Teoría de los Lenguajes de Programación
PEC1 - 2021

Centro Asociado de la UNED en Bizkaia

Martín Romera Sobrado
Bilbao

17 de mayo de 2021

Índice

1. Pregunta 1 **2**

2. Pregunta 2 **2**

3. Pregunta 3 **3**

1. Pregunta 1

*Supongamos una implementación de la práctica en un lenguaje no declarativo (como **Java, Pascal, C**. Comente qué ventahas y qué descentajas tendría frente a la implementación en **Haskell**. Relacione estas ventajas desde el punto de vista de la eficiencia con respecto a la programación y a la ejecución. ¿Cuál sería el principal punto a favor con respecto a la implenetación en los lenguajes no declarativos? ¿Y el de la implementación en **Haskell**?)*

La principal diferencia entre ambas implementaciones es que ambas son más (o menos) eficientes respecto a los aspectos de **programación**, y **ejecución**.

La programación declarativa nos ofrece una forma muy visual y sencilla de programar en general, lo que hace que la **programación sea más eficiente**, ya que la forma de definir la lógica de un problema de este calibre es más cercana a la forma en la que lo plantearía una persoa mediante el paradigma funcional que nos ofrece *Haskell*. Por supuesto esto es relativo al programador, si no está acostumbrado a la programación declarativa esta eficiencia se acerca a la nulidad. Por otra parte la arquitecutra de un computador convencional no está preparada para ejecutar "programas funcionales" de forma eficiente, lo que hace que **ejecución sea lenta**.

Por otra parte los lenguajes de programación no declarativa pertenecientes a un paradigma imperativo más fiel a la arquitectura de un computador convencional nos otorga **mayor eficiencia en la ejecución**, sobre todo con lenguajes más cercanos a la maquina como podría ser *C*, a cambio de **dificultar la programación** de un problema complejo como puede ser este que se plantea en la práctica.

2. Pregunta 2

*Indique, con sus palabras, qué permite gestionar el predicado predefinido no lógico, corte (**!**), en **Prolog**. ¿Cómo se realizaría este efecto en **Java**? Justifique su respuesta.*

El perdicado corte (`!`) de *Prolog* sirve para controlar el flujo de ejecución de una computación lógica. Este permite regular la eficiencia de ejecución de un algoritmo o incluso evitar la entrada en en bucles infinitos. Lo que hace es **podar todas las ramas del nodo padre del que encuentra el predicado "`!`"**. El análogo más cercano en *Java* y otro muchos lenguajes de programación imperativa es el `break`. Podríamos estar explorando la ramificación de un nodo en un árbol mediante un bucle `while` que recorra todos los hijos de ese nodo, y que para comprobar si el nodo que estamos comprobando es el que queremos seguir explorando podemos poner un condicional `if` con una sentencia lógica y a continuación un `break` para salir del bucle. Esto es

el analogo en *Prolog* a introducir una regla (que sería la equivalente a la sentencia lógica del condicional) seguido de un predicado corte. La sentencia lógica del condicional también podría ser la que defina un patrón en un predicado de *Prolog*.

3. Pregunta 3

*Para los tipos de datos del problema definidos en **Haskell**, indique qué clases de constructores de tipos se han utilizado en cada caso (ver capítulo 5 del libro de la asignatura).*

Para

```
type Zone = Int
```

, se define el tipo como un **sinónimo** de un tipo **atómico**, en este caso el tipo `Int` de la propia implementación de *Haskell*.

Para

```
type Row        = [Zone]
type Map        = [Row]
type Solution   = [Color]
```

, se definen como **sinónimos** de tipos **estructurados** de clase **Array**.

Para

```
type Adjacency = (Zone,[Zone])
type Node      = ([Adjacency],Zone,Zone,Solution)
```

, se definen como **sinónimos** de tipos **estructurados** de clase **Tupla**.

Para

```
data Color = Red | Green | Blue | Yellow
    deriving (Enum, Eq, Show)
```

, se define como un tipo **enumerado** derivando las propiedades de algunas clases de la propia implementación de *Haskell*.