CS 3513 - Programming Languages
Programming Project 01

Group 47

JAYAWEERA U.D.	200269J
RAJAPAKSHA S.D	200499M
SIRIWARDANA H.B.K.S.	200607V
THENNAKOON B.W.	200644F

Introduction

The project is to build an interpreter to evaluate RPAL programs.
To evaluate programs it standardizes the abstract syntax tree and uses 13 rules of Control Stack Environment Machine.

Implementation is done using java.

Executing the Program

• **Building with Makefile**

To compile the program run **make**. Then, it will compile the code in src folder to the root directory following the same structure in the src folder structure.

• **Executing the program**

To execute the program run
Input format: java rpal20 <file_name>

[Where file_name is the name of the file that has the RPAL program as the input.]

Project Structure

Following are the main directories of the project.

```
|
|-- src – contain java files and packages of the project.
|     |
|     |-- cseMachine -Include classes to evaluate the standardized tree.
|     |-- lexer -Regex matchers that comply with RPAL's lexical grammar. Used by the scanner to tokenize.
|     |-- parser - Recursive descent parser that complies with RPAL's phrase structure grammar.
|     |-- treeGenerator - Include classes needed for parsing AST tree from the provided file and
|     |                   standardizing it.
|     |-- rpal20.java – main class
|     |
|-- test_files – contain sample RPAL programs used to test the implementation.
|
```

Classes and Significant Functions

• rpal20.java

the rpal20 class is the entry point of the RPAL interpreter. It reads an RPAL input file, constructs an abstract syntax tree (AST), standardizes the AST, and then evaluates the RPAL program using the CSEMachine. The parsing and interpreting tasks are performed by the “Parser” and “CSEMachine” classes, respectively.

Classes in lexer package

- **Scanner.java**

  Scanner: Combines a lexer and a screener. Complies with RPAL's Lexicon.
  a class called Scanner, which is responsible for scanning the RPAL input code and producing tokens to be used by the parser. The Scanner class is responsible for reading characters from the input file, recognizing different token types based on the defined patterns, and building the appropriate tokens to be used in the parsing and interpreting phases of the RPAL interpreter.

- **Token.java**

  The Token class serves as a container for holding the relevant information about each token, such as its type, value (if applicable), and the line number where it was encountered. During the scanning process, the lexer will create instances of the Token class and set these fields accordingly based on the identified tokens in the input RPAL code. The parser will then use these tokens to understand the structure and semantics of the code during the parsing and interpreting phase.

- **LexicalRegexPatterns.java**

  “public class LexicalRegexPatternsthat” contains static regex patterns used for tokenizing RPAL code in the lexer. The regex patterns defined in this class are used by the scanner (lexer) to tokenize the input RPAL code. Each pattern is used to recognize specific types of tokens (like identifiers, operators, punctuation, etc.) and assists in the process of parsing the RPAL program.

- **TokenType.java**

  “public enum TokenType “ which represents the different types of tokens that can be constructed by the scanner in an RPAL interpreter. By defining these token types, the lexer (scanner) will tokenize the input RPAL code into a stream of meaningful tokens.

Classes in parser package

- **ParseException.java**

  the “ParseException” class is used to create custom exceptions specific to parsing errors. When an issue occurs during parsing, such as syntax errors or unexpected input, an instance of “ParseException” can be thrown with a descriptive message, providing information about what went wrong. This can help with debugging and handling parsing-related problems in the RPAL interpreter.

- **Parser.java**

  Recursive Descent Parser implementation for RPAL's phrase structure grammar. This parser is responsible for building an Abstract Syntax Tree (AST) from the input RPAL source code. The grammar and the corresponding parsing rules are outlined in the comments of the code.

  - public class Parser {...}: This class is the main recursive descent parser for RPAL. It defines various methods for parsing different parts of the RPAL grammar.

Classes in treeGenerator package

- **AST.java**

  The AST class represents the Abstract Syntax Tree (AST) used by the CSE (Common Subexpression Elimination) machine. The AST is represented using a first-child next-sibling representation. The class provides methods to manipulate and standardize the AST, and creates delta structures for evaluation.

- **ASTNode.java**

  the “ASTNode” class serves as the building block for constructing the Abstract Syntax Tree, allowing the representation and manipulation of program constructs during compilation or interpretation. Each node represents a specific element in the source code, and the tree structure helps to preserve the syntactic relationships between these elements, aiding in further processing and analysis of the program.

- **ASTNodeType.java**

  The “ASTNodeType” enumeration provides a clear and organized way to represent and distinguish the different types of nodes that can be encountered in the AST for the RPAL language. It ensures that each node type is uniquely identified and associated with its appropriate “printName” for proper visualization and analysis of the AST during the interpreter or compiler processes.

- **StandardizeException.java**

  The “StandardizeException” allows the RPAL interpreter or compiler to signal and handle exceptional situations that may arise during the standardization phase of the AST. This custom exception helps in

improving the clarity and maintainability of the code by providing a specialized error representation for standardization-related issues. When an error occurs during standardization, an instance of "StandardizeException" can be thrown, containing a descriptive error message that can aid in debugging and understanding the cause of the issue.

Classes in cseMachine package

- **Beta.java**

  the "Beta" class is an important component of the CSE machine and is used to handle conditionals in the RPAL interpreter, ensuring proper evaluation and control flow of the program based on the conditions specified in the source code.

- **CSEMachine.java**

  The "CSEMachine" class is responsible for evaluating programs represented as an AST in the RPAL language. It performs program evaluation by processing the control stack and value stack based on certain rules specified in the RPAL language.

- **Delta.java**

  a class named Delta, which represents a lambda closure in the CSE machine for evaluating RPAL programs. A lambda closure is an encapsulation of a function definition along with its environment. The Delta class is used to hold the function body and the list of bound variables (formal parameters) for a lambda expression.

- **Environment.java**

  a class named Environment, which represents an environment in the CSE machine for evaluating RPAL programs. An environment is a data structure that stores variable bindings (mappings) used during the evaluation of RPAL expressions and function calls.

- **Eta.java**

  The Eta class represents an Eta closure, which is created when the RPAL program applies the Y-combinator (fixpoint operator) to a function F. It effectively represents the recursive function F (Y F) without evaluating it. The Eta node is used to manage the recursive function during the CSE machine's evaluation.

- **EvaluationError.java**

  The "EvaluationError" class is a utility class used for handling and reporting evaluation errors that may occur during the execution of the RPAL program in the CSE machine.

- **NodeCopier.java**

  The NodeCopier class is responsible for making deep copies of AST nodes used in the CSE machine. It ensures that each copied node and its sub-nodes are independent of the original nodes, preventing unintended side effects that might occur during the evaluation process.

- **Tuple.java**

  The Tuple class represents a tuple node in the AST. A tuple is a collection of values that can be of different types and is used to hold multiple expressions or arguments.

Run and Testing

Sample Input

E.g: Here is one input file [mytest.txt]

```
let Sum(A) = Psum (A, Order A)
        where rec Psum (T, N) = N eq 0 -> 0
                              | Psum (T, N-1) +T N
in Print (Sum (1,2,3,4,5))
```

Output Format

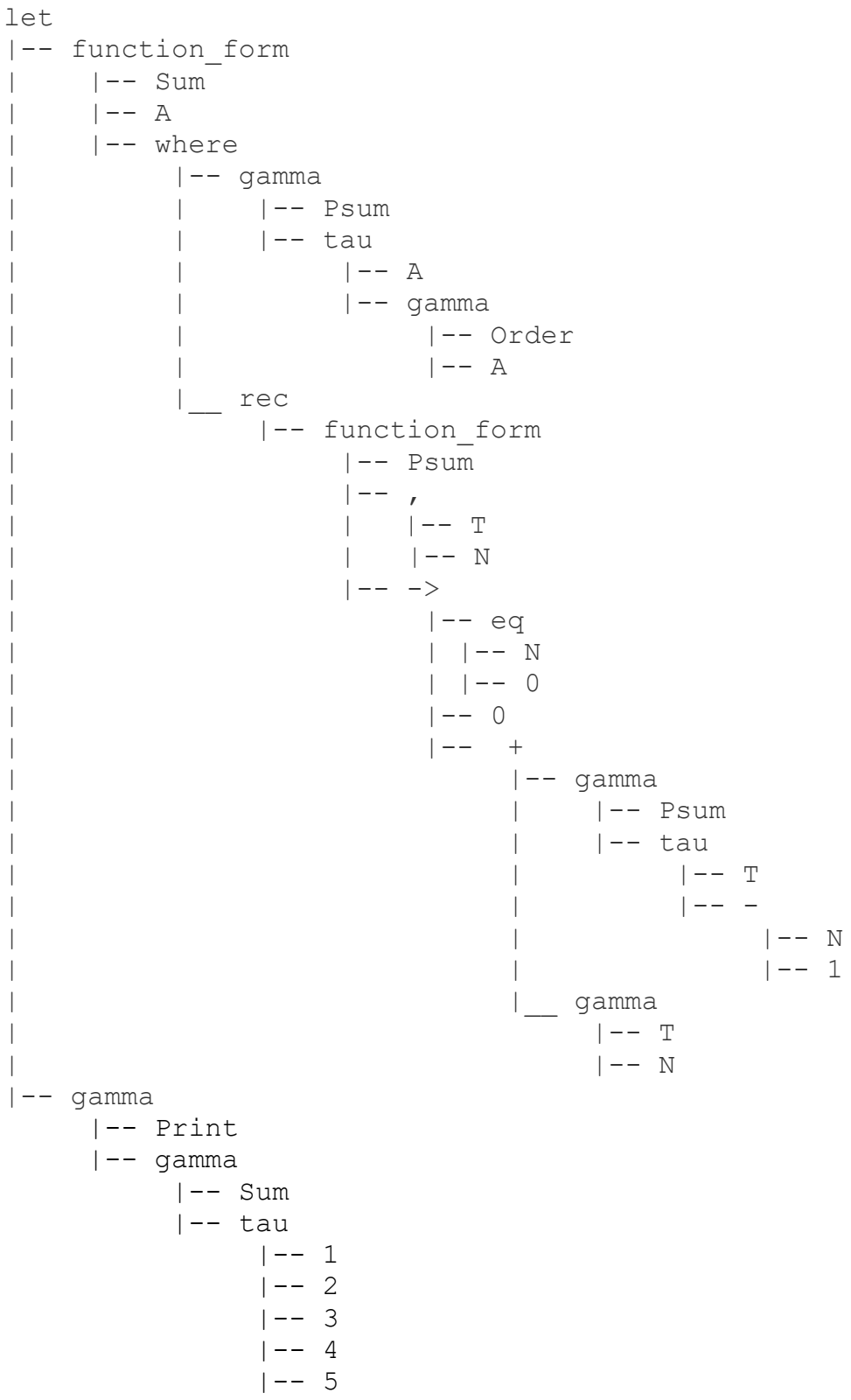
Expected Output of the above program is:

15

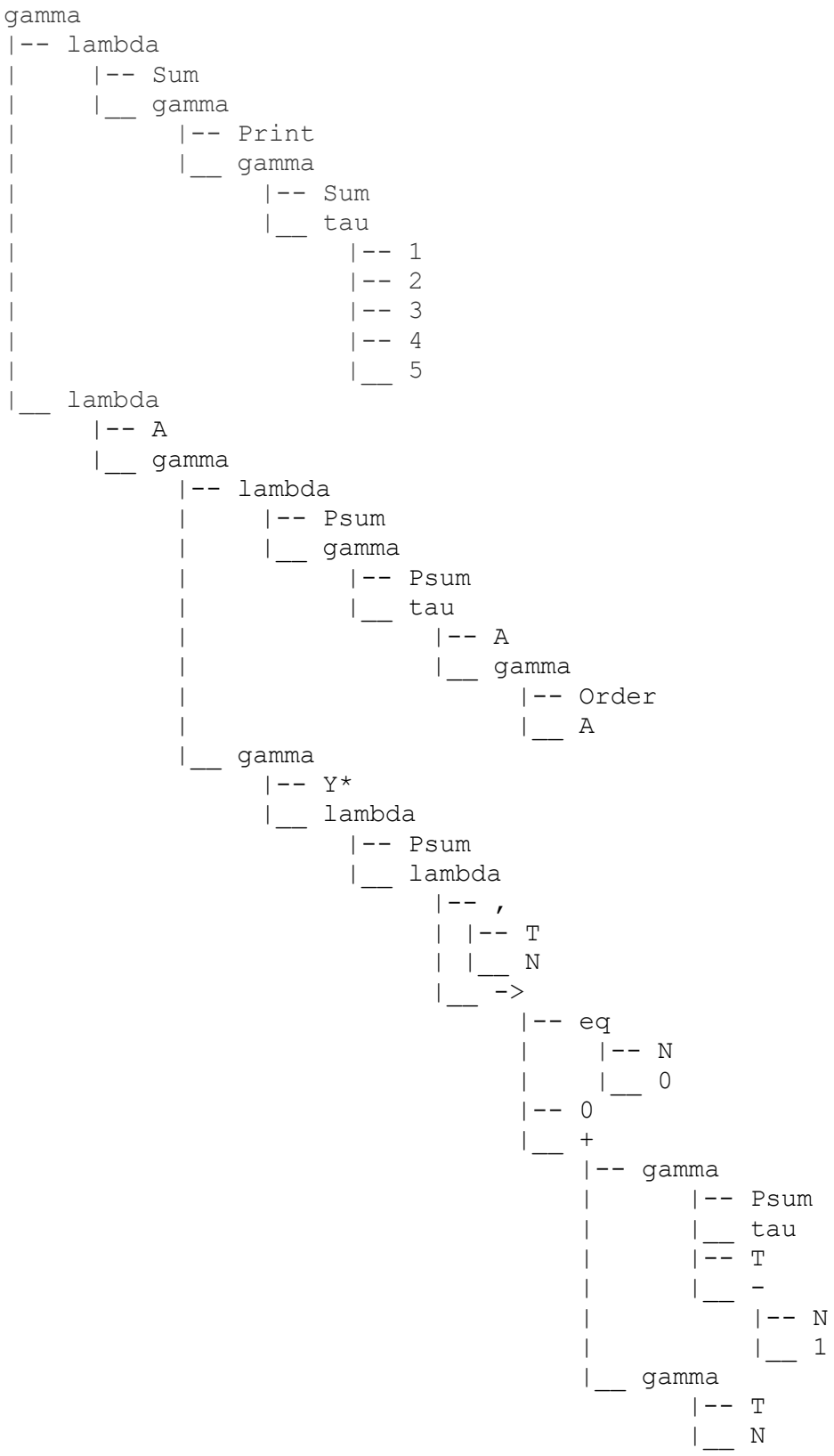
Run sample test cases

Sample Input: java rpal20 mytest

Abstract Syntax Tree Generation:



Standardizing RPAL AST's Generation:



CSE Machine evaluates the ST and gives:

Output:

15

Summary

The RPAL interpreter project aims to evaluate RPAL programs using an interpreter implemented in Java. The interpreter follows a standardized Abstract Syntax Tree (AST) representation and employs the Control Stack Environment (CSE) Machine with 13 rules for program evaluation.

The project's main components are divided into several packages:

1. The lexer package handles lexical analysis, breaking the RPAL input code into meaningful tokens using the Scanner class. Tokens are represented by the Token class, containing information like type, value, and line number.

2. The parser package is responsible for syntactic analysis. The Parser class is a Recursive Descent Parser that constructs the AST from the input RPAL source code, using RPAL's phrase structure grammar. Custom exceptions are defined in the ParseException class to handle parsing errors.

3. The treeGenerator package manipulates and standardizes the AST generated by the parser. The AST class represents the AST using a first-child next-sibling representation. It provides methods for manipulation, standardization, and creates delta structures for evaluation. The ASTNode class serves as the building block of the AST, and ASTNodeType helps distinguish node types. The StandardizeException class handles exceptional situations during standardization.

4. The cseMachine package contains classes required for the CSE Machine-based evaluation of RPAL programs. The CSEMachine class evaluates programs using the AST, processing control and value stacks based on RPAL rules. The Beta class handles conditionals, and Delta represents a lambda closure. The Environment class stores variable bindings, and Eta manages recursive functions. EvaluationError handles evaluation errors, and NodeCopier ensures deep copies of AST nodes. The Tuple class represents tuple nodes in the AST.

The project structure includes main directories: src containing Java files and packages, cseMachine for evaluation classes, lexer for lexical analysis, parser for parsing, and treeGenerator for AST manipulation. The rpal20.java class serves as the entry point, reading RPAL input, constructing the AST, standardizing it, and then evaluating the RPAL program using the CSEMachine.

To execute the program, one can compile it with the provided Makefile or directly run it with Java using the command java rpal20 <file_name>, where file_name is the name of the file containing the RPAL program as input.

Overall, the RPAL interpreter project's well-organized packages facilitate the complete evaluation of RPAL programs, ensuring accuracy and efficiency in interpreting RPAL source code.