Throughout this discussion we will refer to the following LL(1) expression grammar. Our goal will be to write a parse function for \(<\text{term\_tail}>\) as would be done in writing a recursive descent parser, and in the process uncover first, follow, and predict sets.

1. \(<\text{expr}>\) \(\rightarrow\) \(<\text{term}>\) \(<\text{expr\_tail}>\) $\$
2. \(<\text{expr\_tail}>\) \(\rightarrow\) \(<\text{add\_op}>\) \(<\text{term}>\) \(<\text{expr\_tail}>\) | ε
3. \(<\text{term}>\) \(\rightarrow\) \(<\text{factor}>\) \(<\text{term\_tail}>\)
4. \(<\text{term\_tail}>\) \(\rightarrow\) \(<\text{mult\_op}>\) \(<\text{factor}>\) \(<\text{term\_tail}>\) | ε
5. \(<\text{factor}>\) \(\rightarrow\) NUM | ( \(<\text{expr}>\) )
6. \(<\text{add\_op}>\) \(\rightarrow\) + | -
7. \(<\text{mult\_op}>\) \(\rightarrow\) * | /

The parse function we want to write is:

\[
\text{term\_tail}()
\{
    \text{if (lookahead} \in \text{Set A)}
    \{
        \text{mult\_op();}
        \text{factor();}
        \text{term\_tail();}
    \}
    \text{else if (lookahead} \in \text{Set B)}
    \{/*	ext{do nothing */}
    \text{else}
    \text{parse\_error();}
\}
\]

We must compute the predict sets Set A and Set B.

Predicting the First RHS

What terminal value for lookahead will predict that we will
parse the first right hand side of production 4? Looking at <mult_op> we see from production 7 (<mult_op>) that the first symbol in the first RHS must derive either '*' or '/'. So '*' and '/' will both predict that we will parse this RHS and they belong in Set A. Furthermore, since we are only "looking ahead" by one token and the first terminal derived from this RHS must be either '*' or '/' these are the only possible values for lookahead (assuming a correct parse). No other terminals should go into Set A and:

\[ \text{Set A} = \{*, /\}. \]

If, instead, production 7 had been:

7. \(<\text{mult_op}> \rightarrow * \mid / \mid \varepsilon\)

then in the RHS:

\(<\text{mult_op}> <\text{factor}> <\text{term_tail}>\)

<mult_op> could have derived \(\varepsilon\) so in computing Set A we would have had to consider which terminal <factor> could derive. If it had been the case that <factor> could also have derived \(\varepsilon\) we would also consider terminals derived from <term_tail>. Since <term_tail> can derive \(\varepsilon\) it would then be the case that he entire RHS could derive \(\varepsilon\) and we would have to ... wait a moment for this one.

The set we have computed for this RHS is its first set. The first set of a string of grammar symbols (they could terminals and/or non-terminals) is just the set of terminals which can be the first symbols in a derivation of that string. That is:

\[ \text{first}(S) = \{u \mid u \text{ is a terminal and } S \Rightarrow^* tS' \text{ for some string } S' \text{ of grammar symbols}\} \]

Note that this definition is slightly different from the definition in the Dragon Book which allows \(\varepsilon\) as an element of a first set.

**Predicting the Second RHS**

This is where the work is. You might want to just skim this
section to get the flavor of what is going on, then read the
next section (Computing follow sets), and then come back and
read this section more closely.

The second RHS of production 4 is $\varepsilon$. We know that to compute Set
B we must compute its first set. First{$\varepsilon$} is empty so there is
nothing gained here. To see which lookahead values will predict
this RHS we must see what happens when it does derive $\varepsilon$ (which
this one can only do). In this case as we were parsing our
input the LHS non-terminal <term_tail> must have occured
somewhere in a derivation but contributed no terminal symbols.
Hence the next terminal in the input stream (the lookahead),
assuming a correct parse, must be contributed by symbols which
follow <term_tail> in this derivation.

We must look at every RHS in which <term_tail> appears and see
what terminal symbol can immediately follow it in some
derivation (i.e. find the first set of the remainder of the
RHS). <term_tail> occurs in RHSs of two productions, 3 (<term>)
and 4 (term_tail).

Look first at the RHS of production 4:

4. <term_tail> $\rightarrow$ <mult_op> <factor> <term_tail> | $\varepsilon$

Here <term_tail> occurs at the end of the RHS. If this RHS had
instead been:

<mult_op> <factor> <term_tail> NUM

then we would have included NUM in Set B.

Since <term_tail> is at the end of this RHS we conclude that the
next terminal from the input must be from a place in the grammar
where the LHS <term_tail> appears. Hence we must see in what
productions <term_tail> appears in the RHS and see ... We just
went in a circle so <term_tail> occuring in this RHS takes us
nowhere and we can skip over it.

The other production where <term_tail> appears in a RHS is:

3. <term> $\rightarrow$ <factor> <term_tail>

What follows <term_tail> is nothing, so we have to see what
follows <term> (the LSH of this production) and include those terminals in Set B.

This brings us to finding RHSs where <term> appears and see what follows it. (Are you still following this?) <term> appears in RHSs of productions 1 and 2.

What can follow <term> in production 1?

1. <expr> → <term> <expr_tail> $

Here "<expr_tail> $" follows <term>. <expr_tail> can derive strings beginning with '+' and '-' (from <add_op>) so '+' and '-' go into Set B. <expr_tail> can also derive ε so <term> could be immediately followed by $. $ goes into Set B.

What can follow <term> in production 2?

2. <expr_tail> → <add_op> <term> <expr_tail> | ε

In the first RHS here <term> is followed by <expr_tail> which can derive strings beginning with '+' and '-' so '+' and '-' go into Set B. We already knew this.

<expr_tail>, following <term> in this RHS, can derive ε so to see what is following <term> here we must see what can follow the LHS of this production, <expr_tail> in all RHS where it appears. Ouch!

Finding what can follow <expr_tail> takes us to production 1:

1. <expr> → <term> <expr_tail> $

where we pick up $ to add to Set B which is now {+, -, $}. It also appears at the end of the first RHS of production 2:

2. <expr_tail> → <add_op> <term> <expr_tail> | ε

telling us that we must find what follows the LHS (<expr_tail> again) taking us in another circle, so we skip this one.

We have found every lookahead value which can predict the second RHS of production 4 (do you remember that we were trying to parse production 4?) and so we have all of Set B: {+, -, $}.  

- 4 -
What we have done is seen what can follow \(<\text{term\_tail}\rangle\) in RHSs of productions of our grammar. This involved finding what can follow \(<\text{term}\rangle\) in RHSs. We have computed "follow(\(<\text{term\_tail}\rangle\)"
and this computation required us to first compute 
"follow(\(<\text{term}\rangle\)"
. Follow set is defined by:

\[
\text{follow}(\langle nt \rangle) = \{ u \mid u \text{ is a terminal and } \langle nt' \rangle \Rightarrow^* S \langle nt \rangle u \\
S' \text{ for some non-terminal } \langle nt' \rangle \text{ and strings } S \text{ and } S' \}
\]

This is a horrible definition which tells us little about follow sets. The next section might help.

Computing Follow Sets

We only compute follow sets for a single non-terminal at a time. Say \(<\text{nt}\rangle\) is a non-terminal and we wish to compute follow(\(<\text{nt}\rangle\)).
We use the following recursive algorithm:

for each RHS in which \(<\text{nt}\rangle\) appears
let S be all of the RHS following \(<\text{nt}\rangle\). include all of 
first(S) in follow(\(<\text{nt}\rangle\));
if S can derive \(\varepsilon\) (often it is just \(\varepsilon\))
you must include follow(\(<\text{nt}'\rangle\)) where \(<\text{nt}'\rangle\) is the LHS of 
the production whose RHS you are looking at.

In the preceding section we found that in order to compute the predict set Set B we needed follow(\(<\text{term\_tail}\rangle\)). Using the above algorithm we saw that \(<\text{term\_tail}\rangle\) appeared in the RHSs of productions 3 and 4. In neither RHS did anything follow \(<\text{term\_tail}\rangle\) (i.e. "S" from the algorithm was \(\varepsilon\)) so in both productions we had to do the recursive step of computing the follow sets of the LHSs.

The LHS of production 4 is \(<\text{term\_tail}\rangle\) telling us that computing \(<\text{term\_tail}\rangle\) requires computing \(<\text{term\_tail}\rangle\). This leads to an endless recursion so we prudently stepped aside and let it go.

The LHS of production 3 is \(<\text{term}\rangle\) so the recursive step says we must compute follow(\(<\text{term}\rangle\)) and include its values into follow(\(<\text{term\_tail}\rangle\)). So we did. This led us to compute first(\(<\text{expr\_tail}\rangle\$) from production 1 and first(\(<\text{expr\_tail}\rangle\)
from production 2 and since \(<\text{expr\_tail}\rangle\) can derive \(\varepsilon\) the recursion led us to compute follow(\(<\text{expr\_tail}\rangle\)) (LHS of
production 2). Which let us to ... well, you know.

Predict Sets

Now we can say what the predict sets are and how they can be found. For the production

<nt> → RHS1 | RHS2 | ... | RHSn

you can define a predict set for each RHS:

\[
predict(<nt> → RHS_i) =
\begin{align*}
& \text{first}(RHS_i) \text{ if } RHS_i \text{ cannot derive } \varepsilon \\
& \text{or } \text{first}(RHS_i) \cup \text{follow}(<nt>) \text{ if } RHS_i \text{ can derive } \varepsilon
\end{align*}
\]

A parse function for <nt> is now:

nt()
{
    if (lookahead ∈ predict(<nt> → RHS1))
        parse RHS1
    else if (lookahead ∈ predict(<nt> → RHS2))
        parse RHS2
    ...
    else if (lookahead ∈ predict(<nt> → RHSn))
        parse RHSn
    else
        parse_error();
}