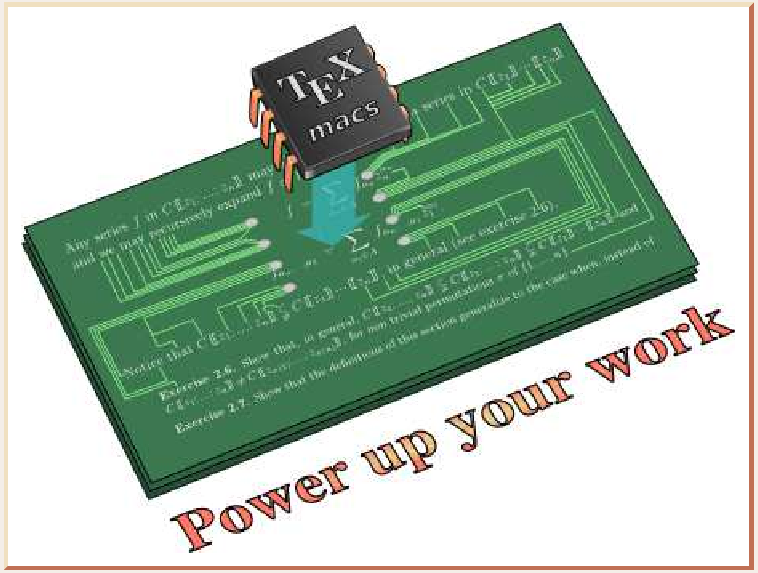
Introduction to Mathemagix

Joris van der Hoeven, Grégoire Lecerf, Bernard Mourrain, ...



Luminy 2010
http://www.TeXmacs.org
http://www.   .org

Overview

- **Aims**
  - Strongly typed language with a compiler.
  - Access to low level details and powerful abstractions.
  - Well integrated with other languages, such as C++.
  - Libraries for computer algebra and computer analysis.
  - User friendly interface based on GNU TeXmacs.

- **Current status**
  - Interpreter with inperfect typing.
  - Experimental compiler under development.
  - Efficient C++ libraries for computer algebra and analysis.

Running the interpreter

```
Mmx] 1 + 1
Mmx] "Hello" >< "There"
Mmx] for i in 1 to 5 do mmout << i << " -> " << i*i << "\n";
Mmx] type_mode? := true;
Mmx] 1 + 1
Mmx] use "algebramix";
Mmx] 1 + 1
Mmx] z == series (0, 1)
Mmx] sin z
Mmx] help series
Mmx] help Series Rational
Mmx]
```

Running the compiler

⇑ Hello world

```
Shell] mmc
Shell] cat hello.mmx
Shell] mmc hello.mmx
Shell] ./hello
Shell]
```

⇑ Separate compilation

```
Shell] mmc --clean-cache
Shell] cat slave.mmx
Shell] cat master.mmx
Shell] mmc --verbose master.mmx
Shell] ./master
Shell] mmc --verbose master.mmx
Shell] touch slave.mmx
Shell] touch master.mmx
Shell]
```

Atomic types

⇑ Booleans

```
Mmx] type_mode? := true;
```

```
Mmx] 2 = 2
```

```
Mmx]
```

⇑ Strings

```
Mmx] "ho" >< "ho"
```

```
Mmx] replace ("hoho", "ho", "haha")
```

```
Mmx]
```

⇑ Syntactic types

```
Mmx] 'x
```

```
Mmx] '(f (x, y, z))
```

```
Mmx] '(f (x, y, z)) [2]
```

```
Mmx] $document ("Some ", $with ("color", "red", "red"), " text.";
                "Pythagoras said ", $math ('(a^2 + b^2 = c^2)), ".")
```

```
Mmx] mmout << $document ("More ", $strong ("very important"), " stuff.") << "\n";
```

```
Mmx]
```

⇑ Ports

```
Mmx] mmout
Mmx] mmout << "Hello\n";
Mmx] output_file_port ("toto.txt") << "Hi there\n";
Mmx] load ("toto.txt")
Mmx]
```

Numeric types

⇑ Integers

```
Mmx] use "numerix";
Mmx] 100!
Mmx]
```

Rational numbers

```
Mmx] 1/1 + 1/2 + 1/3 + 1/4 + 1/5 + 1/6 + 1/7 + 1/8 + 1/9 + 1/10
Mmx]
```

Floating point numbers

```
Mmx] exp 1.0
Mmx] bit_precision := 128;
Mmx] exp 1.0
Mmx] significant_digits := 5
Mmx] exp 1.0
Mmx] exp 1.0000000000000000000000001 - exp 1.0
Mmx] significant_digits := 0
Mmx]
```

Complex numbers

```
Mmx] complex (1, 2/3)
Mmx] square complex (1, 2/3)
Mmx]
```

Tangent numbers

```
Mmx] tangent (1, 2)

Mmx] tangent (1, 2) * tangent (1, 2)

Mmx]
```

Balls

```
Mmx] b == ball (1.0, 0.00000000001)

Mmx] 1.0001 * b - b

Mmx]
```

Compound types and syntactic sugar

⇑ Vectors

```
Mmx] use "algebramix";
Mmx] v == [1, 2, 3, 4, 5, 6]
Mmx] v >< map (square, v)
Mmx] v*v + 3*v + 2
Mmx] #v
Mmx] v[1]
Mmx] v[2,4]
Mmx]
```

⇑ Iterators and syntactic sugar

```
Mmx] 1 to 10
Mmx] [ 1 to 10 ]
Mmx] [ 0..10 ]
Mmx] [ k^3 | k in 1 to 10 ]
Mmx] [ k^3 | k in 1 to 10, k mod 3 = 1 ]
Mmx]
```

⇑ Matrices

```
Mmx] [ 1, 2; 3, 4 ]
Mmx] M == [ 1 / (i+j+1) | j in 0..4 || i in 0..4 ]
Mmx] invert M
Mmx]
```

⇑ Tables

```
Mmx] t := table (0)
Mmx] t['x]
Mmx] t['x] := "Hallo";
Mmx] t
Mmx] t['x]
Mmx] t[polynomial(1,1)^10] := "Boeh";
Mmx] t
Mmx]
```

Simple declarations

⇑ Constant and mutable variables

```
Mmx] cst: Integer == 111111111^2
Mmx] mut: Integer := 1010101010101010101010101010101^2
Mmx] cst := square cst
Mmx] mut := square mut
Mmx]
```

⇑ Function definitions and overloading

```
Mmx] cube (x: Integer): Integer == x*x*x;
```

```
Mmx] cube (s: String): String == s >< s >< s;
```

```
Mmx] cube 1001001001001001
```

```
Mmx] cube "haha"
```

```
Mmx]
```

⇑ The generic type

Disclaimer: special arithmetic on generic objects only available in interpreter.

```
Mmx] haha (ha) == ha * ha;
```

```
Mmx] haha 1111
```

```
Mmx] infix * (s1: String, s2: String): String == s1 >< s2;
```

```
Mmx] haha "ahah"
```

```
Mmx] haha (ha: Integer) == ha * ha * ha * ha;
```

```
Mmx] haha 2
```

```
Mmx] haha "ohoh"
```

```
Mmx]
```

⇑ Functional programming

```
Mmx] fib (n: Integer): Integer ==
       if n <= 1 then 1 else fib (n-1) + fib (n-2);
Mmx] [ fib n | n in 1 to 20 ]
Mmx] shift (x: Integer) (y: Integer): Integer == x + y;
Mmx] shift 3
Mmx] (shift 3) 4
Mmx] map (shift 3, [ 1 to 20 ])
Mmx]
```

Class definitions

```
Mmx] use "numerix"; type_mode? := true; significant_digits := 5;

Mmx] class Color == {
       mutable { r: Floating; g: Floating; b: Floating; }
       constructor grey (x: Floating) == {
         r == x; g == x; b == x; }
       constructor rgb (r2: Floating, g2: Floating, b2: Floating) == {
         r == r2; g == g2; b == b2; }
     }

Mmx] rgb (1, 0, 0)

Mmx] flatten (c: Color): Syntactic ==
       syntactic ('rgb (as_generic flatten c.r,
                        as_generic flatten c.g,
                        as_generic flatten c.b));

Mmx] rgb (1, 0, 0)

Mmx] mix (c1: Color, c2: Color, a: Floating): Color ==
       rgb (a * c1.r + (1-a) * c2.r,
            a * c1.g + (1-a) * c2.g,
            a * c1.b + (1-a) * c2.b);

Mmx] mix (rgb (1, 0, 0), grey (0.5), 0.5)

Mmx]
```

Implicit type conversion and inheritance

```
Mmx] upgrade (x: Floating): Color == grey x;
Mmx] mix (1.0, rgb (0, 1, 0), 0.2)
Mmx] mix (1, rgb (0, 1, 0), 0.4)
Mmx] class Alpha_color == {
       mutable { c: Color; a: Floating; }
       constructor alpha_color (c2: Color, a2: Floating) == { c == c2; a == a2; }
     };
Mmx] alpha_color (0, 0.5)
Mmx] downgrade (ac: Alpha_color): Color == mix (ac.c, 1, ac.a);
Mmx] alpha_color (0, 0.5) :> Color
Mmx] postfix .greyed (c: Color): Color == (c.r + c.g + c.b) / 3;
Mmx] rgb (1, 0, 0).greyed
Mmx]
```

Control structures

⇑ Loops

```
Mmx] for i in [ 1, 4, 7] for j in [ 3, 5, 11 ] do
       mmout << i << ", " << j << " -> " << i * j << "\n";
```

```
Mmx] for i: Int in 0..100 until i*i >= 5000 do {
       if i mod 11 <= 8 then continue;
       mmout << "i= " << i << "\n";
     }
```

```
Mmx]
```

⇑ Exceptions

```
Mmx] risky (x: Rational): Rational == {
       if x = 5 then raise "not in domain";
       return 1 / (x - 5);
     }
     try {
       for i in 1 to 10 do mmout << i << " -> " << risky i << "\n";
       catch (err: String) { mmout << "error: " << err << "\n"; }
     };
```

```
Mmx]
```

Categories

```
Shell] mmc --verbose --keep ring.mmx
Shell] ./ring
Shell]
```

Functional programming

```
Shell] mmc --verbose --keep recursive.mmx
```

```
Shell] ./recursive
```

```
Shell]
```

Gluing C++ functionality

```
Shell] mmc --verbose syracuse.mmx
Shell] ./syracuse
Shell]
```