Report

BNF Documentation and Language Syntax

Backus–Naur Form (BNF) Grammar:

1. **Expression Types:**
   - **INTEGER:** `Integer ::= [0-9]+`
   - **BOOLEAN:** `Boolean ::= 'True' | 'False'`
2. **Operators:**
   - **Arithmetic Operations:**
     - `Add ::= '+'`
     - `Subtract ::= '-'`
     - `Multiply ::= '*'`
     - `Divide ::= '/'`
     - `Modulo ::= '%'`
   - **Boolean Operations:**
     - `And ::= '&&'`
     - `Or ::= '||'`
     - `Not ::= '!'`
   - **Comparison Operations:**
     - `Equal ::= '=='`
     - `NotEqual ::= '!='`
     - `GreaterThan ::= '>'`
     - `LessThan ::= '<'`
     - `GreaterThanOrEqual ::= '>='`
     - `LessThanOrEqual ::= '<='`
3. **Functions:**
   - **Named Function Definition:**
     - `FunctionDef ::= 'Defun' '{' 'name' ':' Identifier ',' 'arguments' ':' '(' Identifier (',' Identifier)* ')' '}' Expression`
   - **Lambda Expression:**
     - `Lambda ::= 'Lambd' Identifier '.' Expression`
   - **Function Application:**
     - `FunctionCall ::= Identifier '(' Expression (',' Expression)* ')'`
4. **Recursion:**
   - **Recursive Function Calls:** Included in function definitions and lambda expressions.

5. **Immutability:**
    - **No Variable Assignments:** All values are immutable, meaning no state changes are allowed.

Syntax Overview:

Lambda Expressions:
```
(Lambd arg. expression)
```
Function Application:
```
functionName(arg1, arg2)
```
Arithmetic and Boolean Operations:
```
(3 + 4) * (2 - 1)
(x > 0) && (y < 10)
```

Design Decisions:

1. **Lexer and Parser Implementation:**
    - **Lexer:** Designed to handle various tokens including operators, function names, and literals. Ensured that the lexer could handle complex expressions and function definitions.
    - **Parser:** Implemented to build an Abstract Syntax Tree (AST) based on BNF. It handles arithmetic operations, boolean expressions, function definitions, and lambda expressions.
2. **Interpreter:**
    - **Evaluation:** Designed to handle function application, lambda expressions, and recursion. Implemented a call stack to manage recursive function calls and local environments.
    - **Error Handling:** Incorporated comprehensive error checking for syntax and runtime errors. Error messages are informative and help in debugging.

Challenges:

1. **Handling Recursion:**
    - **Challenge:** Implementing recursion and ensuring it worked as a replacement for a while loop.
    - **Solution:** Designed a recursive function call mechanism within the interpreter and ensured proper environment handling to manage recursive calls.
2. **Lambda Expressions:**
    - **Challenge:** Correctly parsing and interpreting lambda expressions.
    - **Solution:** Implemented a lambda expression parser and evaluator that correctly applies functions and manages local scopes.
3. **Error Handling:**
    - **Challenge:** Providing meaningful error messages and handling edge cases.
    - **Solution:** Detailed error messages and comprehensive error handling were added to both lexer and parser stages.

Trade-offs and Limitations:

- **Trade-offs:**
  - **Complexity vs. Performance:** Ensuring robust error handling and comprehensive language support may impact performance. Emphasis was placed on correct implementation and error handling.
- **Limitations:**
  - **Limited Standard Library:** The language does not include built-in functions beyond arithmetic and boolean operations, limiting its usability for more complex tasks.
  - **No Built-in State Management:** The language's immutability and lack of variable assignments may limit its expressiveness for certain programming paradigms.

This report provides a comprehensive overview of the functional programming interpreter project, covering the BNF grammar, language features, user guide, test suite, and a discussion of design considerations and challenges.