MuPAD-Combinat

Implementing Algebraic Combinatorics
Some feedback from the development of MuPAD-Combinat

Nicolas M. Thiéry and al.

Laboratoire de Mathématiques d'Orsay, Université Paris Sud

April 27-th 2006

mupad-combinat-devel@lists.sf.net

http://mupad-combinat.sf.net/

MuPAD-Combinat figures

- 8 developers, 20 contributers
- 15+ research articles
- Official MuPAD combinat library since 2002
- 5 years, 9 official releases, 4 stable ones
- GNU/Linux, MacOS X, Windows, Zaurus
- 76000 lines of MuPAD + 15000 lines of C++
- 20000 lines of tests, 575 pages of doc
- In 2005: 1500 messages on the mailing list, 5000 visits of the web page and 400 downloads.
- Integrated software: μ-EC, CS, PerMuVAR, Symmetrica, Ircalc, Nauty, rigged configuration kernel
- How many users?

Why am I here?

Feasibility of Axiom-Combinat?

- Users asked for it
- Sympathy to an open source project

MuPAD-Combinat community?

- Future of MuPAD?
- Quality of the Axiom/Aldor language
- Improving the MuPAD-Combinat design
- Fostering cross pollination
  (code, tests, documentation, expertise, design, interface)

Why am I here?

Feasibility of Axiom-Combinat?

- Users asked for it
- Sympathy to an open source project

MuPAD-Combinat community?

- Future of MuPAD?
- Quality of the Axiom/Aldor language
- Improving the MuPAD-Combinat design
- Fostering cross pollination
  (code, tests, documentation, expertise, design, interface)

Motivations: a Loday-Ronco calculator

Our needs

- Tools:
  - Manipulation of all sorts of combinatorial objects (counting, generation, random generation, ranking, ...)
  - Computations in combinatorial algebraic structures (Hopf algebras, modules, operads, ...)
  - Standard CAS stuff (solvers, ...)
- Rapid development of calculators for computer exploration:
  - Interpreter (compiler is a bonus)
  - Expressive and natural syntax
  - Generic tools
- Keep time to do maths!
  - Code sharing

Steps to build a *-Combinat package

1. Experiment / Design / Experiment / Design
2. Support tools:
   - Counting functions
     (lazy Karatsuba product, plethysm, implicit equation)
   - Generators, continuations
   - Data structures for combinatorial objects
     (partitions, trees, tableaux, permutations, graphs, ...)
3. Combinatorial class lego:
   - Basic combinatorial classes (finite classes, integers, ...)
   - Functors (union, product, graded products, multisets, image class, implicit equations)
4. Other generic tools:
   - Linear extensions of a poset
   - Lexicographic enumeration of list of integers
   - Integral points of a polyhedron

Steps to build a *-Combinat package

1. Experiment / Design / Experiment / Design
2. Support tools:
   - Counting functions
     (lazy Karatsuba product, plethysm, implicit equation)
   - Generators, continuations
   - Data structures for combinatorial objects
     (partitions, trees, tableaux, permutations, graphs, ...)
3. Combinatorial class lego:
   - Basic combinatorial classes (finite classes, integers, ...)
   - Functors (union, product, graded products, multisets, image class, implicit equations)
4. Other generic tools:
   - Linear extensions of a poset
   - Lexicographic enumeration of list of integers
   - Integral points of a polyhedron

Steps to build a *-Combinat package

1. Experiment / Design / Experiment / Design
2. Support tools:
   - Counting functions
     (lazy Karatsuba product, plethysm, implicit equation)
   - Generators, continuations
   - Data structures for combinatorial objects
     (partitions, trees, tableaux, permutations, graphs, ...)
3. Combinatorial class lego:
   - Basic combinatorial classes (finite classes, integers, ...)
   - Functors (union, product, graded products, multisets, image class, implicit equations)
4. Other generic tools:
   - Linear extensions of a poset
   - Lexicographic enumeration of list of integers
   - Integral points of a polyhedron

Steps to build a *-Combinat package

1. Experiment / Design / Experiment / Design
2. Support tools:
   - Counting functions
     (lazy Karatsuba product, plethysm, implicit equation)
   - Generators, continuations
   - Data structures for combinatorial objects
     (partitions, trees, tableaux, permutations, graphs, ...)
3. Combinatorial class lego:
   - Basic combinatorial classes (finite classes, integers, ...)
   - Functors (union, product, graded products, multisets, image class, implicit equations)
4. Other generic tools:
   - Linear extensions of a poset
   - Lexicographic enumeration of list of integers
   - Integral points of a polyhedron

Steps to build a *-Combinat package

1. `FreeModule(Combinatorial Class, Coefficient Ring)`
   Categories: AlgebraWithBasis and friends
   Support for seamless linear algebra
   Unification with polynomials, ...
2. Modules with several bases, ...
3. Functors: tensor product, tensor, exterior, and symmetric algebra, submodules, quotients
4. Generic Gröbner/Involutive elimination tools
5. Permutation groups with basic Shreier-Simms algorithms

Steps to build a *-Combinat package

1. `FreeModule(Combinatorial Class, Coefficient Ring)`
   Categories: AlgebraWithBasis and friends
   Support for seamless linear algebra
   Unification with polynomials, ...
2. Modules with several bases, ...
3. Functors: tensor product, tensor, exterior, and symmetric algebra, submodules, quotients
4. Generic Gröbner/Involutive elimination tools
5. Permutation groups with basic Shreier-Simms algorithms

Steps to build a *-Combinat package

1. `FreeModule(Combinatorial Class, Coefficient Ring)`
   Categories: AlgebraWithBasis and friends
   Support for seamless linear algebra
   Unification with polynomials, ...
2. Modules with several bases, ...
3. Functors: tensor product, tensor, exterior, and symmetric algebra, submodules, quotients
4. Generic Gröbner/Involutive elimination tools
5. Permutation groups with basic Shreier-Simms algorithms

Steps to build a *-Combinat package

1. `FreeModule(Combinatorial Class, Coefficient Ring)`
   Categories: AlgebraWithBasis and friends
   Support for seamless linear algebra
   Unification with polynomials, ...
2. Modules with several bases, ...
3. Functors: tensor product, tensor, exterior, and symmetric algebra, submodules, quotients
4. Generic Gröbner/Involutive elimination tools
5. Permutation groups with basic Shreier-Simms algorithms

Steps to build a *-Combinat package

1. `FreeModule(Combinatorial Class, Coefficient Ring)`
   Categories: AlgebraWithBasis and friends
   Support for seamless linear algebra
   Unification with polynomials, ...
2. Modules with several bases, ...
3. Functors: tensor product, tensor, exterior, and symmetric algebra, submodules, quotients
4. Generic Gröbner/Involutive elimination tools
5. Permutation groups with basic Shreier-Simms algorithms

How to model a combinatorial class?

Examples:

- List Integer l := [1, 5, 9];
- $\{x^2,\ x \in 1, \ldots, 10\}$
- Integer, Odd Integer
- Integer partitions, trees
- Finite field $\mathbb{F}_2$
- Permutation group $G \subset \mathcal{S}_4$

Clear separation:

- Data structure and operations on the elements of the set
- Data structure and operations on the set

Two questions for a combinatorial class C:

- Should C be represented by an object or by a domain?
- If $x \in C$, should the type of x be C?

How to model a combinatorial class?

Examples:

- List Integer l := [1, 5, 9];
- $\{x^2,\ x \in 1, \ldots, 10\}$
- Integer, Odd Integer
- Integer partitions, trees
- Finite field $\mathbb{F}_2$
- Permutation group $G \subset \mathcal{S}_4$

Clear separation:

- Data structure and operations on the elements of the set
- Data structure and operations on the set

Two questions for a combinatorial class C:

- Should C be represented by an object or by a domain?
- If $x \in C$, should the type of x be C?

How to model a combinatorial class?

Examples:

- List Integer I := [1, 5, 9];
- $\{x^2,\ x \in 1, \ldots, 10\}$
- Integer, Odd Integer
- Integer partitions, trees
- Finite field $\mathbb{F}_2$
- Permutation group $G \subset \mathcal{S}_4$

Clear separation:

- Data structure and operations on the elements of the set
- Data structure and operations on the set

Two questions for a combinatorial class C:

- Should C be represented by an object or by a domain?
- If $x \in C$, should the type of x be C?

How to model a combinatorial class?

Examples:

- List Integer I := [1, 5, 9];
- $\{x^2,\ x \in 1, \ldots, 10\}$
- Integer, Odd Integer
- Integer partitions, trees
- Finite field $\mathbb{F}_2$
- Permutation group $G \subset \mathcal{S}_4$

Clear separation:

- Data structure and operations on the elements of the set
- Data structure and operations on the set

Two questions for a combinatorial class C:

- Should C be represented by an object or by a domain?
- If $x \in C$, should the type of x be C?

How to model a combinatorial class?

Examples:

- List Integer l := [1, 5, 9];
- $\{x^2,\ x \in 1, \ldots, 10\}$
- Integer, Odd Integer
- Integer partitions, trees
- Finite field $\mathbb{F}_2$
- Permutation group $G \subset \mathcal{S}_4$

Clear separation:

- Data structure and operations on the elements of the set
- Data structure and operations on the set

Two questions for a combinatorial class C:

- Should C be represented by an object or by a domain?
- If $x \in C$, should the type of x be C?

How to model a combinatorial class?

Examples:

- List Integer I := [1, 5, 9];
- $\{x^2,\ x \in 1, \ldots, 10\}$
- Integer, Odd Integer
- Integer partitions, trees
- Finite field $\mathbb{F}_2$
- Permutation group $G \subset \mathcal{S}_4$

Clear separation:

- Data structure and operations on the elements of the set
- Data structure and operations on the set

Two questions for a combinatorial class C:

- Should C be represented by an object or by a domain?
- If $x \in C$, should the type of x be C?

How to model a combinatorial class?

Examples:

- List Integer I := [1, 5, 9];
- $\{x^2,\ x \in 1, \ldots, 10\}$
- Integer, Odd Integer
- Integer partitions, trees
- Finite field $\mathbb{F}_2$
- Permutation group $G \subset \mathcal{S}_4$

Clear separation:

- Data structure and operations on the elements of the set
- Data structure and operations on the set

Two questions for a combinatorial class C:

- Should C be represented by an object or by a domain?
- If $x \in C$, should the type of x be C?

How to model a combinatorial class?

Examples:

- List Integer I := [1, 5, 9];
- $\{x^2,\ x \in 1, \ldots, 10\}$
- Integer, Odd Integer
- Integer partitions, trees
- Finite field $\mathbb{F}_2$
- Permutation group $G \subset \mathcal{S}_4$

Clear separation:

- Data structure and operations on the elements of the set
- Data structure and operations on the set

Two questions for a combinatorial class C:

- Should C be represented by an object or by a domain?
- If $x \in C$, should the type of x be C?

How to model a combinatorial class?

Examples:

- List Integer I := [1, 5, 9];
- $\{x^2,\ x \in 1, \ldots, 10\}$
- Integer, Odd Integer
- Integer partitions, trees
- Finite field $\mathbb{F}_2$
- Permutation group $G \subset \mathcal{S}_4$

Clear separation:

- Data structure and operations on the elements of the set
- Data structure and operations on the set

Two questions for a combinatorial class C:

- Should C be represented by an object or by a domain?
- If $x \in C$, should the type of x be C?

How to model a combinatorial class?

Examples:

- List Integer I := [1, 5, 9];
- $\{x^2,\ x \in 1, \ldots, 10\}$
- Integer, Odd Integer
- Integer partitions, trees
- Finite field $\mathbb{F}_2$
- Permutation group $G \subset \mathcal{S}_4$

Clear separation:

- Data structure and operations on the elements of the set
- Data structure and operations on the set

Two questions for a combinatorial class C:

- Should C be represented by an object or by a domain?
- If $x \in C$, should the type of x be C?

Decomposable combinatorial classes / Species

```
combinat::decomposableObjects([
    Tree = Union(Leaf, NonTrivialTree),
    NonTrivialTree = Product(Label, Childs),
    Childs = Product(Tree, Tree)]):
```

```
NonTrivialTree:  CombinatorialClass;
Tree := Union (Leaf, NonTrivialTree);
NonTrivialTree := Product(Label, Childs)
Childs := Product(Tree, Tree);
```

- Modularity, extensibility, use of the system parser
- Integration with the rest of the system, sharing
- Systematic use of a virtual wrapper?
- Generalize to other gradings

Decomposable combinatorial classes / Species

```
combinat::decomposableObjects([
    Tree = Union(Leaf, NonTrivialTree),
    NonTrivialTree = Product(Label, Childs),
    Childs = Product(Tree, Tree)]):

    NonTrivialTree:  CombinatorialClass;
    Tree := Union (Leaf, NonTrivialTree);
    NonTrivialTree := Product(Label, Childs)
    Childs := Product(Tree, Tree);
```

- Modularity, extensibility, use of the system parser
- Integration with the rest of the system, sharing
- Systematic use of a virtual wrapper?
- Generalize to other gradings

Decomposable combinatorial classes / Species

```
combinat::decomposableObjects([
    Tree = Union(Leaf, NonTrivialTree),
    NonTrivialTree = Product(Label, Childs),
    Childs = Product(Tree, Tree)]):

    NonTrivialTree:  CombinatorialClass;
    Tree := Union (Leaf, NonTrivialTree);
    NonTrivialTree := Product(Label, Childs)
    Childs := Product(Tree, Tree);
```

- Modularity, extensibility, use of the system parser
- Integration with the rest of the system, sharing
- Systematic use of a virtual wrapper?
- Generalize to other gradings

First experiment: decomposable classes

Count and generator for:

- Epsilon (contains one element "epsilon" of size 0)
- Atom (contains one element "atom" of size 1)
- Integers, PositiveIntegers
- Union(A,B)
- Product(A,B)

Examples of use:

- Atom::count(1): (yields 1)
- Atom::list(1): (yields ["atom"])
- PositiveIntegers::count(): (yields infinity)
- PositiveIntegers::count(3): (yields 1)
- PositiveIntegers::list(3): (yields [3])
- IntegerVectors := Product(PositiveIntegers,PositiveIntegers)
- Comps := Union(Epsilon, Product(PositiveIntegers, Comps))
- Trees := Union(Atom, Product(Trees, Trees))

First experiment: decomposable classes

Count and generator for:

- Epsilon (contains one element "epsilon" of size 0)
- Atom (contains one element "atom" of size 1)
- Integers, PositiveIntegers
- Union(A,B)
- Product(A,B)

Examples of use:

- Atom::count(1): (yields 1)
- Atom::list(1): (yields ["atom"])
- PositiveIntegers::count(): (yields infinity)
- PositiveIntegers::count(3): (yields 1)
- PositiveIntegers::list(3): (yields [3])
- IntegerVectors := Product(PositiveIntegers,PositiveIntegers)
- Comps := Union(Epsilon, Product(PositiveIntegers, Comps))
- Trees := Union(Atom, Product(Trees, Trees))

Objects with several representations

- We need automatic coercion (implicit conversions)
- At interactive-level *and* inside code
- Possibly with > 1000 domains simultaneously

Problem: find the appropriate setting with the right balance

- Safety
- Practicalness
- Efficiency

Specificities of the MuPAD language

- Types are on values, not on variables:
  - Notion of facade domain
- You choose the level of strictness
- Copy semantic (no reference effect, except with closures or domains)
- No name-based overloading of functions:
  - Clumsy notation for method calling
  - Tendency to not overload functions
  - No overloading of methods
- No optimization (compiler / ...); no inlining:
  - Tendency to avoid wrappers (BAD!)
- Functions and domains are not strongly typed
- Almost no garbage collection of domains

And now?

What's the goal?

- How much work power?
- How many potential user?
- What are their needs?
- Aldor-Combinat? Axiom-Combinat?

How to proceed?

- Select specific goals with high value/time ratio
- Setup the stage: coding party
- Fill in the holes: progressively, while you need them

And now?

What's the goal?

- How much work power?
- How many potential user?
- What are their needs?
- Aldor-Combinat? Axiom-Combinat?

How to proceed?

- Select specific goals with high value/time ratio
- Setup the stage: coding party
- Fill in the holes: progressively, while you need them