CS3513 – Programming Languages

Programming Project

Implementation of RPAL Compiler

Group - TechX

NIVEDSHANKER K - 220436M

SHAAMMA M.S - 220602U

Introduction

This project focuses on building a compiler for RPAL (Right-reference Pedagogic Algorithmic Language), a functional programming language that traces its roots to PAL, developed at MIT in the 1970s by J. Wozencraft and A. Evans. RPAL emphasizes the evaluation of expressions rather than sequential execution.

The initial stage involves constructing a lexical analyzer using the rules specified in the RPAL Lexicon ([1]). This component reads the input RPAL source code and converts it into a stream of meaningful tokens.

Next, we develop a parser based on RPAL's grammar rules as documented in the Phrase Structure Grammar ([2]). The parser processes the token stream to generate an Abstract Syntax Tree (AST), which provides a simplified representation of the program's syntactic structure.

The AST is then converted into a Standardized Tree (ST) using transformation rules described in the Subtree Transformational Grammar ([3]). This step ensures uniform representation for further processing.

Finally, the RPAL program is evaluated using a CSE (Control-Stack-Environment) machine. The CSE machine simulates the execution environment following the operational semantics outlined in [4]. Our implementation is written in Python.

Core Features of RPAL

RPAL is a purely functional language and supports the following fundamental concepts:

Supported Data Types:

- **Integer** – for numerical operations.
- **Boolean** – logical values `true` and `false`.
- **String** – sequences of characters.
- **Tuple** – groupings of values of possibly different types.
- **Function** – treated as first-class entities, supporting higher-order functionality.

Key Language Constructs:

- **Conditional Logic** – Implemented through the `if-then-else` expression, enabling decision-making.

- **Constants** – Immutable values declared for consistent reuse.

- **User-Defined Functions** – Aligned with lambda calculus; functions can be assigned, passed, and returned.
- **Function Application** – Supports direct application of functions to arguments.
- **Operators** – Includes arithmetic (`+`, `-`, `*`, `/`, `**`), logical (`or`, `&`, `not`, `eq`, `ne`), and string manipulation operations (`eq`, `ne`, `Stem`, `Stern`, `Conc`).
- **Recursion** – Enables elegant, self-referential function definitions, a hallmark of functional programming.

Class Architecture

ASTNode
Represents nodes within the AST or ST. Each node maintains links to its children and siblings (previous and next), and encapsulates a Token object containing the token type, lexeme, and position info.

ControlStructure
Manages control structures by maintaining a mapping of control frames and a queue of instruction triples. Supports AST traversal and control block construction.

CSEMachine
Implements the abstract machine for RPAL execution. Maintains stacks for control, data, and environment scopes, and provides logic for processing control structures according to RPAL's evaluation rules.

Lexical
Handles lexical analysis by scanning the RPAL source line-by-line and producing tokens for identifiers, literals, operators, delimiters, and comments. Uses the `scan()` method to iterate through tokens.

Parser
Constructs the AST using a recursive-descent approach. Includes methods such as `E()` for expression parsing and `D()` for definitions, following the grammar structure.

Standardizer
Transforms the AST into a Standardized Tree by applying pattern-specific transformations in a post-order traversal. Ensures that the program structure adheres to a uniform internal format.

TreeBuilder
Aids in dynamically constructing AST nodes during the parsing process, supporting creation and linking of nodes as per the syntactic rules.

Executing the Compiler

Step 1: Extract Project Files

```
tar xvf <project_archive>.tar
```

Step 2: Run the RPAL Compiler

To run a standard RPAL program:

```
python ./myrpal.py <input_file>
```

To output the Abstract Syntax Tree:

```
python ./myrpal.py -ast <input_file>
```

Replace `<input_file>` with the path or name of the RPAL source file.

References

RPAL Lexicon – https://rpal.sourceforge.net/doc/lexer.pdf
Phrase Structure Grammar – https://rpal.sourceforge.net/doc/grammar.pdf
Transformational Grammar – https://rpal.sourceforge.net/doc/semantics.pdf
CSE Machine Operational Rules – https://pdfslide.net/documents/the-cse-machine.html