CREP Manual
Part IV

Peter Dräxler
Rainer Nörenberg
(editors)

Contents

Introduction

With this volume of the Crep Manual, we present three packages.

The first one called `ufstrong` by M. Barot and J. A. de la Peña from UNAM, Mexico deals with unit forms and (strong) positive and non negative definiteness.

The other two packages both are concerned with counting representations over finite fields. One of these, by Jiuzhao Hua, called `nor` does so for reasonably small dimension vectors over certain finite quivers. The other one, by R. Nörenberg, deals with Hall polynomials over an algebra of type E_8.

Our thanks go to the authors of the packages for their contributions to the system.

Crep manuals:

1 *The Crep Manual*, Ergänzungsreihe 96-002, Sonderforschungsbereich 343, Diskrete Strukturen in der Mathematik, Universität Bielefeld, 1996

2 *The Crep Manual, Part II*, Ergänzungsreihe 97-009, Sonderforschungsbereich 343, Diskrete Strukturen in der Mathematik, Universität Bielefeld, 1997

3 *The Crep Manual, Part III*, Ergänzungsreihe 99-007, Sonderforschungsbereich 343, Diskrete Strukturen in der Mathematik, Universität Bielefeld, 1997

Crep can be obtained via `ftp` from the server
`ftp.uni-bielefeld.de`
where it can be found in the directory
`pub/math/f-d-alg/crep`
or via the web page:
`www.mathematik.uni-bielefeld.de/birep/crep/ftpindex.html`

Running Crep will require the Maple system, preferably MapleV release 4 or later.
For installation of some packages of Crep, a Pascal compiler or a C/C++ compiler will be needed.
In particular, the `ufstrong` package described in this volume will require a C++ compiler.

Questions or remarks on Crep may be directed at
`fdowner@mathematik.uni-bielefeld.de`

1 The ufstrong package
by M. Barot and J. A. de la Peña

The ufstrong package contains functions dealing mainly with unit forms, some of them may also be used in a more general context. The package is loaded with the read command:

```
read(‘ufstrong.src‘):
```

1.1 ufs_decompose - decompose a matrix into blocks

Calling sequence:
`ufs_decompose(`M`)`
Parameters:
M – a matrix
Synopsis:

- `ufs_decompose` decomposes the given matrix M into blocks $B_1, \ldots, B_t$, the smallest matrices such that M is equivalent (where the equivalence is obtained by row and column permutations) to the matrix with $B_1, \ldots, B_t$ in the diagonal and zero entries elsewhere.
- `ufs_decompose` returns a list of matrices.

Example:

```
> A:=matrix([[0,1,0,1,0,0],[0,0,1,0,0,1],[1,0,0,0,1,0],[0,0,0,0,0,1],
> [0,1,0,0,0,0]]);

                          [0    1    0    1    0    0]
                          [                         ]
                          [0    0    1    0    0    1]
                          [                         ]
                     A := [1    0    0    0    1    0]
                          [                         ]
                          [0    0    0    0    0    1]
                          [                         ]
                          [0    1    0    0    0    0]

> ufs_decompose(A);

                      [[1    1]  [1    1]          ]
                      [[       ], [       ], [1    1]]
                      [[1    0]  [0    1]          ]
```

1.2 ufs_diagonalize, ufs_lagrange - diagonalize a matrix using the Lagrange method

Calling sequence:
`ufs_diagonalize(`M`)`
`ufs_lagrange(`M`)`

Parameters:

M – a symmetric matrix

Synopsis:

- `ufs_diagonalize` diagonalizes the given matrix using completion of squares (Lagrange's method).
- The argument must be a symmetric matrix, otherwise an error occurs.

Example:

```
> A:=matrix([[2,1,0,0],[1,2,3,0],[0,3,2,1],[0,0,1,1]]);

                                [2    1    0    0]
                                [                ]
                                [1    2    3    0]
                           A := [                ]
                                [0    3    2    1]
                                [                ]
                                [0    0    1    1]

> B:=ufs_diagonalize(A);

                              [2      0      0     0 ]
                              [                      ]
                              [0     3/2     0     0 ]
                         B := [                      ]
                              [0      0     -4     0 ]
                              [                      ]
                              [0      0      0    5/4]

> L:=ufs_lagrange(A);

                             [1    -1/2     1     1/4 ]
                             [                        ]
                             [0     1      -2    -1/2]
                        L := [                        ]
                             [0     0       1     1/4 ]
                             [                        ]
                             [0     0       0      1  ]

> C:=evalm(transpose(L) &* A &* L);

                              [2      0      0     0 ]
                              [                      ]
                              [0     3/2     0     0 ]
                         C := [                      ]
                              [0      0     -4     0 ]
                              [                      ]
                              [0      0      0    5/4]
```

1.3 ufs_radicalbase, ufs_corank - compute the radical of a quadratic form

Calling sequence:
`ufs_radicalbase(`M`)`
`ufs_corank(`M`)`
Parameters:
M – a symmetric matrix
Synopsis:

- `ufs_radicalbase` calculates a basis of the radical of the quadratic form associated to the matrix M.
- `ufs_corank` calculates just the length of such a base.
- The argument must be a symmetric matrix, otherwise an error occurs.

Example:

```
> A:=matrix([[2,-2,0,0,1],[-2,2,0,0,-1],[0,0,2,2,0],[0,0,2,2,0],[1,-1,0,0,2]]);

                          [ 2    -2    0    0     1]
                          [                        ]
                          [-2     2    0    0    -1]
                          [                        ]
                     A := [ 0     0    2    2     0]
                          [                        ]
                          [ 0     0    2    2     0]
                          [                        ]
                          [ 1    -1    0    0     2]

> ufs_radicalbase(A);

                 [[1, 1, 0, 0, 0], [0, 0, -1, 1, 0]]

> ufs_corank(A);

                                  2
```

1.4 ufs_ispositive, ufs_isnonnegative, ufs_type - check a quadratic form for definiteness

Calling sequence:
`ufs_ispositive(`M`)`
`ufs_isnonnegative(`M`)`
`ufs_type(`M`)`
Parameters:
M – a symmetric matrix defining a quadratic form q
Synopsis:

- `ufs_ispositive` checks whether q is positive definite or not.
- `ufs_isnonnegative` checks whether q is positive semidefinite or not.

- `ufs_type` calculates the definiteness type of q. This function returns either ‘indefinite‘, ‘non negative definite‘ or ‘positive definite‘.
- The argument must be a symmetric matrix, otherwise an error occurs.

Example:

```
> A:=matrix([[2,-2,0,0,1],[-2,2,0,0,-1],[0,0,2,2,0],[0,0,2,2,0],[1,-1,0,0,2]]);

                         [ 2    -2    0    0     1]
                         [                        ]
                         [-2     2    0    0    -1]
                         [                        ]
                    A := [ 0     0    2    2     0]
                         [                        ]
                         [ 0     0    2    2     0]
                         [                        ]
                         [ 1    -1    0    0     2]

> ufs_ispositive(A);

                                 false

> ufs_isnonnegative(A);

                                  true

> ufs_type(A);

                         non negative definite
```

1.5 ufs_rootsystem, ufs_rootlength - calculate a rootsystem of a quadratic form

Calling sequence:
`ufs_rootsystem`(M)
`ufs_rootlength`(M)
Parameters:
M – a symmetric matrix defining a unit form
Synopsis:

- `ufs_rootsystem` calculates a root system of the quadratic form associated to the matrix M. A rootsystem is a set of vectors $\{v_1, \dots, v_s\}$ such that any vector w with $q(w) = 1$ may be written as $w = v_i + r$ or $w = -v_i + r$ for some i and some radical vector r.
- `ufs_rootlength` gives just the length of such a root system The argument must be a symmetric matrix defining a unit form, otherwise an error occurs.

Example:

```
> A:=matrix([[2,1,1,0,0],[1,2,1,1,0],[1,1,2,1,1],[0,1,1,2,1],[0,0,1,1,2]]);

                      [2    1    1    0    0]
                      [                     ]
                      [1    2    1    1    0]
                      [                     ]
                 A := [1    1    2    1    1]
                      [                     ]
                      [0    1    1    2    1]
                      [                     ]
                      [0    0    1    1    2]

> ufs_rootsystem(A);

  [[1, 0, 0, 0, 0], [0, 1, 0, 0, 0], [0, 0, 1, 0, 0], [0, 0, 0, 1, 0],

        [0, 0, 0, 0, 1], [1, -1, 0, 0, 0], [1, 0, -1, 0, 0],

        [0, 1, -1, 0, 0], [0, 1, 0, -1, 0], [0, 0, 1, -1, 0],

        [0, 0, 1, 0, -1], [0, 0, 0, 1, -1], [1, -1, 0, 1, 0],

        [1, 0, -1, 1, 0], [1, 0, -1, 0, 1], [0, 1, -1, 0, 1],

        [0, 1, 0, -1, 1], [1, -1, -1, 1, 0], [1, -1, 0, 1, -1],

        [0, 1, -1, -1, 1]]

> ufs_rootlength(A);

                                   20
```

1.6 ufs_dynkintype - calculates the Dynkin type of a quadratic form

Calling sequence:
`ufs_dynkintype(`M`)`
Parameters:
M – a symmetric matrix defining a non negative unit form q
Synopsis:

- `ufs_dynkintype` calculates the Dynkin type of q, as described below.
- The argument must be a symmetric matrix defining a unit form, otherwise an error occurs.

It is a classic result, that the equivalence classes of positive unit forms are characterized by the Dynkin diagrams
A_n $(n \geq 1)$, D_n $(n \geq 4)$ and E_n $(n = 6, 7, 8)$,
namely a Dynkin Δ diagram defines a unit form q_Δ in the following way: Suppose Δ has $1, \ldots, n$

as vertices then q_Δ is a form in the variables $x_1, \ldots, x_n$ and $q_\Delta(x_i+x_j) = q_\Delta(x_i)+q_\Delta(x_j)-\epsilon x_i x_j$ where ϵ is the number of edges between i and $j$in Δ. Now each positive unit form q is equivalent to a unit form q_Δ where Δ is one of those Dynkin diagrams, called the Dynkin type of q. It has been shown in [1], that the equivalence classes of non-negative unit forms are characterized by two data: the corank and a Dynkin diagram as above, namely each unit form q in the variables $x_1, \ldots, x_n$ which has corank r is equivalent to some q_Δ, where Δ is one of those Dynkin diagrams with $n-r$ vertices. Again, Δ is called the Dynkin type of q.

Example:

```
> A:=matrix([[2,1,1,0,0],[1,2,1,1,0],[1,1,2,1,1],[0,1,1,2,1],[0,0,1,1,2]]);
> B:=linalg[diag](A,A):
```

```
                         [2    1    1    0    0]
                         [                     ]
                         [1    2    1    1    0]
                         [                     ]
                    A := [1    1    2    1    1]
                         [                     ]
                         [0    1    1    2    1]
                         [                     ]
                         [0    0    1    1    2]
```

```
> ufs_dynkintype(A);
> ufs_dynkintype(B);
```

```
                              D[5]


                                 2
                             D[5]
```

Reference: [1] M. Barot and J. A. de la Peña: The Dynkin type of a non-negative unit form. Expo. Math. 17 (1999), 339-348.

2 Functions for counting representations of quivers
Jiuzhao Hua

The functions of this package, which can be used independently form the main package of Crep are made available by loading the file `nor.src`.

2.1 numberofreps - compute numbers of representations of quivers over a finite field

Calling Sequence:

`numberofreps1`(a_{11}, N)
`numberofreps2`$(a_{11}, a_{22}, a_{12}, N)$
`numberofreps3`$(a_{11}, a_{22}, a_{33}, a_{12}, a_{23}, a_{31}, N)$

Parameters:

a_{ij} - number of edges between vertices i and j in a graph Γ
N - upper bound for the entries of the dimension vector

Synopsis:

- This program computes the number of isomorphism classes of representations, indecomposable representations and absolutely indecomposable representations of a quiver Γ with up to three vertices over a finite field F_q.
- Γ is a graph with up to three vertices.
- These functions compute polynomials $M_\Gamma(\alpha, q)$, $I_\Gamma(\alpha, q)$ and $A_\Gamma(\alpha, q)$ with α_i less than or equal to N simultaneously where
  - $M_\Gamma(\alpha, q)$ is the number of isoclasses of representations of Γ (with an arbitrary orientation) over F_q with dimension vector α
  - $I_\Gamma(\alpha, q)$ is the number of isoclasses of indecomposable representations of Γ (with an arbitrary orientation) over F_q with dimension vector α
  - $A_\Gamma(\alpha, q)$ is the number of isoclasses of absolutely indecomposable representations of Γ (with an arbitrary orientation) over F_q with dimension vector α.
- For the function `numberofreps1` to work properly, the integer N has to be less than or equal 16, for `numberofreps2` N has to be less than or equal 6 and for `numberofreps3` N has to be less than or equal 4.

Examples:

```
> numberofreps1(2,3);
```

$$A_\Gamma(1, q) = q^2$$
$$A_\Gamma(2, q) = q^5 + q^3$$
$$A_\Gamma(3, q) = q^{10} + q^8 + q^7 + q^6 + q^5 + q^4$$
$$I_\Gamma(1, q) = q^2$$
$$I_\Gamma(2, q) = q^5 + 1/2q^4 + q^3 - 1/2q^2$$
$$I_\Gamma(3, q) = q^{10} + q^8 + q^7 + 4/3q^6 + q^5 + q^4 - 1/3q^2$$
$$M_\Gamma(1, q) = q^2$$
$$M_\Gamma(2, q) = q^5 + q^4 + q^3$$
$$M_\Gamma(3, q) = q^{10} + q^8 + 2q^7 + 2q^6 + 2q^5 + q^4$$

3 Hall Polynomials for E_8

R. Nörenberg

The package `hall_e8` contains functions about Hall polynomials for an algebra of type E_8. The package can be used independently from the main package of Crep and is made available by reading the file `hall_e8.src`.

3.1 hall_e8

Description:

- The CREP package `hall_e8` provides access to Hall polynomials for a certain algebra of type E_8 corresponding to short exact sequences with first and middle term indecomposable.
- To use a function of the package it should once have been defined using the command `with(hall_e8,function)` for a single function or `with(hall_e8)` for all functions of the package simultaneously. Alternatively, a such a function can be accessed by the explicit call `hall_e8[function]`.
- The functions available are:

Algebra	Indecomposables	Polynomials
Projective	ind_num	proj_num

- For more information on a particular function see `hall_e8[function]`.

3.2 hall_e8[Algebra]

Calling Sequence:
`Algebra()`
Parameters:
none
Description:

- The function call `Algebra()` returns the underlying algebra of the CREP package `hall_e8`. This is a quiver algebra of type E_8 (in 'subspace orientation')
- Output is in the format of a posetalgebra of CREP.

Examples:

```
> with(hall_e8):
> Algebra();
              [8, [[1, 2, 3, 5], [3, 4], [5, 6], [6, 7], [7, 8]]]
```

SEE ALSO: `posetalgebra`

3.3 hall_e8[Indecomposables]

Calling Sequence:
`Indecomposables()`
`Indecomposables(`i`)`
Parameters:
i – an integer in the range 1..120

Description:

- If the function is called without parameter, it returns a list - ordered lexicographically- of the dimension vectors of the indecomposable modules of a quiver algebra of type E_8, of which there are 120.
- If the parameter i is specified, the i-th entry in this list is returned.

Examples:

```
> with(hall_e8):
> Indecomposables();
[[0, 0, 0, 0, 0, 0, 0, 1], [0, 0, 0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 0, 1, 1],

    [0, 0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 1, 1, 0],

    [0, 0, 0, 0, 0, 1, 1, 1], [0, 0, 0, 0, 1, 0, 0, 0],

    [0, 0, 0, 0, 1, 1, 0, 0], [0, 0, 0, 0, 1, 1, 1, 0],

    [0, 0, 0, 0, 1, 1, 1, 1], [0, 0, 0, 1, 0, 0, 0, 0],

    [0, 0, 1, 0, 0, 0, 0, 0], [0, 0, 1, 1, 0, 0, 0, 0],

    [0, 1, 0, 0, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0, 0, 0],

    [1, 0, 0, 0, 1, 0, 0, 0], [1, 0, 0, 0, 1, 1, 0, 0],

    [1, 0, 0, 0, 1, 1, 1, 0], [1, 0, 0, 0, 1, 1, 1, 1],

    [1, 0, 1, 0, 0, 0, 0, 0], [1, 0, 1, 0, 1, 0, 0, 0],

    [1, 0, 1, 0, 1, 1, 0, 0], [1, 0, 1, 0, 1, 1, 1, 0],

    [1, 0, 1, 0, 1, 1, 1, 1], [1, 0, 1, 1, 0, 0, 0, 0],

    [1, 0, 1, 1, 1, 0, 0, 0], [1, 0, 1, 1, 1, 1, 0, 0],

    [1, 0, 1, 1, 1, 1, 1, 0], [1, 0, 1, 1, 1, 1, 1, 1],

    [1, 1, 0, 0, 0, 0, 0, 0], [1, 1, 0, 0, 1, 0, 0, 0],

    [1, 1, 0, 0, 1, 1, 0, 0], [1, 1, 0, 0, 1, 1, 1, 0],

    [1, 1, 0, 0, 1, 1, 1, 1], [1, 1, 1, 0, 0, 0, 0, 0],

    [1, 1, 1, 0, 1, 0, 0, 0], [1, 1, 1, 0, 1, 1, 0, 0],

    [1, 1, 1, 0, 1, 1, 1, 0], [1, 1, 1, 0, 1, 1, 1, 1],
```

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

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

[3, 2, 2, 1, 2, 2, 2, 1], [3, 2, 2, 1, 3, 2, 1, 0],

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

[3, 2, 2, 1, 3, 3, 2, 1], [4, 2, 2, 1, 3, 2, 1, 0],

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

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

[4, 2, 3, 1, 3, 2, 1, 0], [4, 2, 3, 1, 3, 2, 1, 1],

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

[4, 2, 3, 1, 4, 3, 2, 1], [4, 2, 3, 2, 3, 2, 1, 0],

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

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

[5, 2, 3, 1, 4, 3, 2, 1], [5, 2, 3, 2, 4, 3, 2, 1],

[5, 2, 4, 2, 4, 3, 2, 1], [5, 3, 3, 1, 4, 3, 2, 1],

[5, 3, 3, 2, 4, 3, 2, 1], [5, 3, 4, 2, 4, 3, 2, 1],

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

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

[6, 3, 4, 2, 5, 4, 3, 2]]

> Indecomposables(115);
                              [5, 3, 4, 2, 4, 3, 2, 1]
```

SEE ALSO: `hall_e8[Algebra]`, `hall_e8[ind_num]`

3.4 hall_e8[Polynomials]

Calling Sequence:

Polynomials();
Polynomials(*i*);
Polynomials(*a*);
Polynomials(*i*,*m*);
Polynomials(*a*,*m*);
Polynomials(*i*,*m*,*c*);
Polynomials(*a*,*m*,*c*);

Parameters:

i – an integer in range 1..8

a,m – dimension vectors of indecomposable modules

c – a set of dimension vectors of indecomposable modules

Description:

- If the function is called with tree arguments a,m,c, of which the first and second are dimension vectors of indecomposable modules A and M respectively and the third is a set of dimension vectors of indecomposable modules C_j, then the corresponding Hall Polynomial is returned, i.e. the number of submodules of M isomorphic to A with factor M/A isomorphic to the direct sum C of the modules C_j. (This number is a polynomial in the cardinality of the base field.)
- Of course these Polynomials are zero unless the dimension vectors of A and C add up to the dimension vector of M. Also, A and M being indecomposable, the polynomial is known to be zero, if C should have a multiple direct summand. Thus, working with a set of dimension vectors in the third argument, which does not allow for multiple summands, is indeed sufficient.
- A list of the dimension vectors of all indecomposable modules can be obtained using the function `Indecomposables`.
- If the function is called with two parameters a,m, which are dimension vectors of indecomposable modules A and M, it returns a list which is indexed by all possible sets C of dimension vectors for which the Hall polynomial as returned by a call of `Polynomials(`a, m, c`)` is not zero. The entries of this list are the corresponding polynomials.
- If the function is called with exactly one argument a, which is the dimension vector of an indecomposable module A, it returns a list of all nonzero Hall polynomials obtained for A with arbitrary indecomposable modules M (and arbitrary C).
- Finally, if the function is called without any argument, it returns a list of all polynomials obtained as described above for indecomposable modules A and M.
- In the first argument, instead of the dimension vector of an indecomposable module, an integer i in range 1..8 can be used. In this case the dimension vector of the indecomposable projective module corresponding to the vertex i of the underlying algebra, as given by a call of `Projective(`i`)`; is substituted as first argument. Thus a call of `Polynomials(`$i, \ldots$`)` is equivalent to a call of `Polynomials(Projective(`i`),`$\ldots$`)` .

Examples:

```
> with(hall_e8):
> Polynomials();
                      2              2              2              2
[1, x - 2, x - 1, x  - 4 x + 4, x  - 3 x + 2, x  - 3 x + 3, x  - 2 x + 1,

     2       3      2              3      2              3      2
    x  - x, x  - 6 x  + 12 x - 8, x  - 5 x  + 8 x - 4, x  - 5 x  + 9 x - 6,

     3      2              3      2              3      2
    x  - 5 x  + 10 x - 7, x  - 4 x  + 5 x - 2, x  - 4 x  + 6 x - 4,
```

$$x^3 - 4x^2 + 6x - 3,\ x^3 - 4x^2 + 7x - 5,\ x^3 - 4x^2 + 7x - 4,$$

$$x^3 - 4x^2 + 8x - 6,\ x^3 - 3x^2 + 2x,\ x^3 - 3x^2 + 3x - 1,$$

$$x^3 - 3x^2 + 4x - 2,\ x^3 - 3x^2 + 5x - 4,\ x^3 - 2x^2 + 2x - 1,$$

$$x^4 - 6x^3 + 16x^2 - 22x + 12,\ x^4 - 5x^3 + 10x^2 - 11x + 6,$$

$$x^4 - 5x^3 + 10x^2 - 10x + 4,\ x^4 - 5x^3 + 10x^2 - 10x + 5,$$

$$x^4 - 5x^3 + 11x^2 - 14x + 8,\ x^4 - 5x^3 + 11x^2 - 13x + 7,$$

$$x^4 - 4x^3 + 6x^2 - 5x + 2,\ x^4 - 4x^3 + 6x^2 - 4x + 1,$$

$$x^4 - 4x^3 + 7x^2 - 9x + 6,\ x^4 - 4x^3 + 7x^2 - 8x + 5,$$

$$x^4 - 4x^3 + 7x^2 - 6x + 2,\ x^4 - 4x^3 + 8x^2 - 10x + 6,$$

$$x^5 - 6x^4 + 15x^3 - 23x^2 + 25x - 13,$$

$$x^5 - 5x^4 + 10x^3 - 13x^2 + 14x - 8,\ x^5 - 5x^4 + 10x^3 - 13x^2 + 15x - 9,$$

$$x^5 - 5x^4 + 10x^3 - 12x^2 + 11x - 6,\ x^5 - 5x^4 + 10x^3 - 12x^2 + 12x - 7$$

]

```
> Polynomials(2);
```

$$[1,\ x - 2,\ x - 1,\ x^2 - 4x + 4,\ x^2 - 3x + 3]$$

```
> Polynomials([1,1,0,0,0,0,0,0]);
```

$$[1,\ x - 2,\ x - 1,\ x^2 - 4x + 4,\ x^2 - 3x + 3]$$

```
> Polynomials([6,3,4,2,5,3,2,1],[6,3,4,2,5,4,3,2]);
table([

                                                                  2
    {[0, 0, 0, 0, 0, 0, 1, 1], [0, 0, 0, 0, 0, 1, 0, 0]} = x  - 3 x + 2

                                 3      2
    {[0, 0, 0, 0, 0, 1, 1, 1]} = x  - 5 x  + 10 x - 7

                                                                  2
    {[0, 0, 0, 0, 0, 0, 0, 1], [0, 0, 0, 0, 0, 1, 1, 0]} = x  - 3 x + 2

    {[0, 0, 0, 0, 0, 0, 0, 1], [0, 0, 0, 0, 0, 0, 1, 0],

    [0, 0, 0, 0, 0, 1, 0, 0]} = x - 1

    ])


> Polynomials([6,3,4,2,5,3,2,1],[6,3,4,2,5,4,3,2],
              {[0,0,0,0,0,0,1,1],[0,0,0,0,0,1,0,0]});
                                 2
                                x  - 3 x + 2


> Polynomials([6,3,4,2,5,3,2,1],[6,3,4,2,5,4,3,2],{[0,0,0,0,0,1,1,1]});

                             3      2
                            x  - 5 x  + 10 x - 7
```

SEE ALSO: `hall_e8[Indecomposables]`, `hall_e8[Projective]`

3.5 hall_e8[Projective]

Calling Sequence:
`Projective(`i`);`
Parameters:
i – an integer in the range 1..8
Description:

- The call `Projective(`i`)` returns the dimension vector of the indecomposable projective module associated with the vertex i of the algebra of type E_8 underlying the `hall_e8` package.
- output is in format of a list.

Examples:

```
> with(hall_e8):
```

```
> Projective(5);
                              [1, 0, 0, 0, 1, 0, 0, 0]
```

SEE ALSO: hall_e8[Algebra], hall_e8[proj_num]

3.6 hall_e8[ind_num]

Calling Sequence:
ind_num();

Description:

- The call ind_num(i) returns number of indecomposable modules of the algebra of type E_8 underlying the hall_e8 package. (Which, as for all algebras of this type, is 120.)

Examples:

```
> with(hall_e8):
> ind_num();
                                      120
```

SEE ALSO: hall_e8[Algebra], hall_e8[Indecomposables], hall_e8[proj_num]

3.7 hall_e8[proj_num]

Calling Sequence:
proj_num();

Description:

- The call proj_num(i) returns number of indecomposable projective modules of the algebra of type E_8 underlying the hall_e8 package. (Which, as for all algebras of this type, is eight.)

Examples:

```
> with(hall_e8):
> proj_num();
                                       8
```

SEE ALSO: hall_e8[Algebra], hall_e8[Projective], hall_e8[ind_num]