COMP3000 – PROGRAMMING LANGUAGES

ASSIGNMENT 2

SYNTAX ANALYSIS

TRIDEEP LAL DAS

45532125

6/10/2020

Introduction

The aim of this assignment is to build and create a simple functional programming language called FunLang. As with building a programming language, this assignment aims to develop the lexical analyser, parser, and a tree builder for the programming language.

This report will contain a description of the design and implementation to be done including a section on associativity and precedence, the purpose of the functions and brief overview of the test cases and the sufficiency of the test cases.

Design and Implementation

Name Analysis

The bulk of name analysis can be conceptualised by annotating each node in a program's syntax tree with a list of variables which are visible at that node. Name analysis is primarily facilitated by the Kiama library.

Type Analysis

Since the code is static and strong typed, we can make types rather easily, it just depends its expressions; we can use this to infer the type of an expression recursively. Since type is aided by Kiama, we can do things such as tipe for example, where we can check to ensure that the type of each expression matches up with the context of that expression.

Expressions

The implementation of exp allows the SyntaxAnalysis.java to identify special symbols and parse them and ultimately perform actions accordingly. A table can be seen below to identify these special symbols used:

Arithmetic Symbols	Addition(+), Subtraction(-),Multiplication(*), Division(/)
Punctuation	Semicolon(;),
Comparison	==
Conditional	If-then-else
Definition	Definition-expression
App	Expression

Associativity and Precedence

Associativity: The order in which the operations take place. An example would be a == b == c. If the code is left associative, it would look like this: (a == b) == c and right associative would look like this: a == (b == c) and non-associative would give us a syntax error in this instance.

Precedence: The best way to look at precedence is to take an example from normal mathematics in which we have BODMAS or PEDMAS where we do brackets"()" to exponents "^" to Division"/" then Multiplication"*" and addition and subtraction. We can essentially assign our operators to have higher priority of operation based on a precedence we have set for it.

What good will this do for us?

We could use associativity to help us order how we want our expressions to be carried out. Lets say we want the ability to make multiple things equal to each other, we can set it to left associative and like in the example of a == b == c, everything can equal to a(or vice-versa for right associative). And we can set the precedence such as if we don't have parenthesis and we have a == b / 6. Would we want to check a==b first and then divide by 6 or divide b by 6 and then check equality with a.

Testing

A fair number of type and name tests were devised, including tests that check for associativity and precedence. These included tests covering multiple cases where:

- The arithmetic operations worked as expected (this included associative and precedent testing)
- If statements had Boolean-valued conditions, and if- and else- bodies of matching type.
- Let expressions had the correct type.
- Functions were typed correctly.
- Variables could not be used unless they were declared.
- Variables had the correct visibility based on their scope.

The test cases include the base test cases given and the added test cases to check if operations functioned as expected.