CREP Manual

Version 1.0 using Maple as surface

Peter Dräxler and Rainer Nörenberg

Contents

Introduction **7**

1 Using CREP with Maple **8**

1.1 Starting the system . . . 8
1.2 Caveats . . . 8
1.3 The CREP home directory . . . 8
1.4 Contents of the CREP package . . . 8
1.4.1 Posets and bound poset algebras . . . 9
1.4.2 Quadratic forms . . . 9
1.4.3 Representation type . . . 9
1.4.4 Auslander Reiten quivers . . . 10
1.4.5 Lists . . . 10
1.4.6 Output control functions . . . 10

2 The CREP package **11**

2.1 CREP - Combinatorial REPresentation theory . . . 11
2.2 crep/datatypes . . . 11
2.3 The CREP data format for posets . . . 11
2.4 The CREP data format for bound posetalgebras . . . 12
2.5 The CREP data format for unit forms . . . 13
2.6 crep/predefined_variables . . . 14
2.7 crep/internal_variables . . . 15
2.8 cartanm - compute the Cartan-matrix of a bound poset algebra . . . 15
2.9 complsep - test for a completely separating bound poset algebra . . . 16
2.10 coxeterm - compute the Coxeter-matrix of a bound poset algebra . . . 17
2.11 cre_num - the numbers of frames of completely separating tame concealed algebras of type $\tilde{E}$. . . . 17
2.12 crep_silent - turn CREP into silent mode . . . 18
2.13 crep_verbose - turn CREP into verbose mode . . . 18
2.14 critical_alge - representatives of frames of completely separating tame concealed algebras of type $\tilde{E}$. . . . 19
2.15 drwpst - draw a poset or a bound poset algebra . . . 19
2.16 dual - generate the opposite of a poset or bound poset algebra . . . 20
2.17 eulerform - the Euler-form of a completely separating bound poset algebra . . . 21
2.18 hcd_num - the numbers of frames of completely separating hypercritical algebras of type $\tilde{\tilde{D}}$. . . . 22
2.19 hce_num - the numbers of frames of completely separating hypercritical algebras of type $\tilde{\tilde{E}}$. . . . 23
2.20 hypercritical_algd - representatives of frames of completely separating hypercritical algebras of type $\tilde{\tilde{D}}$ . . . 23
2.21 hypercritical_alge - representatives of frames of completely separating hypercritical algebras of type $\tilde{\tilde{E}}$ . . . 23
2.22 iscriticald - test for a tame concealed algebra of type $\tilde{D}$. . . . 24
2.23 iscriticale - test for a tame concealed algebra of type $\tilde{E}$. . . . 24
2.24 ishypercriticald - test for a hypercritical algebra of type $\tilde{\tilde{D}}$ . . . 25

2.25 ishypercriticale - test for a hypercritical algebra of type $\tilde{\tilde{E}}$ 26
2.26 isoalg - test whether two bound poset algebras are isomorphic 26
2.27 isodual - test whether two algebras are dual to one another 27
2.28 mchpg - test an algebra for being of polynomial growth 27
2.29 mchrepfin - test an algebra for being representation finite 29
2.30 mchwild - for a given bound poset algebra look for a poset of minimal chain sequences which is wild 31
2.31 preproj - compute the beginning of the postprojective component of a bound poset algebra 32
2.32 preprojposet - compute the beginning of the postprojective component for a poset 34
2.33 pspg - test a poset for polynomial growth 35
2.34 psrepfin - test a poset for finite representation type 36
2.35 pswild - test a poset for wild representation type 37
2.36 rdrwpst - repeat the last drawing made by drwpst 38
2.37 readalg - interactive input of a bound poset algebra 39
2.38 rsrdtest - test for regular representation directed algebras 40
2.39 rstartsets - the poset of start sets induced from minimal chain sequences 40
2.40 sameframe - determine whether two bound poset algebras have the same frame . 41
2.41 subposet - look for a given subposet within a poset 42
2.42 tctubes - compute the mouths of the exceptional tubes of a tame concealed bound poset algebra 43
2.43 titsform - the Tits-form of a completely separating bound poset algebra 44
2.44 titsform_p - compute the quadratic form of a poset 44
2.45 tubetest - test for tubular extensions of $\tilde{E}_n$ 45
2.46 weakly_nonneg - test a unit form for weak non-negativity 47
2.47 weakly_positive - test a unit form for weak positivity 48

3 The esrd package 50
3.1 Using the esrd package 50
3.2 The CREP-package esrd for exceptional sincere representation directed algebras 50
3.3 esrdalg - representatives of frames of exceptional sincere representation directed algebras 50
3.4 esrd_num - the numbers of frames of exceptional sincere representation directed algebras 51
3.5 esrd_orbittypes - the orbit types of the exceptional sincere representation directed algebras 51
3.6 esrd_rootsa - positive roots of the Tits-form 52
3.7 esrd_rootss - sincere positive roots of the Tits-form 53
3.8 esrdtest - test for exceptional sincere representation directed algebras 54

4 The heralg package 56
4.1 Introduction to the heralg package 56
4.2 the type heralg/quiverdata 56
4.3 heralg[cartan_matrix] - the Cartan matrix of a quiver algebra 57
4.4 heralg[cartan_inverse] - the inverse of the Cartan matrix of a quiver algebra . . . 58
4.5 heralg[coxeter_matrix] - the Coxeter matrix of a quiver algebra 59
4.6 heralg[coxeter_inverse] - the inverse of the Coxeter matrix of a quiver algebra . . 59

4.7 heralg[tau_translate] - the Coxeter transformation of the dimension vector of a module . . . 60
4.8 heralg[homol_blf] - the homological bilinear form of two dimension vectors of modules . . . 61
4.9 heralg[coxeter_polynomial] - the characteristic polynomial of the Coxeter matrix of a quiver algebra . . . 62
4.10 heralg[coxeter_roots] - the eigenvalues of the Coxeter matrix of a quiver algebra 63
4.11 heralg[growth_number] - the spectral radius of the Coxeter matrix of a quiver algebra . . . 63
4.12 heralg[one_pt_cartmatrix] - the Cartan matrix of a one-point extension of a quiver algebra . . . 64
4.13 heralg[one_pt_coxmatrix] - the Coxeter matrix of a one-point extension of a quiver algebra . . . 65
4.14 heralg[one_pt_coxpoly] - the Coxeter polynomial of a one-point extension of a quiver algebra . . . 66

Historical remarks **68**

References **70**

Introduction

CREP is a package of programs to work with particular classes of finite-dimensional algebras (e.g. completely separating algebras, hereditary algebras) resp. related structures (e.g. unit forms, partially ordered sets) and some combinatorial aspects of their finite-dimensional representations (e.g. dimension vectors, Tits and Euler forms, thin modules, postprojective, preinjective and tubular Auslander-Reiten components). Moreover there are data bases containing some important classifications (e.g. tame concealed algebras, sincere representation-directed algebras, minimal wild concealed algebras).

The programs are partly written in Pascal and partly in Maple programming language. Nevertheless, in contrast to the previous versions (consisting of collections of stand-alone Pascal programs) in version 1.0 the user will only see a package of Maple routines. Thus he only has to learn how to use Maple and to understand the data structures of the input and output.

CREP is not yet a general purpose system but very specific. Hence, for many problems it will not help but it will be very effective for various important questions which appear in the present research. The vision is to accumulate more and more routines in order to obtain an efficient and general system for representation theory of finite-dimensional algebras. People are invited to contribute.

CREP can be retrieved via `ftp` from the server `ftp.uni-bielefeld.de` where it will be found in the directory `pub/math/f-d-alg`.

Running CREP will require the Maple system to be present, preferably MapleV release 3. Usually for installation a Pascal compiler will be required. For DEC and Sun workstations and on the PC under DOS also compiled versions of CREP are available.

In case of questions, remarks, or suggestions concerning CREP please feel free to send a message by e-mail to

`fdowner@mathematik.uni.bielefeld.de`

or send a letter to

Peter Dräxler
SFB 343
Universität Bielefeld
POBox 100 131
D-33501 Bielefeld
Germany

1 Using CREP with Maple

1.1 Starting the system

In order to use CREP with Maple as surface, you first have to start a Maple session. This has to be done from the CREP home directory, otherwise Maple will not be able to execute CREP commands properly.

Thus to start CREP with Maple, first change the current directory to the CREP home directory and then start Maple from there.

Next you have to tell Maple to read in the file `crep.src`, which is located in the CREP home directory. You can do this by typing the command `read ‘crep.src‘` in your current Maple session.

By reading the file `crep.src` the functions and data of the CREP package are made available. (See section 1.4 for a brief overview and section 2 for a more detailed description.) You may use the Maple interactive help feature to obtain information about the CREP system by typing `?crep`. Also you will get help on any specific function via `?<function>`.

Instead of reading in the file `crep.src` manually each time you wish to use CREP, you may configure Maple so that it will automatically read in this file, and thus start CREP, each time it is invoked from the CREP home directory. This usually is done by an initialisation file for Maple, located in the CREP home directory, containing the line `read ‘crep.src‘`. (See the Maple reference manual for more detailed information.)

1.2 Caveats

While most dialogue of the CREP system is done via Maple, the CREP-functions `readalg`, `preproj` and `preprojposet` will start some dialogue of their own. This will most probably fail if the current Maple session is running in the background. So one should make sure not to start Maple as a background process, when wishing to use one of these functions.

1.3 The CREP home directory

The CREP home directory is most commonly called `crep`. It contains among others a file called `crep.src` containing the definitions of the CREP system for Maple. It further contains a subdirectory called `bin` containing executables of the CREP system and subdirectories like `bhv` and `rsrddat` containing data for CREP. While the name of the CREP home directory itself may change, the names of these subdirectories may not.

Many CREP functions use calls to external programs written in Pascal (often performing computations that would be too time consuming when done by a system capable of symbolic computation like Maple). All communication between the Maple system and these external programs also is performed inside the CREP home directory. In particular data are passed to the external programs using a file called `crep.com` and passed back to Maple using files called `crep.mpl`, `crep1.mpl`, or `crep2.mpl`.

1.4 Contents of the CREP package

This section gives a short overview of the functions of the CREP package. More detailed descriptions of the single functions are given in section 2.

1.4.1 Posets and bound poset algebras

complsep - test for a completely separating bound poset algebra

drwpst - draw a poset or a bound poset algebra

dual - generate the opposite of a poset or bound poset algebra

isoalg - test whether two bound poset algebras are isomorphic

isodual - test whether two algebras are dual to one another

rdrwpst - repeat the last drawing made by drwpst

readalg - interactive input of a bound poset algebra

sameframe - determine whether two bound poset algebras have the same frame

subposet - look for a given subposet within a poset

1.4.2 Quadratic forms

cartanm - compute the Cartan-matrix of a bound poset algebra

coxeterm - compute the Coxeter-matrix of a bound poset algebra

eulerform - the Euler-form of a completely separating bound poset algebra

titsform - the Tits-form of a completely separating bound poset algebra

titsform_p - compute the quadratic form of a poset

weakly_nonneg - test a unit form for weak non-negativity

weakly_positive - test a unit form for weak positivity

1.4.3 Representation type

mchpg - test an algebra for being of polynomial growth

mchrepfin - test an algebra for being representation finite

mchwild - for a given bound poset algebra look for a poset of minimal chain sequences which is wild

pspg - test a poset for polynomial growth

psrepfin - test a poset for finite representation type

pswild - test a poset for wild representation type

rstartsets - the poset of start sets induced from minimal chain sequences

1.4.4 Auslander Reiten quivers

preproj - compute the beginning of the postprojective component of a bound poset algebra

preprojposet - compute the beginning of the postprojective component for a poset

tctubes - compute the mouths of the exceptional tubes of a tame concealed bound poset algebra

1.4.5 Lists

cre_num - the numbers of frames of completely separating tame concealed algebras of type $\tilde{E}$

critical_alge - representatives of frames of completely separating tame concealed algebras of type $\tilde{E}$

hcd_num - the numbers of frames of completely separating hypercritical algebras of type $\tilde{\tilde{D}}$

hce_num - the numbers of frames of completely separating hypercritical algebras of type $\tilde{\tilde{E}}$

hypercritical_algd - representatives of frames of completely separating hypercritical algebras of type $\tilde{\tilde{D}}$

hypercritical_alge - representatives of frames of completely separating hypercritical algebras of type $\tilde{\tilde{E}}$

iscriticald - test for a tame concealed algebra of type $\tilde{D}$

iscriticale - test for a tame concealed algebra of type $\tilde{E}$

ishypercriticald - test for a hypercritical algebra of type $\tilde{\tilde{D}}$

ishypercriticale - test for a hypercritical algebra of type $\tilde{\tilde{E}}$

rsrdtest - test for regular representation directed algebras

tubetest - test for tubular extensions of $\tilde{E}_n$

1.4.6 Output control functions

crep_silent - turn CREP into silent mode

crep_verbose - turn CREP into verbose mode

2 The CREP package

2.1 CREP - Combinatorial REPresentation theory

Calling Sequence:

`<function>`(*args*)

Synopsis:

- CREP provides a variety of functions and data related to the representation theory of finite-dimensional algebras.
- The functions available are:

cartanm	complsep	coxeterm	cre_num
crep_silent	crep_verbose	critical_alge	drwpst
dual	eulerform	hcd_num	hce_num
hypercritical_algd	hypercritical_alge	iscriticald	iscriticale
ishypercriticald	ishypercriticale	isoalg	isodual
mchpg	mchrepfin	mchwild	preproj
preprojposet	pspg	psrepfin	pswild
rdrwpst	readalg	rsrdtest	rstartsets
sameframe	subposet	tctubes	titsform
titsform_p	tubetest	weakly_nonneg	weakly_positive

- For more information on a particular function see `<function>`.
- Some data are made available by means of global variables rather than via functions. See `crep/predefined_variables` for a list of these.

See Also: `crep/datatypes, crep/internal_variables, crep/predefined_variables`

2.2 crep/datatypes

Synopsis:

- In dealing with algebraic or combinatorial objects CREP uses specific data formats. This currently is the case for posets, bound poset algebras and unit forms.
(See `poset`, `posetalgebra` or `unitform` for more detail.)
- Although the functions of the CREP-package depend on the correct use of datatypes when passing information via their arguments, in most cases in the current version of CREP no type testing is done.

See Also: `poset, posetalgebra, unitform`

2.3 The CREP data format for posets

Synopsis:

- A partially ordered set S is denoted by a pair

$$[n, l]$$

where n is an integer, the number of elements of S; and l is a list or set

$$l = [l_1, \ldots, l_k]$$

or

$$l = \{l_1, \ldots, l_k\}$$

containing information about the Hasse-diagram of S in the following way:

Each entry of l_i of l is a list of positive integers

$$l_i = [l_{i,1}, l_{i,2} \ldots l_{i,m_i}]$$

where $\{l_{i,2}, \ldots l_{i,m_i}\}$ is the set of upper neighbours of $l_{i,1}$ in the Hasse-diagram of S, when identifying S with the set $\{1, \ldots, n\}$.

Examples:

```
[6,{[1,2,3],[2,4],[3,4,5],[5],[4,6],[6]}];
[6,{[1,2,3],[2,4],[3,4,5],[4,6]}];
[6,[[1,2,3],[2,4],[3,4,5],[4,6]]];

- Any of these three examples denotes the same poset.

- use the function  drwpst  to obtain a graphical
  display of the Hasse-diagram of this poset.
```

SEE ALSO: `posetalgebra, drwpst`

2.4 The CREP data format for bound posetalgebras

Synopsis:

- For a poset $(S, \leq)$ the incidence algebra $k[S]$ over a field k is the algebra generated by all the pairs (x, y) with $x \leq y$ in S, and the multiplication rule $(x, y) * (u, v) = (x, v)$ if $y = u$ and $(x, y) * (u, v) = 0$ otherwise.
- By a bound poset algebra we mean a factor $k[S]/I$ of an incidence algebra of a poset by an admissible ideal I of $k[S]$, i.e. an ideal generated by elements of the form (x, y) such that there is an element z of S such that $x < z < y$ in S. In particular I might be trivial.
- A bound poset algebra $A = k[S]/I$, is denoted by a triple

  $$[n, l, r]$$

  where n and l denote the structure of the underlying poset, i.e.: n is an integer, the number of elements of S; and l is a list or set

  $$l = [l_1, \ldots, l_k]$$

  or

  $$l = \{l_1, \ldots, l_k\}$$

containing information about the Hasse-diagram of S in the following way:
Each entry of l_i of l is a list

$$l_i = [l_{i,1}, l_{i,2} ... l_{i,m_i}]$$

where $\{l_{i,2} \ldots l_{i,m_i}\}$ is the set of upper neighbours of $l_{i,1}$ in the Hasse-diagram of S, when identifying S with the set $\{1, \ldots, n\}$.

r is a set or list of pairs

$$r = \{r_1, \ldots, r_t\}$$

or

$$r = [r_1, \ldots, r_t]$$

where each pair $r_i = [r_{i1}, r_{i2}]$ stands for a generator (r_{i1}, r_{i2}) of the ideal I.

- In case $I = 0$, $A = K[S]$ is denoted just by the pair $[n, l]$. Thus in this case the data format of the posetalgebra A is the same as that for the underlying poset S.

Examples:

```
[6,{[1,2,3],[2,4],[3,4,5],[5],[4,6],[6]},{[1,5],[1,6]}];
[6,{[1,2,3],[2,4],[3,4,5],[5],[4,6],[6]},[[1,5],[1,6]]];
[6,{[1,2,3],[2,4],[3,4,5],[4,6]},{[1,5],[1,6]}];
[6,[[1,2,3],[2,4],[3,4,5],[4,6]],[[1,5],[1,6]]];
```

```
- Any of these three examples denotes the same bound poset
  algebra.

- use the function  drwpst  to obtain a graphical display.
```

SEE ALSO: `poset, drwpst`

2.5 The CREP data format for unit forms

Synopsis:

- a unit form q is a map

$$\mathbb{Z}^n \longrightarrow \mathbb{Z}, x \mapsto \sum_{i=1}^{n} x_i^2 + \sum_{i<j}^{n} a_{i,j} x_i x_j$$

where $\mathbb{Z}$ denotes the ring of integers and $a_{i,j}$ are integers.

- A unit form q is represented by its symmetric integer-coefficient matrix Q, i.e. a matrix Q such that

$$q(x) = 1/2x^T Qx$$

Examples:

```
> with(linalg);

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

```
                 [ 2   1   0 ]
                 [           ]
            q := [ 1   2  -1 ]
                 [           ]
                 [ 0  -1   2 ]
```

```
> x:=vector(3);
```

```
                         x := array(1 .. 3, [])
```

```
> print(x);
```

```
                          [ x[1], x[2], x[3] ]
```

```
> 1/2*innerprod(x,Q,x);
```

```
                 2                   2                   2
             x[1]  + x[1] x[2] + x[2]  - x[2] x[3] + x[3]
```

SEE ALSO: `crep/datatypes`

2.6 crep/predefined_variables

Synopsis:

- Some data in CREP are provided by means of global variables rather than by functions. The names of such global variables start with 'cr_'.
- These are (together with their definition):
  - The list of critical posets:

    ```
    cr_K[1]:=[4];
    cr_K[2]:=[6,[[1,4],[2,5],[3,6]]];
    cr_K[3]:=[7,[[2,4],[3,6],[4,5],[6,7]]];
    cr_K[4]:=[8,[[1,4],[2,4,5],[3,6],[6,7],[7,8]]];
    cr_K[5]:=[8,[[2,4],[3,5],[5,6],[6,7],[7,8]]];
    ```

  - The list of hypercritical posets:

    ```
    cr_N[1]:=[5];
    cr_N[2]:=[5,[[4,5]]];
    cr_N[3]:=[7,[[1,3],[2,4],[5,6],[6,7]]];
    cr_N[4]:=[8,[[2,3],[3,4],[5,6],[6,7],[7,8]]];
    cr_N[5]:=[9,[[1,2],[3,2,4],[5,6],[6,7],[7,8],[8,9]]];
    cr_N[6]:=[9,[[2,3],[4,5],[5,6],[6,7],[7,8],[8,9]]];
    ```

  - The tame poset minimally of non polynomial growth:

    ```
    cr_NZ:=[6,[[1,3,4],[2,3,4]]];
    ```

- some functions of CREP (such as `psrefin`, `pspg`, `pswild`) use these names when referring to the corresponding poset.
- users should take care not to redefine these names while using CREP.

Examples:

```
> p:=[10,{[1, 3, 4], [4, 2], [3, 2], [9, 3, 6], [10, 6], [6, 4], [7, 1, 9]}];

 p := [10, {[4, 2], [3, 2], [9, 3, 6], [10, 6], [6, 4], [7, 1, 9], [1, 3, 4]}]

> psrepfin(p,'c');
                                    false

> print(c);

                                   cr_K[1]

> cr_K[1];

                                     [4]
```

SEE ALSO: `crep/internal_variables, psrepfin`

2.7 crep/internal_variables

Synopsis:

- CREP uses some global variables for internal communication, the names of which begin with the sequence '%cr' (in most cases at least).
- In order to avoid confusion users should refrain from using such names when working with CREP.

SEE ALSO: `crep/predefined_variables`

2.8 cartanm - compute the Cartan-matrix of a bound poset algebra

Calling Sequence:
`cartanm`(p)
Parameters:
p – a bound poset algebra
Synopsis:

- `cartanm`(p) computes the Cartan-matrix of the bound poset algebra p.

Examples:

```
> a:=critical_alge(7,14);
     [8, {[1, 2, 4], [2, 5, 6], [3, 2], [4, 6], [6], [7, 2], [8, 7], [5]},

         {[1, 5], [3, 6]}]
```

```
> cartanm(a);
                    [ 1  1  0  1  0  1  0  0 ]
                    [                        ]
                    [ 0  1  0  0  1  1  0  0 ]
                    [                        ]
                    [ 0  1  1  0  1  0  0  0 ]
                    [                        ]
                    [ 0  0  0  1  0  1  0  0 ]
                    [                        ]
                    [ 0  0  0  0  1  0  0  0 ]
                    [                        ]
                    [ 0  0  0  0  0  1  0  0 ]
                    [                        ]
                    [ 0  1  0  0  1  1  1  0 ]
                    [                        ]
                    [ 0  1  0  0  1  1  1  1 ]
```

SEE ALSO: `coxeterm`, `eulerform`

2.9 complsep - test for a completely separating bound poset algebra

Calling Sequence:
`complsep(`p`)`
Parameters:
p – a bound poset algebra
Synopsis:

- `complsep(`p`)` tests whether the bound poset algebra p is completely separating.

Examples:

```
> p:=[4,{[1,2,3],[4,2,3]}];

                  p := [4, {[1, 2, 3], [4, 2, 3]}]

> complsep(p);
                              false

> p:=[8,{[2, 1, 3, 7], [4, 1], [5, 2], [6, 2, 4], [7, 8]},{[6, 3], [5, 1]}];

 p := [8, {[7, 8], [2, 1, 3, 7], [4, 1], [5, 2], [6, 2, 4]}, {[6, 3], [5, 1]}]

> complsep(p);
                               true
```

SEE ALSO: `posetalgebra`

2.10 coxeterm - compute the Coxeter-matrix of a bound poset algebra

Calling Sequence:
`coxeterm(`p`)`

Parameters:
p – a bound poset algebra

Synopsis:

- `coxeterm(`p`)` computes the Coxeter-matrix of the bound poset algebra p

---

Examples:

```
> p:=hypercritical_algd(4,1);
              [6, {[3, 1], [2], [6], [1, 2, 6], [4, 1], [5, 4]}]

> coxeterm(p);
                              [ 1   1   1   1   0   1 ]
                              [                       ]
                              [ 1   0   0   0   0   1 ]
                              [                       ]
                              [-1  -1  -1   0   0  -1 ]
                              [                       ]
                              [ 0   0   0   0   1   0 ]
                              [                       ]
                              [-1  -1   0  -1  -1  -1 ]
                              [                       ]
                              [ 1   1   0   0   0   0 ]
```

SEE ALSO: `cartanm, eulerform`

2.11 cre_num - the numbers of frames of completely separating tame concealed algebras of type $\tilde{E}$.

Calling Sequence:
`cre_num(`n`)`

Parameters:
n – 6,7 or 8

Synopsis:

- `cre_num(`n`)` returns the number of frames of completely separating tame concealed algebras of type $\tilde{E}_n$.

Examples:

```
> cre_num(7);
                                      21
```

SEE ALSO: `critical_alge`

2.12 crep_silent - turn CREP into silent mode

Calling Sequence:
`crep_silent()`
Parameters:
none
Synopsis:

- `crep_silent()` disables the enforced printing of results of CREP-functions by setting the internal control variable '%cr_silent' to `true`.
- By default, CREP is in non-silent (or verbose) mode.
- `crep_silent` will always return the value `true`.

Examples:

```
> q:=iscriticale(critical_alge(6,1));
        [7, {[3, 2], [1], [4, 1], [2, 1], [5, 4], [6, 1], [7, 6]}]

                                    [6, 1]

> crep_silent();

                                     true

> q:=iscriticale(critical_alge(6,1));
> q;

                                    [6, 1]
```

SEE ALSO: `crep_verbose()`, `crep/internal_variables`

2.13 crep_verbose - turn CREP into verbose mode

Calling Sequence:
`crep_verbose();`
Parameters:
none
Synopsis:

- `crep_verbose()` enforces printing of results of CREP-functions by setting the internal control variable '%cr_silent' to `false`.
- By default, CREP is in non-silent (or verbose) mode.
- `crep_verbose` will always return the value `false`.

Examples:

```
> crep_silent();

                                     true
```

```
> q:=iscriticale(critical_alge(6,1));
> q;
```

```
                                  [6, 1]
```

```
> crep_verbose();
```

```
                                  false
```

```
> q:=iscriticale(critical_alge(6,1));
         [7, {[3, 2], [1], [4, 1], [2, 1], [5, 4], [6, 1], [7, 6]}]

                                  [6, 1]
```

SEE ALSO: `crep_silent`, `crep/internal_variables`

2.14 critical_alge - representatives of frames of completely separating tame concealed algebras of type $\tilde{E}$.

Calling Sequence:
`critical_alge(`n,m`)`

Parameters:
n – 6,7 or 8
m – a positive integer less than or equal to `cre_num`(n)

Synopsis:

- `critical_alge`(n,m) returns an algebra which belongs to the m-th frame of completely separating tame concealed algebras of type $\tilde{E}_n$.
- The number of frames available may by obtained by calling the function `cre_num`.

Examples:

```
> critical_alge(7,14);
    [8, {[5], [1, 2, 4], [2, 5, 6], [3, 2], [4, 6], [6], [7, 2], [8, 7]},

        {[3, 6], [1, 5]}]
```

SEE ALSO: `cre_num`, `hypercritical_algd`, `hypercritical_alge`, `posetalgebra`

2.15 drwpst - draw a poset or a bound poset algebra

Calling Sequence:
`drwpst(`p`);`
`drwpst(`p,l`);`
`drwpst(`p,s`);`
`drwpst(`p,l,s`);`

Parameters:
p – a bound poset algebra
l – a list of length p[1] (i.e. the number of points of p)
s – a set of positive integers between 1 and p[1]

Synopsis:

- `drwpst` uses Maple plot output to draw the Hasse-diagram of p (if p is a poset) or of the underlying poset of p (in case p is a bound poset algebra).
- By default the vertices of the Hasse-diagram are marked by '`o`'. If an optional argument s is given, the points corresponding to the elements of s are marked by '`x`'.
- By default the vertices of the Hasse-diagram are labelled by the integers $1, \ldots, p[1]$. The optional argument l provides a list of labels to use instead of these.

Examples:

```
> a:=critical_alge(7,14);
    [8, {[5], [7, 2], [8, 7], [3, 2], [6], [4, 6], [1, 2, 4], [2, 5, 6]},

        {[3, 6], [1, 5]}]

> drwpst(a);
-----------------------------------------------------------------------------

> p:=[9, {[5, 6], [7, 2], [3, 2], [4, 1], [6], [8, 1, 9], [9, 3, 7], [1, 3]}];

 p := [9, {[5, 6], [7, 2], [3, 2], [4, 1], [6], [8, 1, 9], [9, 3, 7], [1, 3]}]

> subposet(cr_K[2],p,'incl','img');
                                         true

> print(img,incl);

                [1, 0, 0, 1, 1, 1, 1, 0, 1], [4, 5, 9, 1, 6, 7]

> drwpst(p);

> s:=convert(incl,set);

> drwpst(p,s);

> drwpst(p,img);
```

SEE ALSO: `poset, posetalgebra, subposet, rdrwpst, crep/predefined_variables`

2.16 dual - generate the opposite of a poset or bound poset algebra

Calling Sequence:
`dual(`p`)`

Parameters:
p – a poset or bound poset algebra
Synopsis:

- `dual`(p) returns the opposite of p, i.e. the poset (resp. bound poset algebra) obtained from p by reversing all relations.

Examples:

```
> p:= [8, {[1, 2, 4],[2, 5, 6],[3, 2],[4, 6],[7, 2],[8, 7]},{[3, 6], [1, 5]}];

p :=

  [8, {[1, 2, 4], [2, 5, 6], [3, 2], [4, 6], [7, 2], [8, 7]}, {[3, 6], [1, 5]}]

> dual(p);

     [8, [[1], [2, 1, 3, 7], [3], [4, 1], [5, 2], [6, 2, 4], [7, 8], [8]],

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

SEE ALSO: `poset, posetalgebra, isodual`

2.17 eulerform - the Euler-form of a completely separating bound poset algebra

Calling Sequence:
`eulerform`(p)
Parameters:
p – a bound poset algebra
Synopsis:

- `eulerform`(p) returns the symmetric integer coefficient matrix of the Euler-form of p.

Examples:

```
> p:=[11,[[1,2],[2,3,4],[3,5,6,7],[4,5,8],[5,9],[8,9,10],[9,11],[10,11]],
>        [[1,5],[2,8],[4,10]]];

 p := [11,

     [[1, 2], [2, 3, 4], [3, 5, 6, 7], [4, 5, 8], [5, 9], [8, 9, 10], [9, 11],

         [10, 11]],

     [[1, 5], [2, 8], [4, 10]]]

> chi:=eulerform(p);
```

```
[  2  -1   0   0   1   0   0   0  -1   0   0 ]
[                                            ]
[ -1   2  -1  -1   1   0   0   1   0  -1   0 ]
[                                            ]
[  0  -1   2   0  -1  -1  -1   0   0   0   0 ]
[                                            ]
[  0  -1   0   2  -1   0   0  -1   1   1   0 ]
[                                            ]
[  1   1  -1  -1   2   0   0   0  -1   0   0 ]
[                                            ]
[  0   0  -1   0   0   2   0   0   0   0   0 ]
[                                            ]
[  0   0  -1   0   0   0   2   0   0   0   0 ]
[                                            ]
[  0   1   0  -1   0   0   0   2  -1  -1   1 ]
[                                            ]
[ -1   0   0   1  -1   0   0  -1   2   0  -1 ]
[                                            ]
[  0  -1   0   1   0   0   0  -1   0   2  -1 ]
[                                            ]
[  0   0   0   0   0   0   0   1  -1  -1   2 ]
```

```
> h:=vector(11,[n,2*n,2*n+1,n,1,n,n,0,1,1,1]);

        h := [ n, 2 n, 2 n + 1, n, 1, n, n, 0, 1, 1, 1 ]

> 1/2*innerprod(h,chi,h);

                          - n + 1
```

SEE ALSO: `cartanm, coxeterm, titsform, unitform`

2.18 hcd_num - the numbers of frames of completely separating hypercritical algebras of type $\tilde{\tilde{D}}$.

Calling Sequence:
`hcd_num(`n`)`
Parameters:
n – 4,5,6,7, or 8
Synopsis:

- `hcd_num(`n`)` returns the number of frames of completely separating hypercritical algebras of type $\tilde{\tilde{D}}_n$.

Examples:

```
> hcd_num(4)
```

2

SEE ALSO: hce_num, hypercritical_algd

2.19 hce_num - the numbers of frames of completely separating hypercritical algebras of type $\tilde{\tilde{E}}$.

Calling Sequence:

hce_num(n)

Parameters:

n – 6,7, or 8

Synopsis:

- hce_num(n) returns the number of frames of completely separating hypercritical algebras of type $\tilde{\tilde{E}}_n$.

Examples:

```
> hce_num(7);
```

29

SEE ALSO: hcd_num, hypercritical_alge

2.20 hypercritical_algd - representatives of frames of completely separating hypercritical algebras of type $\tilde{\tilde{D}}$

Calling Sequence:

hypercritical_algd(n,m)

Parameters:

n – 4,5,6,7 or 8

m – a positive integer less than or equal to hcd_num(n)

Synopsis:

- hypercritical_algd(n, m) returns an algebra which belongs to the m-th frame of completely separating hypercritical algebras of type $\tilde{\tilde{D}}_n$.
- The number of frames available may by obtained by calling the function hcd_num.

Examples:

```
> hypercritical_algd(4,2);
```

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

SEE ALSO: critical_alge, hcd_num, hypercritical_alge, ishypercriticald

2.21 hypercritical_alge - representatives of frames of completely separating hypercritical algebras of type $\tilde{\tilde{E}}$

Calling Sequence:

hypercritical_alge(n,m)

Parameters:

n – 6,7 or 8

m – a positive integer less than or equal to hce_num(n)

Synopsis:

- `hypercritical_alge`(n, m) returns an algebra which belongs to the m-th frame of completely separating hypercritical algebras of type $\tilde{\tilde{E}}_n$.
- The number of frames available may by obtained by calling the function `hce_num`.

Examples:

```
> hypercritical_alge(7,13);
 [9, {[6], [7, 2], [8, 7], [3, 6], [2, 4, 5], [5], [1, 2, 3], [4, 6], [9, 3]},

    {[1, 5]}]
```

SEE ALSO: `critical_alge`, `hce_num`, `hypercritical_algd`, `ishypercriticale`

2.22 iscriticald - test for a tame concealed algebra of type $\tilde{D}$.

Calling Sequence:
`iscriticald`(p);
Parameters:
p – a bound poset algebra
Synopsis:

- `iscriticald`(p) tests whether the bound poset algebra p is tame concealed of type $\tilde{D}$.

Examples:

```
> iscriticald([5,{[1,2,3,4,5]}]);
                                        true

> iscriticald([5,{[1,2,3,4],[4,5]}]);
                                       false
```

SEE ALSO: `posetalgebra`

2.23 iscriticale - test for a tame concealed algebra of type $\tilde{E}$.

Calling Sequence:
`iscriticale`(p)
Parameters:
p – a bound poset algebra
Synopsis:

- `iscriticale`(p) tests whether the bound poset algebra p is tame concealed of type $\tilde{E}$. If so, the function will return a pair $[n, m]$, such that p and the algebra given by the call of `critical_alge`(n, m) have the same frame. Otherwise the value `false` will be returned.

Examples:

```
> p:=[8,{[2, 1, 3, 7], [4, 1], [5, 2], [6, 2, 4], [7, 8]},{[6, 3], [5, 1]}];

 p := [8, {[7, 8], [2, 1, 3, 7], [4, 1], [5, 2], [6, 2, 4]}, {[6, 3], [5, 1]}]

> iscriticale(p);
                                   [7, 14]

> a:=critical_alge(7,14);
    [8, {[5], [1, 2, 4], [2, 5, 6], [3, 2], [4, 6], [6], [7, 2], [8, 7]},

        {[3, 6], [1, 5]}]

> sameframe(a,p);
                                    true
```

SEE ALSO: `critical_alge`, `sameframe`, `posetalgebra`

2.24 ishypercriticald - test for a hypercritical algebra of type $\tilde{\tilde{D}}$

Calling Sequence:
`ishypercriticald`(p)

Parameters:
p – a bound poset algebra

Synopsis:

- `ishypercriticald`(p) tests whether the bound poset algebra p is hypercritical of type $\tilde{\tilde{D}}$. If so, the function will return a pair $[n, m]$, such that p and the algebra given by the call of `hypercritical_algd`(n, m) have the same frame. Otherwise the value `false` will be returned.

Examples:

```
> p:=[6,{[6,5,4,3,2,1]}];

                        p := [6, {[6, 5, 4, 3, 2, 1]}]

> ishypercriticald(p);
                                   [4, 2]

> a:=hypercritical_algd(4,2);
              [6, {[6], [1, 2, 3, 4, 5, 6], [2], [3], [4], [5]}]

> sameframe(a,p);
                                    true
```

SEE ALSO: `iscriticald`, `iscriticale`, `hypercritical_algd`, `ishypercriticale`

2.25 ishypercriticale - test for a hypercritical algebra of type $\tilde{\tilde{E}}$

Calling Sequence:
`ishypercriticale(`p`)`
Parameters:
p – a bound poset algebra
Synopsis:

- `ishypercriticale`(p) tests whether the bound poset algebra p is hypercritical of type $\tilde{\tilde{E}}$. If so, the function will return a pair $[n, m]$, such that p and the algebra given by the call of `hypercritical_alge`(n, m) have the same frame. Otherwise the value `false` will be returned.

Examples:

```
> p:=[9,{[1,2,3],[2,4,5],[3,6],[6,5,8],[7,6],[9,8]},{[7,5]}];

 p := [9, {[3, 6], [2, 4, 5], [1, 2, 3], [6, 5, 8], [7, 6], [9, 8]}, {[7, 5]}]

> ishypercriticale(p);
                                  [7, 13]

> a:=hypercritical_alge(7,13);
 [9, {[6], [7, 2], [8, 7], [3, 6], [2, 4, 5], [5], [1, 2, 3], [4, 6], [9, 3]},

     {[1, 5]}]

> sameframe(a,p);
                                   true
```

SEE ALSO: `iscriticald, iscriticale, hypercritical_alge, ishypercriticald`

2.26 isoalg - test whether two bound poset algebras are isomorphic

Calling Sequence:
`isoalg(`a,b`)`
Parameters:
a,b – bound poset algebras
Synopsis:

- `isoalg`(a, b) tests whether the two bound poset algebras a and b are isomorphic.

Examples:

```
> p:=critical_alge(7,14);
     [8, {[5], [7, 2], [8, 7], [3, 2], [6], [4, 6], [1, 2, 4], [2, 5, 6]},

         {[3, 6], [1, 5]}]

> q:=[8, {[1,2],[2,3],[3,4,8],[5,4],[6,3,5],[7,3]}, {[6,8],[7,4]}];
```

```
q :=

  [8, {[1, 2], [2, 3], [3, 4, 8], [5, 4], [6, 3, 5], [7, 3]}, {[6, 8], [7, 4]}]

> isoalg(p,q);
                                     true
```

SEE ALSO: `isodual`

2.27 isodual - test whether two algebras are dual to one another

Calling Sequence:
`isodual(`a,b`)`
Parameters:
a,b – bound poset algebras
Synopsis:

- `isodual`(a, b) tests whether the two bound poset algebras a and b are dual to one another.

Examples:

```
> p:=critical_alge(7,14);
     [8, {[5], [7, 2], [8, 7], [3, 2], [6], [4, 6], [1, 2, 4], [2, 5, 6]},

         {[3, 6], [1, 5]}]

> q:=dual(p);

  q := [8, [[1], [2, 7, 3, 1], [3], [4, 1], [5, 2], [6, 4, 2], [7, 8], [8]],

       [[6, 3], [5, 1]]]

> isoalg(p,q);
                                    false

> isodual(p,q);
                                     true
```

SEE ALSO: `dual, isoalg`

2.28 mchpg - test an algebra for being of polynomial growth

Calling Sequence:
`mchpg(`a`)`
`mchpg(`a,s,p,m,c,$incl$,img`)`
Parameters:
a – a completely separating bound poset algebra
s,p,m,c,$incl$,img – optional arguments evaluating to names
Synopsis:

- `mchpg` tests whether its first argument, the bound poset algebra a, is of polynomial growth.

- For `mchpg` to work properly, the algebra a has to be completely separating.
- Testing is done by constructing for each point s of the algebra the poset of minimal chain sequences starting in s and then looking first for minimal wild (or hypercritical) subposets there, and then for a minimal tame subposet.
- If the algebra is not of polynomial growth, various data will be returned through the optional arguments. In particular:
  - s - will be a point for which there is a poset of minimal chain sequences that is not of polynomial growth.
  - p - will contain the poset structure of the poset belonging to s.
  - m - a table containing for each minimal chain sequence in p the corresponding s-start set.
  - c - will give the name of the hypercritical subposet or of the minimal tame non polynomial growth subposet of p found by the function.
  - $incl$ - a list containing data about the inclusion $c \to p$ found by the function. In fact $incl[i]$ is the image in p of the point i of c under this inclusion.
  - img - a vector marking the points of p belonging to the image of c by 1, those not belonging to the image by 0.

Examples:

```
> a:=[8,{[3,1,2],[4,3,8],[5,4],[6,5],[7,5]},{[6,8],[7,8]}];

   a := [8, {[4, 3, 8], [6, 5], [7, 5], [3, 1, 2], [5, 4]}, {[7, 8], [6, 8]}]

> mchpg(a,'s','p','m','c','incl','img');
                                false

> print(s);

                                  3

> print(p);

          [7, {[7], [4], [5], [1, 2, 3], [3, 4, 5], [6], [2, 4, 5]}]

> print(m);

                       table([
                           1 = {3}
                           2 = {3, 4, 5, 8}
                           3 = {3, 4}
                           4 = {3, 4, 5, 7}
                           5 = {3, 4, 5, 6}
                           6 = {1, 3, 4, 5, 6, 7}
                           7 = {2, 3, 4, 5, 6, 7}
                       ])
```

```
> print(c);
```

cr_NZ

```
> print(incl);
```

[2, 3, 4, 5, 6, 7]

```
> print(img);
```

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

SEE ALSO: `mchrepfin`, `mchwild`, `pspg`, `rstartsets`, `subposet`, `crep/predefined_variables`, `complsep`

2.29 mchrepfin - test an algebra for being representation finite

Calling Sequence:
`mchrepfin(`*a*`)`
`mchrepfin(`*a,s,p,m,c,incl,img*`)`

Parameters:
a – a completely separating bound poset algebra
s,p,m,c,incl,img – optional arguments evaluating to names

Synopsis:

- `mchrepfin` tests whether its first argument, the bound poset algebra a, is of finite representation type.
- For `mchrepfin` to work properly, the algebra a has to be completely separating.
- Testing is done by constructing for each point s of the algebra the poset of minimal chain sequences starting in s and then testing this poset for finite representation type by looking for minimal non-representation finite (or critical) subposets.
- If the algebra is not of finite representation type, various data will be returned through the optional arguments. In particular:
  - s - will be a point for which there is a poset of minimal chain sequences that is not of finite representation type.
  - p - will contain the poset structure of the poset belonging to s.
  - m - a table containing for each minimal chain sequence in p the corresponding s-start set.
  - c - will give the name of the critical subposet of p found by the function.
  - *incl* - a list containing data about the inclusion $c \to p$ found by the function. In fact *incl*[i] is the image in p of the point i of c under this inclusion.
  - *img* - a vector marking the points of p belonging to the image of c by 1, those not belonging to the image by 0.

Examples:

```
> a:=critical_alge(7,10);
     [8, {[3, 4], [4, 5], [5], [1, 2, 3], [8, 5], [6, 2], [2, 5], [7, 3]}]

> mchrepfin(a,'s','p','m','c','incl','img');
                                    false

> print(s);

                                      2

> print(p);

[10, {[1, 3, 4], [4, 2], [5], [8, 7], [2], [3, 2], [9, 3, 6], [10, 6], [6, 4],

         [7, 1, 9]}

    ]

> print(m);

                        table([
                            1 = {2, 6}
                            2 = {1, 2, 3, 6, 7}
                            3 = {1, 2, 3, 6}
                            4 = {1, 2, 3, 4, 6, 7}
                            5 = {1, 2}
                            6 = {1, 2, 3, 4, 5, 6, 7, 8}
                            7 = {2, 4, 5, 6, 8}
                            8 = {2, 5, 6, 8}
                            9 = {1, 2, 3, 4, 5, 6, 8}
                            10 = {1, 2, 3, 4, 5, 6, 7}
                        ])

> print(c);

                                   cr_K[1]

> print(incl);

                                 [1, 5, 9, 10]

> print(img);

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

SEE ALSO: `mchpg`, `mchwild`, `psrepfin`, `rstartsets`, `subposet`, `crep/predefined_variables`, `complsep`

2.30 mchwild - for a given bound poset algebra look for a poset of minimal chain sequences which is wild

Calling Sequence:

`mchwild(`*a*`)`

`mchwild(`*a,s,,p,m,c,incl,img*`)`

Parameters:

a	– a completely separating bound poset algebra
s,p,m,c,incl,img	– optional arguments evaluating to names

Synopsis:

- `mchwild` constructs for each point s of the bound poset algebra a the poset of minimal chain sequences starting in s and tests whether this is wild.
- For `mchwild` to work properly, the algebra a has to be completely separating.
- Testing of the posets is done by looking for minimal wild (or hypercritical) subposets.
- If the procedure finds a wild poset, various information will be returned via the optional arguments. In particular:
  - s - will be a point for which there is a poset of minimal chain sequences that is of wild representation type.
  - p - will contain the poset structure of the poset belonging to s.
  - m - a table containing for each minimal chain sequence in p the corresponding s-start set.
  - c - will give the name of the hypercritical subposet of p found by the function.
  - *incl* - a list containing data about the inclusion $c \to p$ found by the function. In fact $incl[i]$ is the image in p of the point i of c under this inclusion.
  - *img* - a vector marking the points of p belonging to the image of c by 1, those not belonging to the image by 0.

Examples:

```
> a:=hypercritical_algd(7,2);
   [9, {[5, 2], [6, 3], [9, 2], [2, 1], [3, 4], [8, 7], [4, 1], [1], [7, 3]}]

> mchwild(a,'s','p','m','c','incl','img');
                                    true

> print(s);

                                     1

> print(p);

        [8, {[7], [8], [1, 4, 8], [4], [6, 7], [5], [3, 2], [2, 5, 6]}]
```

```
> print(m);
```

```
table([
    1 = {1, 3, 4, 6, 7, 8}
    2 = {1, 2, 4, 5, 9}
    3 = {1, 2, 5, 9}
    4 = {1, 2, 3, 4, 6, 7, 8, 9}
    5 = {1, 2, 3, 4, 5, 7, 8, 9}
    6 = {1, 2, 3, 4, 5, 6, 9}
    7 = {1, 2, 3, 4, 5, 6, 7, 9}
    8 = {1, 2, 3, 4, 5, 6, 7, 8}
])
```

```
> print(c);
```

```
cr_N[2]
```

```
> print(incl);
```

```
[4, 5, 8, 6, 7]
```

```
> print(img);
```

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

SEE ALSO: `mchrepfin`, `mchpg`, `pswild`, `rstartsets`, `subposet`, `crep/predefined_variables`, `complsep`

2.31 preproj - compute the beginning of the postprojective component of a bound poset algebra

Calling Sequence:
`preproj(`p`)`
Parameters:
p – a bound poset algebra
Synopsis:

- `preproj` passes the algebra p to an external program, that then tries to compute the beginning of a preprojective component of the Auslander Reiten quiver of p.
- The program enters an interactive menue, asking for various additional information concerning its output.

  First the program asks, whether you wish to see the dimension vectors of the modules themselves, their tops or their socles, or whether you wish to terminate execution. During execution the program may repeatedly return to that menue. If you enter '0' here, you will leave the program and return to your CREP/Maple session.

Next you may enter a visualization of the algebra p in such a way, that you may arrange the points of the algebra on a rectangular grid. If you choose this option, you are first asked for the size of the grid. Having answered that, you are asked to arrange the points of the algebra (labelled from 1 to $p[1]$ - the number of points), filling vacant places with zeros.

Next the program tells you, whether the algebra is separating, ('separating') whether it was able to compute a complete preprojective component containing all projectives and injectives ('repfinite') and whether it did stop knitting the component at some place ('cut').

You then may take influence on the way, the Auslander Reiten quiver is printed in output. The orbits of the Auslander Reiten quiver are printed in columns, where two orbits may share the same column.

If you choose to reorder the orbits, you first are asked for the number of columns (' length of section ?'). Then you may distribute orbits among the columns. Finally, being prompted for 'insertions' you may state, into which columns to fit the remaining orbits, giving first the number of the column, then that of the orbit. The neighbourhood-list of the orbitgraph of the Auslander-Reiten quiver printed before entering that menue may give a hint on how to rearrange the orbits.

Finally you are asked for the number of Auslander Reiten translation to print.

- Output of the program is to a file called `preproj.res`, in the CREP home directory. (This is the directory Maple has to be invoked from for use of CREP.)
- All dialog will happen in the environment Maple was called from. So make sure, your current Maple session is not running in the background, when invoking `preproj` from Maple.
- The program `preproj` can also be invoked from the CREP home directory without using maple. In this case, the program also prompts for interactive input of the algebra p.

Examples:

```
> a:=[4,[[1,2,3,4]]];

                          a := [4, [[1, 2, 3, 4]]]


> preproj(a);
which output (0=none,1=dim,2=top,3=soc,4=top+soc)?
1
visualization of the algebra (y/n) ?
y
number of rows ?
2
number of columns ?
3
     1  2  3
  1  2  3  4
  2  0  1  0
```

```
separating repfinite       cut
      true      true     false
orbit graph (by neighbours):
 1: 2 3 4
 2: 1
 3: 1
 4: 1
graduation of the orbit graph (start and stop numbers)
  1  2  3  4
  0  1  1  1
  4  5  5  5
reordering of the tau-orbits (y/n) ?
y
length of section ?
3
  1  2  3
  2  3  4
insertions ?
2 1

number of tau translations ?
10
sincere !
more output (0=none,1=dim,2=top,3=soc,4=top+soc)?
0
                                      done
```

SEE ALSO: `preprojposet`

2.32 preprojposet - compute the beginning of the postprojective component for a poset

Calling Sequence:
`preprojposet(`p`)`
Parameters:
p – a poset
Synopsis:

- `preprojposet` passes the poset p to an external program, that then tries to compute the beginning of a preprojective component of the Auslander Reiten quiver of p.

  First the program asks, whether you wish to see the dimension vectors (enter `'d'`) or the coordinate vectors (enter `'c'`) of the representations.

  You next may take influence on the way, the Auslander Reiten quiver is printed in output. The orbits of the Auslander Reiten are printed in columns, one column for each orbit. If you answer the question `'Permutation (y/n) ?'` with `'y'`, you are asked to specify the order in which the orbits are arranged.

- Output of the program is to a file called `preprojposet.res` (or `preprojp.res`) in the CREP home directory. (This is the directory Maple has to be invoked from for use of CREP.)
- All dialog will happen in the environment, Maple was called from. So make sure, your current Maple session is not running in the background, when invoking `preprojposet` from Maple.
- The program `preprojposet` can also be invoked from the CREP home directory without using maple. In this case, the program also prompts for interactive input of the poset p.

Examples:

```
> a:=[4, [[1, 2, 3, 4]]];

                                a := [4, [[1, 2, 3, 4]]]

> preprojposet(a);
compute coordinate vectors (c)
     or  dimension vectors (d) ?
d
Permutation (y/n) ?
y
  0  1  2  3  4
  3  0  2  1  4
                                          done
```

SEE ALSO: `preproj`

2.33 pspg - test a poset for polynomial growth

Calling Sequence:
`pspg(`p`)`
`pspg(`$p,c,incl,img$`)`
Parameters:
p – a poset
c,incl,img – optional arguments evaluating to names
Synopsis:

- `pspg` tests, whether the poset p is of polynomial growth.
- Testing is done by first looking for hypercritical subposets and then for a minimal non-polynomial growth subposet. (Criterion of Nazarova-Zavadskij)
- If the poset is not of polynomial growth, various data will be returned via the optional arguments. In particular:
  - c - will give the name of the hypercritical subposet or of the minimal tame non polynomial growth subposet of p found by the function.
  - *incl* - a list containing data about the inclusion $c \to p$ found by the function. In fact $incl[i]$ is the image in p of the point i of c under this inclusion.

– *img*- a vector marking the points of p belonging to the image of c by 1, those not belonging to the image by 0.

Examples:

```
> p:= [7,{[1, 2, 3], [3, 4, 5], [2, 4, 5]}];

                  p := [7, {[1, 2, 3], [3, 4, 5], [2, 4, 5]}]

> pspg(p,'c','incl','img');
                                    false

> print(c);

                                    cr_NZ

> print(incl);

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

> print(img);

                              [2, 3, 4, 5, 6, 7]
```

SEE ALSO: `psrepfin, pswild, subposet, crep/predefined_variables`

2.34 psrepfin - test a poset for finite representation type

Calling Sequence:
`psrepfin(`p`)`
`psrepfin(`$p,c,incl,img$`)`
Parameters:
p – a poset
$c,incl,img$ – optional arguments evaluating to names
Synopsis:

- `psrepfin` tests, whether the poset p is of finite representation type.
- Testing is done by looking for critical subposets of p. (Criterion of Klejner)
- If the poset is not of finite representation type, various data will be returned via the optional arguments. In particular:
  - c - will give the name of the critical subposet of p found by the function.
  - *incl* - a list containing data about the inclusion $c \to p$ found by the function. In fact $incl[i]$ is the image in p of the point i of c under this inclusion.
  - *img* - a vector marking the points of p belonging to the image of c by 1, those not belonging to the image by 0.

Examples:

```
> p:=[10,{[1, 3, 4], [4, 2], [3, 2], [9, 3, 6], [10, 6], [6, 4], [7, 1, 9]}];

 p := [10, {[1, 3, 4], [4, 2], [3, 2], [9, 3, 6], [10, 6], [6, 4], [7, 1, 9]}]

> psrepfin(p,'c','incl','img');
                                   false

> print(c);

                                  cr_K[1]

> print(incl);

                                [1, 5, 6, 8]

> print(img);

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

SEE ALSO: `pspg, pswild, subposet, crep/predefined_variables`

2.35 pswild - test a poset for wild representation type

Calling Sequence:

`pswild`(p)
`pswild`(p,c,$incl$,img)

Parameters:

p – a poset
c,$incl$,img – optional arguments evaluating to names

Synopsis:

- `pswild` tests, whether the poset p is of wild representation type.
- Testing is done by looking for hypercritical subposets of p. (Criterion of Nazarowa)

  - If the poset is of wild representation type, various data will be returned via the optional arguments. In particular:

  - c - will give the name of the hypercritical subposet of p found by the function.
  - $incl$ - a list containing data about the inclusion $c \to p$ found by the function. In fact $incl[i]$ is the image in p of the point i of c under this inclusion.
  - img - a vector marking the points of p belonging to the image of c by 1, those not belonging to the image by 0.

Examples:

```
> p:=[8,{[1, 4, 8], [6, 7], [3, 2], [2, 5, 6]}];
```

```
p := [8, {[1, 4, 8], [6, 7], [3, 2], [2, 5, 6]}]
```

```
> pswild(p,'c','incl','img');
                                  true

> print(c);

                                 cr_N[2]

> print(incl);

                             [4, 5, 8, 6, 7]

> print(img);

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

SEE ALSO: psrepfin, pspg, subposet, crep/predefined_variables

2.36 rdrwpst - repeat the last drawing made by drwpst

Calling Sequence:

rdrwpst()
rdrwpst(l)
rdrwpst(s)
rdrwpst(l,s)

Parameters:

l – a list
s – a set of positive integers

Synopsis:

- rdrwpst repeats the last drawing of a poset or bound poset algebra made by drwpst (or rdrwpst itself).
- before using rdrwpst you should at least once have called drwpst.
- The output of rdrwpst may be modified by optional arguments (s, l) in the same way as that of drwpst. This changes the marking and labelling of the points but not the shape of the poset or bound poset algebra that was last drawn using the function drwpst.

Examples:

```
> p:=[9, {[5, 6], [7, 2], [3, 2], [4, 1], [6], [8, 1, 9], [9, 3, 7], [1, 3]}];

 p := [9, {[3, 2], [6], [7, 2], [5, 6], [1, 3], [4, 1], [8, 1, 9], [9, 3, 7]}]

> drwpst(p);

> subposet(cr_K[2],p,'incl','img');
                                  true
```

```
> print(img,incl);

                [1, 0, 0, 1, 1, 1, 1, 0, 1], [4, 5, 9, 1, 6, 7]

> s:=convert(incl,set);

                          s := {1, 4, 5, 6, 7, 9}

> rdrwpst(s);

> rdrwpst(img);
```

SEE ALSO: `drwpst`

2.37 readalg - interactive input of a bound poset algebra

Calling Sequence:
`readalg()`
Parameters:
none
Synopsis:

- `readalg` enters a dialog that allows to input a bound poset algebra interactively in the following way:

  First you are asked for the number of elements of the underlying poset.

  Next you are prompted for the upper neighbours of each element of the poset. For that purpose the elements are assumed to be the first positive integers from 1 up to the number of elements. Make sure, there are no trailing blanks at the end of the lists of upper neighbours for each point. If a point is maximal and thus has no upper neighbours, just type `<return>`.

  Finally you enter by pairs of points the definig zero relations. After having given all these relations, when the program waits for further input, terminate input with single `<return>`.

- as the data format for bound poset algebras without zero-relations is the same as for posets, `readalg` may also be used for interactive input of a poset.

Examples:

```
> a:=readalg();
Cardinality of the poset of the algebra ?
5
Input of the poset by upper neighbours :
  1  :  2 3
  2  :  4
  3  :  4 5
  4  :
  5  :
```

```
Input of the relation pairs :
1 5

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

> a;

            [5, {[4], [2, 4], [5], [1, 2, 3], [3, 4, 5]}, {[1, 5]}]
```

SEE ALSO: poset, posetalgebra

2.38 rsrdtest - test for regular representation directed algebras

Calling Sequence:
rsrdtest(p)
rsrdtest(p,t)
Parameters:
p – a bound poset algebra
t – an optional parameter evaluating to a name
Synopsis:

- `rsrdtest` tests whether the bound poset algebra p is regular representation directed.
- If p is regular representation directed, via the optional second argument the name of a family (Bo1 .. Bo24) of regular representation directed algebras that contains p is returned.

Examples:

```
> p:=[10,{[2,1,7],[3,2,6],[4,1],[5,4],[6,5],[7,8],[8,9],[9,10]}];

p :=

  [10, {[7, 8], [8, 9], [4, 1], [9, 10], [2, 1, 7], [3, 2, 6], [5, 4], [6, 5]}]

> rsrdtest(p);
                                   true

> rsrdtest(p,t);
                                   true

> print(t);

                                   Bo3
```

SEE ALSO: posetalgebra

2.39 rstartsets - the poset of start sets induced from minimal chain sequences

Calling Sequence:
rstartsets(p,s)
rstartsets(p,s,m)

Parameters:
  p – a bound poset algebra
  s – a positive integer $\leq$ p[1] (the number of points of p)
  m – an optional argument evaluating to a name

Synopsis:

- `rstartsets` determines the poset of s-start sets induced by minimal chain sequences starting in the point s of the bound poset algebra p.
- The poset structure will be returned as value of the function.
- A list of the s-start sets themselves may be returned via the optional parameter m.

Examples:

```
> p:= critical_alge(7,14);
     [8, {[5], [1, 2, 4], [2, 5, 6], [3, 2], [4, 6], [6], [7, 2], [8, 7]},

         {[1, 5], [3, 6]}]
> q:=rstartsets(p,2,'m');
       [8, {[4, 1], [2], [3], [1], [5, 3], [6, 1, 2], [7, 4], [8, 4, 6]}]

> print(m);

                           table([
                               1 = {1, 2, 3, 4, 7, 8}
                               2 = {1, 2, 7, 8}
                               3 = {1, 2, 3, 7}
                               4 = {2, 3, 7, 8}
                               5 = {1, 2, 3}
                               6 = {1, 2, 4, 6, 7, 8}
                               7 = {2, 3, 5, 7, 8}
                               8 = {2, 6, 7, 8}
                           ])
> h:=convert(m,list);

h := [{1, 2, 3, 4, 7, 8}, {1, 2, 7, 8}, {1, 2, 3, 7}, {2, 3, 7, 8}, {1, 2, 3},

    {1, 2, 4, 6, 7, 8}, {2, 3, 5, 7, 8}, {2, 6, 7, 8}]

> drwpst(q,h);
```

SEE ALSO: `mchpg`, `mchrepfin`, `mchwild`

2.40 sameframe - determine whether two bound poset algebras have the same frame

Calling Sequence:
  `sameframe`(p,q)

Parameters:
  p,q – bound poset algebras

Synopsis:

- `sameframe`(p, q) tests whether the two bound poset algebras p and q have the same frame.

Examples:

```
> p:=[5,{[1,2],[2,3,4,5]},{[1,4]}];

                  p := [5, {[1, 2], [2, 3, 4, 5]}, {[1, 4]}]

> q:=[5,{[1,2,3],[4,1],[5,4]}];

                  q := [5, {[4, 1], [5, 4], [1, 2, 3]}]

> sameframe(p,q);
                                   true

> isoalg(p,q);
                                   false
```

SEE ALSO: `iosalg, isodual`

2.41 subposet - look for a given subposet within a poset

Calling Sequence:
`subposet(`*q,p*`)`
`subposet(`*q,p,incl*`)`
`subposet(`*q,p,incl,img*`)`
Parameters:
q,p – posets
incl, img – optional arguments evaluating to a name
Synopsis:

- `subposet` tests whether the poset q (its first argument) is a full subposet of the poset p (its second argument).
- if an inclusion of q into p as full subposet can be found, additional data about that inclusion may be obtained via the optional parameters *incl* and *img*. In particular:
  - *incl* - a list containing data about the inclusion $q \to p$ found. In fact $incl[i]$ is the image in p of the point i of q under this inclusion.
  - *img* - a vector marking the points of p belonging to the image of q by 1, those not belonging to the image by 0.

Examples:

```
> p:=[9, {[5, 6], [7, 2], [3, 2], [4, 1], [6], [8, 1, 9], [9, 3, 7], [1, 3]}];

 p := [9, {[5, 6], [7, 2], [3, 2], [4, 1], [6], [8, 1, 9], [9, 3, 7], [1, 3]}]

> subposet(cr_K[2],p,'incl','img');
                                   true
```

```
> print(img,incl);

               [1, 0, 0, 1, 1, 1, 1, 0, 1], [4, 5, 9, 1, 6, 7]
```

SEE ALSO: pspg, psrepfin, pswild, crep/predefined_variables

2.42 tctubes - compute the mouths of the exceptional tubes of a tame concealed bound poset algebra

Calling Sequence:

tctubes(p)

Parameters:

p – a bound poset algebra

Synopsis:

- For a tame concealed bound poset algebra, the dimension vectors of the modules in the mouths of the exceptional tubes are computed.
- tctubes returns list of lists, each list consisting of the dimension vectors of the mouth of one tube. The Auslander-Reiten translate of a module is given by the (cyclic) predecessor in the corresponding list.

Examples:

```
> d:=[7,{[1,3],[3,2,4],[4,5],[5,6],[7,5]}];

            d := [7, {[7, 5], [4, 5], [5, 6], [1, 3], [3, 2, 4]}]

> l:=tctubes(d);
    {[[0, 1, 1, 0, 0, 0, 0], [0, 0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 1, 0, 1],

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

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

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

> l[1];

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

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

> l[2];

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

> l[3];
```

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

SEE ALSO: iscriticald, iscriticale

2.43 titsform - the Tits-form of a completely separating bound poset algebra

Calling Sequence:
titsform(p)

Parameters:
p – a bound poset algebra

Synopsis:

- titsform(p) returns the symmetric integer coefficient matrix of the Tits-form of p.
- for titsform to work properly, the algebra p has to be completely separating.

Examples:

```
> p:= critical_alge(7,14);
    [8, {[1, 2, 4], [2, 5, 6], [3, 2], [4, 6], [6], [7, 2], [8, 7], [5]},

        {[1, 5], [3, 6]}]

> titsform(p);
                            [  2  -1   0  -1   1   1   0   0 ]
                            [                                ]
                            [ -1   2  -1   0  -1  -1  -1   0 ]
                            [                                ]
                            [  0  -1   2   0   0   1   0   0 ]
                            [                                ]
                            [ -1   0   0   2   0  -1   0   0 ]
                            [                                ]
                            [  1  -1   0   0   2   0   0   0 ]
                            [                                ]
                            [  1  -1   1  -1   0   2   0   0 ]
                            [                                ]
                            [  0  -1   0   0   0   0   2  -1 ]
                            [                                ]
                            [  0   0   0   0   0   0  -1   2 ]
```

SEE ALSO: complsep, weakly_positive, weakly_nonneg, titsform_p, unitform, eulerform

2.44 titsform_p - compute the quadratic form of a poset

Calling Sequence:
titsform_p(p)

Parameters:
p – a poset

Synopsis:

- `titsform_p`(p) returns the symmetric integer coefficient matrix of the unit form associated with the poset p.

Examples:

```
> cr_K[3];
```

```
[7, [[2, 4], [3, 6], [4, 5], [6, 7]]]
```

```
> titsform_p(");
```

```
[  2   0   0   0   0   0   0  -1 ]
[                                ]
[  0   2   0   1   1   0   0  -1 ]
[                                ]
[  0   0   2   0   0   1   1  -1 ]
[                                ]
[  0   1   0   2   1   0   0  -1 ]
[                                ]
[  0   1   0   1   2   0   0  -1 ]
[                                ]
[  0   0   1   0   0   2   1  -1 ]
[                                ]
[  0   0   1   0   0   1   2  -1 ]
[                                ]
[ -1  -1  -1  -1  -1  -1  -1   2 ]
```

SEE ALSO: `weakly_positive`, `weakly_nonneg`, `titsform`, `unitform`, `crep/predefined_variables`

2.45 tubetest - test for tubular extensions of $\tilde{E}_n$

Calling Sequence:

`tubetest`(p)
`tubetest`(p,sap)
`tubetest`(p,sap,tc)

Parameters:

p – a bound poset algebra
sap,tc – optional arguments evaluating to a name

Synopsis:

- `tubetest` tests whether the bound poset algebra p is a tubular extension of a tame concealed algebra of type $\tilde{E}_n$. If so, the function returns the tubular type of the algebra, else the value `false`.
- Testing is done in several steps. The results of the intermediary steps can be obtained via the optional arguments sap and tc.
- In case p is a branch algebra over the support algebra of the postprojective component of p, this support algebra will be returned via the optional argument sap. (Otherwise, if an optionl argument ist given, sap will get the value `false`.)

- Also, in case p is a branch extension over the support algebra of the postprojective component of p, the optional argument tc will tell whether this support algebra is tame concealed of type $\tilde{E}_n$.

Examples:

```
> p:=[8,{[1,2,3,4],[2,5,8],[3,6,8],[4,7,8]}];

          p := [8, {[1, 2, 3, 4], [2, 5, 8], [3, 6, 8], [4, 7, 8]}]

> tubetest(p,'sap','tc');
                                    [3, 3, 3]

> print(sap);

          [7, [[1, 2, 3, 4], [2, 5], [3, 6], [4, 7], [5], [6], [7]]]

> iscriticale(sap);
                                      [6, 1]

-----------------------------------------------------------------------
> p:=[8,{[1,2,3],[2,4],[3,4],[5,2],[6,3],[7,4],[2,8]},{[6,8],[7,8]}];

     p := [8, {[1, 2, 3], [5, 2], [6, 3], [7, 4], [2, 8], [2, 4], [3, 4]},

         {[6, 8], [7, 8]}]

> tubetest(p,'sap','tc');
                                      false

> print(sap);

     [8, [[1, 2, 3], [2, 4, 5, 6], [3, 4, 7], [4, 8], [5], [6], [7], [8]]]

> print(tc);

                                      false

-----------------------------------------------------------------------
> p:=[10,{[1,2,3,4],[2,5,8],[3,6,8],[4,7,8],[8,9,10]}];

     p := [10, {[8, 9, 10], [1, 2, 3, 4], [2, 5, 8], [3, 6, 8], [4, 7, 8]}]

> tubetest(p,'sap','tc');
                                      false

> print(sap);

                                      false
```

SEE ALSO: critical_alge, iscriticale

2.46 weakly_nonneg - test a unit form for weak non-negativity

Calling Sequence:

weakly_nonneg(q)

weakly_nonneg(q,$a1$,$a2$,$a3$)

Parameters:

q – a unit form

$a1$, $a2$, $a3$ – optional parameters evaluating to a name

Synopsis:

- weakly_nonneg tests, using an algorithm of J.A de la Peña and A. Dean, whether the unit form q given as first argument is weakly non-negative.
- in case q is weakly non-negative the first optional argument will contain the 1-roots and the second optional argument will contain the 0-roots constructed by the algorithm used. A possible third optional argument will be left untouched.
- in case q is not weakly non-negative, the last of the optional arguments will return a vector v with non-negative entries and $q(v) < 0$. The other optional arguments (if there are any) will contain the lists of the 1-roots and the 0-roots constructed by the algorithm.

Examples:

```
> a:=hypercritical_algd(4,1);
              [6, {[6], [2], [4, 1], [1, 2, 6], [3, 1], [5, 4]}]

> q:=titsform(a);
                         [ 2  -1  -1  -1   0  -1 ]
                         [                       ]
                         [ -1  2   0   0   0   0 ]
                         [                       ]
                         [ -1  0   2   0   0   0 ]
                         [                       ]
                         [ -1  0   0   2  -1   0 ]
                         [                       ]
                         [ 0   0   0  -1   2   0 ]
                         [                       ]
                         [ -1  0   0   0   0   2 ]

> weakly_nonneg(q,'a1','a2','a3');
                                 false

> print(a3);

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

> weakly_nonneg(q,'v');
```

```
                              false

> print(v);

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

> 1/2*innerprod(v,q,v);

                               -1
```

SEE ALSO: `unitform`, `weakly_positive`, `titsform`, `titsform_p`

2.47 weakly_positive - test a unit form for weak positivity

Calling Sequence:
`weakly_positive(`q`)`
`weakly_positive(`q,rt`)`

Parameters:
q – a unit form
rt – an optional parameter evaluating to a name

Synopsis:

- `weakly_positive` tests, whether the unit form q given as first argument is weakly positive.
- In case q is weakly positive, via the optional parameter rt a list of all positive 1-roots of q may be obtained (i.e. a list of all $v \geq 0$ with $q(v) = 1$).

Examples:

```
> p:=[4,{[1,2,3],[4,1]}];

                   p := [4, {[1, 2, 3], [4, 1]}]

> q:=titsform(p);
                        [  2  -1  -1  -1 ]
                        [                ]
                        [ -1   2   0   0 ]
                        [                ]
                        [ -1   0   2   0 ]
                        [                ]
                        [ -1   0   0   2 ]

> weakly_positive(q,'rt');
                              true
> print(rt);

                     table([
                         1 = [ 1, 0, 0, 0 ]
                         2 = [ 0, 1, 0, 0 ]
                         3 = [ 0, 0, 1, 0 ]
```

```
        4 = [ 0, 0, 0, 1 ]
        5 = [ 1, 1, 0, 0 ]
        6 = [ 1, 1, 1, 0 ]
        7 = [ 1, 1, 1, 1 ]
        8 = [ 2, 1, 1, 1 ]
        9 = [ 1, 1, 0, 1 ]
        10 = [ 1, 0, 1, 0 ]
        11 = [ 1, 0, 1, 1 ]
        12 = [ 1, 0, 0, 1 ]
    ])
```

SEE ALSO: `unitform, weakly_nonneg, titsform, titsform_p`

3 The esrd package

3.1 Using the esrd package

The esrd package is a supplement to CREP containing functions and data about the exceptional sincere representation directed algebras. It can be used only together with the CREP system.

To use the package, you thus first have to start CREP as described in section 1.1. Reading in the file `esrd.src` located in the CREP home directory using the command `read 'esrd.src'` will then make available the functions of the esrd package, provided the package is installed in the CREP home directory.

Instead of reading in the file `esrd.src` manually each time you want to use the esrd package, you might configure Maple so that it will automatically start CREP (by reading the file `crep.src`) and then read the definitions for the esrd package (by reading the file `esrd.src`). This usually is done by an initialisation file for Maple, located in the CREP home directory, containing the lines `read 'crep.src'` and `read 'esrd.src'`. (See the Maple reference manual for more detailed information.)

3.2 The CREP-package esrd for exceptional sincere representation directed algebras

Calling Sequence:
`<function>(`*args*`)`
Synopsis:

- The CREP-package esrd gives access to data about exceptional sincere representation directed algebras.
- The functions available are:

esrdalg	esrd_num	esrd_orbittypes
esrd_rootsa	esrd_rootss	esrdtest

- For more information on a particular function see `<function>`.

3.3 esrdalg - representatives of frames of exceptional sincere representation directed algebras

Calling Sequence:
`esrdalg(`o,n`)`
Parameters:
o – the orbit type of an exceptional sincere representation directed algebra
n – a positive integer less than or equal to `esrd_num`(o)
Synopsis:

- `esrdalg`(o, n) returns an algebra which belongs to the n-th frame of exceptional sincere representation directed algebra of orbit type o.
- The number of frames available may by obtained by calling the function `esrd_num`.
- See `esrd_orbittypes` for valid values of o.

Examples:

```
> esrdalg(21112,5);
    [8, {[7, 8], [3, 5], [2, 3, 4], [1, 2, 6], [4, 7], [5, 7], [6, 8], [8]}]
```

SEE ALSO: `esrd_num`, `esrd_orbittypes`, `esrdtest`

3.4 esrd_num - the numbers of frames of exceptional sincere representation directed algebras

Calling Sequence:
`esrd_num`(o)

Parameters:
o – the orbit type of an exceptional sincere representation directed algebra

Synopsis:

- `esrd_num`(o) returns the number of frames of exceptional sincere representation directed algebras whose Auslander Reiten quiver is of orbit type o.
- See `esrd_orbittypes` for valid values of o.

Examples:

```
> esrd_num(21112);

                                   7
```

SEE ALSO: `esrdalg`, `esrd_orbittypes`

3.5 esrd_orbittypes - the orbit types of the exceptional sincere representation directed algebras

Calling Sequence:
`esrd_orbittypes()`
`esrd_orbittypes`(n)

Parameters:
n – an integer between 1 and 38

Synopsis:

- If the function is called without any arguments, it will return a list of all orbit types of exceptional sincere representation directed algebras.
- Calling `esrd_orbittypes`(n) with n an integer between 1 and 38 will return the orbit type that is in the n-th place in the list of orbit types.

Examples:

```
> esrd_orbittypes();

 [21111, 21112, 21113, 21114, 21115, 21116, 21211, 21212, 21213, 21214, 221,

     2211, 222, 31111, 31112, 31211, 31212, 321, 3211, 322, 331, 332, 41111,
```

```
41112, 421, 4211, 422, 431, 432, 441, 521, 522, 531, 541, 621, 721, 821,

921]
```

```
> esrd_orbittypes(1);

                                   21111

> esrd_orbittypes(38);

                                    921
```

SEE ALSO: esrdalg, esrd_num

3.6 esrd_rootsa - positive roots of the Tits-form

Calling Sequence:
esrd_rootsa(*o*,*n*)
esrd_rootsa(*o*,*n*,*smin*)
esrd_rootsa(*o*,*n*,*smin*,*smax*)

Parameters:

o – the orbit type of an exceptional sincere representation directed algebra
n – a positive integer less than or equal to esrd_num(*o*)
smin,*smax* – optional parameters evaluating to a name

Synopsis:

- esrd_rootsa(*o*, *n*) returns for the class *n* of exceptional sincere representation directed algebras belonging to orbit type *o* a list of all positive roots of the corresponding Tits-form.
- Via the optional third parameter *smin* the unique minimal sincere positive root of the Tits-form may be obtained.
- Via the optional forth parameter *smax* a list of all maximal positive roots may be obtained.
- See esrd_orbittypes for valid values of *o*.

Examples:

```
> esrd_rootsa(221,3,'smin','smax');
                          table([
                              1 = [1, 0, 0, 0, 0, 0]
                              2 = [1, 1, 1, 0, 0, 0]
                              3 = [2, 2, 2, 1, 1, 1]
                              4 = [1, 2, 2, 1, 1, 1]
                              5 = [1, 2, 2, 2, 1, 1]
                              6 = [0, 1, 1, 1, 1, 1]
                              7 = [0, 0, 0, 1, 0, 0]
                              8 = [1, 1, 0, 0, 0, 0]
                              9 = [1, 1, 1, 0, 0, 1]
```

```
            10 = [1, 1, 2, 1, 1, 0]
            11 = [1, 2, 1, 1, 1, 1]
            12 = [0, 1, 1, 1, 0, 1]
            13 = [0, 0, 1, 1, 1, 0]
            14 = [1, 0, 1, 0, 0, 0]
            15 = [1, 1, 1, 0, 1, 0]
            16 = [1, 2, 1, 1, 0, 1]
            17 = [1, 1, 2, 1, 1, 1]
            18 = [0, 1, 1, 1, 1, 0]
            19 = [0, 1, 0, 1, 0, 1]
            20 = [1, 1, 1, 1, 0, 0]
            21 = [1, 1, 1, 0, 1, 1]
            22 = [0, 1, 1, 1, 0, 0]
            23 = [1, 1, 1, 1, 1, 1]
            24 = [1, 0, 1, 0, 1, 0]
            25 = [0, 1, 0, 0, 0, 0]
            26 = [1, 1, 1, 1, 0, 1]
            27 = [0, 0, 1, 0, 1, 0]
            28 = [0, 1, 0, 1, 0, 0]
            29 = [0, 0, 0, 0, 0, 1]
            30 = [1, 1, 0, 0, 0, 1]
            31 = [0, 0, 1, 0, 0, 0]
            32 = [1, 1, 1, 1, 1, 0]
            33 = [0, 1, 0, 0, 0, 1]
            34 = [0, 0, 1, 1, 0, 0]
            35 = [0, 0, 0, 0, 1, 0]
        ])
> print(smin);

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

> print(smax);

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

SEE ALSO: `esrd_rootss`

3.7 esrd_rootss - sincere positive roots of the Tits-form

Calling Sequence:

`esrd_rootss(`*o*,*n*`)`
`esrd_rootss(`*o*,*n*,*smin*`)`
`esrd_rootss(`*o*,*n*,*smin*,*smax*`)`

Parameters:

o	– the orbit type of an exceptional sincere representation directed algebra
n	– a positive integer less than or equal to `esrd_num`(*o*)
smin,*smax*	– optional parameters evaluating to a name

Synopsis:

- `esrd_rootss`(o, n) returns for the class n of exceptional sincere representation directed algebras belonging to orbit type o a list of all sincere positive roots of the corresponding Tits-form.
- Via the optional third parameter *smin* the unique minimal sincere positive root of the Tits-form may be obtained.
- Via the optional forth parameter *smax* a list of all maximal positive roots may be obtained.
- See `esrd_orbittypes` for valid values of o.

Examples:

```
> esrd_rootss(221,3);
                          table([
                              1 = [2, 2, 2, 1, 1, 1]
                              2 = [1, 2, 2, 1, 1, 1]
                              3 = [1, 2, 2, 2, 1, 1]
                              4 = [1, 2, 1, 1, 1, 1]
                              5 = [1, 1, 2, 1, 1, 1]
                              6 = [1, 1, 1, 1, 1, 1]
                          ])
```

SEE ALSO: `esrd_rootsa`

3.8 esrdtest - test for exceptional sincere representation directed algebras

Calling Sequence:
`esrdtest`(p)
Parameters:
p – a bound poset algebra
Synopsis:

- `esrdtest`(p) test whether the bound poset algebra p has the frame of an exceptional sincere representation directed algebra. If so, the function will return a pair $[o, n]$ such that p and the algebra given by the call of `esrdalg`(o, n) have the same frame. Otherwise the value `false` will be returned.

Examples:

```
> p:=[9,{[2,1],[3,2,4],[4,5,6],[6,7,9],[8,7]},{[3,7]}];

    p := [9, {[3, 2, 4], [4, 5, 6], [6, 7, 9], [8, 7], [2, 1]}, {[3, 7]}]

> esrdtest(p);
                                   [521, 2]

> q:=esrdalg(521,2);
  [9, {[7, 1], [8], [9, 7], [1, 2], [2, 3], [3, 4], [4, 8], [5, 3], [6, 2]},
```

```
    {[1, 4]}]

> sameframe(p,q);
                                    true

> isoalg(p,q);
                                   false
```

SEE ALSO: esrdalg, sameframe

4 The heralg package

The heralg package is a supplement for CREP dealing with hereditary algebras. It is written wholly in Maple and organised as a Maple library. It can thus also be used independently from CREP.

To make the functions of the package available, read in the file `heralg.src` located in the CREP home directory.

The heralg package was written by Stefan Lache during a stay at the University of Bielefeld in 1995, based on an earlier Pascal program by Frank Lukas.

4.1 Introduction to the heralg package

Calling Sequence:

`<function>`(*args*)

`heralg[<function>]`(*args*)

Synopsis:

- The `heralg` package is a collection of routines for calculations with quiver algebras.
- To use a heralg function, either define that function alone using the command `with(heralg, <function>)`, or define all `heralg` functions using the command `with(heralg)`. Alternatively, invoke the function using the long form `heralg[<function>]`.
- The functions available are:

cartan_matrix	cartan_inverse	coxeter_matrix
coxeter_inverse	tau_translate	homol_blf
coxeter_polynomial	coxeter_roots	growth_number
one_pt_cartmatrix	one_pt_coxmatrix	one_pt_coxpoly

- For more information on a particular function see `heralg[<function>]`.
- The heralg package uses the type `heralg/quiverdata` to describe quivers

4.2 the type heralg/quiverdata

Synopsis:

- The type `heralg/quiverdata` is used to describe quivers for use with the functions of the heralg package.
- A quiver Q is denoted by a pair

$$[n, l]$$

where n is an integer - the number of vertices of Q - and l is a list

$$l = [l_1, \ldots, l_k]$$

giving the arrows of Q in the following way:
Each entry l_i of l is a list

$$l_i = [s_i, t_i]$$

or

$$l_i = [s_i, t_i, r_i]$$

with two or three entries.
Here, s_i and t_i give the starting and ending point of the arrow(s), while the optional third entry r_i is the multiplicity, i.e. the number of arrows from s_i to t_i (if not 1).

Examples:

```
> # a quiver of type \tilde D_4:
>
> Q := [ 5, [ [1,3], [3,2], [3,4], [3,5] ] ];

                  Q := [5, [[1, 3], [3, 2], [3, 4], [3, 5]]]

> type(Q, `heralg/quiverdata`);

                                   true

> # the generalized Kronecker quiver with r arrows:
>
> type( [ 2, [ [ 1, 2, r ] ] ], `heralg/quiverdata`);

                                   true
```

4.3 heralg[cartan_matrix] - the Cartan matrix of a quiver algebra

Calling Sequence:
`cartan_matrix(`Q`)`
Parameters:
Q – a quiver given by a quiverdata structure
Synopsis:

- The function `cartan_matrix(`Q`)` computes the Cartan matrix of the path algebra of the quiver Q. The columns of the Cartan matrix are the dimension vectors of the indecomposable projective modules.
- This function is part of the heralg package, and so can be used in the form `cartan_matrix(..)` only after performing the command `with(heralg)` or `with(heralg, cartan_matrix)`. The function can always be accessed in the long form `heralg[cartan_matrix](..)`.

Examples:

```
> with(heralg):
> cartan_matrix( [2,[[1,2,r]]] );

                                 [ 1  0 ]
                                 [      ]
                                 [ r  1 ]

> Q := [ 3, [[1,2], [2,3,2]]];
```

```
                    Q := [3, [[1, 2], [2, 3, 2]]]
```

```
> cartan_matrix(Q);
```

```
                              [ 1  0  0 ]
                              [         ]
                              [ 1  1  0 ]
                              [         ]
                              [ 2  2  1 ]
```

SEE ALSO: `heralg/quiverdata`

4.4 heralg[cartan_inverse] - the inverse of the Cartan matrix of a quiver algebra

Calling Sequence:

`cartan_inverse(`Q`)`

Parameters:

Q – a quiver given by a quiverdata structure

Synopsis:

- The `function cartan_inverse`(Q) computes the inverse of the Cartan matrix of the path algebra of the quiver Q. The columns of the Cartan matrix are the dimension vectors of the indecomposable projective modules.
- This function is part of the heralg package, and so can be used in the form `cartan_inverse(..)` only after performing the command `with(heralg)` or `with(heralg, cartan_inverse)`. The function can always be accessed in the long form `heralg[cartan_inverse](..)`.

Examples:

```
> with(heralg):
> cartan_inverse( [2,[[1,2,r]]] );
```

```
                               [  1   0 ]
                               [        ]
                               [ - r  1 ]
```

```
> Q := [ 3, [[1,2], [2,3,2]]];
```

```
                    Q := [3, [[1, 2], [2, 3, 2]]]
```

```
> cartan_inverse(Q);
```

```
                             [  1   0  0 ]
                             [           ]
                             [ -1   1  0 ]
                             [           ]
                             [  0  -2  1 ]
```

SEE ALSO: heralg/quiverdata

4.5 heralg[coxeter_matrix] - the Coxeter matrix of a quiver algebra

Calling Sequence:
coxeter_matrix(Q)

Parameters:
Q – a quiver given by a quiverdata structure

Synopsis:

- The function coxeter_matrix(Q) computes the Coxeter matrix of the path algebra of the quiver Q. When applied to the dimension vector of a module without projective direct summand, the Coxeter matrix gives the dimension vector of the Auslander-Reiten translate of this module.
- This function is part of the heralg package, and so can be used in the form coxeter_matrix(..) only after performing the command with(heralg) or with(heralg, coxeter_matrix). The function can always be accessed in the long form heralg[coxeter_matrix](..).

Examples:

```
> with(heralg):
> coxeter_matrix( [2,[[1,2,r]]] );

                    [        2     ]
                    [ - 1 + r    r ]
                    [              ]
                    [    - r    -1 ]

> Q := [ 3, [[1,2], [2,3,2]]];

              Q := [3, [[1, 2], [2, 3, 2]]]

> coxeter_matrix(Q);

                    [  0   1   0 ]
                    [            ]
                    [  3   3   2 ]
                    [            ]
                    [ -2  -2  -1 ]
```

SEE ALSO: heralg/quiverdata

4.6 heralg[coxeter_inverse] - the inverse of the Coxeter matrix of a quiver algebra

Calling Sequence:
coxeter_inverse(Q)

Parameters:
Q – a quiver given by a quiverdata structure
Synopsis:

- The function `coxeter_inverse`(Q) computes the inverse of the Coxeter matrix of the path algebra of the quiver Q. When applied to the dimension vector of a module without injective direct summand, the inverse of the Coxeter matrix gives the dimension vector of the negative Auslander-Reiten translate of this module.
- This function is part of the heralg package, and so can be used in the form `coxeter_inverse(..)` only after performing the command `with(heralg)` or `with(heralg, coxeter_inverse)`. The function can always be accessed in the long form `heralg[coxeter_inverse](..)`.

Examples:

```
> with(heralg):
> coxeter_inverse( [2,[[1,2,r]]] );

                              [ -1      - r   ]
                              [               ]
                              [             2 ]
                              [  r   - 1 + r  ]

> Q := [ 3, [[1,2], [2,3,2]]];

                         Q := [3, [[1, 2], [2, 3, 2]]]

> coxeter_inverse(Q);

                               [ -1  -1  -2 ]
                               [            ]
                               [  1   0   0 ]
                               [            ]
                               [  0   2   3 ]
```

SEE ALSO: `heralg/quiverdata`

4.7 heralg[tau_translate] - the Coxeter transformation of the dimension vector of a module

Calling Sequence:
`tau_translate`(x,t,Q)
Parameters:
x – a vector
t – an integer
Q – a quiver given by a quiverdata structure
Synopsis:

- The function `tau_translate`(x,t,Q) applies the t'th power of the Coxeter matrix of the path algebra of the quiver Q to the vector x. Applying the Coxeter matrix to the dimension vector of a module without projective direct summand yields the dimension vector of the Auslander-Reiten translate of this module.

- This function is part of the heralg package, and so can be used in the form `tau_translate(..)` only after performing the command `with(heralg)` or `with(heralg, tau_translate)`. The function can always be accessed in the long form `heralg[tau_translate](..)`.

Examples:

```
> with(heralg):
> tau_translate( array([0,1]), -2, [2,[[1,2,r]]] );

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

> Q := [ 3, [[1,2], [2,3,2]]];

                    Q := [3, [[1, 2], [2, 3, 2]]]

> v := array([0,1,2]);

                             v := [ 0, 1, 2 ]

> tau_translate(v,1,Q);

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

SEE ALSO: `heralg/quiverdata`, `heralg[coxeter_matrix]`

4.8 heralg[homol_blf] - the homological bilinear form of two dimension vectors of modules

Calling Sequence:
`homol_blf`(x,y,Q)

Parameters:
x, y – vectors
Q – a quiver given by a quiverdata structure

Synopsis:

- The function `homol_blf`(x, y, Q) applies the homological bilinear form of the path algebra of the quiver Q to the vectors x and y. This bilinear form is determined by the inverse of the Cartan matrix of the algebra. If x and y are dimension vectors of modules X and Y, the bilinear form gives the difference $\dim \mathrm{Hom}(X, Y) - \dim \mathrm{Ext}^1(X, Y)$.

- This function is part of the heralg package, and so can be used in the form `homol_blf(..)` only after performing the command `with(heralg)` or `with(heralg, homol_blf)`. The function can always be accessed in the long form `heralg[homol_blf](..)`.

Examples:

```
> with(heralg):
> homol_blf( array([1,0]), array([0,1]), [2,[[1,2,r]]] );

                                   - r

> Q := [ 3, [[1,2], [2,3,2]]];

                        Q := [3, [[1, 2], [2, 3, 2]]]

> v := array([0,1,2]);

                              v := [ 0, 1, 2 ]

> w := array([3,3,4]);

                              w := [ 3, 3, 4 ]

> homol_blf(v,w,Q);

                                    3
```

SEE ALSO: heralg/quiverdata, heralg[cartan_matrix]

4.9 heralg[coxeter_polynomial] - the characteristic polynomial of the Coxeter matrix of a quiver algebra

Calling Sequence:
coxeter_polynomial(Q)
Parameters:
Q – a quiver given by a quiverdata structure
Synopsis:

- The function coxeter_polynomial(Q) computes the Coxeter polynomial, i.e. the characteristic polynomial of the Coxeter matrix of the path algebra of the quiver Q.
- This function is part of the heralg package, and so can be used in the form coxeter_polynomial(..) only after performing the command with(heralg) or with(heralg, coxeter_polynomial). The function can always be accessed in the long form heralg[coxeter_polynomial](..).

Examples:

```
> with(heralg):
> coxeter_polynomial( [2,[[1,2,r]]] );

                            2         2
                           T  + (2 - r ) T + 1
```

```
> Q := [ 3, [[1,2], [2,3,2]]];

                         Q := [3, [[1, 2], [2, 3, 2]]]

> coxeter_polynomial(Q);

                              3      2
                             T  - 2 T  - 2 T + 1
```

SEE ALSO: heralg/quiverdata, heralg[coxeter_matrix]

4.10 heralg[coxeter_roots] - the eigenvalues of the Coxeter matrix of a quiver algebra

Calling Sequence:
`coxeter_roots(`Q`)`

Parameters:
Q – a quiver given by a quiverdata structure

Synopsis:

- The function `coxeter_roots`(Q) computes all eigenvalues of the Coxeter matrix of the path algebra of the quiver Q.
- This function is part of the heralg package, and so can be used in the form `coxeter_roots(..)` only after performing the command `with(heralg)` or `with(heralg, coxeter_roots)`. The function can always be accessed in the long form `heralg[coxeter_roots](..)`.

Examples:

```
> with(heralg):
> coxeter_roots( [2,[[1,2,r]]] );

            2              2    4 1/2              2              2    4 1/2
  - 1 + 1/2 r  + 1/2 (- 4 r  + r )   , - 1 + 1/2 r  - 1/2 (- 4 r  + r )

> Q := [ 3, [[1,2], [2,3,2]]];

                         Q := [3, [[1, 2], [2, 3, 2]]]

> coxeter_roots(Q);

                    -1.000000000, 2.618033989, .3819660113
```

SEE ALSO: heralg/quiverdata, heralg[coxeter_matrix], heralg[growth_number]

4.11 heralg[growth_number] - the spectral radius of the Coxeter matrix of a quiver algebra

Calling Sequence:
`growth_number(`Q`)`

Parameters:
Q – a quiver given by a quiverdata structure
Synopsis:

- The function `growth_number`(Q) computes the growth number of the path algebra of the quiver Q. The growth number is the spectral radius of the Coxeter matrix of the algebra, i.e. the maximal absolute value of the eigenvalues of the Coxeter matrix.

- This function is part of the heralg package, and so can be used in the form `growth_number(..)` only after performing the command `with(heralg)` or `with(heralg, growth_number)`. The function can always be accessed in the long form `heralg[growth_number](..)`.

Examples:

```
> with(heralg):
> growth_number( [2,[[1,2,r]]] );

                              2              2    4 1/2
           max(abs(- 1 + 1/2 r  + 1/2 (- 4 r  + r )   ),

                              2              2    4 1/2
               abs(- 1 + 1/2 r  - 1/2 (- 4 r  + r )   ))

> Q := [ 3, [[1,2], [2,3,2]]];

                   Q := [3, [[1, 2], [2, 3, 2]]]

> growth_number(Q);

                            2.618033989
```

SEE ALSO: `heralg/quiverdata`, `heralg[coxeter_matrix]`, `heralg[coxeter_roots]`

4.12 heralg[one_pt_cartmatrix] - the Cartan matrix of a one-point extension of a quiver algebra

Calling Sequence:
`one_pt_cartmatrix(`Q,x`)`
Parameters:
Q – a quiver given by a quiverdata structure
x – a vector
Synopsis:

- The function `one_pt_cartmatrix`(Q, x) computes the Cartan matrix of the one-point extension of the path algebra of the quiver Q by a module with dimension vector x.

- This function is part of the heralg package, and so can be used in the form `one_pt_cartmatrix(..)` only after performing the command `with(heralg)` or `with(heralg, one_pt_cartmatrix)`. The function can always be accessed in the long form `heralg[one_pt_cartmatrix](..)`.

Examples:

```
> with(heralg):
> one_pt_cartmatrix( [2,[[1,2,r]]], array([1,1]) );

                                  [ 1  0  1 ]
                                  [         ]
                                  [ r  1  1 ]
                                  [         ]
                                  [ 0  0  1 ]

> Q := [ 3, [[1,2], [2,3,2]]];

                         Q := [3, [[1, 2], [2, 3, 2]]]

> x := array( [0,2,1] );

                                x := [ 0, 2, 1 ]

> one_pt_cartmatrix(Q,x);
                                 [ 1  0  0  0 ]
                                 [            ]
                                 [ 1  1  0  2 ]
                                 [            ]
                                 [ 2  2  1  1 ]
                                 [            ]
                                 [ 0  0  0  1 ]
```

SEE ALSO: `heralg/quiverdata`, `heralg[cartan_matrix]`

4.13 heralg[one_pt_coxmatrix] - the Coxeter matrix of a one-point extension of a quiver algebra

Calling Sequence:

`one_pt_coxmatrix(`Q,x`)`

Parameters:

Q – a quiver given by a quiverdata structure

x – a vector

Synopsis:

- The function `one_pt_coxmatrix`(Q, x) computes the Coxeter matrix of the one-point extension of the path algebra of the quiver Q by a module with dimension vector x.
- This function is part of the heralg package, and so can be used in the form `one_pt_coxmatrix(..)` only after performing the command `with(heralg)` or `with(heralg, one_pt_coxmatrix)`. The function can always be accessed in the long form `heralg[one_pt_coxmatrix](..)`.

Examples:

```
> with(heralg):
> one_pt_coxmatrix( [2,[[1,2,r]]], array([1,1]) );

                    [         2                  ]
                    [  - 1 + r       r    - 1 + r ]
                    [                            ]
                    [     - r       -1      -1   ]
                    [                            ]
                    [          2                 ]
                    [ 1 + r - r   1 - r   1 - r  ]

> Q := [ 3, [[1,2], [2,3,2]]];

                    Q := [3, [[1, 2], [2, 3, 2]]]

> x := array( [0,2,1] );
                           x := [ 0, 2, 1 ]

> one_pt_coxmatrix(Q,x);

                          [  0   1   0   2 ]
                          [                ]
                          [  3   3   2   0 ]
                          [                ]
                          [ -2  -2  -1  -1 ]
                          [                ]
                          [ -4  -4  -3   0 ]
```

SEE ALSO: `heralg/quiverdata`, `heralg[coxeter_matrix]`

4.14 heralg[one_pt_coxpoly] - the Coxeter polynomial of a one-point extension of a quiver algebra

Calling Sequence:

`one_pt_coxpoly`(Q,x)

Parameters:

Q – a quiver given by a quiverdata structure

x – a vector

Synopsis:

- The function `one_pt_coxpoly`(Q, x) computes the Coxeter polynomial (i.e. the characteristic polynomial of the Coxeter matrix) of the one-point extension of the path algebra of the quiver Q by a module with dimension vector x.
- This function is part of the heralg package, and so can be used in the form `one_pt_coxpoly(..)` only after performing the command `with(heralg)` or `with(heralg, one_pt_coxpoly)`. The function can always be accessed in the long form `heralg[one_pt_coxpoly](..)`.

Examples:

```
> with(heralg):
> one_pt_coxpoly( [2,[[1,2,r]]], array([1,1]) );

                   3             2   2             2
                  T  + (1 + r - r ) T  + (1 + r - r ) T + 1

> Q := [ 3, [[1,2], [2,3,2]]];

                         Q := [3, [[1, 2], [2, 3, 2]]]

> x := array( [0,2,1] );

                                x := [ 0, 2, 1 ]

> one_pt_coxpoly(Q,x);

                           4      3      2
                          T  - 2 T  + 3 T  - 2 T + 1
```

SEE ALSO: heralg/quiverdata, heralg[coxeter_polynomial]

Historical remarks

The use of computers has a long history in the representation theory of finite-dimensional algebras. The reason is that frequently classifications of algebras with certain properties had to be made, leading to calculations which were not feasible by hand any more. Classical examples are the classification of the representation-finite selfinjective algebras of Dynkin class $\mathbb{E}_6$, $\mathbb{E}_7$, $\mathbb{E}_8$ by Bretscher-Läser-Riedtmann (see [BLR]) and the classification of the tame concealed algebras by Happel-Vossieck (see [HV]).

In [Bo] the sincere representation-directed algebras with a large number of simples were classified and an algorithm to construct the exceptional algebras with few simples was presented which was based on the gradings of the corresponding orbit graphs. For some questions arising in [Dr2] it turned out that I needed the explicit classification of these algebras. It was clear that a computer was required and I decided to implement a different algorithm given in [Ri] which inductively produces the algebras. The results of the computations can be found in [Dr3] and [Dr4].

In order to use a computer for algebraic calculations I had to adjoin proper data structures acceptable for the machine to the relevant algebraic structures. This was achieved by considering bound poset algebras i.e. factor algebras $k[S]/I$ of the incidence algebras $k[S]$ of finite partially ordered sets (briefly posets) S by admissible ideals I generated by minimal sets of 0-relations $R = \{\rho_1, \ldots, \rho_r\}$. Note, that the ρ_i can be considered as pairs (s, t) of elements of S such that $s < t$ and s, t are not neighbours in the Hasse diagram $H(S)$ of S. On one hand any sincere representation-directed basic algebra A over an algebraically closed field k can be written as $k[S]/I$ and therefore the pair $(H(S), R)$ is a discrete data structure for A. On the other hand the isomorphism class of $k[S]/I$ is described uniquely up to permuting the elements of S by $(H(S), R)$.

In the sequel it turned out that the routines developed for the sincere representation-directed algebras were useful as well for considering other questions and calculating interesting examples. In 1991 I got the opportunity to work at the Sonderforschungsbereich 343 *Diskrete Strukturen in der Mathematik* in the project C2 on arranging and completing the meanwhile existing bunch of programs to a consistent program package on representation theory of finite-dimensional algebras. As title for the package we chose CREP which is an abbreviation of Combinatorial REPresentation theory. It is pointed out by this title that the basic intention is to create an efficient tool for working with combinatorial invariants of particular algebras and specific representations. As key words for the phenomena we had in mind one should think of quivers, partially ordered sets, bigraphs, dimension vectors, postprojective and preinjective components, tubes, ...

Before starting we had to ask ourselves which class of algebras will form an appropriate framework. The answer came from the study of criteria for representation type via thin modules started in [Dr1]. In this paper algebras associated with simply connected, chain-finite ray-categories where considered. They are a satisfactory framework for studying finite representation type but too restrictive for tame and wild representation type. It turned out that the proofs and ideas developed in [Dr1] worked well for the class of completely separating algebras. These are by definition the basic schurian algebras with directed quivers such that each convex subalgebra satisfies the separation condition of Bautista-Larrión (see [BL]). Completely separating algebras were introduced and studied in [Dr5]. They form a generalization of the algebras associated with simply connected, chain-finite ray-categories which in turn contain the sincere representation-directed algebras as subclass. Of importance is the fact that every completely separating algebra can be written as $A = k[S]/I$ as above. Hence the data $(H(S), R)$ still describe well the algebras and are well-suited for the computer.

It is interesting that the good algebras introduced in [Pe] in the study of the connection of integer quadratic forms and finite-dimensional algebras turned out to be completely separating as well. Obviously, integer quadratic forms resp. their associated bigraphs are combinatorial objects offering themselves for implementation. Meanwhile besides the algorithms using thin modules also the algorithms using quadratic forms build a major part of CREP.

The strongly simply connected algebras introduced in [Sk] are more general than the completely separating algebras because the 'schurian' assumption is dropped. Nevertheless, the completely separating algebras form a rather representative subclass which can be seen from the fact that most of the critical strongly simply connected and hypercritical strongly simply connected algebras happen to be schurian (see [HV] and [Un]) . Thus CREP also fits well into the presently developing theory of tame strongly simply connected algebras.

Rather important and interesting components of the Auslander-Reiten quiver of a representation finite or tame algebra are postprojective components and tubes (see [Ri]). One will find in CREP several programs dealing with these components and the dimension vectors of the indecomposable modules appearing as their vertices. Another feature which has to be mentioned are the data bases containing e.g. the lists of the critical, hypercritical, sincere representation-directed algebras to which one has access via the package.

Having discussed the scope of CREP inside the area of representation theory of finite-dimensional algebras, it remains to say something about its practical usability as computer system. The first version which I wrote in 1991 was just a collection of stand-alone Pascal programs with some standardized input and output, but input and output were still cumbersome. A surface giving easy access to the routines was missing. This was still the case in the fall of 1994 when R. Nörenberg started to work at the SFB 343. In discussions with him we decided that he should rewrite CREP to behave like a package of routines for a general purpose computer algebra system such that the user only had to learn the syntax of that system and no specific syntax for CREP. Several systems offering nice surfaces and features were available. Motivated by the collaboration with the research group of H. Lenzing in which representation theoretic programs using Maple as programming language had been written we did chose this system for the present version of CREP. Nevertheless, we refrained from rewriting most programs as Maple programs. In fact, frequently the original Pascal programs are still hidden behind the Maple routines because it turned out that in this way the programs ran much faster.

There lies some time and some work between a plan and its realization. I heartly thank R. Nörenberg for working hard to rewrite CREP, for adding many new routines, for splitting old programs into routines giving CREP a reasonable modular structure and, in particular, for writing this manual for version 1.0 which is a real improvement compared with the scarce instructions I produced for the 1991 version.

Already in the first version of CREP there were programs written by F. Lukas dealing with combinatorial structures associated with hereditary algebras like Cartan matrices and Coxeter transformations. The data structures for the input are similar but no generators of an ideal I are needed and multiple arrows between vertices are allowed. I am grateful to S. Lache who based on the work of Lukas wrote a package of Maple routines for CREP dealing with hereditary algebras. The documentation of these routines is also contained in this manual.

There are new routines in CREP for drawing bigraphs, a feature which was desperately needed for working in an area where a lot of information is encoded in quivers or graphs. My thanks go to A. Zylke and S. Ovsienko for contributing these routines to CREP and for help and advice during the development of the system.

Let me finish with a look into the future. In a workshop about computer applications in the representation theory of finite-dimensional algebras organized at Bielefeld in May 1994 it turned out that there are many programs around to solve specific questions. Unfortunately, there were no standards for input and output at that time. This made it hard for a non-experienced user to understand what he had to do. We hope that the present version of CREP provides some of these missing standards together with a reasonable framework in which other programs can be put. In fact, there are several projects to incorporate isolated programs as routines into the package.

Bielefeld, January 1996
Peter Dräxler

References

[BL] R. Bautista, F. Larrión, *Auslander-Reiten quivers for certain algebras of finite representation type,* J. London Math. Soc. 26 (1982), 43-52.

[Bo] K. Bongartz, *Treue einfach zusammenhängende Algebren I,* Comment. Math. Helv. 57 (1982), 331-378.

[BLR] O. Bretscher, C. Läser, C. Riedtmann, *Selfinjective and simply connected algebras,* Manuscr. Math. 36 (1981/82), 253-307.

[Dr1] P. Dräxler: *Über einen Zusammenhang zwischen darstellungsendlichen Algebren und geordneten Mengen,* Manuscr. Math. 60 (1988), 349-377.

[Dr2] P. Dräxler: *Fasersummen über dünnen s-Startmoduln,* Arch. Math. 54 (1990), 252-257.

[Dr3] P. Dräxler: *Aufrichtige gerichtete Ausnahmealgebren,* Bayreuther Math. Schr. 29 (1989), 2-191.

[Dr4] P. Dräxler: *Sur les algèbres exceptionelles de Bongartz,* C. R. Acad. Sci. Paris 311 (1990), 495-498.

[Dr5] P. Dräxler, *Completely separating algebras,* J. Algebra 165 (1994), 550-565.

[HV] D. Happel, D. Vossieck, *Minimal algebras of infinite representation type with preprojective component,* Manuscr. Math. 42 (1983), 221-243.

[Pe] J.A. de la Peña, *Algebras with hypercritical Tits form,* in: Topics in Algebra, Banach Center Publ. 26, part I (1990), 353-369.

[Ri] C.M. Ringel, *Tame algebras and integral quadratic forms,* Lecture Notes in Math. 1099 (1984).

[Sk] A. Skowroński, *Simply connected algebras and Hochschild cohomologies,* In Representations of Algebras, CMS Conf. Proc. 14 (1993), 431-447.

[Un] L. Unger, *The concealed algebras of the minimal wild hereditary algebras,* Bayreuther Math. Schr. 31 (1990), 145-154.