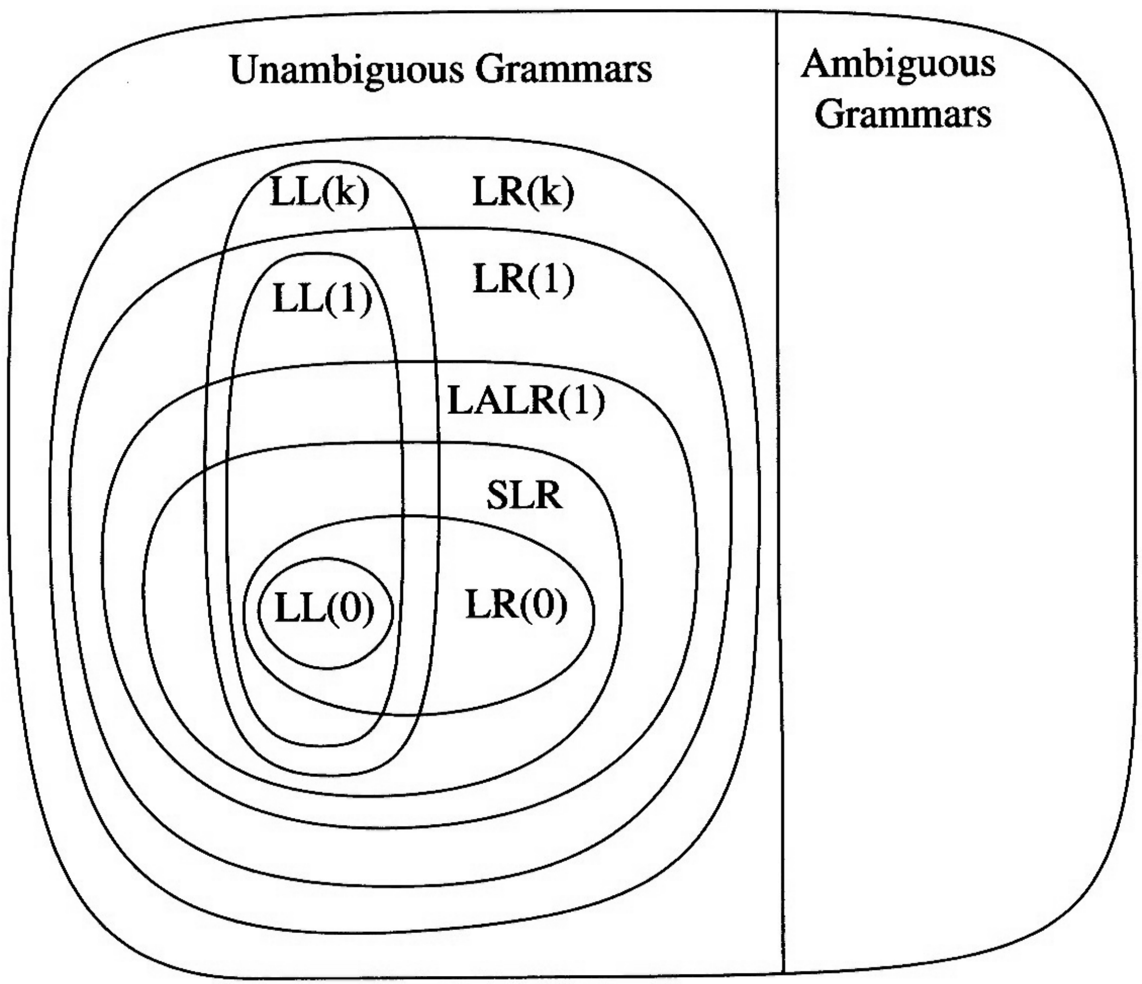
Unit 8.
LL(k) grammars

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) 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)

FIRST$_k$(α)

The **FIRST$_k$** set of a string of symbols in a grammar is a set of k-length strings of terminal symbols that may begin a sentential form derivable from the string of symbols in the grammar. More specifically, for a grammar G = (Σ, Δ,P, S)

FIRST$_k$(α) =

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

$FOLLOW_k(\alpha)$

The **FOLLOW$_k$** set of a string of symbols in a grammar is a set of k-length terminal symbol strings in the grammar that may follow the string of symbols in some sentential form derivable in the grammar.

More specifically, for a grammar

$G = (\Sigma, \Delta, P, S)$:

$FOLLOW_k(\alpha) =$

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

LL(k) Grammars

Definition Let $G = (\Sigma, \Delta, P, S)$ is a CFG and $k \in N$. 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 $FIRST_k(Z_1) = FIRST_k(Z_2)$ then $\beta_1 = \beta_2$

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

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 FIRST(X) for a grammar symbol X, apply the following rules until no more symbols can be added to FIRST(X)

- If X is a terminal FIRST(X) is $\{X\}$
- If $X \rightarrow \varepsilon$ is a prodcution then ε is in FIRST(X)
- If X is a nonterminal and $X \rightarrow Y_1 Y_2 \ldots Y_k$ is a production then put every symbol in FIRST(Y_1) other than ε to FIRST(X)
- If X is a nonterminal and $X \rightarrow Y_1 Y_2 \ldots Y_k$ is a production, then put terminal a in FIRST(X) if a is in FIRST(Y_i) and ε is in FIRST(Y_j) for all $1 \leq j < i$
- If X is a nonterminal and $X \rightarrow Y_1 Y_2 \ldots Y_k$ is a production, then put ε in FIRST(X) if ε is in 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 $X_1X_2 \ldots X_n$ (given the FIRST sets for symbols X_1 , X_2 , ... X_n) apply the following rules.

FIRST($X_1X_2 \ldots X_n$) contains:

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

Example

The following grammar G:

$$S \rightarrow aAS \mid b$$

$$A \rightarrow bSA \mid a$$

is LL(1)

Simple LL(1) Grammars

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

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

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?

<u>*Theorem*</u> A context-free grammar G = (Σ, Δ, P, S) is LL(1) if and if only if for every nonterminal *A* and every strings of symbols

$$A \to \alpha_1 \mid \alpha_2 \mid . . . . \mid \alpha_n \text{ , } n \geq 2 \text{ we have}$$

$$FIRST_1(\alpha_i) \cap FIRST_1(\alpha_j) = \varnothing, \ i \neq j$$

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

$$FIRST_1(\alpha_i) \cap FOLLOW_1(A) = \varnothing \text{ , } i \neq j$$

KPL is nearly LL(1)

A	FIRST(A)	FOLLOW(A)
Block	CONST, VAR,TYPE, PROCEDURE,BEGIN	.,;
Unsignedconst	ident, number,’	
Constant	+,-,’,ident,number	
Type	ident,integer, char,array	
Statement	ident, CALL, BEGIN, WHILE,FOR	.,;, END
Expression	+,-,(,ident,number	.,;, END,TO,THEN,DO,),-,.),<,<=,>,>=,=,!=
Term	ident,number, (	.,;,END,TO,THEN,DO,),-,<,<=,>,>=,=,!=
Factor	ident, number, (	.,;,END,TO,THEN, DO, +, -, *,/,) ,<,<=,>,>=,=,!=

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:

  $S \rightarrow aT \mid b$

  $T \rightarrow aS \mid b$.