Universidad de La Laguna
Escuela Superior de Ingeniería y Tecnología
Sección de Ingeniería Informática

# Construcción de un Compilador y un Intérprete de Scheme Usando Haskell

*Building a Compiler and Interpreter for Scheme Using Haskell*

Francisco Nebrera Perdomo

La Laguna, 16 de abril de 2015

D. **Casiano Rodríguez León**, con N.I.F. 42.020.072-S profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

**C E R T I F I C A**

Que la presente memoria titulada:

"*Construcción de un Compilador y un Intérprete de Scheme Usando Haskell.*"

ha sido realizada bajo su dirección por D. **Francisco Nebrera Perdomo**, con N.I.F. 79.064.507-Y.

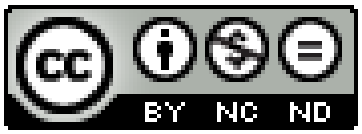
Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 16 de abril de 2015

# Agradecimientos

Casiano Rodríguez León
Jorge Riera Ledesma
Luz Marina Moreno de Antonio
Francisco de Sande González
Francisco Carmelo Almeida Rodríguez
Blas C. Ruiz
F. Gutiérrez
P. Guerrero
E. Gallardo
Daniel Díaz Casanueva

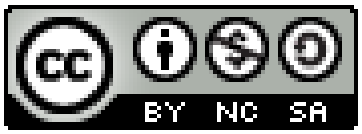
# Licencia

* Si NO quiere permitir que se compartan las adaptaciones de tu obra y NO quieres permitir usos comerciales de tu obra indica:



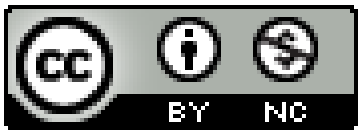



* Si quiere permitir que se compartan las adaptaciones de tu obra mientras se comparta de la misma manera y NO quieres permitir usos comerciales de tu obra indica:

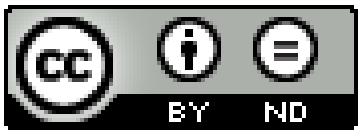





* Si quiere permitir que se compartan las adaptaciones de tu obra y NO quieres permitir usos comerciales de tu obra indica:

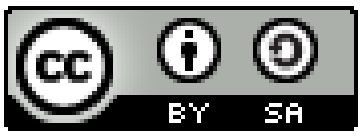


*Si NO quiere permitir que se compartan las adaptaciones de tu obra y quieres permitir usos comerciales de tu obra indica:






* Si quiere permitir que se compartan las adaptaciones de tu obra mientras se comparta de la misma manera y quieres permitir usos comerciales de tu obra (licencia de Cultura Libre) indica:






* Si quiere permitir que se compartan las adaptaciones de tu obra y quieres permitir usos comerciales de tu obra (licencia de Cultura Libre) indica:

## Resumen

*El objetivo de este trabajo ha sido crear un compilador e intérprete del lenguaje Scheme, usando Haskell. Para ello se ha hecho uso de un parser monádico como es Parsec.*

*La competencia [E6], que figura en la guía docente, indica que en la memoria del trabajo se ha de incluir: antecedentes, problemática o estado del arte, objetivos, fases y desarrollo del proyecto, conclusiones, y líneas futuras.*

*Se ha incluido el apartado de 'Licencia' con todas las posibles licencias abiertas (Creative Commons). En el caso en que se decida hacer público el contenido de la memoria, habrá que elegir una de ellas (y borrar las demás). La decisión de hacer pública o no la memoria se indica en el momento de subir la memoria a la Sede Electrónica de la ULL, paso necesario en el proceso de presentación del TFG.*

*El documento de memoria debe tener un máximo de 50 páginas.*

*No se deben dejar páginas en blanco al comenzar un capítulo, ya que el documento no está pensado para se impreso sino visionado con un lector de PDFs.*

*También es recomendable márgenes pequeños ya que, al firmar digitalmente por la Sede, se coloca un marco alrededor del texto original.*

*El tipo de letra base ha de ser de 14ptos.*



**Palabras clave:** Palabra reservada1, Palabra reservada2, ...

## Abstract

Here should be the abstract in a foreing language...



***Keywords:*** *Keyword1, Keyword2, Keyword2, ...*

# Índice general

# Índice de figuras

# Índice de tablas

# Capítulo 1

# Introducción

## 1.1. Sección Uno

Todo empezó con un hilo en el foro de internet "forocoches.com", en él hablaban de que la programación funcional iba a tener cada día más relevancia porque cuenta con ventajas de las cuales la imperativa carece.

A raíz de ello, me interesé por este paradigma y empecé a (e incluso terminé de) leer numerosos libros sobre el lenguaje y la programación funcional en general, y a crear pequeños programas en Haskell. Haskell es muy interesante debido a que es el lenguaje con mayor nivel de abstracción en el que he programado hasta hoy.

Haskell es idóneo para crear lenguajes de dominio específico. En otras palabras, antes de escribir un compilador se captura el lenguaje a compilar (el lenguaje fuente) en un tipo. Las expresiones de ese tipo representarán términos en el lenguaje fuente y normalmente son bastante similares al mismo, a pesar de ser, realmente, tipos de Haskell.

Luego se representa el lenguaje objetivo como otro tipo más. Finalmente, el compilador es realmente una función del tipo fuente al tipo objetivo y las traducciones son fáciles de escribir y leer. Las optimiza-

ciones también son funciones como cualquier otra (ya que realmente en Haskell todo es una función, y además, currificada) que mapean del dominio del lenguaje fuente al codominio del lenguaje objetivo.

Por ello los lenguajes funcionales con sintaxis ligera y un fuerte sistema de tipos se consideran muy adecuados para crear compiladores y muchas otras cosas cuya finalidad es la traducción.

Además, Haskell cuenta con mecanismos de abstracción muy fuertes que permiten escribir códigos escuetos que se comportan muy bien, como por ejemplo:

- reconocimiento de patrones
- tipos de datos algebraicos (generalizados o no)
- lambdas (y por ello, mónadas)
- plegados de listas

A continuación se describen los mencionados constructos; el reconocimiento de patrones y los tipos de datos algebraicos se incluyen en la misma sección porque son conceptos que están muy relacionados.

## 1.2. Reconocimiento de patrones y tipos de datos algebraicos

## 1.3. Lambdas

Bla, bla, bla

## 1.4. Plegados de listas

Pondremos un ejemplo real codificado por mí, un DFA hecho mediante un plegado de listas por la izquierda.

```
probarDFA :: DFA -> [Char] -> Bool
probarDFA (DFA i a t) = a . foldl' t i
```

Un DFA se podría implementar en programación imperativa con un bucle for que fuera sobreescribiendo el estado en cada iteración, haciendo un lookup en su tabla de estados dependiendo de su estado actual y el símbolo leído.
Esto, en Haskell, se puede hacer usando la función **foldl** (aunque aquí por temas de rendimiento y uso de memoria se ha optado por **foldl'**).
Se trata de empezar con un acumulador (en este caso, el estado inicial), y nos vamos moviendo por la lista (cadena de entrada) de izquierda a derecha, haciendo un lookup...

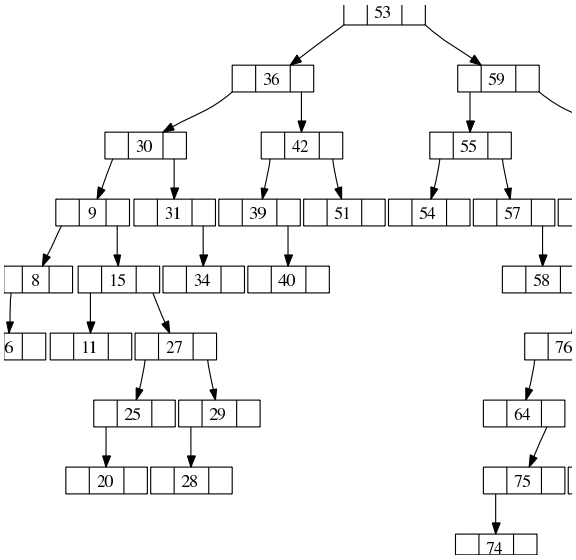



Figura 1.1: Ejemplo

# Capítulo 2

# Título del Capítulo Dos

En el capítulo anterior se ha realizado una introducción a la programación funcional con Haskell, y ahora se describirá un módulo muy útil en la creación del intérprete, Parsec.
Parsec es un módulo de Haskell, un conjunto de funciones exportables que suelen tener una finalidad común y se pueden importar en otros programas. En nuestro intérprete hemos importado Parsec, pues es el módulo utilizado en el tutorial que seguimos, Write Yourself a Scheme in 48 hours.
Parsec se diseñó desde cero como una librería de parsers con capacidades industriales. Es simple, segura, está bien documentada, provee de buenos mensajes de error y es rápida. Se define como un transformador de mónadas que puede ser apilado sobre mónadas arbitrarias, y también es paramétrico en el tipo de flujo de entrada. La documentación de la versión usada en el presente Trabajo Fin de Grado se puede consultar online en https://hackage.haskell.org/package/parsec-3.1.9

## 2.1. Primer apartado de otro capitulo

# Capítulo 3

# Título del Capítulo Tres

Bla, Bla, Bla, .....

## 3.1. Primer apartado de este capitulo

## 3.2. Segundo apartado de este capitulo

## 3.3. Tercer apartado de este capitulo

# Capítulo 4

# Título del Capítulo Cuatro

En el capitulo  1 se describio bla, bla, bla.....

# Capítulo 5

# Conclusiones y trabajos futuros

Este capítulo es obligatorio. Toda memoria de Trabajo de Fin de Grado debe incluir unas conclusiones y unas líneas de trabajo futuro

# Capítulo 6

# Summary and Conclusions

This chapter is compulsory. The memory should include an extended summary and conclusions in english.

## 6.1. First Section

# Capítulo 7

# Presupuesto

Este capítulo es obligatorio. Toda memoria de Trabajo de Fin de Grado debe incluir un presupuesto.

## 7.1. Sección Uno

| **Tipos** | **Descripcion** |
|---|---|
| AAAA | BBBB |
| CCCC | DDDD |
| EEEE | FFFF |
| GGGG | HHHH |

Tabla 7.1: Tabla resumen de los Tipos

# Apéndice A

# Título del Apéndice 1

## A.1. Algoritmo XXX

```
*****************************************************************************
*
* Fichero .h
*
*****************************************************************************
*
* AUTORES
*
*
* FECHA
*
*
* DESCRIPCION
*
*
*****************************************************************************/
```

## A.2. Algoritmo YYY

```
/*****************************************************************************
 *
 * Fichero .h
 *
```

```
*******************************************************************************
*
* AUTORES
*
* FECHA
*
* DESCRIPCION
*
*
*******************************************************************************/
```

# Apéndice B

# Título del Apéndice 2

## B.1. Otro apendice: Seccion 1

Texto

## B.2. Otro apendice: Seccion 2

Texto

# Bibliografía


[1] D. H. Bailey and P. Swarztrauber. The fractional Fourier transform and applications. *SIAM Rev.*, 33(3):389–404, 1991.

[2] Miran Lipovaca. *Learn You a Haskell for Great Good!: A Beginner's Guide*. No Starch Press, 2011.

[3] Blas C. Ruiz. *Razonando con Haskell. Un curso sobre programación funcional*. Thomson, 2004.

[4] David D. Spivak. *Category Theory for the Sciences*. MIT Press, 2014.