Programación Declarativa

ETSI Informática

Dpto. Lenguajes y Ciencias de la Computación

Ingeniería Técnica en Informática (Sistemas A)

Curso 2007–2008

Profesor

Pepe Gallardo
`http://www.lcc.uma.es/~pepeg`
`pepeg@lcc.uma.es`
Despacho: 3.2.50

Horario

Aula: 3.0.2

- Jueves 8:50–10:40
- Miércoles 10:50–12:40

Sesiones de laboratorio por determinar

Una asignatura, dos lenguajes

- Estudiaremos dos estilos de programación declarativa:
  - Programación funcional (en Haskell)

    $$\text{cómputo} = \text{reducción de expresiones}$$

  - Programación lógica (en Prolog)

    $$\text{cómputo} = \text{deducción controlada}$$

- Ambos estilos difieren **radicalmente** de la programación imperativa (C, C++, Java,...)
- Ambos estilos difieren entre sí
- Por lo tanto, la asignatura tiene dos partes (Haskell y Prolog) que se califican de forma independiente

Normas de evaluación I

Examen parcial voluntario / Convocatoria de repetidores (16 diciembre 2008, 16:30)

- Examen parcial voluntario (sólo Haskell)
  - Elimina materia si se obtiene una calificación **igual o superior a 5**, conservándose esta calificación **hasta septiembre de 2009**
- Convocatoria de repetidores (Haskell y Prolog)
  - Si alguna parte tiene una calificación **inferior a 3**, la calificación es **suspenso**
  - Si alguna parte tiene una calificación **igual o superior a 5**, se conserva dicha calificación **hasta septiembre de 2009**
  - Calificación final: media aritmética de las calificaciones de Haskell y Prolog

Examen final de febrero (16 febrero 2009, 9:30)

- Sólo una parte (Haskell o Prolog)
  - La parte **pendiente** tiene que superarse con una calificación **igual o superior a 5**
  - Calificación final: media aritmética de las calificaciones de Haskell y Prolog
- Ambas partes (Haskell y Prolog)
  - Si alguna parte tiene una calificación **inferior a 3**, la calificación es **suspenso**
  - Si alguna parte tiene una calificación **igual o superior a 5**, se conserva dicha calificación **hasta septiembre de 2009**
  - Calificación final: media aritmética de las calificaciones de Haskell y Prolog

Normas de evaluación II

Examen final de septiembre (1 septiembre 2009, 16:30)

- Sólo una parte (Haskell o Prolog)
  - La parte **pendiente** tiene que superarse con una calificación **igual o superior a 5**
  - Calificación final: media aritmética de las calificaciones de Haskell y Prolog
- Ambas partes (Haskell y Prolog)
  - Si alguna parte tiene una calificación **inferior a 3**, la calificación es **suspenso**
  - Calificación final: media aritmética de las calificaciones de Haskell y Prolog

Programación Funcional

Temario

1. Introducción y semántica operacional
2. Tipos predefinidos
3. Patrones y Definiciones de Funciones
4. Funciones de orden superior
5. Polimorfismo
6. Definiciones de tipos
7. El sistema de clases
8. Listas
9. Árboles

Bibliografía Básica

Razonando con Haskell. Un curso sobre programacion funcional

Blas Ruiz, Francisco Gutiérrez, Pablo Guerrero y José Gallardo. Thomson, 2004.

`http://www.lcc.uma.es/RazonandoConHaskell`

Bibliografía Complementaria

Programming in Haskell
Graham Hutton. Addison-Wesley, 2007.

Introduction to Functional Programming using Haskell
Richard Bird. Prentice Hall, 1998.

The Haskell School of Expression. Learning Functional Programming through Multimedia
Paul Hudak. Cambridge University Press, 2000.

Haskell. The Craft of Functional Programming
Simon Thompson. Addison-Wesley, 1999.

Software

- Hugs (Haskell User Gofer System)
  - intérprete «ligero», entorno para Windows (Winhugs)
- GHC (Glasgow Haskell Compiler)
  - compilador e intérprete, mejores mensajes de error, rendimiento y bibliotecas

Ambos sistemas están disponibles en los laboratorios. En casa basta instalar Hugs.

Programando con estados

Ejemplo: Dada una secuencia de números (p.ej. del 1 al 15), sumar los cuadrados de los números pares.

```
public class Ejemplo1 {
   static public void main(String [] args) {
      int[] numbers = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 };
      int sum = 0;

      for(int i=0; i<numbers.length; i++) {
         if (numbers[i] % 2 == 0) {
            sum += numbers[i]*numbers[i];
         }
      }
      System.out.println("Sum: ", sum);
   }
}
```

- Hay asignaciones (`i`, `sum`)
- Hay bucles
- Hay estado (`i`, `sum`)

> Can Programming Be Liberated from the von Neumann Style?
>
> *John Backus*

Programando sin estados

```
public class Ejemplo2 {

    static public void main(String [] args) {
        int[] numbers = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 };
        System.out.println("Sum: " +  suma(numbers, 0, 0));
    }

    static private int suma(int[] ns, int i, int ac) {
        if (i == ns.length) {
                return ac;
        } else if (ns[i] % 2 == 0) {
                return suma(ns, i+1, ac + ns[i]*ns[i]);
        } else {
                return suma(ns, i+1, ac);
        }
    }
}
```

Programando con funciones

Can Programming Be Liberated from the von Neumann Style?
A **Functional** Style and its Algebra of Programs

John Backus

```
suma :: [Int] -> Int
suma xs = suma' xs 0
   where suma' [] ac = ac
         suma' (x:xs) ac
             | x `mod` 2 == 0 = suma' xs (ac+x*x)
             | otherwise      = suma' xs  ac
```

- No hay asignaciones
- No hay bucles
- No hay estado

Programando bien con funciones

- un verdadero programador funcional escribiría:

```
suma xs =
 foldr (\x s -> x + s) 0 . map (\x -> x*x) . filter (\x -> x `mod` 2 == 0) $ xs
```

- o mejor aún:

```
suma = foldr (+) 0 . map (^2) . filter even
```

- o incluso:

```
suma xs = sum [ x*x | x <- xs, even x ]
```

> **Elegance is not optional**. What do I mean by that? I mean that in Prolog (Haskell), as in most halfway decent programming languages, there is no tension between writing a beautiful program and writing an efficient one. **If your Prolog (Haskell) code is ugly**, the chances are that you either don't understand your problem or **you don't understand your programming language**, and in neither case does your code stand much chance of being efficient. In order to ensure that your program is efficient, you need to know what it is doing, and if your code is ugly, you will find it hard to analyse.
>
> *Richard A. O'Keefe*

El mismo programa en C# 3.0

```
static void Main() {
  int[] numbers = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 };

  int sum;

  sum= numbers.Filter(x => (x % 2) == 0).Map(x => x * x).Reduce(0, (x,s) => x + s);

  Console.WriteLine("Sum: {0}", sum);
}
```

¿Suena familiar?

Evolución de los lenguajes de programación

Características de los lenguajes declarativos que están apareciendo en otros lenguajes:

- Recolección de basura (Java, C#, Scala, Python, Perl, Ruby, Javascript)
- Orden superior (Java, C#, Scala, Python, Perl, Ruby, Javascript)
- Genericidad/Polimorfismo (Java, C#, Scala)
- Comprensiones (C#, Scala, Python, Perl 6, Javascript)
- Clases de tipos (interfaces en Java y C#, «conceptos» en C++)

Puede que no uses Haskell ni Prolog en tu vida profesional, pero seguro que utilizarás características como éstas en el próximo «gran lenguaje» (si lo usas bien, recuerda **elegance is not optional**)