Symmetric functions and Hopf algebras Usage and design in MuPAD-Combinat

François Descouens   Nicolas M. THIÉRY

Francois.Descouens@univ-mlv.fr
Institut Gaspard Monge, Université de Marne-la-Vallée, France

nthiery@users.sf.net
Laboratoire de Mathématiques d'Orsay, Université Paris Sud, France

RISC - June 14th 2007

Prehistory

- Schützenberger, Thibon, ...
- Programs in Pascal, C, ...
- Not really user friendly
- Hard to maintain, not really distributed

And then came Maple

- Ease of use
- Higher level programming language

Software

- SF (Stembridge)
- ACE (Veigneau, Lascoux, Thibon, Ung, ...)
- μ-EC (Prosper, Carré)

Design

- Data-structure : expressions
- Operators : expansion, bases change, scalar product, inner product, plethysm, ...
- Hall Littlewood, Mac Donald, NCSF, QSym, ...

Advantages

- Flexibility
- Easy to use (at least apparently ?)
- Widely available platform

Drawbacks

- Sloppy data structure (expression parsing)
- Non trivial coefficient rings ($\mathbb{Z}/2\mathbb{Z}$, degree 1 elements, ...) ?
- Non commutative Hopf algebras ?
- Naming conflicts
- Speed ?
- Maple

White book for MuPAD-Combinat

Goals

- Experimentation tool in the study of (Hopf) algebras
- Ease of use, expressiveness, flexibility, extensibility
- Speed ?
- Managing 30+ algebras, algebras with 10+ bases
- Code sharing, long term maintenance

White book for MuPAD-Combinat

Design decisions

- Object orientation
- MuPAD platform
- Reuse of existing software (Symmetrica, lrcalc, ...)
- Open source
- Core development by ”senior” researchers
- Decentralized development

The MuPAD platform

- Developed by Padderborn / Sciface since 1980's
- Not open source (bummer, bummer, bummer)
- Fairly open
- Reasonably priced, fairly widespread

Reasonable programming language

- Object oriented
  (encapsulation, Domains/Axioms/Categories, reflection)
- Functional programming (closures, ...)
- Dynamic modules (C++ integration)
- Very (too ?) flexible
- But special purpose

MuPAD-Combinat figures

- 8 developers, 20 contributers, 25+ research articles
- Official MuPAD library since 2002, NSF Sponsored
- 7 years, 10 official releases, 6 stable ones
- GNU/Linux, MacOS X, Windows, Zaurus
- 100000 lines of MuPAD, 15000 lines of C++
- 26000 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, lrcalc, Nauty, rigged configuration kernel
- How many users ?

Using symmetric functions in MuPAD-Combinat

The Hopf algebra framework

Building bricks

- Combinatorial classes
- Free-modules
- Category hierarchy
- Overloading mechanism
- Domains with several representations

Algorithmic

Internal algorithms

External software

- Symmetrica
- lrcalc
- gordan

Advanced demos

- Plethysms and other operators
- Hall-Littlewood, Macdonald
- LLT

What's wrong ?

With symmetric functions in MuPAD-combinat

- Few users (ecological niche ?)
- Very few contributers of new algorithmic (technological barrier)
- Remaining ACE / Lascoux algorithmic to be ported
- Too monolithic (lazier definitions, plug-in mechanisms)
- Speed ?

With MuPAD-combinat

- Reaching the complexity limits of MuPAD
- MuPAD is not open source

Computing with symmetric functions ? ? ?

What do you mean, really ?

- What is it exactly that you want to compute ?
- What does it mean to be efficient ?

Examples

- Combinatorics :
  - very sparse symmetric functions of high degree
  - Symmetric series
- Symmetric polynomials
- Symmetric functions on alphabets
- Symmetric functions on concrete alphabets
- Schur-Schubert polynomials

More than one model for symmetric functions

- Sparse expanded representation
- Lazy (dense ?) representation for series
  - Implementation by duality
  - Holonomic approach (Chyzac, Salvy, and co)
- Factorized / mixed expressions
- Straight line programs
- ...

Which one(s) to implement ?

FreeModules

- Encapsulation
- Internal data structure : kernel polynomials (variants possible)
- $\rightsquigarrow$ fast linear algebra (over kernel fields)
- Rankers (ranking/unranking of basis elements)
- Polynomials $\rightsquigarrow$ fast tensor products !

Overloading I

Conversions

- Fully centralized conversion graph
- Implicit conversions : canonical morphisms for all structures !
- Explicit conversions : aid to the user
- All domains are referenced there $\rightsquigarrow$ no memory recollection

Overloading II

Overloading

- Operator : list of signatures
- Resolution : scan through signatures and find cheapest required conversions
- $\rightsquigarrow$ non natural liftings
  (natural $\leftrightarrow$ strongly connected components ?)
- $\rightsquigarrow$ linear in the number of signatures
- Caching $\rightsquigarrow$ fast later overloading resolution (one table lookup)
- Each modification invalidates the cache $\rightsquigarrow$ bummer

It's good to be back at RISC !

Thanks Martin and Ralf !