Syantx Analysis

Oh Gyuhyeok (2020-10485) and Hwang SeungJun (2020-15313)

SNUCSE

October 27, 2023

Abstract

abstract

1 Introduction

In the second phase of our term project, our primary focus was on implementing a top-down parser for the SnuPL/2 language. This report provides an overview of the work completed during this phase, including the design, implementation, and testing of the parser. cite [1, cite1] cite [2, cite2] cite [3, cite3]

2 Progress Overview

The main objectives of this phase were as follows:
1. Design and implement a top-down parser for the SnuPL/2 language.
2. Generate an abstract syntax tree (AST) in both textual and graphical forms.
3. Perform syntactical checks, such as ensuring symbols are defined before use.
4. Enforce constraints related to the SnuPL/2 syntax, such as matching declaration and end identifiers.

3 Design

3.1 Grammar

EBNF notation is used to describe the grammar of the SnuPL/2 language.

3.2 Method

We implemented additional method to implement parser. List of method is below.

- `parseModule` : parse module
- `parseConstDecl` : parse const declaration

- `parseVarDecl` : parse variable declaration
- `parseSubroutineDecl` : parse subroutine declaration
- `parseFormalParam` : parse formal parameter
- `parseSubroutineBody` : parse subroutine body
- `parseStatSequence` : parse statement sequence
- `parseStatement` : parse statement
- `parseAssignment` : parse assignment
- `parseSubroutineCall` : parse subroutine call
- `parseIfStatement` : parse if statement
- `parseWhileStatement` : parse while statement
- `parseReturnStatement` : parse return statement
- `parseExpression` : parse expression
- `parseSimpleExpr` : parse simple expression
- `parseTerm` : parse term
- `parseFactor` : parse factor
- `parseQualident` : parse qualident
- `parseNumber` : parse number
- `parseBoolean` : parse boolean
- `parseChar` : parse char
- `parseString` : parse string
- `parseType` : parse type
- `parseBasetype` : parse basetype
- `parseIdent` : parse identifier
- `parseRelOp` : parse relational operator
- `parseTermOp` : parse term operator
- `parseFactOp` : parse factor operator
- `constChar` : parse constant char

3.3 Algorithm

Algorithm 1 Description of the algorithm

Input: input
Output: output
AAA

4 Result

We utilized a provided test program that printed the AST to validate our parser's functionality. The test program allowed us to examine the generated AST for different test cases. In addition to the provided test cases, we created custom test cases to cover more complex and special scenarios, ensuring the robustness of our parser.

References

[1] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compiler: Principles, Techniques, and Tools*. Pearson Addison-Wesley, 2007.

[2] Bernhard Egger. *Compiler Lecture Note*. 2023.

[3] Bernhard Egger. Snupl2 compiler, Sep 2023.

A Appendix

A.1 Additional Test Cases

Input:

```
    import numpy as np

    def incmatrix(genl1,genl2):
        m = len(genl1)
        n = len(genl2)
        M = None #to become the incidence matrix
        VT = np.zeros((n*m,1), int)  #dummy variable

        #compute the bitwise xor matrix
        M1 = bitxormatrix(genl1)
        M2 = np.triu(bitxormatrix(genl2),1)

        for i in range(m-1):
            for j in range(i+1, m):
                [r,c] = np.where(M2 == M1[i,j])
                for k in range(len(r)):
                    VT[(i)*n + r[k]] = 1;
                    VT[(i)*n + c[k]] = 1;
                    VT[(j)*n + r[k]] = 1;
                    VT[(j)*n + c[k]] = 1;

                    if M is None:
                        M = np.copy(VT)
                    else:
                        M = np.concatenate((M, VT), 1)

                    VT = np.zeros((n*m,1), int)

        return M
```

Listing 1: caption text

Output:

```
result
```