Caso de estudio

Desarrollo de un intérprete de un subconjunto de un lenguaje imperativo utilizando el lenguaje Haskell

Jaime Cepeda Villamayor

Andrés Montoro Montarroso

Objetivos

- Desarrollar un intérprete de un lenguaje de programación imperativo
- Utilizar Haskell para el desarrollo de este
- Aplicar los conceptos de los procesadores de lenguaje al lenguaje de programación Haskell para el desarrollo del intérprete

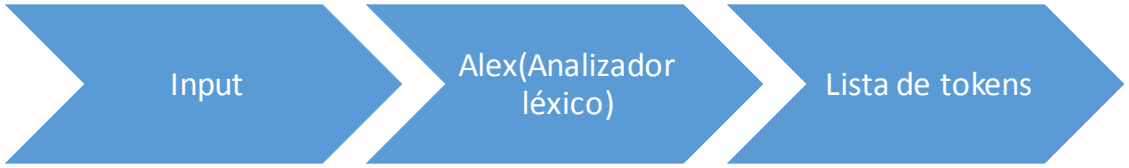
¿Que lenguaje vamos a implementar?

- El lenguaje a implementar es un subconjunto de multiples lenguajes.
- Ese subconjunto se ha declarado en formato BNF.

```
exp = assignation |
    if_expression |
print_expression |
while_expression |
    operation ;
```

Desarrollo (Analizador léxico)

- El analizador léxico es el encargado de leer el código fuente y extraer los tokens que se encuentren en esto
- Se ha utilizado la herramienta Alex, la cual dada una serie de reglas, nos permite generar una lista de tokens
- Ejemplo de código



Desarrollo (Analizador sintáctico)

- El analizador sintáctico es aquel que se encarga de convertir la lista de tokens en un Arbol Sintactico Abstracto (AST), generando la estructura completa del programa.
- Nuestro AST será una lista con cada una de las instrucciones del código fuente, distinguibles entre unas y otras.
- Se ha utilizado la herramienta Happy, que permite generar un AST a partir de una especificación BNF, y las estructuras de datos que le indiques (Ejemplo de código).

Desarrollo (Intérprete) (I)

- Para el intérprete se han usado funciones de Haskell, recogidas en un módulo del programa
- Se han utilizado técnicas propias de los lenguajes funcionales como Haskell, tales como Pattern Matching, Inmutabilidad de los datos, recursividad, map/reduce, etc.
- El módulo recoge el AST generado por Happy, y devuelve una lista de las variables y el valor asociado a ellas al finalizar el programa
- También permite imprimir valores (enteros) por pantalla mientras se ejecuta el programa.

Desarrollo (Intérprete) (II)

- Para poder imprimir por pantalla se ha usado la mónada IO, por lo que gracias a esta, se han podido imprimir por pantalla los distintos valores que se querían imprimir
- Ejemplo de código

Uniendo las tres partes

- Gracias a Haskell y al uso de las herramientas Alex y Happy, se ha podido encapsular correctamente cada una de las tres partes en 3 funciones.
- El código resultante de la parte principal es este:

```
main = do
content <- readFile "app/example"
let tokens = scanTokens content
let structure = parseTokenss tokens
eval structure emptyDataStore
```

Ejecución del programa

- Primero, hay que compilar el proyecto:
  - Stack build
- Segundo, hay que ejecutarlo
  - Stack exec functional-compiler-exe
- Ejemplo de código

Conclusiones

- Haskell, junto con las herramientas Alex y Happy, nos permite realizar de forma sencilla un intérprete de un lenguaje imperativo.
- Las capacidades de Haskell nos permiten hacer mucha funcionalidad con poco código escrito, y además a alto nivel.
- Gracias a la inmutabilidad de las estructuras de haskell, no tenemos comportamientos indeseados a la hora de ejecutar el programa, por lo que la fiabilidad del programa aumenta
- Esto es aplicable para realizar cualquier tipo de parser, ya sea de CSV, JSON, u otro lenguaje de programación

Muchas gracias por venir

- El código se encuentra disponible para uso en
  - https://github.com/JCepedaVillamayor/functional-compiler

Jaime Cepeda Villamayor
Andrés Montoro Montarroso