pathalg — A library for MuPAD

Amrey Krause

Fakultät für Mathematik
Universität Bielefeld

30th March 2001

Computing with path algebras

The pathalg package is a library for computing with modules and chain complexes over arbitrary factors of path algebras which are given by a finite quiver and an admissible ideal. The main purpose is to deal with objects and morphisms in the homotopy category of chain complexes. In particular, the library provides a method for finding the quiver with relations associated to the endomorphism algebra of a chain complex in the homotopy category.

Moreover, there are functions available to compute

- a noncommutative Gröbner basis of a finitely generated ideal;
- projective and injective modules over a given path algebra factor;
- homomorphism spaces of modules and chain complexes;
- a projective resolution of a module;
- the Auslander-Reiten translate of a module.

The library package as well as the full documentation and a Postscript version of this manual can be downloaded from `http://www.mathematik.uni-bielefeld.de/~akrause/PATHALG`.

The library is loaded by the function call `package(dirname)` where `dirname` is a string for the library directory which contains the package. The functions and data structures are loaded on demand on the first usage.

After loading the package, an overview of all available functions can be obtained by the MuPAD command `info(pathalg)`:

```
Library 'pathalg': Computing with path algebras

-- Interface:
pathalg::Cokernel,            pathalg::ComplexOfProjectives,
pathalg::EndAlgebraComplex,   pathalg::GroebnerBasis,
pathalg::HomComplex,          pathalg::HomComplexProjective,
pathalg::HomSpace,            pathalg::HomSpaceProjective,
pathalg::Injective,           pathalg::Kernel,
pathalg::Module,              pathalg::Nontips,
pathalg::NormalForm,          pathalg::OpAlgebra,
pathalg::PathAlgebra,         pathalg::Projective,
pathalg::ProjectiveComplex,   pathalg::Quiver,
pathalg::Radical,             pathalg::RadicalComplex,
pathalg::Shift,               pathalg::Simple,
...
```

After being exported, library functions can also be used in their short notation. The function call `export(pathalg)` exports all functions of the library pathalg and enables the user to use the function `projectiveCover` instead of `pathalg::projectiveCover`, for example.

Data types and constants

The library pathalg is based on the data type of the **path algebra**. It is defined by calling `pathalg::PathAlgebra` with two arguments: a base field and a finite quiver. As a base field any MuPAD data structure of the category `Cat::Field` may be used. For example, the finite fields are constructed by calling the function `Dom::GaloisField`, the complex numbers are predefined as `Dom::Complex`. Note that the computations performed over the field of complex, rational or real numbers (`Dom::Complex`, `Dom::Rational`, `Dom::Real`) may use numerical methods of the MuPAD system. This may result in approximations rather than exact results.

A **quiver** is given by the number of points N and a list of arrows. An arrow is defined by a list containing its source and its target, each a positive integer in $1, \ldots, N$, and a name, which is an arbitrary positive integer. Names are required to be unique. For example, the arrow $1 \xrightarrow{6} 3$ is represented by `[1,3,6]`.

An **element of a path algebra** is a linear combination of paths in the quiver. Thus, each element is a list of paths, each together with a coefficient. The zero element is represented by the empty list `[]`. A path is a list of arrow names, composed from left to right, e.g. `[1,2,5]` represents the path $\bullet \xrightarrow{1} \bullet \xrightarrow{2} \bullet \xrightarrow{5} \bullet$. The coefficient must be an element of the field.

For example, a path with coefficient is `[F(1), [1,2,5]]` where `F` is the ground field. The function call `F(1)` converts the integer `1` to an element of `F`.

The paths of length 0 are not represented by a list but only by the number of the point they are associated to. For example, `[F(1), 2]` is the path of length 0 starting in point `2` and with coefficient `1`. Note that ideals of a path algebra are not admissible if they contain paths of length 0.

Let A be the path algebra with quiver

$$\begin{tikzcd}
 & \bullet \arrow[r, "\alpha_2"] & \bullet \arrow[dr, "\alpha_3"] & \\
\bullet \arrow[ur, "\alpha_1"] \arrow[r, "\alpha_4"] \arrow[dr, "\alpha_6"'] & \bullet \arrow[r, "\alpha_5"] & \bullet \arrow[r, "\alpha_6"] & \bullet \\
 & \bullet \arrow[r, "\alpha_8"] & \bullet \arrow[ur, "\alpha_9"'] &
\end{tikzcd}$$

over the field `F`. The element $\alpha_3\alpha_2\alpha_1 - \alpha_6\alpha_5\alpha_4 - 3\alpha_9\alpha_8\alpha_7$ of A then is represented by

```
[ [F(1), [1,2,3]], [F(-1), [4,5,6]], [F(-3), [7,8,9]] ]
```

in MuPAD.

A **module** over a path algebra is a list `[dim, maps]`, where `dim` is a list of non-negative integers, representing the dimension vector of the module. The list must have as many entries as the number of objects of the algebra. The entry `maps` is a table where table entry `i` is a matrix representing the linear map corresponding to arrow `i`. Each arrow of the path algebra must occur in the table, even if the corresponding linear map is zero. The matrices in the table `maps` must be defined over the ground field. Although a module can be constructed "by hand" in the way described above, it is more convenient to use the method `pathalg::Module` to define a module.

Let M and N be two modules over a path algebra. A **module morphism** $f : M \longrightarrow N$ is a list of matrices over the ground field such that the i-th element of the list represents the linear map $f_i : M(i) \longrightarrow N(i)$.

A **chain complex** of modules over a path algebra is a list `[bounds, modules, maps]` where:

- `bounds` is a list of two integers `[lower, upper]` defining the bounds of the complex.
- `modules` is a table with entries indexed by integers `lower`, ..., `upper`. The entry `modules[i]` is the module in degree i of the complex.
- `maps` is also a table but entries are indexed from `lower` to `upper-1` where `maps[i]` is the boundary map `modules[i]` $\longrightarrow$ `modules[i+1]`.

A **chain complex of projectives** is also a list `[bounds, modules, maps]` where `bounds` and `maps` are as before, and `modules` is a table with entries indexed by `lower`, ..., `upper` but `modules[i]` is a list of objects of the underlying path algebra which represents a projective module.

Let C and D be two chain complexes. A **morphism of chain complexes** or a **chain map** is represented by a table with entries indexed by integers. The entry i contains the module homomorphism $C_i \longrightarrow D_i$ in degree i.

Some functions of the pathalg library can print out information messages on the computations. This depends on the verbosity level which is stored as an integer in the global variable `pathalg_VERBOSITY`. On loading the package, `pathalg_VERBOSITY` is initialized with zero (keep silent). A higher value means more messages, depending on the function called. Negative values are valid but have the same effect as zero.

Although it is possible to modify `pathalg_VERBOSITY` directly, please use the library functions `pathalg::incVerbosity`, `pathalg::decVerbosity` and `pathalg::setVerbosity` to adjust the verbosity level.

`pathalg::Cokernel` – the cokernel of a module morphism

`pathalg::Cokernel(A, f, M, N)` computes the cokernel of the map $f : M \longrightarrow N$ and returns the cokernel module as well as the cokernel morphism.

Call(s):

`pathalg::Cokernel(A, f, M, N)`

Parameters:

`A`	—	a path algebra
`f`	—	a module morphism `M` $\longrightarrow$ `N`
`M, N`	—	two modules over `A`

Return Value: a module and a module morphism

Related Functions: `pathalg::Kernel`

Details:

- For a description of the format of a module or a module morphism cf section Data types.
- For performance reasons it is not checked whether f, M and N are valid input values.

Example 1. Let $A = FQ$ be the hereditary algebra with quiver $Q = 1 \xrightarrow{1} 2 \xrightarrow{2} 3 \xrightarrow{3} 4$ over the field $F =$ `Dom::Real`.

```
>> F:= Dom::Real:
   A:= pathalg::PathAlgebra(F, [4, [[1,2,1], [2,3,2], [3,4,3]]]):
```

We define two modules M and N over A.

```
>> M:= pathalg::Module(A, [2,3,3,1], table(1=[[1,2],[1,0],[1,1]],
   2=[[1,0,0],[0,1,0],[1,1,1]], 3=[[1,1,1]])):
```

```
>> N:= pathalg::Module(A, [4,2,2,3], table(1=[[1,4,3,4],[0,1,1,0]],
   2=[[1,0],[1,0]], 3=[[1,1],[2,0],[0,3]])):
```

Now we define a module morphism $f : M \longrightarrow N$. This is a list of matrices representing the linear maps $f_i : M(i) \longrightarrow N(i)$. To make things easier, we first define a short cut for the MuPAD matrix constructor over F.

```
>> MC:= Dom::Matrix(F):
```

```
>> f:= [MC(4,2,[[8,8],[-3,-3],[3,3]]), MC(2,3,[[2,2,1]]), MC(2,3,1),
   MC([2,2,3])]
```

```
-- +-        -+                                             --
|  |  8,  8  |                                   +-   -+  |
|  |         |  +-         -+  +-         -+  |  2  |  |
|  | -3, -3  |  |  2, 2, 1  |  |  1, 1, 1  |  |     |  |
|  |         |, |           |, |           |, |  2  |  |
|  |  3,  3  |  |  0, 0, 0  |  |  1, 1, 1  |  |     |  |
|  |         |  +-         -+  +-         -+  |  3  |  |
|  |  0,  0  |                                   +-   -+  |
-- +-        -+                                             --
```

After defining f we compute its cokernel.

```
>> pathalg::Cokernel(A, f, M, N)
```

```
--                table(              --
|                     +-     -+        |
|                     |   -1  |        |
|                 3 = |       |,       |
|                     |  3/2  |        |
|                     +-     -+        |
|  [3, 1, 1, 2],      +- -+            |,
|                 2 = | 0 |,           |
|                     +- -+            |
|                     +-        -+     |
|                 1 = | 1, 1, 0 |      |
|                     +-        -+     |
--                )                   --

   +-                                                              -+
   |  +-                -+                                          |
   |  |   3/8, 1, 0, 0  |                        +-            -+   |
   |  |                 |  +-    -+  +-     -+  |   -1,  1, 0  |   |
   |  |  -3/8, 0, 1, 0  |, | 0, 1 |, | -1, 1 |, |              |   |
   |  |                 |  +-    -+  +-     -+  |  -3/2, 0, 1  |   |
   |  |    0,  0, 0, 1  |                        +-            -+   |
   |  +-                -+                                          |
   +-                                                              -+
```

`pathalg::ComplexOfProjectives` – construct a complex of projectives

`pathalg::ComplexOfProjectives(A, G, N, bounds, modules, maps)` returns a chain complex where the modules are given as list of objects in `A` which represent a direct sum of indecomposable projectives. A boundary map is given by an array of elements of `A`.

Call(s):

```
pathalg::ComplexOfProjectives(A, G, N, bounds, modules, maps)
```

Parameters:

`A`	—	a path algebra
`G`	—	a Gröbner basis of an ideal I of `A`
`N`	—	the list of nontips of I
`bounds`	—	a list of two integers; the bounds of the defined complex
`modules`	—	a table where entry i is a list of objects of `A` which represents the projective module in degree i
`maps`	—	a table where each entry is a matrix of elements of `A`

Return Value: a chain complex

Related Functions: `pathalg::homOfProjectives`

Details:

- If the bounds of the complex are `[m,n]` then the table `modules` may have entries `m, m+1, ..., n-1, n` where entry i represents the module in degree i. If an entry i does not occur the module in degree i is assumed to be the zero module.
- The table `maps` may have entries `m, m+1, ..., n-1` where entry i represents the boundary map $C_i \longrightarrow C_{i+1}$. The boundary maps are arrays of elements of the path algebra as described in `pathalg::homOfProjectives`. If an entry is not specified, the map is assumed to be zero.
- For a detailed description of the format of a chain complex see section Data types.

Example 1. Let A be the algebra given by the quiver

$$\begin{tikzcd}
 & 2 \arrow[dr, "\alpha_2"] & \\
1 \arrow[ur, "\alpha_1"] \arrow[r, "\alpha_3"] \arrow[dr, "\alpha_5"'] & 3 \arrow[r, "\alpha_4"] & 5 \\
 & 4 \arrow[ur, "\alpha_6"'] &
\end{tikzcd}$$

with ideal generated by $\{\alpha_2\alpha_1 + \alpha_4\alpha_3 + \alpha_6\alpha_5\}$ over the base field `F:= Dom::Rational`.

```
>> F:= Dom::Rational:
   A:= pathalg::PathAlgebra(F, [5, [[1,2,1], [2,5,2], [1,3,3], [3,5,4],
   [1,4,5], [4,5,6]]]):
   G:= pathalg::GroebnerBasis(A, [ [[1, [1,2]], [1, [3,4]], [1, [5,6]]] ]):
   N:= pathalg::Nontips(A, G):
```

We want to define the complex of projectives

$$0 \longrightarrow P_5 \xrightarrow{\begin{pmatrix}\alpha_2\\ \alpha_4\\ \alpha_6\end{pmatrix}} P_2 \oplus P_3 \oplus P_4 \xrightarrow{(\alpha_1,\alpha_2,\alpha_3)} P_1 \longrightarrow 0$$

in degrees $-2, -1, 0$. First we construct the arrays of algebra elements which give the boundary maps.

```
>> a:= array(1..3, 1..1): a[1,1]:= [[F(1), [2]]]:
   a[2,1]:= [[F(1), [4]]]: a[3,1]:= [[F(1), [6]]]: a

                        +-             -+
                        |  [[1, [2]]]   |
                        |               |
                        |  [[1, [4]]]   |
                        |               |
                        |  [[1, [6]]]   |
                        +-             -+
```

```
>> b:= array(1..1, 1..3): b[1,1]:= [[F(1), [1]]]:
   b[1,2]:= [[F(1), [3]]]: b[1,3]:= [[F(1), [5]]]: b

              +-                                    -+
              | [[1, [1]]], [[1, [3]]], [[1, [5]]] |
              +-                                    -+
```

Now we define the chain complex.

```
>> pathalg::ComplexOfProjectives(A, G, N, [-2,0],
   table(-2=[5], -1=[2,3,4], 0=[1]), table(-2=a, -1=b))

 --          table(
 |                --                    table(          --
 |                |                         +-     -+    |
 |                |                         |  -1  |    |
 |                |                     6 = |      |,   |
 |                |                         |  -1  |    |
 |                |                         +-     -+    |
 |                |                         +- -+        |
 |                |                     5 = | 1 |,       |
 |                |                         +- -+        |
 |                |                         +-    -+     |
 |                |                         |  0  |      |
```

```
|                 |  +-            -+   4 = |     |,   |
|           0 =  |  | 1, 1, 1, 1, 2 |,       |  1  |    |,     . . .
|                 |  +-            -+       +-   -+    |
|                 |                          +- -+      |
|                 |                      3 = | 1 |,     |
|                 |                          +- -+      |
|                 |                          +-   -+    |
|                 |                          |  1  |    |
|                 |                      2 = |     |,   |
|                 |                          |  0  |    |
|                 |                          +-   -+    |
|                 |                          +- -+      |
|                 |                      1 = | 1 |      |
|                 |                          +- -+      |
|                --                    )              --
--          )
```

The output is not printed in full here due to its size.

`pathalg::decVerbosity` – decrease the verbosity level

Call(s):

`pathalg::decVerbosity()`

Return Value: an integer; the current value of `pathalg_VERBOSITY`.

Related Functions: `pathalg::incVerbosity`, `pathalg::setVerbosity`

Details:

- The verbosity level is decreased by 1. The amount of information messages printed depends on the function called.

Example 1. The verbosity level is stored in `pathalg_VERBOSITY`.

```
>> pathalg_VERBOSITY

                              1

>> pathalg::decVerbosity()

                              0
```

`pathalg::dimRad` – dimension vector of the radical of a module

`pathalg::dimRad(A, N, M)` computes the dimension vector of the radical of a module M over a quotient of the path algebra A.

Call(s):

```
pathalg::dimRad(A, N, M)
```

Parameters:

- `A` — a path algebra
- `N` — the list of nontips of an ideal I of A
- `M` — a module over A/I

Return Value: a list of non-negative integers; the dimension vector of the radical of `M`.

Related Functions: `pathalg::dimTop`

Example 1. We compute the dimension vector of the radical of an indecomposable projective module over the algebra A given by the quiver

$$1 \underset{\alpha_2}{\overset{\alpha_1}{\rightleftarrows}} 2$$

with ideal generated by $\alpha_1\alpha_2\alpha_1$.

```
>> F:= Dom::Rational:
   A:= pathalg::PathAlgebra(F, [2, [[1,2,1], [2,1,2]]]):
   G:= pathalg::GroebnerBasis(A, [ [[1, [1,2,1]]] ]):
   N:= pathalg::Nontips(A, G):
```

Now we construct the indecomposable projective module P corresponding to object 2.

```
>> P:= pathalg::Projective(A, G, N, 2)
```

```
--           table(           --
|                  +-      -+   |
|                  |  1, 0  |   |
|              2 = |        |,  |
|  +-    -+        |  0, 1  |   |
|  | 2, 2 |,       +-      -+   |
|  +-    -+        +-      -+   |
|                  |  0, 0  |   |
|              1 = |        |   |
|                  |  1, 0  |   |
|                  +-      -+   |
--           )                --
```

Then we compute the dimension vector of the radical of P.

```
>> pathalg::dimRad(A, N, P)

                              +-      -+
                              | 2, 1 |
                              +-      -+
```

Background:

- The radical $\operatorname{rad} M$ of a module M is the intersection of all maximal submodules of M or, equivalently, $\operatorname{rad} M$ is the intersection of all submodules U of M such that the factor M/U is semi-simple.

`pathalg::dimTop` – dimension vector of the top of a module

`pathalg::dimTop(A, N, M)` computes the dimension vector of the semi-simple factor module $\operatorname{top} M$ of the module M over A.

Call(s):

`pathalg::dimTop(A, N, M)`

Parameters:

`A` — a path algebra
`N` — the list of nontips of an ideal I of A
`M` — a module over A/I

Return Value: a list of non-negative integers which represents the dimension vector of the module top `M`.

Related Functions: `pathalg::dimRad`

Example 1. We compute the dimension vector of the top of an indecomposable projective module over the algebra A given by the quiver

$$1 \underset{\alpha_2}{\overset{\alpha_1}{\rightleftarrows}} 2$$

with ideal generated by $\alpha_1\alpha_2\alpha_1$.

```
>> F:= Dom::Rational:
   A:= pathalg::PathAlgebra(F, [2, [[1,2,1], [2,1,2]]]):
   G:= pathalg::GroebnerBasis(A, [ [[1, [1,2,1]]] ]):
   N:= pathalg::Nontips(A, G):
```

Now we construct the indecomposable projective module P corresponding to object 2.

```
>> P:= pathalg::Projective(A, G, N, 2)
```

```
--            table(            --
|                 +-      -+    |
|                 |  1, 0  |    |
|             2 = |        |,   |
|  +-    -+       |  0, 1  |    |
|  | 2, 2 |,      +-      -+    |
|  +-    -+       +-      -+    |
|                 |  0, 0  |    |
|             1 = |        |    |
|                 |  1, 0  |    |
|                 +-      -+    |
--            )                 --
```

Then we compute the dimension vector of the top of P.

```
>> pathalg::dimTop(A, N, P)
```

```
+-    -+
| 0, 1 |
+-    -+
```

The top of P is the simple module corresponding to 2.

Background:

- The top of a module M is defined as $\operatorname{top} M := M/\operatorname{rad} M$, so there exists a short exact sequence of modules
$$0 \longrightarrow \operatorname{rad} M \longrightarrow M \longrightarrow \operatorname{top} M \longrightarrow 0.$$
- The top of a module M is a semi-simple factor module of M, i.e. $\operatorname{top} M$ is a sum of simple modules. Therefore $\operatorname{top} M$ is determined by its dimension vector.

`pathalg::directSum` – the direct sum of a list of modules

Call(s):

`pathalg::directSum(A, L)`

Parameters:

`A` — a path algebra
`L` — a list of modules over `A`

Return Value: the direct sum of the modules in the list `L`.

Details:

- The list `L` of modules may contain any number of modules. It is not checked whether they are really modules over `A`.

Example 1. Let $A = FQ$ be the hereditary path algebra where $Q = 1 \xrightarrow{1} 2 \xrightarrow{2} 3$ and F is the field of complex numbers. We define two A-modules M_1 and M_2 and compute their direct sum.

```
>> A:= PathAlgebra(Dom::Complex, [3, [[1,2,1], [2,3,2]] ]):

>> M1:= Module(A, [1,2,2], table(1=1, 2=[[1,1],[1,1]]) )

          --             table(            --
          |                   +-      -+    |
          |                   |  1, 1  |    |
          |               2 = |        |,   |
          |                   |  1, 1  |    |
          |  [1, 2, 2],       +-      -+    |
          |                   +-   -+       |
          |                   |  1  |       |
          |               1 = |     |       |
          |                   |  0  |       |
          |                   +-   -+       |
          --             )                 --

>> M2:= Module(A, [3,3,0], table(1=2) )

          --             table(               --
          |                   +-       -+       |
          |               2 = | 0, 0, 0 |,      |
          |                   +-       -+       |
          |                   +-           -+   |
          |  [3, 3, 0],       |  2, 0, 0   |    |
          |                   |            |    |
          |               1 = |  0, 2, 0   |    |
          |                   |            |    |
          |                   |  0, 0, 2   |    |
          |                   +-           -+   |
          --             )                    --

>> directSum(A, [M1, M2])
```

```
--              table(                       --
|                     +-                -+    |
|                     |  1, 1, 0, 0, 0  |    |
|                 2 = |                 |,   |
|                     |  1, 1, 0, 0, 0  |    |
|                     +-                -+    |
|                     +-            -+        |
|  +-        -+       |  1, 0, 0, 0  |        |
|  | 4, 5, 2 |,       |              |        |
|  +-        -+       |  0, 0, 0, 0  |        |
|                     |              |        |
|                 1 = |  0, 2, 0, 0  |        |
|                     |              |        |
|                     |  0, 0, 2, 0  |        |
|                     |              |        |
|                     |  0, 0, 0, 2  |        |
|                     +-            -+        |
--              )                            --
```

Background:

- Let M and N be two A-modules. Then their direct sum $M \oplus N$ is given by $(M \oplus N)(x) := M(x) \oplus N(x)$ for any object x in A and by

$$(M \oplus N)(\alpha) := \begin{pmatrix} M(\alpha) & 0 \\ 0 & N(\alpha) \end{pmatrix}$$

for an arrow α in A.

`pathalg::EndAlgebraComplex` – the endomorphism algebra of a complex

`pathalg::EndAlgebraComplex(A, G, N, [T1, T2, ...])` computes a quiver Q' and an ideal I' of FQ' such that the endomorphism algebra of $\texttt{T1} \oplus \texttt{T2} \oplus \ldots$ is isomorphic to FQ'/I'.

Call(s):

`pathalg::EndAlgebraComplex(A, G, N, L)`

Parameters:

`A` — a path algebra
`G` — a Gröbner basis of an ideal I of A
`N` — the list of nontips of I
`L` — a list of complexes of projectives

Return Value: a path algebra B, the quiver Q' of B and a Gröbner basis of an ideal I' of B such that $B \cong FQ'/I'$ or `FAIL` if no such quiver and ideal exist.

Related Functions: `pathalg::isTiltingComplex`

Side Effects: The amount of information messages printed depends on the verbosity level `pathalg_VERBOSITY`.

Details:

- The complexes in `L` are assumed to be pairwise non-isomorphic and indecomposable.
- To compute the endomorphism algebra of a module given by its direct summands, use `pathalg::projectiveResolution` with the format option *`ProjComplex`* to represent the module summands as complexes of projectives and pass the list of these complexes to `pathalg::EndAlgebraComplex`.
- For details on the format of a complex of projectives see section Data types.

Example 1. Let $A = FQ/I$ be the algebra with quiver $1 \xrightarrow{\alpha} 2 \xrightarrow{\beta} 3$ and ideal I generated by $\{\beta\alpha\}$ over the field $\mathbb{F}_7$.

```
>> F:= Dom::IntegerMod(7): MC:= Dom::Matrix(F):
   A:= pathalg::PathAlgebra(F, [3, [[1,2,1], [2,3,2]]]):
   G:= pathalg::GroebnerBasis(A, [ [[1, [1,2]]] ]):
   N:= pathalg::Nontips(A, G):
```

We define three complexes of projectives T_1, T_2 and T_3 over A such that their direct sum T is a tilting complex.

```
>> T1:= [ [0,0], table(0=[2]), table() ]:
   T2:= [ [-1,0], table(-1=[3], 0=[2]),
   table(-1=[MC([0]), MC([0]), MC([1])]) ]:
   T3:= [ [0,0], table(0=[1]), table() ]:
```

Now we compute the quiver and ideal of the endomorphism algebra B of T.

```
>> pathalg::EndAlgebraComplex(A, G, N, [T1, T2, T3])

 ....................
                              Elapsed real time: 1 sec

            table(
              "field" = Dom::IntegerMod(7),
              "arrowsfrom3" = [],
              "arrowsfrom2" = [2],
              "arrowsfrom1" = [1],         , [3, [[1, 2], [2, 3]]], []
              "arrowlist" = [1, 2],
              "arrow2" = [2, 3],
              "arrow1" = [1, 2],
              "points" = 3
            )
```

We receive as first return value the path algebra FQ', as second return value the quiver $Q' = 1 \longrightarrow 2 \longrightarrow 3$ and the third return value is a Gröbner basis of an ideal I' of FQ', in this case the zero ideal generated by the empty set, such that $B \cong FQ'/I'$.

If we increase the level of verbosity we get information messages on the homomorphism space $\mathrm{Hom}(T_i, T_j)$ which is currently computed, and the quiver and the relations that have been found.

```
>> pathalg::incVerbosity()

                                   1

>> pathalg::EndAlgebraComplex(A, G, N, [T1, T2, T3])

                Computing HomComplex(1, 1) ...

          Computed HomComplex (1, 1), dimension is 1.

                Computing HomComplex(1, 2) ...

          Computed HomComplex (1, 2), dimension is 1.

                Computing HomComplex(1, 3) ...

          Computed HomComplex (1, 3), dimension is 1.

                Computing HomComplex(2, 1) ...

          Computed HomComplex (2, 1), dimension is 0.

                Computing HomComplex(2, 2) ...

          Computed HomComplex (2, 2), dimension is 1.

                Computing HomComplex(2, 3) ...

          Computed HomComplex (2, 3), dimension is 1.

                Computing HomComplex(3, 1) ...

          Computed HomComplex (3, 1), dimension is 0.

                Computing HomComplex(3, 2) ...

          Computed HomComplex (3, 2), dimension is 0.

                Computing HomComplex(3, 3) ...

          Computed HomComplex (3, 3), dimension is 1.
 ...................
```

```
                    Quiver:

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

              Checking for relations ...
..
                       []

              Elapsed real time: 1 sec

table(
  "field" = Dom::IntegerMod(7),
  "arrowsfrom3" = [],
  "arrowsfrom2" = [2],
  "arrowsfrom1" = [1],          , [3, [[1, 2], [2, 3]]], []
  "arrowlist" = [1, 2],
  "arrow2" = [2, 3],
  "arrow1" = [1, 2],
  "points" = 3
)
```

Even more information messages are printed at higher verbosity levels.

`pathalg::GroebnerBasis` – compute a Gröbner basis of an ideal

`pathalg::GroebnerBasis(A, S)` computes a Gröbner basis (with respect to the length-lexicographic order) of the ideal of `A` generated by the elements in the list `S`.

Call(s):

`pathalg::GroebnerBasis(A, S <, maxlength = 20>)`

Parameters:

`A`	—	a path algebra
`S`	—	a list of elements of `A` which generate an ideal
`maxlength`	—	a positive integer

Return Value: a list of elements of `A` which form a Gröbner basis of the ideal generated by `S`.

Related Functions: `pathalg::NormalForm`, `pathalg::Tips`

Details:

- The arrows are ordered according to their names in the usual order of integers. Then the length-lexicographic order generated by the order of arrows is used as the admissible order on the basis of `A` as a vector space.

- Since a Gröbner basis can be infinite (if `A/I` is not finite dimensional) the number of elements in the Gröbner basis is bounded by `maxlength`. The default value of `maxlength` is 20.
- The elements in `G` are lists of paths with coefficients. These coefficients are converted to elements of the ground field if possible.
- The format of a path algebra element is described in the section Data types.

Example 1. Let A be the algebra FQ/I with Q the quiver

$$\begin{tikzcd} 1 \arrow[rr, shift left, "\beta"'] \arrow[dr, "\gamma"'] & & 2 \arrow[ll, shift left, "\alpha"'] \\ & 3 \arrow[ur, "\delta"'] & \end{tikzcd}$$

over the field $F =$ `Dom::Real` and the ideal I generated by the elements $\beta\alpha\beta\alpha\beta\alpha$, $\delta\gamma\alpha - \beta\alpha\beta\alpha$ and $\alpha\beta\alpha\delta$. A Gröbner basis of I is

$$\{\beta\alpha\delta\gamma\alpha, \beta\alpha\beta\alpha - \delta\gamma\alpha, \alpha\beta\alpha\delta, \delta\gamma\alpha\beta\alpha, \delta\gamma\alpha\delta\}.$$

The arrows are numbered corresponding to the order $\alpha, \beta, \gamma, \delta$.

```
>> F:= Dom::Real:
   A:= pathalg::PathAlgebra(F, [3, [[2,1,1], [1,2,2], [1,3,3], [3,2,4]]]):

>> S:= [ [[F(1), [1,2,1,2,1,2]]],
         [[F(1), [1,3,4]], [F(-1), [1,2,1,2]]],
         [[F(1), [4,1,2,1]]] ]:
```

Now we compute a Gröbner basis of the ideal generated by `S`.

```
>> pathalg::GroebnerBasis(A, S)

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

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

It is also possible to give the coefficients of the paths without converting them to elements of the field F because `pathalg::GroebnerBasis` does this automatically.

```
>> S:= [ [[1, [1,2,1,2,1,2]]],
         [[1, [1,3,4]], [-1, [1,2,1,2]]],
         [[1, [4,1,2,1]]] ]:

>> pathalg::GroebnerBasis(A, S)

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

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

`pathalg::HomComplex` – the homomorphism space of two complexes

`pathalg::HomComplex(A, C1, C2)` computes a basis of the vector space of homomorphisms `C1` $\to$ `C2` where `C1` and `C2` are chain complexes over `A`.

Call(s):

`pathalg::HomComplex(A, C1, C2)`

Parameters:

`A` — a path algebra
`C1, C2` — chain complexes over `A`

Return Value: the dimension and a list of chain maps which form a basis of the vector space Hom(`C1`, `C2`) in the category of chain complexes.

Related Functions: `pathalg::HomComplexProjective`

Side Effects: The amount of information messages printed depends on the verbosity level `pathalg_VERBOSITY`.

Details:

- `pathalg::HomComplex` does not check if `C1` and `C2` are really chain complexes over `A`.
- If `C1` is a complex of projectives please use `pathalg::HomComplexProjective` instead, it will be faster.
- A description of the format of a chain complex can be found in the section Data types.

Example 1. Let A be the hereditary algebra with the quiver $2 \longleftarrow 1 \longrightarrow 3$ over the field $F =$ `Dom::Complex`.

```
>> F:= Dom::Complex: MC:= Dom::Matrix(F):
   A:= pathalg::PathAlgebra(F, [3, [[1,2,1], [1,3,2]]]):
```

We define two complexes `C1` and `C2` over `A`.

```
>> M1:= pathalg::Module(A, [2,2,2], table(1=[[1,1],[1,0]], 2=I)):
   M2:= pathalg::Module(A, [1,3,2], table(1=[[1],[1+I],[1-I]], 2=1)):
   M3:= pathalg::Module(A, [1,0,2]):
   d1:= [MC([[0,I]]), MC([[I, -I],[-1+I, 1-I],[1+I,-1-I]]),
        MC([[0,1],[0,0]])]:
   d2:= [ MC(1,1), MC(1,3), MC([[0,1],[0,0]]) ]:
```

```
>> C1:= [[-1,1], table(-1=M1, 0=M2, 1=M3), table(-1=d1, 0=d2)]:
```

```
>> N1:= pathalg::Module(A, [3,2,2], table(1=[[2-2*I,1,0],[0,1,1/3]],
   2=[[1,0,2],[1,1,1-I]]) ):
   e1:= [ MC(3,3,1,Diagonal), MC(2,2,1,Diagonal), MC(2,2,1,Diagonal) ]:
```

```
>> C2:= [ [0,1], table(0=N1, 1=N1), table(0=e1) ]:
```

Now we compute an F-basis of the space Hom(`C1`, `C2`). The first return value is the dimension of the space.

```
>> h:= pathalg::HomComplex(A, C1, C2)
```

```
    -- table(
    |        +-                              -+
    |        |  +-   -+                       |
    |        |  |  0  |  +-   -+  +-       -+  |
    |        |  |     |  |  0  |  |  1, 0  |  |
    |     1 = |  |  0  |, |     |, |        |  |,
    |        |  |     |  |  0  |  |  0, 0  |  |
    |        |  |  0  |  +-   -+  +-       -+  |
    |        |  +-   -+                       |
 4, |        +-                              -+        ,    . . .
    |        +-                                    -+
    |        |  +-   -+                             |
    |        |  |  0  |  +-         -+  +-      -+  |
    |        |  |     |  |  0, 0, 0  |  |  0, 1  |  |
    |     0 = |  |  0  |, |           |, |        |  |
    |        |  |     |  |  0, 0, 0  |  |  0, 0  |  |
    |        |  |  0  |  +-         -+  +-      -+  |
    |        |  +-   -+                             |
    |        +-                                    -+
    -- )
```

(The output was cut short.) We can also compute a basis of Hom(`C1`, `C2`[-1]). In the following we will only show the dimension as return value to shorten the output a bit.

```
>> pathalg::HomComplex(A, C1, pathalg::Shift(C2, -1))[1]
```

```
9
```

Now we increase the verbosity to get some more information on the computations going on and compute the homomorphism space again.

```
>> pathalg::incVerbosity()
```

```
1
```

```
>> pathalg::HomComplex(A, C1, pathalg::Shift(C2, -1))[1]
```

```
Computed HomSpace -1, dimension is 6.

Computed HomSpace 0, dimension is 9.

Solving system of linear equations, size: (14,15)

9
```

---

pathalg::HomComplexProjective – the homomorphism space of a complex of projectives and an arbitrary complex

`pathalg::HomComplexProjective(A, G, N, P, C)` computes a basis of the vector space of homomorphisms `P` $\to$ `C` where `P` is a complex of projectives and `C` is an arbitrary chain complex over the algebra given by the data (`A`, `G`, `N`).

Call(s):

`pathalg::HomComplexProjective(A, G, N, P, C)`

Parameters:

- `A` — a path algebra
- `G` — a Gröbner basis of an ideal I of `A`
- `N` — the list of nontips of I
- `P` — a complex of projectives
- `C` — a chain complex

Return Value: the dimension of Hom(`P`, `C`) and a list of morphisms `P` $\longrightarrow$ `C` which form a basis of the space Hom(`P`, `C`).

Related Functions: `pathalg::HomComplex`

Side Effects: The information messages printed during computations depends on the verbosity level `pathalg_VERBOSITY`.

Details:

- For details on the format of a chain complex or a chain complex of projectives see the section Data types.

Example 1. Let $A = FQ/I$ be the algebra with quiver $Q = 1 \xrightarrow{\alpha} 2 \xrightarrow{\beta} 3$ and ideal I generated by $\{\beta\alpha\}$ over the field $F := \mathbb{Q}$ of rational numbers.

```
>> F:= Dom::Rational: MC:= Dom::Matrix(F):
   A:= pathalg::PathAlgebra(F, [3, [[1,2,1], [2,3,2]]]):
   G:= pathalg::GroebnerBasis(A, [ [[1, [1,2]]] ]):
   N:= pathalg::Nontips(A, G):
```

We define a complex of projectives

$$P = 0 \longrightarrow P_3 \oplus P_2 \xrightarrow{f} P_2 \oplus P_1 \oplus P_1 \longrightarrow 0$$

where f is the map determined by $a := \begin{pmatrix} \beta & id_2 \\ 0 & \alpha \\ 0 & \alpha \end{pmatrix}$.

```
>> a:= array(1..3, 1..2):
   a[1,1]:= [[F(1), [2]]]: a[1,2]:= [[F(1),2]]: a[2,1]:= []:
   a[2,2]:= [[F(1),[1]]]: a[3,1]:= []: a[3,2]:= [[F(1),[1]]]:

>> f:= pathalg::homOfProjectives(A, G, N, [3,2], [2,1,1], a):

>> P:= [[0,1], table(0=[3,2], 1=[2,1,1]), table(0=f)]

   --                            table(                                   --
   |                                   --          +-    -+            --   |
   |                                   |  +-    -+  |  1  |             |   |
   |          table(                   |  |  0  |   |     |  +-     -+  |   |
   |  [0, 1],    1 = [2, 1, 1],,   0 = |  |     |,  |  1  |, | 1, 1 |   |   |
   |             0 = [3, 2]            |  |  0  |   |     |  +-     -+  |   |
   |          )                        |  +-    -+  |  1  |             |   |
   |                                   --          +-    -+            --   |
   --                            )                                        --
```

Now we define a complex $C : 0 \to C_1 = C_1 \to 0$ where C_1 is as follows:

```
>> C1:= pathalg::Module(A, [2,2,3], table(1=[[1,1],[1,1]],
   2=[[1,-1],[0,0],[-1,1]]), G):
   d:= [ MC(2,2,1,Diagonal), MC(2,2,1, Diagonal), MC(3,3,1,Diagonal) ]:

>> C:= [[0,1], table(0=C1, 1=C1), table(0=d)]:
```

Then we compute the F-dimension and an F-basis of $\mathrm{Hom}(P, C)$.

```
>> pathalg::HomComplexProjective(A, G, N, P, C)
```

```
    -- table(
    |        +-                                         -+
    |        |                              +-     -+   |
    |        |  +-       -+  +-          -+ |   1   |   |
    |        |  |  0, 0  |  |  1, 0, 0  |  |       |   |
    |    1 = |  |        |, |           |, |   0   |   |,
    |        |  |  0, 0  |  |  0, 0, 0  |  |       |   |
    |        |  +-       -+  +-          -+ |  -1   |   |
    |        |                              +-     -+   |
 6, |        +-                                         -+ ,        . . .
    |        +-                                    -+
    |        |                    +-         -+    |
    |        |  +-    -+  +-   -+ |   1,  1  |    |
    |        |  |  0  |  |  1  |  |          |    |
    |    0 = |  |     |, |     |, |   0,  0  |    |
    |        |  |  0  |  |  0  |  |          |    |
    |        |  +-    -+  +-   -+ |  -1, -1  |    |
    |        |                    +-         -+    |
    |        +-                                    -+
    -- )
```

(The output was cut short.) If we increase the level of verbosity we get information messages about the dimension of the space $\mathrm{Hom}(P_i, C_i)$ in each degree i and the size of the system of linear equations that must be solved.

```
>> pathalg::incVerbosity()

                                          1

>> pathalg::HomComplexProjective(A, G, N, P, C)

                       Computed HomSpace 0, dimension is 5.

                       Computed HomSpace 1, dimension is 6.

                Solving system of linear equations, size: (10,11)

    -- table(
    |        +-                                         -+
    |        |                              +-     -+   |
    |        |  +-       -+  +-          -+ |   1   |   |
    |        |  |  0, 0  |  |  1, 0, 0  |  |       |   |
 6, |    1 = |  |        |, |           |, |   0   |   |,  . . .
    |        |  |  0, 0  |  |  0, 0, 0  |  |       |   |
    |        |  +-       -+  +-          -+ |  -1   |   |
    |        |                              +-     -+   |
    |        +-                                         -+
    -- )
```

`pathalg::homOfProjectives` – construct a morphism of projective modules

`pathalg::homOfProjectives(A, G, N, P1, P2, a)` constructs the homomorphism of projective modules $P1 \to P2$ which is given by an element `a` of the algebra `(A, G, N)`.

Call(s):

```
pathalg::homOfProjectives(A, G, N, P1, P2, a)
```

Parameters:

`A`	—	a path algebra
`G`	—	a Gröbner basis of an ideal I
`N`	—	the list of nontips with respect to I
`P1, P2`	—	lists of positive integers
`a`	—	a list of paths with coefficient; an element of `A`

Return Value: the module morphism from `P1` to `P2` which is determined by `a`.

Details:

- The parameters `P1` and `P2` are lists of positive integers which are points of the quiver of A. Such a list represents the projective A-module which is the sum of the indecomposable projective modules corresponding to points in the list.
- The format of an element of a path algebra and the format of a module homomorphism is described in the section Data types.

Example 1. Let A be the hereditary path algebra with quiver

$$\begin{tikzcd}
& 2 \arrow[r, "\alpha_2"] & 3 \arrow[dr, "\alpha_3"] & \\
1 \arrow[ur, "\alpha_1"] \arrow[dr, "\alpha_4"] & & & 6 \\
& 4 \arrow[r, "\alpha_5"] & 5 \arrow[ur, "\alpha_6"] &
\end{tikzcd}$$

over the field F of rational numbers.

```
>> F:= Dom::Rational:
   A:= pathalg::PathAlgebra(F, [6, [[1,2,1], [2,3,2], [3,6,3], [1,4,4],
   [4,5,5], [5,6,6]]]):
   N:= pathalg::Nontips(A, []):
```

Now we define the morphism of projective modules $\hom_A(\alpha_2\alpha_1, -) : P_3 \longrightarrow P_1$. First we have to input a MuPAD array with the only entry the element $\alpha_2\alpha_1$.

```
>> a:= array(1..1, 1..1): a[1,1]:= [[F(1), [1,2]]]: a

                     +-               -+
                     | [[1, [1, 2]]] |
                     +-               -+
```

Then we can pass this argument `a` to the function `pathalg::homOfProjectives`.

```
>> pathalg::homOfProjectives(A, [], N, [3], [1], a)

      --                                              +-     -+ --
      |   +- -+   +- -+   +- -+   +- -+   +- -+   |   1   |   |
      |   | 0 |,  | 0 |,  | 1 |,  | 0 |,  | 0 |,  |       |   |
      |   +- -+   +- -+   +- -+   +- -+   +- -+   |   0   |   |
      --                                              +-     -+ --
```

If we want to construct morphisms of decomposable modules we must define the maps as an array. Let us construct the map $P_3 \oplus P_6 \longrightarrow P_1 \oplus P_2$ corresponding to

$$b := \begin{pmatrix} \alpha_2\alpha_1 & \alpha_3\alpha_2\alpha_1 + \alpha_6\alpha_5\alpha_4 \\ \alpha_2 & \alpha_3\alpha_2 \end{pmatrix}.$$

First we define b.

```
>> b:= array(1..2, 1..2):

>> b[1,1]:= [[F(1), [1,2]]]:
   b[1,2]:= [[F(1), [1,2,3]], [F(1), [4,5,6]]]:
   b[2,1]:= [[F(1), [2]]]:
   b[2,2]:= [[F(1), [2,3]]]:

>> b

   +-                                                      -+
   |  [[1, [1, 2]]], [[1, [1, 2, 3]], [1, [4, 5, 6]]]  |
   |                                                        |
   |    [[1, [2]]],            [[1, [2, 3]]]               |
   +-                                                      -+
```

Now we construct the module morphism corresponding to b.

```
>> pathalg::homOfProjectives(A, [], N, [3,6], [1,2], b)

   --                                                   +-        -+ --
   |          +-    -+  +-    -+                     |   1, 1   |   |
   |  +- -+   |  0   |  |  1   |   +- -+   +- -+   |          |   |
   |  | 0 |,  |      |, |      |,  | 0 |,  | 0 |,  |   0, 1   |   |
   |  +- -+   |  0   |  |  1   |   +- -+   +- -+   |          |   |
   |          +-    -+  +-    -+                     |   1, 1   |   |
   --                                                   +-        -+ --
```

`pathalg::HomSpace` – the homomorphism space of two modules

`pathalg::HomSpace(A, M, N)` computes the homomorphism space of two `A`-modules `M` and `N` and returns the dimension and a basis of this space.

Call(s):

```
pathalg::HomSpace(A, M, N <, ReturnBasis = TRUE>)
```

Parameters:

`A` — a path algebra
`M, N` — two modules over `A`

Options:

`ReturnBasis` — `TRUE` if a basis of the homomorphism space should be returned, `FALSE` if not. The default is `TRUE`.

Return Value: a positive integer and, if `ReturnBasis` is `TRUE`, a list of module homomorphisms.

Related Functions: `pathalg::HomSpaceProjective`

Details:

- `pathalg::HomSpace` does not check whether `M` and `N` are really modules over `A`. To avoid problems use `pathalg::Module` to construct a module.
- If M is a projective module, please use the function `pathalg::HomSpaceProjective` instead of `pathalg::HomSpace` because it is faster.

Example 1. Let A be the hereditary algebra with quiver

$$1 \underset{\beta}{\overset{\alpha}{\rightrightarrows}} 2$$

over the finite field F with 25 elements.

```
>> F:= Dom::GaloisField(25):
   A:= pathalg::PathAlgebra( F, [2, [[1,2,1],[1,2,2]]] ):
```

We define two A-modules M and N as follows:

```
>> M:= pathalg::Module( A, [2,2], table(1=1, 2=[[1,0],[0,0]]) )
```

```
--         table(            --
|               +-      -+   |
|               |  1, 0  |   |
|           2 = |        |,  |
|               |  0, 0  |   |
|  [2, 2],      +-      -+   |
|               +-      -+   |
|               |  1, 0  |   |
|           1 = |        |   |
|               |  0, 1  |   |
|               +-      -+   |
--         )                 --
```

```
>> N:= pathalg::Module( A, [4,2], table(1=[[1,2,1,0],[0,4,3,2]], 2=2) )
```

```
--         table(                   --
|               +-             -+    |
|               |  2, 0, 0, 0  |     |
|           2 = |              |,    |
|               |  0, 2, 0, 0  |     |
|  [4, 2],      +-             -+    |
|               +-                -+ |
|               |  1,  2,  1, 0   |  |
|           1 = |                 |  |
|               |  0, -1, -2, 2   |  |
|               +-                -+ |
--         )                        --
```

Now we can compute the F-dimension of $\mathrm{Hom}_A(M, N)$.

```
>> pathalg::HomSpace(A, M, N, FALSE):
```

```
                4
```

Without the fourth argument we receive the dimension of $\mathrm{Hom}_A(M, N)$ and a basis:

```
>> pathalg::HomSpace(A, M, N)
```

```
    -- +-                          -+  +-                          -+
    |  |  +-       -+               |  |  +-       -+               |
    |  |  |  -2, 0  |               |  |  |  0, 0  |               |
    |  |  |         |  +-      -+   |  |  |         |  +-      -+   |
    |  |  |   0, 0  |  |  1, 0  |   |  |  |  0, 0  |  |  0, 1  |   |
 4, |  |  |         |, |        |   |, |  |         |, |        |   |, ...
    |  |  |  -2, 0  |  |  0, 0  |   |  |  |  0, 1  |  |  0, 0  |   |
    |  |  |         |  +-      -+   |  |  |         |  +-      -+   |
    |  |  |  -2, 0  |               |  |  |  0, 1  |               |
    |  |  +-       -+               |  |  +-       -+               |
    -- +-                          -+  +-                          -+
```

The default for ReturnBasis is `TRUE`, so the call

```
>> pathalg::HomSpace(A, M, N, TRUE)

    -- +-                          -+  +-                     -+
    |  |  +-       -+                |  |  +-       -+          |
    |  |  |  -2, 0  |                |  |  |  0, 0  |          |
    |  |  |         |  +-      -+    |  |  |        |  +-      -+  |
    |  |  |   0, 0  |  |  1, 0  |    |  |  |  0, 0  |  |  0, 1  |  |
 4, |  |  |         |, |        |    |, |  |        |, |        |  |, ...
    |  |  |  -2, 0  |  |  0, 0  |    |  |  |  0, 1  |  |  0, 0  |  |
    |  |  |         |  +-      -+    |  |  |        |  +-      -+  |
    |  |  |  -2, 0  |                |  |  |  0, 1  |          |
    |  |  +-       -+                |  |  +-       -+          |
    -- +-                          -+  +-                     -+
```

yields the same results as above.

`pathalg::HomSpaceProjective` – the homomorphism space of a projective and an arbitrary module

`pathalg::HomSpaceProjective(A, G, N, P, M)` returns a vector space basis of $\mathrm{Hom}_A(P, M)$ where P is a projective module over (A, G, N) and M is an arbitrary A-module.

Call(s):

`pathalg::HomSpaceProjective(A, G, N, P, M <, TRUE>)`

Parameters:

`A` — a path algebra
`G` — a Gröbner basis of an ideal I of `A`
`N` — a list of nontips with respect to I
`P` — a list of positive integers
`M` — a module over `A`

Options:

`TRUE` — return the projective module corresponding to `P`

Return Value: a non-negative integer and a list of module homomorphisms

Related Functions: `pathalg::HomSpace`

Details:

- The positive integers in the list `P` must be numbers of points in the quiver of `A`. The list defines a projective module as the direct sum of indecomposable projectives corresponding to the points in `P`.

- The projective module corresponding to `P` is returned if the optional argument is ***TRUE***.
- `pathalg::HomSpaceProjective` does not check whether `M` is really a module over `A`. The module `M` is best constructed using the function `pathalg::Module`.

Example 1. Let A be the hereditary algebra with quiver

$$\begin{tikzcd} 1 \arrow[dr, "\alpha"] & & 4 \\ & 3 \arrow[ur, "\gamma"] \arrow[dr, "\delta"] & \\ 2 \arrow[ur, "\beta"] & & 5 \end{tikzcd}$$

over the field `Dom::Rational`.

```
>> F:= Dom::Rational:
   A:= pathalg::PathAlgebra(F, [5, [[1,3,1], [2,3,2], [3,4,3], [3,5,4]]]):
   N:= pathalg::Nontips(A, []):
```

We define a module M over A:

```
>> M:= pathalg::Module(A, [2,1,1,0,3], table(1=[[1,2]], 2=3/2, 4=2))
```

```
--                  table(          --
|                        +-   -+     |
|                        |  2  |     |
|                        |     |     |
|                    4 = |  0  |,    |
|                        |     |     |
|                        |  0  |     |
|                        +-   -+     |
|  [2, 1, 1, 0, 3],      +- -+       |
|                    3 = | 0 |,      |
|                        +- -+       |
|                        +-    -+    |
|                    2 = | 3/2 |,    |
|                        +-    -+    |
|                        +-     -+   |
|                    1 = | 1, 2 |    |
|                        +-     -+   |
--                  )               --
```

Now we compute $\mathrm{Hom}_A(P, A)$ where P is the direct sum of the indecomposable projective modules P_1, P_2 and P_4.

```
>> pathalg::HomSpaceProjective(A, [], N, [1,2,4], M)
```

```
    -- --                                                 +-      -+ --
    |  |  +-   -+                                         |  2, 0  |  |
    |  |  |  1  |  +- -+  +-    -+  +-       -+  |        |  |
 3, |  |  |     |, | 0 |, | 1, 0 |, | 0, 0, 0 |, |  0, 0  |  |,
    |  |  |  0  |  +- -+  +-    -+  +-       -+  |        |  |
    |  |  +-   -+                                         |  0, 0  |  |
    -- --                                                 +-      -+ --

    --                                                 +-      -+ --
    |  +-   -+                                         |  4, 0  |  |
    |  |  0  |  +- -+  +-    -+  +-       -+  |        |  |
    |  |     |, | 0 |, | 2, 0 |, | 0, 0, 0 |, |  0, 0  |  |,
    |  |  1  |  +- -+  +-    -+  +-       -+  |        |  |
    |  +-   -+                                         |  0, 0  |  |
    --                                                 +-      -+ --

    --                                                      +-      -+ -- --
    |  +-   -+                                              |  0, 3  |  |  |
    |  |  0  |  +- -+  +-      -+  +-       -+  |        |  |  |
    |  |     |, | 1 |, | 0, 3/2 |, | 0, 0, 0 |, |  0, 0  |  |  |
    |  |  0  |  +- -+  +-      -+  +-       -+  |        |  |  |
    |  +-   -+                                              |  0, 0  |  |  |
    --                                                      +-      -+ -- --
```

Example 2. Let B be the algebra with quiver $1 \xrightarrow{\alpha_1} 2 \xrightarrow{\alpha_2} 3 \xrightarrow{\alpha_3} 4 \xrightarrow{\alpha_4} 5$ and relations $\alpha_3\alpha_2\alpha_1 = 0$, $\alpha_4\alpha_3\alpha_2 = 0$.

```
>> B:= pathalg::PathAlgebra( Dom::IntegerMod(11), [5, [[1,2,1], [2,3,2],
   [3,4,3], [4,5,4]]] ):
   G:= pathalg::GroebnerBasis(B, [ [[1, [1,2,3]]], [[1, [2,3,4]]] ]):
   N:= pathalg::Nontips(B, G):
```

We define a module M over B.

```
>> M:= pathalg::Module(B, [1,1,2,3,2], table(1=3, 2=1, 3=[[0,0],[0,1],[0,2]],
   4=4) ):
```

Now we compute the space of module homomorphisms $P \to M$ as well as the module $P = P_1 \oplus P_2$.

```
>> pathalg::HomSpaceProjective(B, G, N, [1,2], M, TRUE)

    -- --
    |  |                                                   +-                    -+
    |  |  +-         -+  +-                   -+  |  3 mod 11, 0 mod 11  |
 2, |  |  | 1 mod 11 |, | 3 mod 11, 0 mod 11 |, |                      |,
    |  |  +-         -+  +-                   -+  |  0 mod 11, 0 mod 11  |
    |  |                                                   +-                    -+
    -- --
```

```
+-          -+              --           --
|  0 mod 11  |  +-         -+  |           |
|            |  |  0 mod 11  |  |           |
|  0 mod 11  |, |            |  |,   ...    |,
|            |  |  0 mod 11  |  |           |
|  0 mod 11  |  +-         -+  |           |
+-          -+              --           --

--                  table(                         --
|                         +-        -+              |
|                     4 = | 0 mod 11 |,             |
|                         +-        -+              |
|                         +-                  -+    |
|                     3 = | 0 mod 11, 1 mod 11 |,   |
|                         +-                  -+    |
|  +-           -+        +-                    -+  |
|  | 1, 2, 2, 1, 0 |,      |  1 mod 11, 0 mod 11  |  |
|  +-           -+    2 = |                      |, |
|                         |  0 mod 11, 1 mod 11  |  |
|                         +-                    -+  |
|                         +-          -+            |
|                         |  1 mod 11  |            |
|                     1 = |            |            |
|                         |  0 mod 11  |            |
|                         +-          -+            |
--                      )                          --
```

(Parts of the output are omitted.)

`pathalg::incVerbosity` – increase the verbosity level

Call(s):

`pathalg::incVerbosity()`

Return Value: an integer; the current value of `pathalg_VERBOSITY`.

Related Functions: `pathalg::decVerbosity`, `pathalg::setVerbosity`

Details:

- The verbosity level is increased by 1. The amount of information messages printed depends on the function called.

Example 1. On loading the package pathalg the verbosity level is set to zero.

```
>> pathalg_VERBOSITY
                                        0
>> pathalg::incVerbosity()
                                        1
```

`pathalg::Injective` – construct an indecomposable injective module

Call(s):

`pathalg::Injective(A, G, N, x, <, Basis>)`

Parameters:

- `A` — a path algebra
- `G` — a Gröbner basis of an ideal I of `A`
- `N` — the set of nontips of I
- `x` — the number of an object of `A`

Options:

- *`Basis`* — Return a basis of the injective module

Return Value: the indecomposable injective module over `A` corresponding to the object `x`.

Side Effects: `pathalg::Injective` uses the remember mechanism.

Related Functions: `pathalg::Module`, `pathalg::Projective`, `pathalg::Simple`

Example 1. Let F be the field of complex numbers and let A be the hereditary path algebra with quiver $1 \xrightarrow{1} 2 \xrightarrow{2} 3$.

```
>> A:= PathAlgebra(Dom::Complex, [ 3, [[1,2,1],[2,3,2]] ]):
   N:= Nontips(A, []):
```

We construct the indecomposable injective module corresponding to the object 2 of A.

```
>> Injective(A, [], N, 2)

              --              table(       --
              |                   +- -+     |
              |  +-       -+    2 = | 0 |,  |
              |  | 1, 1, 0 |,       +- -+   |
              |  +-       -+        +- -+   |
              |                 1 = | 1 |   |
              |                     +- -+   |
              --              )            --
```

With the optional argument *Basis* we also receive an F-basis of the injective module.

```
>> Injective(A, [], N, 2, Basis)

                --              table(       --
                |                    +- -+    |
                |  +-        -+    2 = | 0 |,  |
                |  | 1, 1, 0 |,        +- -+    |,
                |  +-        -+        +- -+    |
                |                  1 = | 1 |    |
                |                      +- -+    |
                --              )            --

                      [[[1]], [2], []]
```

Background:

- The indecomposable injective modules of a spectroid A/I are the duals of the representable contravariant functors and thus are in bijection with the objects of A/I. For each x in A/I we have $Q_x := D((A/I)(-,x))$ where D is the usual duality functor $\mathrm{Hom}_k(-,k)$.

`pathalg::isTiltingComplex` – test if a complex satisfies the Toda condition

`pathalg::isTiltingComplex(A, G, N, [T1, T2, ...])` tests if the complex T given as the direct sum of the complexes of projectives `T1`, `T2`, ... satisfies the condition $\mathrm{Hom}(T, T[i]) = 0$ for $i \neq 0$ on a tilting complex in the homotopy category.

Call(s):

`pathalg::isTiltingComplex(A, G, N, L)`

Parameters:

`A` — a path algebra
`G` — a Gröbner basis of an ideal I of `A`
`N` — the list of nontips of I
`L` — a list of complexes of projectives

Return Value: either `TRUE` or `FALSE`

Side Effects: The information messages printed during the computations depend on the verbosity level `pathalg_VERBOSITY`.

Details:

- `pathalg::isTiltingComplex` returns `TRUE` if the direct sum T of the complexes in `L` satisfies $\mathrm{Hom}(T, T[i]) = 0$ for $i \neq 0$ in the homotopy category and `FALSE` otherwise.

Example 1. Let $A = FQ$ be the hereditary algebra with quiver Q

$$\begin{tikzcd} & & 3 \\ 1 \arrow[r, "\alpha"] & 2 \arrow[ur, "\beta"] \arrow[dr, "\gamma"'] & \\ & & 4 \end{tikzcd}$$

over the field $F =$ `Dom::Complex`.

```
>> F:= Dom::Complex: MC:= Dom::Matrix(F):
   A:= pathalg::PathAlgebra(F, [4, [[1,2,1], [1,3,2], [2,4,3], [3,4,4]]]):
   G:= pathalg::GroebnerBasis( A, [ [[1, [1,3]], [-1, [2,4]]] ] ):
   N:= pathalg::Nontips( A, G ):
```

We define the following four complexes of projectives over A.

```
>> C1:= [[0,0], table(0=[2]), table()]:
   C2:= [[0,0], table(0=[3]), table()]:
   C3:= [[-1,0], table(-1=[4], 0=[2,3]),
   table(-1=[MC(1,1), MC(1,1), MC(1,1), MC([-1,1])])]:
   C4:= [[0,0], table(0=[1]), table()]:
```

Now we test if $C_1 \oplus C_2 \oplus C_3 \oplus C_4$ can be a tilting complex in the homotopy category of chain complexes of projectives.

```
>> pathalg::isTiltingComplex(A, G, N, [C1, C2, C3, C4])

                                  TRUE
```

If we increase the verbosity level some information is printed about the homomorphism spaces which are computed.

```
>> pathalg::incVerbosity()

                                   1

>> pathalg::isTiltingComplex(A, G, N, [C1, C2, C3, C4])

        Computing dimension of Hom(C[1], C[3][1]) modulo homotopy...

        Computing dimension of Hom(C[2], C[3][1]) modulo homotopy...

       Computing dimension of Hom(C[3], C[1][-1]) modulo homotopy...
.
       Computing dimension of Hom(C[3], C[2][-1]) modulo homotopy...
.
       Computing dimension of Hom(C[3], C[3][-1]) modulo homotopy...
```

```
...
          Computing dimension of Hom(C[3], C[3][1]) modulo homotopy...

         Computing dimension of Hom(C[3], C[4][-1]) modulo homotopy...
....
          Computing dimension of Hom(C[4], C[3][1]) modulo homotopy...

                                    TRUE
```

`pathalg::Kernel` – the kernel of a module morphism

`pathalg::Kernel(A, f, M, N)` computes the kernel of the map $f : M \longrightarrow N$ and returns the kernel module as well as the kernel morphism.

Call(s):

`pathalg::Kernel(A, f, M, N)`

Parameters:

`A` — a path algebra
`f` — a module morphism `M` $\longrightarrow$ `N`
`M, N` — two modules over `A`

Return Value: a module and a module morphism

Related Functions: `pathalg::Cokernel`

Details:

- For a description of the format of a module or a module morphism cf section Data types.
- For performance reasons it is not checked whether f, M and N are valid input values.

Example 1. Let A be the hereditary algebra given by the quiver

$$\begin{tikzcd} & & & 4 \\ 1 \arrow[r, "\alpha"] & 2 \arrow[r, "\beta"] & 3 \arrow[ur, "\gamma"] \arrow[dr, "\delta"'] & \\ & & & 5 \end{tikzcd}$$

over the finite field on 3^5 elements.

```
>> F:= Dom::GaloisField(3,5):
   A:= pathalg::PathAlgebra(F, [5, [[1,2,1], [2,3,2], [3,4,3], [3,5,4]]]):
```

We define two modules M and N over A.

```
>> M:= pathalg::Module(A, [2,2,1,1,1], table(1=[[1,0],[1,0]], 2=[[1,2]],
   3=1)):
   N:= pathalg::Module(A, [3,4,2,2,1], table(1=1, 2=[[1,1,1,1],[0,1,2,1]],
   3=2, 4=[[1,1]]) ):
```

Now we define a module morphism $f : M \longrightarrow N$. This is a list of matrices representing the linear maps $f_i : M(i) \longrightarrow N(i)$. To make things easier, we first define a short cut for the MuPAD matrix constructor over F.

```
>> MC:= Dom::Matrix(F):
```

```
>> f:= [ MC(3,2), MC(4,2, [[-1,1],[-1,1]]), MC(2,1,[[1],[-1]]),
   MC(2,1,[[1],[-1]]), MC([1])]

              --              +-       -+                              --
              |  +-      -+   |  -1, 1  |                               |
              |  |  0, 0  |   |         |  +-    -+  +-    -+           |
              |  |        |   |  -1, 1  |  |  1   |  |  1   |  +- -+    |
              |  |  0, 0  |,  |         |, |      |, |      |, | 1 |    |
              |  |        |   |   0, 0  |  |  -1  |  |  -1  |  +- -+    |
              |  |  0, 0  |   |         |  +-    -+  +-    -+           |
              |  +-      -+   |   0, 0  |                               |
              --              +-       -+                              --
```

After defining f we compute its kernel.

```
>> pathalg::Kernel(A, f, M, N)

 --                  table(          --
 |                         +- -+      |
 |                     4 = | 0 |,     |
 |                         +- -+      |
 |                         +- -+      |
 |                     3 = | 0 |,     |
 |   [2, 1, 0, 0, 0],      +- -+      |,
 |                         +- -+      |
 |                     2 = | 0 |,     |
 |                         +- -+      |
 |                         +-    -+   |
 |                     1 = | 1, 0 |   |
 |                         +-    -+   |
 --                  )               --
```

```
+-                                            -+
|  +-      -+  +-   -+                         |
|  |  1, 0  |  |  1  |  +- -+  +- -+  +- -+    |
|  |        |, |     |, | 0 |, | 0 |, | 0 |    |
|  |  0, 1  |  |  1  |  +- -+  +- -+  +- -+    |
|  +-      -+  +-   -+                         |
+-                                            -+
```

The first return value is the kernel module $\mathrm{Ker}(f)$, the second return value is the kernel map $\ker(f)$: $\mathrm{Ker}(f) \longrightarrow M$ with the property that $f \circ \ker(f) = 0$.

`pathalg::Module` – define a module over a path algebra

Call(s):

`pathalg::Module(A, dim <, maps> <, G>)`

Parameters:

`A`	—	a path algebra in the internal format
`dim`	—	the dimension vector of the module as a list of positive integers
`maps`	—	a table with the linear maps corresponding to arrows of the quiver of `A`, the default value is `table()`
`G`	—	a Gröbner basis of an ideal of `A`

Return Value: A list `[dim, maps]` which represents a module.

Related Functions: `pathalg::Injective`, `pathalg::Projective`, `pathalg::Simple`

Details:

- Let M be a module, then $M(x)$ is a vector space over the base field of A for each object x in A, and $M(\alpha)$ is a linear map $M(x) \to M(y)$ (represented by a matrix) for each arrow $\alpha : x \to y$.
- The optional parameter `maps` is a table where entry i is the linear map corresponding to arrow i. `maps` is the empty table by default.
- If the optional parameter `G`, a Gröbner basis of an ideal I, is given, `pathalg::Module` also decides if the arguments really define a module over the path algebra A/I. Passing `G` means that also `maps` must be given as argument.
- An entry of the table `maps` can be one of the following:
  - the integer 1; then the corresponding map is given by the identity matrix
  - an element λ of the base field; then the corresponding matrix will be the matrix with λ on the diagonal and zero else.
  - a matrix over the base field or any object which can be converted into such a matrix
  - not defined; then the corresponding matrix will be zero

- Let $\alpha : x \to y$ be an arrow. If $M(x)$ has dimension zero and the dimension of $M(y)$ is n, then the $M(\alpha)$ is a $n \times 0$ matrix. However, this is not possible in MuPAD. As a workaround, $M(\alpha)$ is defined as the $n \times 1$ null matrix.

Example 1. Let A be the path algebra of the quiver

$$1 \xrightarrow{\alpha} 2 \xrightarrow{\beta} 3 \xrightarrow{\gamma} 4$$

over the ground field of complex numbers.

```
>> A:= pathalg::PathAlgebra(Dom::Complex, [4, [[1,2,1], [2,3,2], [3,4,3]] ]):
```

```
>> pathalg::Module(A, [2,2,1,3], table(1=1, 2=[[1,2]]))
```

```
               --             table(           --
               |                  +-   -+       |
               |                  | 0  |        |
               |                  |    |        |
               |              3 = | 0  |,       |
               |                  |    |        |
               |                  | 0  |        |
               |                  +-   -+       |
               |  [2, 2, 1, 3],   +-     -+     |
               |              2 = | 1, 2 |,     |
               |                  +-     -+     |
               |                  +-       -+   |
               |                  |  1, 0  |    |
               |              1 = |        |    |
               |                  |  0, 1  |    |
               |                  +-       -+   |
               --             )                --
```

```
>> pathalg::Module(A, [1,1,2,3], table(1=3, 2=1, 3=[[1,0],[0,1],[1,1]]))
```

```
               --             table(           --
               |                  +-       -+   |
               |                  |  1, 0  |    |
               |                  |        |    |
               |              3 = |  0, 1  |,   |
               |                  |        |    |
               |                  |  1, 1  |    |
               |                  +-       -+   |
               |  [1, 1, 2, 3],   +-   -+       |
               |                  | 1  |        |
               |              2 = |    |,       |
               |                  | 0  |        |
               |                  +-   -+       |
               |                  +- -+         |
               |              1 = | 3 |         |
               |                  +- -+         |
               --             )                --
```

We can define a semi-simple module by only specifying its dimension vector.

```
>> pathalg::Module(A, [2,1,1,2])

          --                table(           --
          |                       +-    -+    |
          |                       |  0  |     |
          |                   3 = |      |,   |
          |                       |  0  |     |
          |                       +-    -+    |
          |  [2, 1, 1, 2],        +- -+       |
          |                   2 = | 0 |,      |
          |                       +- -+       |
          |                       +-     -+   |
          |                   1 = | 0, 0 |    |
          |                       +-     -+   |
          --                )                --
```

Example 2. Now we define an ideal I of A, generated by $\beta\alpha$ and $\gamma\beta$ and consider the algebra A/I.

```
>> G:= pathalg::GroebnerBasis(A, [ [[1,[1,2]]], [[1,[2,3]]] ]):
```

Now we try to construct a module over A/I and pass the Gröbner basis to check the input data.

```
>> pathalg::Module(A, [2,2,1,3], table(1=1, 2=[[1,2]]), G)

 Error: Cannot convert arguments to a module over the input algebra.
 [pathalg::Module]
```

In this case, the function returns an error since the data did not define a module over A/I. In particular, the relation $\beta\alpha$ was not satisfied.

So we try with another set of data.

```
>> pathalg::Module(A, [2,2,2,3], table(1=[[0,1],[0,0]], 2=[[0,1],[0,0]],
   3=[[0,1],[0,-1],[0,4]]), G)

          --                table(              --
          |                       +-       -+    |
          |                       |  0,  1  |    |
          |                       |         |    |
          |                   3 = |  0, -1  |,   |
          |                       |         |    |
          |                       |  0,  4  |    |
          |                       +-       -+    |
          |                       +-      -+     |
          |  [2, 2, 2, 3],        |  0, 1  |     |
```

```
|                    2 = |         |,    |
|                        |  0, 0  |      |
|                        +-      -+      |
|                        +-      -+      |
|                        |  0, 1  |      |
|                    1 = |        |      |
|                        |  0, 0  |      |
|                        +-      -+      |
--                 )                    --
```

This time, we have successfully constructed a module over A/I.

`pathalg::NormalForm` – the normal form of an element

Call(s):

`pathalg::NormalForm(A, G, x)`

Parameters:

`A` — a path algebra
`G` — a list of elements which is a Gröbner basis of an ideal of `A`
`x` — an element of `A`

Return Value: an element of the path algebra `A`

Related Functions: `pathalg::GroebnerBasis`

Details:

- `pathalg::NormalForm` returns the normal form of an element of A/I with respect to a Gröbner basis of I.
- The format of a path algebra element is described in the section Data types.
- It is necessary that the coefficients occurring in `x` are elements of the ground field of `A`.

Example 1. We define an algebra A over `Dom::Complex` with quiver

$$\begin{tikzcd}
& 2 \arrow[r, "\alpha_2"] & 5 \arrow[dr, "\alpha_3"] & \\
1 \arrow[ur, "\alpha_1"] \arrow[r, "\alpha_4"] \arrow[dr, "\alpha_6"] & 3 \arrow[r, "\alpha_5"] & 6 \arrow[r, "\alpha_6"] & 8 \\
& 4 \arrow[r, "\alpha_8"] & 7 \arrow[ur, "\alpha_9"] &
\end{tikzcd}$$

and ideal generated by $\{\alpha_3\alpha_2\alpha_1 - \alpha_6\alpha_5\alpha_4 - \alpha_9\alpha_8\alpha_7\}$.

```
>> C:= Dom::Complex:
   A:= pathalg::PathAlgebra(Dom::Complex, [8, [[1,2,1], [2,5,2], [5,8,3],
   [1,3,4], [3,6,5], [6,8,6], [1,4,7], [4,7,8], [7,8,9]] ]):
   G:= pathalg::GroebnerBasis(A, [ [[1, [1,2,3]], [-1, [4,5,6]],
   [-1, [7,8,9]]] ]):
```

Now we compute the normal form of the element $a := \alpha_9\alpha_8\alpha_7$.

```
>> pathalg::NormalForm(A, G, [[C(1), [7,8,9]]])

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

Thus the normal form $N(a)$ is the element $\alpha_3\alpha_2\alpha_1 - \alpha_6\alpha_5\alpha_4$.

`pathalg::OpAlgebra` – the opposite algebra of a path algebra

Call(s):

`pathalg::OpAlgebra(A, <, G>)`

Parameters:

- `A` — a path algebra
- `G` — a list of elements in `A`, usually a Gröbner basis of an ideal of `A`

Return Value: the opposite algebra of `A` and, if an argument `G` was given, the converted elements of `G` as elements of the opposite algebra

Related Functions: `pathalg::PathAlgebra`, `pathalg::Quiver`

Details:

- For details on the data structure of a path algebra see section Data types.

Example 1. We consider the algebra $A = FQ$ where Q is the quiver

$$\begin{tikzcd}
 & & 6 \arrow[d, "5"'] & & \\
1 \arrow[r, "1"] & 2 \arrow[r, "2"] & 3 & 4 \arrow[l, "3"'] & 5 \arrow[l, "4"']
\end{tikzcd}$$

and F is the field `Dom::IntegerMod(5)`.

```
>> A:= pathalg::PathAlgebra( Dom::IntegerMod(5),
   [6, [[1,2,1], [2,3,2], [4,3,3], [5,4,4], [6,3,5]]] ):

>> pathalg::OpAlgebra(A)
```

```
                    table(
                      "arrowsfrom6" = [],
                      "arrowsfrom5" = [],
                      "arrowsfrom4" = [4],
                      "arrowsfrom3" = [2, 3, 5],
                      "arrowsfrom2" = [1],
                      "arrowsfrom1" = [],
                      "arrowlist" = [1, 2, 3, 4, 5],, []
                      "arrow5" = [3, 6],
                      "arrow4" = [4, 5],
                      "arrow3" = [3, 4],
                      "arrow2" = [3, 2],
                      "arrow1" = [2, 1],
                      "points" = 6,
                      "field" = Dom::IntegerMod(5)
                    )
```

Example 2. Let $A = FQ/I$ with Q the quiver

$$\begin{tikzcd} 1 \arrow[rr, shift left, "\alpha_1"] & & 2 \arrow[ll, shift left, "\alpha_2"] \arrow[dl, "\alpha_3"] \\ & 3 \arrow[ul, "\alpha_4"] & \end{tikzcd}$$

and the ideal I generated by the elements $\alpha_2\alpha_1\alpha_2\alpha_1$, $\alpha_4\alpha_3\alpha_1 - \alpha_2\alpha_1$ and $\alpha_1\alpha_2\alpha_1\alpha_4$ over the field $F =$ `Dom::Real`.

```
>> F:= Dom::Real:
   A:= pathalg::PathAlgebra(F, [3, [[1,2,1], [2,1,2], [2,3,3], [3,1,4]]]):
   G:= pathalg::GroebnerBasis( A, [ [[F(1), [1,2,1,2]]],
   [[F(1), [1,3,4]], [F(-1), [1,2]]], [[F(1), [4,1,2,1]]] ] ):

>> pathalg::OpAlgebra(A, G)

 table(
   "arrowsfrom3" = [3],
   "arrowsfrom2" = [1],
   "arrowsfrom1" = [2, 4],
   "arrowlist" = [1, 2, 3, 4],
   "arrow4" = [1, 3],          , [[[1, [2, 1, 2, 1]]],
   "arrow3" = [3, 2],
   "arrow2" = [1, 2],
   "arrow1" = [2, 1],
   "points" = 3,
   "field" = Dom::Real
```

```
)

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

Background:

- The opposite algebra of an algebra kQ/I is the path algebra of the opposite quiver Q^{op}, i.e. all arrows are reversed. The elements of the ideal I^{op} are the elements of I where all paths are in reverse order.

`pathalg::PathAlgebra` – define a path algebra over a field

Call(s):

```
pathalg::PathAlgebra(F, Q)
```

Parameters:

`F` — base field of the path algebra
`Q` — a finite quiver

Return Value: a table which represents a path algebra.

Related Functions: `pathalg::OpAlgebra`, `pathalg::Quiver`

Details:

- A finite quiver `Q` is a list `[N]` or `[N, l]`, where `N` is the number of points in `Q` and `l` is a list of arrows. An arrow is given by a list `[source, target, name]` of positive integers. As a name of an arrow only a positive integer is allowed.
- For details on the data structure of a quiver or a path algebra see section Data types.

Example 1. We define the path algebra corresponding to the quiver

$$\begin{tikzcd} & 2 \arrow[dr, "2"] & \\ 1 \arrow[ur, "1"] \arrow[dr, "3"'] & & 4 \\ & 3 \arrow[ur, "4"'] & \end{tikzcd}$$

over the base field `Dom::Complex` of complex numbers.

```
>> A:= pathalg::PathAlgebra(Dom::Complex, [4, [ [1,2,1], [2,4,2],
   [1,3,3], [3,4,4] ] ]);
```

```
                         table(
                           "arrowsfrom4" = [],
                           "arrowsfrom3" = [4],
                           "arrowsfrom2" = [2],
                           "arrowsfrom1" = [1, 3],
                           "arrowlist" = [1, 2, 3, 4],
                           "arrow4" = [3, 4],
                           "arrow3" = [1, 3],
                           "arrow2" = [2, 4],
                           "arrow1" = [1, 2],
                           "points" = 4,
                           "field" = Dom::Complex
                         )
```

Example 2. We can define path algebras over a finite field, here the Galois field with 3^5 elements. The quiver is semisimple (i.e. without arrows) on 5 points.

```
>> F:= Dom::GaloisField(3, 5);

      Dom::GaloisField(3, 5, poly(X1  - X1  + 1, [X1], IntMod(3)))

>> A:= pathalg::PathAlgebra(F, [5]);

  table(
    "arrowsfrom5" = [],
    "arrowsfrom4" = [],
    "arrowsfrom3" = [],
    "arrowsfrom2" = [],
    "arrowsfrom1" = [],
    "arrowlist" = [],
    "points" = 5,
                                              5     4
    "field" = Dom::GaloisField(3, 5, poly(X1  - X1  + 1, [X1], IntMod(3)))
     )
```

`pathalg::Projective` – construct an indecomposable projective module

Call(s):

`pathalg::Projective(A, G, N, x <, *Basis*>)`

Parameters:

`A` — a path algebra
`G` — a Gröbner basis of an ideal I of `A`
`N` — the set of nontips of I
`x` — the number of an object of `A`

Options:

Basis — Return a basis of the projective module

Return Value: the indecomposable projective module over `A` corresponding to the object `x`.

Side Effects: `pathalg::Projective` uses `option remember`.

Related Functions: `pathalg::Injective`, `pathalg::Module`, `pathalg::Simple`

Example 1. Let F be the Galois field on 49 elements and let $A = FQ/I$ be the algebra with quiver

$$\begin{tikzcd} & 2 \arrow[dr, "\alpha_2"] & \\ 1 \arrow[ur, "\alpha_1"] \arrow[dr, "\alpha_3"'] & & 4 \\ & 3 \arrow[ur, "\alpha_4"'] & \end{tikzcd}$$

and ideal I generated by $\alpha_2\alpha_1 - \alpha_4\alpha_3$.

```
>> F:= Dom::GaloisField(49):
   A:= pathalg::PathAlgebra(F, [4, [[1,2,1], [2,4,2], [1,3,3], [3,4,4]] ]):
   G:= pathalg::GroebnerBasis(A, [ [[F(1), [1,2]], [F(-1), [3,4]]] ]):
   N:= pathalg::Nontips(A, G):
```

We construct the indecomposable projective module corresponding to the object 1 of A.

```
>> pathalg::Projective(A, G, N, 1);
```

```
--                  table(       --
|                         +- -+   |
|                     4 = | 1 |,  |
|                         +- -+   |
|                         +- -+   |
|  +-           -+   2 = | 1 |,  |
|  | 1, 1, 1, 1 |,       +- -+   |
|  +-           -+       +- -+   |
|                    3 = | 1 |,  |
|                         +- -+   |
|                         +- -+   |
|                    1 = | 1 |   |
|                         +- -+   |
--                  )            --
```

Passing the optional argument *Basis* to `pathalg::Projective` we additionally receive an F-basis of the projective module.

```
>> pathalg::Projective(A, G, N, 3, Basis)

                     --                  table(       --
                     |                       +- -+    |
                     |                     4 = | 1 |,  |
                     |                       +- -+    |
                     |                       +- -+    |
                     |  +-          -+     2 = | 0 |,  |
                     |  | 0, 0, 1, 1 |,      +- -+    |,
                     |  +-          -+       +- -+    |
                     |                     3 = | 0 |,  |
                     |                       +- -+    |
                     |                       +- -+    |
                     |                     1 = | 0 |   |
                     |                       +- -+    |
                     --                  )            --

                        +-                -+
                        | [], [], [3], [[4]] |
                        +-                -+
```

Background:

- The indecomposable projective modules of a spectroid A/I are the representable covariant functors and thus are in bijection with the objects of A/I.

---

pathalg::ProjectiveComplex – construct a projective chain complex

`pathalg::ProjectiveComplex(A, G, N, [e1, e2, ...])` returns a projective complex of projectives.

Call(s):

`pathalg::ProjectiveComplex(A, G, N, L <, n = 0>)`

Parameters:

`A` — a path algebra
`G` — a Gröbner basis of an ideal I of `A`
`N` — the list of nontips of I
`L` — a list of objects of `A`
`n` — an integer; default value is 0

Return Value: the projective complex $0 \longrightarrow P = P \longrightarrow 0$ in degrees `n` and `n+1` where P is the projective `A`-module corresponding to the objects in the list `L`.

Related Functions: `pathalg::Projective`

Example 1. Let A be the hereditary algebra with quiver $2 \longleftarrow 1 \longrightarrow 3$ over the finite field on 256 elements.

```
>> F:= Dom::GaloisField(256):
   A:= pathalg::PathAlgebra(F, [3, [[1,2,1], [1,3,2]]] ):
   N:= pathalg::Nontips(A, []):
```

We define the projective complex $0 \to P_1 = P_1 \to 0$ in degrees 0 and 1.

```
>> pathalg::ProjectiveComplex(A, [], N, [1])

 --         table(
 |                --              table(       --
 |                |                     +- -+   |
 |                |  +-       -+    2 = | 1 |,  |
 |           1 =  |  | 1, 1, 1 |,       +- -+   |,
 |                |  +-       -+        +- -+   |
 |                |                 1 = | 1 |   |
 |                |                     +- -+   |
 |  [0, 1],       --              )            -- ,
 |                --              table(       --
 |                |                     +- -+   |
 |                |  +-       -+    2 = | 1 |,  |
 |           0 =  |  | 1, 1, 1 |,       +- -+   |
 |                |  +-       -+        +- -+   |
 |                |                 1 = | 1 |   |
 |                |                     +- -+   |
 |                --              )            --
 --         )

                                   --
                                    |
                                    |
                                    |
                                    |
  table(                            |
        +-                     -+   |
        |  +- -+  +- -+  +- -+  |   |
    0 = |  | 1 |, | 1 |, | 1 |  |   |
        |  +- -+  +- -+  +- -+  |   |
        +-                     -+   |
  )                                 |
                                    |
                                    |
                                    |
                                    |
                                    |
                                   --
```

We can also specify the lower bound of the projective complex. Here, we define the projective complex $0 \longrightarrow P_2 \oplus P_3 = P_2 \oplus P_3 \longrightarrow 0$ in degrees 4 and 5.

```
>> pathalg::ProjectiveComplex(A, [], N, [2,3], 4)

 --         table(                                          --
 |              --            table(       --              |
 |              |                 +- -+    |               |
 |              |  +-       -+  2 = | 0 |,  |               |
 |          5 = |  | 0, 1, 1 |,     +- -+    |,              |
 |              |  +-       -+      +- -+    |               |
 |              |               1 = | 0 |    |               |
 |              |                   +- -+    |               |
 |  [4, 5],     --              )            -- ,  . . .    |
 |              --            table(       --              |
 |              |                 +- -+    |               |
 |              |  +-       -+  2 = | 0 |,  |               |
 |          4 = |  | 0, 1, 1 |,     +- -+    |               |
 |              |  +-       -+      +- -+    |               |
 |              |               1 = | 0 |    |               |
 |              |                   +- -+    |               |
 |              --              )            --              |
 --         )                                               --
```

(Parts of the output are omitted.)

`pathalg::projectiveCover` – the projective cover of a module

`pathalg::projectiveCover(A, G, N, M)` constructs the projective cover P of the module M together with the projection $P \longrightarrow M$.

Call(s):

`pathalg::projectiveCover(A, G, N, M)`

Parameters:

- `A` — a path algebra
- `G` — a Gröbner basis of an ideal I of `A`
- `N` — the list of nontips with respect to I
- `M` — a module over `A`

Return Value: a list of positive integers, the projective cover P of M, and a module morphism $P \longrightarrow M$.

Related Functions: `pathalg::projectiveResolution`

Details:

- The first return value is a list of positive integers which represents a projective module as a sum of indecomposable projectives corresponding to objects in the list.
- The second return value is the projective cover P of the input module M.
- The third return value is a module morphism, the projection $P \longrightarrow M$. For a description of the format of module morphism confer the section Data types.

Example 1. We define the hereditary algebra A by the quiver

$$\begin{tikzcd} & 2 \\ 1 \arrow[ur, "\alpha"] \arrow[r, "\beta"] \arrow[dr, "\gamma"'] & 3 \\ & 4 \end{tikzcd}$$

over the field of complex numbers $\mathbb{C}$.

```
>> F:= Dom::Complex:
   A:= pathalg::PathAlgebra(F, [4, [[1,4,1], [2,4,2], [3,4,3]]]):
   N:= pathalg::Nontips(A, []):
```

We define a module S over A as the direct sum of the simple modules S_1 and S_2.

```
>> S:= pathalg::directSum(A, [pathalg::Simple(A,1), pathalg::Simple(A,2)])
```

Now we compute the projective cover of S.

```
>> pathalg::projectiveCover(A, [], N, S)
```

```
        --                table(         --
        |                     +- -+      |
        |                 3 = | 1 |,     |
        |                     +- -+      |
        |                     +- -+      |
        |  +-          -+   2 = | 1 |,   |
 [1, 2], |  | 1, 2, 1, 1 |,     +- -+      |,
        |  +-          -+     +-   -+    |
        |                     |  1  |    |
        |                 1 = |     |    |
        |                     |  0  |    |
        |                     +-   -+    |
        --                )              --
```

```
  +-                              -+
  |  +- -+  +-     -+  +- -+  +- -+  |
  |  | 1 |, | 0, 1 |, | 0 |, | 0 |  |
  |  +- -+  +-     -+  +- -+  +- -+  |
  +-                              -+
```

From the first return value `[1, 2]` it follows that the projective cover of S is the projective module $P_1 \oplus P_2$. The projection $P_1 \oplus P_2 \longrightarrow M$ is given by the last return value.

We define another module M over A:

```
>> M:= pathalg::Module(A, [2,2,3,1], table(1=1, 2=[[1,0],[0,I],[1,1]],
   3=[[1/2,5]]))
```

```
                  --              table(           --
                  |                   +-      -+    |
                  |                3 = | 1/2, 5 |,   |
                  |                   +-      -+    |
                  |                   +-      -+    |
                  |                   |  1, 0  |    |
                  |                   |        |    |
                  |                2 = |  0, I  |,   |
                  |  [2, 2, 3, 1],    |        |    |
                  |                   |  1, 1  |    |
                  |                   +-      -+    |
                  |                   +-      -+    |
                  |                   |  1, 0  |    |
                  |                1 = |        |    |
                  |                   |  0, 1  |    |
                  |                   +-      -+    |
                  --              )                --
```

. . . and compute its projective cover.

```
>> pathalg::projectiveCover(A, [], N, M)
```

```
            --                table(            --
            |                     +-      -+    |
            |                     |  1, 0  |    |
            |                  3 = |        |,   |
            |                     |  0, 1  |    |
            |                     +-      -+    |
            |                     +-      -+    |
            |                     |  1, 0  |    |
            |  +-          -+     |        |    |
 [1, 1, 3], |  | 2, 2, 3, 2 |,  2 = |  0, 1  |,   |,
            |  +-          -+     |        |    |
            |                     |  0, 0  |    |
            |                     +-      -+    |
            |                     +-      -+    |
```

```
         |                   |  1, 0  |     |
         |               1 = |        |     |
         |                   |  0, 1  |     |
         |                   +-      -+     |
         --             )                  --

+-                                                    -+
|                           +-         -+               |
|  +-      -+  +-      -+  |  0, 1, 0  |               |
|  |  0, 1  |  |  0, 1  |  |           |  +-        -+  |
|  |        |, |        |, |  I, 0, 0  |, |  5, 1/2  |  |
|  |  1, 0  |  |  1, 0  |  |           |  +-        -+  |
|  +-      -+  +-      -+  |  1, 1, 1  |               |
|                           +-         -+               |
+-                                                    -+
```

Thus the projective cover of M is the direct sum of the indecomposable projective modules P_1, P_1 and P_3.

`pathalg::projectiveResolution` – the projective resolution of a module

`pathalg::projectiveResolution(A, G, N, M)` computes a minimal projective resolution of the A-module M.

Call(s):

`pathalg::projectiveResolution(A, G, N, M <, Format><, maxlength>)`

Parameters:

- `A` — a path algebra
- `G` — a Gröbner basis of an ideal I of `A`
- `N` — the list of nontips with respect to I
- `M` — a module over `A`
- `Format` — one of *`Silent`*, *`PDim`*, *`All`*, *`Complex`*, *`ProjComplex`*
- `maxlength` — a positive integer

Options:

- *`All`* — returns the projective dimension of M, the projective modules in the resolution, the syzygy modules and the maps.
- *`Complex`* — returns the projective resolution of M as a complex
- *`PDim`* — returns the projective dimension of M
- *`ProjComplex`* — returns the projective resolution of M as a complex of projectives
- *`Silent`* — return value is null()

Return Value: a projective resolution of `M`, depending on the optional argument `Format`, or `FAIL` if the end of the resolution has not been reached.

Related Functions: `pathalg::projectiveCover`

Side Effects: The amount of information messages printed depends on the verbosity level `pathalg_VERBOSITY`.

Details:

- The optional argument `Format` has default value *PDim*.
- The optional argument `maxlength` has default value 5. It can be set to any positive integer. Then also `Format` has to specified.
- For a description of the format of a complex or a complex of projectives see the section Data types.
- The output returned by option *ProjComplex* can be used directly as input for the function `pathalg::EndAlgebraComplex`.
- The table of projective modules returned by *All* contains in entry $-i$ the projective module P_i where $0 \longrightarrow P_n \longrightarrow P_{n-1} \longrightarrow \cdots \longrightarrow P_2 \longrightarrow P_1 \longrightarrow P_0$ is a projective resolution of M.
- The table of syzygy modules returned by *All* contains in entry i (for $i = 0, 1, \dots$) the kernel Ω_i of the projection $P_i \longrightarrow \Omega_{i-1}$. Here, we define $\Omega_{-1} := M$.
- The projective resolution returned by `pathalg::projectiveResolution` is minimal.

Example 1. Let A be the algebra defined by the quiver

$$1 \xrightarrow{\alpha} 2 \xrightarrow{\beta} 3$$

and the ideal generated by $\beta\alpha$ over the base field $\mathbb{C}$.

```
>> A:= pathalg::PathAlgebra(Dom::Complex, [3, [[1,2,1], [2,3,2]]]):
   G:= pathalg::GroebnerBasis(A, [ [[1, [1,2]]] ]):
   N:= pathalg::Nontips(A, G):
```

We define the module M as the simple module corresponding to the object 1 of A:

```
>> M:= pathalg::Simple(A, 1)

                      --            table(        --
                      |                  +- -+    |
                      |               2 = | 0 |,  |
                      |  [1, 0, 0],      +- -+    |
                      |                  +- -+    |
                      |               1 = | 0 |   |
                      |                  +- -+    |
                      --            )             --
```

Now we compute a projective resolution of M.

```
>> pathalg::projectiveResolution(A, G, N, M)

                                        2
```

By default, only the projective dimension is returned. In this case, pdim $M = 2$. If we set the verbosity level to 1, we can watch the progress in the computations. This may be useful if the modules are very large.

```
>> pathalg::setVerbosity(1):
   pathalg::projectiveResolution(A, G, N, M)

                Projective cover P[0] of input module: Done.

                 Kernel K[0] of the projective cover: Done.

                                P[1]: Done.

                                K[1]: Done.

                                P[2]: Done.

                                K[2]: Done.

                  End of projective resolution reached.

                        Projective Dimension is 2.

                                    2
```

Example 2. If we choose the output format *All* we receive the projective dimension, the projective modules in the resolution, the syzygy modules, and the maps in the resolution.

```
>> pathalg::setVerbosity(0):
   pathalg::projectiveResolution(A, G, N, M, All)

               table(
                    --            table(       --
                    |                 +- -+    |
                    |             2 = | 0 |,   |
                2 = |  [0, 0, 0],     +- -+    |,
                    |                 +- -+    |
                    |             1 = | 0 |    |
                    |                 +- -+    |
                    --            )            --
```

```
                     --            table(       --
   table(            |                   +- -+   |
     -2 = [3],       |                 2 = | 0 |,  |
 2,  -1 = [2],,   1 = |  [0, 0, 1],         +- -+   |,,
     0 = [1]         |                   +- -+   |
   )                 |                 1 = | 0 |   |
                     |                   +- -+   |
                     --            )            --
                     --            table(       --
                     |                   +- -+   |
                     |                 2 = | 0 |,  |
                  0 = |  [0, 1, 0],         +- -+   |
                     |                   +- -+   |
                     |                 1 = | 0 |   |
                     |                   +- -+   |
                     --            )            --
                )

   table(
          +-                      -+
          |  +- -+  +- -+  +- -+  |
     -2 = |  | 0 |, | 0 |, | 1 |  |,
          |  +- -+  +- -+  +- -+  |
          +-                      -+
          +-                      -+
          |  +- -+  +- -+  +- -+  |
     -1 = |  | 0 |, | 1 |, | 0 |  |
          |  +- -+  +- -+  +- -+  |
          +-                      -+
   )
```

Example 3. If we choose the output format as *ProjComplex* we receive a complex of projectives.

```
>> pathalg::projectiveResolution(A, G, N, M, ProjComplex)
```

```
       --                       table(                         --
       |                               +-                   -+   |
       |                               |  +- -+  +- -+  +- -+  |   |
       |            table(         -2 = |  | 0 |, | 0 |, | 1 |  |,  |
       |              -2 = [3],         |  +- -+  +- -+  +- -+  |   |
       |  [-2, 0],    -1 = [2],,        +-                   -+   |
       |              0 = [1]           +-                   -+   |
       |            )                   |  +- -+  +- -+  +- -+  |   |
       |                           -1 = |  | 0 |, | 1 |, | 0 |  |   |
       |                                |  +- -+  +- -+  +- -+  |   |
       |                                +-                   -+   |
       --                       )                              --
```

Example 4. This time we compute the projective resolution of the simple module S_3 over A. As output format we choose *`Complex`*.

```
>> pathalg::projectiveResolution(A, G, N, pathalg::Simple(A,3), Complex)
```

```
--          table(                                          --
|                 --              table(        --           |
|                 |                   +- -+     |            |
|                 |  +-        -+    2 = | 0 |,  |            |
|  [0, 0],    0 = |  | 0, 0, 1 |,        +- -+   | , table()  |
|                 |  +-        -+        +- -+   |            |
|                 |                 1 = | 0 |   |            |
|                 |                     +- -+   |            |
|                 --              )             --           |
--          )                                               --
```

Example 5. Let A be the algebra $F[X]/(X^2)$ where $F := \mathbb{R}$. Then A is given by the quiver with one point and the loop X module the ideal generated by $\{X^2\}$.

```
>> A:= pathalg::PathAlgebra(Dom::Real, [1, [[1,1,1]]]):
   G:= pathalg::GroebnerBasis(A, [ [[1, [1,1]]] ]):
   N:= pathalg::Nontips(A, G):
```

We define the module S to be the (unique) simple module of A.

```
>> S:= pathalg::Simple(A, 1)
```

```
--      table(       --
|            +- -+    |
|  [1],   1 = | 0 |    |
|            +- -+    |
--      )            --
```

Now we try to compute a projective resolution of S.

```
>> pathalg::projectiveResolution(A, G, N, S)
```

```
Warning: Computation stopped, projective Dimension is at least 5.
[pathalg::projectiveResolution]

                                FAIL
```

It seems that the projective dimension of S is larger than 5, but the computation is stopped after `maxlength`. In this case `maxlength` had the default value 5. We can set it to a higher value.

```
>> pathalg::projectiveResolution(A, G, N, S, Silent, 10)
```

```
Warning: Computation stopped, projective Dimension is at least 10.
[pathalg::projectiveResolution]
```

Again the bound 10 on the maximum length of the projective resolution has been reached. It can be shown that S has infinite projective dimension.

`pathalg::Quiver` – define a quiver

Call(s):

`pathalg::Quiver(N <, L>)`

Parameters:

`N` — the number of points in the quiver
`L` — the list of arrows without names

Return Value: a quiver in the internal format.

Related Functions: `pathalg::OpAlgebra`, `pathalg::PathAlgebra`

Details:

- An arrow is defined as a list `[source, target]` where both entries are positive integers between `1` and `N`.
- If no arrow list is given, the quiver is assumed to be the semisimple quiver on `N` points and no arrows.
- A quiver has almost the same format as a path algebra except that there is no ground field defined. For details on the data structure of a quiver see section Data types.
- The return value `Q` of `pathalg::Quiver` can be changed easily to the data type of a `PathAlgebra` by inserting information about the ground field `F` into the table. This is done by the call `Q["field"]:= F`.

Example 1. We define a quiver on six points without arrows.

```
>> Quiver(6)
```

```
table(
  "arrowsfrom6" = [],
  "arrowsfrom5" = [],
  "arrowsfrom4" = [],
  "arrowsfrom3" = [],
  "arrowsfrom2" = [],
  "arrowsfrom1" = [],
  "arrowlist" = [],
  "points" = 6
)
```

Example 2. We first define the quiver Q

$$\begin{tikzcd} 1 \arrow[r] \arrow[dr] & 2 \\ 3 \arrow[r] \arrow[ur] & 4 \end{tikzcd}$$

and then change it to the path algebra FQ where F is the field of rational numbers.

```
>> A:= Quiver(4, [ [1,2], [1,4], [3,2], [3,4] ])

                    table(
                      "arrowsfrom4" = [],
                      "arrowsfrom3" = [3, 4],
                      "arrowsfrom2" = [],
                      "arrowsfrom1" = [1, 2],
                      "arrowlist" = [1, 2, 3, 4],
                      "arrow4" = [3, 4],
                      "arrow3" = [3, 2],
                      "arrow2" = [1, 4],
                      "arrow1" = [1, 2],
                      "points" = 4
                    )

>> A["field"]:= Dom::Rational

                             Dom::Rational

>> A

                    table(
                      "field" = Dom::Rational,
                      "arrowsfrom4" = [],
                      "arrowsfrom3" = [3, 4],
                      "arrowsfrom2" = [],
                      "arrowsfrom1" = [1, 2],
                      "arrowlist" = [1, 2, 3, 4],
                      "arrow4" = [3, 4],
                      "arrow3" = [3, 2],
                      "arrow2" = [1, 4],
                      "arrow1" = [1, 2],
                      "points" = 4
                    )
```

`pathalg::Radical` – **radical of the endomorphism algebra of a module**

`pathalg::Radical(A, M, EndM)` computes the radical of the endomorphism algebra `EndM` of the `A`-module `M`.

Call(s):

```
pathalg::Radical(A, M, EndM)
```

Parameters:

`A`	—	a path algebra
`M`	—	an `A`-module
`EndM`	—	a list of endomorphisms of `M` which form a basis of End(`M`)

Return Value: a list of endomorphisms of `M` which form a basis of the radical of End(`M`).

Related Functions: `pathalg::RadicalComplex`

Details:

- The characteristic of the ground field of A is arbitrary. There are applied different algorithms for zero characteristic and for non-zero characteristic.
- A basis of the endomorphism space of M can be obtained by calling `pathalg::HomSpace(A, M, M)`.

Example 1. We define an algebra given by the quiver

$$1 \underset{\beta}{\overset{\alpha}{\rightleftarrows}} 2$$

with ideal generated by $\{\beta\alpha\beta\alpha\beta\alpha\}$ over the field of complex numbers $\mathbb{C}$.

```
>> A:= pathalg::PathAlgebra(Dom::Complex, [2, [[1,2,1], [2,1,2]]]):
   G:= pathalg::GroebnerBasis(A, [ [[1, [1,2,1,2,1,2]]] ]):
   N:= pathalg::Nontips(A, G):
```

As a module we choose the indecomposable projective P corresponding to point 1.

```
>> P:= pathalg::Projective(A, G, N, 1):
```

Now we compute a $\mathbb{C}$-basis of the endomorphism space of P.

```
>> h:= pathalg::HomSpaceProjective(A, G, N, [1], P):
```

```
    -- -- +-          -+  +-          -+ --
    |  |  |  1, 0, 0  |  |  1, 0, 0  |  |
    |  |  |           |  |           |  |
 3, |  |  |  0, 1, 0  |, |  0, 1, 0  |  |,
    |  |  |           |  |           |  |
    |  |  |  0, 0, 1  |  |  0, 0, 1  |  |
    -- -- +-          -+  +-          -+ --
    -- +-          -+  +-          -+ --
    |  |  0, 0, 0  |  |  0, 0, 0  |  |
    |  |           |  |           |  |
    |  |  1, 0, 0  |, |  1, 0, 0  |  |,
    |  |           |  |           |  |
    |  |  0, 1, 0  |  |  0, 1, 0  |  |
    -- +-          -+  +-          -+ --
    -- +-          -+  +-          -+ -- --
    |  |  0, 0, 0  |  |  0, 0, 0  |  |  |
    |  |           |  |           |  |  |
    |  |  0, 0, 0  |, |  0, 0, 0  |  |  |
    |  |           |  |           |  |  |
    |  |  1, 0, 0  |  |  1, 0, 0  |  |  |
    -- +-          -+  +-          -+ -- --
```

We take the second return value as input for the call of `pathalg::Radical`.

```
>> pathalg::Radical(A, P, h[2])
```

```
   -- +-                            -+
   |  |  +-          -+  +-          -+  |
   |  |  |  0, 0, 0  |  |  0, 0, 0  |  |
   |  |  |           |  |           |  |
   |  |  |  1, 0, 0  |, |  1, 0, 0  |  |,
   |  |  |           |  |           |  |
   |  |  |  0, 1, 0  |  |  0, 1, 0  |  |
   |  |  +-          -+  +-          -+  |
   -- +-                            -+
    +-                            -+ --
    |  +-          -+  +-          -+  |  |
    |  |  0, 0, 0  |  |  0, 0, 0  |  |  |
    |  |           |  |           |  |  |
    |  |  0, 0, 0  |, |  0, 0, 0  |  |  |
    |  |           |  |           |  |  |
    |  |  1, 0, 0  |  |  1, 0, 0  |  |  |
    |  +-          -+  +-          -+  |  |
    +-                            -+ --
```

Background:

- The functions `pathalg::Radical` and `pathalg::RadicalComplex` are based on an algorithm for finding the radical of a matrix algebra due to A. Cohen, G. Ivanyos and D. Wales[1].

[1] A. Cohen, G. Ivanyos, D. Wales, Finding the radical of an algebra of linear transformations, *Journal of Pure and Applied Algebra 117 & 118 (1997) 177-193*

`pathalg::RadicalComplex` – radical of the endomorphism algebra of a complex

`pathalg::RadicalComplex(A, C, EndC)` computes the radical of the endomorphism algebra of the chain complex `C` over `A`.

Call(s):

`pathalg::RadicalComplex(A, C, EndC)`

Parameters:

`A`	—	a path algebra
`C`	—	a chain complex
`EndC`	—	a list of endomorphisms of `C` which form a basis of End(`C`)

Return Value: a list of endomorphisms of `C` which form a basis of the radical of End(`C`).

Related Functions: `pathalg::Radical`

Details:

- The characteristic of the base field of A is arbitrary. There are applied different algorithms for zero characteristic and for non-zero characteristic.
- A basis of the endomorphism space of C can be obtained by calling the library function `pathalg::HomComplex(A, C, C)`.

Example 1. We define the algebra A by the quiver

$$1 \underset{\beta}{\overset{\alpha}{\rightleftarrows}} 2$$

with ideal generated by $\{\beta\alpha\beta\alpha\beta\alpha\}$ over the finite field $\mathbb{F}_2$.

```
>> F:= Dom::IntegerMod(2):
   A:= pathalg::PathAlgebra(F, [2, [[1,2,1], [2,1,2]]]):
   G:= pathalg::GroebnerBasis(A, [ [[1, [1,2,1,2,1,2]]] ]):
   N:= pathalg::Nontips(A, G):
```

We define a complex of projectives

$$C = 0 \longrightarrow P_2 \xrightarrow{\begin{pmatrix} \alpha\beta\alpha \\ \alpha\beta\alpha\beta\alpha \end{pmatrix}} P_1 \oplus P_1 \longrightarrow 0$$

in degrees 0 and 1.

```
>> a:= array(1..2,1..1):
   a[1,1]:= [[F(1), [1,2,1]]]: a[2,1]:= [[F(1), [1,2,1,2,1]]]:
```

```
>> C:= pathalg::ComplexOfProjectives(A, G, N, [0,1], table(0=[2], 1=[1,1]),
   table(0=a)):
```

Before calling `pathalg::RadicalComplex` we must compute a basis of the endomorphism space of C.

```
>> EndC:= pathalg::HomComplex(A, C, C):
```

Now we increase the verbosity to get some information about the computations going on.

```
>> pathalg::incVerbosity()

                                        1
```

And finally we can start the computation of a basis of the radical of $\mathrm{End}(C)$.

```
>> pathalg::RadicalComplex(A, C, EndC[2]):

                                  Matrix size is 19.

                                      Step 1:

                      Determining linear equation, size: (12, 12).
 1 2 3 4 5 6 7 8 9 10 11 12
                               Computing solution space.

                                       Done.

                                      Step 2:

                      Determining linear equation, size: (12, 11).
 1 2 3 4 5 6 7 8 9 10 11
                               Computing solution space.

                                       Done.

                                      Step 4:

                      Determining linear equation, size: (11, 10).
 1 2 3 4 5 6 7 8 9 10
                               Computing solution space.

                                       Done.

                                      Step 8:
```

```
                   Determining linear equation, size: (11, 10).
 1 2 3 4 5 6 7 8 9 10
                            Computing solution space.

                                      Done.

                                     Step 16:

                   Determining linear equation, size: (11, 10).
 1 2 3 4 5 6 7 8 9 10
                            Computing solution space.

                                      Done.
```

The steps that have to be computed are numbered by p^i for $i = 0, 1, 2, \ldots$ until p^i is larger than the matrix size printed in the first line; p is the characteristic of the base field. If $p = 0$ then another algorithm is applied (cf the example for `pathalg::Radical`).

In this case, the matrix size is 19, so the algorithm goes through the steps 2^0, 2^1, 2^2, 2^3, and 2^4.

Background:

- The functions `pathalg::RadicalComplex` and `pathalg::RadicalComplex` are based on an algorithm for finding the radical of a matrix algebra due to A. Cohen, G. Ivanyos and D. Wales (confer the documentation of `pathalg::Radical`).

`pathalg::setVerbosity` – set the verbosity level

Call(s):

`pathalg::setVerbosity(<n = 0>)`

Parameters:

`n` — an integer; the new value of `pathalg_VERBOSITY`

Return Value: an integer; the current value of `pathalg_VERBOSITY`.

Related Functions: `pathalg::incVerbosity`, `pathalg::setVerbosity`

Details:

- The default value of `n` is 0.
- The verbosity level is set to `n`. The amount of information messages printed depends on the function called.

Example 1. On loading the package pathalg the verbosity level is set to zero.

```
>> pathalg_VERBOSITY
```

0

If `pathalg::setVerbosity` is called with an argument, the verbosity level is set to this argument.

```
>> pathalg::setVerbosity(5)
```

5

Calling `pathalg::setVerbosity` without an argument resets the verbosity level to 0.

```
>> pathalg::setVerbosity()
```

0

`pathalg::Shift` – apply the shift functor to a complex

`pathalg::Shift(C, n)` applies the shift functor to the chain complex `C` and returns the complex `C[n]`.

Call(s):

`pathalg::Shift(C <, n = 1>)`

Parameters:

`C` — a chain complex or a complex of projectives
`n` — an integer; the default value is 1

Return Value: the shifted complex `C[n]`.

Details:

- For details on the format of a chain complex or a complex of projectives see section Data types.

Example 1. Let A be the hereditary algebra with the quiver $2 \longleftarrow 1 \longrightarrow 3$ over the finite field on 5^2 elements.

```
>> F:= Dom::GaloisField(5,2): X:= F::Variable:
   A:= pathalg::PathAlgebra(F, [3, [[1,2,1], [1,3,2]]]):
   N:= pathalg::Nontips(A, []):
```

We define the complex

$$0 \longrightarrow C_0 \xrightarrow{d_0} C_1 \xrightarrow{d_1} C_2 \longrightarrow 0$$

in degrees 0, 1 and 2 as follows:

```
>> C0:= pathalg::Module(A, [1,2,1], table(1=[[1],[1]], 2=X)):
   C1:= pathalg::Module(A, [2,2,3], table(1=[[1,0],[X,2]],
   2=[[X+1, 2], [0,1], [3, X+2]])):
   C2:= pathalg::Module(A, [2,0,1]):
```

```
>> M:= Dom::Matrix(F):
```

```
>> d0:= [M([2*X,0]), M([[2*X,0], [2*X-2, 0]]), M([2*X+2,0,1])]
```

```
               --                           +-           -+ --
               |  +-      -+  +-           -+  |  2 X + 2   |  |
               |  |  2 X   |  |    2 X,  0  |  |            |  |
               |  |        |, |             |, |      0     |  |
               |  |    0   |  |  2 X - 2, 0 |  |            |  |
               |  +-      -+  +-           -+  |      1     |  |
               --                           +-           -+ --
```

```
>> d1:= [M(2,2), M(1,2), M([[X - 2, 2*X + 2, 1]])]
```

```
               -- +-       -+                                --
               |  |  0, 0  |  +-    -+  +-                -+  |
               |  |        |, | 0, 0 |, | X - 2, 2 X + 2, 1 |  |
               |  |  0, 0  |  +-    -+  +-                -+  |
               -- +-       -+                                --
```

```
>> C:= [[0,2], table(0=C0, 1=C1, 2=C2), table(0=d0, 1=d1)]:
```

Now we apply the shift functor to C.

```
>> SC:= pathalg::Shift(C): SC[1]
```

```
                                   [1, 3]
```

The return value `SC` is the shifted complex `C[1]`. The output was omitted apart from the first entry which is the list of bounds. So the shifted complex is defined in degrees $1, 2, 3$. Let us shift C some more.

```
>> SC:= pathalg::Shift(C, -5): SC[1]
```

```
                                  [-5, -3]
```

`pathalg::Simple` – **construct a simple module**

`pathalg::Simple(A, x)` constructs the simple module corresponding to the object `x` of a path algebra `A`.

Call(s):

`pathalg::Simple(A, x)`

Parameters:

`A` — a path algebra
`x` — the number of an object of `A`

Return Value: a module over `A`.

Related Functions: `pathalg::Injective`, `pathalg::Module`, `pathalg::Projective`

Details:

- For details on the format of a module see section Data types.

Example 1. Let A be the hereditary algebra over the rational numbers with the quiver

$$\begin{tikzcd} & & & 4 \\ 1 \arrow[r, "1"] & 2 \arrow[r, "2"] & 3 \arrow[ur, "3"] \arrow[dr, "4"'] & \\ & & & 5 \end{tikzcd}$$

```
>> F:= Dom::Rational:
   A:= pathalg::PathAlgebra(F, [5, [[1,2,1], [2,3,2], [3,4,3], [3,5,4]] ]):
```

We construct the simple modules S_1 and S_2 over A.

```
>> pathalg::Simple(A, 1)
```

```
--                   table(        --
|                          +-  -+   |
|                      4 = | 0 |,   |
|                          +-  -+   |
|                          +-  -+   |
|                      3 = | 0 |,   |
|  [1, 0, 0, 0, 0],        +-  -+   |
|                          +-  -+   |
|                      2 = | 0 |,   |
|                          +-  -+   |
|                          +-  -+   |
|                      1 = | 0 |    |
|                          +-  -+   |
--                   )             --
```

```
>> pathalg::Simple(A, 2)
```

```
--                  table(         --
|                         +- -+     |
|                     4 = | 0 |,    |
|                         +- -+     |
|                         +- -+     |
|                     3 = | 0 |,    |
|  [0, 1, 0, 0, 0],       +- -+     |
|                         +- -+     |
|                     2 = | 0 |,    |
|                         +- -+     |
|                         +- -+     |
|                     1 = | 0 |     |
|                         +- -+     |
--                  )              --
```

Background:

- The simple module S_x which corresponds to an object x of A looks as follows: $S_x(y)$ is the base field for $y = x$ and 0 otherwise. The maps $S(\alpha)$ are zero for any arrow α in A.

`pathalg::tau` – compute the AUSLANDER-REITEN translate of a module

`pathalg::tau(A, G, N, M)` computes the AUSLANDER-REITEN translate τM of the module M over the algebra A.

`pathalg::tau(A, G, N, M, n)` computes the n-th AUSLANDER-REITEN translate $\tau^n M$ of the module M over A.

Call(s):

`pathalg::tau(A, G, N, M <, n = 1>)`

Parameters:

`A` — a path algebra
`G` — a Gröbner basis of an ideal I of A
`N` — the list of nontips of I
`M` — a module over A
`n` — an integer

Return Value: the module $\tau^n M$ or FAIL if $\tau^n M$ does not exist.

Side Effects: The amount of information messages printed depends on the verbosity level `pathalg_VERBOSITY`.

Details:

- The default value of n is 1.
- If $n = 0$ then M is returned.
- For negative values of n the inverse of τ is applied. This is equivalent to the function call `pathalg::tau_inverse(A, G, N, M, -n)`.
- For details on the format of a module see section Data types.

Example 1. We define A as the hereditary algebra with quiver

$$\begin{tikzcd}
1 \arrow[dr, "\alpha"] & & 4 \\
& 3 \arrow[ur, "\gamma"] \arrow[dr, "\delta"] & \\
2 \arrow[ur, "\beta"] & & 5
\end{tikzcd}$$

```
>> A:= pathalg::PathAlgebra(Dom::Rational, [5, [[1,3,1], [2,3,2], [3,4,3],
   [3,5,4]]]): G:= []: N:= pathalg::Nontips(A, G):
```

Let M be the indecomposable injective module I_5.

```
>> M:= pathalg::Injective(A, G, N, 5):
```

We compute the Auslander-Reiten translate τM of M.

```
>> pathalg::tau(A, G, N, M)
```

```
--                      table(          --
|                            +-    -+    |
|                        4 = | 0, 0 |,   |
|                            +-    -+    |
|                            +-    -+    |
|                        3 = | 0, 1 |,   |
|                            +-    -+    |
|                            +-    -+    |
|  [1, 1, 2, 1, 0],          |  -1  |    |
|                        2 = |      |,   |
|                            |   1  |    |
|                            +-    -+    |
|                            +-   -+     |
|                            |  0  |     |
|                        1 = |     |     |
|                            |  1  |     |
|                            +-   -+     |
--                      )               --
```

By specifying the optional argument we can compute $\tau^4 M$.

```
>> pathalg::tau(A, G, N, M, 4)
```

```
--                table(                   --
|                      +-              -+   |
|                      |  0, 0, 1, 0, 0  |   |
|                      |                 |   |
|                  4 = |  0, 0, 0, 1, 0  |,  |
|                      |                 |   |
|                      |  0, 0, 0, 0, 1  |   |
|                      +-              -+   |
|                      +-              -+   |
|                      |  1, 0, 0, 0, 0  |   |
|                  3 = |                 |,  |
|                      |  0, 1, 0, 0, 0  |   |
|                      +-              -+   |
|                      +-          -+        |
|                      |  -1,  0, 0  |       |
|                      |             |       |
|                      |   0, -1, 0  |       |
|  [3, 3, 5, 2, 3],    |             |       |
|                  2 = |   1,  0, 0  |,      |
|                      |             |       |
|                      |   0,  1, 0  |       |
|                      |             |       |
|                      |   0,  0, 1  |       |
|                      +-          -+        |
|                      +-         -+         |
|                      |  0, 0,  1 |         |
|                      |           |         |
|                      |  0, 1, -1 |         |
|                      |           |         |
|                  1 = |  1, 0,  0 |         |
|                      |           |         |
|                      |  0, 1,  0 |         |
|                      |           |         |
|                      |  0, 0,  1 |         |
|                      +-         -+         |
--                )                        --
```

We define another module P as the indecomposable projective module P_2.

```
>> P:= pathalg::Projective(A, G, N, 2):
```

Since P is projective, the AUSLANDER-REITEN translate does not exist and `pathalg::tau` returns `FAIL`.

```
>> pathalg::tau(A, G, N, P)
```

```
FAIL
```

But we can compute $\tau^{-1}P$ since P is not injective. This is done by specifying -1 as the optional argument.

```
>> pathalg::tau(A, G, N, P, -1)

  --                     table(              --
  |                            +-       -+    |
  |                        4 = | -1, 1 |,     |
  |                            +-       -+    |
  |                            +-      -+     |
  |                        3 = | 1, 0 |,      |
  |                            +-      -+     |
  |                            +-    -+       |
  |  [1, 0, 2, 1, 1],          |  0   |       |
  |                        2 = |      |,      |
  |                            |  0   |       |
  |                            +-    -+       |
  |                            +-    -+       |
  |                            |  1   |       |
  |                        1 = |      |       |
  |                            |  0   |       |
  |                            +-    -+       |
  --                     )                   --
```

Background:

- Let M be an indecomposable module which is not projective. Then there exists an AUSLANDER-REITEN sequence $0 \to \tau M \to E \to M \to 0$ and τM is unique up to isomorphism.
- Let M be an indecomposable module which is not injective. Then there exists an AUSLANDER-REITEN sequence $0 \to M \to E \to \tau^{-1}M \to 0$ and $\tau^{-1}M$ is unique up to isomorphism.

`pathalg::tipreduce` – tip reduce a list of algebra elements

`pathalg::tipreduce(A, [a1, a2, ...])` tip reduces a list of elements `a1`, `a2`, … in `A`.

Call(s):

`pathalg::tipreduce(A, L)`

Parameters:

`A` — a path algebra
`L` — a list of elements of a path algebra

Return Value: a tip reduced list of elements of `A`

Related Functions: `pathalg::GroebnerBasis`, `pathalg::Tips`

Details:

- The coefficients occuring in `L` must be elements of the ground field of `A`.
- The format of a path algebra element is described in the section Data types.

Example 1. Let A be the path algebra with quiver

$$\begin{tikzcd} 1 \arrow[r, shift left, "\alpha_1"] & 2 \arrow[l, shift left, "\alpha_4"] \arrow[r, shift left, "\alpha_2"] & 4 \arrow[l, shift left, "\alpha_3"] \end{tikzcd}$$

```
>> F:= Dom::IntegerMod(11):
   A:= pathalg::PathAlgebra(F, [3, [[1,2,1],[2,3,2],[3,2,3],[2,1,4]] ]):
```

We define a list of elements in A and apply tip reduction.

```
>> L:= [ [[F(1), [1,2,3,4,1,4,1,4]]],
   [[F(1), [1,4,1,4]], [F(3), [1,2,3,4]]],
   [[F(3), [2,3,2,3,2,3]], [F(-3),[4,1,2,3]]], [[F(1), [2,3]]] ];

 [[[1 mod 11, [1, 2, 3, 4, 1, 4, 1, 4]]],

    [[1 mod 11, [1, 4, 1, 4]], [3 mod 11, [1, 2, 3, 4]]],

    [[3 mod 11, [2, 3, 2, 3, 2, 3]], [8 mod 11, [4, 1, 2, 3]]],

    [[1 mod 11, [2, 3]]]]

>> pathalg::tipreduce(A, L)

 [[[1 mod 11, [1, 4, 1, 4]], [3 mod 11, [1, 2, 3, 4]]],

    [[1 mod 11, [2, 3]]]]
```

Background:

- A set of elements of a path algebra is called *tip reduced* if $\mathrm{Tip}(a)$ does not divide $\mathrm{Tip}(b)$ for any two elements $a \neq b$ in A.

`pathalg::Tips` – compute the tips of elements

Call(s):

```
pathalg::Tips(a1, a2, ...)
```

Parameters:

`a1, a2, ...` — elements of a path algebra

Return Value: the tips of the elements `a1, a2, ...`

Related Functions: `pathalg::GroebnerBasis`, `pathalg::tipreduce`

Details:

- The tips are computed with respect to the length-lexicographic order. The arrow names are non-negative integers and they are ordered in the usual order of integers.
- The format of a path algebra element is described in the section Data types.

Example 1. We define two elements over the hereditary path algebra with quiver

$$\begin{tikzcd} 1 \arrow[r, "1"] & 2 \arrow[r, "2"] \arrow[d, "4"] \arrow[dr, "5"] & 3 \\ 4 \arrow[ur, "3"] \arrow[r, "6"] & 5 \arrow[r, "7"] & 6 \end{tikzcd}$$

```
>> pathalg::PathAlgebra(Dom::Complex, [6, [[1,2,1],[2,3,2],[4,2,3],[2,5,4],
   [2,6,5],[4,5,6],[5,6,7]]]):
```

```
>> a1:= [[1,[1,4,7]], [2,[1,5]]]; a2:= [[5,[3,5]], [1,[6,7]]];
```

```
[[1, [1, 4, 7]], [2, [1, 5]]]

[[5, [3, 5]], [1, [6, 7]]]
```

and compute their tips.

```
>> pathalg::Tips(a1, a2)
```

```
[1, [1, 4, 7]], [1, [6, 7]]
```

Background:

- Suppose that B is a multiplicative basis of A as a k-vector space and $<$ is an admissible order on B. The tip of an element $a \in A$ is defined as follows: Let $a = \sum_{i=1}^{n} \lambda_i b_i$ with $0 \neq \lambda_i \in k$ and $b_i \in B$. Then b_j is the tip of a if $b_i \leq b_j$ for all $i = 1, \ldots, n$.

Library functions