

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. $\langle E \rangle \$$
2. $\langle T \rangle \text{ plus } \langle E \rangle \$$
4. $\langle T \rangle \text{ plus } \langle T \rangle \text{ times } \langle F \rangle \$$
6. $\langle T \rangle \text{ plus } \langle T \rangle \text{ times } \langle (E) \rangle \$$
1. $\langle T \rangle \text{ plus } \langle T \rangle \text{ times } \langle T \rangle \text{ plus } \langle E \rangle \$$
5. $\langle F \rangle \text{ plus } \langle F \rangle \text{ times } \langle F \rangle \text{ plus } \langle E \rangle \$$
7. num plus $\langle F \rangle \text{ times } \langle F \rangle \text{ plus } \langle E \rangle \$$
7. num plus num times $\langle F \rangle \text{ plus } \langle E \rangle \$$
7. num plus num times num plus $\langle E \rangle \$$
3. num plus num times num plus $\langle T \rangle \$$
5. num plus num times num plus $\langle F \rangle \$$
7. num plus num times num plus num \$

num times num plus num times num \$

(b) goal (right most derivation)

1. $\langle E \rangle \$$
2. $\langle T \rangle \text{ plus } \langle E \rangle \$$
3. $\langle T \rangle \text{ plus } \langle T \rangle \$$
4. $\langle T \rangle \text{ plus } \langle T \rangle \text{ times } \langle F \rangle \$$
7. $\langle T \rangle \text{ plus } \langle T \rangle \text{ times num } \$$
5. $\langle T \rangle \text{ plus } \langle F \rangle \text{ times num } \$$
7. $\langle T \rangle \text{ plus num times num } \$$
4. $\langle T \rangle \text{ times } \langle F \rangle \text{ plus num times num } \$$
7. $\langle T \rangle \text{ times num plus num times num } \$$
5. $\langle F \rangle \text{ times num plus num times num } \$$
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.  $\epsilon$  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)

