UTF-8

Practica 2 Programacion logica y declarativa

Juan Jimenez Perez 190204

Table of Contents

code 1
Pruebas de ejecucion 1
Predicado 1. Particiones M-arias de un numero. 1
Predicado 2. Aranyas de expansion de un grafo. 1
Usage and interface 2
Documentation on exports 2
pots/3 (pred) 2
pots_aux/4 (pred) 2
mpart/3 (pred) 3
mpart_aux/4 (pred) 3
maria/3 (pred) 3
arista/2 (pred) 3
guardar_grafo/1 (pred) 3
guardar_grafo_aux/1 (pred) 4
obtener_nodos/2 (pred) 4
aranya/0 (pred) 4
aranya_aux/4 (pred) 4
vertices_aux/3 (pred) 5
Documentation on multifiles 5
^^Fcall_in_module/2 (pred) 5
Documentation on imports 5

References 7

code

Pruebas de ejecucion

Predicado 1. Particiones M-arias de un numero.

1. Predicado 1.1: pots/3:

```
?- pots(3,9,Ps).

Ps = [9,3,1] ? ;
no
?-
```

2. Predicado 1.2: mpart/3:

```
?- mpart(3,9,P).

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] ? ;
no
?-
```

3. Predicado 1.3: maria/3:

```
?- maria(3,9,M).

M = 5.
?-
```

Predicado 2. Aranyas de expansion de un grafo.

4. Predicado 2.1: guardar_grafo/1:

```
?- guardar_grafo(
[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)]).

yes
?-
```

5. Predicado 2.2: aranya/0:

```
?- aranya.

yes
?-
```

Usage and interface

- **Library usage:**
  `:- use_module(/home/juan/Documents/Prolog/Practica2/code.pl).`
- **Exports:**
  - *Predicates:*
    `pots/3`, `pots_aux/4`, `mpart/3`, `mpart_aux/4`, `maria/3`, `arista/2`, `guardar_grafo/1`, `guardar_grafo_aux/1`, `obtener_nodos/2`, `aranya/0`, `aranya_aux/4`, `vertices_aux/3`.
  - *Multifiles:*
    `Σcall_in_module/2`.

Documentation on exports

pots/3: PREDICATE

Devuelve la lista de potencias de M menores o iguales que N:

```
pots(1,_1,[1]) :- !.
pots(0,_1,[1]) :- !.
pots(M,N,Ps) :-
    integer(M),
    integer(N),
    pots_aux(M,N,1,L),
    reverse(L,Ps).
```

Llama al predicado pots_aux/3 para ir realizando las diferentes potencias hasta que el valor donde se guardan las potencias es mayor que N, en cuyo caso para y vuelve a pots donde se invierte la lista.

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

`Ps` lista de potencias de `M` menores o iguales que `N`.

pots_aux/4: PREDICATE

```
pots_aux(_1,N,I,[]) :-
    I>N,
    !.
pots_aux(M,N,I,[I|L]) :-
    I=<N,
    X is M*I,
    pots_aux(M,N,X,L).
```

.

mpart/3: PREDICATE

Devuelve la lista de todas las particiones M-arias de N:

```
mpart(M,N,P) :-
    integer(M),
    integer(N),
    pots(M,N,L),
    mpart_aux(N,L,0,P).
```

Primero llama a pots/3 y con el resultado llama a mpart_aux/3, donde se van realizando sumas con el primer elemento de la lista hasta que el valor es igual a N, en caso de que el valor de la suma sea mayor que N se vuelve a llamar al predicado mpart_aux/3 eliminando el primer elemento de la lista.

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

`P` lista de todas las particiones `M`-arias de `N`.

mpart_aux/4: PREDICATE

```
mpart_aux(N,[X|_1],A,[X]) :-
    A1 is A+X,
    A1==N.
mpart_aux(N,[X|L],A,[X|L1]) :-
    A1 is A+X,
    A1<N,
    mpart_aux(N,[X|L],A1,L1).
mpart_aux(N,[_1|L],A,L1) :-
    mpart_aux(N,L,A,L1).
```

.

maria/3: PREDICATE

Devuelve el numero de todas las particiones M-arias de N:

```
maria(M,N,P) :-
    integer(M),
    integer(N),
    setof(L,mpart(M,N,L),L1),
    length(L1,P).
```

Se llama a setof/3 para obtener la lista de todas las soluciones de mpart y por ultimo se llama a length/2 para calcular la longitud de dicha lista.

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

`P` numero de todas las particiones `M`-arias de `N`.

arista/2: PREDICATE

. The predicate is of type *dynamic*.

guardar_grafo/1: PREDICATE

Grafo representado por aristas a dejar asertados en la base de datos como hechos del predicado arista/2:

```
guardar_grafo(L) :-
    retractall(arista(_1,_2)),
    guardar_grafo_aux(L).
```

Primero se vacia la base de datos llamando a retractall/2 y despues se llama a guardar_grafo_aux/1 para recorrer la lista de aristas y hacer los assert/1 de dichas aristas.

Usage: `guardar_grafo(G)`

`G` grafo representado por aristas a dejar asertados en la base de datos como hechos del predicado arista/2.

guardar_grafo_aux/1: PREDICATE

```
guardar_grafo_aux([]).
guardar_grafo_aux([X|L]) :-
    assert(X),
    guardar_grafo_aux(L).
```

.

obtener_nodos/2: PREDICATE

```
obtener_nodos([],[]).
obtener_nodos([arista(A,B)|L],[A,B|L1]) :-
    obtener_nodos(L,L1).
```

.

aranya/0: PREDICATE

Comprueba si el grafo de la base de datos contiene una aranya de expansion

```
aranya :-
    setof(arista(A,B),arista(A,B),L),
    obtener_nodos(L,L1),
    setof(X,member(X,L1),L2),
    aranya_aux(L2,L2,L,[]).
```

Primero se saca la lista de aristas y de nodos del grafo y se llama a la funcion aranya_aux/4 con la lista de nodos, la lista de aristas por visitar y la lista de aristas visitadas. Una vez en el predicado auxiliar se selecciona una arista de la lista de aristas por visitar y se comprueba si los nodos de dicha arista pertenecen a la lista de nodos, en caso de que pertenezcan a la lista se incluye la arista a la lista de aristas visitadas y se borran ambos nodos de la lista de nodos. Una vez la lista de nodos es vacia, se llama a vertices_aux/3 con la lista de aristas visitadas donde se cuenta el grado de cada vertice y se comprueba que hay como maximo un vertice con grado superior o igual a tres.

aranya_aux/4: PREDICATE

```
aranya_aux(N,N1,L,L1) :-
    select(arista(X,Y),L,LR),
    ( member(X,N1)
    ; member(Y,N1)
    ),
```

```
            append(L1,[X,Y],L2),
            delete(N1,X,N2),
            delete(N2,Y,NR),
            !,
            aranya_aux(N,NR,LR,L2).
        aranya_aux(N,[],_1,L) :-
            vertices_aux(N,L,0).
```

.

vertices_aux/3: PREDICATE

```
        vertices_aux([],_1,_2).
        vertices_aux([X|Y],L,B) :-
            bagof(A,(member(A,L),A==X),L1),
            length(L1,P),
            ( P>=3 ->
                B==0,
                B1 is 1
            ; B1 is 0
            ),
            vertices_aux(Y,L,B1).
```

.

Documentation on multifiles

Σcall_in_module/2: PREDICATE

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`, `lists`.

– *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`.

References

(this section is empty)