Constructing the Predictive Parser Table

A predictive parse table has a row for each non-terminal X, and a column for each input token t. Entries table[X,t] contain productions:

```
for each X -> gamma
    for each t in FIRST(gamma)
        table[X,t] = X->gamma
    if gamma is nullable
        for each t in FOLLOW(X)
            table[X,t] = X->gamma
```

Compute the predictive parse table for

Z → d | X Y Z

X → a | Y

Y → c | ε

	a	*c*	*d*
X	*X→a* *X→Y*	*X→Y*	*X→Y*
Y	*Y→ ε*	*Y→ ε* *Y→c*	*Y→ ε*
Z	*Z→XYZ*	*Z→XYZ*	*Z→XYZ* *Z→d*

Multiple entries in the Predictive parse table: Ambiguity

An ambiguous grammar will lead to multiple entries in the parse table.

Our grammar IS ambiguous, e.g. Z → d

but also Z→XYZ→YZ→d

For grammars with no multiple entries in the table, we can use the table to produce one parse tree for each valid sentence. We call these grammars LL(1): Left to right parse, Left-most derivation, 1 symbol lookahead.

A recursive descent parser examines input left to right. The order it expands non-terminals is leftmost first, and it looks ahead 1 token.

One more time

Balanced parentheses grammar 1:

S → (S) | SS | ε

1. **Augment the grammar with EOF/$**
2. **Construct Nullable, First and Follow**
3. **Build the predictive parse table, what happens?**

One more time, but this time with feeling …

Balanced parentheses grammar 2:

S → (S)S | ε

1. **Augment the grammar with EOF/$**
2. **Construct Nullable, First and Follow**
3. **Build the predictive parse table**
4. **Using the predictive parse table, construct the parse tree for**

   ( ) (( )) $

   and

   ( ) ( ) ( ) $