Lab 3 – Parsing

Ryan Munger
Ryan.Munger1@marist.edu

February 25, 2025

1 Crafting a Compiler

1.1 4.7 – Derivations

A grammar for infix expressions follows:

```
1  Start → E $
2  E     → T plus E
3        | T
4  T     → T times F
5        | F
6  F     → ( E )
7        | num
```

Figure 1: Grammar for 4.7

(A) Show the leftmost derivation of the following string: [num plus num times num plus num $]
Start
E $
T plus E $
F plus E $
num plus E $
num plus T plus E $
num plus T times F plus E $
num plus F times F plus E $

num plus num times F plus E $
num plus num times num plus E $
num plus num times num plus T $
num plus num times num plus F $
num plus num times num plus num $

(B) Show the rightmost derivation of the following string: [num times num plus num times num $]
Start
E $
T plus E $
T plus T $
T plus T times F $
T plus T times num $
T plus F times num $
T plus num times num $
T times F plus num times num $
T times num plus num times num $
F times num plus num times num $
num times num plus num times num $

1.2 5.2c – Recursive Descent Parser

(C) Construct a recursive-descent parser based on the grammar below (psuedo code only)

```
1 Start  → Value $
2 Value  → num
3        | lparen Expr rparen
4 Expr   → plus Value Value
5        | prod Values
6 Values → Value Values
7        | λ
```

Figure 2: Grammar for 5.2c

```
struct Tree {
    ...
}

func (tree Tree) moveUp() {
    tree.current = tree.current.parent
}

struct Node {
    ...
}

func match(expected) {
    if currentTok == expected {
        addNode(leaf, x)
    } else {
        err
    }
    currentTokIdx++
    currentTok = tokens[currentTokIdx]
}

var tree Tree = Tree{}
var tokens []Token = []Token{}
var currentTok Token
var currentTokIdx int = 0
currentTok = tokens[currentTokIdx]

func start() {
    parseValue()
    match($)
}

func parseValue() {
    if currentTok == num {
        match(num)
    } else if currentTok == lparen {
        match(lparen)
        parseExpr()
        match(rparen)
    } else {
        err
    }
    tree.moveUp()
}
```

```
func parseExpr() {
    if currentTok == plus {
        match(plus)
        parseValue()
        parseValue()
    } else if currentTok == prod {
        match(prod)
        parseValues()
    } else {
        err
    }
    tree.moveUp()
}

func parseValues() {
    if currentTok == num || currentTok == lparen {
        parseValue()
        parseValues()
    } else {
        /* Epsilon */
    }
    tree.moveUp()
}
```

2 Dragon Book

2.1 4.2.1 A, B, C – Derivations and Parse Trees

Consider the grammar $S \rightarrow SS+|SS*|a$ and the string aa+a*.
(A) Give a leftmost derivation for the string.
S
SS*
SS+S*
aS+S*
aa+S*
aa+a*

(B) Give a rightmost derivation for the string.
S
SS*
SS+S*
SS+a*
Sa+a*
aa+a*

(C) Give a parse tree for the string.