Motivating example

- In this lecture we discuss techniques (that sometimes work) to convert a grammar that is not LL(1) into an equivalent grammar that is LL(1).

Left and right recursion continue

- In general, rules of the form
  $A \longrightarrow A\alpha \mid \beta$ are called left recursive
  and rules of the form
  $A \longrightarrow \alpha A \mid \beta$ right recursive.
- Grammars equivalent to the regular expression a^* are given by pause $A \longrightarrow Aa \mid \varepsilon$ or $A \longrightarrow aA \mid \varepsilon$

Left recursion removal continue

- If we remove the left recursion from the rule

$exp \longrightarrow exp \ + \ term \mid exp \ - \ term \mid term$

we obtain

Left Factoring

- Left factoring is required when two or more grammar rule choices share a common prefix string, as in the rule

Left Factoring

- Left factoring is required when two or more grammar rule choices share a common prefix string, as in the rule

  $A \rightarrow \alpha\ \beta \mid \alpha\ \gamma$

Left Factoring

- Left factoring is required when two or more grammar rule choices share a common prefix string, as in the rule

  $A \rightarrow \alpha\ \beta \mid \alpha\ \gamma$

- Obviously, an LL(1) parser cannot distinguish between the production choices in such a situation.
- In the following example we have exactly this problem:

Left factoring continue

- Consider the following grammar of if-statements:

  if-stmt $\longrightarrow$ **if** (*exp*) *statement* | **if** (*exp*) *statement* **else** *statement*

  The left factored form of this grammar is

Left factoring continue

- Here is a typical example where a programming language fails to be LL(1):

statement $\longrightarrow$ *assign-stmt* | *call-stmt* | **other**
assign-stmt $\longrightarrow$ **identifier** := *exp*
call-stmt $\longrightarrow$ **identifier** (*exp-list*)

- This grammar is not in a form that can be left factored. We must first replace *assign-stmt* and *call-stmt* by the right-hand sides of their defining productions:

Left factoring continue

- Here is a typical example where a programming language fails to be LL(1):

statement $\longrightarrow$ *assign-stmt* | *call-stmt* | **other**
assign-stmt $\longrightarrow$ **identifier** := *exp*
call-stmt $\longrightarrow$ **identifier** (*exp-list*)

- This grammar is not in a form that can be left factored. We must first replace *assign-stmt* and *call-stmt* by the right-hand sides of their defining productions:

statement $\longrightarrow$ **identifier** := *exp* | **identifier** (*exp-list*) | **other**

- Then we left factor to obtain:

Left factoring continue

- Here is a typical example where a programming language fails to be LL(1):

statement $\longrightarrow$ *assign-stmt* | *call-stmt* | **other**
assign-stmt $\longrightarrow$ **identifier** := *exp*
call-stmt $\longrightarrow$ **identifier** (*exp-list*)

- This grammar is not in a form that can be left factored. We must first replace *assign-stmt* and *call-stmt* by the right-hand sides of their defining productions:

statement $\longrightarrow$ **identifier** := *exp* | **identifier** (*exp-list*) | **other**

- Then we left factor to obtain:

statement $\longrightarrow$ **identifier** *statement*$'$ | **other**
statement$'$ $\longrightarrow$:= *exp* | (*exp-list*)