```
eval(p,x,10)

             2        2         4
   (2)  - 4z  + 40y z - 100y  + 1000

   Type: P I
```

The value need not be a constant. You can substitute another polynomial for the variable.

```
eval(p,x,z+1)

   (3)
     3       4      2     2        4     2
    z  + (- y  + 4y  - 1)z  + (- 2y  + 4y  + 3)z
   +
       4
    - y  + 1

   Type: P I
```

There also a form of *eval* that does *simultaneous* substitution for several variables.

```
eval(p,[x,y,z],[y,z,x])

            2 4        2    3     2
   (4)  - y z  + 4x y z  + y  - 4x

   Type: P I
```

As one would expect, this is different from sequential substitution.

```
eval(eval(eval(p,x,y),y,z),z,x)

            6     4    3     2
   (5)  - x  + 4x  + x  - 4x

   Type: P I
```

Q: What is the difference between f and g as defined by

```
f == 3
g() == 3
```

A: The difference is that g is a nullary function that may be called to return the value 3 while f is a rule that evaluates to 3. You can see this by defining and g as above and and then asking for their values.

```
f
   Compiling body of rule f to compute value of type I

   (3)  3

   Type: I
```

So when you mention f you get 3.

```
g

   (4)  g () == 3

   Type: □
```

That is, g is a function. You must call the function to get 3.

```
g()
   Compiling function g with signature () -> I

   (5)  3

   Type: I
```

Rules are convenient to use when the definitions have dependencies on values that are changing. Functions may also be used in this case and must be used when you have one or more arguments or when you want to create a nullary function object. For example, the function ***apply*** will take a nullary function object and then call it, returning the result.

```
apply nullaryFun == nullaryFun()

h : () -> I

h() == 3

apply h
   Compiling function apply with signature
      (() -> I) -> I
   Compiling function h with signature () -> I

   (8)  3
```

Streams and Power Series

Streams have been in Scratchpad II for some time. They are now implemented by a domain constructor `Stream` and may be infinite, unlike lists. A stream is represented by a list whose last element is a function that contains the wherewithal to create the rest of the list from that point, should it ever be required.

The stream functions provided are ***take***, ***drop***, ***elt***, ***null***, ***cons***, ***first*** and ***rest*** (similar to the list functions), together with functions for creating finite and infinite streams. There are also packages that supply several general purpose functions from streams to streams. Since some of these functions operate on functions, another package called `MappingPackage` has been provided to simplify the expression of functional arguments. Some examples follow:

```
)set streams calculate 5
```

All evaluated elements are printed, but at least the first 5 will be evaluated.

```
a := [1..]

   (1)  [1,2,3,4,5,...]

b := [i+1 for i in a]

   (2)  [2,3,4,5,6,...]
```

Select the 20th element:

```
b.20

   (3)  22
```

The first 21 elements of b are evaluated:

```
b

   (4)
   [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,

    16, 17, 18, 19, 20, 21, 22, ...]
```

The stream of odd integers

```
[i for i in a | oddp i]

   (5)  [1,3,5,7,9,...]

[[i,j] for i in a for j in b]

   (6)  [[1,2],[2,3],[3,4],[4,5],[5,6],...]

)set streams calculate 10
```

append(a,b) concatenates streams a and b

```
append([i for i in a while i<7],a)

   (7)
   [1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,

    11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, ...]
```

The sum of a finite stream of integers:

```
reduce(0,_+$I,take(a,10))

   (8)  55
```

A stream of partial sums:

```
scan(0,_+$I,a)

   (9)  [1,3,6,10,15,21,28,36,45,55,...]
```

To save space infinite streams of the same element have a loop at the end.

```
a:ST I := repeating([8])

   (10)  [8̄]

[i+1 for i in a]

   (11)  [9̄]
```

The functions in the Stream domain and stream packages are particularly suitable for the implementation of algorithms on power series. The domain PowerSeries is provided as a field, and the domain UnivariatePowerSeries and an elementary function package adds to it the functions ***exp, log, sin, cos, tan,***, composition, lagrange inversion, reversion together with the solution of linear differential equations in power series.

A general method of producing programs which solve recursion or differential equations in power series by the method of undetermined coefficients has been developed in which the program can be written down almost immediately from the defining relation. In the method of undetermined coefficients a trial series together with an initial value or two is substituted into the recursion or differential equation, and then coefficients of equal powers are equated.

In these programs the trial series is made up of the initial values followed by the as yet unevaluated stream. The tail of the stream is then defined in terms of the whole stream and when elements are required the trial series becomes the resulting stream. The program, because it uses functions that operate on whole streams, rather than stream elements has the same structure as the defining relation.

For example e raised to the power series power $A(x)$, has defining relation

$$(e^{A(x)})' \equiv A'(x)e^{A(x)}$$

The corresponding program for generating the power series exp A, in Scratchpad II, where A is a power series is

```
exp A == integrate(1,deriv A*exp A))
```

in which ***integrate*** and ***deriv*** integrate and differentiate power series. Some examples follow. The command

```
)set streams calculate n
```

will cause the series up to the n^{th} coefficient to be printed.

The following declares and assigns x to be a UPS(x,RN), in other words a UnivariatePowerSeries with variable x and with rational number coefficients.

```
x := ps x
```

$$(12) \quad x$$

We can now compute easily with power series involving x.

```
exp x
```

$$(13) \quad 1 + x + (\frac{1}{2})x^{2} + (\frac{1}{6})x^{3} + (\frac{1}{24})x^{4} + (\frac{1}{120})x^{5} + (\frac{1}{720})x^{6} + (\frac{1}{5040})x^{7} + (\frac{1}{40320})x^{8} + (\frac{1}{362880})x^{9} + (\frac{1}{3628800})x^{10} + O(x^{11})$$

```
cos x ** cos x
```

$$(14) \quad 1 - (\frac{1}{2})x^{2} + (\frac{7}{24})x^{4} - (\frac{19}{180})x^{6} + (\frac{1597}{40320})x^{8} - (\frac{373}{32400})x^{10} + O(x^{11})$$

```
x/(exp x-1)
```

$$(15) \quad 1 - (\frac{1}{2})x + (\frac{1}{12})x^{2} - (\frac{1}{720})x^{4} + (\frac{1}{30240})x^{6} - (\frac{1}{1209600})x^{8} + (\frac{1}{47900160})x^{10} + O(x^{11})$$

```
exp(exp x-1)
```

$$(16) \quad 1 + x + x^{2} + (\frac{5}{6})x^{3} + (\frac{5}{8})x^{4} + (\frac{13}{30})x^{5} + (\frac{203}{720})x^{6} + (\frac{877}{5040})x^{7} + (\frac{23}{224})x^{8} + (\frac{1007}{17280})x^{9} + (\frac{4639}{145152})x^{10} + O(x^{11})$$

```
atan (x)
```

$$(18) \quad x - (\frac{1}{3})x^{3} + (\frac{1}{5})x^{5} - (\frac{1}{7})x^{7} + (\frac{1}{9})x^{9} + O(x^{11})$$

Power series provide a method of solving differential equations when all else fails. The function ***lde*** solves the n^{th} order linear differential equation, its argument is a list of power series coefficients. The two solutions of

$$y'' + (\cos x)y' + (\sin x)y = 0$$

are

```
lde([sin x,cos x])
```

$$(19) \quad \left[\, 1 - (\frac{1}{2})x^{2} + (\frac{1}{6})x^{4} - (\frac{31}{720})x^{6} + (\frac{379}{40320})x^{8} - (\frac{1639}{907200})x^{10} + O(x^{11}),\ x - (\frac{1}{3})x^{3} + (\frac{1}{10})x^{5} - (\frac{59}{2520})x^{7} + (\frac{31}{6480})x^{9} + O(x^{11}) \,\right]$$

Power series are also used as enumerating generating functions. For example, the function ***lambert*** will transform one series into another in which the coefficient A_n of x^n is the sum of the coefficients of the original a_n for all i that divide n, including 1 and n. In other words, if $f(x)$ is a power series, then *lambert(f)* is the power series

$$f(x) + f(x^2) + f(x^3) + f(x^4) + \ldots$$

The series for the number of divisors of n is

```
lambert(x/(1-x))
```

$$(20) \quad x + 2x^{2} + 2x^{3} + 3x^{4} + 2x^{5} + 4x^{6} + 2x^{7} + 4x^{8} + 3x^{9} + 4x^{10} + O(x^{11})$$

Using this function it is possible to expand certain infinite products as power series. For example the enumerating generating function for partitions is

$$\prod_{n=1}^{\infty} \frac{1}{(1-q^n)}$$

```
partitions := exp(lambert(log(1/(1-x))))

   (21)
                2     3     4     5      6      7
      1 + x + 2x  + 3x  + 5x  + 7x  + 11x  + 15x
    +
         8      9      10       11
      22x  + 30x  + 42x   + O(x  )

euler := 1/partitions

                  2    5    7      11
   (22)  1 - x - x  + x  + x  + O(x  )
```

The generating function for partitions into distinct parts is:

$$\prod_{n=1}^{\infty}(1+q^n)$$

```
exp(lambert(log(1+x)))

   (23)
                2     3     4     5     6     7     8
      1 + x + x  + 2x  + 2x  + 3x  + 4x  + 5x  + 6x
    +
       9      10       11
     8x  + 10x   + O(x  )
```

The generating function for the Legendre polynomials is

$$\frac{1}{(1-2xt+t^2)^{1/2}}$$

and with suitable declarations for x and t may be expanded directly, as follows.

```
(1-2*x*t+t**2)**(-1/2)

   (24)
                         2         2
      1 + x*t + ((3/2)x  - 1/2)t
    +
                3           3
      ((5/2)x  - (3/2)x)t
    +
                 4          2        4
      ((35/8)x  - (15/4)x  + 3/8)t
    +
                 5          3             5
      ((63/8)x  - (35/4)x  + (15/8)x)t
    +
                   6            4             2          6
      ((231/16)x  - (315/16)x  + (105/16)x  - 5/16)t
    +
                        7            5             3
            (429/16)x  - (693/16)x  + (315/16)x
          +
            - (35/16)x
     *
          7
         t
    +
            8
        O(t )
```

These examples show the present capability of writing expressions that denote power series. It should be possible in the future to enter differential or recursion equations that define new power series in terms of existing ones as suggested in the example for exp above. Other plans are to make multivariate power series more usable and to add Puiseux series.

William H. Burge
Stephen M. Watt
Scott C. Morrison

Primary Decomposition of Ideals

Scratchpad II now provides a facility for the primary decomposition of polynomial ideals over fields of characteristic zero. The algorithm is discussed in [1] and works in essentially two steps:

1. the problem is solved for 0-dimensional ideals by "generic" projection on the last coordinate
2. a "reduction process" uses localization and ideal quotients to reduce the general case to the 0-dimensional one.

The Scratchpad II constructor IdealDomain represents ideals with coefficients in any field and supports the basic ideal operations, including intersection, sum and quotient. IdealDecompositionPackage contains the specific functions for the primary decomposition and the computation of the radical of an ideal with polynomial coefficients in a field of characteristic 0 with an effective algorithm for factoring polynomials.

The follow examples illustrate the capabilities of this facility. First consider the ideal generated by x^2+y^2-1 (which defines a circle in the (x,y)-plane) and the ideal generated by x^2-y^2 (corresponding to the straight lines $x = y$ and $x = -y$.

```
f,g : DMP([x,y],RN)
n,m : L DMP([x,y],RN)

m := [x**2+y**2-1]

          2    2
   (3)  [x  + y  - 1]

   Type: L DMP([x,y],RN)

n := [x**2-y**2]

          2    2
   (4)  [x  - y ]

   Type: L DMP([x,y],RN)
```