.GapElement

Derived class of GapElement for GAP integers.

EXAMPLES:

```
sage: i = libgap(123)
sage: type(i)
<type 'sage.libs.gap.element.GapElement_Integer'>
sage: ZZ(i)
123
```

is C int ()

Return whether the wrapped GAP object is a immediate GAP integer.

An immediate integer is one that is stored as a C integer, and is subject to the usual size limits. Larger integers are stored in GAP as GMP integers.

OUTPUT:

Boolean.

EXAMPLES:

```
sage: n = libgap(1)
sage: type(n)
<type 'sage.libs.gap.element.GapElement_Integer'>
sage: n.is_C_int()
True
sage: n.IsInt()
true

sage: N = libgap(2^130)
sage: type(N)
<type 'sage.libs.gap.element.GapElement_Integer'>
sage: N.is_C_int()
False
sage: N.IsInt()
true
```

sage (ring=None)

Return the Sage equivalent of the GapElement_Integer

•ring - Integer ring or None (default). If not specified, a the default Sage integer ring is used.

OUTPUT:

A Sage integer

```
sage: libgap([ 1, 3, 4 ]).sage()
[1, 3, 4]
sage: all( x in ZZ for x in _ )
True
```

```
sage: libgap(132).sage(ring=IntegerModRing(13))
2
sage: parent(_)
Ring of integers modulo 13
```

class sage.libs.gap.element. GapElement_IntegerMod

Bases: sage.libs.gap.element.GapElement

Derived class of GapElement for GAP integers modulo an integer.

EXAMPLES:

```
sage: n = IntegerModRing(123)(13)
sage: i = libgap(n)
sage: type(i)
<type 'sage.libs.gap.element.GapElement_IntegerMod'>
```

lift()

Return an integer lift.

OUTPUT:

A GapElement_Integer that equals self in the integer mod ring.

EXAMPLES:

```
sage: n = libgap.eval('One(ZmodnZ(123)) * 13')
sage: n.lift()
13
sage: type(_)
<type 'sage.libs.gap.element.GapElement_Integer'>
```

sage (ring=None)

Return the Sage equivalent of the GapElement_IntegerMod

INPUT:

•ring - Sage integer mod ring or None (default). If not specified, a suitable integer mod ringa is used automatically.

OUTPUT:

A Sage integer modulo another integer.

EXAMPLES:

```
sage: n = libgap.eval('One(ZmodnZ(123)) * 13')
sage: n.sage()
13
sage: parent(_)
Ring of integers modulo 123
```

class sage.libs.gap.element. GapElement List

Bases: sage.libs.gap.element.GapElement

Derived class of GapElement for GAP Lists.

Note: Lists are indexed by 0..len(l) - 1, as expected from Python. This differs from the GAP convention where lists start at 1.

EXAMPLES:

```
sage: lst = libgap.SymmetricGroup(3).List(); lst
[ (), (1,3), (1,2,3), (2,3), (1,3,2), (1,2) ]
sage: type(lst)
<type 'sage.libs.gap.element.GapElement_List'>
sage: len(lst)
6
sage: lst[3]
(2,3)
```

We can easily convert a Gap List object into a Python list:

```
sage: list(lst)
[(), (1,3), (1,2,3), (2,3), (1,3,2), (1,2)]
sage: type(_)
<... 'list'>
```

Range checking is performed:

```
sage: lst[10]
Traceback (most recent call last):
...
IndexError: index out of range.
```

matrix (ring=None)

Return the list as a matrix.

GAP does not have a special matrix data type, they are just lists of lists. This function converts a GAP list of lists to a Sage matrix.

OUTPUT:

A Sage matrix.

```
sage: F = libgap.GF(4)
sage: a = F.PrimitiveElement()
sage: m = libgap([[a,a^0],[0*a,a^2]]); m
[ [Z(2^2), Z(2)^0],
 [0*Z(2), Z(2^2)^2]
sage: m.IsMatrix()
true
sage: matrix(m)
  a 1]
    0 a + 1
sage: matrix(GF(4, 'B'), m)
  в 1]
    0 B + 1]
sage: M = libgap.eval('SL(2,GF(5))').GeneratorsOfGroup()[1]
sage: type(M)
<type 'sage.libs.gap.element.GapElement_List'>
sage: M[0][0]
Z(5)^2
sage: M.IsMatrix()
t.rue
sage: M.matrix()
```

```
[4 1]
[4 0]
```

sage (**kwds)

Return the Sage equivalent of the GapElement

OUTPUT:

A Python list.

EXAMPLES:

```
sage: libgap([ 1, 3, 4 ]).sage()
[1, 3, 4]
sage: all( x in ZZ for x in _ )
True
```

```
vector ( ring=None)
```

Return the list as a vector.

GAP does not have a special vetor data type, they are just lists. This function converts a GAP list to a Sage vector.

OUTPUT:

A Sage vector.

EXAMPLES:

```
sage: F = libgap.GF(4)
sage: a = F.PrimitiveElement()
sage: m = libgap([0*a, a, a^3, a^2]); m
[ 0*Z(2), Z(2^2), Z(2)^0, Z(2^2)^2 ]
sage: type(m)
<type 'sage.libs.gap.element.GapElement_List'>
sage: m[3]
Z(2^2)^2
sage: vector(m)
(0, a, 1, a + 1)
sage: vector(GF(4,'B'), m)
(0, B, 1, B + 1)
```

```
class sage.libs.gap.element. GapElement_MethodProxy
```

```
Bases: sage.libs.gap.element.GapElement_Function
```

Helper class returned by GapElement.__getattr__.

Derived class of GapElement for GAP functions. Like its parent, you can call instances to implement function call syntax. The only difference is that a fixed first argument is prepended to the argument list.

```
sage: lst = libgap([])
sage: lst.Add
<Gap function "Add">
sage: type(_)
<type 'sage.libs.gap.element_MethodProxy'>
sage: lst.Add(1)
sage: lst
[ 1 ]
```

```
class sage.libs.gap.element.GapElement_Permutation
```

Bases: sage.libs.gap.element.GapElement

Derived class of GapElement for GAP permutations.

Note: Permutations in GAP act on the numbers starting with 1.

EXAMPLES:

```
sage: perm = libgap.eval('(1,5,2)(4,3,8)')
sage: type(perm)
<type 'sage.libs.gap.element_GapElement_Permutation'>
```

sage ()

Return the Sage equivalent of the GapElement

EXAMPLES:

```
sage: perm_gap = libgap.eval('(1,5,2)(4,3,8)'); perm_gap
(1,5,2)(3,8,4)
sage: perm_gap.sage()
(1,5,2)(3,8,4)
sage: type(_)
<type 'sage.groups.perm_gps.permgroup_element.PermutationGroupElement'>
```

class sage.libs.gap.element. GapElement Rational

Bases: sage.libs.gap.element.GapElement

Derived class of GapElement for GAP rational numbers.

EXAMPLES:

```
sage: r = libgap(123/456)
sage: type(r)
<type 'sage.libs.gap.element_GapElement_Rational'>
```

sage (ring=None)

Return the Sage equivalent of the GapElement.

INPUT:

•ring - the Sage rational ring or None (default). If not specified, the rational ring is used automatically.

OUTPUT:

A Sage rational number.

```
sage: r = libgap(123/456); r
41/152
sage: type(_)
<type 'sage.libs.gap.element.GapElement_Rational'>
sage: r.sage()
41/152
sage: type(_)
<type 'sage.rings.rational.Rational'>
```

```
class sage.libs.gap.element.GapElement_Record
```

Bases: sage.libs.gap.element.GapElement

Derived class of GapElement for GAP records.

EXAMPLES:

```
sage: rec = libgap.eval('rec(a:=123, b:=456)')
sage: type(rec)
<type 'sage.libs.gap.element.GapElement_Record'>
sage: len(rec)
2
sage: rec['a']
123
```

We can easily convert a Gap rec object into a Python dict:

```
sage: dict(rec)
{'a': 123, 'b': 456}
sage: type(_)
<... 'dict'>
```

Range checking is performed:

record_name_to_index (py_name)

Convert string to GAP record index.

INPUT:

•py_name - a python string.

OUTPUT:

A UInt, which is a GAP hash of the string. If this is the first time the string is encountered, a new integer is returned(!)

EXAMPLES:

```
sage: rec = libgap.eval('rec(first:=123, second:=456)')
sage: rec.record_name_to_index('first') # random output
1812L
sage: rec.record_name_to_index('no_such_name') # random output
3776L
```

sage ()

Return the Sage equivalent of the GapElement

```
sage: libgap.eval('rec(a:=1, b:=2)').sage()
{'a': 1, 'b': 2}
sage: all( isinstance(key,str) and val in ZZ for key,val in _.items() )
True
sage: rec = libgap.eval('rec(a:=123, b:=456, Sym3:=SymmetricGroup(3))')
```

```
sage: rec.sage()
{'Sym3': NotImplementedError('cannot construct equivalent Sage object',),
    'a': 123,
    'b': 456}
```

class sage.libs.gap.element.GapElement_RecordIterator

Bases: object

Iterator for GapElement_Record

Since Cython does not support generators yet, we implement the older iterator specification with this auxiliary class.

INPUT:

•rec - the GapElement_Record to iterate over.

EXAMPLES:

```
sage: rec = libgap.eval('rec(a:=123, b:=456)')
sage: list(rec)
[('a', 123), ('b', 456)]
sage: dict(rec)
{'a': 123, 'b': 456}
```

next ()

x.next() -> the next value, or raise StopIteration

class sage.libs.gap.element.GapElement_Ring

Bases: sage.libs.gap.element.GapElement

Derived class of GapElement for GAP rings (parents of ring elements).

EXAMPLES:

```
sage: i = libgap(ZZ)
sage: type(i)
<type 'sage.libs.gap.element.GapElement_Ring'>
```

ring_cyclotomic()

Construct an integer ring.

EXAMPLES:

```
sage: libgap.CyclotomicField(6).ring_cyclotomic()
Cyclotomic Field of order 3 and degree 2
```

```
ring_finite_field (var='a')
```

Construct an integer ring.

EXAMPLES:

```
sage: libgap.GF(3,2).ring_finite_field(var='A')
Finite Field in A of size 3^2
```

```
ring_integer ()
```

Construct the Sage integers.

```
sage: libgap.eval('Integers').ring_integer()
Integer Ring
```

ring_integer_mod ()

Construct a Sage integer mod ring.

EXAMPLES:

```
sage: libgap.eval('ZmodnZ(15)').ring_integer_mod()
Ring of integers modulo 15
```

ring_rational ()

Construct the Sage rationals.

EXAMPLES:

```
sage: libgap.eval('Rationals').ring_rational()
Rational Field
```

sage (**kwds)

Return the Sage equivalent of the <code>GapElement_Ring</code> .

INPUT:

•**kwds - keywords that are passed on to the ring_ method.

OUTPUT:

A Sage ring.

EXAMPLES:

```
sage: libgap.eval('Integers').sage()
Integer Ring

sage: libgap.eval('Rationals').sage()
Rational Field

sage: libgap.eval('ZmodnZ(15)').sage()
Ring of integers modulo 15

sage: libgap.GF(3,2).sage(var='A')
Finite Field in A of size 3^2

sage: libgap.CyclotomicField(6).sage()
Cyclotomic Field of order 3 and degree 2
```

class sage.libs.gap.element. GapElement_String

Bases: sage.libs.gap.element.GapElement

Derived class of GapElement for GAP strings.

```
sage: s = libgap('string')
sage: type(s)
<type 'sage.libs.gap.element_String'>
sage: s
"string"
sage: print(s)
string
```

```
sage ()
```

Convert this <code>GapElement_String</code> to a Python string.

OUTPUT:

A Python string.

```
sage: s = libgap.eval(' "string" '); s
"string"
sage: type(_)
<type 'sage.libs.gap.element.GapElement_String'>
sage: str(s)
'string'
sage: s.sage()
'string'
sage: type(_)
<... 'str'>
```

CHAPTER

THIRTYSEVEN

LIBGAP WORKSPACE SUPPORT

The single purpose of this module is to provide the location of the libgap saved workspace and a time stamp to invalidate saved workspaces.

```
sage.libs.gap.saved_workspace.timestamp ()
    Return a time stamp for libgap
```

OUTPUT:

Float. Unix timestamp of the most recently changed LibGAP file.

EXAMPLES:

```
sage: from sage.libs.gap.saved_workspace import timestamp
sage: timestamp() # random output
1406642467.25684
sage: type(timestamp())
<... 'float'>
```

```
sage.libs.gap.saved_workspace. workspace ( name='workspace')
```

Return the filename of the gap workspace and whether it is up to date.

INPUT:

•name - string. A name that will become part of the workspace filename.

OUTPUT:

Pair consisting of a string and a boolean. The string is the filename of the saved libgap workspace (or that it should have if it doesn't exist). The boolean is whether the workspace is up-to-date. You may use the workspace file only if the boolean is True.

```
sage: from sage.libs.gap.saved_workspace import workspace
sage: ws, up_to_date = workspace()
sage: ws
'/.../gap/libgap-workspace-...'
sage: isinstance(up_to_date, bool)
True
```

Sage Reference Manual: C/C++ Library Interfaces, Release 7.6		

LIBRARY INTERFACE TO EMBEDDABLE COMMON LISP (ECL)

```
class sage.libs.ecl. EclListIterator
    Bases: object
```

Iterator object for an ECL list

This class is used to implement the iterator protocol for EclObject. Do not instantiate this class directly but use the iterator method on an EclObject instead. It is an error if the EclObject is not a list.

EXAMPLES:

```
sage: from sage.libs.ecl import *
sage: I=EclListIterator(EclObject("(1 2 3)"))
sage: type(I)
<type 'sage.libs.ecl.EclListIterator'>
sage: [i for i in I]
[<ECL: 1>, <ECL: 2>, <ECL: 3>]
sage: [i for i in EclObject("(1 2 3)")]
[<ECL: 1>, <ECL: 2>, <ECL: 3>]
sage: EclListIterator(EclObject("1"))
Traceback (most recent call last):
...
TypeError: ECL object is not iterable
```

next ()

x.next() -> the next value, or raise StopIteration

```
class sage.libs.ecl. EclObject
```

Bases: object

Python wrapper of ECL objects

The Eclobject forms a wrapper around ECL objects. The wrapper ensures that the data structure pointed to is protected from garbage collection in ECL by installing a pointer to it from a global data structure within the scope of the ECL garbage collector. This pointer is destroyed upon destruction of the Eclobject.

EclObject() takes a Python object and tries to find a representation of it in Lisp.

EXAMPLES:

Python lists get mapped to LISP lists. None and Boolean values to appropriate values in LISP:

```
sage: from sage.libs.ecl import *
sage: EclObject([None,true,false])
<ECL: (NIL T NIL)>
```

Numerical values are translated to the appropriate type in LISP:

Floats in Python are IEEE double, which LISP has as well. However, the printing of floating point types in LISP depends on settings:

```
sage: a = EclObject(float(10^40))
sage: ecl_eval("(setf *read-default-float-format* 'single-float)")
<ECL: SINGLE-FLOAT>
sage: a
<ECL: 1.d40>
sage: ecl_eval("(setf *read-default-float-format* 'double-float)")
<ECL: DOUBLE-FLOAT>
sage: a
<ECL: 1.e40>
```

Tuples are translated to dotted lists:

```
sage: EclObject( (false, true))
<ECL: (NIL . T)>
```

Strings are fed to the reader, so a string normally results in a symbol:

```
sage: EclObject("Symbol")
<ECL: SYMBOL>
```

But with proper quotation one can construct a lisp string object too:

```
sage: EclObject('"Symbol"')
<ECL: "Symbol">
```

EclObjects translate to themselves, so one can mix:

```
sage: EclObject([1,2,EclObject([3])])
<ECL: (1 2 (3))>
```

Calling an EclObject translates into the appropriate LISP apply , where the argument is transformed into an EclObject itself, so one can flexibly apply LISP functions:

```
sage: car=EclObject("car")
sage: cdr=EclObject("cdr")
sage: car(cdr([1,2,3]))
<ECL: 2>
```

and even construct and evaluate arbitrary S-expressions:

```
sage: eval=EclObject("eval")
sage: quote=EclObject("quote")
sage: eval([car, [cdr, [quote, [1,2,3]]]])
<ECL: 2>
```

atomp ()

Return True if self is atomic, False otherwise.

```
sage: from sage.libs.ecl import *
sage: EclObject([]).atomp()
True
sage: EclObject([[]]).atomp()
False
```

caar ()

Return the caar of self

EXAMPLES:

```
sage: from sage.libs.ecl import *
sage: L=EclObject([[1,2],[3,4]])
sage: L.car()
<ECL: (1 2)>
sage: L.cdr()
<ECL: ((3 4))>
sage: L.caar()
<ECL: 1>
sage: L.cadr()
<ECL: (3 4)>
sage: L.cdar()
<ECL: (3 4)>
sage: L.cdar()
<ECL: (1)</pre>
```

cadr ()

Return the cadr of self

EXAMPLES:

```
sage: from sage.libs.ecl import *
sage: L=EclObject([[1,2],[3,4]])
sage: L.car()
<ECL: (1 2)>
sage: L.cdr()
<ECL: ((3 4))>
sage: L.caar()
<ECL: 1>
sage: L.cadr()
<ECL: (3 4)>
sage: L.cddr()
<ECL: (2)>
sage: L.cddr()
```

car ()

Return the car of self

```
sage: from sage.libs.ecl import *
sage: L=EclObject([[1,2],[3,4]])
sage: L.car()
<ECL: (1 2)>
sage: L.cdr()
<ECL: ((3 4))>
sage: L.caar()
<ECL: 1>
```

```
sage: L.cadr()
<ECL: (3 4)>
sage: L.cdar()
<ECL: (2)>
sage: L.cddr()
<ECL: NIL>
```

cdar ()

Return the cdar of self

EXAMPLES:

```
sage: from sage.libs.ecl import *
sage: L=EclObject([[1,2],[3,4]])
sage: L.car()
<ECL: (1 2)>
sage: L.cdr()
<ECL: ((3 4))>
sage: L.caar()
<ECL: 1>
sage: L.cadr()
<ECL: (3 4)>
sage: L.cdr()
<ECL: (3 4)>
sage: L.cdr()
<ECL: (2)>
sage: L.cddr()
<ECL: (NIL>
```

cddr ()

Return the cddr of self

EXAMPLES:

```
sage: from sage.libs.ecl import *
sage: L=EclObject([[1,2],[3,4]])
sage: L.car()
<ECL: (1 2)>
sage: L.cdr()
<ECL: ((3 4))>
sage: L.caar()
<ECL: 1>
sage: L.cadr()
<ECL: (3 4)>
sage: L.cddr()
<ECL: (3 2)>
sage: L.cddr()
<ECL: (2)>
sage: L.cddr()
```

cdr ()

Return the cdr of self

```
sage: from sage.libs.ecl import *
sage: L=EclObject([[1,2],[3,4]])
sage: L.car()
<ECL: (1 2)>
sage: L.cdr()
<ECL: ((3 4))>
sage: L.caar()
```

```
<ECL: 1>
sage: L.cadr()
<ECL: (3 4)>
sage: L.cdar()
<ECL: (2)>
sage: L.cddr()
```

characterp ()

Return True if self is a character, False otherwise

Strings are not characters

EXAMPLES:

```
sage: from sage.libs.ecl import * sage: EclObject("'a"').characterp() False
```

cons(d)

apply cons to self and argument and return the result.

EXAMPLES:

```
sage: from sage.libs.ecl import *
sage: a=EclObject(1)
sage: b=EclObject(2)
sage: a.cons(b)
<ECL: (1 . 2)>
```

consp ()

Return True if self is a cons, False otherwise. NIL is not a cons.

EXAMPLES:

```
sage: from sage.libs.ecl import *
sage: EclObject([]).consp()
False
sage: EclObject([[]]).consp()
True
```

eval ()

Evaluate object as an S-Expression

EXAMPLES:

```
sage: from sage.libs.ecl import *
sage: S=EclObject("(+ 1 2)")
sage: S
<ECL: (+ 1 2)>
sage: S.eval()
<ECL: 3>
```

fixnump ()

Return True if self is a fixnum, False otherwise

```
sage: from sage.libs.ecl import *
sage: EclObject(2**3).fixnump()
True
```

```
sage: EclObject(2**200).fixnump()
False
```

listp()

Return True if self is a list, False otherwise. NIL is a list.

EXAMPLES:

```
sage: from sage.libs.ecl import *
sage: EclObject([]).listp()
True
sage: EclObject([[]]).listp()
True
```

nullp ()

Return True if self is NIL, False otherwise

EXAMPLES:

```
sage: from sage.libs.ecl import *
sage: EclObject([]).nullp()
True
sage: EclObject([[]]).nullp()
False
```

python ()

Convert an EclObject to a python object.

EXAMPLES:

```
sage: from sage.libs.ecl import *
sage: L=EclObject([1,2,("three",'"four"')])
sage: L.python()
[1, 2, ('THREE', '"four"')]
```

rplaca (d)

Destructively replace car(self) with d.

EXAMPLES:

```
sage: from sage.libs.ecl import *
sage: L=EclObject((1,2))
sage: L
<ECL: (1 . 2)>
sage: a=EclObject(3)
sage: L.rplaca(a)
sage: L
<ECL: (3 . 2)>
```

rplacd (d)

Destructively replace cdr(self) with d.

```
sage: from sage.libs.ecl import *
sage: L=EclObject((1,2))
sage: L
<ECL: (1 . 2)>
sage: a=EclObject(3)
```

```
sage: L.rplacd(a)
sage: L
<ECL: (1 . 3)>
```

symbolp ()

Return True if self is a symbol, False otherwise.

EXAMPLES:

```
sage: from sage.libs.ecl import *
sage: EclObject([]).symbolp()
True
sage: EclObject([[]]).symbolp()
False
```

sage.libs.ecl. ecl_eval (s)

Read and evaluate string in Lisp and return the result

EXAMPLES:

```
sage.libs.ecl. init_ecl ()
```

Internal function to initialize ecl. Do not call.

This function initializes the ECL library for use within Python. This routine should only be called once and importing the ecl library interface already does that, so do not call this yourself.

EXAMPLES:

```
sage: from sage.libs.ecl import *
```

At this point, init_ecl() has run. Explicitly executing it gives an error:

```
sage: init_ecl()
Traceback (most recent call last):
...
RuntimeError: ECL is already initialized
```

```
sage.libs.ecl. print_objects ()
```

Print GC-protection list

Diagnostic function. ECL objects that are bound to Python objects need to be protected from being garbage collected. We do this by including them in a doubly linked list bound to the global ECL symbol *SAGE-LIST-OF-OBJECTS*. Only non-immediate values get included, so small integers do not get linked in. This routine prints the values currently stored.

```
sage: from sage.libs.ecl import *
sage: a=EclObject("hello")
sage: b=EclObject(10)
sage: c=EclObject("world")
sage: print_objects() #random because previous test runs can have left objects
```

```
NIL
WORLD
HELLO
```

```
sage.libs.ecl. shutdown_ecl ()
```

Shut down ecl. Do not call.

Given the way that ECL is used from python, it is very difficult to ensure that no ECL objects exist at a particular time. Hence, destroying ECL is a risky proposition.

```
sage: from sage.libs.ecl import *
sage: shutdown_ecl()
```

```
sage.libs.ecl. test_ecl_options ()
    Print an overview of the ECL options
sage.libs.ecl. test_sigint_before_ecl_sig_on ()
```

CHAPTER THIRTYNINE

GSL ARRAYS

CHAPTER

FORTY

INDICES AND TABLES

- Index
- Module Index
- Search Page

```
g
sage.gsl.gsl_array,219
sage.libs.ecl, 211
sage.libs.eclib.constructor, 29
sage.libs.eclib.homspace, 25
sage.libs.eclib.interface, 3
sage.libs.eclib.mat, 21
sage.libs.eclib.mwrank, 19
sage.libs.eclib.newforms, 23
sage.libs.flint.arith, 147
sage.libs.flint.flint, 141
sage.libs.flint.fmpz_poly, 143
sage.libs.gap.context_managers, 177
sage.libs.gap.element, 195
sage.libs.gap.gap_functions, 179
sage.libs.gap.libgap, 185
sage.libs.gap.saved_workspace, 209
sage.libs.gap.test, 193
sage.libs.gap.test_long, 181
sage.libs.gap.util, 183
sage.libs.lcalc.lcalc_Lfunction, 31
sage.libs.libecm, 163
sage.libs.linbox.linbox, 139
sage.libs.lrcalc.lrcalc, 165
sage.libs.mpmath.utils, 157
sage.libs.ntl.all, 161
sage.libs.pari, 39
sage.libs.pari.convert_sage, 43
sage.libs.ppl, 79
sage.libs.ratpoints, 49
sage.libs.readline, 173
sage.libs.singular.function, 51
sage.libs.singular.function_factory,61
sage.libs.singular.groebner_strategy,75
sage.libs.singular.option, 67
```

Sage Reference Manual: C/C++ Library Interfaces, Release 7.6

```
sage.libs.singular.polynomial,65
sage.libs.singular.ring,73
sage.libs.singular.singular,63
sage.libs.symmetrica.symmetrica,149

r
sage.rings.pari_ring,47
```

224 Python Module Index

Α add constraint() (sage.libs.ppl.MIP Problem method), 103 add_constraint() (sage.libs.ppl.Polyhedron method), 113 add_constraints() (sage.libs.ppl.MIP_Problem method), 103 add_constraints() (sage.libs.ppl.Polyhedron method), 114 add_generator() (sage.libs.ppl.Polyhedron method), 114 add_generators() (sage.libs.ppl.Polyhedron method), 115 add scalar() (sage.libs.eclib.mat.Matrix method), 21 add space dimensions and embed() (sage.libs.ppl.MIP Problem method), 103 add_space_dimensions_and_embed() (sage.libs.ppl.Polyhedron method), 116 add_space_dimensions_and_project() (sage.libs.ppl.Polyhedron method), 117 add_to_integer_space_dimensions() (sage.libs.ppl.MIP_Problem method), 103 affine dimension() (sage.libs.ppl.Polyhedron method), 117 ainvs() (sage.libs.eclib.interface.mwrank_EllipticCurve method), 4 all_homogeneous_terms_are_zero() (sage.libs.ppl.Linear_Expression method), 100 all singular poly wrapper() (in module sage.libs.singular.function), 54 all vectors() (in module sage.libs.singular.function), 55 ascii_dump() (sage.libs.ppl.Constraint method), 83 ascii dump() (sage.libs.ppl.Constraint System method), 87 ascii dump() (sage.libs.ppl.Generator method), 91 ascii_dump() (sage.libs.ppl.Generator_System method), 97 ascii_dump() (sage.libs.ppl.Linear_Expression method), 100 ascii_dump() (sage.libs.ppl.Poly_Con_Relation method), 109 ascii_dump() (sage.libs.ppl.Poly_Gen_Relation method), 111 ascii_dump() (sage.libs.ppl.Polyhedron method), 117 ascii_dump() (sage.libs.ppl.Variables_Set method), 136 atomp() (sage.libs.ecl.EclObject method), 212 В BaseCallHandler (class in sage.libs.singular.function), 52 bdg_symmetrica() (in module sage.libs.symmetrica.symmetrica), 149 bell number() (in module sage.libs.flint.arith), 147 bernoulli number() (in module sage.libs.flint.arith), 147 bitcount() (in module sage.libs.mpmath.utils), 157 bounds from above() (sage.libs.ppl.Polyhedron method), 118 bounds from below() (sage.libs.ppl.Polyhedron method), 119

C

```
C Polyhedron (class in sage.libs.ppl), 81
caar() (sage.libs.ecl.EclObject method), 213
cadr() (sage.libs.ecl.EclObject method), 213
call() (in module sage.libs.mpmath.utils), 157
car() (sage.libs.ecl.EclObject method), 213
cdar() (sage.libs.ecl.EclObject method), 214
cddr() (sage.libs.ecl.EclObject method), 214
cdr() (sage.libs.ecl.EclObject method), 214
certain() (sage.libs.eclib.interface.mwrank_EllipticCurve method), 4
characteristic() (sage.libs.singular.function.RingWrap method), 52
characteristic() (sage.rings.pari ring.PariRing method), 47
characterp() (sage.libs.ecl.EclObject method), 215
charpoly() (sage.libs.eclib.mat.Matrix method), 21
charpoly() (sage.libs.linbox.linbox.Linbox_integer_dense method), 139
chartafel_symmetrica() (in module sage.libs.symmetrica.symmetrica), 149
charvalue symmetrica() (in module sage.libs.symmetrica.symmetrica), 149
clear() (sage.libs.ppl.Constraint_System method), 88
clear() (sage.libs.ppl.Generator_System method), 97
clear() (sage.libs.ppl.MIP Problem method), 104
clear_signals() (in module sage.libs.readline), 173
closure point() (in module sage.libs.ppl), 137
closure point() (sage.libs.ppl.Generator static method), 91
coefficient() (sage.libs.ppl.Constraint method), 83
coefficient() (sage.libs.ppl.Generator method), 91
coefficient() (sage.libs.ppl.Linear_Expression method), 100
coefficients() (sage.libs.ppl.Constraint method), 83
coefficients() (sage.libs.ppl.Generator method), 92
coefficients() (sage.libs.ppl.Linear_Expression method), 101
collect() (sage.libs.gap.libgap.Gap method), 188
command() (in module sage.libs.gap.util), 183
compute elmsym with alphabet symmetrica() (in module sage.libs.symmetrica.symmetrica), 149
compute homsym with alphabet symmetrica() (in module sage.libs.symmetrica.symmetrica), 150
compute monomial with alphabet symmetrica() (in module sage.libs.symmetrica.symmetrica), 150
compute powsym with alphabet symmetrica() (in module sage.libs.symmetrica.symmetrica), 150
compute rank() (sage.libs.lcalc.lcalc Lfunction.Lfunction method), 31
compute_schur_with_alphabet_det_symmetrica() (in module sage.libs.symmetrica.symmetrica), 151
compute schur with alphabet symmetrica() (in module sage.libs.symmetrica.symmetrica), 151
concatenate assign() (sage.libs.ppl.Polyhedron method), 119
conductor() (sage.libs.eclib.interface.mwrank_EllipticCurve method), 5
cons() (sage.libs.ecl.EclObject method), 215
consp() (sage.libs.ecl.EclObject method), 215
constrains() (sage.libs.ppl.Polyhedron method), 120
Constraint (class in sage.libs.ppl), 82
Constraint System (class in sage.libs.ppl), 87
Constraint System iterator (class in sage.libs.ppl), 89
constraints() (sage.libs.ppl.Polyhedron method), 120
contains() (sage.libs.ppl.Polyhedron method), 121
contains_integer_point() (sage.libs.ppl.Polyhedron method), 121
```

```
Converter (class in sage.libs.singular.function), 52
coprod() (in module sage.libs.lrcalc.lrcalc), 167
copy text() (in module sage.libs.readline), 173
count GAP objects() (sage.libs.gap.libgap.Gap method), 188
CPS_height_bound() (sage.libs.eclib.interface.mwrank_EllipticCurve method), 4
CremonaModularSymbols() (in module sage.libs.eclib.constructor), 29
currRing wrapper() (in module sage.libs.singular.ring), 73
D
dedekind_sum() (in module sage.libs.flint.arith), 147
degree() (sage.libs.flint.fmpz_poly.Fmpz_poly method), 143
derivative() (sage.libs.flint.fmpz poly.Fmpz poly method), 143
det() (sage.libs.linbox.linbox.Linbox integer dense method), 139
difference_assign() (sage.libs.ppl.Polyhedron method), 122
dimension() (sage.libs.eclib.homspace.ModularSymbols method), 25
dimension schur symmetrica() (in module sage.libs.symmetrica.symmetrica), 151
dimension symmetrization symmetrica() (in module sage.libs.symmetrica.symmetrica), 151
div_rem() (sage.libs.flint.fmpz_poly.Fmpz_poly method), 143
divdiff_perm_schubert_symmetrica() (in module sage.libs.symmetrica.symmetrica), 151
divdiff schubert symmetrica() (in module sage.libs.symmetrica.symmetrica), 151
divisor() (sage.libs.ppl.Generator method), 92
drop_some_non_integer_points() (sage.libs.ppl.Polyhedron method), 122
ecl eval() (in module sage.libs.ecl), 217
EclListIterator (class in sage.libs.ecl), 211
EclObject (class in sage.libs.ecl), 211
ecmfactor() (in module sage.libs.libecm), 163
ECModularSymbol (class in sage.libs.eclib.newforms), 23
Element (sage.libs.gap.libgap.Gap attribute), 188
Element (sage.rings.pari ring.PariRing attribute), 47
empty() (sage.libs.ppl.Constraint_System method), 88
empty() (sage.libs.ppl.Generator_System method), 98
end() (in module sage.libs.symmetrica.symmetrica), 151
equation() (in module sage.libs.ppl), 137
error_enter_libgap_block_twice() (in module sage.libs.gap.util), 183
error exit libgap block without enter() (in module sage.libs.gap.util), 184
euler_number() (in module sage.libs.flint.arith), 147
eval() (sage.libs.ecl.EclObject method), 215
eval() (sage.libs.gap.libgap.Gap method), 188
evaluate objective function() (sage.libs.ppl.MIP Problem method), 104
F
find_zeros() (sage.libs.lcalc.lcalc_Lfunction.Lfunction method), 31
find zeros via N() (sage.libs.lcalc.lcalc Lfunction.Lfunction method), 32
fixnump() (sage.libs.ecl.EclObject method), 215
Fmpz poly (class in sage.libs.flint.fmpz poly), 143
forced_update_display() (in module sage.libs.readline), 174
free_flint_stack() (in module sage.libs.flint.flint), 141
from man exp() (in module sage.libs.mpmath.utils), 158
```

```
function factory() (sage.libs.gap.libgap.Gap method), 189
G
Gap (class in sage.libs.gap.libgap), 188
gap root() (in module sage.libs.gap.util), 184
GapElement (class in sage.libs.gap.element), 195
GapElement_Boolean (class in sage.libs.gap.element), 197
GapElement_Cyclotomic (class in sage.libs.gap.element), 198
GapElement FiniteField (class in sage.libs.gap.element), 198
GapElement_Function (class in sage.libs.gap.element), 199
GapElement_Integer (class in sage.libs.gap.element), 200
GapElement IntegerMod (class in sage.libs.gap.element), 201
GapElement_List (class in sage.libs.gap.element), 201
GapElement_MethodProxy (class in sage.libs.gap.element), 203
GapElement Permutation (class in sage.libs.gap.element), 203
GapElement Rational (class in sage.libs.gap.element), 204
GapElement Record (class in sage.libs.gap.element), 204
GapElement_RecordIterator (class in sage.libs.gap.element), 206
GapElement_Ring (class in sage.libs.gap.element), 206
GapElement String (class in sage.libs.gap.element), 207
gen_to_sage() (in module sage.libs.pari.convert_sage), 43
Generator (class in sage.libs.ppl), 90
Generator System (class in sage.libs.ppl), 96
Generator_System_iterator (class in sage.libs.ppl), 98
generators() (sage.libs.ppl.Polyhedron method), 123
gens() (sage.libs.eclib.interface.mwrank EllipticCurve method), 5
get end() (in module sage.libs.readline), 174
get_global() (sage.libs.gap.libgap.Gap method), 189
get_owned_objects() (in module sage.libs.gap.util), 184
get point() (in module sage.libs.readline), 174
get precision() (in module sage.libs.eclib.interface), 3
get_precision() (in module sage.libs.eclib.mwrank), 19
global_context() (sage.libs.gap.libgap.Gap method), 189
GlobalVariableContext (class in sage.libs.gap.context managers), 177
GroebnerStrategy (class in sage.libs.singular.groebner strategy), 75
gupta_nm_symmetrica() (in module sage.libs.symmetrica.symmetrica), 151
gupta tafel symmetrica() (in module sage.libs.symmetrica.symmetrica), 151
Н
hall littlewood symmetrica() (in module sage.libs.symmetrica.symmetrica), 152
hardy z function() (sage.libs.lcalc.lcalc Lfunction.Lfunction method), 33
harmonic number() (in module sage.libs.flint.arith), 148
has_equalities() (sage.libs.ppl.Constraint_System method), 88
has_strict_inequalities() (sage.libs.ppl.Constraint_System method), 89
hecke_matrix() (sage.libs.eclib.homspace.ModularSymbols method), 25
id() (sage.libs.ppl.Variable method), 135
ideal() (sage.libs.singular.groebner_strategy.GroebnerStrategy method), 75
ideal() (sage.libs.singular.groebner strategy.NCGroebnerStrategy method), 76
```

```
implies() (sage.libs.ppl.Poly Con Relation method), 109
implies() (sage.libs.ppl.Poly_Gen_Relation method), 111
inequality() (in module sage.libs.ppl), 137
inhomogeneous term() (sage.libs.ppl.Constraint method), 84
inhomogeneous_term() (sage.libs.ppl.Linear_Expression method), 101
init_ecl() (in module sage.libs.ecl), 217
initialize() (in module sage.libs.readline), 174
initprimes() (in module sage.libs.eclib.mwrank), 19
insert() (sage.libs.ppl.Constraint_System method), 89
insert() (sage.libs.ppl.Generator System method), 98
insert() (sage.libs.ppl.Variables Set method), 136
interleaved output (class in sage.libs.readline), 175
intersection assign() (sage.libs.ppl.Polyhedron method), 123
is bool() (sage.libs.gap.element.GapElement method), 195
is bounded() (sage.libs.ppl.Polyhedron method), 124
is_C_int() (sage.libs.gap.element.GapElement_Integer method), 200
is_closure_point() (sage.libs.ppl.Generator method), 92
is commutative() (sage.libs.singular.function.RingWrap method), 53
is cuspidal() (sage.libs.eclib.homspace.ModularSymbols method), 26
is_discrete() (sage.libs.ppl.Polyhedron method), 124
is disjoint() (sage.libs.ppl.Poly Con Relation static method), 110
is disjoint from() (sage.libs.ppl.Polyhedron method), 125
is_empty() (sage.libs.ppl.Polyhedron method), 125
is_equality() (sage.libs.ppl.Constraint method), 84
is_equivalent_to() (sage.libs.ppl.Constraint method), 84
is equivalent to() (sage.libs.ppl.Generator method), 93
is_field() (sage.rings.pari_ring.PariRing method), 47
is_function() (sage.libs.gap.element.GapElement method), 195
is_included() (sage.libs.ppl.Poly_Con_Relation static method), 110
is inconsistent() (sage.libs.ppl.Constraint method), 85
is inequality() (sage.libs.ppl.Constraint method), 85
is line() (sage.libs.ppl.Generator method), 93
is line or ray() (sage.libs.ppl.Generator method), 93
is_list() (sage.libs.gap.element.GapElement method), 196
is nonstrict inequality() (sage.libs.ppl.Constraint method), 85
is_permutation() (sage.libs.gap.element.GapElement method), 196
is_point() (sage.libs.ppl.Generator method), 94
is ray() (sage.libs.ppl.Generator method), 94
is_record() (sage.libs.gap.element.GapElement method), 196
is sage wrapper for singular ring() (in module sage.libs.singular.function), 55
is satisfiable() (sage.libs.ppl.MIP Problem method), 105
is_singular_poly_wrapper() (in module sage.libs.singular.function), 55
is_strict_inequality() (sage.libs.ppl.Constraint method), 86
is string() (sage.libs.gap.element.GapElement method), 196
is_tautological() (sage.libs.ppl.Constraint method), 86
is topologically closed() (sage.libs.ppl.Polyhedron method), 125
is_universe() (sage.libs.ppl.Polyhedron method), 126
is_zero() (sage.libs.ppl.Linear_Expression method), 101
isogeny class() (sage.libs.eclib.interface.mwrank EllipticCurve method), 5
isqrt() (in module sage.libs.mpmath.utils), 159
```

K KernelCallHandler (class in sage.libs.singular.function), 52 kostka number symmetrica() (in module sage.libs.symmetrica.symmetrica), 152 kostka_tab_symmetrica() (in module sage.libs.symmetrica.symmetrica), 152 kostka tafel symmetrica() (in module sage.libs.symmetrica.symmetrica), 152 kranztafel symmetrica() (in module sage.libs.symmetrica.symmetrica), 153 left_shift() (sage.libs.flint.fmpz_poly.Fmpz_poly method), 144 level() (sage.libs.eclib.homspace.ModularSymbols method), 26 Lfunction (class in sage.libs.lcalc.lcalc Lfunction), 31 Lfunction_C (class in sage.libs.lcalc.lcalc_Lfunction), 34 Lfunction D (class in sage.libs.lcalc.lcalc Lfunction), 35 Lfunction from character() (in module sage.libs.lcalc.lcalc Lfunction), 36 Lfunction_from_elliptic_curve() (in module sage.libs.lcalc.lcalc_Lfunction), 36 Lfunction_I (class in sage.libs.lcalc.lcalc_Lfunction), 35 Lfunction Zeta (class in sage.libs.lcalc.lcalc Lfunction), 36 lib() (in module sage.libs.singular.function), 55 LibraryCallHandler (class in sage.libs.singular.function), 52 LibSingularOptions (class in sage.libs.singular.option), 68 LibSingularOptions_abstract (class in sage.libs.singular.option), 71 LibSingularOptionsContext (class in sage.libs.singular.option), 70 LibSingularVerboseOptions (class in sage.libs.singular.option), 71 lift() (sage.libs.gap.element.GapElement FiniteField method), 199 lift() (sage.libs.gap.element.GapElement IntegerMod method), 201 Linbox_integer_dense (class in sage.libs.linbox.linbox), 139 Linbox_modn_sparse (class in sage.libs.linbox.linbox), 139 line() (in module sage.libs.ppl), 137 line() (sage.libs.ppl.Generator static method), 94 Linear_Expression (class in sage.libs.ppl), 99 list() (sage.libs.flint.fmpz_poly.Fmpz_poly method), 144 list of functions() (in module sage.libs.singular.function), 56 listp() (sage.libs.ecl.EclObject method), 216 load() (sage.libs.singular.option.LibSingularOptions_abstract method), 71 lrcoef() (in module sage.libs.lrcalc.lrcalc), 168 lrcoef unsafe() (in module sage.libs.lrcalc.lrcalc), 168 lrskew() (in module sage.libs.lrcalc.lrcalc), 169 Μ Matrix (class in sage.libs.eclib.mat), 21 matrix() (sage.libs.gap.element.GapElement_List method), 202 MatrixFactory (class in sage.libs.eclib.mat), 22 max_space_dimension() (sage.libs.ppl.Polyhedron method), 126 maximize() (sage.libs.ppl.Polyhedron method), 126 mem() (sage.libs.gap.libgap.Gap method), 190 memory usage() (in module sage.libs.gap.util), 184 minimize() (sage.libs.ppl.Polyhedron method), 127 minimized_constraints() (sage.libs.ppl.Polyhedron method), 128 minimized_generators() (sage.libs.ppl.Polyhedron method), 128 minpoly() (sage.libs.linbox.linbox.Linbox integer dense method), 139

```
MIP Problem (class in sage.libs.ppl), 102
ModularSymbols (class in sage.libs.eclib.homspace), 25
mpmath to sage() (in module sage.libs.mpmath.utils), 159
mult() (in module sage.libs.lrcalc.lrcalc), 169
mult_monomial_monomial_symmetrica() (in module sage.libs.symmetrica.symmetrica), 153
mult_schubert() (in module sage.libs.lrcalc.lrcalc), 170
mult schubert schubert symmetrica() (in module sage.libs.symmetrica.symmetrica), 153
mult schubert variable symmetrica() (in module sage.libs.symmetrica.symmetrica), 153
mult_schur_schur_symmetrica() (in module sage.libs.symmetrica.symmetrica), 153
mwrank EllipticCurve (class in sage.libs.eclib.interface), 3
mwrank MordellWeil (class in sage.libs.eclib.interface), 9
Ν
NCGroebnerStrategy (class in sage.libs.singular.groebner_strategy), 76
ncols() (sage.libs.eclib.mat.Matrix method), 21
ndg symmetrica() (in module sage.libs.symmetrica.symmetrica), 153
newtrans symmetrica() (in module sage.libs.symmetrica.symmetrica), 154
next() (sage.libs.ecl.EclListIterator method), 211
next() (sage.libs.gap.element.GapElement RecordIterator method), 206
next() (sage.libs.ppl.Constraint System iterator method), 90
next() (sage.libs.ppl.Generator_System_iterator method), 99
ngens() (sage.libs.singular.function.RingWrap method), 53
NNC Polyhedron (class in sage.libs.ppl), 107
normal form() (sage.libs.singular.groebner strategy.GroebnerStrategy method), 75
normal_form() (sage.libs.singular.groebner_strategy.NCGroebnerStrategy method), 76
normalize() (in module sage.libs.mpmath.utils), 160
nothing() (sage.libs.ppl.Poly Con Relation static method), 110
nothing() (sage.libs.ppl.Poly_Gen_Relation static method), 112
npars() (sage.libs.singular.function.RingWrap method), 53
nrows() (sage.libs.eclib.mat.Matrix method), 22
nullp() (sage.libs.ecl.EclObject method), 216
number_of_cusps() (sage.libs.eclib.homspace.ModularSymbols method), 26
number_of_partitions() (in module sage.libs.flint.arith), 148
O
objective function() (sage.libs.ppl.MIP Problem method), 105
ObjWrapper (class in sage.libs.gap.util), 183
odd_to_strict_part_symmetrica() (in module sage.libs.symmetrica.symmetrica), 154
odg_symmetrica() (in module sage.libs.symmetrica.symmetrica), 154
OK() (sage.libs.ppl.Constraint method), 83
OK() (sage.libs.ppl.Constraint_System method), 87
OK() (sage.libs.ppl.Generator method), 90
OK() (sage.libs.ppl.Generator System method), 97
OK() (sage.libs.ppl.Linear_Expression method), 100
OK() (sage.libs.ppl.MIP Problem method), 102
OK() (sage.libs.ppl.Poly_Con_Relation method), 109
OK() (sage.libs.ppl.Poly_Gen_Relation method), 111
OK() (sage.libs.ppl.Polyhedron method), 112
OK() (sage.libs.ppl.Variable method), 135
OK() (sage.libs.ppl.Variables_Set method), 136
```

```
one() (sage.libs.gap.libgap.Gap method), 190
opt (sage.libs.singular.option.LibSingularOptionsContext attribute), 71
optimal_value() (sage.libs.ppl.MIP_Problem method), 105
optimization mode() (sage.libs.ppl.MIP Problem method), 106
optimizing_point() (sage.libs.ppl.MIP_Problem method), 106
ordering_string() (sage.libs.singular.function.RingWrap method), 53
outerproduct schur symmetrica() (in module sage.libs.symmetrica.symmetrica), 154
Р
par_names() (sage.libs.singular.function.RingWrap method), 54
Pari (class in sage.rings.pari_ring), 47
PariRing (class in sage.rings.pari_ring), 47
part skewschur symmetrica() (in module sage.libs.symmetrica.symmetrica), 154
plethysm_symmetrica() (in module sage.libs.symmetrica.symmetrica), 154
point() (in module sage.libs.ppl), 138
point() (sage.libs.ppl.Generator static method), 95
points() (sage.libs.eclib.interface.mwrank MordellWeil method), 12
poison currRing() (in module sage.libs.singular.ring), 73
Poly_Con_Relation (class in sage.libs.ppl), 108
poly difference assign() (sage.libs.ppl.Polyhedron method), 129
Poly Gen Relation (class in sage.libs.ppl), 111
poly_hull_assign() (sage.libs.ppl.Polyhedron method), 130
Polyhedron (class in sage.libs.ppl), 112
pow truncate() (sage.libs.flint.fmpz poly.Fmpz poly method), 144
print_currRing() (in module sage.libs.singular.ring), 73
print objects() (in module sage.libs.ecl), 217
print_status() (in module sage.libs.readline), 175
process() (sage.libs.eclib.interface.mwrank MordellWeil method), 12
pseudo_div() (sage.libs.flint.fmpz_poly.Fmpz_poly method), 144
pseudo_div_rem() (sage.libs.flint.fmpz_poly.Fmpz_poly method), 144
python() (sage.libs.ecl.EclObject method), 216
Q
q_core_symmetrica() (in module sage.libs.symmetrica.symmetrica), 154
R
random_element() (sage.rings.pari_ring.PariRing method), 47
random_partition_symmetrica() (in module sage.libs.symmetrica.symmetrica), 154
rank() (sage.libs.eclib.interface.mwrank EllipticCurve method), 5
rank() (sage.libs.eclib.interface.mwrank_MordellWeil method), 14
rank_bound() (sage.libs.eclib.interface.mwrank_EllipticCurve method), 6
ratpoints() (in module sage.libs.ratpoints), 49
ray() (in module sage.libs.ppl), 138
ray() (sage.libs.ppl.Generator static method), 95
record_name_to_index() (sage.libs.gap.element.GapElement_Record method), 205
regulator() (sage.libs.eclib.interface.mwrank EllipticCurve method), 6
regulator() (sage.libs.eclib.interface.mwrank MordellWeil method), 14
relation_with() (sage.libs.ppl.Polyhedron method), 130
remove_higher_space_dimensions() (sage.libs.ppl.Polyhedron method), 131
replace line() (in module sage.libs.readline), 175
```

```
reset default() (sage.libs.singular.option.LibSingularOptions method), 70
reset_default() (sage.libs.singular.option.LibSingularVerboseOptions method), 72
Resolution (class in sage.libs.singular.function), 52
right shift() (sage.libs.flint.fmpz poly.Fmpz poly method), 144
ring() (sage.libs.singular.function.Converter method), 52
ring() (sage.libs.singular.groebner_strategy.GroebnerStrategy method), 75
ring() (sage.libs.singular.groebner strategy.NCGroebnerStrategy method), 76
ring cyclotomic() (sage.libs.gap.element.GapElement Ring method), 206
ring_finite_field() (sage.libs.gap.element.GapElement_Ring method), 206
ring_integer() (sage.libs.gap.element.GapElement_Ring method), 206
ring integer mod() (sage.libs.gap.element.GapElement Ring method), 207
ring rational() (sage.libs.gap.element.GapElement Ring method), 207
ring_wrapper_Py (class in sage.libs.singular.ring), 74
RingWrap (class in sage.libs.singular.function), 52
rplaca() (sage.libs.ecl.EclObject method), 216
rplacd() (sage.libs.ecl.EclObject method), 216
S
sage() (sage.libs.gap.element.GapElement method), 197
sage() (sage.libs.gap.element.GapElement_Boolean method), 197
sage() (sage.libs.gap.element.GapElement Cyclotomic method), 198
sage() (sage.libs.gap.element.GapElement_FiniteField method), 199
sage() (sage.libs.gap.element.GapElement Integer method), 200
sage() (sage.libs.gap.element.GapElement_IntegerMod method), 201
sage() (sage.libs.gap.element.GapElement List method), 203
sage() (sage.libs.gap.element.GapElement Permutation method), 204
sage() (sage.libs.gap.element.GapElement_Rational method), 204
sage() (sage.libs.gap.element.GapElement_Record method), 205
sage() (sage.libs.gap.element.GapElement Ring method), 207
sage() (sage.libs.gap.element_GapElement_String method), 207
sage.gsl.gsl array (module), 219
sage.libs.ecl (module), 211
sage.libs.eclib.constructor (module), 29
sage.libs.eclib.homspace (module), 25
sage.libs.eclib.interface (module), 3
sage.libs.eclib.mat (module), 21
sage.libs.eclib.mwrank (module), 19
sage.libs.eclib.newforms (module), 23
sage.libs.flint.arith (module), 147
sage.libs.flint.flint (module), 141
sage.libs.flint.fmpz_poly (module), 143
sage.libs.gap.context_managers (module), 177
sage.libs.gap.element (module), 195
sage.libs.gap.gap_functions (module), 179
sage.libs.gap.libgap (module), 185
sage.libs.gap.saved_workspace (module), 209
sage.libs.gap.test (module), 193
sage.libs.gap.test long (module), 181
sage.libs.gap.util (module), 183
sage.libs.lcalc.lcalc_Lfunction (module), 31
```

```
sage.libs.libecm (module), 163
sage.libs.linbox.linbox (module), 139
sage.libs.lrcalc.lrcalc (module), 165
sage.libs.mpmath.utils (module), 157
sage.libs.ntl.all (module), 161
sage.libs.pari (module), 39
sage.libs.pari.convert sage (module), 43
sage.libs.ppl (module), 79
sage.libs.ratpoints (module), 49
sage.libs.readline (module), 173
sage.libs.singular.function (module), 51
sage.libs.singular.function factory (module), 61
sage.libs.singular.groebner_strategy (module), 75
sage.libs.singular.option (module), 67
sage.libs.singular.polynomial (module), 65
sage.libs.singular.ring (module), 73
sage.libs.singular.singular (module), 63
sage.libs.symmetrica.symmetrica (module), 149
sage.rings.pari ring (module), 47
sage_matrix_over_ZZ() (sage.libs.eclib.mat.Matrix method), 22
sage to mpmath() (in module sage.libs.mpmath.utils), 160
saturate() (sage.libs.eclib.interface.mwrank EllipticCurve method), 7
saturate() (sage.libs.eclib.interface.mwrank_MordellWeil method), 15
saturates() (sage.libs.ppl.Poly_Con_Relation static method), 110
save() (sage.libs.singular.option.LibSingularOptions_abstract method), 71
scalarproduct schubert symmetrica() (in module sage.libs.symmetrica.symmetrica), 154
scalarproduct_schur_symmetrica() (in module sage.libs.symmetrica.symmetrica), 154
schur_schur_plet_symmetrica() (in module sage.libs.symmetrica.symmetrica), 154
sdg symmetrica() (in module sage.libs.symmetrica.symmetrica), 154
search() (sage.libs.eclib.interface.mwrank MordellWeil method), 17
selmer rank() (sage.libs.eclib.interface.mwrank EllipticCurve method), 7
set global() (sage.libs.gap.libgap.Gap method), 190
set objective function() (sage.libs.ppl.MIP Problem method), 106
set_optimization_mode() (sage.libs.ppl.MIP_Problem method), 106
set point() (in module sage.libs.readline), 175
set_precision() (in module sage.libs.eclib.interface), 18
set_precision() (in module sage.libs.eclib.mwrank), 20
set signals() (in module sage.libs.readline), 175
set_verbose() (sage.libs.eclib.interface.mwrank_EllipticCurve method), 8
show() (sage.libs.gap.libgap.Gap method), 191
shutdown ecl() (in module sage.libs.ecl), 218
sign() (sage.libs.eclib.homspace.ModularSymbols method), 26
silverman_bound() (sage.libs.eclib.interface.mwrank_EllipticCurve method), 9
singular function() (in module sage.libs.singular.function), 56
SingularFunction (class in sage.libs.singular.function), 54
SingularFunctionFactory (class in sage.libs.singular.function factory), 61
SingularKernelFunction (class in sage.libs.singular.function), 54
SingularLibraryFunction (class in sage.libs.singular.function), 54
skew() (in module sage.libs.lrcalc.lrcalc), 171
smithform() (sage.libs.linbox.linbox.Linbox_integer_dense method), 139
```

```
solve() (sage.libs.ppl.MIP Problem method), 107
space_dimension() (sage.libs.ppl.Constraint method), 86
space_dimension() (sage.libs.ppl.Constraint_System method), 89
space dimension() (sage.libs.ppl.Generator method), 96
space_dimension() (sage.libs.ppl.Generator_System method), 98
space_dimension() (sage.libs.ppl.Linear_Expression method), 101
space dimension() (sage.libs.ppl.MIP Problem method), 107
space dimension() (sage.libs.ppl.Polyhedron method), 132
space_dimension() (sage.libs.ppl.Variable method), 135
space dimension() (sage.libs.ppl.Variables Set method), 136
specht dg symmetrica() (in module sage.libs.symmetrica.symmetrica), 154
start() (in module sage.libs.symmetrica.symmetrica), 154
str() (sage.libs.eclib.mat.Matrix method), 22
strict inequality() (in module sage.libs.ppl), 138
strict to odd part symmetrica() (in module sage.libs.symmetrica.symmetrica), 154
strictly_contains() (sage.libs.ppl.Polyhedron method), 132
strictly_intersects() (sage.libs.ppl.Poly_Con_Relation static method), 110
subsumes() (sage.libs.ppl.Poly Gen Relation static method), 112
symbolp() (sage.libs.ecl.EclObject method), 217
t_ELMSYM_HOMSYM_symmetrica() (in module sage.libs.symmetrica.symmetrica), 154
t ELMSYM MONOMIAL symmetrica() (in module sage.libs.symmetrica.symmetrica), 154
t ELMSYM POWSYM symmetrica() (in module sage.libs.symmetrica.symmetrica), 154
t_ELMSYM_SCHUR_symmetrica() (in module sage.libs.symmetrica.symmetrica), 155
t HOMSYM ELMSYM symmetrica() (in module sage.libs.symmetrica.symmetrica), 155
t HOMSYM MONOMIAL symmetrica() (in module sage.libs.symmetrica.symmetrica), 155
t_HOMSYM_POWSYM_symmetrica() (in module sage.libs.symmetrica.symmetrica), 155
t_HOMSYM_SCHUR_symmetrica() (in module sage.libs.symmetrica.symmetrica), 155
t MONOMIAL ELMSYM symmetrica() (in module sage.libs.symmetrica.symmetrica), 155
t MONOMIAL HOMSYM symmetrica() (in module sage.libs.symmetrica.symmetrica), 155
t_MONOMIAL_POWSYM_symmetrica() (in module sage.libs.symmetrica.symmetrica), 155
t_MONOMIAL_SCHUR_symmetrica() (in module sage.libs.symmetrica.symmetrica), 155
t POLYNOM ELMSYM symmetrica() (in module sage.libs.symmetrica.symmetrica), 155
t POLYNOM MONOMIAL symmetrica() (in module sage.libs.symmetrica.symmetrica), 155
t POLYNOM POWER symmetrica() (in module sage.libs.symmetrica.symmetrica), 155
t POLYNOM SCHUBERT symmetrica() (in module sage.libs.symmetrica.symmetrica), 155
t POLYNOM SCHUR symmetrica() (in module sage.libs.symmetrica.symmetrica), 155
t_POWSYM_ELMSYM_symmetrica() (in module sage.libs.symmetrica.symmetrica), 155
t POWSYM HOMSYM symmetrica() (in module sage, libs, symmetrica, symmetrica), 155
t_POWSYM_MONOMIAL_symmetrica() (in module sage.libs.symmetrica.symmetrica), 155
t_POWSYM_SCHUR_symmetrica() (in module sage.libs.symmetrica.symmetrica), 155
t_SCHUBERT_POLYNOM_symmetrica() (in module sage.libs.symmetrica.symmetrica), 155
t_SCHUR_ELMSYM_symmetrica() (in module sage.libs.symmetrica.symmetrica), 155
t SCHUR HOMSYM symmetrica() (in module sage.libs.symmetrica.symmetrica), 155
t SCHUR MONOMIAL symmetrica() (in module sage.libs.symmetrica.symmetrica), 155
t_SCHUR_POWSYM_symmetrica() (in module sage.libs.symmetrica.symmetrica), 155
test ecl options() (in module sage.libs.ecl), 218
test integer() (in module sage.libs.symmetrica.symmetrica), 155
test_iterable_to_vector() (in module sage.libs.lrcalc.lrcalc), 171
```

```
test loop_1() (in module sage.libs.gap.test_long), 181
test_loop_2() (in module sage.libs.gap.test_long), 181
test_loop_3() (in module sage.libs.gap.test_long), 181
test sigint before ecl sig on() (in module sage.libs.ecl), 218
test_skewtab_to_SkewTableau() (in module sage.libs.lrcalc.lrcalc), 171
test_write_to_file() (in module sage.libs.gap.test), 193
timestamp() (in module sage.libs.gap.saved workspace), 209
topological_closure_assign() (sage.libs.ppl.Polyhedron method), 132
trait_names() (sage.libs.singular.function_factory.SingularFunctionFactory method), 61
truncate() (sage.libs.flint.fmpz_poly.Fmpz_poly method), 145
two descent() (sage.libs.eclib.interface.mwrank EllipticCurve method), 9
type() (sage.libs.ppl.Constraint method), 87
type() (sage.libs.ppl.Generator method), 96
U
unconstrain() (sage.libs.ppl.Polyhedron method), 133
unpickle GroebnerStrategy() (in module sage.libs.singular.groebner strategy), 77
unpickle_NCGroebnerStrategy0() (in module sage.libs.singular.groebner_strategy), 77
unset_global() (sage.libs.gap.libgap.Gap method), 191
upper bound assign() (sage.libs.ppl.Polyhedron method), 133
V
value() (sage.libs.lcalc.lcalc_Lfunction.Lfunction method), 33
var_names() (sage.libs.singular.function.RingWrap method), 54
Variable (class in sage.libs.ppl), 134
Variables_Set (class in sage.libs.ppl), 135
vector() (sage.libs.gap.element.GapElement_List method), 203
W
workspace() (in module sage.libs.gap.saved workspace), 209
Ζ
zero() (sage.libs.gap.libgap.Gap method), 191
zero_element() (sage.libs.gap.libgap.Gap method), 191
zeta() (sage.rings.pari ring.PariRing method), 47
```