Teaching Experiences In A Functional-first Multi-paradigm Programming Course

Francisco Saiz
School of Computer Science and Engineering
Autonomous University of Madrid

TFPIE 2015

June 2th

Introduction

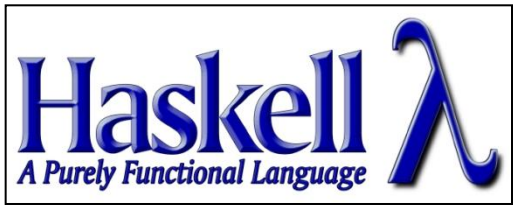
- Functional Programming (**FP**) is more and more **embedded** into mainstream **imperative** languages
  - Microsoft **.NET Framework**
    - C#, F#
  - Oracle **Java**
    - Java 8 (functional interfaces, streams)

Further/Advanced Programming

- **Elective** course (final year)
  - *Graduate on Computer Science Engineering*
    - 240 ECTS Bachelor Degree (4 years)
    - School of Computer Science and Engineering
      - Autonomous University of Madrid
- Delivered for the past **three years**
  - Average of 20 students
    - Greater appeal from other elective courses (*Mobile Interaction, Video Games*, etc.), with 70 student at most

Overview

- The course consists of two parts
  - **Haskell** (50%)
  - **.NET** (50%)
    - C#
    - F#
- F# at the end of the course
  - No practical assignments, and serving as a summing up part



IDEs

- Haskell
  - ***eclipsefp*** (eclipse plug-in)
    - Open source
    - Syntax highlighting, code completion, error messages and suggestions, integration with GHC, quick fixes, etc.
- .NET
  - ***Microsoft Visual Studio***
    - Proprietary
    - Much more robust than *eclipsefp*

Haskell

- It makes easier for students to learn **FP concepts**
  - lazy evaluation, currying, immutability, monads, etc.
- Similarity to **Math language**
  - A more declarative programming style
- **Students' background**
  - Proficient in Java and C
  - Programs as sequences of instructions based on imperative patterns
    - Openly modifying states defined by variables, neglecting both referential transparency and side-effect issues
- **Higher-order functions**, as useful abstractions
- **Monadic operations**
  - Bridge between FP and imperative programming
  - IO actions and state handling as meaningful examples

Choice of .NET languages

- **FP is broadly used** in C#/F#
  - Microsoft research teams' high interest in FP
- Extensive use of .NET in the **industry**
  - Increasing appeal for students
- **Multi-paradigm** setting
  - Students to enhance OOP knowledge through C#
- C# improves many **Java features**
  - Harnessing C++-like language constructs
    - While using managed memory
  - More powerful and declarative

C# language, I

- **Imperative**-first (OOP)
- **Comparison to Java**
  - Properties, indexers, value/reference types, boxing, generics, operator overloading, variance, etc
- **Parametric polymorphism**
  - Revisiting Java generics concepts
  - Haskell polymorphism versus C# generics
- **Functions**
  - Generic delegates
  - No currying (operator ".")
  - Parameters for higher-order methods
    - Extension methods

Functions

```
type Func a b = a->b                                (Haskell)
delegate B Func<A,B> (A)                            (C#)
interface Function<A,B> {B apply(A)}                (Java)
type Func<'a,'b> = 'a->'b                           (F#)
```

C# language, II

- **Collections** as inputs for higher-order operations
  - C# `IEnumerable` interfaces, similar to:
    - Haskell list comprehensions
    - F# lists or sequences
    - Java streams
- **Values of C# delegates**
  - C# lambda expressions
  - Methods
- Concepts **Function/Closure/Method**
- C# delegates / Java functional interfaces

C# higher-order operations, I

- **User-defined** generic extension methods, e.g. for composition:

```
static Func<T1,T3> Compose<T1,T2,T3>
       (this Func<T2,T3> f,
        Func<T1,T2> g)
  {return x => f(g(x));}
```

C# higher-order operations, II

- **Predefined** generic extension methods, e.g. *map/reduce* operations

```
public static IEnumerable<T2>
  Select<T1, T2>                    // map
     (this IEnumerable<T1>,
      Func<T1, T2>)
public static T2
  Aggregate<T1, T2>                 // reduce
     (this IEnumerable<T1>,
      T2, Func<T2, T1, T2>)
```

C# language, III

- ***Iterators***
  - Declarative expressions and laziness
  - Marking return values through `yield` construct
    - Relation with Haskell list comprehensions
- **LINQ**
  - **Declarative** language in.NET Framework
    - Reminds SQL queries and Haskell list comprehensions
  - Featuring declarative, **functional** coding style

Coding the infinite primes

```
[n | n<-[1..], isPrime n]                    (Haskell)
IEnumerable<int> Primes() {                  (C#, yield)
  for (int num = 2; ; num++)
      if (IsPrime(num)) yield return num;
}
IEnumerable<int> Primes() {                  (C#, LINQ)
  IEnumerable<int> query =
      from num in IntegersFrom(2)
      where IsPrime(num)
      select num;
  return query;
}
```

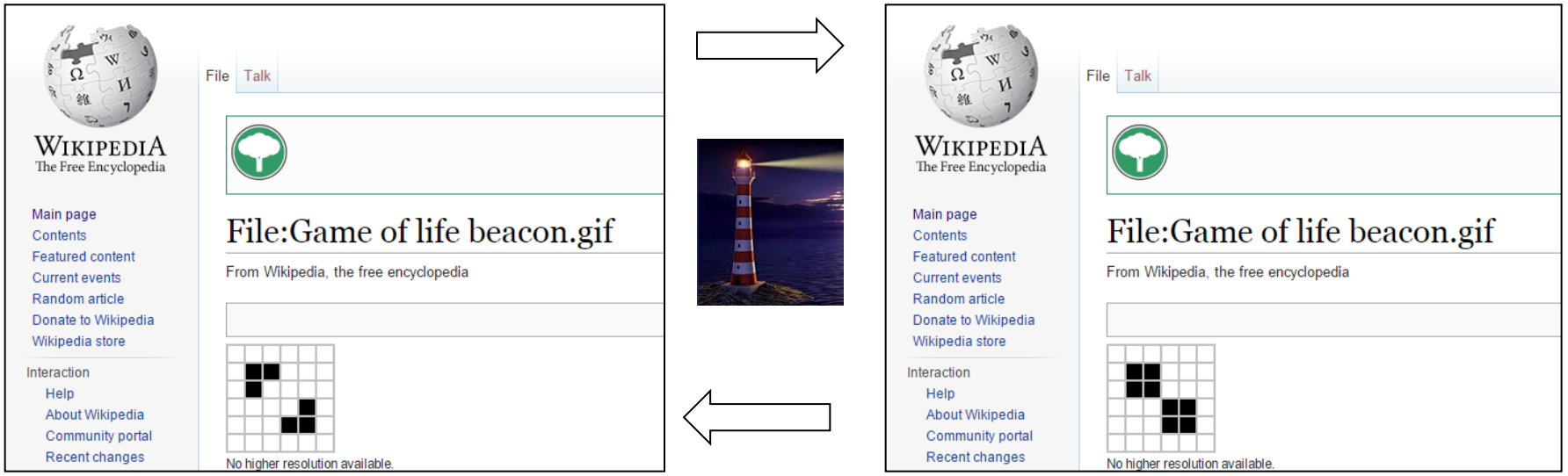
F# overview

- Open source functional-first **multi-paradigm**
- **.NET classes** available both to C# and F#
- Haskell algebraic data types as **F# discriminated unions**
- **Pattern matching** for input data
- Functions (**F# values**) are curried and immutable-first
  - Imperative instructions also possible on mutable values, e.g. ***while loops***, instead of recursive functions, as in Haskell
- **Collections** (lists, arrays or sequences)
- LINQ and computation expressions

Comparing *Haskell/F#*

	HASKELL	F#
Type parameter	`a, b, ...`	Usual → `'a, 'b, ...` Unusual → `^a, ^b, ...`
Lists	Value → `[2,3]` Type → `[Int]` Operators → `(:) (++)`	Value → `[2;3]` Type → `int list` Operators → `(::) (@)`
Tuples	Value → `(2,3,4)` Type → `(Int,Int,Int)`	Value → `2,3,4` Type → `int * int * int`
List comprehensions	`[ … | …<-…, …]`	`[for … do … yield …]`
Lambda expressions	`\ x y -> x+y`	`fun x y -> x+y`

Conway's Game of Life




```
type Pos = (Int,Int)
type Board = [Pos]
nextgen, survivors, births :: Board -> Board
nextgen b = survivors b ++ births b
liveneighbs :: Board -> Pos -> Int
```

Coding around the “Game of Life”

```
liveneighbs b =                                          (Haskell)
   length . filter (isAlive b) . neighbs
let liveneighbs b =                                      (F#)
  List.length << List.filter (isAlive b) << neighbs

survivors b =                                            (Haskell)
  [p | p <- b, elem (liveneighbs b p) [2,3]]
let survivors b =                                        (F#)
  [for p in b do
     if (elem (liveneighbs b p) [2;3]) then yield p]
```

Course management, I

- **15 weeks**
- As a **weekly** basis
  - 3 hours (theory classes)
  - 2 hours (practical assignments)
- 6 **practical assignments**
  - 3 on Haskell, and 3 on C#
  - 30% for each student's overall mark
- **Assignments 1-2** to introduce Haskell concepts
- **Assignment 3** about a Haskell ***REFERENCE EXAMPLE***, benefitting from it, which is to be extended with new functionality, e.g.,
  - The ***Game of Life*** code by G. Hutton, with new survival rules, an enhanced user interface and error handling

Course management, II

- **Assignments 4-5**
  - implement in C# the reference example, plus a Windows Forms graphical user interface
    - Using just **OOP imperative** programming
- **Assignment 6**
  - Using **FP in C#**, according to the Haskell code developed on assignment 3, appreciating FP assets:
    - higher-order operations
    - lambda expressions
    - yield keyword and LINQ statements
- No practical assignments on F#
  - An F# version about the Haskell reference example is discussed in a classroom session

Guiding exercises

- One hour a week
- With the teacher's support, the students solve **guiding exercises**
  - At the following class, students are provided with the solved exercises, which are discussed in detail
- Method much **appreciated by students**
  - Interactive learning
  - Direct application of theoretical concepts
  - Preventing students from getting lost

Final thoughts

- While many FP subjects are left during the course, students encounter a **different way of programming**
- They become aware of its **extensiveness** and **usefulness** in modern languages
- They regard Haskell as a **more academic** language, not having an IDE as robust as .NET languages
- **Functional algebra** in programming turns is definitely a great achievement for students
- Relying on **Math language**, programming turns out to be a **rewarding** abstraction

Comments and Questions?

- **Acknowledgments**
  - Work funded by the ***Comunidad Autonoma de Madrid*, project e-Madrid** (S2013/ICE-2715)
  - Thanks to **Alejandro Serrano Mena**, author of the **book *Beginning Haskell***, *a Project-Based Approach* (2014) and former Autonomous University of Madrid student, for encouraging me for this submission