code

Particiones M-arias de un número

Dada una partción de un número entero N un conjunto de números enteros positivos que suman N, escritos en orden descendente. Por ejemplo,

```
10 = 4+3+2+1
```

Una partición es M-aria si cada término de dicha partición es una potencia de M. Por ejemplo,

```
Particiones 3-arias de 9:
9
3+3+3
3+3+1+1+1
3+1+1+1+1+1+1
1+1+1+1+1+1+1+1+1
```

Arañas de expansión de un grafo

Una araña de expansión de un grafo se define como un árbol de expansión que además es una araña.

Un árbol de expansión es un sub-grafo que es un árbol y que contiene todos los vértices del grafo.

Una araña es un grafo con como máximo un vértice cuyo grado —número de aristas incidentes a él— es 3 o más.

Por tanto, una araña de expansión de un grafo es un sub-grafo que es un árbol, contiene todos los vértices del grafo, y como máximo un sólo vértice de grado 3 o más.

Un ejemplo de grafo que contiene una araña de expansión sería

```
arista(d,a),
arista(d,b),
arista(d,c),
arista(d,e),
arista(a,x),
arista(b,y),
arista(c,z),
arista(a,b),
arista(y,c)
```

Usage and interface

Library usage:

```
:- use_module(/home/mlg/code.pl).
```

Exports:

- *Predicates:*

  `pots/3` , `pots2/4` , `mpart/3` , `mpart2/5` , `asignar/2` , `maria/3` , `arista/2` , `guardar_grafo/1` , `guardar_grafo2/1` .

- *Multifiles:*

  `call_in_module/2` .

Documentation on exports

PREDICATE pots/3

Usage: `pots(M,N,Ps)`

`Ps` es una lista con las potencias de `M` que son menores o iguales que `N` , en orden descendente.

```
pots(1,_1,[P]) :-
    P is 1.
pots(M,N,Ps) :-
    M>1,
    pots2(M,N,P,0),
    reverse(P,Ps).
```

Other properties:

Test: `pots(M,N,Ps)`

- *If the following properties hold at call time:*

  `M=3` (= /2)

  `N=9` (= /2)

  then the following properties should hold upon exit:

  `Ps=[9,3,1]` (= /2)

  then the following properties should hold globally:

  All the calls of the form `pots(M,N,Ps)` do not fail. (not_fails/1)

Test: `pots(M,N,Ps)`

- *If the following properties hold at call time:*

  `M=5` (= /2)

  `N=123` (= /2)

  then the following properties should hold upon exit:

  `Ps=[25,5,1]` (= /2)

  then the following properties should hold globally:

  All the calls of the form `pots(M,N,Ps)` do not fail. (not_fails/1)

PREDICATE **pots2/4**

Usage: `pots2(M,N,Ps,C)`

Auxiliar recursiva para pots con un contador añadido `C` .

```
pots2(M,N,[P|Ps],C) :-
    C1 is C+1,
    Q is M**C1,
    Q=<N,
    P1 is M**C,
    P is round(P1),
    pots2(M,N,Ps,C1).
pots2(M,N,[P],C) :-
    C1 is C+1,
    Q is M**C1,
    Q>N,
    P1 is M**C,
    P is round(P1).
```

PREDICATE **mpart/3**

Usage: `mpart(M,N,P)`

La lista `P` devuelve por backtracking todas las particiones `M` -arias de `N` , representadas como listas de enteros.

```
mpart(M,N,P) :-
    pots(M,N,A),
    mpart2(M,N,P,A,0).
```

Other properties:

Test: `mpart(M,N,P)`

- *If the following properties hold at call time:*
  `M=3` (`= /2`)
  `N=9` (`= /2`)
  then the following properties should hold upon exit:
  `P=[9];P=[3,3,3];P=[3,3,1,1,1];P=[3,1,1,1,1,1,1];P=[1,1,1,1,1,1,1,1,1]` (undefined property)
  then the following properties should hold globally:
  `try_sols(mpart(M,N,P),10)` (undefined property)
  All the calls of the form `mpart(M,N,P)` do not fail. (`not_fails/1`)

Test: `mpart(M,N,P)`

- *If the following properties hold at call time:*
  `M=5` (`= /2`)
  `N=26` (`= /2`)

then the following properties should hold upon exit:

`P=[25,1];P=[5,5,5,5,5,1];P=[5,5,5,5,1,1,1,1,1,1];P=[5,5,5,1,1,1,1,1,1,1,1,1,1];P=[5,5,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1];P=[5,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1];P=[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]` (undefined property)

then the following properties should hold globally:

`try_sols(mpart(M,N,P),10)` (undefined property)

All the calls of the form `mpart(M,N,P)` do not fail. (`not_fails/1`)

PREDICATE mpart2/5

Usage: `mpart2(M,N,P,A,C)`

Auxiliar recursiva para mpart.

```
mpart2(M,N,[P|Ps],[A|As],C) :-
    ( C1 is C+A,
      C1<N,
      P is A,
      mpart2(M,N,Ps,[A|As],C1)
    ; ( C1 is C+A,
        C1=:=N,
        asignar([P|Ps],A)
      ; mpart2(M,N,[P|Ps],As,C)
      )
    ).
```

PREDICATE asignar/2

Usage: `asignar(X,X)`

Auxiliar que iguala una lista [`A`] a `A` .

```
asignar([X],X).
```

PREDICATE maria/3

Usage: `maria(M,N,NPart)`

`NPart` es el número de particiones `M` -arias de `N` .

```
maria(M,N,NPart) :-
    findall(A,mpart(M,N,A),B),
    length(B,NPart).
```

Other properties:

Test: `maria(M,N,NPart)`

- *If the following properties hold at call time:*
  `M=3` (`= /2`)
  `N=9` (`= /2`)
  then the following properties should hold upon exit:
  `NPart=5` (`= /2`)
  then the following properties should hold globally:
  All the calls of the form `maria(M,N,NPart)` do not fail. (`not_fails/1`)

Test: `maria(M,N,NPart)`

- *If the following properties hold at call time:*
  `M=5` (`= /2`)
  `N=26` (`= /2`)
  then the following properties should hold upon exit:
  `NPart=7` (`= /2`)
  then the following properties should hold globally:
  All the calls of the form `maria(M,N,NPart)` do not fail. (`not_fails/1`)

PREDICATE arista/2

Usage: `arista(X,Y)`

Arista que conecta el vértice `X` con el vértice `Y`

The predicate is of type *dynamic*.
Other properties:

Test: `arista(X,Y)`

- *The following properties should hold upon exit:*
  `X=a,Y=e;X=b,Y=e;X=e,Y=f;X=f,Y=d;X=f,Y=c` (undefined property)
- *The following properties should hold globally:*
  `try_sols(arista(X,Y),10)` (undefined property)
  All the calls of the form `arista(X,Y)` do not fail. (`not_fails/1`)

Test: `arista(X,Y)`

- *If the following properties hold at call time:*
  `X=a` (`= /2`)
  `Y=b` (`= /2`)
  then the following properties should hold globally:
  Calls of the form `arista(X,Y)` fail. (`fails/1`)

Test: `arista(X,Y)`

- *The following properties should hold upon exit:*
  `X=a,Y=b;X=b,Y=c;X=c,Y=d;X=d,Y=e;X=e,Y=f` (undefined property)

- *The following properties should hold globally:*
  `try_sols(arista(X,Y),10)` (undefined property)
  All the calls of the form `arista(X,Y)` do not fail. (`not_fails/1`)

Test: `arista(X,Y)`

- *If the following properties hold at call time:*
  `X=f` (`= /2`)
  `Y=a` (`= /2`)
  then the following properties should hold globally:
  Calls of the form `arista(X,Y)` fail. (`fails/1`)

PREDICATE guardar_grafo/1

Usage: `guardar_grafo(G)`

Aserta en la base de datos como hechos del predicado arista/2 los elementos de `G` .

```
guardar_grafo(G) :-
    retractall(arista(_1,_2)),
    guardar_grafo2(G).
```

Other properties:

Test: `guardar_grafo(G)`

- *If the following properties hold at call time:*
  `G=[arista(a,e),arista(b,e),arista(e,f),arista(f,d),arista(f,c)]` (`= /2`)
  then the following properties should hold globally:
  All the calls of the form `guardar_grafo(G)` do not fail. (`not_fails/1`)

Test: `guardar_grafo(G)`

- *If the following properties hold at call time:*
  `G=[arista(a,b),arista(b,c),arista(c,d),arista(d,e),arista(e,f)]` (`= /2`)
  then the following properties should hold globally:
  All the calls of the form `guardar_grafo(G)` do not fail. (`not_fails/1`)

PREDICATE guardar_grafo2/1

Usage: `guardar_grafo2(G)`

Auxiliar recursiva para guardar_grafo.

```
guardar_grafo2([G|Gs]) :-
    assert(G),
    guardar_grafo2(Gs).
guardar_grafo2([]).
```

Documentation on multifiles

PREDICATE **call_in_module/2**

No further documentation available for this predicate. The predicate is *multifile*.

Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  `operators` , `dcg_phrase_rt` , `datafacts_rt` , `dynamic_rt` , `classic_predicates` , `native_props` .
- *Internal (engine) modules:*
  `term_basic` , `arithmetic` , `atomic_basic` , `basiccontrol` , `exceptions` , `term_compare` , `term_typing` , `debugger_support` , `hiord_rt` , `stream_basic` , `io_basic` , `runtime_control` , `basic_props` .
- *Packages:*
  `prelude` , `initial` , `condcomp` , `classic` , `runtime_ops` , `dcg` , `dcg/dcg_phrase` , `dynamic` , `datafacts` , `assertions` , `assertions/assertions_basic` , `regtypes` , `nativeprops` .