Memoria Práctica 1 de Programación Declarativa

Kiyoshi José Omaza Saldaña
v13m080

Sergio Fontán Llamas
u12m073

Pablo Puado García
v13m047

Viernes 4 de Mayo, 2018

Contenidos

1 Redondeo de números decimales **a**

1.1 Código utilizado . . . . . b

1.1.1 Predicado *agregaUltimo*/3 . . . . . b

1.1.2 Predicado *mayorIgual*/2 . . . . . b

1.1.3 Predicado *menor*/2 c

1.1.4 Predicado *reverse*/3 c

1.1.5 Predicado *acarreoDecimal*/3 c

1.1.6 Predicado *acarreoEntero*/2 c

1.1.7 Predicado *sumaUno*/3 . . . . . d

1.1.8 Predicado *acarreoTotal*/4 . . . . . d

1.1.9 Predicado *completaAcarreo*/4 . . . . . d

1.1.10 Predicado *redondeo*/3 . . . . . e

1.1.11 Predicado *parteEnteraDecimal*/4 . . . . . e

1.1.12 Predicado *redondearDecimal*/3 . . . . . f

1.2 Pruebas realizadas . . . . . f

2 Cuadrados fantásticos y secretos **f**

2.1 Código utilizado . . . . . g

2.1.1 Predicado *primerElemento*/2 . . . . . g

2.1.2 Predicado *ultimoElemento*/2 . . . . . g

2.1.3 Predicado *suma*/2 . . . . . g

2.1.4 Predicado *triplicar*/2 . . . . . h

2.1.5 Predicado *esCuadradoFantasticoSecreto*/2 . . . . . h

2.2 Pruebas realizadas . . . . . h

1 Redondeo de números decimales

El redondeo es el proceso de descartar ciertas cifras en un número decimal. Se utiliza con el objetivo de manejar los números con mayor facilidad para poder realizar cálculos con ellos.
Hay diferentes formas de realizar el redondeo. En esta práctica se va a tener en cuenta el siguiente enfoque:

- *Redondeo a las unidades*: Si la primera cifra despues de la coma del número decimal es menor que 5, no hay que realizar ninguna operación y el resultado del proceso es la parte entera del número decimal. Por el contrario, si dicha cifra es mayor o igual que 5, entonces se debe sumar 1 (una unidad) al número. Por ejemplo:

$$5,36 \rightarrow 5$$

$$32,74 \rightarrow 33$$

- *Redondeo a las décimas*: Si la cifra de las centésimas (es decir, la segunda cifra después de la coma) es menor que 5, no hay que realizar ninguna operación y el resultado del proceso es el número original hasta la cifra que representa las décimas. Por el contrario, si dicha cifra es mayor o igual que 5, entonces se debe sumar 1 a la cifra que representa las décimas (descartando en este caso el resto de decimales). Por ejemplo:

$$32,74 \rightarrow 32,7$$

$$5,36 \rightarrow 5,4$$

- *Redondeo a las centésimas*: Si la cifra de las milésimas (es decir, la tercera cifra después de la coma) es menor que 5, no hay que realizar ninguna operación y el resultado del proceso es el número original hasta la cifra que representa las centésimas. Por el contrario, si dicha cifra es mayor o igual que 5, entonces se debe sumar 1 a la cifra que representa las centésimas (descartando en este caso el resto de decimales). Por ejemplo:

$$32,743 \rightarrow 32,74$$

$$5,369 \rightarrow 5,37$$

Para facilitar el autoaprendizaje del proceso de redondeo, la escuela María Martín quiere contar con un programa que realice dicho proceso y que mantenga la representación del número original a redondear así como del número obtenido después del redondeo.
Para cubrir esta necesidad se plantea la realización de un **programa lógico puro** en el que:

1. El número a redondear se representa como una lista de sus cifras (entre 0 y 9, en representación de **Peano**) en el orden correspondiente, incluyendo en la posición adecuada el separador decimal (la coma).

2. El resultado del redondeo se representa como una estructura *redondeo*/3, en la que el primer argumento representa el tipo de redondeo (a unidad, a las décimas o a las centésimas), el segundo argumento representa el número a redondear mediante la estructura *numeroOriginal*/3, y el tercer argumento representa el número redondeado con la estructura *numeroRedondeado*/3.
   Tanto *numeroOriginal*/3 como *numeroRedondeado*/3 tienen como primer argumento el separador decimal, como segundo argumento una lista con las cifras de la parte entera del número, y como tercer argumento una lista con las cifras de la parte decimal del número.

Se pide que los alumnos escriban el predicado *redondearDecimal*/3:

```
1 redondearDecimal(NumeroInicial, TipoRedondeo, NumeroFinal).
```

Por ejemplo:

```
1 ?-redondearDecimal([s(s(s(s(s(0))))),',',s(s(s(0)))],
      redondeoUnidad,redondeo(redondeoUnidad,numeroOriginal(',',[s(
      s(s(s(s(0)))))],[s(s(s(0)))]),numeroRedondeado(',',[s(s(s(s
      (0)))))],[])).
2 yes
```

1.1 Código utilizado

1.1.1 Predicado *agregaUltimo*/3

```
1 agregaUltimo([],X,[X]).
2 agregaUltimo([H|T],X,[H|L]):-agregaUltimo(T,X,L).
```

Este predicado recibe como argumento dos listas y un elemento.
El predicado agrega un elemento dado al final de una lista

1.1.2 Predicado *mayorIgual*/2

```
1 mayorIgual(0,0).
2 mayorIgual(s(_),0).
3 mayorIgual(s(X),s(Y)):-mayorIgual(X,Y).
```

Este predicado recibe como argumentos dos elementos.
El predicado realiza una comparación del valor de dos elementos e indica si uno de ellos es mayor o igual que el otro.

1.1.3 Predicado *menor*/2

```
menor(0,s(_)).
menor(s(X),s(Y)):-menor(X,Y).
```

Este predicado recibe como argumentos dos elementos.
El pedicado realiza una comparación del valor de dos elementos e indica si uno es menor que el otro.

1.1.4 Predicado *reverse*/3

```
reverse([],Z,Z).
reverse([H|T],Z,Acc) :- reverse(T,Z,[H|Acc]).
```

Este predicado recibe como argumentos una lista y dos elementos.
El predicado invierte una lista dada.

1.1.5 Predicado *acarreoDecimal*/3

```
acarreoDecimal([X|[]], [X|[]]):-mayorIgual(X,s(s(s(s(s(s(s(s(s(s(
    0))))))))))).
acarreoDecimal([X|Y], [X|Y]):- menor(X,s(s(s(s(s(s(s(s(s(0))))
    ))))))).
acarreoDecimal([X|[Y|Z]], [0|F]):-
    mayorIgual(X,s(s(s(s(s(s(s(s(s(0))))))))))),
    acarreoDecimal([s(Y)|Z], F).
```

Este predicado recibe como argumentos dos listas.
El predicado obtiene los acarreos de la parte decimal del número para luego realizar la operación necesaria para el redondeo.

1.1.6 Predicado *acarreoEntero*/2

```
acarreoEntero([X|[]], [0,s(0)]):-mayorIgual(X,s(s(s(s(s(s(s(s(
    s(0))))))))))).
acarreoEntero([X|Y], [X|Y]):- menor(X,s(s(s(s(s(s(s(s(s(0)))))
    ))))).
acarreoEntero([X|[Y|Z]], [0|F]):-
    mayorIgual(X,s(s(s(s(s(s(s(s(s(0))))))))))),
```

```
5     acarreoEntero([s(Y)|Z], F).
```

El predicado recibe como argumentos dos listas.
El predicado obtiene los acarreos de la parte entera del número para luego realizar la operación necesaria para el redondeo.

1.1.7 Predicado *sumaUno*/3

```
1 acarreoEntero([X|[]], [0,s(0)]):-mayorIgual(X,s(s(s(s(s(s(s(s(
      s(0)))))))))).
2 acarreoEntero([X|Y], [X|Y]):- menor(X,s(s(s(s(s(s(s(s(s(0)))))
      ))))).
3 acarreoEntero([X|[Y|Z]], [0|F]):-
4     mayorIgual(X,s(s(s(s(s(s(s(s(s(0))))))))),
5     acarreoEntero([s(Y)|Z], F).
```

Este predicado recibe como argumentos tres elementos y uno de ellos puede ser una lista.
El predicado suma uno a la parte del número indicada dependiendo del acarreo.

1.1.8 Predicado *acarreoTotal*/4

```
1 acarreoTotal(ParteEntera, [X|Y], ParteEntera, [X|Y]):- menor(X,s
      (s(s(s(s(s(s(s(s(0))))))))).
2 acarreoTotal([Y|Z], [X|_], F, []):-
3     mayorIgual(X,s(s(s(s(s(s(s(s(s(0))))))))),
4     acarreoEntero([s(Y)|Z], F).
```

Este predicado recibe como argumentos dos elementos y dos listas.
Esta función muestra el acarreo total que se obtiene al realizar el redondeo.

1.1.9 Predicado *completaAcarreo*/4

```
1 completaAcarreo(ParteEntera, ParteDecimal, EnteroFinal,
      DecimalFinal):-
2     reverse(ParteEntera, InversoEntero, []),
3     reverse(ParteDecimal, InversoDecimal, []),
4     acarreoDecimal(InversoDecimal, DecimalConAcarreo),
5     reverse(DecimalConAcarreo, ParteDecimalNueva, []),
6     acarreoTotal(InversoEntero, ParteDecimalNueva,
      EnteroConAcarreo, DecimalFinal),
7     reverse(EnteroConAcarreo, EnteroFinal, []).
```

Este predicado recibe como argumentos cuatro elementos.
Se trata de una función auiliar para poder realizar las demas operaciones de acarreo correctamente.

1.1.10 Predicado *redondeo*/3

```
redondeo(redondeoUnidad, numeroOriginal(',', ParteEntera, [Y|_])
    , numeroRedondeado(',',EnteroFinal,[])):-
    reverse(ParteEntera, [X|Z]),
    sumaUno(X,Y, XNuevo),
    acarreoEntero([XNuevo|Z], EnteroConAcarreo),
    reverse(EnteroConAcarreo, EnteroFinal, []).

redondeo(redondeoDecima, numeroOriginal(',', ParteEntera, [Y]),
    numeroRedondeado(',',ParteEntera, [Y])).
redondeo(redondeoDecima, numeroOriginal(',', ParteEntera, [Y,Z|_
    ]), numeroRedondeado(',',EnteroFinal, DecimalFinal)):-
    sumaUno(Y,Z, YNueva),
    completaAcarreo(ParteEntera, [YNueva], EnteroFinal,
    DecimalFinal).

redondeo(redondeoCentesima, numeroOriginal(',', ParteEntera, [Y,
    Z]), numeroRedondeado(',',ParteEntera, [Y,Z])).
redondeo(redondeoCentesima, numeroOriginal(',', ParteEntera, [Y,
    Z,W|_]), numeroRedondeado(',',EnteroFinal, DecimalFinal)):-
    sumaUno(Z,W, ZNueva),
    completaAcarreo(ParteEntera, [Y,ZNueva], EnteroFinal,
    DecimalFinal).
```

Este predicado recibe como argumentos tres elementos.
El funcionamiento de *redondeo*/3 recibe un número, el tipo de redondeo y el número redondeado y compara ambos números.

1.1.11 Predicado *parteEnteraDecimal*/4

```
parteEnteraDecimal([','|ParteDecimal], ParteEntera, ParteEntera,
    ParteDecimal).
parteEnteraDecimal([X|Z], ParteEntera, ParteEnteraFinal,
    ParteDecimal):-
    agregaUltimo(ParteEntera, X, ParteEnteraNueva),
    parteEnteraDecimal(Z, ParteEnteraNueva, ParteEnteraFinal,
    ParteDecimal).
```

Este predicado recibe como argumentos una lista y 3 elementos.
El predicado devuelve el número dado en el formato que más nos conviene para el buen funcionamiento de las otras funciones.

1.1.12 Predicado *redondearDecimal*/3

```
1 redondearDecimal(NumeroInicial, TipoRedondeo, NumeroFinal):-
2     parteEnteraDecimal(NumeroInicial,[],ParteEntera,
      ParteDecimal),
3     redondeo(TipoRedondeo, numeroOriginal(',', ParteEntera,
      ParteDecimal), NumeroFinal).
```

Este predicado recibe como argumentos tres elementos.
El predicado, dado el número inicial divide este en parte entera y parte decimal, y dependiendo de el tipo de redondeo que se le haya indicado lo realizará y se devolverá el número final junto otra vez.

1.2 Pruebas realizadas

2 Cuadrados fantásticos y secretos

Un cuadrado fantástico es una tabla o matriz de números enteros pares, de tal forma que la suma de los números por columnas y por filas es la misma (es decir, todas las columnas y todas las filas suman la misma cantidad). Los números pares empleados para rellenar las casillas de dicho cuadrado son consecutivos, de 2 a $(2 * N)$, siendo N el número de columnas y filas del cuadrado fantástico.
Por ejemplo, las siguiente matriz es un cuadrado fantástico en el que todas las columnas y todas las filas suman 12.

2	4	6
6	2	4
4	6	2

Un cuadrado fantástico es secreto cuando cumple la propiedad de las tres esquinas. Esta propiedad se verifica cuando 2 esquinas están rellenas con el mismo número par (A), y la suma de las otras dos esquinas es igual a dicho par $(B + C = A)$. Dicho número par (A) se denomina número secreto.
Por ejemplo, la siguiente matriz es un cuadrado fantástico secreto $(A = 8; B = 2; C = 6)$ cuyo número secreto es 8.

2	4	6	8
4	6	8	2
6	8	2	4
8	2	4	6

Se plantea la realización de un programa lófico puro en el que las matrices se representan como listas de listas (es decir, una matriz es una lista cuyos

elementos son listas de números, en representación de Peano, que representan las filas).
Se pide que los alumnos escriban el predicado *esCuadradoFantasticoSecreto*/2 que se verifique si su primer argumento (una matriz $M * M$) representa un cuadrado fantástico 2 secreto y su segundo argumento es el número secreto. Por ejemplo:

```
?-esCuadradoFantasticoSecreto([[s(s(0)),s(s(s(s(0)))),s(s(s(s(s(s(0))))))],[s(s(s(s(0)))),s(s(s(s(s(s(0)))))),s(s(0))],[s(s(s(s(s(s(0)))))),s(s(0)),s(s(s(s(0))))]],s(s(s(s(s(s(s(0))))))).
yes
```

2.1 Código utilizado

2.1.1 Predicado *primerElemento*/2

```
primerElemento([X|_],X).
```

Este predicado tiene como argumentos una lista y un elemento dado (que puede ser una lista).
La consulta devuelve un valor true si el elemento dado es la cabeza de la lista que se le esta pasando, y en caso de que sea una matriz se devuelve la primera fila. En caso contrario devolverá false.

2.1.2 Predicado *ultimoElemento*/2

```
ultimoElemento([Y],Y).
ultimoElemento([_|Xs],Y):-ultimoElemento(Xs,Y).
```

Este predicado tiene como argumentos una lista y un elemento dado (que puede ser una lista).
La consulta devuelve true si el elemento dado es el último elemento de la lista que se le esta pasando, y en caso de que sea una matriz se devuelve la última fila. En caso contrario devolverá false.

2.1.3 Predicado *suma*/2

```
suma(X,0,X).
suma(A,s(B),s(Y)):-suma(A,B,Y).
```

A este predicado se le pasan como argumentos tres elementos. La consulta devuelve true si el tercer elemento dado es la suma de los dos anteriores. En caso contrario se devuelve false.

2.1.4 Predicado *triplicar*/2

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

A este predicado se le pasan como argumentos dos elementos. La consulta devuelve true si el segundo elemento dado tiene un valor igual al triple del primer elemento. En caso contrario devolverá false.

2.1.5 Predicado *esCuadradoFantasticoSecreto*/2

```
esCuadradoFantasticoSecreto(M,S):-
  primerElemento(M,L1),
  primerElemento(L1,X1),

  ultimoElemento(M,Ln),
  ultimoElemento(Ln,Xn),

  suma(X1,Xn,S1),

  ultimoElemento(L1,X2),
  primerElemento(Ln,Xn1),

  suma(X2,Xn1,S2),
  suma(S1,S2,S3),

  triplicar(S,S3).
```

Este predicado tiene como argumentos una lista y un elemento dado. La consulta devuelve true si el elemento es *número secreto*. En caso contrario devolverá false.

2.2 Pruebas realizadas

```
?-esCuadradoFantasticoSecreto([[s(s(0)),s(s(s(s(0)))),s(s(s(s(s(
    s(0))))))],[s(s(s(s(0)))),s(s(s(s(s(s(0)))))),s(s(0))],[s(s(s
    (s(s(s(0))))))),s(s(0)),s(s(s(s(0))))]],s(s(s(s(s(s(0)))))))).
yes

?-esCuadradoFantasticoSecreto([[s(s(0)),s(s(s(s(0)))),s(s(s(s(s(
    s(0))))))],[s(s(s(s(0)))),s(s(s(s(s(s(0)))))),s(s(0))],[s(s(s
    (s(s(s(0))))))),s(s(0)),s(s(s(s(0))))]],S).
```

```
5 S = s(s(s(s(s(0))))))
6
7 ?-esCuadradoFantasticoSecreto([[s(s(0)),s(s(s(s(0)))),s(s(s(s(
      s(0))))),s(s(s(s(s(s(s(0))))))))],[s(s(s(s(0)))),s(s(s(s
      (s(0))))),s(s(s(s(s(s(s(0)))))))),s(s(0))],[s(s(s(s(s(s(0
      ))))),s(s(s(s(s(s(s(0)))))))),s(s(0)),s(s(s(s(0))))],[s(s(s
      (s(s(s(s(0)))))))),s(s(0)),s(s(s(s(0)))),s(s(s(s(s(0)))))
      )]],S).
8 S = s(s(s(s(s(s(s(0)))))))
```