**Eliminate Left Recursion.**

E::= T | E + T | E - T

T::= F | T \* F | T / F

F::= 

Rules for *E* and *T* have left recursive options. The algorithm below eliminates direct left recursion while preserving the meaning of the syntax description:

**Input:**  Left Recursive Syntax Rule in a form wherein left recursive options

precede non-left recursive options:

B ::= B1 | B2 | ... | Bm | m+1 | m+2 |... | m+n ;

**Output:**  Two rules which are not directly left recursive but are equivalent to the

input rule.

**Procedure:**

Replace B with two productions:

B ::= m+1B1 | m+2 B1 |... | m+n B1

B1::= 1B1 | 2B1 | ... | m B1 | ∈

Applying this algorithm to the grammar above we obtain:

E ::= T E1 $

E1 ::= + T E1

::= - T E1

::=

T ::= F T1

T1 ::= \* F T1

::= / F T1

::=

F ::= id

Occasionally, the result of the algorithm above includes obvious indirect recursive such as that found in:

E ::= EP | P E ::= PE'

P ::= ET | T | (E) | () *Eliminate Left Recursion* E'::= PE' | ∈

T ::= [E] | [] P ::= ET | T | (E) | ()

T ::= [E] | []

In such cases, it is necessary to perform a **forward** **substitution** of the RHS options of the recursive symbol. These RHS options replace the recursive symbol in the subsequent production's RHS prefix. For example, in the modified grammar above (i.e., the one on the right), *E* leads to *PE'* which can lead to *ET.* Therefore, the RHS options of *E* will replace *E* in the RHS option of *P:*

E ::= PE'

E'::= PE' | ∈

P ::= PE'T | T | (E) | ()

T ::= [E] | []

Notice that the elimination of the indirect left recursion has led to direct left recursion. Therefore, we need to reapply the algorithm to eliminate direct left recursion:

E ::= PE'

E'::= PE' | ∈

P ::= TP' | (E)P' | ()P'

P'::= E'TP' | ∈

T ::= [E] | []

At this point there is no remaining left recursion (direct or indirect). However, there are common prefixes in two sets of RHS options. E.G., the rule *P ::= TP' | (E)P' | ()P'* has the common prefix *(* in the second and third *RHS* options.

E ::= PE'

E'::= PE' | ∈

**P ::= TP' | (E)P' | ()P'**

P'::= E'TP' | ∈

**T ::= [E] | []**

**Input:**  Common prefix rule:

B ::= 1 | 2 | ... | m | m+1 | m+2 |... | m+n ;

**Output:**  Two rules which have no common prefixes.

**Procedure:**

Replace B with two productions:

B ::= B1 | m+1 | m+2 |... | m+n

B1::= 1 | 2 | ... | m

E ::= PE'

E'::= PE' | ∈

**P ::= TP' | (P"**

**P"::= E)P' | )P'**

P'::= E'TP' | ∈

**T ::= [T"**

**T"::= E] | ]**

**Selection Set Generation.**

E ::= T E1 $

E1 ::= + T E1

::= - T E1

::=

T ::= F T1

T1 ::= \* F T1

::= / F T1

::=

F ::= id

In order to generate the selection set, one must determine the grammar's **first** and **follow sets**. These sets lead to the selection set.

1. **Determine the first set.** A first set contains all vocabulary symbols that can be generated from the leftmost symbols of a rule's RHS options. For example, in performing a leftmost derivation beginning with symbol *E* one sees that one can derive only the vocabulary symbol *id:*

*E ⇒+* *id E1 $*

Thus, {id} is *E's* first set. The following sets are obtained in a similar fashion:

**RULES FIRST SET**

E ::= T E1 $ {id}

E1 ::= + T E1 {+}

::= - T E1 {-}

::=

T ::= F T1 {id}

T1 ::= \* F T1 {\*}

::= / F T1 {/}

::=

F ::= id {id}

2. **Determine the follow set.** There are three steps in generating the follow sets.

a. Set the follow set of the start symbol to {$}

b. For each RHS A, add non-∈ elements of FIRST() to FOLLOW(A)

The application of step 2 results in the following intermediate follow sets:

**RULES FIRST SET FOLLOW SET**

E ::= T E1 $ {id} {$}

E1 ::= + T E1 {+} {$}

::= - T E1 {-}

::=

T ::= F T1 {id} {+,-}

T1 ::= \* F T1 {\*}

::= / F T1 {/}

::=

F ::= id {id} {\*,/}

The rule with LHS *E1* obtains *$* because *A* corresponds to *TE1$* so that the *FIRST($),* which is *$* (i.e., the FIRST of a terminal is the terminal), is added to the *FOLLOW(E1).* This substitution is based upon the following matches: * = T, A = E1,* and ** *= $.* The final step is an iterative step until no changes to follow sets occur.

c. repeat

for each rule B::=A, where * ⇒\** *∈* , add FOLLOW(B) to FOLLOW(A)

until no changes.

Notice that in zero steps of every production (B) there is an epsilon. Therefore, if any RHS ends with a nonterminal symbol *A,* then *A's* associated production will have, added to its follow set, the FOLLOW(B):

**RULES FIRST SET FOLLOW SET**

E ::= T E1 $ {id} {$}

E1 ::= + T E1 {+} {$}

::= - T E1 {-}

::=

T ::= F T1 {id} {+,-}

T1 ::= \* F T1 {\*} {+,-}

::= / F T1 {/}

::=

F ::= id {id} {\*,/}

Both *E1* and *T1* can produce epsilon. Therefore, FOLLOW(E) is added to FOLLOW(T), FOLLOW(T) is added to (F), and FOLLOW(T1) is added to FOLLOW(F):

**RULES FIRST SET FOLLOW SET**

E ::= T E1 $ {id} {$}

E1 ::= + T E1 {+} {$}

::= - T E1 {-}

::=

T ::= F T1 {id} {+,-,$}

T1 ::= \* F T1 {\*} {+,-,$}

::= / F T1 {/}

::=

F ::= id {id} {\*,/,+,-,$}

3. **Determine the selection sets**. The selection set of a non-epsilon RHS options is the option's FIRST set. The selection set of an epsilon option is the rule's follow set:

**RULES FIRST SET FOLLOW SET SELECTION**

**SET**

E ::= T E1 $ {id} {$} {id}

E1 ::= + T E1 {+} {$} {+}

::= - T E1 {-} {-}

::= {$}

T ::= F T1 {id} {+,-,$} {id}

T1 ::= \* F T1 {\*} {+,-,$} {\*}

::= / F T1 {/} {/}

::= {+,-,$}

F ::= id {id} {\*,/,+,-,$} {id}

Once the selection sets are generated the full specification for a recursive descent Syntax Analyzer exists. The selection sets provide the intelligence needed by syntax analysis in order to apply an empty production correctly. If all other RHS's have been attempted and the next input symbol is an element of the empty production's selection set, then it is legal to apply the empty production. For a given production (e.g., E1) the intersection of selection sets is null (e.g., *{+}∩ {-} ∩ {$} = * ). The null intersection provides for determinism in the application of RHS's.