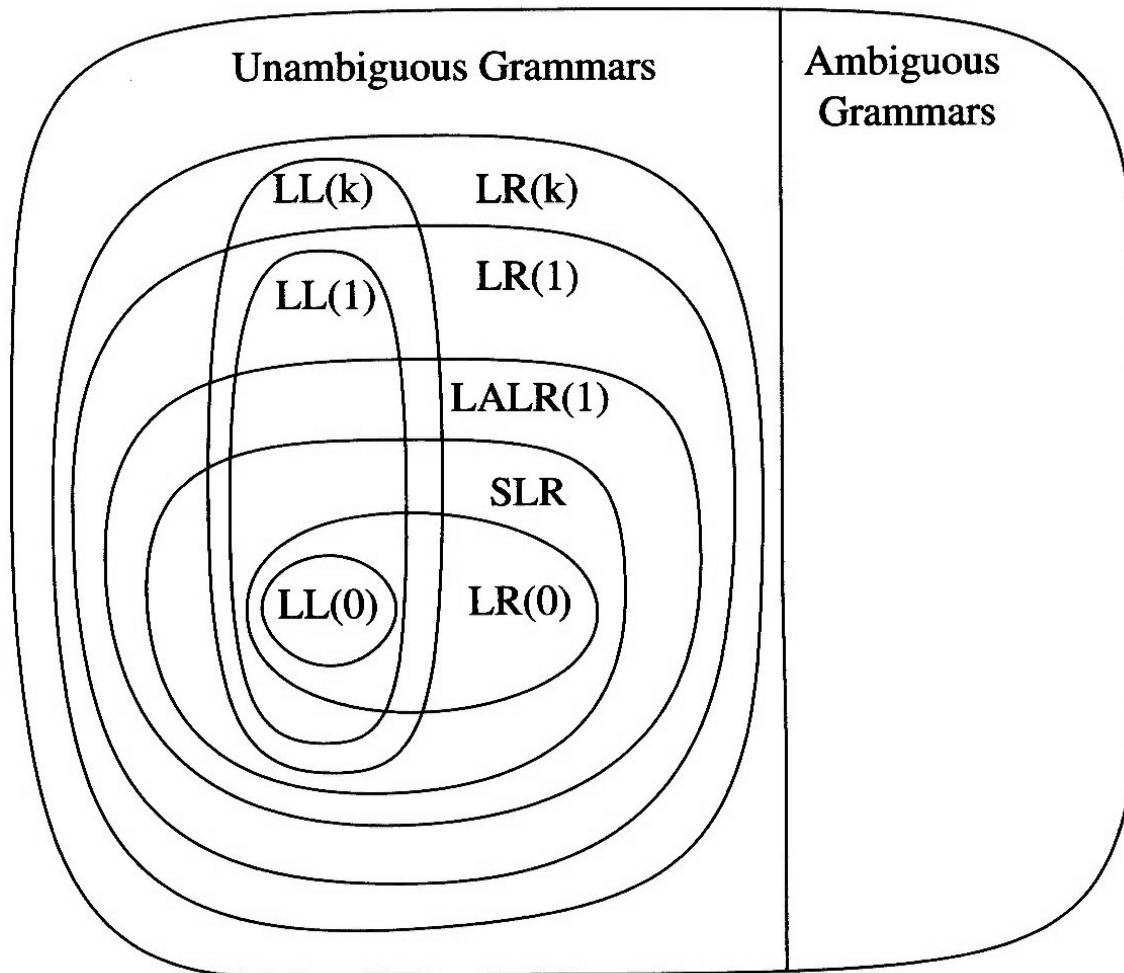




Unit 8.

LL(k) grammars

Hierarchy of grammar classes





LL(k) grammar

■ What is LL(k)?

- The first *L* stands for scanning the input from **l**eft to right,
- the second *L* stands for producing a **l**eftmost derivation,
- And *k* stands for using **k** input symbols of lookahead at each step to make parsing action decision.



LL(k) Grammar

- Subset of CFG's
- Permits deterministic left-to-right recognition with a look ahead of k symbols
- Builds the parse tree top-down
- If the correct production can be deduced from the partially constructed tree and the next k symbols in the unscanned string, for every possible step, then the grammar is said to be LL(k)
- If a parse table can be constructed for the grammar, then it is LL(k), if it can't, it is not LL(k)



LL(k) Grammars

- An LL(k) grammar has the property that a parser can be constructed to scan an input string from left to right and build a leftmost derivation by examining next k input symbols to determine the unique production for each derivation step.
- If a language has an LL(k) grammar, it is called an LL(k) language.
- LL(k) languages are **deterministic context-free languages**, but there are deterministic context-free languages that are not LL(k)



How to Build Parse Tables?

FIRST and FOLLOW Sets

For a string of grammar symbols α define
FIRST(α) as

- The set of tokens that appear as the first symbol in some string that derives from α
- If $\alpha \Rightarrow^* \varepsilon$, then ε is in FIRST(α)

For a non-terminal symbol A , define FOLLOW(A)
as

The set of terminal symbols that can appear immediately to the right of A in some sentential form



FIRST Set Construction

To construct $\text{FIRST}(X)$ for a grammar symbol X , apply the following rules until no more symbols can be added to $\text{FIRST}(X)$

- If X is a terminal $\text{FIRST}(X)$ is $\{X\}$
- If $X \rightarrow \varepsilon$ is a production then ε is in $\text{FIRST}(X)$
- If X is a nonterminal and $X \rightarrow Y_1 Y_2 \dots Y_k$ is a production then put every symbol in $\text{FIRST}(Y_1)$ other than ε to $\text{FIRST}(X)$
- If X is a nonterminal and $X \rightarrow Y_1 Y_2 \dots Y_k$ is a production, then put terminal a in $\text{FIRST}(X)$ if a is in $\text{FIRST}(Y_i)$ and ε is in $\text{FIRST}(Y_j)$ for all $1 \leq j < i$
- If X is a nonterminal and $X \rightarrow Y_1 Y_2 \dots Y_k$ is a production, then put ε in $\text{FIRST}(X)$ if ε is in $\text{FIRST}(Y_i)$ for all $1 \leq i \leq k$



Computing FIRST Sets for Strings of Symbols

To construct the FIRST set for any string of grammar symbols $X_1X_2 \dots X_k$ (given the FIRST sets for symbols X_1, X_2, \dots, X_k) apply the following rules.

$\text{FIRST}(X_1X_2 \dots X_k)$ contains:

- Any symbol in $\text{FIRST}(X_1)$ other than ε
- Any symbol in $\text{FIRST}(X_i)$ other than ε , if ε is in $\text{FIRST}(X_j)$ for all $1 \leq j < i$
- ε , if ε is in $\text{FIRST}(X_j)$ for all $1 \leq i \leq n$



$\text{FIRST}_k(\alpha)$

Description: Given a CFG, k is a natural number, α contains both terminals and nonterminals.

$\text{FIRST}_k(\alpha)$ consists of all terminal prefixes of length k (or less if α derives a terminal string of length less than k) of terminal strings that can be derived from α

Definition: Given grammar $G = (\Sigma, \Delta, P, S)$, natural number k , $\alpha \in V^*$

$\text{FIRST}_k(\alpha) =$

$\{ x \in \Sigma^* \mid \alpha \Rightarrow x\beta \text{ and } |x| = k \text{ or } \alpha \Rightarrow x \text{ and } |x| < k \}$



$\text{FOLLOW}_k(\alpha)$

includes the set of terminal strings that can occur immediately to the right of α in any sentential form

Especially, if α is nonterminal A and βA is a sentential form then ε is in $\text{FOLLOW}_1(A)$.



$\text{FOLLOW}_k(\alpha)$

$\text{FOLLOW}_k(\alpha) =$

$\{x \in \Sigma^* \mid S \Rightarrow^* \beta\alpha\delta \text{ and } x \in \text{FIRST}_k(\delta)\}$

Especially, if $\alpha = A \in \Delta^*$, $S \Rightarrow^* \beta A$ then
 $\text{FOLLOW}_1(A) = \{\varepsilon\}$ or $\text{FOLLOW}_1(A) = \{\$ \}$ (EOF)

$\text{FOLLOW}(A)$ is $\text{FOLLOW}_1(A)$



LL(k) Grammars

Definition Let $G = (\Sigma, \Delta, P, S)$ is a CFG and $k \in \mathbb{N}$. G is LL(k) if for any two leftmost derivations

$$S \Rightarrow xA\alpha \Rightarrow x\beta_1\alpha \Rightarrow xZ_1$$

$$S \Rightarrow xA\alpha \Rightarrow x\beta_2\alpha \Rightarrow xZ_2$$

if $\text{FIRST}_k(Z_1) = \text{FIRST}_k(Z_2)$ then $\beta_1 = \beta_2$

It can be shown that LL(k) grammars are **not ambiguous** and **not left-recursive**.




Example

Grammar G :

$$S \rightarrow aSb \mid ab$$

is not LL(1), but is LL(2)



Simple LL(1) Grammars

For simple LL(1) grammars all rules have the form

$$A \rightarrow a_1\alpha_1 \mid a_2\alpha_2 \mid \dots \mid a_n\alpha_n$$

where

- a_i is a terminal, $1 \leq i \leq n$
- $a_i \neq a_j$ for $i \neq j$ and
- α_i is a sequence of terminals and non-terminal or is empty, $1 \leq i \leq n$



How to recognize a LL(1) grammar?

Theorem A context-free grammar $G = (\Sigma, \Delta, P, S)$ is LL(1) if and only if for every nonterminal A and every strings of symbols

$A \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_n, n \geq 2$ we have

$$\text{FIRST}_1(\alpha_i) \cap \text{FIRST}_1(\alpha_j) = \emptyset, i \neq j$$

If $\alpha_i \Rightarrow^* \varepsilon$ then

$$\text{FIRST}_1(\alpha_i) \cap \text{FOLLOW}_1(A) = \emptyset, i \neq j$$

KPL is LL(1)?– FIRST & FOLLOW

A	FIRST(A)	FOLLOW(A)
Block	KW_CONST, KW_VAR, KW_TYPE, KW_FUNCTION, KW_PROCEDURE, KW_BEGIN	SB_PERIOD, SB_SEMICOLON
Unsignedconst	TK_IDENT, TK_NUMBER, TK_CHAR.	
Constant	SB_PLUS,SB_MINUS, TK_IDENT TK_NUMBER	
Type	TK_IDENT, TK_NUMBER, TK_CHAR, KW_ARRAY	
Statement	TK_IDENT, KW_CALL, KW_BEGIN, KW_IF, KW_WHILE, KW_FOR	KW_ELSE,SB_SEMICOLON, KW_END
Expression3	SB_PLUS,SB_MINUS, ϵ	SB_COMMA,SB_SEMICOLON, KW_END, KW_TO, KW_THEN, KW_DO, SB_RPAR, SB_RSEL, SB_EQ, SB_NEQ, SB_LT, SB_LE, SB_GT, SB_GE, KW_ELSE



KPL is LL(1)?

Consider the following set of production with Statement on LHS

```
Statement ::= AssignSt  
Statement ::= CallSt  
Statement ::= GroupSt  
Statement ::= IfSt  
Statement ::= WhileSt  
Statement ::= ForSt  
Statement ::=  $\epsilon$ 
```

FIRST (RHS1)={TK_IDEN}

FIRST (RHS2)={KW_CALL}

FIRST (RHS3)={KW_BEGIN}

FIRST (RHS4)={KW_IF}

FIRST (RHS5)={KW_WHILE}

FIRST (RHS6)={KW_FOR}

FIRST (RHS7)={ ϵ }

FOLLOW(LHS)={SB_SEMICOLON, KW_END, KW_ELSE}

The set of productions of Statement satisfies LL(1) condition



KPL is LL(1)?

```
Factor ::= UnsignedConstant  
Factor ::= Variable  
Factor ::= FunctionApplication  
Factor ::= SB_LPAR Expression SB_RPAR
```

FIRST(RHS1)={TK_IDENT, TK_NUMBER, TK_CHAR}

FIRST(RHS2)={TK_IDENT}

FIRST(RHS3)={TK_IDENT}

FIRST(RHS4)={SB_LPAR}

The set of productions of Factor does not satisfy LL(1) condition, but satisfies LL(2) condition



KPL is LL(2)

```
Factor ::= TK_NUMBER
Factor ::= TK_CHAR
Factor ::= TK_IDENT
Factor ::= TK_IDENT Arguments
Factor ::= TK_IDENT Indexes
Factor ::= SB_LPAR Expression SB_RPAR
```

```
FIRST2(RHS1)={TK_NUMBER}
FIRST2(RHS2)={TK_CHAR}
FIRST2(RHS3)={TK_IDENT}
FIRST2(RHS4)={TK_IDENT SB_LPAR}
FIRST2(RHS5)={TK_IDENT SB_LSEL}
FIRST2(RHS6)={SB_LPAR TK_IDENT, SB_LPAR TK_NUMBER,
SB_LPAR SB_LPAR , SB_LPAR TK_CHAR}
```

The set of productions satisfies LL(2) condition



Grammar Transformations

- Left factoring: Sometimes we can “left-factor” an LL(k) grammar to obtain an equivalent LL(n) grammar where $n < k$.
- Example. The grammar $S \rightarrow aaS \mid ab \mid b$ is LL(2) but not LL(1). But we can factor out the common prefix a from productions $S \rightarrow aaS \mid ab$ to obtain
$$S \rightarrow aT$$
$$T \rightarrow aS \mid b.$$
This gives the new grammar:
$$S \rightarrow aT \mid b$$
$$T \rightarrow aS \mid b.$$



Left Factoring to obtain LL(1) grammars

When more than one production for nonterminal A starts with the same symbols, the FIRST sets are not disjoint

$$\begin{aligned} ifstmt &\rightarrow \mathbf{if} \textit{ cond } \mathbf{then} \textit{ stmt} \\ &\quad | \mathbf{if} \textit{ cond } \mathbf{then} \textit{ stmt } \mathbf{else} \textit{ stmt} \end{aligned}$$

Use *left factoring* to fix the problem

$$\begin{aligned} ifstmt &\rightarrow \mathbf{if} \textit{ expr } \mathbf{then} \textit{ stmt } \textit{ elsestmt} \\ \textit{elsestmt} &\rightarrow \mathbf{else} \textit{ stmt} \\ &\quad | \epsilon \end{aligned}$$