Abstract

Interface to Axiom

Axiom is a free GPL-compatible (modified BSD license) general purpose computer algebra system whose development started in 1973 at IBM. It contains symbolic manipulation algorithms, as well as implementations of special functions, including elliptic functions and generalized hypergeometric functions. Moreover, Axiom has implementations of many functions relating to the invariant theory of the symmetric group S_n. For many links to Axiom documentation see

http://wiki.axiom-developer.org.

EXAMPLES: We evaluate a very simple expression in axiom.

```
sage: axiom('3 * 5')
15
                                                    Type: PositiveInteger
```

We factor $x^5 - y^5$ in Axiom in several different ways. The first way yields a Axiom object.

```
sage: F = axiom.factor('x^5 - y^5')
sage: type(F)
<class 'axiom.interfaces.axiom.AxiomElement'>
```

Note that Axiom objects are normally displayed using "ASCII art"; to see a normal linear representation of any Axiom object x, use `str(x). no emph`

```
sage: F
                                4      3    2  2    3      4
                    - (y - x) (y  + x y  + x  y  + x  y + x )
```

```
You can always use x.str() to obtain the linear representation
of an object, even without changing the display2d flag.  This can
be useful for moving axiom data to other systems.
```

```
sage: F.str()
'-(y - x)*(y^4 + x*y^3 + x^2*y^2 + x^3*y + x^4)'

sage: axiom.display2d(False)
sage: F
-(y - x)*(y^4 + x*y^3 + x^2*y^2 + x^3*y + x^4)
```

```
The axiom.eval command evaluates an expression in axiom
   and returns the result as a string.
```

```
sage: print axiom.eval('factor(x^5 - y^5)')
-(y - x)*(y^4 + x*y^3 + x^2*y^2 + x^3*y + x^4)
\end{varbatim}
We can create the polynomial $f$ as a Axiom polynomial, then call
the factor method on it.  Notice that the notation \code{f.factor()}
is consistent with how the rest of \sage works.
\begin{verbatim}
sage: f = axiom('x^5 - y^5')
sage: f^2
(x^5 - y^5)^2
sage: f.factor()
-(y - x)*(y^4 + x*y^3 + x^2*y^2 + x^3*y + x^4)
```

Control-C interruption works well with the axiom interface, because of the excellent implementation of axiom. For example, try the following sum but with a much bigger range, and hit control-C.

```
sage: axiom('sum(1/x^2, x, 1, 10)')
1968329/1270080
```

0.1 Tutorial

We follow the tutorial at *http://wiki.axiom-developer.org/AxiomTutorial*.

```
sage: axiom('1/100 + 1/101')
201/10100

sage: a = axiom('(1 + sqrt(2))^5'); a
(sqrt(2) + 1)^5
sage: a.expand()
29*sqrt(2) + 41

sage: a = axiom('(1 + sqrt(2))^5')
sage: float(a)
82.012193308819747
sage: a.numer()
82.01219330881975

sage: axiom.eval('fpprec : 100')
'100'
sage: a.bfloat()
8.2012193308819756415248973002081244278520484385931494122123712401731241875401104126661 23

sage: axiom('100!')
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639

sage: f = axiom('(x + 3*y + x^2*y)^3')
sage: f.expand()
x^6*y^3 + 9*x^4*y^3 + 27*x^2*y^3 + 27*y^3 + 3*x^5*y^2 + 18*x^3*y^2 + 27*x*y^2 + 3*x^4*y +
sage: f.subst('x=5/z')
(5/z + 25*y/z^2 + 3*y)^3
sage: g = f.subst('x=5/z')
sage: h = g.ratsimp(); h
(27*y^3*z^6 + 135*y^2*z^5 + (675*y^3 + 225*y)*z^4 + (2250*y^2 + 125)*z^3 + (5625*y^3 + 18
  + 9375*y^2*z + 15625*y^3)/z^6
sage: h.factor()
(3*y*z^2 + 5*z + 25*y)^3/z^6

sage: eqn = axiom(['a+b*c=1', 'b-a*c=0', 'a+b=5'])
sage: s = eqn.solve('[a,b,c]'); s
{\Tt{}a\ =\ (25*sqrt(79)*{\%}i\ +\ 25)/(6*sqrt(79)*{\%}i\ -\ 34),b\ =\ (5*sqrt(79)*{\%}i\
\ \ \ \ \ \ c\ =\ (sqrt(79)*{\%}i\ +\ 1)/10],\nwnewline
```

```
\ \ \ \ \ [a\ =\ (25*sqrt(79)*{\%}i\ -\ 25)/(6*sqrt(79)*{\%}i\ +\ 34),b\ =\ (5*sqrt(79)*{\%}i
\ \ \ \ \ \ c\ =\ \ -\ (sqrt(79)*{\%}i\ -\ 1)/10\nwendquote}
\end{varbatim}
Here is an example of solving an algebraic equation:
\begin{verbatim}
    sage: axiom('x^2+y^2=1').solve('y')
    [y =  - sqrt(1 - x^2),y = sqrt(1 - x^2)]
    sage: axiom('x^2 + y^2 = (x^2 - y^2)/sqrt(x^2 + y^2)').solve('y')
    [y =  - sqrt(( - y^2 - x^2)*sqrt(y^2 + x^2) + x^2),y = sqrt(( - y^2 - x^2)*sqrt(y^2 + x^2
```

You can even nicely typeset the solution in latex:

```
    sage: latex(s)
    \left[ \left[ a=\frac{25 \sqrt{79} i+25}{6 \sqrt{79} i-34} , b=  \frac{5 \sqrt{79} i+5}{\
```

To have the above appear onscreen via xdvi, type view(s).
(TODO: For OS X should create pdf output and use preview instead?)

```
    sage: e = axiom('sin(u + v) * cos(u)^3'); e
    cos(u)^3*sin(v + u)
    sage: f = e.trigexpand(); f
    cos(u)^3*(cos(u)*sin(v) + sin(u)*cos(v))
    sage: f.trigreduce()
    (sin(v + 4*u) + sin(v - 2*u))/8 + (3*sin(v + 2*u) + 3*sin(v))/8
    sage: w = axiom('3 + k*%i')
    sage: f = w^2 + axiom('%e')^w
    sage: f.realpart()
    %e^3*cos(k) - k^2 + 9

    sage: f = axiom('x^3 * %e^(k*x) * sin(w*x)'); f
    x^3*%e^(k*x)*sin(w*x)
    sage: f.diff('x')
    k*x^3*%e^(k*x)*sin(w*x) + 3*x^2*%e^(k*x)*sin(w*x) + w*x^3*%e^(k*x)*cos(w*x)
    sage: f.integrate('x')
    (((k*w^6 + 3*k^3*w^4 + 3*k^5*w^2 + k^7)*x^3 + (3*w^6 + 3*k^2*w^4 - 3*k^4*w^2 - 3*k^6)*x^2

    sage: f = axiom('1/x^2')
    sage: f.integrate('x', 1, 'inf')
    1
    sage: g = axiom('f/sinh(k*x)^4')
    sage: g.taylor('x', 0, 3)
    f/(k^4*x^4) - 2*f/(3*k^2*x^2) + 11*f/45 - 62*k^2*f*x^2/945

    sage: axiom.taylor('asin(x)','x',0, 10)
    x + x^3/6 + 3*x^5/40 + 5*x^7/112 + 35*x^9/1152
```

0.2 Examples involving matrices

We illustrate computing with the matrix whose i,j entry
is i/j, for $i,j=1,...,4$.

```
    sage: f = axiom.eval('f[i,j] := i/j')
```

```
sage: A = axiom('genmatrix(f,4,4)'); A
matrix([1,1/2,1/3,1/4],[2,1,2/3,1/2],[3,3/2,1,3/4],[4,2,4/3,1])
sage: A.determinant()
0
sage: A.echelon()
matrix([1,1/2,1/3,1/4],[0,0,0,0],[0,0,0,0],[0,0,0,0])
sage: A.eigenvalues()
{\Tt{}0,4],[3,1\nwendquote}
sage: A.eigenvectors()
{\Tt{}[0,4],[3,1\nwendquote}],[1,0,0, - 4],[0,1,0, - 2],[0,0,1, - 4/3],[1,2,3,4]]
```

We can also compute the echelon form in :

```
sage: B = matrix(QQ, A)
sage: B.echelon_form()
[  1 1/2 1/3 1/4]
[  0   0   0   0]
[  0   0   0   0]
[  0   0   0   0]
sage: B.charpoly().factor()
(x - 4) * x^3
```

0.3 Laplace Transforms

We illustrate Laplace transforms:

```
sage: _ = axiom.eval("f(t) := t*sin(t)")
sage: axiom("laplace(f(t),t,s)")
2*s/(s^2 + 1)^2

sage: axiom("laplace(delta(t-3),t,s)") #Dirac delta function
%e^-(3*s)

sage: _ = axiom.eval("f(t) == exp(t)*sin(t)")
sage: axiom("laplace(f(t),t,s)")
1/(s^2 - 2*s + 2)

sage: _ = axiom.eval("f(t) := t^5*exp(t)*sin(t)")
sage: axiom("laplace(f(t),t,s)")
360*(2*s - 2)/(s^2 - 2*s + 2)^4 - 480*(2*s - 2)^3/(s^2 - 2*s + 2)^5 + 120*(2*s - 2)^5/(s^
sage: axiom("laplace(f(t),t,s)")
                                        3                 5
          360 (2 s - 2)    480 (2 s - 2)     120 (2 s - 2)
         --------------- - --------------- + ---------------
           2           4     2           5     2           6
         (s  - 2 s + 2)    (s  - 2 s + 2)    (s  - 2 s + 2)

sage: axiom("laplace(diff(x(t),t),t,s)")
s*laplace(x(t),t,s) - x(0)

sage: axiom("laplace(diff(x(t),t,2),t,s)")
```

```
-at('diff(x(t),t,1),t = 0) + s^2*laplace(x(t),t,s) - x(0)*s

sage.: axiom("laplace(diff(x(t),t,2),t,s)")
                        !
            d           !         2
          - -- (x(t))!      + s  laplace(x(t), t, s) - x(0) s
            dt          !
                        !t = 0
```

Even better, use view(axiom("laplace(diff(x(t),t,2),t,s)")) to see a typeset version.

0.4 Continued Fractions

A continued fraction $a + 1/(b + 1/(c + \cdots))$ *is* represented in axiom by the list $[a, b, c, ...]$.

```
sage: axiom("cf((1 + sqrt(5))/2)")
[1,1,1,1,2]
sage: axiom("cf ((1 + sqrt(341))/2)")
[9,1,2,1,2,1,17,1,2,1,2,1,17,1,2,1,2,1,17,2]
```

0.5 Special examples

In this section we illustrate calculations that would be awkward to do (as far as I know) in non-symbolic computer algebra systems like MAGMA or GAP.
We compute the gcd of $2x^n+4 - x^n+2$ and $4x^n+1 + 3x^n$ for arbitrary n.

```
sage: f = axiom('2*x^(n+4) - x^(n+2)')
sage: g = axiom('4*x^(n+1) + 3*x^n')
sage: f.gcd(g)
x^n
```

You can plot 3d graphs (via gnuplot):

```
sage.: axiom('plot3d(x^2-y^2, [x,-2,2], [y,-2,2], [grid,12,12])')
[displays a 3 dimensional graph]
```

You can formally evaluate sums (note the nusum command):

```
sage: S = axiom('nusum(exp(1+2*i/n),i,1,n)')
sage.: S
                          2/n + 3                   2/n + 1
                         %e                        %e
                    ---------------------- - ----------------------
                       1/n          1/n          1/n          1/n
                    (%e    - 1) (%e    + 1)  (%e    - 1) (%e    + 1)
```

We formally compute the limit as $n\to \infty$ *of* $2S/n$ *as follows*:

```
sage: T = S*axiom('2/n')
sage: T.tlimit('n','inf')
%e^3 - %e
```

0.6 Miscellaneous

Obtaining digits of π:

```
sage: axiom.eval('fpprec : 100')
'100'
sage: axiom(pi).bfloat()
3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803
```

Defining functions in axiom:

```
sage: axiom.eval('fun(a) == a^2')
'fun[a] := a^2'
sage: axiom('fun(10)')
100
```

0.7 Interactivity

Axiom has a non-interactive mode that is initiated via the command
")read file.input".

0.8 Latex Output

The latex output of Axiom is not perfect. E.g.,

```
sage: axiom.eval('tex(sin(u) + sinh(v^2))')
'$$\\sinhv^2 + \\sinu$$false'
```

Notice the lack of space after the sin macro, which is a latex syntax error. In this is automatically fixed via a substition for trig functions, which may have potentially bad side effects:

```
sage: latex(axiom('sin(u) + sinh(v^2)'))
\sinh v^2+\sin u
```

It would be nice if somebody would fix this problem. One way would be to improve Axiom by making the fix to Axiom and giving this back to the Axiom people.
Here's another example:

```
sage: g = axiom('exp(3*%i*x)/(6*%i) + exp(%i*x)/(2*%i) + c')
sage: latex(g)
 -\frac{i e^{3 i x}}{6}-\frac{i e^{i x}}{2}+c
```

0.9 Long Input

The Axiom interface reads in even very long input (using files) in a robust manner, as long as you are creating a new object.
Using axiom.eval for long input
is much less robust, and is not recommended.

```
sage: t = '"%s"'%10^10000   # ten thousand character string.
sage: a = axiom(t)
```

A Sage Interface to Axiom

AUTHORS OF THIS MODULE: - William Stein (2006-10): Initial version based

March 21, 2013

⟨*axiom.py*⟩≡

```
r"""
Interface to Axiom

Axiom is a free GPL-compatible (modified BSD license) general purpose
computer algebra system whose development started in 1973 at IBM. It
contains symbolic manipulation algorithms, as well as implementations
of special functions, including elliptic functions and generalized
hypergeometric functions. Moreover, Axiom has implementations of many
functions relating to the invariant theory of the symmetric group $S_n$.
For many links to Axiom documentation see
         \url{http://wiki.axiom-developer.org}.

AUTHORS OF THIS MODULE:
    - William Stein (2006-10): Initial version based on Maxima interface
    - Bill Page (2006-10): Axiom version

If the string "error" (case insensitive) occurs in the output of
anything from axiom, a RuntimeError exception is raised.

EXAMPLES:
We evaluate a very simple expression in axiom.
    sage: axiom('3 * 5')
    15
                                          Type: PositiveInteger

We factor $x^5 - y^5$ in Axiom in several different ways.
The first way yields a Axiom object.
    sage: F = axiom.factor('x^5 - y^5')
    sage: type(F)
    <class 'axiom.interfaces.axiom.AxiomElement'>

Note that Axiom objects are normally displayed using ``ASCII art'';
to see a normal linear representation of any Axiom object x,
use \code{str(x)}.
```

```
    sage: F
                                4      3    2 2    3      4
                  - (y - x) (y  + x y  + x  y  + x  y + x )
```

You can always use \code{x.str()} to obtain the linear representation of an object, even without changing the display2d flag. This can be useful for moving axiom data to other systems.

```
    sage: F.str()
    '-(y - x)*(y^4 + x*y^3 + x^2*y^2 + x^3*y + x^4)'

    sage: axiom.display2d(False)
    sage: F
    -(y - x)*(y^4 + x*y^3 + x^2*y^2 + x^3*y + x^4)
```

The \code{axiom.eval} command evaluates an expression in axiom and returns the result as a string.

```
    sage: print axiom.eval('factor(x^5 - y^5)')
    -(y - x)*(y^4 + x*y^3 + x^2*y^2 + x^3*y + x^4)
```

We can create the polynomial f as a Axiom polynomial, then call the factor method on it. Notice that the notation \code{f.factor()} is consistent with how the rest of \sage works.

```
    sage: f = axiom('x^5 - y^5')
    sage: f^2
    (x^5 - y^5)^2
    sage: f.factor()
    -(y - x)*(y^4 + x*y^3 + x^2*y^2 + x^3*y + x^4)
```

Control-C interruption works well with the axiom interface, because of the excellent implementation of axiom. For example, try the following sum but with a much bigger range, and hit control-C.

```
    sage: axiom('sum(1/x^2, x, 1, 10)')
    1968329/1270080
```

\subsection{Tutorial}

We follow the tutorial at \url{http://wiki.axiom-developer.org/AxiomTutorial}.

```
    sage: axiom('1/100 + 1/101')
    201/10100

    sage: a = axiom('(1 + sqrt(2))^5'); a
    (sqrt(2) + 1)^5
```

```
    sage: a.expand()
    29*sqrt(2) + 41

    sage: a = axiom('(1 + sqrt(2))^5')
    sage: float(a)
    82.012193308819747
    sage: a.numer()
    82.01219330881975

    sage: axiom.eval('fpprec : 100')
    '100'
    sage: a.bfloat()
    8.20121933088197564152489730020812442785204843859314941221237124017312418754
```

```
    sage: axiom('100!')
    933262154439441526816992388562667004907159682643816214685929638952175999932299
```

```
    sage: f = axiom('(x + 3*y + x^2*y)^3')
    sage: f.expand()
    x^6*y^3 + 9*x^4*y^3 + 27*x^2*y^3 + 27*y^3 + 3*x^5*y^2 + 18*x^3*y^2 + 27*x*y^2
    sage: f.subst('x=5/z')
    (5/z + 25*y/z^2 + 3*y)^3
    sage: g = f.subst('x=5/z')
    sage: h = g.ratsimp(); h
    (27*y^3*z^6 + 135*y^2*z^5 + (675*y^3 + 225*y)*z^4 + (2250*y^2 + 125)*z^3 + (56
    sage: h.factor()
    (3*y*z^2 + 5*z + 25*y)^3/z^6
```

```
    sage: eqn = axiom(['a+b*c=1', 'b-a*c=0', 'a+b=5'])
    sage: s = eqn.solve('[a,b,c]'); s
    [[a = (25*sqrt(79)*%i + 25)/(6*sqrt(79)*%i - 34),b = (5*sqrt(79)*%i + 5)/(sqrt
```

Here is an example of solving an algebraic equation:

```
    sage: axiom('x^2+y^2=1').solve('y')
    [y =  - sqrt(1 - x^2),y = sqrt(1 - x^2)]
    sage: axiom('x^2 + y^2 = (x^2 - y^2)/sqrt(x^2 + y^2)').solve('y')
    [y =  - sqrt(( - y^2 - x^2)*sqrt(y^2 + x^2) + x^2),y = sqrt(( - y^2 - x^2)*sqr
```

You can even nicely typeset the solution in latex:

```
    sage: latex(s)
    \left[ \left[ a=\frac{25 \sqrt{79} i+25}{6 \sqrt{79} i-34} , b=  \frac{5 \sqrt
```

To have the above appear onscreen via \code{xdvi}, type \code{view(s)}.
(TODO: For OS X should create pdf output and use preview instead?)

```
    sage: e = axiom('sin(u + v) * cos(u)^3'); e
```

```
    cos(u)^3*sin(v + u)
    sage: f = e.trigexpand(); f
    cos(u)^3*(cos(u)*sin(v) + sin(u)*cos(v))
    sage: f.trigreduce()
    (sin(v + 4*u) + sin(v - 2*u))/8 + (3*sin(v + 2*u) + 3*sin(v))/8
    sage: w = axiom('3 + k*%i')
    sage: f = w^2 + axiom('%e')^w
    sage: f.realpart()
    %e^3*cos(k) - k^2 + 9

    sage: f = axiom('x^3 * %e^(k*x) * sin(w*x)'); f
    x^3*%e^(k*x)*sin(w*x)
    sage: f.diff('x')
    k*x^3*%e^(k*x)*sin(w*x) + 3*x^2*%e^(k*x)*sin(w*x) + w*x^3*%e^(k*x)*cos(w*x)
    sage: f.integrate('x')
    (((k*w^6 + 3*k^3*w^4 + 3*k^5*w^2 + k^7)*x^3 + (3*w^6 + 3*k^2*w^4 - 3*k^4*w^2 -

    sage: f = axiom('1/x^2')
    sage: f.integrate('x', 1, 'inf')
    1
    sage: g = axiom('f/sinh(k*x)^4')
    sage: g.taylor('x', 0, 3)
    f/(k^4*x^4) - 2*f/(3*k^2*x^2) + 11*f/45 - 62*k^2*f*x^2/945

    sage: axiom.taylor('asin(x)','x',0, 10)
    x + x^3/6 + 3*x^5/40 + 5*x^7/112 + 35*x^9/1152
```

\subsection{Examples involving matrices}
We illustrate computing with the matrix whose i,j entry
is i/j, for $i,j=1,\ldots,4$.

```
    sage: f = axiom.eval('f[i,j] := i/j')
    sage: A = axiom('genmatrix(f,4,4)'); A
    matrix([1,1/2,1/3,1/4],[2,1,2/3,1/2],[3,3/2,1,3/4],[4,2,4/3,1])
    sage: A.determinant()
    0
    sage: A.echelon()
    matrix([1,1/2,1/3,1/4],[0,0,0,0],[0,0,0,0],[0,0,0,0])
    sage: A.eigenvalues()
    [[0,4],[3,1]]
    sage: A.eigenvectors()
    [[[0,4],[3,1]],[1,0,0, - 4],[0,1,0, - 2],[0,0,1, - 4/3],[1,2,3,4]]
```

We can also compute the echelon form in \sage:

```
    sage: B = matrix(QQ, A)
    sage: B.echelon_form()
```

```
[  1 1/2 1/3 1/4]
[  0   0   0   0]
[  0   0   0   0]
[  0   0   0   0]
sage: B.charpoly().factor()
(x - 4) * x^3
```

\subsection{Laplace Transforms}
We illustrate Laplace transforms:

```
sage: _ = axiom.eval("f(t) := t*sin(t)")
sage: axiom("laplace(f(t),t,s)")
2*s/(s^2 + 1)^2

sage: axiom("laplace(delta(t-3),t,s)") #Dirac delta function
%e^-(3*s)

sage: _ = axiom.eval("f(t) == exp(t)*sin(t)")
sage: axiom("laplace(f(t),t,s)")
1/(s^2 - 2*s + 2)

sage: _ = axiom.eval("f(t) := t^5*exp(t)*sin(t)")
sage: axiom("laplace(f(t),t,s)")
360*(2*s - 2)/(s^2 - 2*s + 2)^4 - 480*(2*s - 2)^3/(s^2 - 2*s + 2)^5 + 120*(2*s
sage: axiom("laplace(f(t),t,s)")
                                          3                 5
           360 (2 s - 2)    480 (2 s - 2)     120 (2 s - 2)
          --------------- - --------------- + ---------------
            2           4     2           5     2           6
          (s  - 2 s + 2)    (s  - 2 s + 2)    (s  - 2 s + 2)

sage: axiom("laplace(diff(x(t),t),t,s)")
s*laplace(x(t),t,s) - x(0)

sage: axiom("laplace(diff(x(t),t,2),t,s)")
-at('diff(x(t),t,1),t = 0) + s^2*laplace(x(t),t,s) - x(0)*s

sage.: axiom("laplace(diff(x(t),t,2),t,s)")
                      !
            d         !         2
          - -- (x(t))!      + s  laplace(x(t), t, s) - x(0) s
            dt        !
                      !t = 0
```

Even better, use \code{view(axiom("laplace(diff(x(t),t,2),t,s)"))} to see
a typeset version.

```
\subsection{Continued Fractions}

A continued fraction $a + 1/(b + 1/(c + \cdots))$ is
represented in axiom by the list $[a, b, c, \ldots]$.

    sage: axiom("cf((1 + sqrt(5))/2)")
    [1,1,1,1,2]
    sage: axiom("cf ((1 + sqrt(341))/2)")
    [9,1,2,1,2,1,17,1,2,1,2,1,17,1,2,1,2,1,17,2]

\subsection{Special examples}

In this section we illustrate calculations that would be awkward to do
(as far as I know) in non-symbolic computer algebra systems like MAGMA
or GAP.

We compute the gcd of $2x^{n+4} - x^{n+2}$ and $4x^{n+1} + 3x^n$
for arbitrary $n$.

    sage: f = axiom('2*x^(n+4) - x^(n+2)')
    sage: g = axiom('4*x^(n+1) + 3*x^n')
    sage: f.gcd(g)
    x^n

You can plot 3d graphs (via gnuplot):

    sage.: axiom('plot3d(x^2-y^2, [x,-2,2], [y,-2,2], [grid,12,12])')
    [displays a 3 dimensional graph]

You can formally evaluate sums (note the \code{nusum} command):

    sage: S = axiom('nusum(exp(1+2*i/n),i,1,n)')
    sage.: S
                              2/n + 3                    2/n + 1
                             %e                         %e
                    ----------------------- - -----------------------
                       1/n          1/n           1/n          1/n
                    (%e    - 1) (%e    + 1)   (%e    - 1) (%e    + 1)

We formally compute the limit as $n\to\infty$ of $2S/n$ as follows:

    sage: T = S*axiom('2/n')
    sage: T.tlimit('n','inf')
    %e^3 - %e

\subsection{Miscellaneous}
```

```
Obtaining digits of $\pi$:
    sage: axiom.eval('fpprec : 100')
    '100'
    sage: axiom(pi).bfloat()
    3.14159265358979323846264338327950288419716939937510582097494459230781640628620

Defining functions in axiom:
    sage: axiom.eval('fun(a) == a^2')
    'fun[a] := a^2'
    sage: axiom('fun(10)')
    100

\subsection{Interactivity}
Axiom has a non-interactive mode that is initiated via the command
")read file.input".

\subsection{Latex Output}
The latex output of Axiom is not perfect.  E.g.,

    sage: axiom.eval('tex(sin(u) + sinh(v^2))')
    '$$\\sinhv^2 + \\sinu$$false'

Notice the lack of space after the sin macro, which is a latex syntax
error.  In \sage this is automatically fixed via a substition for
trig functions, which may have potentially bad side effects:

    sage: latex(axiom('sin(u) + sinh(v^2)'))
    \sinh v^2+\sin u

It would be nice if somebody would fix this problem.  One way would
be to improve Axiom by making the fix to Axiom and giving this back
to the Axiom people.

Here's another example:

    sage: g = axiom('exp(3*%i*x)/(6*%i) + exp(%i*x)/(2*%i) + c')
    sage: latex(g)
     -\frac{i e^{3 i x}}{6}-\frac{i e^{i x}}{2}+c

\subsection{Long Input}
The Axiom interface reads in even very long input (using files) in a
robust manner, as long as you are creating a new object.
\note{Using \code{axiom.eval} for long input
is much less robust, and is not recommended.}

    sage: t = '"%s"'%10^10000   # ten thousand character string.
```

```
    sage: a = axiom(t)
"""

#*****************************************************************************
#       Copyright (C) 2005 William Stein <wstein@gmail.com>
#
#  Distributed under the terms of the GNU General Public License (GPL)
#
#    This code is distributed in the hope that it will be useful,
#    but WITHOUT ANY WARRANTY; without even the implied warranty of
#    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
#    General Public License for more details.
#
#  The full text of the GPL is available at:
#
#                  http://www.gnu.org/licenses/
#*****************************************************************************

import os, re

from expect import Expect, ExpectElement, FunctionElement, ExpectFunction, tmp

from sage.misc.misc import verbose

from sage.misc.multireplace import multiple_replace

cnt = 0
seq = 0

from sage.misc.all import pager, verbose, DOT_SAGE, SAGE_ROOT

COMMANDS_CACHE = '%s/axiom_commandlist_cache.sobj'%DOT_SAGE

import sage.server.support

# The Axiom commands ")what thing det" ")show Matrix" and ")display
# op det" commands, gives a list of all identifiers that begin in
# a certain way.  This could maybe be useful somehow... (?)  Also
# axiom has a lot a lot of ways for getting documentation from the
# system -- this could also be useful.

class Axiom(Expect):
    """
    Interface to the Axiom interpreter.
    """
    def __call__(self, x):
```

```
        import sage.rings.all
        if sage.rings.all.is_Infinity(x):
            return Expect.__call__(self, 'inf')
        else:
            return Expect.__call__(self, x)

    def __init__(self, script_subdirectory=None, logfile=None, server=None):
        """
        Create an instance of the Axiom interpreter.
        """
        eval_using_file_cutoff = 200
        self.__eval_using_file_cutoff = eval_using_file_cutoff
        Expect.__init__(self,
                        name = 'axiom',
                        prompt = '\([0-9]+\) -> ',
                        command = "axiom -nox",
                        maxread = 1,  # CRUCIAL to use less buffering for Axiom!
                        script_subdirectory = script_subdirectory,
                        restart_on_ctrlc = False,
                        verbose_start = False,
                        init_code = [],
                        logfile = logfile,
                        eval_using_file_cutoff=eval_using_file_cutoff)

    def __getattr__(self, attrname):
        if attrname[:1] == "_":
            raise AttributeError
        return AxiomExpectFunction(self, attrname)

    def _start(self):
        # For some reason sending a single input line at startup avoids
        # lots of weird timing issues when doing doctests.
        Expect._start(self)
        #out = self._eval_line(')set output algebra off', reformat=False)
        #out = self._eval_line(')set output tex on', reformat="False")
        out = self._eval_line(')set message autoload off', reformat=False)
        self._expect.expect(self._prompt)
        #out = self._expect.before
        #print "out 0 = '%s'"%out
        #self(1)

    def _eval_line_using_file(self, line, tmp):
        F = open(tmp, 'w')
        F.write(line)
        F.close()
        if self._expect is None:
```

```
            self._start()
        # For some reason this trivial comp
        # keeps certain random freezes from occuring.  Do not remove this.
        # The space before the \n is also important.
        self._expect.sendline('')read "%s"\n'%tmp)
        self._expect.expect(self._prompt)
        return ''

    def __reduce__(self):
        return reduce_load_Axiom, tuple([])

    def _quit_string(self):
        return ')lisp (quit)'

    def _eval_line(self, line, reformat=True, allow_use_file=False,
                   wait_for_prompt=True):
        if not wait_for_prompt:
            return Expect._eval_line(self, line)
        line = line.rstrip().rstrip(';')
        if line == '':
            return ''
        global seq
        seq += 1
        #start = SAGE_START + str(seq)
        #end = SAGE_END + str(seq)
        #line = '%s;\n%s; %s;'%(start, line, end)
        if self._expect is None:
            self._start()
        if allow_use_file and self.__eval_using_file_cutoff and \
                           len(line) > self.__eval_using_file_cutoff:
            return self._eval_line_using_file(line, tmp)
        try:
            E = self._expect
            # debug
            # print "in = '%s'"%line
            E.sendline(line)
            self._expect.expect(self._prompt)
            out = self._expect.before
            # debug
            # print "out = '%s'"%out

        except KeyboardInterrupt:
            self._keyboard_interrupt()

        if 'Incorrect syntax:' in out:
            raise RuntimeError, out
```

```
    if not reformat:
        return out
    if 'error' in out:
        return out
    #out = out.lstrip()
    i = out.find('\n')
    out = out[i+1:]
    outs = out.split("\n")
    i = 0
    outline = ''
    for line in outs:
        line = line.rstrip()
        # print "'%s'"%line
        if line[:4] == '   (':
            i = line.find('(')
            i += line[i:].find(')')
            if line[i+1:] == "":
                i = 0
                outs = outs[1:]
            break;
    out = "\n".join(line[i+1:] for line in outs[1:])
    return out

##########################################
# Interactive help
##########################################

def help(self, s):
    if sage.server.support.EMBEDDED_MODE:
        os.system('asq "describe(%s); "< /dev/null'%s)
    else:
        os.system('asq "describe(%s);"'%s)

def example(self, s):
    if sage.server.support.EMBEDDED_MODE:
        os.system('axiom -ht -nogr "example(%s);" < /dev/null'%s)
    else:
        os.system('axiom -ht "example(%s);"'%s)

describe = help

def demo(self, s):
    if sage.server.support.EMBEDDED_MODE:
        os.system('axiom -ht "demo(%s);" < /dev/null'%s)
    else:
```

```
        os.system('axiom -ht "demo(%s);"'%s)

def completions(self, s):
    """
    Return all commands that complete the command starting with the
    string s.   This is like typing s[tab] in the maple interpreter.
    """
    s = self.eval('apropos(%s)'%s).replace('\\ - ','-')
    return [x for x in s[1:-1].split(',') if x[0] != '?']

def _commands(self):
    """
    Return list of all commands defined in Axiom.
    """
    try:
        return self.__commands
    except AttributeError:
        self.__commands = sum([self.completions(chr(97+n)) for n in range(26)]
    return self.__commands

def trait_names(self, verbose=True, use_disk_cache=True):
    try:
        return self.__trait_names
    except AttributeError:
        import sage.misc.persist
        if use_disk_cache:
            try:
                self.__trait_names = sage.misc.persist.load(COMMANDS_CACHE)
                return self.__trait_names
            except IOError:
                pass
        if verbose:
            print "\nBuilding Axiom command completion list (this takes"
            print "a few seconds only the first time you do it)."
            print "To force rebuild later, delete %s."%COMMANDS_CACHE
        v = self._commands()
        self.__trait_names = v
        sage.misc.persist.save(v, COMMANDS_CACHE)
        return v

def _object_class(self):
    return AxiomElement

def _true_symbol(self):
    return 'true'
```

```
def _false_symbol(self):
    return 'false'

def function(self, args, defn, repr=None, latex=None):
    """
    Return the Axiom function with given arguments
    and definition.

    INPUT:
        args -- a string with variable names separated by commas
        defn -- a string (or Axiom expression) that defines
                a function of the arguments in Axiom.
        repr -- an optional string; if given, this is how the function will pr

    EXAMPLES:
        sage: f = axiom.function('x', 'sin(x)')
        sage: f(3.2)
        -0.058374143427580086
        sage: f = axiom.function('x,y', 'sin(x)+cos(y)')
        sage: f(2,3.5)
        sin(2) - 0.9364566872907963
        sage: f
        sin(x)+cos(y)

        sage: g = f.integrate('z'); g
        (cos(y) + sin(x))*z
        sage: g(1,2,3)
        3*(cos(2) + sin(1))

    The function definition can be an axiom object:
        sage: an_expr = axiom('sin(x)*gamma(x)')
        sage: t = axiom.function('x', an_expr)
        sage: t
        gamma(x)*sin(x)
        sage: t(2)
        sin(2)
        sage: float(t(2))
        0.90929742682568171
        sage: loads(t.dumps())
        gamma(x)*sin(x)
    """
    name = self._next_var_name()
    defn = str(defn)
    args = str(args)
    axiom.eval('%s(%s) := %s'%(name, args, defn))
    if repr is None:
```

```
        repr = defn
    f = AxiomFunction(self, name, repr, args, latex)
    return f

def set(self, var, value):
    """
    Set the variable var to the given value.
    """
    cmd = '%s := %s'%(var, value)
    out = self._eval_line(cmd, reformat=False)
    #out = self._eval_line(cmd, reformat=False, allow_use_file=True)

    if out.find("error") != -1:
        raise TypeError, "Error executing code in Axiom\nCODE:\n\t%s\nAxiom ER


def get(self, var):
    """
    Get the string value of the variable var.
    """
    s = self._eval_line('%s'%var)
    return s

#def clear(self, var):
#    """
#    Clear the variable named var.
#    """
#    if self._expect is None:
#        return
#    self._expect.sendline('kill(%s);'%var)
#    self._expect.expect(self._prompt)

def console(self):
    axiom_console()

def plot2d(self, *args):
    r"""
    Plot a 2d graph using Axiom / Viewman.

    axiom.plot2d(f, '[var, min, max]', options)

    INPUT:
        f -- a string representing a function (such as f="sin(x)")
        [var, xmin, xmax]
        options -- an optional string representing plot2d options in gnuplot f
```

```
    EXAMPLES:
        sage.: axiom.plot2d('sin(x)','[x,-5,5]')
        sage.: opts = '[gnuplot_term, ps], [gnuplot_out_file, "sin-plot.eps"]
        sage.: axiom.plot2d('sin(x)','[x,-5,5]',opts)

    The eps file is saved in the current directory.
    """
    self('plot2d(%s)'%(','.join([str(x) for x in args])))

def plot2d_parametric(self, r, var, trange, nticks=50, options=None):
    r"""
    Plots r = [x(t), y(t)] for t = tmin...tmax using gnuplot with options

    INPUT:
        r -- a string representing a function (such as r="[x(t),y(t)]")
        var -- a string representing the variable (such as var = "t")
        trange -- [tmin, tmax] are numbers with tmin<tmax
        nticks -- int (default: 50)
        options -- an optional string representing plot2d options in gnuplot f

    EXAMPLES:
        sage.: axiom.plot2d_parametric(["sin(t)","cos(t)"], "t",[-3.1,3.1])

        sage.: opts = '[gnuplot_preamble, "set nokey"], [gnuplot_term, ps], [g
        sage.: axiom.plot2d_parametric(["sin(t)","cos(t)"], "t", [-3.1,3.1], o

    The eps file is saved to the current working directory.

    Here is another fun plot:
        sage.: axiom.plot2d_parametric(["sin(5*t)","cos(11*t)"], "t", [0,2*pi
    """
    tmin = trange[0]
    tmax = trange[1]
    cmd = "plot2d([parametric, %s, %s, [%s, %s, %s], [nticks, %s]]"%( \
              r[0], r[1], var, tmin, tmax, nticks)
    if options is None:
        cmd += ")"
    else:
        cmd += ", %s)"%options
    self(cmd)

def plot3d(self, *args):
    r"""
    Plot a 3d graph using Axiom / Viewman.

    axiom.plot3d(f, '[x, xmin, xmax]', '[y, ymin, ymax]', '[grid, nx, ny]', op
```

```
        INPUT:
            f -- a string representing a function (such as f="sin(x)")
            [var, min, max]

        EXAMPLES:
            sage.: axiom.plot3d('1 + x^3 - y^2', '[x,-2,2]', '[y,-2,2]', '[grid,12
            sage.: axiom.plot3d('sin(x)*cos(y)', '[x,-2,2]', '[y,-2,2]', '[grid,30
            sage.: opts = '[gnuplot_term, ps], [gnuplot_out_file, "sin-plot.eps"]
            sage.: axiom.plot3d('sin(x+y)', '[x,-5,5]', '[y,-1,1]', opts)

        The eps file is saved in the current working directory.
        """
        self('plot3d(%s)'%(','.join([str(x) for x in args])))

    def plot3d_parametric(self, r, vars, urange, vrange, options=None):
        r"""
        Plot a 3d parametric graph with r=(x,y,z), x = x(u,v), y = y(u,v), z = z(u
        for u = umin...umax, v = vmin...vmax using gnuplot with options.

        INPUT:
            x, y, z -- a string representing a function (such as x="u^2+v^2", ...)
            vars is a list or two strings representing variables (such as vars = [
            urange -- [umin, umax]
            vrange -- [vmin, vmax] are lists of numbers with
            umin < umax, vmin < vmax
            options -- optional string representing plot2d options in gnuplot form

        OUTPUT:
            displays a plot on screen or saves to a file

        EXAMPLES:
            sage.: axiom.plot3d_parametric(["v*sin(u)","v*cos(u)","v"], ["u","v"],
            sage.: opts = '[gnuplot_term, ps], [gnuplot_out_file, "sin-cos-plot.ep
            sage.: axiom.plot3d_parametric(["v*sin(u)","v*cos(u)","v"], ["u","v"],

        The eps file is saved in the current working directory.

        Here is a torus:

            sage.: _ = axiom.eval("expr_1: cos(y)*(10.0+6*cos(x)); expr_2: sin(y)*
            sage.: axiom.plot3d_parametric(["expr_1","expr_2","expr_3"], ["x","y"]

        Here is a Mobius strip:
            sage.: x = "cos(u)*(3 + v*cos(u/2))"
            sage.: y = "sin(u)*(3 + v*cos(u/2))"
```

```
            sage.: z = "v*sin(u/2)"
            sage.: axiom.plot3d_parametric([x,y,z],["u","v"],[-3.1,3.2],[-1/10,1/1
        """
        umin = urange[0]
        umax = urange[1]
        vmin = vrange[0]
        vmax = vrange[1]
        cmd = 'plot3d([%s, %s, %s], [%s, %s, %s], [%s, %s, %s]'%(
            r[0], r[1], r[2], vars[0], umin, umax, vars[1], vmin, vmax)
        if options is None:
            cmd += ')'
        else:
            cmd += ', %s)'%options
        axiom(cmd)

    def de_solve(axiom, de, vars, ics=None):
        """
        Solves a 1st or 2nd order ordinary differential equation (ODE)
        in two variables, possibly with initial conditions.

        INPUT:
            de -- a string representing the ODE
            vars -- a list of strings representing the two variables.
            ics -- a triple of numbers [a,b1,b2] representing
                   y(a)=b1, y'(a)=b2

        EXAMPLES:
            sage.: axiom.de_solve('diff(y,x,2) + 3*x = y', ['x','y'], [1,1,1])
            y = 3*x - 2*%e^(x - 1)
            sage.: axiom.de_solve('diff(y,x,2) + 3*x = y', ['x','y'])
            y = %k1*%e^x + %k2*%e^ - x + 3*x
            sage.: axiom.de_solve('diff(y,x) + 3*x = y', ['x','y'])
            y = (%c - 3*( - x - 1)*%e^ - x)*%e^x
            sage.: axiom.de_solve('diff(y,x) + 3*x = y', ['x','y'],[1,1])
            y =  - %e^ - 1*(5*%e^x - 3*%e*x - 3*%e)
        """
        if not isinstance(vars, str):
            str_vars = '%s, %s'%(vars[1], vars[0])
        else:
            str_vars = vars
        axiom.eval('depends(%s)'%str_vars)
        m = axiom(de)
        a = 'ode2(%s, %s)'%(m.name(), str_vars)
        if ics != None:
            if len(ics) == 3:
                cmd = "ic2("+a+",%s=%s,%s=%s,diff(%s,%s)=%s);"%(vars[0],ics[0], va
```

```
            return axiom(cmd)
        if len(ics) == 2:
            return axiom("ic1("+a+",%s=%s,%s=%s);"%(vars[0],ics[0], vars[1],ic
    return axiom(a+";")

def de_solve_laplace(self, de, vars, ics=None):
    """
    Solves an ordinary differential equation (ODE) using Laplace transforms.

    INPUT:
        de -- a string representing the ODE
              (e.g., de = "diff(f(x),x,2)=diff(f(x),x)+sin(x)")
        vars -- a list of strings representing the variables
              (e.g., vars = ["x","f"])
        ics -- a list of numbers representing initial conditions,
               with symbols allowed which are represented by strings
               (eg, f(0)=1, f'(0)=2 is ics = [0,1,2])

    EXAMPLES:
        sage.: axiom.clear('x'); axiom.clear('f')
        sage.: axiom.de_solve_laplace("diff(f(x),x,2) = 2*diff(f(x),x)-f(x)",
        f(x) = x*%e^x + %e^x

        sage.: axiom.clear('x'); axiom.clear('f')
        sage.: f = axiom.de_solve_laplace("diff(f(x),x,2) = 2*diff(f(x),x)-f(x
        sage.: f
                                              !
                                    x  d      !                   x          x
                    f(x) = x %e  (-- (f(x))!     ) - f(0) x %e  + f(0) %e
                                      dx      !
                                              !x = 0


        \note{The second equation sets the values of $f(0)$ and
        $f'(0)$ in axiom, so subsequent ODEs involving these
        variables will have these initial conditions automatically
        imposed.}
        """
        if not (ics is None):
            d = len(ics)
            for i in range(0,d-1):
                ic = 'atvalue(diff(%s(%s), %s, %s), %s = %s, %s)'%(
                    vars[1], vars[0], vars[0], i, vars[0], ics[0], ics[1+i])
                axiom.eval(ic)
        return axiom('desolve(%s, %s(%s))'%(de, vars[1], vars[0]))
```

```
def solve_linear(self, eqns,vars):
    """
    Wraps axiom's linsolve.

    INPUT:
    eqns is a list of m strings, each rperesenting a linear question
    in m <= n variables
    vars is a list of n strings, each representing a variable

    EXAMPLES:
        sage: eqns = ["x + z = y","2*a*x - y = 2*a^2","y - 2*z = 2"]
        sage: vars = ["x","y","z"]
        sage: axiom.solve_linear(eqns, vars)
        [x = a + 1,y = 2*a,z = a - 1]
    """
    eqs = "["
    for i in range(len(eqns)):
        if i<len(eqns)-1:
            eqs = eqs + eqns[i]+","
        if  i==len(eqns)-1:
            eqs = eqs + eqns[i]+"]"
    vrs = "["
    for i in range(len(vars)):
        if i<len(vars)-1:
            vrs = vrs + vars[i]+","
        if  i==len(vars)-1:
            vrs = vrs + vars[i]+"]"
    return self('linsolve(%s, %s)'%(eqs, vrs))

def unit_quadratic_integer(self, n):
    r"""
    Finds a unit of the ring of integers of the quadratic number
    field $\Q(\sqrt{n})$, $n>1$, using the qunit axiom command.

    EXAMPLE:
        sage: u = axiom.unit_quadratic_integer(101)
        sage: u.parent()
        Number Field in a with defining polynomial x^2 - 101
        sage: u
        a + 10
        sage: u = axiom.unit_quadratic_integer(13)
        sage: u
        5*a + 18
        sage: u.parent()
        Number Field in a with defining polynomial x^2 - 13
    """
```

```
        from sage.rings.all import QuadraticField, Integer
        # Take square-free part so sqrt(n) doesn't get simplified further by axiom
        # (The original version of this function would yield wrong answers if
        # n is not squarefree.)
        n = Integer(n).square_free_part()
        if n < 1:
            raise ValueError, "n (=%s) must be >= 1"%n
        s = str(self('qunit(%s)'%n)).lower()
        r = re.compile('sqrt\(.*\)')
        s = r.sub('a', s)
        a = QuadraticField(n, 'a').gen()
        return eval(s)

    def plot_list(self, ptsx, ptsy, options=None):
        r"""
        Plots a curve determined by a sequence of points.

        INPUT:
            ptsx -- [x1,...,xn], where the xi and yi are real,
            ptsy -- [y1,...,yn]
            options -- a string representing axiom plot2d options.

        The points are (x1,y1), (x2,y2), etc.

        EXAMPLES:
            sage.: zeta_ptsx = [ (pari(1/2 + i*I/10).zeta().real()).precision(1) f
            sage.: zeta_ptsy = [ (pari(1/2 + i*I/10).zeta().imag()).precision(1) f
            sage.: axiom.plot_list(zeta_ptsx, zeta_ptsy)
            sage.: opts='[gnuplot_preamble, "set nokey"], [gnuplot_term, ps], [gnu
            sage.: axiom.plot_list(zeta_ptsx, zeta_ptsy, opts)
        """
        cmd = 'plot2d([discrete,%s, %s]'%(ptsx, ptsy)
        if options is None:
            cmd += ')'
        else:
            cmd += ', %s)'%options
        self(cmd)

    def plot_multilist(self, pts_list, options=None):
        r"""
        Plots a list of list of points pts_list=[pts1,pts2,...,ptsn],
        where each ptsi is of the form [[x1,y1],...,[xn,yn]]
        x's must be integers and y's reals
        options is a string representing axiom plot2d options.

        EXAMPLES:
```

```
            sage.: xx = [ i/10.0 for i in range (-10,10)]
            sage.: yy = [ i/10.0 for i in range (-10,10)]
            sage.: x0 = [ 0 for i in range (-10,10)]
            sage.: y0 = [ 0 for i in range (-10,10)]
            sage.: zeta_ptsx1 = [ (pari(1/2+i*I/10).zeta().real()).precision(1) fo
            sage.: zeta_ptsy1 = [ (pari(1/2+i*I/10).zeta().imag()).precision(1) fo
            sage.: axiom.plot_multilist([[zeta_ptsx1,zeta_ptsy1],[xx,y0],[x0,yy]])
            sage.: zeta_ptsx1 = [ (pari(1/2+i*I/10).zeta().real()).precision(1) fo
            sage.: zeta_ptsy1 = [ (pari(1/2+i*I/10).zeta().imag()).precision(1) fo
            sage.: axiom.plot_multilist([[zeta_ptsx1,zeta_ptsy1],[xx,y0],[x0,yy]])
            sage.: opts='[gnuplot_preamble, "set nokey"]'
            sage.: axiom.plot_multilist([[zeta_ptsx1,zeta_ptsy1],[xx,y0],[x0,yy]],
        """
        n = len(pts_list)
        cmd = '['
        for i in range(n):
            if i < n-1:
                cmd = cmd+'[discrete,'+str(pts_list[i][0])+','+str(pts_list[i][1])
            if i==n-1:
                cmd = cmd+'[discrete,'+str(pts_list[i][0])+','+str(pts_list[i][1])
        # debug
        # print cmd
        if options is None:
            self('plot2d('+cmd+')')
        else:
            self('plot2d('+cmd+','+options+')')


class AxiomElement(ExpectElement):
    def __call__(self, x):
        self._check_valid()
        P = self.parent()
        return P('%s[%s]'%(self.name(), x))

    def _cmp_(self, other):
        """
        EXAMPLES:
            sage: a = axiom(1); b = axiom(2)
            sage: a == b
            False
            sage: a < b
            True
            sage: a > b
            False
            sage: b < a
            False
```

```
        sage: b > a
        True

    We can also compare more complicated object such as functions:
        sage: f = axiom('sin(x)'); g = axiom('cos(x)')
        sage: -f == g.diff('x')
        True
    """

    # thanks to David Joyner for telling me about using "is".
    P = self.parent()
    if P.eval("is (%s < %s)"%(self.name(), other.name())) == P._true_symbol()
        return -1
    elif P.eval("is (%s > %s)"%(self.name(), other.name())) == P._true_symbol
        return 1
    elif P.eval("is (%s = %s)"%(self.name(), other.name())) == P._true_symbol
        return 0
    else:
        return -1  # everything is supposed to be comparable in Python, so we
                   # the comparison thus when no comparable in interfaced syst
def numer(self):
    P = self.parent()
    return P('numeric(%s)'%self._name)

def real(self):
    return self.realpart()

def imag(self):
    return self.imagpart()

def str(self):
    self._check_valid()
    P = self.parent()
    return P.get('%s::InputForm'%self._name)

def __repr__(self):
    self._check_valid()
    P = self.parent()
    return P.get(self._name)

def diff(self, var='x', n=1):
    """
    Return the n-th derivative of self.

    INPUT:
        var -- variable (default: 'x')
```

```
            n -- integer (default: 1)

        OUTPUT:
            n-th derivative of self with respect to the variable var

        EXAMPLES:
            sage: f = axiom('x^2')
            sage: f.diff()
            2*x
            sage: f.diff('x')
            2*x
            sage: f.diff('x', 2)
            2
            sage: axiom('sin(x^2)').diff('x',4)
            16*x^4*sin(x^2) - 12*sin(x^2) - 48*x^2*cos(x^2)

            sage: f = axiom('x^2 + 17*y^2')
            sage: f.diff('x')
            2*x
            sage: f.diff('y')
            34*y
        """
        return ExpectElement.__getattr__(self, 'differentiate')(var, n)

    derivative = diff

    def nintegral(self, var='x', a=0, b=1,
                  desired_relative_error='1e-8',
                  maximum_num_subintervals=200):
        r"""
        Return a numerical approximation to the integral
        of self from a to b.

        INPUT:
            var -- variable to integrate with respect to
            a -- lower endpoint of integration
            b -- upper endpoint of integration
            desired_relative_error -- (default: '1e-8') the desired
                 relative error
            maximum_num_subintervals -- (default: 200) axiom number
                 of subintervals

        OUTPUT:
            -- approximation to the integral
            -- estimated absolute error of the approximation
            -- the number of integrand evaluations
```

```
        -- an error code:
              0 -- no problems were encountered
              1 -- too many subintervals were done
              2 -- excessive roundoff error
              3 -- extremely bad integrand behavior
              4 -- failed to converge
              5 -- integral is probably divergent or slowly convergent
              6 -- the input is invalid

    EXAMPLES:
        sage: axiom('exp(-sqrt(x))').nintegral('x',0,1)
        (0.5284822353142306, 4.1633141378838445E-11, 231, 0)

    Note that GP also does numerical integration, and can do
    so to very high precision very quickly:
        sage: gp('intnum(x=0,1,exp(-sqrt(x)))')
        0.52848223531423071361790491935415653021   # 32-bit
        0.52848223531423071361790491935415653021   # 64-bit
        sage: _ = gp.set_precision(80)
        sage: gp('intnum(x=0,1,exp(-sqrt(x)))')
        0.52848223531423071361790491935415653021675547587292866196865279321015
    """
    from sage.rings.all import Integer
    v = self.quad_qags(var, a, b, desired_relative_error,
                       maximum_num_subintervals)
    return v[0], v[1], Integer(v[2]), Integer(v[3])

def integral(self, var='x', min=None, max=None):
    r"""
    Return the integral of self with respect to the variable x.

    INPUT:
        var -- variable
        min -- default: None
        max -- default: None

    Returns the definite integral if xmin is not None,
    otherwise returns an indefinite integral.

    EXAMPLES:
        sage: axiom('x^2+1').integral()
        x^3/3 + x
        sage: axiom('x^2+ 1 + y^2').integral('y')
        y^3/3 + x^2*y + y
        sage: axiom('x / (x^2+1)').integral()
        log(x^2 + 1)/2
```

```
            sage: axiom('1/(x^2+1)').integral()
            atan(x)
            sage.: axiom('1/(x^2+1)').integral('x', 0, infinity)
            %pi/2
            sage: axiom('x/(x^2+1)').integral('x', -1, 1)
            0

            sage: f = axiom('exp(x^2)').integral('x',0,1); f
            -sqrt(%pi)*%i*erf(%i)/2
            sage: f.numer()         # I wonder how to get a real number (~1.463)??
            -0.8862269254527579*%i*erf(%i)
        """
        I = ExpectElement.__getattr__(self, 'integrate')
        if min is None:
            return I(var)
        else:
            if max is None:
                raise ValueError, "neither or both of min/max must be specified."
            return I(var, min, max)

    integrate = integral

    def __float__(self):
        return float(str(self.numer()))

    def __len__(self):
        """
        Return the length of a list.

        EXAMPLES:
            sage: v = axiom('[x^i for i in 0..5]')
            sage: len(v)
            6
        """
        self._check_valid()
        return int(self.parent().eval('#(%s)'%self.name()))

    def __getattr__(self, attrname):
        if attrname[:1] == "_":
            raise AttributeError
        return AxiomFunctionElement(self, attrname)

    def __getitem__(self, n):
        r"""
        Return the n-th element of this list.
```

```
        \note{Lists are 0-based when accessed via the \sage interface,
        not 1-based as they are in the Axiom interpreter.}

        EXAMPLES:
            sage: v = axiom('[i*x^i for i in 0..5]'); v
            [0,x,2*x^2,3*x^3,4*x^4,5*x^5]
            sage: v[3]
            3*x^3
            sage: v[0]
            0
            sage: v[10]
            Traceback (most recent call last):
            ...
            IndexError: n = (10) must be between 0 and 5
        """
        n = int(n)
        if n < 0 or n >= len(self):
            raise IndexError, "n = (%s) must be between %s and %s"%(n, 0, len(self
        return ExpectElement.__getitem__(self, n+1)

    def subst(self, val):
        P = self.parent()
        return P('subst(%s, %s)'%(self.name(), val))

    def comma(self, args):
        self._check_valid()
        P = self.parent()
        return P('%s, %s'%(self.name(), args))

    def _latex_(self):
        self._check_valid()
        P = self.parent()
        s = axiom._eval_line('outputAsTex(%s)'%self.name(), reformat=False)
        if not '$$' in s:
            raise RuntimeError, "Error texing axiom object."
        i = s.find('$$')
        j = s.rfind('$$')
        s = s[i+2:j]
        s = multiple_replace({'\r\n':' ',
                              '\\%':'',
                              '\\arcsin ':'\\sin^{-1} ',
                              '\\arccos ':'\\cos^{-1} ',
                              '\\arctan ':'\\tan^{-1} '}, s)
        return s

    def trait_names(self):
```

```
        return self.parent().trait_names()

    def _matrix_(self, R):
        r"""
        If self is a Axiom matrix, return the corresponding \sage
        matrix over the \sage ring $R$.

        This may or may not work depending in how complicated the
        entries of self are!  It only works if the entries of self
        can be coerced as strings to produce meaningful elements
        of $R$.

        EXAMPLES:
            sage: _ = axiom.eval("f[i,j] := i/j")
            sage: A = axiom('genmatrix(f,4,4)'); A
            matrix([1,1/2,1/3,1/4],[2,1,2/3,1/2],[3,3/2,1,3/4],[4,2,4/3,1])
            sage: A._matrix_(QQ)
            [  1 1/2 1/3 1/4]
            [  2   1 2/3 1/2]
            [  3 3/2   1 3/4]
            [  4   2 4/3   1]

        You can also use the \code{matrix} command (which is defined
        in \code{sage.misc.functional}):
            sage: matrix(QQ, A)
            [  1 1/2 1/3 1/4]
            [  2   1 2/3 1/2]
            [  3 3/2   1 3/4]
            [  4   2 4/3   1]
        """
        from sage.matrix.all import MatrixSpace
        self._check_valid()
        P = self.parent()
        nrows = int(P.eval('length(%s)'%self.name()))
        if nrows == 0:
            return MatrixSpace(R, 0, 0)(0)
        ncols = int(P.eval('length(%s[1])'%self.name()))
        M = MatrixSpace(R, nrows, ncols)
        s = self.str().replace('matrix','').replace(',',"',").\
            replace("]','[","',").replace('([',"['").replace('])',"']")
        s = eval(s)
        return M([R(x) for x in s])

    def partial_fraction_decomposition(self, var='x'):
        """
        Return the partial fraction decomposition of self with respect to
```

```
        the variable var.

        EXAMPLES:
            sage: f = axiom('1/((1+x)*(x-1))')
            sage: f.partial_fraction_decomposition('x')
                                    1           1
                                --------- - ---------
                                2 (x - 1)   2 (x + 1)
        """
        return self.partfrac(var)


class AxiomFunctionElement(FunctionElement):
    def _sage_doc_(self):
        return self._obj.parent().help(self._name)

class AxiomExpectFunction(ExpectFunction):
    def _sage_doc_(self):
        M = self._parent
        return M.help(self._name)


class AxiomFunction(AxiomElement):
    def __init__(self, parent, name, defn, args, latex):
        AxiomElement.__init__(self, parent, name, is_name=True)
        self.__defn = defn
        self.__args = args
        self.__latex = latex

    def __call__(self, *x):
        self._check_valid()
        P = self.parent()
        if len(x) == 1:
            x = '(%s)'%x
        return P('%s%s'%(self.name(), x))

    def __repr__(self):
        return self.__defn

    def _latex_(self):
        if self.__latex is None:
            return '\\mbox{\\rm %s}'%self.__defn
        else:
            return self.__latex

    def integrate(self, var):
```

```
        return self.integral(var)

    def integral(self, var):
        self._check_valid()
        P = self.parent()
        f = P('integrate(%s(%s), %s)'%(self.name(), self.__args, var))
        if var in self.__args.split(','):
            args = self.__args
        else:
            args = self.__args + ',' + var
        return P.function(args, str(f))

def is_AxiomElement(x):
    return isinstance(x, AxiomElement)

# An instance
axiom = Axiom(script_subdirectory=None)

def reduce_load_Axiom():
    return axiom

import os
def axiom_console():
    os.system('axiom')

def __doctest_cleanup():
    import sage.interfaces.quit
    sage.interfaces.quit.expect_quitall()
```