**Documentation of the BNF: Language Syntax and Features**

**1. Overview**

This document details the syntax and features of the language defined by the BNF grammar provided. The grammar specifies the structure of a programming language that supports function definitions, conditional statements, and expressions, including arithmetic, boolean operations, lambda expressions, and function calls.

**2. BNF Grammar**

The BNF grammar can be seen in the project files.

**3. Language Syntax and Features**

1. **Programs and Statements:**
   * A <program> consists of one or more <statements> followed by an end-of-file marker (<EOF>).
   * <statements> can include function definitions (<function\_def>), if statements (<if\_statement>), and various forms of expressions (<expression>).
2. **Function Definitions:**
   * Functions are defined using the keyword Defun. The syntax includes specifying a function name and a list of arguments within curly braces {}. The function body is an <expression>.
   * Example:

Defun {name: factorial, arguments: (n)} n \* factorial(n - 1)

1. **Conditional Statements:**
   * The language supports if-else statements, allowing conditional execution of code blocks and recursion (for the stop condition). The syntax includes the keyword if, followed by a condition in parentheses (), and the true and false branches in curly braces {}
   * The if statement must be followed by an else statement that includes the keyword else followed by curly braces {} that will hold the expression to be executed.
   * Example:

Defun {name: sum\_of\_digits, arguments: (n,)}

if (n == 0) {False}

else {n % 10 + sum\_of\_digits(n / 10)}

**Expressions:**

* + Expressions include unary operations (<unary\_op>), binary operations (<binary\_op>), integers, booleans, identifiers, function calls, and lambda expressions.
  + Parentheses can be used to group expressions.

1. **Unary and Binary Operations:**
   * Unary operations include negation (!).
   * Binary operations cover arithmetic (+, -, \*, /, %), boolean (&&, ||), and comparison operators (==, !=, >=, <=, >, <).
2. **Lambda Expressions:**
   * Lambda expressions allow the creation of anonymous functions. The syntax uses the Lambd keyword followed by a parameter and a body.
   * Example:

(Lambd x. (Lambd y. (x + y)))

1. **Function Calls:**
   * Functions can be called using their identifier followed by arguments in parentheses and a comma seperates the different values. Lambda expressions can also be called directly and passed values.
   * Example:
     + factorial(5)
     + (Lambd x. (Lambd y. (x + y)))(6, factorial(5))
2. **Identifiers, Integers, and Booleans:**
   * Identifiers are names for variables and functions, following a pattern similar to most programming languages (e.g., letters, digits, and underscores).
   * Integers are sequences of digits, optionally preceded by a minus sign.
   * Booleans are True and False.

**4. Trade-offs and Limitations**

1. **Trade-offs:**
   * **Simplicity vs. Expressiveness:** The grammar is designed to be simple, which may limit some advanced features such as complex data structures or control flow mechanisms.
2. **Limitations:**
   * **No Support for Arrays or Complex Data Structures:** The language lacks arrays or other complex data structures, limiting the types of data that can be manipulated directly.
   * **Limited I/O Operations:** The language seems to focus on expressions and function evaluations without explicit syntax for input/output operations.
   * **No Variable Mutation:** The language syntax as described does not support variable assignments or mutation, aligning with functional programming principles but potentially limiting its applicability for certain tasks.

**5. Conclusion**

The BNF grammar provided defines a language with a clear focus on functional programming principles, with a straightforward syntax for defining functions, conditional logic, and expressions. While the language is simple and powerful within its domain, its limitations in terms of data structures, variable mutation, and I/O operations may restrict its use to specific problem domains.