Grupo: 10

Fecha de entrega: 7 de junio de 2024

Curso : Brasburg, Cano, Raik

Integrantes:

Padrón	**Apellido y nombre**
110845	Cardoso Landaburu Juan Gabriel
110598	Almanza Lucia
110736	López Facundo Gabriel
111019	Fernández Ignacio Agustin

Introducción

El presente informe tiene como objetivo presentar las hipótesis y las decisiones tomadas en el trabajo práctico número dos de la asignatura Paradigmas de la Programación. El proyecto consiste en la implementación de un intérprete de Cálculo Lambda no tipado, cuyo desarrollo se debe llevar a cabo aplicando los conceptos vistos en las clases de programación funcional. La totalidad del proyecto fue realizada en Scala

Hipótesis

A partir de las consignas proporcionadas hemos formulado las siguientes hipótesis para la implementación del programa:

- Los paréntesis sólo serán para aplicaciones.
- El input del usuario siempre será correcto.
- El programa termina ejecucion para el input “exit”
- Reconocemos que ciertas expresiones lambda pueden causar un `StackOverflowError` debido a la recursión infinita. Lo cual es lo esperado por la implementación de nuestro programa.

Diseño:

Para implementar nuestro intérprete, dividimos el proyecto principalmente en tres componentes:

Lexer: Recibe como parámetro un String que representa una expresión lambda. Implementamos un Trait llamado `Token` que define los distintos símbolos característicos del cálculo lambda, permitiendo convertir el input en una expresión lambda.

Parser: Posteriormente a que el lexer transforme el string en un ***List<Token>***, el Parser se encarga de recibir esta lista y transformarla en un *Árbol de Sintaxis.* Así mismo, el Parser debe poder recibir un *AST* y devolver la expresión lambda a modo de *String* es por eso que se encuentra implementado el método ***unparser.*** *Definimos un* ***Sealed Trait*** *llamado* `AST` *para representar las expresiones, las cuales pueden ser* `Var`*,* `Abstr` *y* `App`*.*

Reductor: Toma una expresión lambda representada como un árbol de sintaxis abstracta (AST) y aplica las reglas de reducción correspondientes según la estrategia seleccionada por el usuario. Implementamos métodos para la reducción β y la sustitución α.

Conclusiones:

El intérprete puede interpretar expresiones lambda, analizar su estructura sintáctica y reducirlas utilizando dos estrategias distintas: Call-by-name y Call-by-value. También es capaz de identificar y devolver las variables libres de una expresión lambda. La implementación se realizó utilizando exclusivamente el paradigma de programación funcional, lo cual permitió aplicar y reforzar los conceptos aprendidos en clase y cumplir con los objetivos mencionados.

Desafíos y Resoluciones

Durante el desarrollo, encontramos varios desafíos, como la correcta gestión de la recursión en la reducción de expresiones complejas y la implementación eficiente del lexer y parser. Estos se resolvieron mediante pruebas y refinamiento de nuestro diseño inicial.