Primera práctica - edificios

Programación Declarativa: Lógica y Restricciones

Jaime Bautista Salinero; 150103

Table of Contents

codigo

Este módulo se representa un edificio de viviendas.

Un edificio está representado por una lista, que a su vez contiene sublistas que representarán las distintas plantas del edificio. Los elementos de cada sublista serán las viviendas y su valor el número de habitantes para dicha vivienda.

El número de habitantes estará representado por números en notación de Peano, por lo que deberán ser de la siguiente forma:

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

Aclaración sobre redondeo

En cuanto al redondeo realizado en el predicado `average`, en el que se pedía el redondeo al número natural más cercano, este redondeo se ha implementado siguiendo las directrices del estándar IEEE 754-2008, `https://standards.ieee.org/standard/754-2008.html`, mediante el cual, en caso de empate,que el decimal sea .5, se redondeará al número natural par de entre las dos posibilidades existentes. De esta manera, en la mitad de las ocasiones se redondeará al elemento superior y en la otra mitad al elemento inferior.

Estructura de la documentación

Los predicados explicados en la sección 'Documentation on exports' están ordenados de la siguiente manera para facilitar su estructura:

1. Predicados pedidos en el ejercicio y a continuación sus predicados auxiliares.
2. Predicados empleados para la representación y tratamiento de números naturales en notación de Peano.
3. Predicados sobre listas.

Consultas de comprobación

A continuación, se muestras las peticiones realizadas al programa para validar los resultados y a su derecha el resultado esperado:

basic_building

```
basic_building([0,0]). -> no
basic_building([s(0),0]). -> no
basic_building([[0,0,0],[],[0,0,0]]). -> no
basic_building([[0],[0]]). -> yes
basic_building([[0,0],[0,0]]). -> yes
basic_building([[0,0,0],[0,0,0],[0,0,0]]). -> yes
basic_building([[s(0),0,0],[0,s(0),0],[0,0,s(0)]]). -> yes
basic_building([[s(0),s(s(0)),0],[0,s(s(s(0))),0],[s(0),s(0),s(0)]]). -> yes
basic_building([[s(0),s(s(0))],[0,s(s(s(0))),0],[s(0),s(0),s(0),0]]). -> yes
```

building

```
building([[0,0],[0,0,0],[0,0,0]]). -> no
building([[s(0),s(s(0))],[0,s(s(s(0))),0],[s(0),s(0),s(0),0]]). -> no
building([[0],[0]]). -> yes
building([[0,0],[0,0]]). -> yes
building([[0,0,0],[0,0,0],[0,0,0]]). -> yes
building([[s(0),s(s(0)),0],[0,s(s(s(0))),0],[s(0),s(0),s(0)]]). -> yes
building([[s(0),0,0],[0,s(0),0],[0,0,s(0)]]). -> yes
```

level

```
level([s(0),0,s(s(s(0)))],s(s(s(0))),C). -> no
level([[0,0],[0,0]],s(s(s(0))),C). -> no
level([[0],[0]],s(0),C). -> yes + C = [0]
level([[0,0],[0,0]],s(s(0)),C). -> yes + C = [0,0]
level([[0,0,0],[0,0,0],[0,0,0]],s(s(s(0))),C). -> yes + C = [0,0,0]
level([[s(0),s(s(0)),0],[0,s(s(s(0))),0],[s(0),s(0),s(0)]],s(0),C).
   -> yes + C = [s(0),s(s(0)),0]
level([[s(0),s(s(0)),0],[0,s(s(s(0))),0],[s(0),s(0),s(0)]],s(s(0)),C).
   -> yes + C = [0,s(s(s(0))),0]
level([[s(0),s(s(0)),0],[0,s(s(s(0))),0],[s(0),s(0),s(0)]],s(s(s(0))),C).
   -> yes + C = [s(0),s(0),s(0)]
```

column

```
column([s(0),0,s(s(s(0)))],s(s(s(0))),C). -> no
column([[0],[0]],s(0),C). -> yes + C = [0,0]
column([[0,0],[0,0]],s(s(s(0))),C). -> yes + C = []
column([[0,0,s(0)],[0,0]],s(s(s(0))),C). -> yes + C = [s(0)]
column([[0,0],[0,0]],s(s(0)),C). -> yes + C = [0,0]
column([[0,0,0],[0,0,0],[0,0,0]],s(s(s(0))),C). -> yes + C = [0,0,0]
column([[s(0),s(s(0)),0],[0,s(s(s(0))),0],[s(0),s(0),s(0)]],s(0),C).
   -> yes + C = [s(0),0,s(0)]
column([[s(0),s(s(0)),0],[0,s(s(s(0))),0],[s(0),s(0),s(0)]],s(s(0)),C).
   -> yes + C = [s(s(0)),s(s(s(0))),s(0)]
column([[s(0),s(s(0)),0],[0,s(s(s(0))),0],[s(0),s(0),s(0)]],s(s(s(0))),C).
   -> yes + C = [0,0,s(0)]
```

columns

```
columns([s(0),0,s(s(s(0)))],C). -> no
columns([[0],[0]],C). -> yes + C = [[0,0]]
columns([[0,0,s(0)],[0,0]],C). -> yes + C = [[0,0],[0,0],[s(0)]]
columns([[0,0],[0,0]],C). -> yes + C = [[0,0],[0,0]]
columns([[0,0,0],[0,0,0],[0,0,0]],C). -> yes + C = [[0,0,0],[0,0,0],[0,0,0]]
columns([[s(0),s(s(0)),0],[0,s(s(s(0))),0],[s(0),s(0),s(0)]],C).
   -> yes + C = [[s(0),0,s(0)],[s(s(0)),s(s(s(0))),s(0)],[0,0,s(0)]]
columns([[s(0),0,s(s(s(0))),0],[0,s(s(0)),s(s(s(s(s(0))))),[s(0),s(0),s(0),0,0]],C).
   -> yes + C =
[[s(0),0,s(0)],[0,s(s(0)),s(0)],[s(s(s(0))),s(s(s(s(s(0))))),s(0)],[0,0],[0]]
```

total_people

```
total_people([s(0),0,s(s(s(0)))],C). -> no
total_people([[0],[0]],C). -> yes + C = 0
total_people([[0,0,s(0)],[0,0]],C). -> yes + C = 0
total_people([[0,0],[0,0]],C). -> yes + C = 0
total_people([[0,0,0],[0,0,0],[0,0,0]],C). -> yes + C = 0
total_people([[s(0),s(s(s(0)))],[0,s(s(0))]],C). -> yes + C = s(s(s(s(s(s(0))))))
total_people([[s(0),s(s(0)),0],[0,s(s(s(0))),0],[s(0),s(0),s(0)]],C).
   -> yes + C = s(s(s(s(s(s(s(s(s(0))))))))
total_
people([[s(0),s(s(s(0))),0],[0,s(s(0)),s(s(s(s(s(0)))))],[s(0),s(0),s(0)]],C).
   -> yes + C = s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(0))))))))))))))
total_
people([[s(0),0,s(s(s(0))),0],[0,s(s(0)),s(s(s(s(s(0)))))],[s(0),s(0),s(0),0,0]],C).
   -> yes + C = s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(0))))))))))))))
```

average

```
average([s(0),0,s(s(s(0)))],C). -> no
average([[0],[0]],C). -> yes + C = 0
average([[0,0,s(0)],[0,0]],C). -> yes + C = 0
average([[0,0],[0,0]],C). -> yes + C = 0
average([[0,0,0],[0,0,0],[0,0,0]],C). -> yes + C = 0
average([[s(0),s(0),s(0)],[0,0]],C). -> yes + C = s(0)
average([[s(0),s(s(s(0)))],[0,s(s(0))]],C). -> yes + C = s(s(0))
average([[s(0),s(s(0)),0],[0,s(s(s(0))),0],[s(0),s(0),s(0)]],C). -> yes + C =
s(0)
average([[s(0),s(s(s(0))),0],[0,s(s(0)),s(s(s(s(s(0)))))],[s(0),s(0),s(0)]],C).
-> yes + C = s(s(0))
average([[s(0),0,s(s(s(0))),0],[0,s(s(0)),s(s(s(s(s(0)))))],[s(0),s(0),s(0),0,0]],C).
-> yes + C = s(s(0))
```

Generación de la documentación

Esta documentación ha sido generada automaticamente con la herramienta **lpdoc** (`http://ciao-lang.org/ciao/build/doc/lpdoc.html/`). Para generarla, desde una línea de comandos ubicada en el directorio donde se encuentra el fichero de código, se ha ejecutado:

```
~$ lpdoc -t pdf codigo.pl
```

Usage and interface

- **Library usage:**
  `:- use_module(/home/jaime/edificios/codigo.pl).`
- **Exports:**
  - *Predicates:*
    `alumno_prode/4`, `level/3`, `column/3`, `columns/2`, `total_people/2`, `average/2`, `count_level_people/2`, `longest_level/2`, `longest_level/3`, `columns_aux/3`, `nat_add/3`, `nat_prod/3`, `nat_mod/3`, `nat_round/3`, `list_select/3`, `list_append/3`, `list_length/2`, `list_reverse/2`, `list_reverse/3`, `list_flatten/2`.
  - *Properties:*
    `basic_building/1`,
    `building/1`, `levels_have_home/1`, `all_levels_equal/1`, `levels_equal/2`, `total_people_homes/3`, `nat/1`, `nat_eq/2`, `nat_gt/2`, `nat_geq/2`, `nat_even/1`, `nat_odd/1`, `list/1`, `list_members_nat/1`, `sublist_members_nat/1`, `list_same_length/2`.

Documentation on exports

alumno_prode/4: PREDICATE

No further documentation available for this predicate.

basic_building/1: PROPERTY

Usage: `basic_building(X)`

`X` es un edificio, es decir una lista de listas, donde los elementos de las sublistas son números naturales en notación de Peano.

```
basic_building(X) :-
        sublist_members_nat(X),
        levels_have_home(X).
```

building/1: PROPERTY

Usage: `building(X)`

`X` es un edificio (basic_building) donde todos los niveles tienen el mismo nmero de viviendas.

```
building(X) :-
        basic_building(X),
        all_levels_equal(X).
```

level/3: PREDICATE

Usage: `level(X,N,C)`

`C` es la lista con todas las viviendas del `N`-ésimo nivel de un edificio `X`.

```
level([C|_1],s(0),C) :-
        list(C).
level([_1|Xs],s(N),C) :-
        level(Xs,N,C).
```

column/3: PREDICATE

Usage: `column(X,N,C)`

`C` es la lista formada por las viviendas `N`-ésimas de todos los niveles del edificio `X`. Si se pide una columna que el edificio `X` no tiene se devolverá una lista vacía.

```
column([],_1,[]).
column([X|Xs],N,[C|Cs]) :-
        list_length(X,L),
        nat_geq(L,N),
        list_select(X,N,C),
        column(Xs,N,Cs).
column([X|Xs],N,Cs) :-
        list_length(X,L),
        nat_gt(N,L),
        column(Xs,N,Cs).
```

columns/2: PREDICATE

Usage: `columns(X,C)`

`C` es la lista de las columnas de viviendas del edificio `X`.

```
columns(X,C) :-
        longest_level(X,Ls),
        list_length(Ls,L),
        columns_aux(X,L,Y),
        list_reverse(Y,C).
```

total_people/2: PREDICATE

Usage: `total_people(X,T)`

Calcula el número total de habitantes del edificio `X`, devolviendo el resultado en `T`. El procedimiento que sigue es aplanar la lista `X` y sumar todos los elementos de la lista aplanada.

```
total_people(X,T) :-
        sublist_members_nat(X),
        list_flatten(X,Y),
        count_level_people(Y,T).
```

average/2: PREDICATE

Usage: `average(X,A)`

Calcula la media de personas, en `A`, que viven en cada vivienda del edificio `X` y redondea el valor al número natural más cercano. `A` será el resultado de dividir habitantes / viviendas y el redondeo se realiza en base al estándar IEEE 754 de 2008.

```
average(X,s(0)) :-
        total_people_homes(X,H,V),
        nat_eq(V,H).
average(X,A) :-
        total_people_homes(X,H,V),
        nat_gt(V,H),
        nat_prod(H,s(s(0)),Z),
        nat_gt(Z,V),
        nat_round(s(0),0,A).
average(X,A) :-
        total_people_homes(X,H,V),
        nat_gt(V,H),
        nat_prod(H,s(s(0)),Z),
        nat_gt(V,Z),
        nat_round(0,0,A).
average(X,A) :-
        total_people_homes(X,H,V),
        nat_gt(H,V),
        nat_mod(H,V,R),
        nat_add(Hh,R,H),
        nat_prod(P,V,Hh),
        nat_round(P,R,A).
```

levels_have_home/1: PROPERTY

Usage: `levels_have_home(X)`

Recorre un edificio `X` comprobando que cada nivel tiene al menos una vivienda. Recorre la lista `X` de listas comprobando que todas las sublistas tienes al menos un elemento mayor estricto (`>`) que 0.

```
levels_have_home([_1]).
levels_have_home([_1|Xs]) :-
        levels_have_home(Xs).
```

count_level_people/2: PREDICATE

Usage: `count_level_people(X,Y)`

Cuenta todos los habitantes de un nivel `X` de un edificio, devolviendo el resultado en `Y`. Recorre una lista `X` sumando todos sus elementos en `Y`.

```
count_level_people([],0).
count_level_people([0|Xs],Y) :-
        count_level_people(Xs,Y).
count_level_people([s(X)|Xs],s(Y)) :-
        count_level_people([X|Xs],Y).
```

all_levels_equal/1: PROPERTY

Usage: `all_levels_equal(X)`

Comprueba que todos los niveles de un edificio `X` tienen el mismo número de viviendas. Recorre las sublistas de una lista `X` comprobando que la longitud de una sublista es igual que el resto de sublistas.

```
all_levels_equal([X|Xs]) :-
        list_length(X,Y),
        levels_equal(Xs,Y).
```

levels_equal/2: PROPERTY

Usage: `levels_equal(X,Y)`

Comprueba que todos los niveles dados en el edificio `X` tienen el número de viviendas indicado por `Y`. Recorre la lista `X` de sublistas comprobando que la lista de sus elementos es igual a `Y`.

```
levels_equal([],_1).
levels_equal([X|Xs],Y) :-
        levels_equal(Xs,Y),
        list_length(X,Z),
        nat_eq(Y,Z).
```

longest_level/2: PREDICATE

Usage: `longest_level(X,L)`

Devuelve en `Y` el nivel del edificio `X` de mayor longitud de viviendas.

```
longest_level([X|Xs],L) :-
        longest_level(Xs,X,L).
```

longest_level/3: PREDICATE

Usage: `longest_level(X,Y,L)`

Predicado auxiliar para el anterior predicado.

```
longest_level([],L,L).
longest_level([X|Xs],Y,L) :-
        list_length(X,Z1),
        list_length(Y,Z2),
        nat_geq(Z1,Z2),
        longest_level(Xs,X,L).
longest_level([X|Xs],Y,L) :-
        list_length(X,Z1),
        list_length(Y,Z2),
        nat_gt(Z2,Z1),
        longest_level(Xs,Y,L).
```

columns_aux/3: PREDICATE

Usage: `columns_aux(X,N,C)`

`C` es la lista, del revés, de las columnas del edificio `X`. Dado que se trata de un predicado auxiliar para colums, se le pasa en `N` la longitud (niveles) del edificio `X`.

```
columns_aux(_1,0,[]).
columns_aux(X,s(N),[C|Cs]) :-
        column(X,s(N),C),
        columns_aux(X,N,Cs).
```

total_people_homes/3: PROPERTY

Usage: `total_people_homes(X,H,V)`

Calcula el número total de habitantes, en `H`, y de viviendas, en `V`, de un edificio `X`.

```
total_people_homes(X,H,V) :-
        sublist_members_nat(X),
        list_flatten(X,Y),
        count_level_people(Y,H),
        list_length(Y,V).
```

nat/1: PROPERTY

Usage: `nat(X)`

`X` es un número natural en notación de Peano.

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

nat_eq/2: PROPERTY

Usage: `nat_eq(X,Y)`

`X` y `Y` son iguales la una a la otra.

```
nat_eq(0,0).
nat_eq(s(X),s(Y)) :-
        nat_eq(X,Y).
```

nat_gt/2: PROPERTY

Usage: `nat_gt(X,Y)`

`X` es mayor estricto (`>`) que `Y`.

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

nat_geq/2: PROPERTY

Usage: `nat_geq(X,Y)`

`X` es mayor o igual (`>=`) que `Y`.

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

nat_even/1: PROPERTY

Usage: `nat_even(X)`

`X` es par.

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

nat_odd/1: PROPERTY

Usage: `nat_odd(X)`

`X` es impar.

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

nat_add/3: PREDICATE

Usage: `nat_add(X,Y,S)`

`S` es el resultado de la suma `X + Y`.

```
nat_add(0,X,X).
nat_add(s(X),Y,s(S)) :-
        nat_add(X,Y,S).
```

nat_prod/3: PREDICATE

Usage: `nat_prod(X,Y,P)`

`P` es el producto de multiplicar `X * Y`.

```
nat_prod(0,_1,0).
nat_prod(s(X),Y,P) :-
        nat_add(Y,Z,P),
        nat_prod(X,Y,Z).
```

nat_mod/3: PREDICATE

Usage: `nat_mod(X,Y,R)`

`R` es el resultado de la operación módulo, es decir, el resto de la división `X / Y`.

```
nat_mod(X,Y,X) :-
        nat_gt(Y,X).
nat_mod(X,Y,R) :-
        nat_add(Z,Y,X),
        nat_mod(Z,Y,R).
```

nat_round/3: PREDICATE

Usage: `nat_round(X,R,Y)`

Redondea el número dado por `X` en función del resto, `R`, obtenido en una división anterior al número entero más cercano, devolviendo el resultado del redondeo en `Y`. Para la toma de decisiones, especialmente en la mitad de dos números, se toma como referencia el estándar IEEE 754 de 2008.

```
nat_round(X,R,s(X)) :-
        nat_prod(R,s(s(0)),Y),
        nat_gt(Y,X).
nat_round(X,R,X) :-
        nat_prod(R,s(s(0)),Y),
        nat_gt(X,Y).
nat_round(X,R,s(X)) :-
        nat_prod(R,s(s(0)),Y),
        nat_eq(Y,X),
        nat_odd(X).
nat_round(X,R,X) :-
        nat_prod(R,s(s(0)),Y),
        nat_eq(Y,X),
        nat_even(X).
```

list/1: PROPERTY

Usage: `list(X)`

`X` es una lista.

```
list([]).
list([_1|Xs]) :-
        list(Xs).
list([]).
list([_1|L]) :-
        list(L).
```

list_members_nat/1: PROPERTY

Usage: `list_members_nat(X)`

`X` es una lista compuesta por números naturales en notación de Peano.

```
list_members_nat([X]) :-
        nat(X).
list_members_nat([X|Xs]) :-
        nat(X),
        list_members_nat(Xs).
```

sublist_members_nat/1: PROPERTY

Usage: `sublist_members_nat(X)`

`X` es una lista compuesta por listas donde sus elementos son números naturales en notación de Peano.

```
sublist_members_nat([]).
sublist_members_nat([X|Xs]) :-
        list_members_nat(X),
        sublist_members_nat(Xs).
```

list_same_length/2: PROPERTY

Usage: `list_same_length(Xs,Ys)`

Comprueba si la lista `Xs` tiene la misma longitud que `Ys`.

```
list_same_length(Xs,Ys) :-
        list_length(Xs,Zs1),
        list_length(Ys,Zs2),
        nat_eq(Zs1,Zs2).
```

list_select/3: PREDICATE

Usage: `list_select(X,N,Y)`

Devuelve en `Y` el `N`-ésimo número de la lista `X`.

```
list_select([X|_1],s(0),X).
list_select([_1|Xs],s(N),Y) :-
        list_select(Xs,N,Y).
```

list_append/3: PREDICATE

Usage: `list_append(Xs,Ys,Zs)`

`Zs` será el resultado de introducir la lista `Ys` al final de la lista `Xs`.

```
list_append([],Ys,Ys).
list_append([X|Xs],Ys,[X|Zs]) :-
        list_append(Xs,Ys,Zs).
```

list_length/2: PREDICATE

Usage: `list_length(Xs,L)`

Calcula la longitud de una lista `Xs`, devolviéndolo en `L`.

```
list_length([],0).
list_length([_1|Xs],s(L)) :-
        list_length(Xs,L).
```

list_reverse/2: PREDICATE

Usage: `list_reverse(Xs,Ys)`

`Ys` será la lista `Xs` del revés, es decir, intercambiando cada elemento 'n' de `Xs` por longitud(`Xs`) -'n' - 1.

```
list_reverse(Xs,Ys) :-
        list_reverse(Xs,[],Ys).
```

list_reverse/3: PREDICATE

Usage: `list_reverse(Xs,Acc,Ys)`

`Ys` será la lista `Xs` del revés. Se genera mediante el uso de un acumulador de elementos.

```
list_reverse([],Ys,Ys).
list_reverse([X|Xs],Acc,Ys) :-
        list_reverse(Xs,[X|Acc],Ys).
```

list_flatten/2: PREDICATE

Usage: `list_flatten(Xs,Ys)`

Aplana la lista `Xs`, devolviendo el resultado en `Ys`. El aplanado consiste en generar una lista de elementos, en este caso naturales, a partir de una lista cuyos elementos son listas.

```
list_flatten([],[]).
list_flatten([X|Xs],Y) :-
        list_flatten(X,Ys1),
        list_flatten(Xs,Ys2),
        list_append(Ys1,Ys2,Y).
list_flatten(X,[X]) :-
        nat(X).
```

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)