1 More Grammar Hacking

1.1 Left Recursion

The authors of last week's grammar are indebted to you for adding the correct order of operations to their Context-Free Grammar. In their excitement, they run a parser on some expressions in this grammar and it hangs forever. They now return to you, the resident grammar experts, for help, and you immediately see that the problem is left recursion. Can you eliminate all of the left recursion in this grammar?

$$
\begin{array}{rcl}
\langle \text{Expr} \rangle & \rightarrow & \langle \text{Expr} \rangle + \langle \text{Term} \rangle \\
\langle \text{Expr} \rangle & \rightarrow & \langle \text{Term} \rangle \\
\langle \text{Term} \rangle & \rightarrow & \langle \text{Term} \rangle \ * \ \langle \text{Int} \rangle \\
\langle \text{Int} \rangle & \rightarrow & 2 \mid 3 \mid 5
\end{array}
$$

Answer:

$$
\begin{array}{rcl}
\langle \text{Expr} \rangle & \rightarrow & \langle \text{Term} \rangle \ \langle \text{Expr'} \rangle \\
\langle \text{Expr'} \rangle & \rightarrow & + \ \langle \text{Term} \rangle \ \langle \text{Expr'} \rangle \mid \varepsilon \\
\langle \text{Term} \rangle & \rightarrow & \langle \text{Term} \rangle \ \langle \text{Term'} \rangle \\
\langle \text{Term'} \rangle & \rightarrow & * \ \langle \text{Int} \rangle \mid \langle \text{Term'} \rangle \mid \varepsilon \\
\langle \text{Int} \rangle & \rightarrow & 2 \mid 3 \mid 5
\end{array}
$$

1.2 Left Factoring

You return the fixed grammar to its grateful authors, who are now attempting to expand their grammar. Thanks to the latest advances in technology, they are now able to support multi-digit numbers, with precision up to a single decimal place. They have changed the grammar to account for this by changing **Ints** to **Nums** and adding these lines:

$$
\begin{array}{lcl}
\langle \text{Num} \rangle & \rightarrow & \langle \text{Digit} \rangle \ \langle \text{Num} \rangle \\
\langle \text{Num} \rangle & \rightarrow & \langle \text{Digit} \rangle \\
\langle \text{Num} \rangle & \rightarrow & \langle \text{Digit} \rangle \ . \ \langle \text{Digit} \rangle \\
\langle \text{Digit} \rangle & \rightarrow & 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9
\end{array}
$$

However, when you try to run a predictive parser on some expressions in this grammar, you get some unexpected results. Which of these terminals could the parser be incorrectly parsing?

1. **3**
2. **.**

Answer: 3

Left-factor the grammar to remove this ambiguity.

Answer:

$$
\begin{array}{rcl}
\langle \text{Num} \rangle & \rightarrow & \langle \text{Digit} \rangle \ \langle \text{PostDigit} \rangle \\
\langle \text{PostDigit} \rangle & \rightarrow & \langle \text{Digit} \rangle \mid \ . \ \langle \text{Digit} \rangle \mid \ \varepsilon \\
\langle \text{Digit} \rangle & \rightarrow & 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9
\end{array}
$$

2 Constraint Propagation

You return the fixed grammar to its authors, who now believe they can do anything with you by their side. They have decided to implement a grammar for some fancier arithmetic expressions using specific integers.

$$
\begin{array}{lcl}
\langle \text{A} \rangle & \rightarrow & 6 \ *[\langle \text{B} \rangle] \\
\langle \text{B} \rangle & \rightarrow & \langle \text{C} \rangle \ \text{-} \langle \text{D} \rangle \\
\langle \text{C} \rangle & \rightarrow & (3) \mid 3 \mid \varepsilon \\
\langle \text{D} \rangle & \rightarrow & (5) \langle \text{E} \rangle
\end{array}
$$

$$\langle \text{E} \rangle \quad \rightarrow \quad \text{x} \mid \varepsilon$$

Generate an example expression that satisfies this grammar.

Answer: The following are all acceptable:

- 6 * [(3)-5x]
- 6 * [3-5x]
- 6 * [-5x]
- 6 * [(3)-x]
- 6 * [3-x]
- 6 * [-x]

2.1 Nullable

Compute the set **Nullable**.

Recall: **Nullable**(*NT*) = {*NT: NT* is a non-terminal that is able to derive ε.}

Answer: {B, C, E}

2.2 First

Which of these terminals is not in the set **First**(B)?

1. **5**
2. **(**
3. **3**
4. **-**

Recall: **First**(*NT*) = {*T: T* is a terminal that may appear at the start of a string derived from NT.}

Answer: 5

2.3 Follow

For which three nonterminals is the set **Follow**(*NT*) the same?

Recall: **Follow**(*NT*) = {*T: T* is a terminal that may appear directly after an expanded NT term in any valid derivation from the starting symbol.}

Answer: {B, D, E}

- **Follow**(A) = {*eof*}
- **Follow**(B) = {]}
- **Follow**(C) = {-}
- **Follow**(D) = {]}
- **Follow**(E) = {]}