Programa del curso de Programación Funcional para la licenciatura en matemática.

UdelaR/FCien/CMat.

10 de agosto de 2016

1. Objetivo general

Presentar la informática como una *teoría matemática* y la programación como una *actividad matemática*, abordando un temario que sirva a la vez como un primer curso de programación.

La programación funcional está basada en la noción de recursión. A diferencia de lo que sucede en la programación imperativa, el código de un programa funcional no describe la manipulación de registros en una memoria, sino los casos base y las etapas de recursión que permiten calcular un valor. Las habilidades básicas de programación recursiva se adquieren rápidamente por su analogía con la recursión matemática. Los lenguajes funcionales constituyen entonces un excelente primer lenguaje de programación para alumnos de carreras científicas.

La recursión es un principio muy general de construcción, que requiere apenas de *relaciones bien fundadas*. Esto permite trabajar con estructuras más generales que los naturales, como ser los lenguajes. La sintaxis se puede entonces definir y estudiar como estructura matemática, lo cual es explotable a los efectos de demostrar propiedades acerca de los programas.

La elección del lenguaje Haskell, de reducción *lazy*, permite el trabajo con estructuras infinitas y estructuras cíclicas (con autorreferencias). La programación lazy permite escribir códigos muy sintéticos, para cuya elaboración y comprensión, el razonamiento matemático juega un rol fundamental. La programación lazy introduce naturalmente las nociones básicas de los *dominios de Scott*, que permiten caracterizar la calculabilidad en términos topológicos.

En Haskell, la introducción de instrucciones imperativas y de programas que interactúen con los periféricos se hace mediante mónadas. Además de su utilidad en programación, las mónadas constituyen una construcción usual en matemática. Es entonces la ocasión de mostrar que la programación es una teoría matemática como las otras, al tener asociada una categoría ambiente en la que aparecen naturalmente diversas construcciones matemáticas.

Las mónadas de Haskell y las bibliotecas gráficas permiten escribir programas vistosos y útiles, más cercanos a los que produce la industria. Excepto en lo que respecta a la interacción con los periféricos, el código que puede comprender en

sus detalles más íntimos y produce al ejecutarlo un resultado que satisface al usuario genérico.

En resumen, se trata de mostrar la programación como una actividad matemática y de usarla como hilo conductor y como ejemplo concreto donde comprender las estructuras matemáticas que tiene implícitas.

2. Bolillas

1. Presentación del paradigma de la programación funcional. Expresiones, reducción, valores. Tipos de datos básicos `Int`, `Integer`, `Float`, `Double`, `Char`, `Bool`. Ejemplos. Definiciones recursivas en el tipo `Int`.

2. Funciones (parciales) calculables, tipado, igualdad extensional e igualdad intensional. Polimorfismo. Composición de funciones calculables. Categoría de las funciones calculables. Tipos producto y coproducto, objeto terminal, *curryficación* y exponenciales. Propiedades universales.

3. Clases en Haskell. Clases `Eq`, `Ord`, `Num` y `Enum`. La clase `Show`. Clases en las que están los tipos básicos. Ejemplo: Declaración del tipo: `Bool`. Instanciaciones de `Bool` en las clases `Eq`, `Ord` y `Show`. Instanciaciones hereditarias de clases.

4. Primer ejemplo de un tipo recursivo: el tipo `Nat`. Naturales infinitos, demostraciones por inducción en `Nat`. Recursión en `Nat`. La función `fold`. Orden de Scott en `Nat`. Números de Church.

5. Listas. Ejemplo: `String`. Listas infinitas. Orden de Scott en listas. Inducción y recursión en listas. Comprensión en listas. Funciones `map` y `filter`. Funciones `foldl` y `foldr`. Fusión. Estructuras cíclicas. Interacción basada en *streams*. Listas de Church.

6. Árboles. Ejemplo: árboles binarios finitos e infinitos. Orden de Scott en árboles binarios. Inducción y recursión en árboles binarios. Altura, tamaño, ancho. Funciones de árboles en listas y de listas en árboles. Las funciones `map` y `fold`. Fusión. Otros tipos de árboles: árboles binarios de búsqueda, árboles de aridades arbitrarias. Árboles de Church.

7. Mónadas. Primer ejemplo: `IO`. Interacción. Notación `do`. Mónadas de excepción, mónadas de estado. Biblioteca gráfica `Wxhaskell`. Leyes de las mónadas. Transformaciones de mónadas. Las mónadas como noción categórica. Ejemplos de mónadas en otras categorías.

3. Evaluación

Las instancias previstas de evaluación son las siguientes:

1. Un examen parcial a mitad de semestre.
2. Un proyecto final de programación, obligatorio.
3. Un examen oral, final y obligatorio.

Todas las evaluaciones se puntuarán sobre una escala lineal de 20 puntos.

La ganancia del curso requiere un mínimo del 25 % del puntaje del examen parcial y un 25 % del puntaje del proyecto.

La puntuación final se define como la suma ponderada $F = O \times 0,4 + E \times 0,6$ donde O es el puntaje del obligatorio y E el del examen final. La nota 3 se fija en el 50 % del máximo (10 puntos en 20).

4. Bibliografía

- *Introduction to Funcional Programming using Haskell.* Richard Bird. Prentice Hall Series in Computer Science.
- *Categories for the Working Mathematician.* Saunders Mac Lane. Springer. Graduate Texts in Mathematics.
- *Domains and Lambda-calculi* Roberto Amadio & Pierre-Louis Curien. Association for Symbolic Logic.