**Design of Interpreter for Language of Typed Arithmetic Expressions**

This document details the steps taken to implement the interpreter for the language of Typed Arithmetic Expressions.

**Programming Language used:** Python

An outline of the steps is given here:

1. Build a lexical analyser to parse the given input string into tokens.

2. If there are invalid tokens, report “LexicalError”.

3. Construct an Abstract Syntax Tree from the tokens obtained and the grammar rules.

4. If AST cannot be constructed, report “SyntaxError”.

5. Perform type checking on the AST using the typing rules for the language.

6. If there are type errors, report “TypeError”.

7. If there are no type errors, evaluate the expression using the evaluation rules for the language.

**Tools and Frameworks Used**

No additional tools and frameworks have been used apart from core python. Every module, from lexical analyser to evaluator, has been manually coded.

**Lexical Analyser**

The input sentence is parsed into various tokens depending upon the term. There are total of 12 tokens:

1. IF- if word.

2. THEN- then word.

3. ELSE- else word.

4. TRUE- true term.

5. FALSE- false term.

6. OPPAR- opening parenthesis.

7. CLPAR- closing parenthesis.

8. ZERO- 0.

9. SUCC- succ.

10. PRED- pred.

11. ISZERO- iszero.

12. INV- invalid token.

**AST Constructor**

The list of tokens is passed into an AST constructor. The AST constructor uses the list of tokens and grammar rules for constructing an Abstract Syntax Tree for the term. The constructor is designed based on the pseudocode given in ALSU[1].

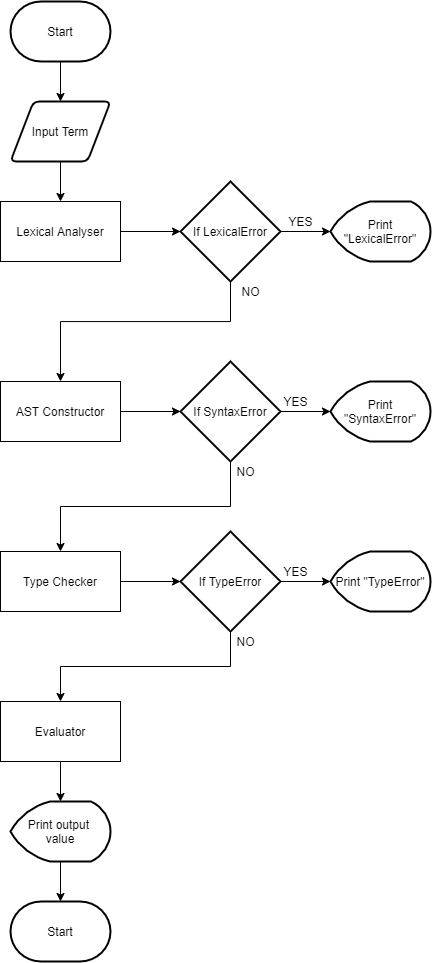
**Type Checker**

The AST is traversed according to the typing rules of the language and checked for any typing errors. There are two types: Nat and Bool.

**Evaluator**

If there are no typing errors in the term, it is evaluated using evaluation rules of the language.

**Design**

****A flowchart demonstrating the basic flow of the interpreter

**References**

1. Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman, “Syntax Analysis” in *Compilers: Principles, Techniques and Tools.* MA, USA: Pearson, 2006, pp. 219-220.