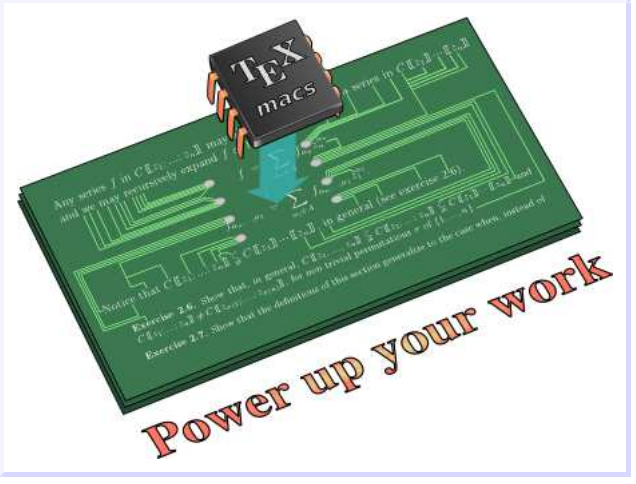
Mathemagix



Nice 2007

http://www.mathemagix.org

http://www.TEXMACS.org

Motivations et objectifs

- Langage de programmation orienté vers les mathématiques
  - Langage fonctionnel fortement typé avec syntaxe agréable.
  - Manipulation simple et intégrée d'objets symboliques.
  - Possibilité de compilation $\rightsquigarrow$ bonnes performances.
  - Contrôle sur le langage.
- Intégration avec des bibliothèques/systèmes externes.
  - Mécanisme de « colle ».
  - Langage de colle.
  - Conventions communes pour la sémantique ; catégories.
  - Interfaces, documentation, etc.
- Système de calcul analytique ; MMXLIB + SYNAPS
  - Arithmétique rapide de base pour des objets denses.
  - Arithmétique rapide pour des objets approchés.
  - Fonctions analytiques effectives.
  - Solveurs.

- Langages sans surcharge : ML, etc.

$$f(x) \equiv x + 1$$

Typage automatique $f : \mathrm{Int} \longrightarrow \mathrm{Int}$

$$f(x) \equiv x +_{\mathrm{Float}} 1.0$$

- Langages avec surcharge : C++, Aldor, etc.

$$\begin{array}{rcl} + & : & \mathrm{Int}, \mathrm{Int} \longrightarrow \mathrm{Int} \\ + & : & \mathrm{Float}, \mathrm{Float} \longrightarrow \mathrm{Float} \end{array}$$

Déclarations typées :

$$f(x : \mathrm{Int}) : \mathrm{Int} \equiv x + 1$$

- Modèles sans vérification : C++, etc.

```
template<typename C> vector<C>
operator + (const vector<C>& v, const vector<C>& w) {
  int i, n= min (dimension (v), dimension (w));
  vector<C> r (n)
  for (i=0; i<n; i++) r[i]= v[i] + w[i];
  return r;
}
```

- Catégories : Aldor, etc.

```
Ring: Category == with {
  +: (%, %) -> %;
  ...
}

Vector (R: Ring): Ring == add {
  (x: %) + (y: %): % == {
    import R;
    ...
  }
  ...
}
```

- Desavantages
  - Importation implicite disagréable.
  - Demande structuration au préalable en catégories.
  - Routines doivent être attachées à des catégories.

- Déclarations sous hypothèses : Mathemagix

```
Ring: Category == category {
  +: (%, %) -> %;
  ...
}

forall (R: Ring)
infix + (v: Vector R, w: Vector R): Vector R == {
  ...
}
```

Typer les ambiguïtés

- Surcharge $\wedge$

  $+ : (\text{Int}, \text{Int}) \longrightarrow \text{Int} \ \wedge \ (\text{Float}, \text{Float}) \longrightarrow \text{Float}$
- Unions $\vee$

  $\text{Failable}(T) \equiv T \vee \{\text{false}\}$
- Modèles $\forall$

  $\text{print} : \forall T, (\text{Printer}, T) \longrightarrow \text{Void}$
- Généricité $\exists$

  $\text{Generic} \equiv \exists T, T$
- Types conditionnels $\Rightarrow$

  $\times : \forall T, T : \text{Ring} \Rightarrow (\text{Vector}\, T, \text{Vector}\, T) \to \text{Vector}\, T$

Conversions implicites

- Transitivité

  Caster. Pas de transitivité.

  Upgrader. À droite : $A \xrightarrow{\text{up}} B, B \longrightarrow C \Rightarrow A \longrightarrow C$ constructeur

  Downgrader. À gauche : $A \longrightarrow B, B \xrightarrow{\text{down}} C \Rightarrow A \longrightarrow C$ héritage

  Converter. Les deux.

- Priorité

  Inclustions. $\text{Integer} \longrightarrow \text{Rational}$

  Homomorphismes. $\text{Integer} \longrightarrow \text{Modular\_integer}(p)$

  Caster. $\text{Integer} \longrightarrow \text{Float}$

  Abstraction. $\text{Integer} \longrightarrow \text{Generic}$

  $\forall T, T \longrightarrow \text{Generic}$ et $\forall T, T \longrightarrow \text{Vector}(T)$

- Décidabilité

$$A_1 \longrightarrow \cdots \longrightarrow A_n \longrightarrow B_n \longrightarrow \cdots \longrightarrow B_0$$

Spécialisations

- Préférence des types les plus spécialisés

```
forall (T: Type) f (x: T): Void == ...;
f (x: Rational): Void == ...;
f (x: Integer): Void == ...;

f(3);
```

- Priorités

```
(v: Vector T) [i: Integer]: T == ...;
(v: Alias Vector T) [i: Integer]: Alias T == ...;

v: Vector T := ...;
mmout << v[3] << "\n";
v[4] := x;
```

- Compatibilité entre Generic et l'environnement

```
x: Generic == 2;
f (x: Integer): Integer == ...;
f(x);
```

- Dispatching automatique

```
(x: dispatch Generic) + (y: dispatch Generic): Generic;
(x: Integer) + (y: Integer): Integer;
(x: Rational) + (y: Rational): Rational;
```

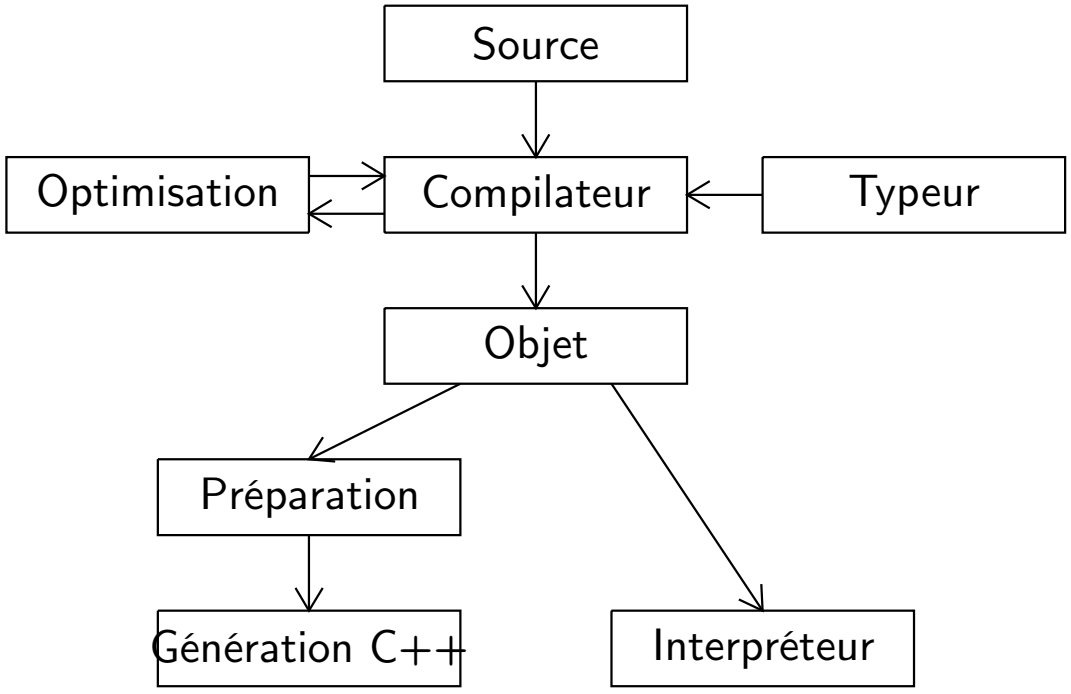
Problèmes de scope ?

- Downgrade automatique

```
2 * ((3 :> Generic) / 2)
```

Spécialisation

```
cast: Rational -> Generic
```


Source
Optimisation
Compilateur
Typeur
Objet
Préparation
Génération C++
Interpréteur