Eliminating left recursion

- An algorithm to eliminate arbitrary left recursion (by replacing it with right recursion) is as follows:

1. Arbitrarily order the non-terminals: N_1, N_2, N_3, ...
2. Apply the following steps to the productions for N_1, then N_2, ...
3. For N_i:
   a) For all productions $N_i \to N_k\ \alpha$, where $k < i$ and if the productions for N_k are $N_k \to \beta_1 \mid \beta_2 \mid \beta_3 \mid \ldots$ then expand the reference to N_k, i.e. replace the production $N_i \to N_k\ \alpha$ by $N_i \to \beta_1 \alpha \mid \beta_2 \alpha \mid \ldots$
   b) If the productions for N_i are now

      $N_i \to \alpha_1 \mid \alpha_2 \mid \ldots \mid N_i\ \beta_1 \mid N_i\ \beta_2 \mid \ldots$

      (where the first few are not left recursive while the latter are) then replace them with

      $N_i \to \alpha_1\ N_i' \mid \alpha_2\ N_i' \mid \ldots$

      $N_i' \to \varepsilon \mid \beta_1\ N_i' \mid \beta_2\ N_i' \mid \ldots$

Example of eliminating left recursion

- Consider the productions:
  $A \to a \mid Ba$ $\quad B \to b \mid Cb$ $\quad C \to c \mid Ac$

1. Arbitrarily order the non-terminals: A, B, C
2. Consider the productions for A: no change
2. Consider the productions for B: no change
3. Consider the productions for C:
   a) Replace $C \to Ac$ by $C \to ac \mid Bac$
   a) Replace $C \to Bac$ by $C \to bac \mid Cbac$

   Productions for C are now: $C \to c \mid ac \mid bac \mid Cbac$

   b) Replace the productions for C by:

      $C \to cC' \mid acC' \mid bacC'$

      $C' \to \varepsilon \mid bacC'$

A Workable Solution

Observation

- The trouble which gives rise to nondeterminacy and backtracking in top down parsers shows itself in only one place – that is when a parser has to choose between several alternatives with the same left hand side.
- The only information which we can use to make the *correct decision* is the input stream itself.
  - —In the example, we (humans) could see which alternative to choose by looking at the input yet-to-be-read.
- If we are going to *look ahead* in order to make the correct decision, we need a buffer in which to store the next few symbols.
- In practice, this buffer is of a fixed length.

Definitions

- A parser which can make a deterministic decision about which alternative to choose when faced with one, if given a buffer of k symbols, is called a LL(*k*) parser.
  - —Left to right scan of input
  - —Left most derivation
  - —*k* symbols of look-ahead
- The grammar that an LL(k) parser recognizes is an LL(k) grammar and any language that has an LL(k) grammar is an LL(k) language.
  - —We are constructing an LL(1) compiler that recognises LL(1) grammars.
  - —So the question is *How do we know when we have an LL(1) grammar?*
- We also have LR(*k*) grammars and other variations, but our focus is currently on LL(1) grammars.

