Sumas de pares de listas y cuadrados

Guillermo Lopez Garcia, a180182

Table of Contents

codigo

Este modulo define dos programas, sumlists/4 y square_lists/3.

Para el primero, dado un numero N par, se devuelven dos listas L1 y L2 que contienen entre las dos los nmeros de Peano de 1 a N y cuya suma es la misma, S.

Ejemplos de uso:

1. Ej:

```
?- sumlists(s(s(s(s(0)))),L1,L2,S).

L1 = [s(s(s(0))),s(s(0))],
L2 = [s(s(s(s(0)))),s(0)],
S = s(s(s(s(s(0))))) ?
yes
?-
```

2. Ej:

```
?- sumlists(s(s(s(s(s(s(s(s(0)))))))),L1,L2,S).

L1 = [s(s(s(s(s(s(s(0))))))),s(s(s(s(s(0))))),s(s(s(s(0)))),s(s(0))],
L2 = [s(s(s(s(s(s(0)))))),s(s(s(s(s(s(s(s(0))))))),s(s(s(0))),s(0)],
S = s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(0)))))))))))))))) ?
yes
?-
```

Por otra parte, para square_lists/4, dado un numero N, se devuelve una matriz cuadrada de N*N que contiene todos los numeros de Peano del 1 a N y cuyas filas suman lo mismo.

Ejemplos de uso:

1. Ej:

```
?- square_lists(s(s(0)),SQ,S).

S = s(s(s(s(0)))),
SQ = [[s(s(s(0))),s(s(0))],[s(s(s(s(0)))),s(0)]] ?

yes
?-
```

2. Ej:

```
?- square_lists(s(s(s(0))),SQ,S).

S = s(s(s(s(s(s(s(s(s(s(s(s(0)))))))))))),
SQ = [[s(s(s(s(s(s(s(0)))))))),s(s(s(s(0)))),s(s(s(0)))],
     [s(s(s(s(s(s(s(0)))))))),s(s(s(s(s(s(0)))))),s(s(0))],
     [s(s(s(s(s(s(s(s(0))))))))),s(s(s(s(s(0))))),s(0)]] ?

yes
?-
```

Mas adelante se muestran los predicados y propiedades que se han utilizado.

Usage and interface

- **Library usage:**
  `:- use_module(/home/guilogar/UPM/6semestre/ProDeclarativa/sumlistPeano/codigo.pl).`
- **Exports:**
  - *Predicates:*
    `alumno_prode/4`, `plus/3`, `nums/2`, `sumlist/2`, `choose_one/3`, `perm/2`, `split/3`, `sumlists/4`, `make_matrix/3`, `take_N/4`, `check_sum/2`, `times/3`, `exp/3`, `greater_zero/1`, `square_lists/3`.
  - *Properties:*
    `nat/1`, `lista/1`.

Documentation on exports

alumno_prode/4: PREDICATE

No further documentation available for this predicate.

nat/1: PROPERTY

Usage: `nat(N)`

Cierto si `N` es un numero natural.

```
nat(0).
nat(s(X)) :-
    nat(X).
```

lista/1: PROPERTY

Usage: `lista(L)`

Cierto si `L` es una lista.

```
lista([]).
lista([_1|Y]) :-
    lista(Y).
```

plus/3: PREDICATE

Usage: `plus(A,B,C)`

Cierto si `A + B = C`.

```
plus(X,0,X) :-
    nat(X).
plus(X,s(Y),s(Z)) :-
    plus(X,Y,Z).
```

Other properties:

Test: `plus(A,B,C)`

2 + 1 = 3

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

`A=s(s(0))` (`= /2`)

`B=s(0)` (`= /2`)

then the following properties should hold upon exit:

`C=s(s(s(0)))` (`= /2`)

then the following properties should hold globally:

All the calls of the form `plus(A,B,C)` do not fail. (`not_fails/1`)

nums/2: PREDICATE

Usage: `nums(N,L)`

Cierto si `L` es una lista descendente de `N` a 1.

```
nums(0,[]).
nums(s(N),[s(N)|Np]) :-
    nums(N,Np).
```

sumlist/2: PREDICATE

Usage: `sumlist(L,N)`

Cierto si `N` es la suma de elementos de `L`.

```
sumlist([],0).
sumlist([N|Np],S) :-
    sumlist(Np,Sp),
    plus(N,Sp,S).
```

choose_one/3: PREDICATE

Usage: `choose_one(E,L,R)`

Cierto si `R` es igual a `L` sin el elemento `E`.

```
choose_one(E,[E|Lp],Lp) :-
    lista(Lp).
choose_one(E,[X|Lp],[X|Rp]) :-
    choose_one(E,Lp,Rp).
```

perm/2: PREDICATE

Usage: `perm(L,Lp)`

Cierto si `Lp` es una permutacion de `L`.

```
perm([],[]).
perm([X|R],L) :-
    perm(R,Lp),
    choose_one(X,L,Lp).
```

split/3: PREDICATE

Usage: `split(L,Lp,Li)`

Cierto si `Lp` tiene los elementos de posicion par de `L`, y `Li`, los de posicion impar.

```
split([],[],[]).
split([X1,X2|Xn],[X1|Xp],[X2|Xpp]) :-
    split(Xn,Xp,Xpp).
```

sumlists/4: PREDICATE

Usage: `sumlists(N,L1,L2,S)`

Cierto si `L1` y `L2` contienen entre las dos los naturales de `N` hasta 1, y ambas suman lo mismo.

```
sumlists(N,L1,L2,S) :-
    nums(N,L),
    perm(L,Lp),
    split(Lp,L1,L2),
    sumlist(L1,S),
    sumlist(L2,S).
```

make_matrix/3: PREDICATE

Usage: `make_matrix(L,N,M)`

Cierto si `M` es una matriz de `N*N` formada por los elementos de `L`.

```
make_matrix([],_1,[]).
make_matrix(Lista,N,[Fila|Filas]) :-
    take_N(Lista,N,Fila,Rest),
    make_matrix(Rest,N,Filas).
```

take_N/4: PREDICATE

Usage: `take_N(L1,N,L2,Resto)`

Cierto si `L2` es una lista formada por los primeros `N` elementos de `L1`. `Resto` contiene el resto de elementos de `L1`.

```
take_N(Rest,0,[],Rest).
take_N([Elem|Lista],s(N),[Elem|Lista2],Rest) :-
    take_N(Lista,N,Lista2,Rest).
```

check_sum/2: PREDICATE

Usage: `check_sum(M,S)`

Cierto si la suma de todas las filas de `M` suman `S`.

```
check_sum([],_1).
check_sum([Fila|Filas],Sum) :-
    sumlist(Fila,Sum),
    check_sum(Filas,Sum).
```

times/3: PREDICATE

Usage: `times(A,B,C)`

Cierto si `A * B = C`

```
times(X,0,0) :-
    nat(X).
times(X,s(Y),Z) :-
    times(X,Y,W),
    plus(X,W,Z).
```

exp/3: PREDICATE

Usage: `exp(Exp,N,S)`

Cierto si `N^Exp = S`

```
exp(0,X,s(0)) :-
    nat(X).
exp(s(N),X,Y) :-
    exp(N,X,W),
    times(W,X,Y).
```

greater_zero/1: PREDICATE

Usage: `greater_zero(N)`

Cierto si `N` es un natural mayor que 0.

```
greater_zero(s(0)).
greater_zero(s(N)) :-
    greater_zero(N).
```

square_lists/3: PREDICATE

Usage: `square_lists(N,SQ,S)`

Cierto si `SQ` es una matriz de `N*N`, cuyas filas suman `S`, y entre todas contienen los numeros de `N^2` hasta 1.

```
square_lists(N,SQ,S) :-
    greater_zero(N),
    exp(s(s(0)),N,N2),
    nums(N2,Lista),
    perm(Lista,ListaP),
    make_matrix(ListaP,N,SQ),
    check_sum(SQ,S).
```

Documentation on imports

This module has the following direct dependencies:

– *Internal (engine) modules:*

`term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`.

– *Packages:*

`prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`.

References

(this section is empty)