CST 320  Grammar Modifications

Remove Left Recursion

There are two types of left recursion - direct vs. Indirect

Examples:  S -> Sa  (direct)
            S -> aA, A-> S  (indirect, S derives A and A derives S)

1. To remove direct left recursion
   a. Separate the productions into 2 group, left-recursive and non-left-recursive.
      Ex.:   A -> x | y        (non-left-recursive)
              A -> Aa | Ab   (left-recursive)
   b. Introduce a new nonterminal (A’)
   c. Append the new nonterminal (A’) onto the non-left-recursive rules

      A -> xA’ | yA’
   d. Rewrite the left-recursive productions in terms of A’. Drop the recursive A and append the A’
      at the end. Add also the lambda production

      A’ -> aA’ | bA’ | lambda

2. To remove indirect left recursion
   a. List the non-terminal in order and go through the production rules in order.
   b. Whenever a cycle has been detected, (eg. S -> aA, A-> S), rewrite the rule which caused the
      cycle.
      Ex. Since the production rule A-> S caused the cycle to be formed between S and A,
      substitute S with S’s productions. Thus the new A production is  A-> aA

   Example:  S-> aA | b | cS
              A -> Sd | e

      Replace A -> Sd  with  3 additional rules because S has 3 rules. The new grammar
      is:

      S -> aA | b | cS
      A -> aAd | bd | cSd | e  <= Note cSd is not left-recursive

Left-factoring

The idea behind left-factoring is to eliminate choices. If it’s not clear which of the alternative
productions to use to expand a non-terminal A, we may be able to rewrite the A productions to
defer the decision until we have seen enough of the input to make the right choice.

General Form:  A-> xB | xC
       rewrite it to

       A -> xA’
       A’ -> B | C

Example:  ST -> if E then ST else ST
          ST -> if E then ST

      Left factor the above grammar to:

      ST -> if E then St  ST’
      ST’ -> lambda | else ST
Remove lambda-productions

The idea behind removing all lambda productions is to generate all possibilities where a non-terminal could be a lambda (nothing)

Example:
\[
S \rightarrow ABaC \\
A \rightarrow BC \\
B \rightarrow b \mid \text{lambda} \\
C \rightarrow D \mid \text{lambda} \\
D \rightarrow d
\]

Since B and C could go to lambda, A could also go to lambda. Look for all the places where A, B and C are used on the right hand side of the productions, generate all possible combinations where A, B or C could go to lambda.

The new grammar:
\[
S \rightarrow ABaC \mid BaC \mid AaC \mid Aba \mid aC \mid Aa \mid Ba \mid a \\
A \rightarrow BC \\
B \rightarrow b \\
C \rightarrow D \\
D \rightarrow d
\]

Remove unit-productions

A \rightarrow B (if A and B are both single nonterminals, this is an unit production)

General form:

A \rightarrow B
B \rightarrow xyz \mid abc

Simple replace A with B’s rules

A \rightarrow xyz \mid abc

In the case where circular references occur, generate a dependency graph and find out the derivation relationship.

Example: S \rightarrow Aa \mid B
B \rightarrow A \mid bb
A \rightarrow a \mid bc \mid B

From the dependency graph above, we can construct the set of derivation relationships:

S \rightarrow B
S \rightarrow A
A \rightarrow B
B \rightarrow A

Procedure for removing unit productions:
1. Break the productions into two groups, unit productions and non-unit-productions.
2. Keep all the non-unit-productions.
   
   \[
   \begin{align*}
   S & \rightarrow Aa \\
   B & \rightarrow bb \\
   A & \rightarrow a \mid bc
   \end{align*}
   \]
3. For each derivation relationship, add the non-unit-productions.

   e.g. Since \( S \rightarrow B \), add \( B \)'s non-unit-productions to \( S \).

   \[
   S \rightarrow Aa \mid bb
   \]

   Since \( S \rightarrow A \), add \( A \)'s non-unit-productions to \( S \) as well.

   \[
   S \rightarrow Aa \mid bb \mid a \mid bc
   \]

   Similarly, add \( A \)'s non-unit-productions to \( B \) and \( B \)'s non-unit-productions to \( A \).

   \[
   \begin{align*}
   A & \rightarrow a \mid bc \mid bb \\
   B & \rightarrow bb \mid a \mid bc
   \end{align*}
   \]