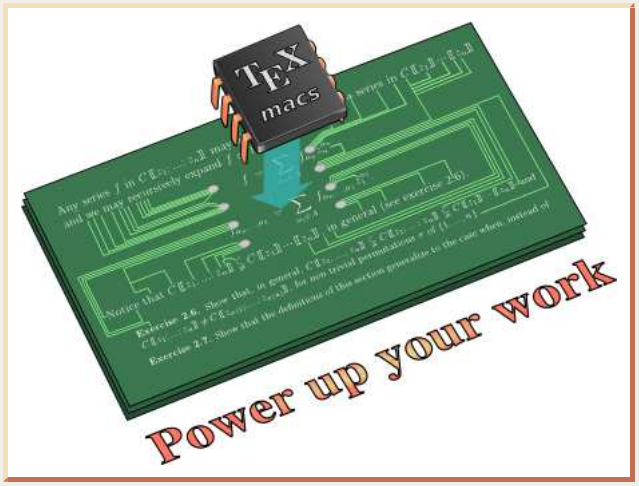
Mathemagix I

Introduction and Language

J. v.d. Hoeven, G. Lecerf, B. Mourrain, O. Ruatta et al. (ANR Gecko)



Gecko/Tera, École Polytechnique, 2008

http://www.mathemagix.org

http://www.TEXMACS.org

Current version of Mathemagix

- Installation
  - Download from `http://www.mathemagix.org`.
  - Compile the SVN version on Gforge.
  - Binary installers under construction (Linux, MacOS, MinGW).
- Programming language
  - Interpreter `mmx-light`.
  - Prototype of a compilateur `mmc`, written in Mathemagix.
  - Mecanism for « gluing » external C/C++ librairies.
- Suite of C++ packages (former MMXLIB + SYNAPS)
  - Fast arithmetic for basic dense objects.
  - Fast arithmetic for approximate objects.
  - Analytic functions, transseries, basic symbolic computations.
  - Solvers.
- Interfaces
  - Textual (shell, emacs, automatic completion, syntactic highlighting).
  - Graphical (GNU TEXMACS).
  - 3D modeler 3D (Axel).

Introductory examples

```
Mmx] use "algebramix"
Mmx] z: Series Rational == series (0, 1);
Mmx] f == exp (exp z - 1)
Mmx] f[500] * 500!
Mmx] M == [ 1/(i+j-1) | i in 1 to 3 || j in 1 to 3 ]
Mmx] invert M
Mmx] fib (n: Integer): Integer ==
       if n <= 1 then 1
       else fib (n-1) + fib (n-2);
Mmx] [ fib n | n in 0..20 ]
Mmx]
```

Language requirements

Natural notations

- Functional (objects $x \mapsto x^2$)
- Polymorphism ($+ : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$, $+ : \mathbb{Q} \times \mathbb{Q} \to \mathbb{Q}$, etc.)
- Implicit conversions ($\mathbb{Z} \subseteq \mathbb{Q} \subseteq \mathbb{Q}[x] \subseteq \mathbb{Q}[[x]]$)

Genericity

- Categories (signatures)
- Generic types $(\mathrm{object}(R : \mathrm{Ring}, x : R))$
- Symbolic types

General

- Compiled
- Modular (large scale programming)
- Cooperative (glue for existing systems)

State of the art

Computer algebra systems

- Maple: interpreted, weak types, recently functional
- Mathematica: interpreted, pattern matching
- Axiom/Aldor: strongly typed, functional, categories, polymorphic

General purpose languages

- Lisp: weakly typed, functional
- C++: moderately typed, not functional
- Ocaml: strongly typed, functional, signatures, not polymorphic

The type system by examples

⇑ Overloading

```
Mmx] f (x: Int): Int == x * x;
```

```
Mmx] f (x: String): String == "Hello " >< x;
```

```
Mmx] f (11111)
```

```
Mmx] f ("Marc")
```

```
Mmx]
```

⇑ Templates

```
Mmx] forall (T: Class)
     print (x: T): Void == mmout << x << "\n";
```

```
Mmx] print "Hello Marc";
```

```
Mmx] category Ring == {
       convert: Integer -> This;
       prefix -: This -> This;
       infix +: (This, This) -> This;
       infix *: (This, This) -> This;
     }
```

```
Mmx] forall (R: Ring)
     square (x: R): R == x * x;
```

```
Mmx] square 3
```

```
Mmx]
```

⇑ Genericity

```
Mmx] forall (K: Field) exists (L: Extension_Field K)
     roots (P: Polynomial K): Vector L;
```

General philosophy

Theorem	Specification
Proof	Implementation

Typing ambiguities

assertion in first order language

$x{:}T \iff x$ can be regarded as an instance of type T

$$
\begin{aligned}
x{:}T\ U &\iff x{:}T \wedge x{:}U \\
x{:}T\ U &\iff x{:}T \vee x{:}U \\
x{:}(\ :\ )\ T_\lambda &\iff (\forall \lambda{:}\Lambda)\, x{:}T_\lambda \\
x{:}(\ :\ )\ T_\lambda &\iff (\exists \lambda{:}\Lambda)\, x{:}T_\lambda \\
x{:}\ T &\iff C \Rightarrow x{:}T
\end{aligned}
$$

$$
\begin{aligned}
\text{Axiom} &\longleftrightarrow \text{Implicit conversion rule} \\
T\ U &\iff \forall x, x{:}T \Rightarrow x{:}U
\end{aligned}
$$

Implicit conversions

One shot converters or casters

- $\text{Double} \rightsquigarrow \text{Floating}$
- $\text{Point} \rightsquigarrow \text{Vector}(\text{Floating})$

Upgraders

- $T \overset{\text{up}}{\rightsquigarrow} U \iff \forall X, X \rightsquigarrow T \Rightarrow X \rightsquigarrow U$
- $\text{Integer} \overset{\text{up}}{\rightsquigarrow} \text{Rational}$
- $\forall X\colon \text{Ring}, X \overset{\text{up}}{\rightsquigarrow} \text{Complex}(X)$

Downgraders

- $T \overset{\text{down}}{\rightsquigarrow} U \iff \forall X, U \rightsquigarrow X \Rightarrow T \rightsquigarrow X$
- $\text{Colored\_point} \overset{\text{down}}{\rightsquigarrow} \text{Point}$

$$\text{Colored\_point} \overset{\text{down}}{\rightsquigarrow} \text{Point} \rightsquigarrow \text{Vector}(\text{Floating}) \overset{\text{up}}{\rightsquigarrow} \text{Complex}(\text{Vector}(\text{Floating}))$$

Implicit conversions

One shot converters or casters

- $\text{Double} \rightsquigarrow \text{Floating}$
- $\text{Point} \rightsquigarrow \text{Vector}(\text{Floating})$

Upgraders

- $T \overset{\text{up}}{\rightsquigarrow} U \iff \forall X, X \rightsquigarrow T \Rightarrow X \rightsquigarrow U$
- $\text{Integer} \overset{\text{up}}{\rightsquigarrow} \text{Rational}$
- $\forall X\colon \text{Ring}, X \overset{\text{up}}{\rightsquigarrow} \text{Complex}(X)$

Downgraders

- $T \overset{\text{down}}{\rightsquigarrow} U \iff \forall X, U \rightsquigarrow X \Rightarrow T \rightsquigarrow X$
- $\text{Colored\_point} \overset{\text{down}}{\rightsquigarrow} \text{Point}$

$$\begin{array}{ccccccccccccccc} T_1 & \overset{\text{down}}{\rightsquigarrow} & T_2 & \overset{\text{down}}{\rightsquigarrow} & \cdots & \overset{\text{down}}{\rightsquigarrow} & T_t & \rightsquigarrow & U_u & \overset{\text{up}}{\rightsquigarrow} & \cdots & \overset{\text{up}}{\rightsquigarrow} & U_2 & \overset{\text{up}}{\rightsquigarrow} & U_1 \\ h_{T_1} & \geqslant & h_{T_2} & \geqslant & \cdots & \geqslant & h_{T_t} & & h_{U_u} & \leqslant & \cdots & \leqslant & h_{U_2} & \leqslant & h_{U_1} \end{array}$$

Resolution of ambiguities

- **First come first served**

```
v: Vector Rational == square ([1, 2, 3]);
```

$$\begin{array}{ccc} [1,2,3]\colon \mathrm{Vector}(\mathbb{Z}) & \longrightarrow & [1,4,9]\colon \mathrm{Vector}(\mathbb{Z}) \\ \downarrow & & \downarrow \\ [1,2,3]\colon \mathrm{Vector}(\mathbb{Q}) & \longrightarrow & [1,4,9]\colon \mathrm{Vector}(\mathbb{Q}) \end{array}$$

- **Affirmative action**

```
forall (R: Ring) square (x: R): R == x * x;
square (x: Boolean): Boolean == x;
b: Boolean == true;
mmout << square b << "\n";
```

- **Work less and earn more**

```
postfix []: (Vector R, Int): R;
assume (access) postfix []: (Alias Vector R, Int): Alias R;
v: Vector R == [ x, y, z ];
mmout << v[1] << "\n";
```

- **User king, not emperor**

```
mmout << polynomial (1, 2, 3) + complex (2, 3) << "\n";
```

Resolution of ambiguities

- **First come first served**

```
v: Vector Rational == square ([1, 2, 3]);
```

$$\begin{array}{ccc} [1,2,3]\colon \mathrm{Vector}(\mathbb{Z}) & \longrightarrow & [1,4,9]\colon \mathrm{Vector}(\mathbb{Z}) \\ \downarrow & & \downarrow \\ [1,2,3]\colon \mathrm{Vector}(\mathbb{Q}) & \longrightarrow & [1,4,9]\colon \mathrm{Vector}(\mathbb{Q}) \end{array}$$

- **Affirmative action**

```
forall (R: Ring) square (x: R): R == x * x;
square (x: Boolean): Boolean == x;
b: Boolean == true;
mmout << square b << "\n";
```

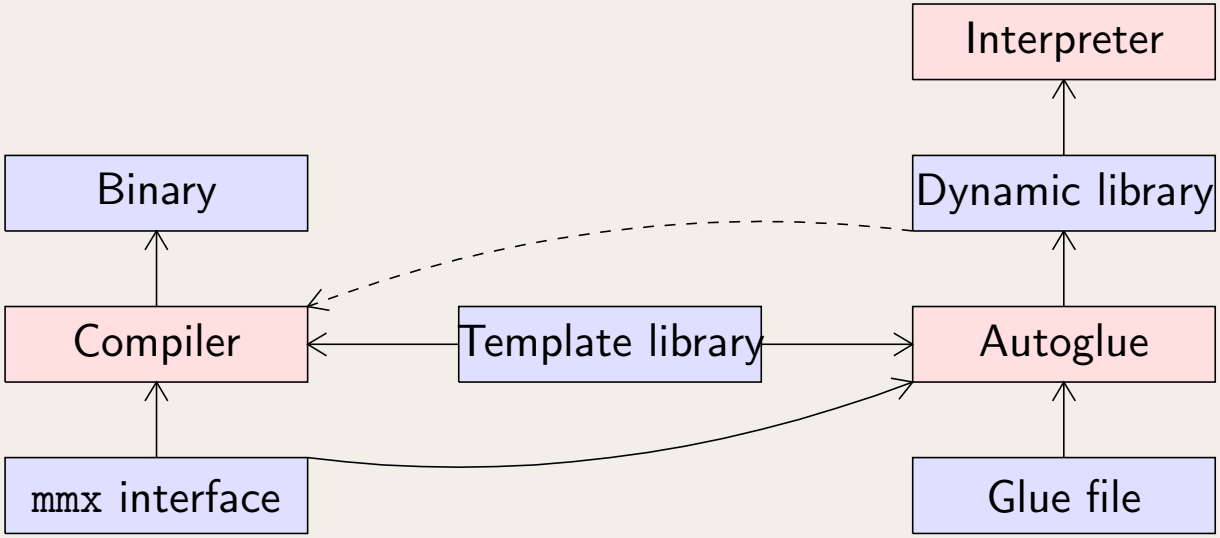
- **Work less and earn more**

```
postfix []: (Vector R, Int): R;
assume (access) postfix []: (Alias Vector R, Int): Alias R;
v: Vector R == [ x, y, z ];
mmout << v[1] << "\n";
```

- **User king, not emperor**

```
mmout << polynomial (1, 2, 3) + complex (2, 3) << "\n";
```

Gluing external C/C++ libraries



Gluing external C/C++ libraries

```
foreign cpp import {
  cpp_flags "`algebramix-config --cppflags`";
  ...

  forall (C: Ring) {
    class Polynomial C == polynomial C;

    polynomial: Tuple C -> Polynomial C == polynomial C;
    deg: Polynomial C -> Int == deg;
    postfix []: (Polynomial C, Int) -> C == postfix [];

    prefix -: Polynomial C -> Polynomial C == prefix -;
    ...
  }
}
```

```
require "algebramix/glue_vector_rational.mmx";
require "algebramix/glue_polynomial_generic.mmx";
specialize Polynomial Rational;
specialize Polynomial Complex Rational;
```