Sumas de pares de listas y cuadrados

Guillermo Lopez Garcia, a180182

Table of Contents

codigo 1
Ejemplos de sumlists/4: 1
Ejemplos de square_lists/3: 1
Usage and interface 2
Documentation on exports 2
alumno_prode/4 (pred) 2
nat/1 (prop) 2
lista/1 (prop) 2
plus/3 (pred) 2
nums/2 (pred) 3
sumlist/2 (pred) 4
choose_one/3 (pred) 5
perm/2 (pred) 6
split/3 (pred) 6
sumlists/4 (pred) 7
make_matrix/3 (pred) 7
take_N/4 (pred) 8
check_sum/2 (pred) 9
times/3 (pred) 9
exp/3 (pred) 10
greater_zero/1 (prop) 11
square_lists/3 (pred) 11
Documentation on imports 11

References 13

Library/Module Index 15

Predicate Index 17

Property Index 19

Regular Type Index 21

Declaration Index 23

Concept Index 25

Author Index 27

Global Index 29

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 numeros de Peano de 1 a N y cuya suma es la misma, S.

Ejemplos de sumlists/4:

1.

```
?- 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.

```
?- 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(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^2 y cuyas filas suman lo mismo.

Ejemplos de square_lists/3:

1.

```
?- 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.

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

S = s(s(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
?-
```

A continuacion, 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`, `square_lists/3`.
  - *Properties:*
    `nat/1`, `lista/1`, `greater_zero/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)`

Caso: 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`)

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

Caso: 0 + 1 = 1

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

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

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

then the following properties should hold upon exit:

`C=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`)

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

Ejemplo de una resta: 4 - 2 = 2

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

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

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

then the following properties should hold upon exit:

`B=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).
```

Other properties:

Test: `nums(N,L)`

Lista desde el 3 hasta a 1.

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

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

then the following properties should hold upon exit:

`L=[s(s(s(0))),s(s(0)),s(0)]` (`=/2`)

then the following properties should hold globally:

All the calls of the form `nums(N,L)` do not fail. (`not_fails/1`)

Test: `nums(N,L)`

Igual que el caso anterior pero nos dan la lista.

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

`L=[s(s(s(0))),s(s(0)),s(0)]` (`=/2`)

then the following properties should hold upon exit:

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

then the following properties should hold globally:

All the calls of the form `nums(N,L)` do not fail. (`not_fails/1`)

Test: `nums(N,L)`

Caso base.

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

`N=0` (`=/2`)

then the following properties should hold upon exit:

`L=[]` (`=/2`)

then the following properties should hold globally:

All the calls of the form `nums(N,L)` do not fail. (`not_fails/1`)

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).
```

Other properties:

Test: `sumlist(L,N)`

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

`L=[s(0),s(0),s(s(0))]` (`=/2`)

then the following properties should hold upon exit:

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

then the following properties should hold globally:

All the calls of the form `sumlist(L,N)` do not fail. (`not_fails/1`)

Test: `sumlist(L,N)`

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

`L=[0,0,0]` (`=/2`)

then the following properties should hold upon exit:

`N=0` (`=/2`)

then the following properties should hold globally:

All the calls of the form `sumlist(L,N)` do not fail. (`not_fails/1`)

Test: `sumlist(L,N)`

Caso base.

– *If the following properties hold at call time:*
`L=[]` (`= /2`)
then the following properties should hold upon exit:
`N=0` (`= /2`)
then the following properties should hold globally:
All the calls of the form `sumlist(L,N)` do not fail. (`not_fails/1`)

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).
```

Other properties:

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

Se quita un elemento de la lista.

– *If the following properties hold at call time:*
`E=s(0)` (`= /2`)
`L=[s(s(0)),s(s(s(0))),s(0)]` (`= /2`)
then the following properties should hold upon exit:
`R=[s(s(0)),s(s(s(0)))]` (`= /2`)
then the following properties should hold globally:
All the calls of the form `choose_one(E,L,R)` do not fail. (`not_fails/1`)

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

Se quita un elemento de la lista.

– *If the following properties hold at call time:*
`E=s(s(s(0)))` (`= /2`)
`L=[s(s(0)),s(s(s(0))),s(0)]` (`= /2`)
then the following properties should hold upon exit:
`R=[s(s(0)),s(0)]` (`= /2`)
then the following properties should hold globally:
All the calls of the form `choose_one(E,L,R)` do not fail. (`not_fails/1`)

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

Se quita un elemento de una lista de un elemento.

– *If the following properties hold at call time:*
`E=s(s(0))` (`= /2`)
`L=[s(s(0))]` (`= /2`)
then the following properties should hold upon exit:
`R=[]` (`= /2`)
then the following properties should hold globally:
All the calls of the form `choose_one(E,L,R)` do not fail. (`not_fails/1`)

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).
```

Other properties:

Test: `perm(L,Lp)`

Caso base.

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

`L=[]` (`=/2`)

then the following properties should hold upon exit:

`Lp=[]` (`=/2`)

then the following properties should hold globally:

All the calls of the form `perm(L,Lp)` do not fail. (`not_fails/1`)

Test: `perm(L,Lp)`

Permutacion de una lista de 3 elementos. Al usar solutions en el test, LPdoc no genera bien la documentacion para este caso. En el codigo se muestra el test original.

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

`L=[a,b,c]` (`=/2`)

then the following properties should hold globally:

For this test of `perm(L,Lp)` get at most `6` solutions (normally 2 solutions are generated, just enough to detect non-determinism). (`try_sols/2`)

Goal `perm(L,Lp)` produces the solutions listed in `[perm([a,b,c],[a,b,c]),perm([a,b,c],[b,a,c]),perm([a,b,c],[b,c,a]),perm([a,b,c],[` (`solutions/2`)

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).
```

Other properties:

Test: `split(L,L1,L2)`

Caso base.

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

`L=[]` (`=/2`)

then the following properties should hold upon exit:

`L1=[]` (`=/2`)

`L2=[]` (`=/2`)

then the following properties should hold globally:

All the calls of the form `split(L,L1,L2)` do not fail. (`not_fails/1`)

Test: `split(L,L1,L2)`

Caso normal dada una lista L.

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

`L=[a,b,c,d]` (`= /2`)

then the following properties should hold upon exit:

`L1=[a,c]` (`= /2`)

`L2=[b,d]` (`= /2`)

then the following properties should hold globally:

All the calls of the form `split(L,L1,L2)` do not fail. (`not_fails/1`)

Test: `split(L,L1,L2)`

Dadas las dos listas devuelve la original.

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

`L1=[a,c]` (`= /2`)

`L2=[b,d]` (`= /2`)

then the following properties should hold upon exit:

`L=[a,b,c,d]` (`= /2`)

then the following properties should hold globally:

All the calls of the form `split(L,L1,L2)` do not fail. (`not_fails/1`)

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. Los ejemplos de uso se encuentran al principio del documento.

```
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` elementos por fila, 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).
```

Other properties:

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

Caso base.

– *If the following properties hold at call time:*
`L=[]` (`= /2`)
`N=s(0)` (`= /2`)
then the following properties should hold upon exit:
`M=[]` (`= /2`)
then the following properties should hold globally:
All the calls of the form `make_matrix(L,N,M)` do not fail. (`not_fails/1`)

Test: `make_matrix(L,N,M)`
Caso con una lista de 4 elementos y filas de 2.
– *If the following properties hold at call time:*
`L=[a,b,c,d]` (`= /2`)
`N=s(s(0))` (`= /2`)
then the following properties should hold upon exit:
`M=[[a,b],[c,d]]` (`= /2`)
then the following properties should hold globally:
All the calls of the form `make_matrix(L,N,M)` do not fail. (`not_fails/1`)

Test: `make_matrix(L,N,M)`
Caso con una lista de 8 elementos y filas de 4.
– *If the following properties hold at call time:*
`L=[a,b,c,d,e,f,g,h]` (`= /2`)
`N=s(s(s(s(0))))` (`= /2`)
then the following properties should hold upon exit:
`M=[[a,b,c,d],[e,f,g,h]]` (`= /2`)
then the following properties should hold globally:
All the calls of the form `make_matrix(L,N,M)` do not fail. (`not_fails/1`)

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).
```

Other properties:
Test: `take_N(L1,N,L2,Resto)`
Se toman 2 elementos de una lista de 5.
– *If the following properties hold at call time:*
`L1=[a,b,c,d,e]` (`= /2`)
`N=s(s(0))` (`= /2`)
`L2=[a,b]` (`= /2`)
then the following properties should hold upon exit:
`Resto=[c,d,e]` (`= /2`)
then the following properties should hold globally:
All the calls of the form `take_N(L1,N,L2,Resto)` do not fail. (`not_fails/1`)

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

Se toman 0 elementos de una lista de 5.

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

`L1=[a,b,c,d,e]` (`=/2`)

`N=0` (`=/2`)

then the following properties should hold upon exit:

`L2=[]` (`=/2`)

`Resto=[a,b,c,d,e]` (`=/2`)

then the following properties should hold globally:

All the calls of the form `take_N(L1,N,L2,Resto)` do not fail. (`not_fails/1`)

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).
```

Other properties:

Test: `check_sum(M,S)`

Matriz de 2*2 cuyos elementos suman 5.

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

`M=[[s(s(0)),s(s(s(0)))],[s(0),s(s(s(s(0))))]]` (`=/2`)

then the following properties should hold upon exit:

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

then the following properties should hold globally:

All the calls of the form `check_sum(M,S)` do not fail. (`not_fails/1`)

Test: `check_sum(M,S)`

Matriz de 3*1 cuyos elementos suman 2.

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

`M=[[s(s(0))],[s(s(0))],[s(s(0))]]` (`=/2`)

then the following properties should hold upon exit:

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

then the following properties should hold globally:

All the calls of the form `check_sum(M,S)` do not fail. (`not_fails/1`)

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).
```

Other properties:

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

Caso base: 2 * 0 = 0

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

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

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

then the following properties should hold upon exit:

`S=0` (`=/2`)

then the following properties should hold globally:

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

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

Caso: 2 * 3 = 6

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

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

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

then the following properties should hold upon exit:

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

then the following properties should hold globally:

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

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).
```

Other properties:

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

Caso base: 2^0 = 1

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

`Exp=0` (`=/2`)

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

then the following properties should hold upon exit:

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

then the following properties should hold globally:

All the calls of the form `exp(Exp,N,S)` do not fail. (`not_fails/1`)

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

Caso: 2^3 = 8

– *If the following properties hold at call time:*
`Exp=s(s(s(0)))` (`= /2`)
`N=s(s(0))` (`= /2`)
then the following properties should hold upon exit:
`S=s(s(s(s(s(s(s(0))))))))` (`= /2`)
then the following properties should hold globally:
All the calls of the form `exp(Exp,N,S)` do not fail. (`not_fails/1`)

greater_zero/1: PROPERTY

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. Los tests se encuentran al principio del documento.

```
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:

– *Application modules:*
`native_props`, `unittest_props`.

– *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`, `nativeprops`.

References

(this section is empty)

Library/Module Index

(Index is nonexistent)

Predicate Index

A

alumno_prode/4 2

C

check_sum/2 9
choose_one/3 5

E

exp/3 10

M

make_matrix/3 7

N

nums/2 3

P

perm/2 6
plus/3 2

S

split/3 6
square_lists/3 11
sumlist/2 4
sumlists/4 7

T

take_N/4 8
times/3 9

Property Index

G

greater_zero/1 11

L

lista/1 2

N

nat/1 2

Regular Type Index

(Index is nonexistent)

Declaration Index

(Index is nonexistent)

Concept Index

(Index is nonexistent)

Author Index

(Index is nonexistent)

Global Index

This is a global index containing pointers to places where concepts, predicates, modes, properties, types, applications, authors, etc., are referred to in the text of the document.

=

= /2 ..... 3, 4, 5, 6, 7, 8, 9, 10, 11

A

A=0 3
A=s(s(0)) 3, 10
alumno_prode/4 2
arithmetic 11
assertions 11
assertions/assertions_basic 11
atomic_basic 11

B

B=0 10
B=s(0) 3
B=s(s(0)) 3
B=s(s(s(0))) 10
basic_props 11
basiccontrol 11

C

C=s(0) 3
C=s(s(s(0))) 3
C=s(s(s(s(0)))) 3
check_sum/2 ..... 2, 9
choose_one/3 ..... 2, 5
condcomp 11

D

debugger_support 11

E

E=s(0) 5
E=s(s(0)) 5
E=s(s(s(0))) 5
exceptions 11
exp/3 ..... 2, 10
Exp=0 10
Exp=s(s(s(0))) 11

G

greater_zero/1 ..... 2, 11

I

initial 11

L

L=[] 4, 5, 6, 8
L=[0,0,0] 4
L=[a,b,c,d,e,f,g,h] 8
L=[a,b,c,d] 7, 8
L=[a,b,c] 6
L=[s(0),s(0),s(s(0))] 4
L=[s(s(0)),s(s(s(0))),s(0)] 5
L=[s(s(0))] 5
L=[s(s(s(0))),s(s(0)),s(0)] 3, 4
L1=[] 6
L1=[a,b,c,d,e] 8, 9
L1=[a,c] 7
L2=[] 6, 9
L2=[a,b] 8
L2=[b,d] 7
lista/1 2
Lp=[] 6

M

M=[[a,b,c,d],[e,f,g,h]] 8
M=[[a,b],[c,d]] 8
M=[[s(s(0)),s(s(s(0)))],[s(0),s(s(s(s(0))))]]
..... 9
M=[[s(s(0))],[s(s(0))],[s(s(0))]] 9
M=[] 8
make_matrix/3 ..... 2, 7

N

N=0 ..... 4, 5, 9
N=s(0) 8
N=s(s(0)) 8, 10, 11
N=s(s(s(0))) 3, 4
N=s(s(s(s(0)))) 4, 8
nat/1 2
native_props 11
nativeprops 11

not_fails/1 ..... 3, 4, 5, 6, 7, 8, 9, 10, 11
nums/2 ..... 2, 3

P

perm/2 ..... 2, 6
plus/3 2
prelude 11

R

R=[] 5
R=[s(s(0)),s(0)] 5
R=[s(s(0)),s(s(s(0)))] 5
Resto=[a,b,c,d,e] 9
Resto=[c,d,e] 8

S

S=0 10
S=s(0) 10
S=s(s(0)) 9
S=s(s(s(s(s(0))))) 9
S=s(s(s(s(s(s(0)))))) 10
S=s(s(s(s(s(s(s(s(0))))))))) 11
solutions/2 6
split/3 ..... 2, 6
square_lists/3 ..... 2, 11
sumlist/2 ..... 2, 4
sumlists/4 ..... 2, 7

T

take_N/4 ..... 2, 8
term_basic 11
term_compare 11
term_typing 11
times/3 ..... 2, 9
try_sols/2 6

U

unittest_props 11