Lab Three

Timothy Polizzi
Timothy.Polizzi1@Marist.edu

October 16, 2019

1 *Crafting a Compiler*

1. 4.7

   (a) Left

   Start $\Rightarrow_{lm}$ E \$
   $\Rightarrow_{lm}$ T plus E \$
   $\Rightarrow_{lm}$ F plus T \$
   $\Rightarrow_{lm}$ num plus F times F \$
   $\Rightarrow_{lm}$ num plus num times (E) \$
   $\Rightarrow_{lm}$ num plus num times (T plus E) \$
   $\Rightarrow_{lm}$ num plus num times (F plus T) \$
   $\Rightarrow_{lm}$ num plus num times (num plus F) \$
   $\Rightarrow_{lm}$ num plus num times (num plus num) \$

   (b) Right

   Start $\Rightarrow_{rm}$ E \$
   $\Rightarrow_{rm}$ T plus E \$
   $\Rightarrow_{rm}$ T plus T \$
   $\Rightarrow_{rm}$ T plus T times F \$
   $\Rightarrow_{rm}$ T plus F times F \$
   $\Rightarrow_{rm}$ T plus num times num \$
   $\Rightarrow_{rm}$ T times F plus num times num \$
   $\Rightarrow_{rm}$ F times num plus num times num \$
   $\Rightarrow_{rm}$ num times num plus num times num \$

(c) precedence
This grammar structures expressions by having order of operations enforced when the grammar is used in a leftmost derivation. Otherwise it will just parse the information.

2. 5.2c

```
procedure Start(tokenStream)
  switch(tokenStream.peek())
    case {num, lparen}
      call Value()
      match($)
  end

procedure Value(tokenStream)
  switch(tokenStream.peek())
    case {num}
      match(num)
    case {lparen}
      match(lparen)
      call Expr()
      match(rparen)
  end

procedure Expr(tokenStream)
  switch(tokenStream.peek())
    case {plus}
      match(plus)
      call Value()
      call Value()
    case {prod}
      match(prod)
      call Values()
  end

procedure Values(tokenStream)
  switch(tokenStream.peek())
    case {num, lparen}
      call Value()
      call Values()
    case {rparen}
      return()
  end

```

2 *Dragon*

1. 4.2.1 a

   S $\Rightarrow$ SS* $\Rightarrow$ SS+S* $\Rightarrow$ aa+a*

2. 4.2.1 b

   $S \Rightarrow SS* \Rightarrow Sa* \Rightarrow SS+a* \Rightarrow aa+a*$

3. 4.2.1 c