MuPAD-Combinat
a package for computer exploration in algebraic combinatorics

Florent Hivert, Nicolas M. Thiéry, et Al.

mupad-combinat-devel@lists.sf.net
http://mupad-combinat.sf.net/

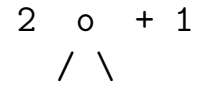
MuPAD-Combinat figures

- 3 core developers, 20 contributers
- Users and Developers meeting (15 persons)
- 65000 lines of code in high level language
  15000 lines of C++
  80000 lines of C imported from Symmetrica
- 9000 lines of tests, 450 pages of doc
- In 2003 : 1500 messages on the mailing list, 2000 visits of the web page and 200 downloads.
- How many users ?

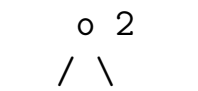
Example : computing in the Loday-Ronco algebra

```
>> lra := examples::LodayRoncoAlgebra():
   alias(BT=combinat::binaryTrees):
   operators::setTensorSymbol("#"):
>> tree := combinat::binaryTrees([NIL, [NIL], [NIL]])
```

```
>> 2*tree + 1
```



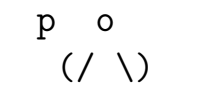
```
>> tree*tree
```



```
>> lra::p::info

  "Domain for the Loday-Ronco algebra on the fundamental basis"

>> el := lra::p(tree)
```



```
>> 2*el + 1
```

```
                          2 p  o   + p()
                             (/ \)
```

```
>> el*el
```

```
 p/  o    \ + p/  o   \ + p/   o  \ + p/  o   \ + p/   o  \ +
  | / \   |    | / \  |    |  / \ |    | / \  |    |  / \ |
  |    \  |    |   /\ |    | /\   |    |   /\ |    | /\   |
  \    /\ /    \   \  /    \   \  /    \  /   /    \  /   /

    p/    o  \
     |   / \ |
     |  /    |
     \ /\    /
```

```
>> el1 := lra::h(combinat::binaryTrees([NIL, [NIL, [NIL]]]))
```

```
                           h/   o \
                            |  /  |
                            \ /   /
```

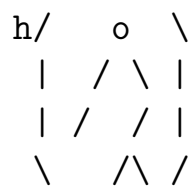
```
>> eh1 := lra::h(combinat::binaryTrees([NIL, [NIL, [NIL]]]))
```

```
                           h/   o \
                            |  /  |
                            \ /   /
```

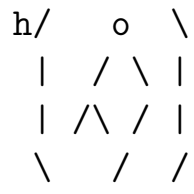
```
>> eh2 := lra::h(combinat::binaryTrees([NIL, [NIL, [NIL], [NIL]]]))
```

```
                           h/   o \
                            |  /  |
                            \ /\  /
```

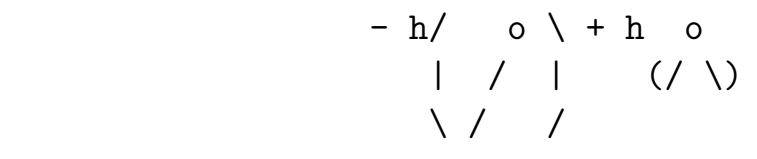
```
>> eh1*eh2
```



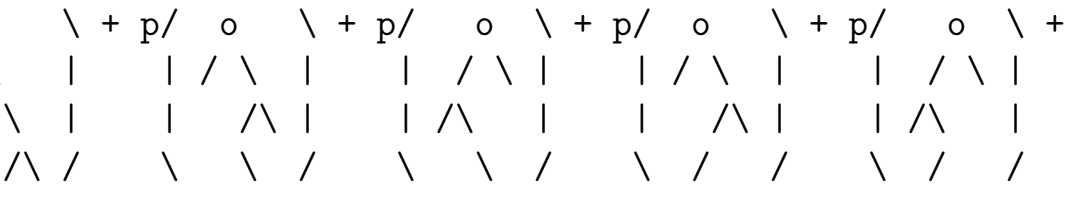
```
>> eh2*eh1
```



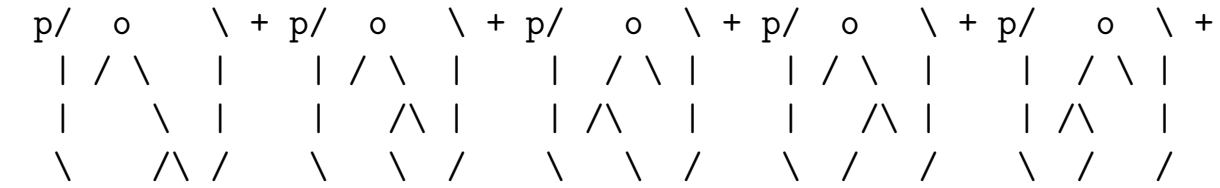
```
>> elh := lra::h(el)
```

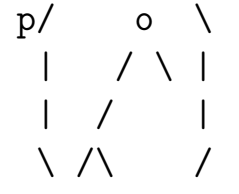


```
>> lra::p(elh^2)
```



```
>> el^2
```

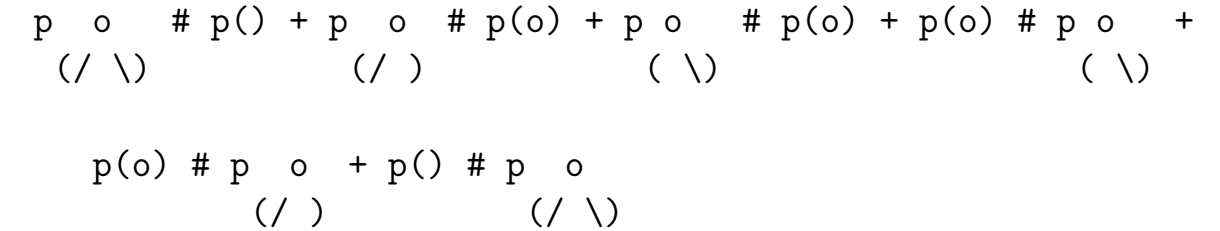




```
>> (r := lra::e(combinat::binaryTrees([NIL, [], combinat::binaryTrees::random(3)]));
   rh := lra::h(r);
   lra::h::nterms(rh))

                                8

>> operators::coproduct(el)
```



MuPAD

We present some basic features of `MuPAD` and its interface `MuPACS` with `emacs`.

Standard computer algebra stuff

```
>> 1+1
>> factor(1-x^10)
```

Long names, libraries, automatic completion

```
>> combinat::partitions::list(10);
>> alias(part=combinat::partitions):
   part::count(4);
   part::list(4);
   map(part::list(4), part::printPretty);
```

Typing (domains)

```
>> P2 := Dom::UnivariatePolynomial(x, Dom::IntegerMod(2)):
   p  := P2(x+1);
   domtype(p);
   p^2
>> Q := Dom::Rational:
   Qx := Dom::Fraction(Dom::DistributedPolynomial([x], Q)):
   F := Dom::AlgebraicExtension(Qx, poly(z^2 - x, [z])):
   P := Dom::DistributedPolynomial([u], F):
>> P(u*z)*P(z)
```

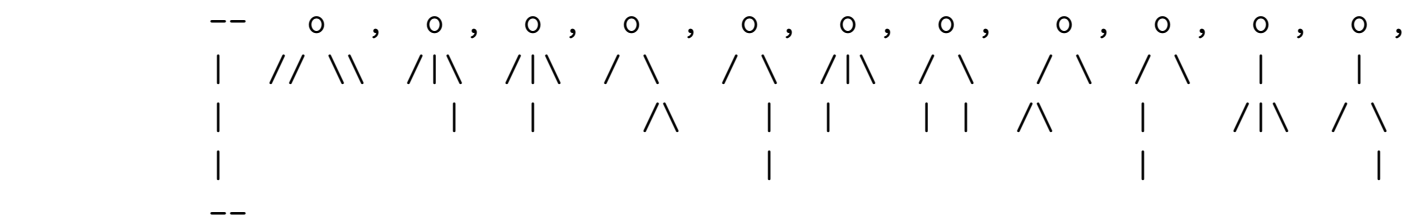
$$x\,u$$

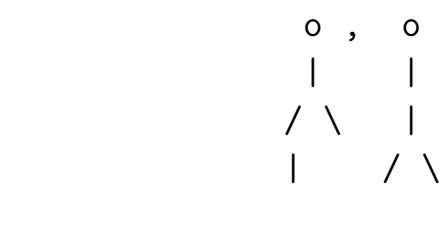
```
>> factor(P(u^2 - x^3))
```

$$(u + x\,z)\,(u - x\,z)$$

Manipulating combinatorial objects

```
>> export(combinat):
>> trees::list(5)
```





```
>> skewTableaux::printPretty([[3, 1],
                              [[5],
                               [3, 6, 8],
                                  [1, 2, 9],
                                        [4, 7]]])
```

```
+---+
| 5 |
+---+---+---+
| 3 | 6 | 8 |
+---+---+---+---+
```

```
                        | 1 | 2 | 9 |
                        +---+---+---+---+
                                | 4 | 7 |
                                +---+---+
```

```
>> skewTableaux::count([[5, 4, 3, 1], [3, 1]])
```

```
                              2205
```

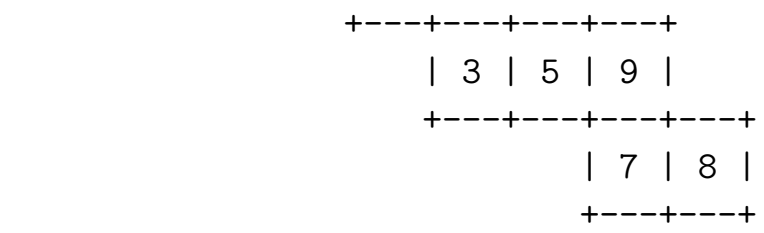
```
>> g := generators::map(skewTableaux::generator([[5, 4, 3, 1], [3, 1]]),
                       skewTableaux::printPretty):
```

```
>> g()
```

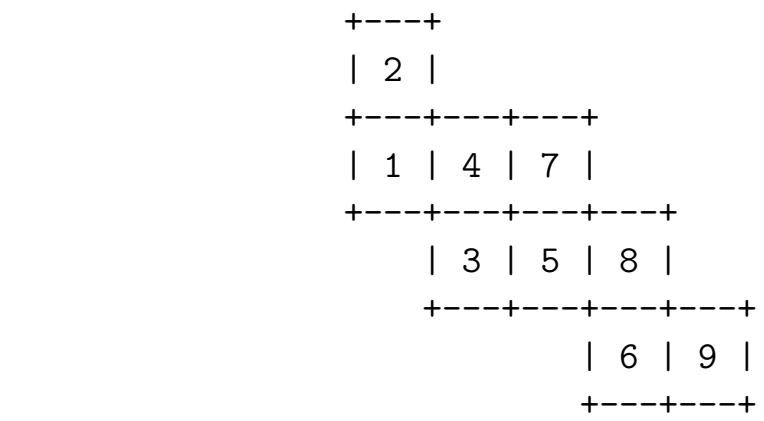
```
                    +---+
                    | 2 |
                    +---+---+---+
                    | 1 | 4 | 6 |
                    +---+---+---+---+
                        | 3 | 5 | 8 |
                        +---+---+---+---+
                                | 7 | 9 |
                                +---+---+
```

```
>> g()
```

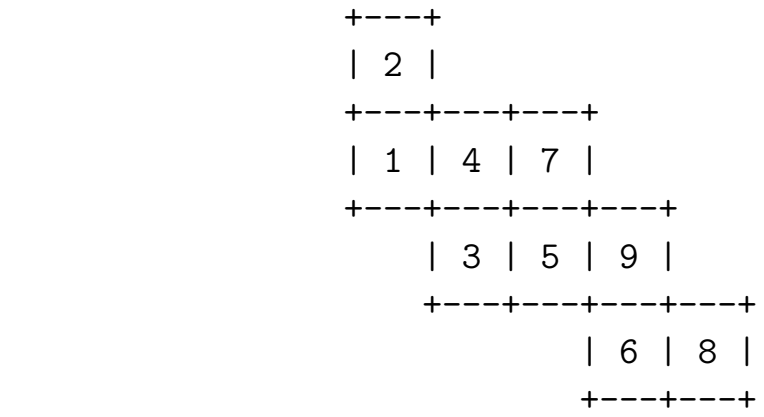
```
                    +---+
                    | 2 |
                    +---+---+---+
                    | 1 | 4 | 6 |
```



```
>> g()
```



```
>> g()
```



Defining new combinatorial classes

```
>> domain oddIntegers
       // This is a domain (not a library):
       inherits Dom::BaseDomain;
       // This is a graded combinatorial class:
       category Cat::GradedCombinatorialClass,
                // and a facade domain:
                Cat::FacadeDomain(DOM_INT);
       info_str := "The class of non negative odd integers";
       isA := n -> bool(testtype(n, Type::PosInt) and
                        n mod 2 <> 0);
       // The size of an odd integer is itself
       size  := n -> n;
       count := n -> if n mod 2 = 1 then 1 else 0 end_if;
       list  := n -> if n mod 2 = 1 then [n] else [] end_if;
       // No need to define generator;
       // it is defined via list by default
   end_domain:
>> oddPartsPartitions := combinat::decomposableObjects
                            ([Part = Multiset(oddIntegers)]):
>> oddPartsPartitions::list(5)

    [Multiset(5), Multiset(1, 1, 3), Multiset(1, 1, 1, 1, 1)]
```

Unlabelled graphs, Pólya enumeration, link with Nauty

```
>> S4 := Dom::SymmetricGroup(4):
   G4 := Dom::PermutationGroupOnSets(S4, 2):
>> Z4 := G4::cycleIndicator()

 1/24 p[1, 1, 1, 1, 1, 1] + 3/8 p[2, 2, 1, 1] + 1/3 p[3, 3] +

    1/4 p[4, 2]

>> Z4([1, 1])

                                11

>> combinat::graphs::list(4)

 [[[], [], [], []], [[], [], [4], [3]], [[], [4], [4], [2, 3]],

    [[4], [4], [4], [1, 2, 3]], [[2], [1], [4], [3]],

    [[3], [4], [1, 4], [2, 3]], [[], [3, 4], [2, 4], [2, 3]],

    [[4], [3, 4], [2, 4], [1, 2, 3]],

    [[2, 3], [1, 4], [1, 4], [2, 3]],

    [[3, 4], [3, 4], [1, 2, 4], [1, 2, 3]],

    [[2, 3, 4], [1, 3, 4], [1, 2, 4], [1, 2, 3]]]
```

```
>> nops(last(1))
```

```
                                  11
```

We can refine the counting to get the generating series of unlabelled graphs on 4 nodes by number of edges :

```
>> expand(Z4([1, q]))
```

```
                    2      3      4    5    6
           q + 2 q  + 3 q  + 2 q  + q  + q  + 1
```

Double check :

```
>> map(combinat::graphs::list(4), g -> combinat::graphs::getEdgeNumber(g)/2)
```

```
                [0, 1, 2, 3, 2, 3, 3, 4, 4, 5, 6]
```

Such computations are carried out in a rather efficient way, so that e.g. counting the number of graphs with 20 nodes takes just a few seconds :

```
>> (Dom::PermutationGroupOnSets(Dom::SymmetricGroup(20),
                               2))::cycleIndicator()([1, 1])
```

```
          64549012279579984185616463849074274944 0
```

Heaps of Pieces

```
>> domain setSpace(N: Type::PosInt, Rng = Dom::ExpressionField() : DOM_DOMAIN)
       inherits Dom::FreeModule(Rng,
                                combinat::subClass(combinat::subsets,
                                                   Parameters = {$1..N}));
       category Cat::FreeModule(Rng);

       dimension := dom::basisIndices::count();

       EiBasis   := proc(st : dom::basisIndices, i : Type::PosInt)
       begin
           if i > N then error("Invalid index") end_if;
           dom::term((st union {i}) minus {i-1, i+1})
       end_proc;
       Ei := dom::moduleMorphism(dom::EiBasis, ImageSet=dom);
   end_domain:
>> N := 3

                                3

>> space := setSpace(N)

                setSpace(3, Dom::ExpressionField())

>> gen := space::genericElement(z -> c[op(z)])

 c[1, 2, 3] B({1, 2, 3}) + c[2, 3] B({2, 3}) +
```

```
    c[1, 3] B({1, 3}) + c[1, 2] B({1, 2}) + c[3] B({3}) +

    c[2] B({2}) + c[1] B({1}) + c[] B({})
```

```
>> oper := z -> (1/N)*_plus(space::Ei(z, i) $ i = 1..N):
>> linsolve({coeff(oper(gen) - gen)})
```

```
 --                 c[1, 3]                          c[1, 3]
 |  c[] = 0, c[1] = -------, c[2] = c[1, 3], c[3] = -------,
 --                    2                               2

                                                    --
    c[1, 2] = 0, c[2, 3] = 0, c[1, 2, 3] = 0        |
                                                    --
```

```
>> statistique := proc(N)
     local space, gen, oper, i, res;
   begin
     space := setSpace(N);
     gen := space::genericElement(z -> c[op(z)]);
     oper := z -> (1/N)*_plus(space::Ei(z, i) $ i = 1..N);
     res := op(solve({coeff(oper(gen) - gen), c[1] = 1}))
   end_proc
```

```
                  proc statistique(N) ... end
```

```
>> statistique(3)
```

```
 [c[] = 0, c[1] = 1, c[2] = 2, c[3] = 1, c[1, 2] = 0,

    c[1, 3] = 2, c[2, 3] = 0, c[1, 2, 3] = 0]
```

```
>> _plus(op(map(statistique(3), op, 2)))
                                6
>> _plus(op(map(statistique(4), op, 2)))
                               24
>> _plus(op(map(statistique(5), op, 2)))
                               120
>> _plus(op(map(statistique(6), op, 2)))
                               720
```

Design goals

– Rapid prototyping
– Speed of development
– Different levels of use

– Expressiveness :
  What you write is what you think
  What's trivial in theory ought to be trivial in practice
– Flexible and generic tools
– Efficiency by using optimized low-level tools

Development model goals

– Share and recycle code quickly
– Deliver high quality
– Minimize long term maintenance
– Reward developers

Development model

– Repository of user contributions :
  Developed by users, for users
– Coherent and well structured framework
– Open source
– Agile programming

Why MuPAD ?

Tough question ! ! !

– General purpose computer algebra system
– Relatively open
– Strong user support
– Smooth learning curve
– Reasonable programming language