Joseph Schmidt

Lab3: Turning tokens into sentences with your Parser

Crafting a Compiler:

4.7

num plus num times num plus num $ (left most derivation)

(a) goal

1. <E>$
2. <T> plus <E>$
4. <T> plus <T> times <F>$
6. <T> plus <T> times <(E)> $
1. <T> plus <T> times <T> plus <E> $
5. <F> plus <F> times <F> plus <E> $
7. num plus <F> times <F> plus <E>$
7. num plus num times <F> plus <E>$
7. num plus num times num plus <E>$
3. num plus num times num plus <T>$
5. num plus num times num plus <F>$
7. num plus num times num plus num $

num times num plus num times num $

(b) goal (right most derivation)

1. <E>$
2. <T> plus <E>$
3. <T> plus <T>$
4. <T> plus <T> times <F>$
7. <T> plus <T> times num $
5. <T> plus <F> times num $
7. <T> plus num times num $
4. <T> times <F> plus num times num $
7. <T> times num plus num times num $
5. <F> times num plus num times num $
7. num times num plus num times num $

(c) Times proceeds plus. Otherwise, there would not be proper associativity between ops.

5.2c.

```
parse(){
        parseValue()
        matchAndConsume(“EOF”)
}
parseValue(){
        if(currentToken.kind == num){
                matchAndConsume(“num”)
        }else if (currentToken.kind == lparen){
                matchAndConsume(“lparen”)
                parseExpr()
                matchAndConsume(“Rparen”)
        }
}
parseExpr(){
        if(currentToken.kind == plus){
                matchAndConsume(“plus”)
                parseValue()
                parseValue()
        }else if (currentToken.kind == prod){
                matchAndConsume(“prod”)
                parseValues()
        }
}
```

```
parseValues(){
    if(previousToken == prod || previousToken == Value){
    //this has to be true since we would not have gotten here otherwise
        parseValue()
        parseValues()
    }else{
        //Do nothing. ε production.
    }
}
```

Dragon Bible

4.2.1

(a) aa + a* (left most derivation)

goal

1. SS*
2. SS+S*
3. aS+S*
3. aa+S*
3. aa+a*

(b) (right most derivation)

goal

2. SS*
3. Sa*
1. SS+a*
3. Sa+a*
3. aa+a*

(c)