MateFun - Manual

Sylvia da Rosa, darosa@fing.edu.uy[3] and Marcos Viera, mviera @fing.edu.uy[3]

[3]Instituto de Computación - Facultad de Ingeniería - Universidad de la República

I. Introducción

El lenguaje MateFun es un lenguaje de programación funcional pura dirigido al aprendizaje de funciones matemáticas, diseñado por investigadores del Instituto de Computación (InCo) de la Facultad de Ingeniería (FING) UDELAR. MateFun puede ser accedido a través de un entorno de programación integrado web[1] que permite gestionar programas, programar, ejecutar programas y visualizar gráficas, figuras y animaciones. La sintaxis del lenguaje fue diseñada con el objetivo de ser minimal y lo más parecida posible a la notación utilizada en matemáticas. La idea es que el lenguaje se pueda asimilar rápidamente y que su relación con los conceptos matemáticos subyacentes pueda ser reconocida por los estudiantes. Considerando que el público objetivo está compuesto por estudiantes de secundaria de habla hispana, las palabras clave se definieron en español.

Un programa en MateFun es una lista de definiciones de **conjuntos** y **funciones**.

En las siguientes secciones se presentan los elementos básicos de MateFun.

II. Conjuntos

Los conjuntos se utilizan para determinar el dominio y codominio de las funciones (corresponden a los tipos de los lenguajes de programación).

```
conj Mes = { Enero , Febrero , Marzo
           , Abril , Mayo , Junio
           , Julio , Agosto , Setiembre
           , Octubre , Noviembre , Diciembre }

conj Rno0 = { x en R | x /= 0 }
```

Figura 1: Ejemplos de conjuntos definidos por el usuario

En la Figura 1 se muestran algunos ejemplos de definiciones de conjuntos. Un conjunto se define usando la palabra reservada **conj**, se le asigna un nombre y los elementos que contiene.

El nombre de un conjunto es una cadena de caracteres alfanuméricos que comienza con un caracter alfabético en mayúscula.

Los elementos se pueden expresar por enumeración o por comprensión. Para definir un conjunto *por enumeración* se deben enumerar, separados por coma, los elementos (cadenas alfanuméricas que comienzan con mayúscula) que lo integran.

Por ejemplo, Mes en la Figura 1 es un conjunto definido por enumeración que contiene los elementos Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio , Agosto, Setiembre, Octubre, Noviembre y Diciembre.

Para definir un conjunto *por comprensión* se debe establecer un conjunto de partida y la condición que deben cumplir los elementos de dicho conjunto para pertenecer al conjunto que se está definiendo. Por lo tanto el nuevo conjunto será subconjunto del conjunto de partida.

Los conjuntos pueden ser elementales o compuestos. Los conjuntos elementales son los primitivos (**R** para los reales, **Z** para los enteros, **Fig** para las figuras en dos dimensiones, **Fig3D** para las figuras en tres dimensiones y **Color** para los colores) y los enumerados definidos por el usuario. También son elementales los conjuntos definidos por comprensión que tienen como conjunto de partida un conjunto elemental. Los conjuntos compuestos son las n-tuplas de conjuntos y las secuencias de un conjunto. La n-tupla es la generalización del producto cartesiano. Por ejemplo la tupla **R** X **R** contiene pares de reales, que se escriben entre paréntesis y separados por coma; un elemento de este conjunto es (2.3,4.2) . La secuencia es lo que en los lenguajes de programación se suele llamar lista; i.e. el conjunto de las secuencias de elementos de un conjunto dado. Una posible secuencia de reales (**R**∗) es 1.8**:**9.3**:**2.4**:[]**.

Una condición puede ser una relación binaria (por ejemplo x == 0) entre dos expresiones o una lista de estas relaciones, entre paréntesis y separadas por comas, por ejemplo (0 < x , x < 10). Las relaciones binarias son las usuales de igualdad y orden (== , /= , < , > , <= , >=). Notar que la relación de igualdad se indica con ==.

En el caso de tener una lista de relaciones, el resultado de su evaluación es la conjunción lógica ($\wedge$) entre todas ellas. El conjunto Rno0 de la Figura 1 representa al conjunto de los reales (**R**) distintos de 0. Notar que las variables, a diferencia de los nombres y elementos de conjuntos, comienzan con minúsculas.

III. Funciones

Para definir una función se debe indicar su **signatura** y la **ecuación** que la define.

La signatura se compone del nombre de la función, el conjunto dominio y el codominio.

Por ejemplo, la signatura de la función cuad de la Figura 2 es cuad **::** **R** −**>R**, indicando que la función cuad va de reales a reales. En la signatura de la función areaTria podemos ver

[1]https://matefun.math.psico.edu.uy/#/es/login

```
cuad :: R -> R
cuad(x) = x * x

inverso :: Rno0 -> R
inverso(x) = 1 / x

areaTria :: R X R -> R
areaTria(base,alt) = (base * alt) / 2
```

Figura 2: Ejemplos de funciones definidas por el usuario

que para representar funciones de múltiples variables (en este caso dos reales) utilizamos tuplas como dominio. Notar que en inverso utilizamos como dominio el conjunto Rno0 definido en la Figura 1.

La ecuación se define dando el nombre de la función, las variables independientes (i.e. parámetros) y el cuerpo de la función. El cuerpo de una función se compone de una expresión o de una lista de expresiones.

En el caso de la función cuad, definida en la Figura 2, la variable independiente es x y el cuerpo de la función es la expresión x ∗ x.

En el caso de que el cuerpo sea una lista de expresiones, cada expresión se ejecuta al cumplirse una condición (si no se cumple ninguna de las anteriores). La lista finaliza con una expresión por defecto, que se ejecuta si no se cumple ninguna de las condiciones. Por ejemplo, en la función abs, definida en la Figura 3, la variable independiente es x y el cuerpo de la función es la lista de las expresiones x (se ejecuta si x >= 0) y -x (se ejecuta si no se cumple x >= 0).

Las condiciones de cada caso se analizan en el orden en que están definidos (de "arriba" hacia "abajo") y una vez que se cumple una de ellas, se evalúa la expresión asociada y no se analizan más casos. Para asegurar que las funciones sean totales en el dominio, el lenguaje impone la existencia de un caso por defecto.

La función dias retorna la cantidad de días que (generalmente) tiene un mes. Hay dos aspectos importantes a destacar sobre la función dias que tienen que ver con las decisiones de diseño del lenguaje. El primero es que la función no es numérica, esto es importante porque uno de los objetivos de enseñar funciones mediante la programación es reforzar la idea de que no todas las funciones son numéricas. El otro es la *verbosidad* de la función, que es un precio a pagar debido a la simplicidad del lenguaje; en este caso por la falta de expresiones condicionales (del tipo if-then-else) y operadores booleanos. Esto no nos resulta particularmente preocupante, dado que priorizamos el rápido aprendizaje en lugar de la utilidad del lenguaje para la implementación de aplicaciones de mediano o gran porte.

Un valor literal puede ser un elemento de un conjunto definido por enumeración, un número, un color, una figura vacía o una secuencia vacía.

Los operadores son los operadores aritméticos (+ , - , * , / , ^), que están definidos para **Z**, **R** y todos sus sub-conjuntos, más la proyección de una tupla (**!**) y el insertar al inicio de una secuencia (**:**).

```
abs :: R -> R
abs(x) =  x si x >= 0
        o -x

max :: R X R -> R
max(x,y) = x si x >= y
           o y

dias :: Mes -> R
dias(m) = 31 si m == Enero
        o 28 si m == Febrero
        o 31 si m == Marzo
        o 30 si m == Abril
        o 31 si m == Mayo
        o 30 si m == Junio
        o 31 si m == Julio
        o 31 si m == Agosto
        o 30 si m == Setiembre
        o 31 si m == Octubre
        o 30 si m == Noviembre
        o 31
```

Figura 3: Ejemplos de funciones definidas por casos

```
areaCirc :: R -> R
areaCirc(r) = 3.14 * cuad(r)

factorial :: R -> R
factorial(x) = x * factorial(x-1) si x > 0
             o 1
```

Figura 4: Ejemplos de funciones con aplicación de funciones

Una expresión puede ser también la aplicación de una función, que se realiza escribiendo el nombre de la función (fun) y entre paréntesis, separados con comas, sus argumentos. Las funciones pueden ser las declaradas por el usuario o las funciones primitivas, definidas para operar con números, secuencias, colores y figuras. Por ejemplo, en el cuerpo de la función areaCirc de la Figura 4 se aplica la función cuad a la variable r. Por lo tanto, esta función calcula el área de un triángulo dado su radio r. La función factorial tiene una definición recursiva, es decir que se invoca a sí misma.

Además de los operadores aritméticos, el lenguaje provee de funciones primitivas para redondear (red), hallar el seno (sen), coseno (cos) y raíz cuadrada (raizcuad) de un número.

IV. SECUENCIAS

Para poder manipular *colecciones de elementos* de una forma simple, el lenguaje incorpora la noción de secuencia. Las secuencias se definen de manera inductiva, con el valor

```
suma :: R* -> R
suma(l) = 0 si l == []
        o primero(l) + suma(resto(l))

largo :: R* -> R
largo(l) = 0 si l == []
         o 1 + largo(resto(l))


conj RSeqNV = { l en R* | largo(l) /= 0 }

maximo :: RSeqNV -> R
maximo (l) = primero(l) si resto(l) == []
           o max( primero(l)
                , maximo(resto(l)))
```

Figura 5: Ejemplos de funciones recursivas sobre secuencias

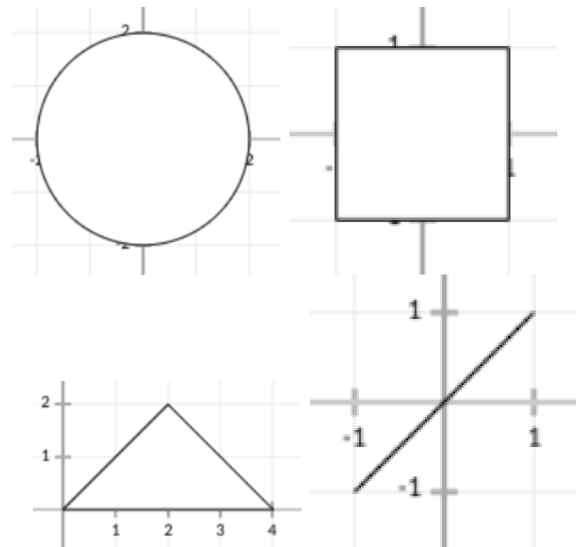

Figura 6: Ejemplos de figuras 2D

[] de secuencia vacía y el operador **:** que a partir de un elemento y una secuencia de elementos (todos pertenecientes al mismo conjunto), retorna la secuencia resultante de agregar el elemento al principio de la secuencia dada.

Por ejemplo una secuencia formada por elementos del conjunto {3,2,1,4} se construye de la siguiente manera: partimos de la secuencia vacía **[]** y vamos agregando elementos al principio de la secuencia anterior con **:**

```
[]
2:[]
3:2:[]
1:3:2:[]
4:1:3:2:[]
```

Se utiliza una notación más simple, escribiendo [4,1,3,2] .

Por ejemplo, utilizando secuencias podemos definir la función "tienen", que dado un número de días, retorna todos los meses que tienen en total ese número de días (y la secuencia vacía si ningún mes los tiene):

```
tienen :: R -> Mes*
tienen(d) = [Abril,Junio,Setiembre,
             Noviembre] si   d == 30
          o [Enero,Marzo,Mayo,Julio,
             Agosto,Octubre,
             Diciembre] si   d == 31
          o [Febrero]    si ( d == 28
                             , d == 29)
          o []
```

Además de esa forma de construcción de secuencias, la función **rango** permite construir secuencias de racionales a partir de un valor inicial, un valor final y un valor de paso. Por ejemplo **rango**(0,100,10) retorna la secuencia [0,10,20,30,40,50,60,70,80,90,100] .

Para poder obtener los elementos de una secuencia existen las funciones *destructoras* **primero** y **resto**, que respectivamente retornan el primer elemento de una secuencia y la secuencia resultante de quitar el primer elemento. Por ejemplo la expresión **primero**(**rango**(0,100,10)) retorna el valor real 0 y **resto**(**rango**(0,100,10)) retorna la secuencia [10,20,30,40,50,60,70,80,90,100] . Notar que en estos ejemplos hemos realizado *composición de funciones*, dado que a **primero** y **resto** les hemos pasado el resultado de aplicar la función **rango**.

Dada la naturaleza inductiva del conjunto secuencia, es natural que muchas funciones que operan sobre secuencias se definan usando recursión. En la Figura 5 se muestran algunos ejemplos de este tipo de funciones. La función suma, para obtener la suma de una secuencia de reales, suma el primer real de la secuencia con el resultado de la suma del resto (considerando que la secuencia vacía suma 0). De forma similar se puede calcular el largo de una secuencia. Para la función maximo, que calcula el máximo de una secuencia de reales no vacía, definimos un nuevo conjunto RSeqNV, de manera que la misma no sea parcial en su dominio. Notar que dentro de las condiciones de los conjuntos se pueden utilizar funciones definidas en el mismo programa.

V. Colores, Figuras y Animaciones

En MateFun es posible definir funciones que permiten la creación y manipulación de figuras. Además de los colores predefinidos, se pueden obtener nuevos colores utilizando la función **rgb**, que a partir de tres números entre 0 y 255 que indican respectivamente el aporte de rojo, verde y azul (*red*, *green*, *blue*), compone el color resultante.

Las figuras pueden crearse en dos dimensiones (**Fig**) o en tres dimensiones (**Fig3D**).

La función **aFig** retorna una figura 2D con el texto de un enumerado, **rect** retorna un rectángulo a partir de su base y altura, **circ** retorna un rectángulo a partir de su radio, **linea** retorna un segmento de recta dados dos puntos (**R** X **R**) y **poli** retorna un polígono que une la secuencia de puntos (**R***) que recibe como parámetro. Salvo **linea** y **poli**, las demás figuras se crean siempre en el centro (0,0) de un sistema de coordenadas cartesianas. En la Figura 6 se muestran

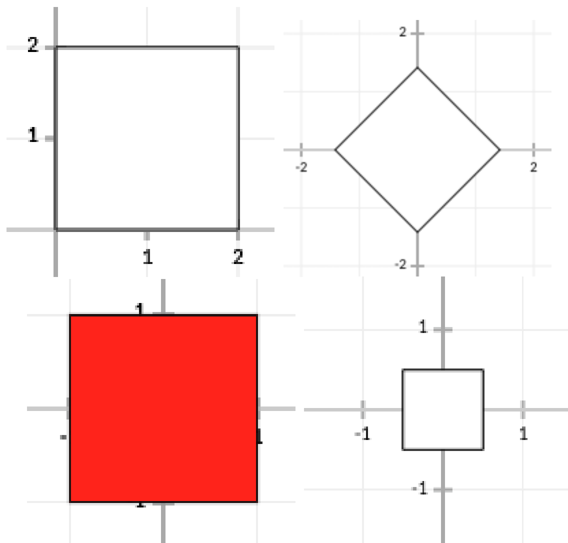

Figura 7: Transformaciones de figuras

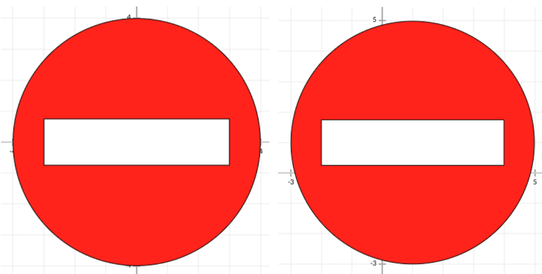

Figura 8: Juntar figuras

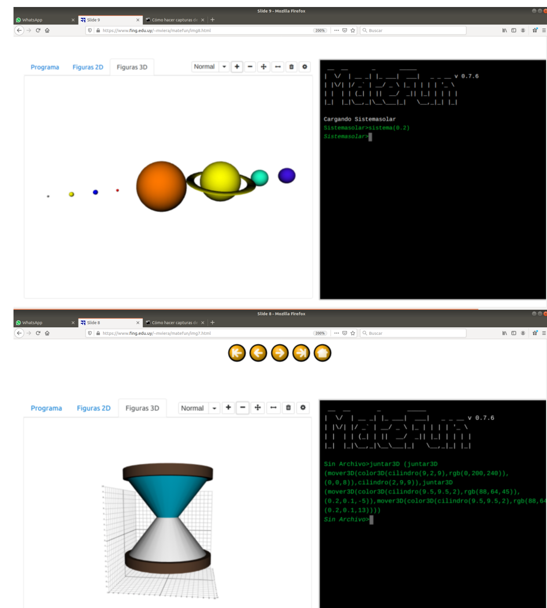

Figura 9: Ejemplos de figuras 3D

las figuras generadas por las expresiones **circ** (2), **rect** (2,2), **poli** ([(0,0),(2,2),(4,0)]) y **linea** ((−1,−1),(1,1)). Las figuras se dibujan en un plano cartesiano; las generadas con **aFig**, **rect** y **circ**, se colocan con centro en el punto (0,0). Dos figuras se pueden unir en una nueva figura utilizando la función **juntar**. A una figura se la puede pintar con un color dado utilizando la función **color**. También a una figura se le puede mover, rotar y escalar utilizando las funciones homónimas. En la Figura 7 se muestran las figuras generadas por las expresiones **mover**(**rect** (2,2),(1,1)) , **rotar**(**rect** (2,2),45) , **color**(**rect** (2,2), **Rojo**) y **escalar**(**rect** (2,2),0.5) .

En la Figura 8 se muestra las figura generada por la expresión **juntar**(**color**(**circ** (4), **Rojo**), **rect** (6,1.5)) , que une un círculo rojo con un rectángulo. También se muestra el efecto de mover esa figura al punto (1,1) haciendo **mover**(**juntar**(**color**(**circ** (4), **Rojo**), **rect** (6,1.5)),(1,1)) .

Para las figuras 3D se cuenta con las funciones **juntar3D**, **color3D**, **mover3D**, **rotar3D** y **escalar3D**, que son similares a sus equivalentes en 2D. Se puede crear una línea 3D usando la función **linea3D**. Tambien se pueden crear esferas (**esfera**), cilindros (**cilindro**), cubos (**cubo**) y anillos (**anillo**). La Figura 9 muestra ejemplos de figuras 3D que se pueden generar combinando algunas de estas funciones.

En MateFun, una animación se define como una secuencia de figuras. Entonces por ejemplo la siguiente función produce una animación que rota una figura 2D dada de a 10 grados tantas veces como se le indique.

```
rotFig :: Fig X R -> Fig*
rotFig(fig, cant)
   = [] si cant <= 0
   o fig : rotFig(rotar(fig,10),cant-1)
```

VI. El Intérprete

MateFun es un lenguaje *interpretado*, lo que significa los programas "se cargan" para ser ejecutados por un intérprete. Al cargar un programa todas sus definiciones quedan disponibles en el intérprete para ser utilizadas en su línea de comandos.

Por ejemplo, consideremos que los códigos de las figuras 1, 2, 3, 4 y 5, forman un programa llamado Ejemplos. Al cargarlo en el intérprete tendríamos disponibles, además de las funciones, operadores y conjuntos primitivos, todas las funciones y conjuntos que se definieron en esas figuras.

El comando más básico del intérprete consiste en evaluar expresiones, las cuales pueden hacer uso de todas las definiciones disponibles. La Figura 12 es un ejemplo de sesión válida.

El entorno de programación integrado web provee de una ventana gráfica en la cual se exhiben las figuras y animaciones en las que resulten las expresiones de este tipo. Las figuras 6, 7 y 8 son capturas de esta ventana.

Para el caso de funciones numéricas y no numéricas de una sola dimensión (i.e. con dominio y codominio **R** o un subconjunto de **R**) se puede utilizar esa ventana para visualizar sus gráficas. Esto se realiza con el comando **?grafica** del intérprete. Entonces al ejecutar

```
Ejemplos>?grafica cuad
```

se obtiene la gráfica de la Figura 11.

```
 __  __       _        _____
|  \/  | __ _| |_ ___|  ___|   _ _ __   v 0.4.5
| |\/| |/ _` | __/ _ \ |_ | | | | '_ \
| |  | | (_| | ||  __/  _|| |_| | | | |
|_|  |_|\__,_|\__\___|_|   \__,_|_| |_|

Cargando Ejemplos
Ejemplos>cuad(4)
16
Ejemplos>areaTria(2,4)
4
Ejemplos>maximo(2:4:1:[]) + 38
42
Ejemplos>
```

Figura 10: Intérprete

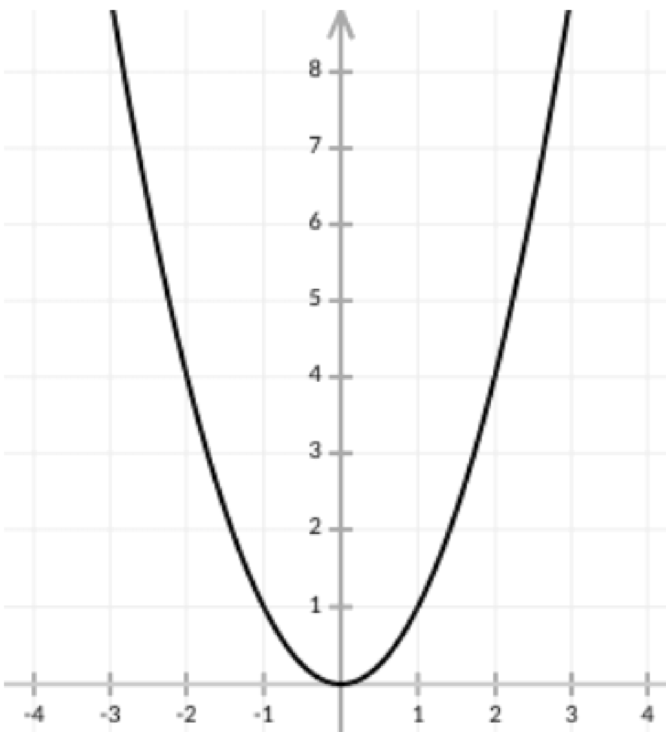

Figura 11: Gráfica de función

El comando **?funs** despliega una lista con la signatura de cada función disponible en el intérprete, como se muestra en la Figure 12.

VII. Errores y advertencias

Al cargar un programa en el intérprete se realiza un chequeo de errores. En el caso de que haya alguno, se despliega el mensaje correspondiente y la línea/columna del texto en el que ocurrió, y el programa no se ejecuta. Todos los textos de los mensajes son en español y refieren a errores conceptuales.

Supongamos que tenemos la función areaTria de la Figura 2 definida en un archivo llamado 'Ejemplos', y definimos una función 'areaRara' así:

```
Ejemplos>?funs
- :: R -> R
red :: R -> Z
sen :: R -> R
cos :: R -> R
raizcuad :: R -> R
rgb :: (R X R X R) -> Color
rect :: (R X R) -> Fig
circ :: R -> Fig
linea :: ((R X R) X (R X R)) -> Fig
poli :: (R X R)* -> Fig
juntar :: (Fig X Fig) -> Fig
color :: (Fig X Color) -> Fig
mover :: (Fig X (R X R)) -> Fig
rotar :: (Fig X R) -> Fig
escalar :: (Fig X R) -> Fig
aFig :: A -> Fig
linea3D :: ((R X R X R) X (R X R X R))
-> Fig3D
esfera :: R -> Fig3D
cilindro :: (R X R X R) -> Fig3D
cubo :: (R X R X R) -> Fig3D
anillo :: (R X R X R) -> Fig3D
juntar3D :: (Fig3D X Fig3D) -> Fig3D
color3D :: (Fig3D X Color) -> Fig3D
mover3D :: (Fig3D X (R X R X R)) ->
Fig3D
rotar3D :: (Fig3D X (R X R X R)) ->
Fig3D
escalar3D :: (Fig3D X R) -> Fig3D
rango :: (R X R X R) -> R*
primero :: A* -> A
resto :: A* -> A*
cuad :: R -> R
inverso :: Rno0 -> R
areaTria :: R X R -> R
abs :: R -> R
max :: R X R -> R
dias :: Mes -> R
areaCirc :: R -> R
factorial :: R -> R
suma :: R* -> R
largo :: R* -> R
maximo :: RSeqNV -> R
```

Figura 12: Comando **?funs**

```
areaRara :: R -> R
areaRara(x) = areaTria(x)
```

Al cargar ‘Ejemplos’ en el intérprete se produce el error

```
Error: {archivo: Ejemplos linea: 53
columna: 25}
Se esperan elementos de (R X R) pero se
encontro R.

No File>
```

También hay advertencias, que no impiden la ejecución del programa pero cuyo origen debe ser revisado porque puede ser fuente de errores futuros. O sea que las advertencias informan posibles problemas que en esa etapa no se han podido todavía detectar como errores. Si en el archivo ’Ejemplos’ definimos la función

```
mismo :: R -> Rno0
mismo(x) = x
```

se despliega una advertencia al cargarlo en el intérprete

```
Advertencia: {archivo: Ejemplos linea:
3 columna: 12}
Conjunto Rno0 requerido es subconjunto
del resultante R.
Por lo que existe la posibilidad de que
su valor quede fuera del conjunto.

Ejemplos>
```

y un error al aplicarla a 0, como se muestra a continuación:

```
Ejemplos>mismo(0)
Error: {archivo: Ejemplos linea: 3
columna: 12}
Valor 0 no pertenece al conjunto Rno0
porque no se cumple: [0 /= 0].
```