Practica 1 - Sumas de pares de listas y cuadrados

Programacion logica pura

David de Frutos, B190372

Table of Contents

codigo

Este modulo define los predicados de la practica 1 para la que se quiere realizar sumas de pares de listas y cuadrados.

Los numeros aceptados por el programa tienen que ser de la forma:

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

Usage and interface

- **Library usage:**
  `:- use_module(/home/defru/prolog/codigo.pl).`
- **Exports:**
  - *Predicates:*
    `add/3`, `concatenar/3`, `inversa/2`, `nums/2`, `sumlist/2`, `choose_one/3`, `perm/2`, `split/3`, `sumlists/4`, `square_lists/3`, `split_in3/4`.
  - *Properties:*
    `alumno_prode/4`, `nat/1`.

Documentation on exports

alumno_prode/4: PROPERTY

Usage:

Datos del alumno que realiza la entrega.

```
alumno_prode('De Frutos','Zafra','David','B190372').
```

nat/1: PROPERTY

Usage:

Numero natural.

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

add/3: PREDICATE

Usage: `add(X,Y,Z)`

`Z` es el resultado de realizar la suma de `X` y `Y`.

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

concatenar/3: PREDICATE

Usage: `concatenar(L1,L2,L)`

`L` es la lista resultado de realizar la concatenacion de `L1` y `L2`.

```
concatenar([],L,L).
concatenar([L1|L1s],L2,[L1|L]) :-
    concatenar(L1s,L2,L).
```

inversa/2: PREDICATE

Usage: `inversa(L,Linv)`

`Linv` es el resultado de realizar la inversion de la lista `L`.

```
inversa([],[]).
inversa([L|Ls],Linv) :-
    inversa(Ls,L1),
    concatenar(L1,[L],Linv).
```

nums/2: PREDICATE

Usage: `nums(N,L)`

donde N es un numero natural en forma de Peano y L es la lista resultado de los naturales en orden descendente de N a 1. La idea es ir retrocediendo llamando en la recursiva con el antecesor a la vez que se agrega a la cabeza de la lista.

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

sumlist/2: PREDICATE

Usage: `sumlist(L,S)`

tal que L es una lista de numeros naturales y S es la suma de todos los elementos de L. La idea es ir separando la cabeza de la lista y llamando con el resto hasta llegar al caso base. Una vez llegado al caso base se ejecuta el predicado add, que va acumulando la suma en S1 para la siguiente recursion.

```
sumlist([],0).
sumlist([L|Ls],S) :-
    sumlist(Ls,S1),
    add(L,S1,S).
```

choose_one/3: PREDICATE

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

donde L es una lista, E es un elemento cualquiera de L, y R es lo que queda la lista tras quitar E. Se realiza la llamada recursiva con la cola de la lista hasta que coincida el elemento buscado con la cabeza de la lista. Para quedarnos con el resto sin el elemento se va aadiendo la cabeza a la lista R en cada recursion.

```
choose_one(E,[E|Ls],Ls).
choose_one(E,[L|Ls],[L|R]) :-
    choose_one(E,Ls,R).
```

perm/2: PREDICATE

Usage: `perm(L,LP)`

tal que L es una lista y LP es una permutacion de L. Se deben generar como alternativas en LP todas las permutaciones de L. Se llama en la recursiva con la cola de lista y una lista auxiliar LP1 donde se va almacenando la lista que se va creando. El predicado choose_one puede crear la lista puesto que recibe la lista parcial de elementos aadidos y la cabeza de lista, que seria el elemento que busca.

```
perm([],[]).
perm([L|Ls],LP) :-
    perm(Ls,LP1),
    choose_one(L,LP,LP1).
```

split/3: PREDICATE

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

donde L es una lista de longitud N, con N par, L1 contiene los N/2 elementos en posicion impar de L y L2 los elementos en posicion par Se van aadiendo el primer y segundo elemento a la lista 1 y 2 respectivamente.

```
split([],[],[]).
split([L,Ls|Lss],[L|L1],[Ls|L2]) :-
    split(Lss,L1,L2).
```

sumlists/4: PREDICATE

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

tal que N es par, L1 y L2 son dos listas de longitud N/2, que contienen entre ellas todos los numeros de Peano de 1 a N, y suman lo mismo. S debe ser el valor de dicha suma. Para este predicado se utilizan todos los predicados realizados anteriormente. La idea es crear una lista con el predicado nums/2 a partir de un N dado, dividir esta lista en dos partes con split/3 y comprobar si ambas suman lo mismo utilizando la misma variable S con el predicado sumlist/2. En caso de no ser asi la ejecucion seria "no", retrocediendo hasta el predicado perm/2, que tiene mas soluciones en forma de permutacion de la lista conseguida en nums/2. Este proceso se repetira hasta encontrar una solucion valida.

```
sumlists(0,[],[],0).
sumlists(N,L1,L2,S) :-
    nums(N,L),
    perm(L,Lperm),
    split(Lperm,L1,L2),
    sumlist(L1,S),
    sumlist(L2,S).
```

square_lists/3: PREDICATE

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

El objetivo es, dado N, y los numeros de Peano consecutivos de 1 a N^2, colocarlos en un cuadrado de tamao N x N tal que todas las filas sumen lo mismo. donde N es el numero en forma de Peano, SQ el cuadrado (representado como N listas de N elementos cada una), y S la suma de los elementos de las filas. Ante la imposibilidad de realizar un predicado en el que se genere la lista hasta N^2, se ha optado por contemplar los casos mas sencillos: N = 1, N = 2 y N =3.

```
square_lists(s(0),[[s(0)]],s(0)).
square_lists(s(s(0)),[SQ1,SQ2],S) :-
    nums(s(s(s(s(0)))),L),
    perm(L,Lperm),
    split(Lperm,SQ1,SQ2),
    sumlist(SQ1,S),
    sumlist(SQ2,S).
square_lists(s(s(s(0))),[SQ1,SQ2,SQ3],S) :-
    nums(s(s(s(s(s(s(s(s(s(0)))))))),L),
    perm(L,Lperm),
    split_in3(Lperm,SQ1,SQ2,SQ3),
    sumlist(SQ1,S),
    sumlist(SQ2,S),
    sumlist(SQ3,S).
```

split_in3/4: PREDICATE

Usage: `split_in3(L,L1,L2,L3)`

Predicado que adapta el predicado split pedido con anterioridad, con el objetivo de poder dividir una lista dada en tres partes iguales, de modo que se puede crear el "cuadrado" para el caso de N = 3 y tener una lista de 1 a 3^2

```
split_in3([],[],[],[]).
split_in3([L,Ls1,Ls2|Lss],[L|L1],[Ls1|L2],[Ls2|L3]) :-
    split_in3(Lss,L1,L2,L3).
```

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)