AXIOM

André Heck
CAN Expertise Center

December 1993

Contents

Chapter 1

A Tour of AXIOM

In this chapter we show most features of AXIOM in the form of a pseudo AXIOM session interleaved with explanatory remarks.

1.1 Launching AXIOM

To start AXIOM enter AXIOM from a Unix shell. When you are using the X Window environment, AXIOM will be started in the shell's window, and Hyperdoc will appear in another window.

The session window will look as follows:

```
Axiom Computer Algebra System Release 1.2
SunOS Version 4.1.2 for SPARC

                        Axiom Computer Algebra System
           Version of Wednesday March 17, 1993 at 18:04:01 on 609
-----------------------------------------------------------------------------
      Copyright The Numerical Algorithms Group Limited 1991.  All rights
 reserved.
      Certain derivative-work portions Copyright (C) 1988 by Leslie
 Lamport.
      Portions (c) Copyright Taiichi Yuasa and Masami Hagiya, 1984. All
 rights reserved.
      This version of the Axiom Computer Algebra System is based on Kyoto
 Common Lisp.
      This version of the Axiom Computer Algebra System is built on AKCL
 (Austin Kyoto Common Lisp) Version 1.609  Enhancements by W. Schelter.
-----------------------------------------------------------------------------
   Issue (top) if you arrive at a lisp error break.
   Issue )cd "directory" to reset the current directory.
   Issue )spool "filename" to save output in the given file.
   Issue )quit to leave Axiom and return to SunOS.
   Issue )help ? to see information about the )help command.
   Issue )hd to start the Hyperdoc help system.
   Issue )set message time on to have computation time data displayed.
-----------------------------------------------------------------------------

(1) ->connected session manager
```

```
(1) ->
```

Now everything is ready for our tour of AXIOM.

1.2 Numerics

We begin with a simple command.

```
(1) ->3 + 5
   loading /usr/local/lib/axiom_1.2/mnt/sun/algebra/PI.NRLIB for domain
   PositiveInteger
   loading /usr/local/lib/axiom_1.2/mnt/sun/algebra/NNI.NRLIB for domain
   NonNegativeInteger
   loading /usr/local/lib/axiom_1.2/mnt/sun/algebra/INT.NRLIB for domain
   Integer

   loading /usr/local/lib/axiom_1.2/mnt/sun/algebra/OUTFORM.NRLIB for
   domain OutputForm
   loading /usr/local/lib/axiom_1.2/mnt/sun/algebra/LIST.NRLIB for domain
   List
   (1)  8
                                                        Type: PositiveInteger
```

Let's try something equally easy.

```
(2) ->3**5

   (2)  243
                                                        Type: PositiveInteger
```

Now, AXIOM is less chatty. The reason is that AXIOM consists of a kernel for running the system, but without mathematical knowledge, and a mathematical library. In the first command $3+5$ AXIOM is going through the following steps.

- 3 is a positive integer. Since no mathematics is known yet in the AXIOM session, the system starts loading code related to the set of positive integers from the library file PI.NRLIB.
- $+$ is the operator for addition; what kind of addition depends on the second argument as well.
- 5 is again a positive integer.
- the two positive integers are added together using the code for the monoid of nonnegative integers from the NNI.NRLIB.
- The result is 8, a positive integer.
- The output formatting code is loaded from OUTFORM.NRLIB.
- Finally the result and its data type are printed.

So, whenever AXIOM needs mathematical routines it will load them automatically and keep them for the rest of the session in memory. That's why the second command does not create messages. In the rest of this tour we shall omit most messages from AXIOM about loading libraries.

Unlike a numerical calculator, AXIOM can compute exactly with large numbers.

```
(3) ->333**555

  (3)
 90479375840584230233975196584780569333401029041167587541803035578942737343576
```

You can get approximate numerical results. For example, the previous result (denoted by %) can be approximated as follows.

```
(4) ->numeric %

   (4)  0.9047937584 0584230234 E 1400
                                                                Type: Float
```

You can compute at any desired precision. Let's compute the numerical value of $\sqrt{3}$ in 40 digits (default is 20 digits).

```
(5) ->digits 40;
                                                     Type: PositiveInteger
```

Due to the semicolon at the end of the command line, the output is not shown on the screen. The space between `digits` and the number 30 has the same effect as the brackets in the equivalent command `digits(40)`

```
(6) ->numeric sqrt(3)

 (6)  1.7320508075 6887729352 7446341505 872366943
                                                                Type: Float
```

AXIOM can also handle complex numbers (the complex number $\sqrt{-1}$ is denoted by %i).

```
(7) ->1/(2+3*%i)
```

```
          1
   (7)  -------
        2 + 3%i
                                          Type: Fraction Complex Integer
```

Maybe not what you expected, but the standard form of a complex number can easily be obtained.

```
(8) ->% :: Complex Fraction Integer

         2    3
   (8)  -- - -- %i
        13   13
                                          Type: Complex Fraction Integer
```

In this example, you see clearly that in AXIOM a mathematical structure is strongly typed and hierarchical: each expression is of a certain type, and this data type determines what values and operations are defined. In AXIOM we speak of a *domain*, and every data object belongs to one and only domain. Expression `(7)` is a fraction of two Gaussian integers, whereas expression `(8)` is a complex number with rational real and imaginary part.

```
(9) ->[real %, imag %]

         2    3
   (9)  [--,- --]
         13   13
                                             Type: List Fraction Integer
```

The conversion from one domain to another is done by two contiguous colons (::) followed by a domain name. In the above example, we converted the complex number $\frac{1}{2+3i}$ into the standard form $\frac{2}{13} - \frac{3}{13}i$ by the command `% :: Complex Fraction Integer`

A domain in AXIOM has itself a type, which is called a *category*. For example, the domain *FractionInteger* and *PolynomialFractionInteger* both belong to the category Field. This allows the interpreter in AXIOM to understand all operations of a *Field* when applied to data objects in the domains. But even more, it allows appliance of all operations that are defined for categories in which *Field* is included such as *EuclidianDomain* or *Ring*. You can therefore code your programs in a general way, say for a *Ring*, and it works for every domain that belongs to this category.

AXIOM "knows" standard mathematical functions.

```
(10) ->sin 3.14

   (10)   0.0015926529 1648695254 0541436324 4432614433 21
                                                             Type: Float
(11) ->sin %pi

   (11)  0
                                                Type: Expression Integer
```

AXIOM provides all kinds of number systems. For example, roman numbers have been implemented and can be combined in the same computation with integers. The interpreter takes care of the necessary type conversions.

```
(12) ->roman 1993
```

```
   (12)  MCMXCIII
                                                    Type: RomanNumeral
(13) ->% + 1

   (13)  MCMXCIV
                                                    Type: RomanNumeral
```

Algebraic numbers and algebraic functions can be represented in AXIOM by their defining polynomials. For example, $\sqrt{2}$ can be represented by

```
(14) ->a | a**2=2
   Your statement has resulted in the following assignments and
   declaration:

   SAEa := SimpleAlgebraicExtension(Fraction Integer,UnivariatePolynomial(a,
Fraction Integer),a*a-2)
   a : SAEa := a

   (14)  a

Type: SimpleAlgebraicExtension(Fraction Integer,UnivariatePolynomial(a,
Fraction Integer),a*a-2)
```

Henceforth, AXIOM makes use of the fact that $\mathtt{a}^2 = 2$ to simplify expressions containing a.

```
(15) ->a**2

   (15)  2

Type: SimpleAlgebraicExtension(Fraction Integer,UnivariatePolynomial(a,
Fraction Integer),a*a-2)

(16) ->1/a

         1
   (16)  - a
         2

Type: SimpleAlgebraicExtension(Fraction Integer,UnivariatePolynomial(a,
Fraction Integer),a*a-2)
```

AXIOM allows you to do many exact calculations such as factorization of numbers.

```
(17) ->factor 9998999

   (17)  269 37171
                                                Type: Factored Integer
(18) ->factor( %::Complex Integer )

   (18)  - %i(10 + 13%i)(13 + 10%i)37171
                                        Type: Factored Complex Integer
```

There are not many tools for numerical tasks like numerical equation solving and numerical integration in AXIOM. The philosophy is that it is sufficient to provide in the system easy-to-use links to the best available numerical tools such as the NAG Fortran 77 library.

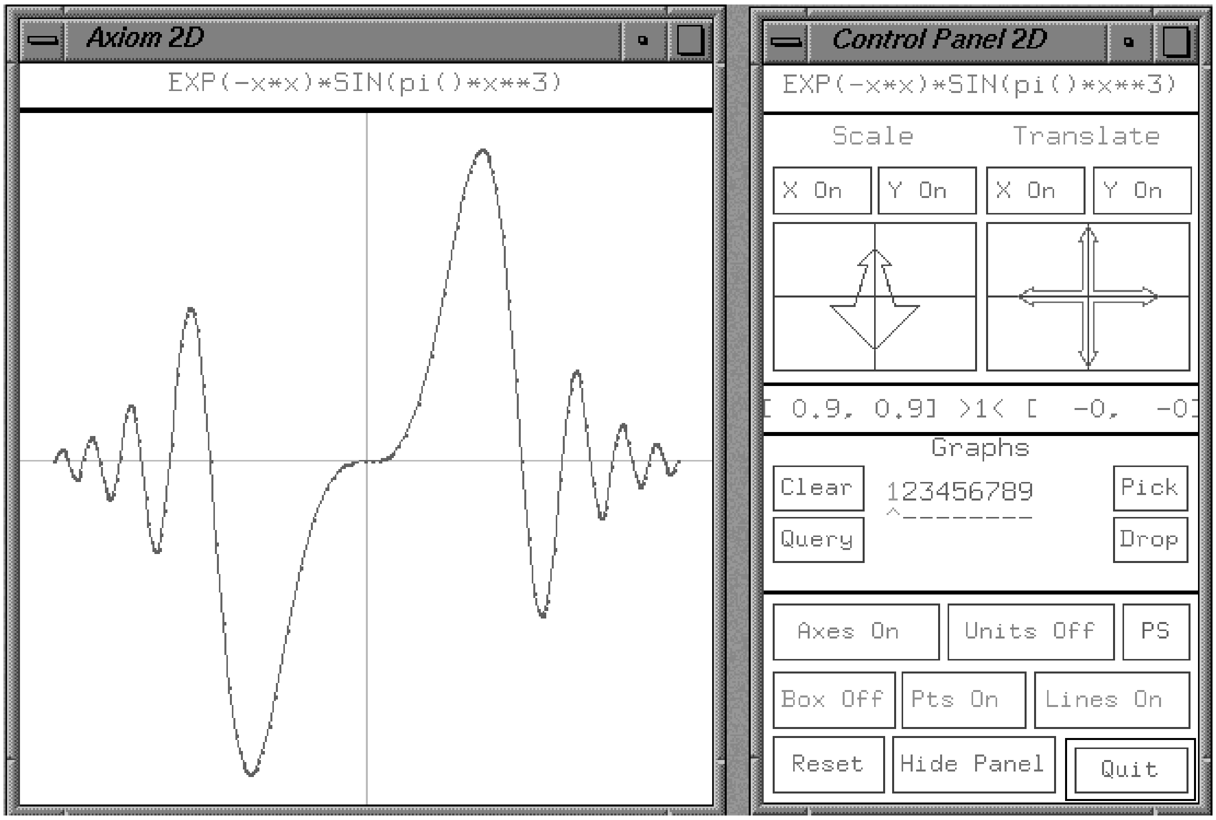

Figure 1.1: 2D graphics window and control panel

1.3 Graphics

You can plot the graph of a function, say of $e^{-x^2}\sin(\pi x^3)$, over some interval.

```
(19) ->draw( exp(-x**2) * sin(%pi*x**3), x=-2..2 )

   (19)  TwoDimensionalViewport: "EXP(-x*x)*SIN(pi()*x**3)"
                                                Type: TwoDimensionalViewport
```

The graph appears in a separate window and with a control panel you can change many of the options that determine how the graph actually looks like (see Figure 1.1).

Special two-dimensional graphics is provided, too. For example, you can plot a parametric curve in polar coordinates without the axes and sample points (see Figure 1.2)

```
(20) ->draw( curve((1-t**2)*cos(%pi*t),t), t=-1..1,_
             coordinates==polar )

   (20)  TwoDimensionalViewport: "((-t*t)+1)*COS(pi()*t)"
                                                Type: TwoDimensionalViewport
```

In the above command you see that the underscore is used in AXIOM as the continuation character.

A three-dimensional surface plot can be made as follows.

```
(21) ->draw( cos(x*y), x=-3..3, y=-3..3, style=="shade" )
```

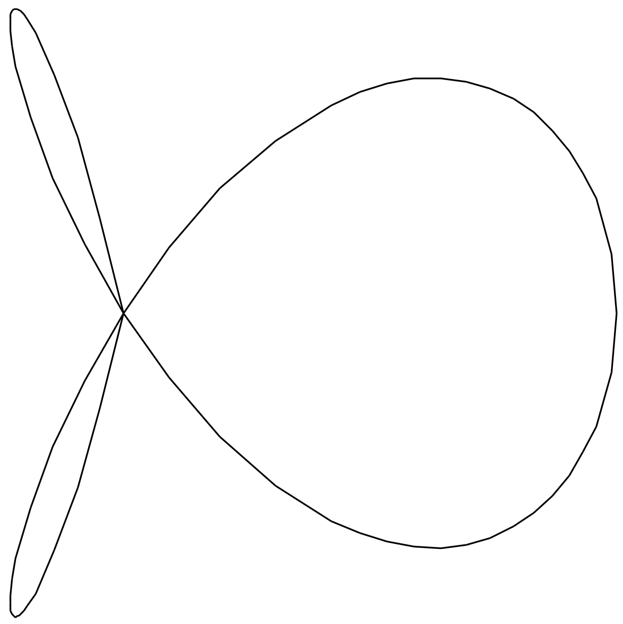

Figure 1.2: 2D parametric curve plot

```
   (21)  ThreeDimensionalViewport: "cos x*y"
                                          Type: ThreeDimensionalViewport
```

Again, many options can be changed interactively. Special 3D graphics is provided, too. For example, two parametric surfaces can be combined in one picture (see Figure 1.5).

```
(22) ->s := create3Space()$(ThreeSpace SF)

   (22)  3-Space with 0 components
                                             Type: ThreeSpace SmallFloat

(23) ->draw( 1, u=0..2*%pi, v=0..%pi, coordinates==spherical,_
            style=="shade", space==s )

   (23)  ThreeDimensionalViewport: "1"
                                          Type: ThreeDimensionalViewport

(24) ->draw( 1/2, u=0..2*%pi, v=-2..2, coordinates==cylindrical,_
            style=="shade", space==s )

   (24)  ThreeDimensionalViewport: "1/2"
                                          Type: ThreeDimensionalViewport
```

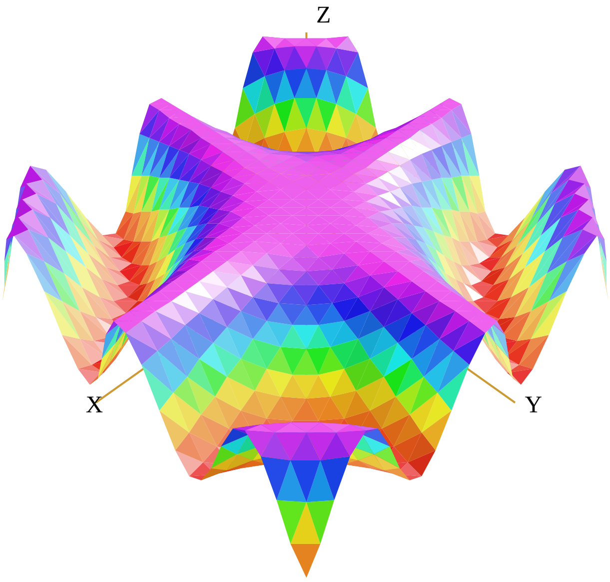

Figure 1.3: 3D surface plot

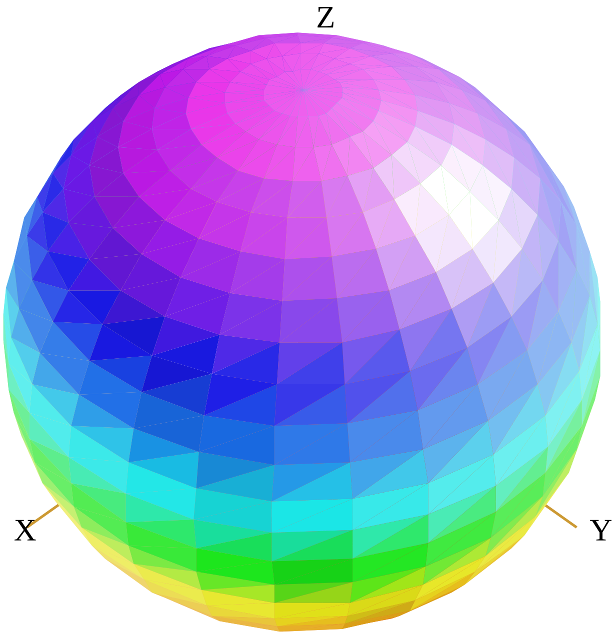

Figure 1.4: 3D spherical plot

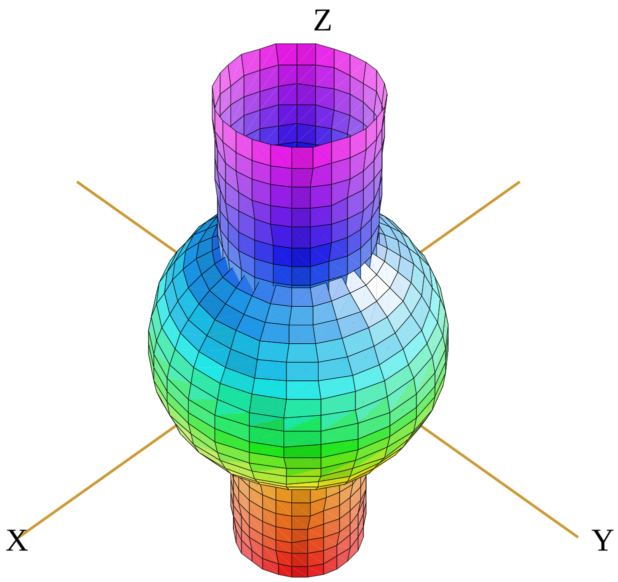

Figure 1.5: Combination of two 3D plots

1.4 Algebraic Facilities

1.4.1 Calculus

With AXIOM you can manipulate formulae. For example, you can expand and factorize polynomials.

```
(25) ->3*(x*y+z**2)**2*(x**2+y**2) - (x*y+z**2)

            2     2  4        3     3       2     2 4     4 2
   (25)  (3y  + 3x )z  + (6x y  + 6x y - 1)z  + 3x y  + 3x y  - x y
                                                     Type: Polynomial Integer
(26) ->%**2

   (26)
        4      2 2     4  8         5      3 3     2      5      2  6
     (9y  + 18x y  + 9x )z  + (36x y  + 72x y  - 6y  + 36x y - 6x )z
   +
         2 6       4 4        3      6 2      3       4
     (54x y  + 108x y  - 18x y  + 54x y  - 18x y + 1)z
   +
         3 7      5 5      2 4      7 3      4 2         2     4 8      6 6
     (36x y  + 72x y  - 18x y  + 36x y  - 18x y  + 2x y)z  + 9x y  + 18x y
   +
         3 5     8 4     5 3    2 2
     - 6x y  + 9x y  - 6x y  + x y
                                                     Type: Polynomial Integer
(27) ->factor %

           2       2    2     2  2       3     3      2
   (27)  (z  + x y) ((3y  + 3x )z  + 3x y  + 3x y - 1)
                                            Type: Factored Polynomial Integer
```

One of the highlights of AXIOM is integration of functions.

```
(28) ->1/(x**2+1)

           1
   (28)  ------
          2
         x  + 1
                                                  Type: Fraction Polynomial Integer
(29) ->integrate(%,x)

   (29)  atan(x)
                                                Type: Union(Expression Integer,...)
(30) ->1/(x**2-1)

           1
   (30)  ------
          2
         x  - 1
                                                  Type: Fraction Polynomial Integer
(31) ->integrate(%,x)

         - log(x + 1) + log(x - 1)
   (31)  -------------------------
                     2
                                                Type: Union(Expression Integer,...)
(32) ->1/(x**2+a)

           1
   (32)  ------
          2
         x  + a
                                                  Type: Fraction Polynomial Integer
(33) ->integrate(%,x)

                                                          x
                                                    atan(----)
               2      +---+              2             +-+
          log((x  - a)\|- a  + 2a x) - log(x  + a)      \|a
   (33)  [-------------------------------------,----------]
                           +---+                     +-+
                         2\|- a                     \|a
                                           Type: Union(List Expression Integer,...)
```

In the last example, AXIOM returns two alternative results of integration, depending on the sign of the parameter a. If the parameters and variables are complex numbers, then the notion of sign cannot be used, but the complex primitive function can be obtained as follows.

```
(34) ->complexIntegrate( 1/(x**2+a), x )
```

```
                +---+              +---+
             - \|- a  + x       - \|- a  - x
         log(------------) - log(------------)
                 +---+              +---+
               2\|- a             2\|- a
   (34)  -------------------------------------
                          +---+
                        2\|- a
                                                    Type: Expression Integer
```

More complicated integrals are:

```
(35) ->tan(1/3*atan(x))

             atan(x)
   (35)  tan(-------)
                3
                                                    Type: Expression Integer
(36) ->integrate(%,x)

                    atan(x) 2              atan(x) 2           atan(x)
         8log(3tan(-------)  - 1) - 3tan(-------)  + 18x tan(-------) + 16
                       3                      3                    3
   (36)  -----------------------------------------------------------------
                                       18
                                           Type: Union(Expression Integer,...)
(37) ->x/(x**3+x+a)

             x
   (37)  ----------
          3
         x  + x + a
                                              Type: Fraction Polynomial Integer
(38) ->integrate(%,x)

   (38)
           +----------------------+
           |      2          2
           |(- 81a  - 12)%%K0  - 4
         ( |----------------------  - %%K0)
           |          2
          \|       27a  + 4
      *
         log
                       2              3                  2
              (((162a  + 24)x + 243a  + 36a)%%K0 + 27a  + 4)
           *
```

```
                +---------------------+
                |     2          2
                |(- 81a  - 12)%%K0  - 4
                |---------------------
                |         2
               \|      27a  + 4
           +
                 2              3           2         2
             ((162a  + 24)x + 243a  + 36a)%%K0  + (- 27a  - 4)%%K0
           +
                 2
             (- 54a  + 4)x + 18a
    +
            +---------------------+
            |     2          2
            |(- 81a  - 12)%%K0  - 4
       (-   |---------------------  - %%K0)
            |         2
           \|      27a  + 4
     *
       log
                    2              3                 2
            (((162a  + 24)x + 243a  + 36a)%%K0 + 27a  + 4)
           *
                +---------------------+
                |     2          2
                |(- 81a  - 12)%%K0  - 4
                |---------------------
                |         2
               \|      27a  + 4
           +
                   2              3           2       2
             ((- 162a  - 24)x - 243a  - 36a)%%K0  + (27a  + 4)%%K0
           +
               2
             (54a  - 4)x - 18a
    +
                                   2
      2%%K0 log((6x + 9a)%%K0  - %%K0 + x)
 /
    2
                                               Type: Union(Expression Integer,...)
```

This strange symbol %%K0 is an algebraic function defined as the root of the following polynomial.

```
(39) ->definingPolynomial %%K0

              3 2            3
        27%%K0 a  + a + 4%%K0  + %%K0
   (39) -----------------------------
                       2
                    27a  + 4
                                                          Type: Expression Integer
```

Let us check the answer.

```
(40) ->differentiate( %%(38), x )

            x
   (40)  ----------
          3
         x  + x + a
                                                     Type: Expression Integer
```

It must be stressed that AXIOM has advanced algorithms such as the Risch algorithm built in to decide whether a closed form of an integral in terms of elementary functions (such as exp, ln, trigonometric functions, etc.) exists or not. If a closed form exist AXIOM will also have found it; if not, the system returns the integral as you can see in the next example.

```
(41) ->log(x+1)/x

         log(x + 1)
   (41)  ----------
             x
                                                     Type: Expression Integer
(42) ->integrate(%,x)

           x
          ++  log(%R + 1)
   (42)   |   ----------- d%R
         ++       %R
                                           Type: Union(Expression Integer,...)
```

For solving this integral the class of functions should be extended with the dilogarithm.

A series approximation of an expression is possible in AXIOM and can be further processed.

```
(43) ->x/(x**3+x+a)

            x
   (43)  ----------
          3
         x  + x + a
                                             Type: Fraction Polynomial Integer
(44) ->series(%,x=0)

   (44)
                              2           2              2
     1     1   2   1   3   - a  - 1  4   2a  + 1  5   - 3a  - 1  6
     - x - -- x  + -- x  + -------- x  + ------- x  + --------- x
     a      2       3          4            5             6
           a       a          a            a             a
   +
      4    2              4    2             4    2
     a  + 4a  + 1  7   - 3a  - 5a  - 1  8   6a  + 6a  + 1  9
     ------------ x  + --------------- x  + ------------- x
           7                  8                   9
          a                  a                   a
   +
```

```
        6       4      2             6      4      2
     - a  - 10a  - 7a  - 1  10   4a  + 15a  + 8a  + 1  11       12
     --------------------- x   + -------------------- x   + O(x  )
               10                          11
              a                           a
                          Type: UnivariatePuiseuxSeries(Expression Integer,x,0)
(45) ->1/%

            - 1        2      10
   (45)  a x    + 1 + x  + O(x  )
                          Type: UnivariatePuiseuxSeries(Expression Integer,x,0)
```

1.4.2 Solving Polynomial Equations

AXIOM has high-level tools to study systems of polynomial equations. We give one example.

```
(46) ->sys := [x**2+y**2=1, x*y=1]

           2    2
   (46)  [y  + x = 1,x y= 1]
                                         Type: List Equation Polynomial Integer
(47) ->vars := [x,y]

   (47)  [x,y]
                                                              Type: List Symbol
(48) ->solve( sys, vars )


                 3      4    2
   (48)  [[x= - y  + y,y  - y  + 1= 0]]
                           Type: List List Equation Fraction Polynomial Integer
```

AXIOM only simplifies the system of equations. You can try to find some rational approximation of the solution.

```
(49) ->solve( sys, 1/100 )

   (49)  []
                           Type: List List Equation Polynomial Fraction Integer
```

It looks as if there is no real solution. So, let's ask for complex solution!

```
(50) ->complexSolve( sys, 1/100 )

   (50)
        443   1       443   1             443   1          443   1
   [[y= --- - - %i,x= --- + - %i], [y= - --- - - %i,x= - --- + - %i],
        512   2       512   2             512   2          512   2
        221   1       221   1             221   1          221   1
    [y= --- + - %i,x= --- - - %i], [y= - --- + - %i,x= - --- - - %i]]
        256   2       256   2             256   2          256   2
                Type: List List Equation Polynomial Complex Fraction Integer
```

4 solutions! In floating-point arithmetic they are the following.

```
(51) ->complexSolve( sys, 0.01 )

   (51)
   [[y= 0.865234375 - 0.5 %i,x= 0.865234375 + 0.5 %i],
    [y= - 0.865234375 - 0.5 %i,x= - 0.865234375 + 0.5 %i],
    [y= 0.86328125 + 0.5 %i,x= 0.86328125 - 0.5 %i],
    [y= - 0.86328125 + 0.5 %i,x= - 0.86328125 - 0.5 %i]]
                           Type: List List Equation Polynomial Complex Float
```

AXIOM could have found the exact solution with `radicalSolve`.

```
(52) ->radicalSolve( sys, vars )

   (52)
                    +------------+
                    |   +---+
         +---+      |- \|- 3  + 1      +------------+
       (\|- 3  + 1) |------------      |   +---+
                   \|     2            |- \|- 3  + 1
   [[x= --------------------------,y=  |------------ ],
                    2                 \|     2
                      +------------+
                      |   +---+
           +---+      |- \|- 3  + 1        +------------+
       (- \|- 3  - 1) |------------        |   +---+
                     \|     2              |- \|- 3  + 1
    [x= ----------------------------,y= -  |------------ ],
                     2                    \|     2
                      +----------+
                      | +---+
           +---+      |\|- 3  + 1      +----------+
       (- \|- 3  + 1) |----------      | +---+
                     \|    2           |\|- 3  + 1
    [x= --------------------------,y=  |---------- ],
                    2                 \|    2
                    +----------+
                    | +---+
         +---+      |\|- 3  + 1        +----------+
       (\|- 3  - 1) |----------        | +---+
                   \|    2             |\|- 3  + 1
    [x= ------------------------,y= -  |---------- ]]
                   2                  \|    2
                                 Type: List List Equation Expression Integer
```

1.4.3 Lists and Streams

AXIOM can manipulate various types of composite data structures: lists, sets, vectors, matrices, and so on. The list of squares of the first five natural numbers can be entered as follows.

```
(53) ->[1,4,9,16,25]

   (53)  [1,4,9,16,25]
                                                  Type: List PositiveInteger
```

AXIOM also provides powerful tools to generate lists.

```
(54) ->[ i**2 for i in 1..15 ]

   (54)  [1,4,9,16,25,36,49,64,81,100,121,144,169,196,225]
                                                  Type: List PositiveInteger
(55) ->[ i**2 for i in 1.. ]

   (55)  [1,4,9,16,25,36,49,64,81,100,...]
                                                Type: Stream PositiveInteger

(56) ->[ i**2 for i in 1.. | odd? i]

   (56)  [1,9,25,49,81,121,169,225,289,361,...]
                                                Type: Stream PositiveInteger

(57) ->[ differentiate( sin(x), x, i ) for i in 1.. ]

   (57)
   [cos(x), - sin(x), - cos(x), sin(x), cos(x), - sin(x), - cos(x), sin(x),
    cos(x), - sin(x), ...]
                                             Type: Stream Expression Integer
```

The last three illustrate how elegantly AXIOM handles structures with infinite elements.

1.4.4 Matrices

To generate a 3×3 matrix whose (i,j)-th element is $\gcd(i,j)$ you can enter in AXIOM the following command.

```
(58) ->M := matrix( [ [ gcd(i,j) for i in 1..3 ] for j in 1..3 ] )

         +1  1  1+
         |       |
   (58)  |1  2  1|
         |       |
         +1  1  3+
                                                        Type: Matrix Integer
```

Standard matrix operations are available.

```
(59) ->M*M

         +3  4  5 +
         |        |
   (59)  |4  6  6 |
         |        |
         +5  6  11+
                                                        Type: Matrix Integer

(60) ->determinant M

   (60)  2
                                                       Type: PositiveInteger
```

```
(61) ->inverse M

        + 5          1+
        | -   - 1  - -|
        | 2          2|
        |             |
   (61) |- 1   1    0 |
        |             |
        |  1        1 |
        |- -   0    - |
        +  2        2 +
                                       Type: Union(Matrix Fraction Integer,...)
(62) ->%*M

        +1  0  0+
        |       |
   (62) |0  1  0|
        |       |
        +0  0  1+
                                                 Type: Matrix Fraction Integer
(63) ->eigenvalues M

   (63)  []
                                         Type: List Fraction Polynomial Integer
(64) ->eigenvectors M

   (64)
                                                         +    2          +
                                                         |- %B  + 6%B - 7|
               3      2                                  |               |
   [[algrel= %B  - 6%B  + 8%B - 2,algmult= 1,algvec= [|   2           |]]]
                                                         | %B  - 5%B + 4 |
                                                         |               |
                                                         +       1       +

Type: List Union(Record(algrel: Fraction Polynomial Integer,algmult: Integer,
algvec: List Matrix Fraction Polynomial Integer),Record(eigval: Fraction
Polynomial Integer,eigmult: Integer,eigvec: List Matrix Fraction Polynomial
Integer))
```

The procedure `eigenvalues` tried to compute the rational eigenvalues; there are none. The procedure `eigenvectors` is more informative: it gives the defining polynomial for an algebraic eigenvalue and the corresponding eigenvector. The procedures `radicalEigenvalues` and `radicalEigenvectors` would give explicit results. As an example of interactive linear algebra, we compute the numerical eigenvalues of the matrix M by elementary matrix algebra. Let's first compute the characteristic matrix.

```
(65) ->M-x
```

```
        +- 1   0    0 +    +1  1  1+
        |             |    |       |
   (65) | 0   - 1   0 |x + |1  2  1|
        |             |    |       |
        + 0    0   - 1+    +1  1  3+
                                    Type: Polynomial SquareMatrix(3,Integer)
```

Oops, AXIOM considers this as a polynomial with matrix coefficients. Luckily, an explicit domain conversion brings us back on the road.

```
(66) ->% :: Matrix Polynomial Integer

        +- x + 1     1        1   +
        |                         |
   (66) |   1     - x + 2     1   |
        |                         |
        +   1        1     - x + 3+
                                          Type: Matrix Polynomial Integer
```

We could also have obtained the previous result by the following command.

```
(67) ->M - x * diagonalMatrix([1,1,1])

        +- x + 1     1        1   +
        |                         |
   (67) |   1     - x + 2     1   |
        |                         |
        +   1        1     - x + 3+
                                          Type: Matrix Polynomial Integer
```

Next, we compute the characteristic polynomial and we find numerical approximations of its roots (in low precision).

```
(68) ->determinant %

           3     2
   (68) - x  + 6x  - 8x + 2
                                                 Type: Polynomial Integer
```

The same result as would have been obtained with the command `characteristicPolynomial( M, x )`.

```
(69) ->solve( %, 0.01 )

   (69) [x= 4.21484375,x= 1.45703125,x= 0.32421875]
                                       Type: List Equation Polynomial Float
```

Of course, this result could have been obtained directly in AXIOM.

```
(70) ->realEigenvalues(M,0.01)

   (70) [4.21484375,1.45703125,0.32421875]
                                                        Type: List Float
```

1.5 Programming

Below is a one-line function definition and some function calls.

```
(71) ->f(x)==(1-x)/(1+x)
                                                                   Type: Void
(72) ->f(3)
   Compiling function f with type PositiveInteger -> Fraction Integer

           1
   (72)  - -
           2
                                                        Type: Fraction Integer
(73) ->f(-1)
   Compiling function f with type Integer -> Fraction Integer
   >> Error detected within library code:
   division by zero
   You are being returned to the top level of the interpreter.

(73)limit(f(x),x=-1)

   (73)  [leftHandLimit= - infinity,rightHandLimit=  + infinity]

Type: Union(Record(leftHandLimit: Union(OrderedCompletion Fraction Polynomial
Integer,"failed"),rightHandLimit: Union(OrderedCompletion Fraction Polynomial
Integer,"failed")),...)
```

The next procedure generates a Hilbert matrix. Typechecking is implemented as well.

```
(74) ->H(n:PositiveInteger): Matrix Fraction Integer ==_
       matrix( [ [ 1/(i+j-1) for i in 1..n ] for j in 1..n ] )

   Function declaration H : PositiveInteger -> Matrix Fraction Integer has
   been added to workspace.
                                                                   Type: Void
(75) ->H(4)

         +   1  1  1+
         |1  -  -  -|
         |   2  3  4|
         |          |
         |1  1  1  1|
         |-  -  -  -|
         |2  3  4  5|
   (75)  |          |
         |1  1  1  1|
         |-  -  -  -|
         |3  4  5  6|
         |          |
         |1  1  1  1|
         |-  -  -  -|
         +4  5  6  7+
                                                  Type: Matrix Fraction Integer
```

```
(76) ->H(0)

   Cannot convert from type Integer to PositiveInteger for value 0

(76) ->[ determinant(H(i)) for i in 1.. ]

   (76)
       1    1       1          1                1
   [1, --, ----, -------, ------------, -----------------,
       12  2160  6048000  266716800000  18631342033920000
                1                              1
    -------------------------, ---------------------------------,
    2067909047925770649600000  365356847125734485878112256000000
                        1
    -------------------------------------------,
    1028781784378569697887052962909388800000000
                             1
    ----------------------------------------------------, ...]
    4620689394791469131629562883903627872698368000000000
                                            Type: Stream Fraction Integer
```

The following program counts the occurrences of elements in a list of integers.

```
(77) ->frequency( x:List Integer ): List List Integer ==_
                 [[a, count(a, x)] for a in removeDuplicates(x) ]

                                                             Type: Void
(78) ->frequency( [1,1,2,3,3,3,1] )

   (78)  [[1,3],[2,1],[3,3]]
                                                 Type: List List Integer
```

As you see, functional programming is one of the programming styles that AXIOM offers. The next program combines two lists of integers into a single list whose first element is the maximum of the two first elements in the original lists, and so on.

```
(79) ->maxList( x:List Integer, y:List Integer): List Integer ==_
               map( max, x, y )
                                                             Type: Void
(80) ->maxList( [2,5,7,3], [3,2,8,2] )

   (80)  [3,5,8,3]
                                                      Type: List Integer
```

Recursive programming is allowed in AXIOM. Below, the function `F` computes Fibonacci polynomials.

```
(81) ->F(0,x) == 0
                                                             Type: Void
(82) ->F(1,x) == 1
                                                             Type: Void
(83) ->F(n,x) == x*F(n-1,x) + F(n-2,x)
                                                             Type: Void
(84) ->F(5,y)
   Compiling function F with type (Integer,Variable y) -> Polynomial Integer
```

```
        4     2
(84)  y   + 3y  + 1
                                           Type: Polynomial Integer
```

Last but not least, we mention the rule-based programming style. The next section gives some examples of rules and pattern matching in AXIOM.

We end this section with an example of a procedural programming style. Below is a procedure to compute the mean of a list of floating-point numbers.

```
(85) ->mean(l:List Float): Float ==                  _
(local sum,length;                                   _
 length := #l;                                       _
 sum := 0;                                           _
 for i in 1..length repeat sum := sum + l(i);       _
 return(sum/length) )

   Function declaration mean : List Float -> Float has been added to
   workspace.
                                                      Type: Void
```

As you see, the usual localisation of variables, loop structures, and explicit returns are available in AXIOM. Let's apply the procedure to a simple list of floating-point numbers.

```
(86) ->L := [1.3, 4.4, 3.0 ]

   (86)  [1.3,4.4,3.0]
                                                Type: List Float
(87) ->mean L

   The type of the local variable sum has changed in the computation.
   We will attempt to interpret the code.
   Compiling function mean with type List Float -> Float

   (87)  2.9
                                                Type: List Float
```

1.6 Rules and Pattern Matching

Let's introduce two rules for expansion of a logarithmic expression.

```
(87) ->rl1 := rule log(x*y) == log(x) + log(y)

   (87)  log(x y) == log(y) + log(x)

                        Type: RewriteRule(Integer,Integer,Expression Integer)

(88) ->rl2 := rule log(x**y) == y * log(x)

                y
   (88)  log(x ) == y log(x)

                        Type: RewriteRule(Integer,Integer,Expression Integer)
```

```
(89) ->logExpandRules := ruleset( [rl1, rl2] )

                                                   y
   (89)  {log(x y) == log(y) + log(x),log(x ) == y log(x)}

                            Type: Ruleset(Integer,Integer,Expression Integer)

(90) ->log(a*b*c**d)

                 d
   (90)  log(a b c )
                                                    Type: Expression Integer
(91) ->logExpandRules %

   (91)  d log(c) + log(b) + log(a)
                                                    Type: Expression Integer
```

Patterns can be restricted by predicates. For example, rule `lr2` can be restricted to integers `y` in the following way.

```
(92) ->rl2 := rule log(x**(y|integer? y)) == y * log(x)

             y
   (92)  log(x ) == y log(x)

                        Type: RewriteRule(Integer,Integer,Expression Integer)

(93) ->logExpandRules2 := ruleset( [rl1, rl2] )

                                                   y
   (93)  {log(x y) == log(y) + log(x),log(x ) == y log(x)}

                            Type: Ruleset(Integer,Integer,Expression Integer)

(94) ->logExpandRules2 log(a*b*c**d)

             d
   (94)  log(c ) + log(b) + log(a)
                                                    Type: Expression Integer
(95) ->logExpandRules2  log(a*b*c**3)

   (95)  3log(c) + log(b) + log(a)
                                                    Type: Expression Integer
```

You can set up rules for any kind of object. The following commands associate a rule with an operator f.

```
(96) ->f := operator("f",1)

   Function definition for f is being overwritten.
   Compiled code for f has been cleared.

   (96)  f
                                                         Type: BasicOperator
```

```
(97) ->frule := rule( f(x) + f(y) == f(x+y) )

   (97)  f(y) + f(x) + %G == 'f(y + x) + %G

                        Type: RewriteRule(Integer,Integer,Expression Integer)

(98) ->f(a) + f(b) + f(c)

   (98)  f(c) + f(b) + f(a)
                                                 Type: Expression Integer
(99) ->frule %

   (99)  f(c + b + a)
                                                 Type: Expression Integer
```

1.7 AXIOM **Packages**

AXIOM comes with various packages for specialized applications. Let's look at one of them: Clifford algebras. In a 2-dimensional Clifford algebra an element can be represented as $x = a + be_1 + ce_2 + de_{12}$, ($e_1$, e_2, e_{12} are the basis elements). Multiplication is subject to the rules $e_1 e_2 = -e_2 e_1 = e_{12}$, $e_1 e_1 = Q(e_1)$, and $e_2 e_2 = Q(e_2)$, where Q is a given quadratic form. A typical computation in the Clifford algebra is expansion of polynomials such as $x^3 + 1$. In AXIOM the computation is organised as follows. First define the field to work over.

```
(100) ->K := Fraction Polynomial Integer

   (100)  Fraction Polynomial Integer
                                                         Type: Domain
```

Next, define the matrix m for the quadratic form.

```
(101) ->m := matrix [[1,0],[0,1]]

          +1  0+
   (101)  |    |
          +0  1+
```

Finally, introduce the domain.

```
                                                 Type: Matrix Integer
(102) ->C := CliffordAlgebra( 2, K, quadraticForm(m) )

   (102)  CliffordAlgebra(2,Fraction Polynomial Integer,MATRIX)
                                                         Type: Domain
```

Now, we are ready for computing in this Clifford algebra.

```
(103) ->x : C := a + b*e(1) + c*e(2) + d*e(1)*e(2)

   (103)  a + b e  + c e  + d e e
                 1      2      1 2

                    Type: CliffordAlgebra(2,Fraction Polynomial Integer,MATRIX)
```

```
(104) ->x**3 + 1

   (104)
          2        2        2    3              2       2    3     2
     - 3a d  + 3a c  + 3a b  + a  + 1 + (- b d  + b c  + b  + 3a b)e
                                                                    1
   +
          2    3     2      2             3      2    2     2
     (- c d  + c  + (b  + 3a )c)e  + (- d  + (c  + b  + 3a )d)e e
                                 2                           1 2
                  Type: CliffordAlgebra(2,Fraction Polynomial Integer,MATRIX)
```

1.8 Formatting

AXIOM allows not only symbols as simple as `x`, but also complicated ones such as

```
(105) ->script( X, [[i],[j],[k],[l]] )

          k j
   (105)   X
          l i
                                                                 Type: Symbol
```

This symbol can be converted into LaTeX format.

```
(106) ->outputAsTex %

\[
{} \sb {l} \sp {k}X \sb {i} \sp {j}
\]
%AXIOM STEP NUMBER: 106

                                                                   Type: Void
```

This is typeset as

$$ {} _{l} ^{k}X _{i} ^{j} $$

You can compute with such strange symbols as with any other one.

```
(107) ->integrate( sin(%% 105), %% 105 )

                k j
   (107)  - cos( X )
                l i
                                          Type: Union(Expression Integer,...)
```

You can also convert formulae in (optimized) Fortran 77. An example.

```
(108) ->)set fortran precision double

(108) ->outputAsFortran( (a+b+c)**3 )

      DOUBLE PRECISION T2,T1
      T1=b*b
      T2=a*a
```

```
      R110=c**3+(3.0D0*b+3.0D0*a)*c*c+(3.0D0*T1+6.0D0*a*b+3.0D0*T2)*c+b*
     &*3+3.0D0*a*T1+3.0D0*T2*b+a**3
                                                              Type: Void
```

In single precision you can set default type declarations.

```
(113) ->)set fortran precision single

(113) ->)set fortran defaulttype REAL

(113) ->outputAsFortran((a+b+c)**3)

      REAL T2,T1
      T1=b*b
      T2=a*a
      R113=c**3+(3.*b+3.*a)*c*c+(3.*T1+6.*a*b+3.*T2)*c+b**3+3.*a*T1+3.*T
     &2*b+a**3
                                                              Type: Void
```

1.9 Interface

AXIOM offers on-line help, examples, tutorials, a browser, a user guide, and reference manual via HyperDoc in hypertext format. You can explicitly start HyperDoc in an AXIOM session by

```
(114) ->)hd
```

or by entering **hypertex** from a Unix shell. A separate HyperDoc window appears (See Figure 1.6), which forms the "home window" in the hypertext documentation and help system.

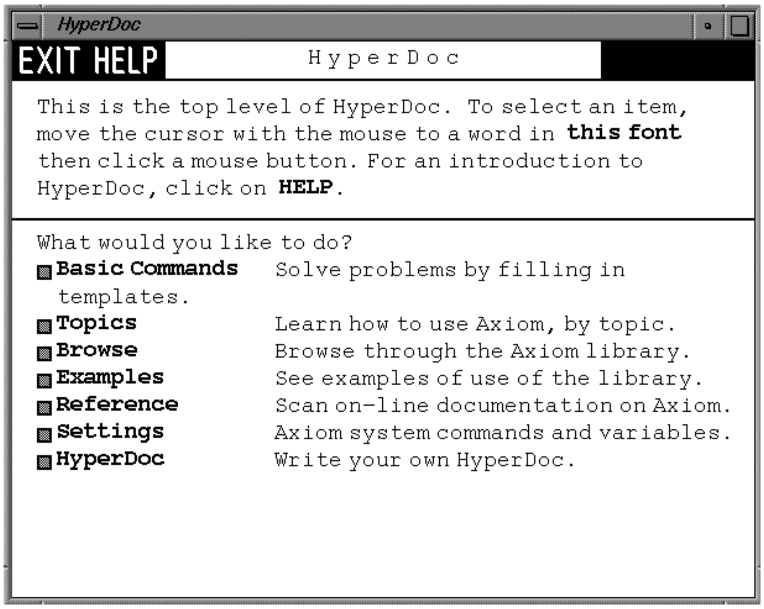

Figure 1.6: Home HyperDoc indow

To get information about a category, say the category *Ring*, you may enter

```
(114) ->)show Ring

 Ring is a category constructor.
 Abbreviation for Ring is RING
 This constructor is exposed in this frame.
 Issue )edit catdef.spad to see algebra source code for RING

------------------------------- Operations --------------------------------

 ?*? : ($,$) -> $                      ?*? : (Integer,$) -> $
 ?**? : ($,NonNegativeInteger) -> $    ?+? : ($,$) -> $
 -? : $ -> $                           ?-? : ($,$) -> $
 ?=? : ($,$) -> Boolean                1 : () -> $
 0 : () -> $                           coerce : Integer -> $
 coerce : $ -> OutputForm              one? : $ -> Boolean
 recip : $ -> Union($,"failed")        zero? : $ -> Boolean
 characteristic : () -> NonNegativeInteger
```

Or you can use the browser inside HyperDoc.

1.10 Quitting

Finally, to quit AXIOM you enter the quit command and confirm this.

```
)quit
   Please enter y or yes if you really want to leave the interactive
   environment and return to the operating system:
y
```