Examples by topic

This is a small excerpt from the **REDUCE User's Manual** [REDUM], translated to `Pure` syntax. For any details we refer to that document. With this guide it should be straightforward to interprete back and forth. The REDUCE User's Manual as well as the documentation of each package and other valuable information may be found at

http://www.reduce-algebra.com/documentation.htm

Differentiation

The operator `df` is used to represent partial differentiation with respect to one or more variables.

```
syntax: df exprn [var <num>]+.
```

Differentiation of the function $x^2y^3z^4$ with respect to x, y, z , two, three and four times respectively, i.e $\frac{\partial^9 x^2y^3z^4}{\partial x^2 \partial y^3 \partial z^4}$:

```
> simplify $ df (x^2*y^3*z^4) x 2 y 3 z 4 ;
288
```

The derivative of $\log \sin(x)^2$:

```
> simplify $ df (log(sin x)^2) x;
2*cos x*log (sin x)/sin x

Note the parentheses.
```

Suppose $z(\cos(x), y)$: Let's calculate $\frac{\partial \sin(z)}{\partial \cos(x)}$ and $\frac{\partial z^2}{\partial x}$:

```
> declare depend [z,cos x,y];
[]
> simplify (df (sin z) (cos x));
cos z*df z (cos x)
> simplify (df (z^2) x);
2*df z x*z

Note how to declare dependencies.
```

The results are $\cos(z) \frac{\partial z}{\partial \cos(x)}$ and $2z \frac{\partial z}{\partial x}$) respectively, as expected.

Integration

`INT` is an operator in `REDUCE` for indefinite integration using a combination of the Risch-Norman algorithm and pattern matching.

```
syntax: intg exprn var.
```

Note that in `Pure` the operator is called `intg` in order not to clash with the integer type `int`.

Example 1:

$$\int \frac{1}{ax+b}\,dx$$

```
> simplify $ intg (1/(a*x+b)) x;
log (a*x+b)/a
```

Example 2:

$$I(a,b,n) = \int x^2 (ax+b)^n dx$$

```
> I a b n = simplify $ intg (x^2*(a*x+b)^n) x;
> I a b n;
((a*x+b)^n*a^3*n^2*x^3+3*(a*x+b)^n*a^3*n*x^3+2*(a*x+b)^n*a^3*x^3+
 (a*x+b)^n*a^2*b*n^2*x^2+(a*x+b)^n*a^2*b*n*x^2-2*(a*x+b)^n*a*b^2*
  n*x+2*(a*x+b)^n*b^3)/(a^3*n^3+6*a^3*n^2+11*a^3*n+6*a^3)
> I a b 0 ;
x^3/3
> I 0 b n;
b^n*x^3/3
> I a 0 k;
x^k*a^k*x^3/(k+3)
```

Example 3:

$$\int \frac{\sqrt{x+\sqrt{x^2+1}}}{x}$$

```
> simplify $ intg (sqrt(x+sqrt(x^2+1))/x) x ;
intg (sqrt (sqrt (x^2+1)+x)/x) x
```

Apparently no solution was found. There is a package `ALGINT` in REDUCE, that is specialized to deal with algebraic functions. The REDUCE User's Manual [REDUM] says

> ... *will analytically integrate a wide range of expressions involving square roots where the answer exists in that class of functions. It is an implementation of the work described in J.H. Davenport* [LNCS102]

```
> reduce::load "algint" ;
0
> simplify $ intg (sqrt(x+sqrt(x^2+1))/x) x ;
atan ((sqrt (sqrt (x^2+1)+x)*sqrt (x^2+1)-sqrt (sqrt (x^2+1)+x)*x-sqrt
(sqrt (x^2+1)+x))/2)+2*sqrt (sqrt (x^2+1)+x)+log (sqrt (sqrt
(x^2+1)+x)-1)-log (sqrt (sqrt (x^2+1)+x)+1)

Note how to load packages.
```

Length and Map

LENGTH is a generic operator for finding the length of various objects in the system, while the MAP operator applies a uniform evaluation pattern to all members of a composite structure: a matrix, a list, or the arguments of an operator expression.

```
syntax: length exprn

syntax: map fun obj

> simplify $ length (a+b);
2
> simplify $ length (x^n+a*x+2);
3

> simplify $ map sqrt [1,2,3];
[1,2^(1/2),3^(1/2)]
> simplify $ map log  [x^n,x^m,sin x] ;
[log (x^n),log (x^m),log (sin x)]

> simplify $ map (\y->df y x)  [x^n,x^m,sin x] ;
[x^n*n/x,x^m*m/x,cos x]
> simplify $ map (\y->intg y x)  [x^n,x^m,sin x] ;
[x^n*x/(n+1),x^m*x/(m+1),-cos x]
```

Note that the lambda expression in REDUCE is replaced by the corresponding Pure version.

Partial Fractions

The PF operator transforms an expression into a list of partial fractions with respect to the main variable. PF does a complete partial fraction decomposition.

```
syntax: pf expr var
```

Let us find the decomposition of:

$$f(x) = \frac{2}{(x+1)^2\,(x+2)}$$

```
> let f = 2/((x+1)^2*(x+2))
> simplify $ pf f x;
[2/(x+2),(-2)/(x+1),2/(x^2+2*x+1)]
```

this means

$$f(x) = \frac{-2}{x+3} + \frac{2}{x+2} + \frac{2}{x^2+2x+1}$$

If one wants the denominators in factored form, one has to use the switch off exp:

```
> reduce::switch "exp" 0 ;
0
>
> simplify $ pf f x;
[2/(x+2),(-2)/(x+1),2/(((x:1):1):1)^2] // (x+1)^2 ???
```

Note how the switch `off exp` is used in `Pure`.

Selection

The `SELECT` operator extracts from a list, or from the arguments of an `n-ary` operator, elements corresponding to a boolean predicate. It is used with the syntax:

```
syntax: select fun list
```

Solving

`SOLVE` is an operator for solving one or more simultaneous algebraic equations. It is used with the syntax:

```
syntax: solve expr [var | varlist]
```

where `expr` is a list of one or more expressions. Each expression is an algebraic equation, or is the difference of the two sides of the equation.

Example 1:

Find the solutions to

$$\log(\sin(x+3))^5 = 8$$

```
> let eqn1 = log(sin (x+3))^5 == 8 ;
> let sol1 = simplify $ solve eqn1 x;
```

The variable `sol1` now contains a huge list of solutions. How many?

```
> #sol1 ;
10

The first one is:

> sol1!0;
x==2*val "\0x256c\(0x2593)5"*pi+asin (e^(2^(3/5)*cos (2*pi/5))/e^(2^(3/5)*
sin (2*pi/5)*i))-3
```

$$x = 2 \cdot n \cdot \pi + \mathbf{asin}\left(\frac{e^{2^{\frac{3}{5}} \cdot \cos\left(\frac{2 \cdot \pi}{5}\right)}}{e^{2^{\frac{3}{5}} \cdot \sin\left(\frac{2 \cdot \pi}{5}\right) \cdot i}}\right) - 3$$

where `n` is an arbitrary integer constant.

It is also possible - for example - to obtain the righthand side of any solution in the list via `REDUCE` commands:

```
> simplify $ rhs $ first $ solve eq1 x;
2*val "\0x256c\(0x2593)10"*pi+asin (e^(2^(3/5)*cos (2*pi/5))/e^(2^(3/5)*
sin (2*pi/5)*i))-3
>
```

where `first` gets the first solution in the list and `rhs` takes the righthand side. Hence there is a wealth of possibilities to process the solution list.

Example 2:

For the sake of clarity some simpler examples:

$$X^2 + 1 = 0$$

```
> simplify $ solve [X^2+1==0] X;
[X==i,X==-i]
```

$$(x + 3\,y = 7) \wedge (y - x = 1)$$

```
> simplify $ solve [x+3*y==7,y-x==1] [x,y] ;
[[x==1,y==2]]
```

To get the multiplicities turn on the switch `multiplicities`:

```
> simplify $ solve [x^2==2*x-1] x;
[x==1]

> reduce::switch "multiplicities" 1;
0

> simplify $ solve [x^2==2*x-1] x;
[x==1,x==1]
```

For details consult the REDUCE user manual.

Even and Odd Operators

An operator can be declared to be even or odd in its first argument by the declarations `EVEN` and `ODD` respectively.

```
> declare operator [f1,f2];
[]
> declare odd f1;
[]
> declare even f2;
[]

> simplify $ f1(-a);
-f1 a

> simplify $ f2 (-a);
f2 a

> simplify $ f1 (-a) (-b);
-f1 a (-b)
```

Linear Operators

An operator can be declared to be linear in its first argument over powers of its second argument.

$$L(a\,x^5 + b\,x + c, x) = L(x^5, x) \cdot a + L(x, x) \cdot b + L(1, x) \cdot c$$

```
> declare operator L;
[]
> declare linear L;
[]
> simplify $ L (a*x^5+b*x+c) x ;
L (x^5) x*a+L x x*b+L 1 x*c
```

$$L(a+b+c+d,y) = L(1,y)\cdot(a+b+c+d)$$

```
> simplify $ L (a+b+c+d) y;
L 1 y*a+L 1 y*b+L 1 y*c+L 1 y*d

Note that L x y binds stronger than (*).
```

Non-commuting Operators

An operator can be declared to be non-commutative under multiplication by the declaration `NONCOM`.

```
>
> declare operator [u,v];
[]
> simplify (u(x)*u(y)-u(y)*u(x));
0
> declare noncom [u,v];
[]
> simplify (u(x)*u(y)-u(y)*u(x));
u x*u y-u y*u x
```

Symmetric and Antisymmetric Operators

An operator can be declared to be symmetric with respect to its arguments by the declaration `SYMMETRIC`, Similarly the declaration `ANTISYMMETRIC` declares an operator antisymmetric.

```
> declare operator [A,S];
[]
> declare symmetric S;
[]
> declare antisymmetric A;
[]

> simplify $ A x x ;
0

> simplify $ (A x y z) + (A x z y) ;
0

> simplify $ S y x ;
S x y

> simplify $ A y x ;
-A x y
```

Creating/Removing Variable

Dependency

There are several facilities in REDUCE, such as the differentiation operator and the linear operator facility, that can utilize knowledge of the dependency between various variables. Such dependency may be expressed by the command DEPEND.

```
> declare operator D ;
[]
> declare depend [D,x,y];
[]

> simplify $ df D a;
0

D does not depend on a => 0, but

> simplify $ df D x;
df D x

because D depends on x by definition.
If we let a also depend on x, then

> declare depend [a,x];
[]
> simplify $ df (D*a) x;
df D x*a+df a x*D
```

Note *that dependencies remain active until they are explicitly removed*

```
> declare nodepend [a,x];
> simplify $ df a x;
0
> simplify $ df (D*a) x;
df D x*a
```

Internal Order of Variables

It is possible for the user to change the internal order of variables by means of the declaration KORDER. The syntax for this is:

```
syntax: declare korder [v1,...,vn];
```

Unlike the ORDER declaration, that has a purely cosmetic effect on the way results are printed, the use of KORDER can have a significant effect on computation time.

```
> declare korder [z,y,x];
[]
> x+y+z;
x+y+z
> simplify $ x+y+z;
z+y+x
```

Parts of Algebraic Expressions

The following operators can be used to obtain a specific part of an

expression, or even change such a part to another expression.

- `coeff expr::polynomial var`
- `coeffn expr::polynomial var n::int`
- `part expr::algebraic [n::int])`

Examples:

```
> simplify $ coeff ((y^2+z)^3/z) y ;
[z^2,0,3*z,0,3,0,1/z]

> simplify $ coeffn ((y^2+z)^3/z) y 6;
1/z

> simplify $ part (a+b) 2 ;
b

> simplify $ part (a+b) 1 ;
a

> simplify $ part (a+b) 0 ;
(+)
```

PART may also be used to substitute for a given part of an expression. In this case, the PART construct appears on the left-hand side of an assignment statement, and the expression to replace the given part on the right-hand side.

```
>  simplify $ xx:=a+b;
a+b
>  simplify $ part xx 2 := c ;
c
>  simplify $ xx;
a+c
```

Polynomials and Rationals

Factorization of Polynomials

REDUCE is capable of factorizing univariate and multivariate polynomials that have integer coefficients, finding all factors that also have integer coefficients. The package for doing this was written by Dr. Arthur C. Norman and Ms. P. Mary Ann Moore at The University of Cambridge. It is described in [SYMSAC81].

```
factorize expr::polynomial [p::prime]
```

Some examples:

```
> simplify $ factorize (x^105-1) ;
[[x^48+x^47+x^46-x^43-x^42-2*x^41-x^40 ... ]
>
> reduce::switch "ifactor" 1;
0
> simplify $ factorize (12*x^2 - 12) ;
[[2,2],[3,1],[x+1,1],[x-1,1]]
> reduce::switch "ifactor" 0;
```

The following operators should be well known:

- **gcd** `expr1::polynomial expr2::polynomial -> polynomial`
- **lcm** `expr1::polynomial expr2::polynomial -> polynomial`
- **remainder** `expr1::polynomial expr2::polynomial -> polynomial`
- **resultant** `expr1::polynomial expr2::polynomial var -> polynomial`
- **decompose** `expr::polynomial -> list`
- **interpol** `<values> <variable> <points>) -> polynomial`
- **deg** `expr::polynomial var ->int`
- **den** `expr::rational -> polynomial`
- **lcof** `expr::polynomial var -> polynomial`
- **lpower** `expr::polynomial var-> polynomial`
- **lterm** `expr::polynomial var -> polynomial`
- **mainvar** `expr::polynomial -> expr`
- **num** `expr::rational -> polynomial`
- **reduct** `expr::polynomial var -> polynomial`

Some examples of each operator:

GCD/LCM

```
> simplify $ gcd (x^2+2*x+1) (x^2+3*x+2) ;
x+1
> simplify $ gcd (2*x^2-2*y^2) (4*x+4*y) ;
2*x+2*y
> simplify $ gcd (x^2+y^2) (x-y) ;
1
>
> simplify $ lcm (x^2+2*x+1) (x^2+3*x+2) ;
x^3+4*x^2+5*x+2
> simplify $ lcm (2*x^2-2*y^2) (4*x+4*y) ;
4*x^2-4*y^2
>
```

REMAINDER/RESULTANT

```
> simplify $ remainder ((x+y)*(x+2*y)) (x+3*y) ;
2*y^2
> simplify $ remainder (2*x+y) 2 ;
y
>
> simplify $ resultant (x/r*u+y) (u*y) u ;
-y^2
>
```

DECOMPOSE

```
> simplify $ decompose (x^8-88*x^7+2924*x^6-43912*x^5+263431*x^4-
>                                 218900*x^3+65690*x^2-7700*x+234) ;
[u^2+35*u+234,u==v^2+10*v,v==x^2-22*x]
>
> simplify $ decompose (u^2+v^2+2*u*v+1) ;
[w^2+1,w==u+v]
>
```

DEG/DEN

```
> simplify $ deg ((a+b)*(c+2*d)^2) d ;
2
> simplify $ deg ((x+b)*(x^6+2*y)^2) x ;
13
>
> simplify $ den (x/y^2) ;
y^2
>
```

LCOF/LPOWER/LTERM

```
> simplify $ lcof ((a+b)*(c+2*d)^2) a ;
c^2+4*c*d+4*d^2
> simplify $ lcof ((a+b)*(c+2*d)^2) d ;
4*a+4*b
> simplify $ lcof ((a+b)*(c+2*d)) ('e) ;
a*c+2*a*d+b*c+2*b*d
>
> simplify $ lpower ((a+b)*(c+2*d)^2) a ;
a
> simplify $ lpower ((a+b)*(c+2*d)^2) d ;
d^2
> simplify $ lpower ((a+b)*(c+2*d)) x ;
1
>
> simplify $ lterm ((a+b)*(c+2*d)^2) a ;
a*c^2+4*a*c*d+4*a*d^2
> simplify $ lterm ((a+b)*(c+2*d)^2) d ;
4*a*d^2+4*b*d^2
> simplify $ lterm ((a+b)*(c+2*d)) x ;
a*c+2*a*d+b*c+2*b*d
>
```

MAINVAR/NUM/REDUCT

```
> simplify $ mainvar ((a+b)*(c+2*d)^2) ;
a
> simplify $ mainvar 2 ;
0
>
> simplify $ num (x/y^2) ;
x
> simplify $ num ('(100/6)) ;
50
> simplify $ num (a/4+b/6) ;
3*a+2*b
>
> simplify $ reduct ((a+b)*(c+2*d)) a ;
b*c+2*b*d
> simplify $ reduct ((a+b)*(c+2*d)) d ;
a*c+b*c
> simplify $ reduct ((a+b)*(c+2*d)) x ;
0
>
```

Substitution

An important class of commands in REDUCE define substitutions for variables and expressions to be made during the evaluation of expressions. Such substitutions use (among others) the prefix operator SUB.

```
syntax: sub <substlist> exprn::algebraic -> algebraic

> simplify $ sub [x==a+y,y==y+1] (x^2+y^2) ;
```

```
a^2+2*a*y+2*y^2+2*y+1

> simplify $ sub [a==sin x, b==sin y] (a^2+b^2) ;
> sin x^2+sin y^2
```

Assignment

One may assign values to variables in the REDUCE environment. Note that in Pure the set operator and := are equivalent, i.e. both sides are evaluated, contrary to the := version in REDUCE.

```
syntax: set expr expr ; or  expr := expr

> simplify $ P := a*x^n + b* x^m + c ;      // P:=a*x^n + b* x^m + c;
x^m*b+x^n*a+c
> simplify P ;                              // return P (from Reduce)
x^m*b+x^n*a+c
> simplify $ df P x;                        // diff P x
(x^m*b*m+x^n*a*n)/x
> simplify $ Q := intg P x ;                // integrate P x, store in Q
(x^m*b*n*x+x^m*b*x+x^n*a*m*x+x^n*a*x+c*m*n*x+c*m*x+c*n*x+c*x)/(m*n+m+n+1)
>

> simplify $ set Q (a*x^n + b* x^m + c) ;
x^m*b+x^n*a+c
```

Matrix Calculations

A very powerful feature of REDUCE is the ease with which matrix calculations can be performed. It fits very well into Pure's native matrix type.

To keep it simple we show the usage of the different operators by examples using the well known Pauli matrices. There is no loss of generality using only (2x2) matrices.

$$\sigma_1 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad \sigma_2 = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad \sigma_3 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

Pauli matrices (sigma 1..3). see e.g. http://en.wikipedia.org/wiki/Pauli_matrices for a reference.

```
let s0 = {1,0;0,1} ;
let s1 = {0,1;1,0} ;
let s2 = {0,-i;i,0};
let s3 = {1,0;0,-1};
```

Check the identities

$$\sigma_1^2 = \sigma_2^2 = \sigma_3^2 = -i\sigma_1\,\sigma_2\,\sigma_3 = \sigma_0$$

where σ_0 denotes the unit matrix.

```
Note: s1^2 or s1*s1 works.

> let r1 = simplify $ (s1*s1) ; r1;
```

```
{1,0;0,1}
> let r2 = simplify $ (s2*s2) ; r2;
{1,0;0,1}
> let r3 = simplify $ (s3*s3) ; r3;
{1,0;0,1}
> let r4 = simplify $ (-i*s1*s2*s3) ; r4;
{1,0;0,1}
> let r5 = all (==s0) [r1,r2,r3,r4] ; r5;
1
```

Check: $\det \sigma_i = -1, \forall i \in \{1,2,3\}$.

```
> map (simplify . det) [s1,s2,s3] ;
[-1,-1,-1]
```

Calculate the Eigenvalues/-vectors of σ_2 :

```
> let r7 = simplify $ mateigen s2 q; r7;
[[q-1,1,{-c1*i;c2}],[q+1,1,{c3*i;c4}]]

> let r8 = map head r7; r8; // -> [q-1,q+1] => Eigenvalues q=+/-1
[q-1,q+1]

> let r9 = map (head.tail) r7 ; r9; // multiplicities
[1,1]

> let r10 = map last r7 ; r10; // eigenvectors
[{-c1*i;c2},{c3*i;c4}]
```

Transpose (operator `tp`):

```
> map (simplify.tp) [s1,s2,s3] ; // -> [s1',s2',s3']
[{0,1;1,0},{0,i;-i,0},{1,0;0,-1}]
```

Trace (operator `trace`):

```
> map (simplify.trace) [s1,s2,s3] ;
[0,0,0]
```

Cofactor (trivial here)

```
> simplify $ cofactor s2 1 1 ;
0
```

Nullspace of σ_2 + {0,i;0,0}

```
> simplify $ nullspace (s2+{0,i;0,0}) ;
[{0;1}]
```

Rank

```
> map (simplify . rank) [s0,s1,s2,s3] ;
[2,2,2,2]
```

Inverse (simply $\frac{1}{matrix}$)

```
> let r15 = simplify $ 1/s2 ; r15;
{0,1/i;(-1)/i,0}

> simplify $ s2*r15 ;
{1,0;0,1}
```

Solving without `solve`:

$$a_{11}\,x_1 + a_{12}\,x_2 = y_1$$
$$a_{21}\,x_1 + a_{22}\,x_2 = y_2$$

```
> simplify $ (1/{a11,a12;a21,a22}*{y1;y2}) ; // A^-1 * y' ;
{(-a12*y2+a22*y1)/(a11*a22-a12*a21);(a11*y2-a21*y1)/(a11*a22-a12*a21)}
```

Limits

From the package description: *LIMITS is a fast limit package for REDUCE for functions which are continuous except for computable poles and singularities, based on some earlier work by Ian Cohen and John P. Fitch.* This package defines a LIMIT operator, called with the syntax:

```
limit expr::alg var limpoint::alg -> alg
```

$$\lim_{x\to\infty} x\,\sin\frac{1}{x} = ? \quad \wedge \lim_{x\to 0}\frac{1}{x} = ?$$

```
> simplify $ limit (x*sin(1/x)) x infinity ;
1

> simplify $ limit (1/x) x 0 ;
inf
```

Notes: This package loads automatically. Author: Stanley L. Kameny.

Ordinary differential equations solver

The ODESOLVE package is a solver for ordinary differential equations.

Problem 1:

$$\frac{dy}{dx} = x^2 + e^x$$

```
> declare depend [y,x];  // declare: y depends on x
[]

> simplify $ odesolve [df y x == x^2+exp(x)] [y] x ;
[y==(3*C+3*e^x+x^3)/3]
```

Problem 2:

$$\frac{d^2\,y}{dx^2} = y(x) \quad \wedge \quad y(0) = A \quad \wedge \quad y(1) = B$$

```
> simplify $ odesolve [(df y x 2) == y] [y] x [[x==0,y==A],[x==1,y==B]] ;
[y==(-e^(2*x)*A+e^(2*x)*B*e+A*e^2-B*e)/(e^x*e^2-e^x)]
```

Remember to remove dependencies

```
> declare nodepend [y,x];
```

```
[]
```

Series Summation and Products

SUM: A package for series summation

From the package description:

The package implements the Gosper algorithm for the summation of series. It defines operators SUM and PROD. The operator SUM returns the indefinite or definite summation of a given expression, and PROD returns the product of the given expression. This package loads automatically. Author: Fujio Kako.

Calculate

$$\sum_{n=1}^{N} n^3 \ldots \sum_{k=0}^{n-1} (a+k\,r) \ldots \sum_{k=1}^{n+1} \frac{1}{(p+(k-1)\,q)\cdot(p+k\,q)} \ldots \prod_{k=1}^{N} \frac{k}{k+2}$$

```
> simplify $ sum (n^3) n 1 N ;
(N^4+2*N^3+N^2)/4

> simplify $ sum (a+k*r) k 0 (n-1) ;
(2*a*n+n^2*r-n*r)/2

> simplify $ sum (1/((p+(k-1)*q)*(p+k*q))) k 1 (n+1) ;
(n+1)/(n*p*q+p^2+p*q)

> simplify $ prod (k/(k+2)) k 1 N ;
2/(N^2+3*N+2)
```

Taylor Series

TAYLOR: Manipulation of Taylor series

From the package description:

This package carries out the Taylor expansion of an expression in one or more variables and efficient manipulation of the resulting Taylor series. Capabilities include basic operations (addition, subtraction, multiplication and division) and also application of certain algebraic and transcendental functions. Author: Rainer Schöpf.

Example:

$$e^{x^2+y^2} = 1 + y^2 + x^2 + y^2 \cdot x^2 + O(x^3, y^3)$$

For details consult the package documentation in the REDUCE distribution.

```
> simplify $ taylor (exp (x^2+y^2)) x 0 2 y 0 2 ;
x^2*y^2+x^2+y^2+1

> simplify $ taylor (exp x) x 0 3;
(x^3+3*x^2+6*x+6)/6
```

```
> simplify $ implicit_taylor (x^2+y^2-1) x y 0 1 5 ;
(-x^4-4*x^2+8)/8

> simplify $ inverse_taylor (exp(x)-1) x y 0 8;
(-105*y^8+120*y^7-140*y^6+168*y^5-210*y^4+280*y^3-420*y^2+840*y)/840
```

Boolean Expressions

The truth values within `REDUCE` are `t` and `nil = ()` but are mapped to `1` and `0` respectively when interchanging results using `simplify`. Not all predicates (functions returning a truth value), however, can be called by `simplify`, so one has to use the `lisp` function in some rare cases.

Some examples:

```
> simplify $ evenp 200 ;
1
> simplify $ evenp 201 ;
0

> lisp (fixp 200) ;
t

where fixp tests for integers.
```

The following example shows a pitfall. Since there is a `numberp` in Pure as well as in REDUCE one has to be careful:

```
 > lisp (numberp x) ;
 0

 > lisp (quote (numberp x)) ;
[]

 > lisp (quote (numberp 111)) ;
 t
```

In the first case `numberp x` evaluates to zero in Pure, so the `lisp` function gets `0` and returns `0`. In the second case (quoted) the function `numberp` is evaluated in REDUCE and returns `nil`, i.e. `[]` in Pure. Of course, both results are correct but there may be other cases where equally named functions have different meanings in the two environments.

Some other useful predicates in REDUCE are `ordp` and `freeof`:

```
> lisp (ordp x y) ;
t
> lisp (ordp y x) ;
[]
> lisp (ordp "abc" "abd") ;
t
> lisp (ordp "abd" "abc") ;
[]
> lisp (ordp 3 5) ;
[]
> lisp (ordp 5 3) ;
t

> simplify $ freeof (x^2+y) x ;
0
```

```
> simplify $ freeof (x^2+y) z ;
1
> simplify $ freeof (x^n*y^m) (y^m) ;
0
```

Mathematical Functions

REDUCE provides many mathematical functions that can take arbitrary scalar expressions as their single argument:

- ACOS ACOSH ACOT ACOTH ACSC ACSCH ASEC ASECH ASIN ASINH
- ATAN ATANH ATAN2 COS COSH COT COTH CSC CSCH DILOG EI EXP
- HYPOT LN LOG LOGB LOG10 SEC SECH SIN SINH SQRT TAN TANH ERF

Note, however, if there is an equally named function in Pure and no quotes are used then the Pure function is used, that is for example, `cos x`, means cos in Pure, `(quote cos) x` means cos in REDUCE ...

See the difference:

```
> simplify $ (cos 4.3);
cos (43/10)
> using math;
warning: external 'exp' shadows previous undefined use of this symbol
warning: external 'sin' shadows previous undefined use of this symbol
warning: external 'cos' shadows previous undefined use of this symbol
> simplify $ (cos 4.3);
(-21601483)/53896027
```

Some examples:

```
> simplify $ cos (-x) ;
cos x
> simplify $ cos (n*pi) ;
cos (80143857*n/25510582)
> simplify $ (quote e)^(3*i*(quote pi)/2) ;
-i
> simplify $ sec (quote pi);
-1
> let simplify $ log10 10 ;
1
> simplify $ erf (-a);
-erf a
```

The special functions are in two separate packages `SPECFN` and `SPECFN2`:

- Bernoulli Numbers and Euler Numbers;
- Stirling Numbers;
- Binomial Coefficients;
- Pochhammer notation;
- The Gamma function;
- The Psi function and its derivatives;
- The Riemann Zeta function;
- The Bessel functions J and Y of the first and second kind;
- The modified Bessel functions I and K;

- The Hankel functions H1 and H2;
- The Kummer hypergeometric functions M and U;
- The Beta function, and Struve, Lommel and Whittaker functions;
- The Airy functions;
- The Exponential Integral, the Sine and Cosine Integrals;
- The Hyperbolic Sine and Cosine Integrals;
- The Fresnel Integrals and the Error function;
- The Dilog function;
- Hermite Polynomials;
- Jacobi Polynomials;
- Legendre Polynomials;
- Spherical and Solid Harmonics;
- Laguerre Polynomials;
- Chebyshev Polynomials;
- Gegenbauer Polynomials;
- Euler Polynomials;
- Bernoulli Polynomials.
- Jacobi Elliptic Functions and Integrals;
- 3j symbols, 6j symbols and Clebsch Gordan coefficients;

In SPECFN2 are the generalized hypergeometric functions and Meijer's G function.

Author: Chris Cannam, with contributions from Winfried Neun, Herbert Melenk, Victor Adamchik, Francis Wright and several others.

Definite Integrals

Package: DEFINT (definite integrals) Calculating integrals by using the Meijer G integration formula.

$$\int_0^\infty e^{-x}\,dx$$

```
> reduce::load "defint" ;
0

> simplify $ intg (exp(-x)) x 0 infinity ;
1
```

$$\int_0^\infty x^2\,\cos(x)\,e^{-2\,x}\,dx$$

```
> simplify $ intg (x^2*cos(x)*exp(-2*x)) x 0 infinity ;
4/125
```

$$\int_0^1 x\,e^{-\frac{1}{2}\,x}\,dx$$

```
> simplify $ intg (x*exp(-1/2*x)) x 0 1 ;
2*sqrt e*(2*sqrt e-3)/e
```

$$\int_0^1 x\,\log(1+x)\,dx$$

```
> simplify $ intg (x*log(1+x)) x 0 1 ;
1/4
```

$$\int_y^{2\,y} \cos(2\,x)\,dx$$

```
> simplify $ intg (cos(2*x)) x y (2*y);
(sin (4*y)-sin (2*y))/2
```

Various transformations:

```
> simplify $ laplace_transform (exp(-a*x)) x ;
1/(a+s)

> simplify $ hankel_transform (exp(-a*x)) x ;
s^(n/2)*gamma (n/2)*hypergeometric [(n+2)/2] [n+1]
((-s)/a)*n/(2*a^(n/2)*gamma (n+1)*a)

> simplify $ y_transform (exp(-a*x)) x ;
(a^n*gamma (n+1)*gamma ((-n)/2)*gamma ((-2*n-1)/2)*gamma
((2*n+3)/2)*hypergeometric [(-n+2)/2] [-n+1] ((-s)/a)+s^n*gamma
(-n)*gamma (n/2)*hypergeometric [(n+2)/2] [n+1] ((-s)/a)*n*pi)/
(2*s^(n/2)*a^(n/2)*gamma ((-2*n-1)/2)*gamma ((2*n+3)/2)*a*pi)

> simplify $ k_transform (exp(-a*x)) x ;
(-a^n*gamma (n+1)*gamma ((-n)/2)*hypergeometric [(-n+2)/2] [-n+1]
(s/a)+s^n*gamma (-n)*gamma (n/2)*hypergeometric [(n+2)/2] [n+1] (s/a)*n)/
(4*s^(n/2)*a^(n/2)*a)

>  simplify $ struveh_transform (exp(-a*x)) x ;
2*s^((n+1)/2)*gamma ((n+3)/2)*hypergeometric [1,(n+3)/2] [(2*n+3)/2,3/2]
((-s)/a)/(sqrt pi*a^((n+1)/2)*gamma ((2*n+3)/2)*a)

> simplify $ fourier_sin (exp(-a*x)) x ;
s/(a^2+s^2)
> simplify $ fourier_cos (exp(-a*x)) x ;
a/(a^2+s^2)
```

Declarations, Switches and Loading

Lisp evaluation can be used in the REDUCE system, in particular, to declare operator symbols and their properties (simplify won't do that). E.g.:

```
> lisp ('operator [myop]);
> lisp ('flag [myop] odd);
> lisp ('prop myop); // => [odd:t,simpfn:simpiden]
> simplify (myop (-x)); // => -myop x
```

For the most common kinds of declarations, the reduce module already provides the 'declare' function which takes care of the necessary Lisp magic and is safe to use. The above example can also be done as follows:

```
> declare operator myop;
> declare odd myop;
> simplify (myop (-x));
```

```
-myop x
```

For a list of supported declarations via `declare` consult the module file `reduce.pure`.

In `Pure` the REDUCE switches can be turned `on/off` as follows:

```
reduce::switch "switch-id" 0/1 ;
```

A package can be loaded by the command

```
reduce::load "package-id" ;
```

A REDUCE source file may be *read-in* by the command:

```
lisp ('in ["path/filename.red"]) ;
```

Plotting

Using GnuPlot

```
> reduce::load "gnuplot";
```

Note that we have to quote the x..y ranges here so that they get through to Reduce, rather than being evaluated on the Pure side.

```
> simplify $ 'plot (sin x/x) (x==(-15..15));

// Multiple ranges.
> simplify $ 'plot (sin(x^2 + y^2) / sqrt(x^2 + y^2))
[x==(-12 .. 12), y==(-12 .. 12)];

// Specifying options.
> simplify $ 'plot (cos (sqrt(x^2 + y^2))) [x==(-3 .. 3),y==(-3 .. 3)] hidde

// Specifying points.
> simplify $ plot [[0,0],[0,1],[1,1],[0,0],[1,0],[0,1],[0.5,1.5],[1,1],[1,0]

// Output options.
> simplify $ plot (sin x) [x=='(0 .. 10),terminal==postscript,output=="sin.p
```

References:

[REDUM]	(1, 2) *REDUCE User's Manual*, Version 3.8, Anthony C. Hearn, Santa Monica, CA, USA.
[LNCS102]	*On the Integration of Algebraic Functions*, LNCS 102, Springer Verlag, 1981.
[SYMSAC81]	P. M. A. Moore and A.C. Norman, *Implementing a Polynomial Factorization and GCD Package*, Proc. SYMSAC '81, ACM (New York) (1981), 109-116.

todo -> replace python by pure