

Chapter 16. LL(K) Grammars

Allow top-down k-symbol lookahead parsing.
Scan is left-to-right with leftmost derivation.

E.G. Scan acbb using

$$S \rightarrow a S \mid c A \quad A \rightarrow bb$$

We know $S \rightarrow a S$ must be used first.

The lookahead set; when III holds parsing can be done without search and the grammar is LL(k) for some k.

$$\text{I. } LA(A) = \bigcup_{i=1}^n LA(A \rightarrow w_i)$$

$$\text{II. } LA(A \rightarrow w_i) = \{X | w_i v \xRightarrow{*} X \in \Sigma^* \text{ and } S \xRightarrow{*} uAv\}$$

$$\text{III. } LA(A \rightarrow w_i) \cap LA(A \rightarrow w_j) = \Phi, \quad 1 \leq i < j \leq n$$

So X is a set of prefixes (or infixes) generated by $A \rightarrow w_i$ that are uniquely different from prefixes generated by $A \rightarrow w_j$. Rule III allows us to know when to use $A \rightarrow w_i$.

Example

$$S \longrightarrow A a b d \mid c A b c d$$

$$A \longrightarrow a \mid b \mid \lambda$$

$$S \longrightarrow a a b d \mid b a b d \mid a b d$$

$$S \longrightarrow c a b c d \mid c b b c d \mid c b c d$$

So,

$$LA(S \rightarrow A a b d) = \{a a b d, b a b d, a b d\}$$

$$LA(S \rightarrow c A b c d) = \{c a b c d, c b b c d, c b c d\}$$

$$LA(A \rightarrow a) = \{a a b d, a b c d\}$$

$$LA(A \rightarrow b) = \{b a b d, b b c d\}$$

$$LA(A \rightarrow \lambda) = \{a b d, b c d\}$$

A rule such as " $A \rightarrow a$ " has a lookahead set that includes:

1. What A generates directly.
2. What can follow A .

This grammar can be deterministically parsed in a top down fashion by a 3-symbol lookahead.

Example

$$S \longrightarrow A B C a b c d$$

$$A \longrightarrow a \mid \lambda$$

$$B \longrightarrow b \mid \lambda$$

$$C \longrightarrow c \mid \lambda$$

$$LA(A \rightarrow a) = \{a + \lambda\} \cdot \{LA(B \rightarrow b, B \rightarrow \lambda)\}$$

$$LA(B \rightarrow b) = \{b + \lambda\} \cdot \{LA(C \rightarrow c, C \rightarrow \lambda)\}$$

Perhaps best to compute these lookaheads is reverse order.

$$LA(C \rightarrow c) = \{c a b c d\}$$

$$LA(C \rightarrow \lambda) = \{a b c d\}$$

$$LA(B \rightarrow b) = \{b c a b c d, b a b c d\}$$

$$LA(B \rightarrow \lambda) = \{c a b c d, a b c d\}$$

$$LA(A \rightarrow a) = \{abcabcd, ababcd, acabcd, aabcd\}$$

$$LA(A \rightarrow \lambda) = \{b c a b c d, b a b c d, c a b c d, a b c d\}$$

We don't need a lookahead for $S \rightarrow ABCabcd$

because only 1 rule. C and B require on 1 lookahead.

A requires 4 lookahead symbols because

$LA(A \rightarrow a)$ includes $a b c \underline{a} b c d$

$LA(A \rightarrow \lambda)$ includes $a b c \underline{d}$

Left Factoring

One can factor a grammar to break apart rules. One can also use Griebach Normal Form type transformations. Both can change the lookahead set.

| | <i>Lookahead</i> |
|--------------------------------------|------------------|
| $G1 : S \rightarrow a\underline{S}c$ | $\{a a a\}$ |
| $S \rightarrow a\underline{a} b c$ | $\{a a b\}$ |
| $G2 : S \rightarrow aA$ | |
| $A \rightarrow Sc$ | $\{a a\}$ |
| $A \rightarrow abc$ | $\{a b\}$ |

Expand G2 : $S \rightarrow aA$
 $A \rightarrow aAc$ *Note* $A \rightarrow a_c$
 $A \rightarrow abc$ *Note* $A \rightarrow a_c$

Factor A : $S \rightarrow aA$
 $A \rightarrow aBc$
 $B \rightarrow A$
 $B \rightarrow b$

Remove A : G3 : $S \rightarrow aaBc$
 $B \rightarrow aBc$
 $B \rightarrow b$

If we wish to compute a k-lookahead set, we can use a *trunc_k* function to limit the lookahead.

FIRST_k(A) generates upto k symbols that derive directly from A.
FOLLOW_k(A) generates upto k symbols that follow A.

$$LA_k(A) = trunc_k(FIRST_k(A) FOLLOW_k(A))$$

$$LA_k(A \rightarrow \omega) = trunc_k(FIRST(\omega) FOLLOW_k(A))$$

Example

$$S \rightarrow A a b d \mid c A b c d$$
$$A \rightarrow a \mid b \mid \lambda$$

$$FIRST_3(S) = \{a a b, b a b, a b d, c a b, c b b, c b c\}$$

$$FIRST_3(A) = \{a, b, \lambda\}$$

$$FIRST_3(a) = \{a\}$$

$$FIRST_3(b) = \{b\}$$

$$FIRST_3(c) = \{c\}$$

$$FIRST_3(d) = \{d\}$$

$$FOLLOW_3(S) = \{\lambda\}$$

$$FOLLOW_3(A) = \{a b d, b c d\}$$

$$\begin{aligned}
& LA_3(S \rightarrow A a b d) \\
= & \text{trunc}_3(\text{FIRST}_3(A), \text{FIRST}_3(a), \text{FIRST}_3(b), \text{FIRST}_3(d), \\
& \text{FOLLOW}_3(S)) \\
= & \text{trunc}_3(\{a, b, \lambda\} a b d \lambda) \\
= & \text{trunc}_3(\{a a b, b a b, a b d\})
\end{aligned}$$

16.3 Strong LL(K) Grammars

A grammar G is strong LL(K) *IFF* the sets $LA_k(A \rightarrow \omega_i)$ partitions $LA(A)$ for each variables $A \in V$.

G is strong $LL(K)$ if for any 2 leftmost derivations

$$S \xRightarrow{*} u_1Av_1 \Rightarrow u_1xv_1 \Rightarrow u_1zw_1$$

$$S \xRightarrow{*} u_2Av_2 \Rightarrow u_2yv_2 \Rightarrow u_2zw_2$$

where $u_i, v_i, w_i, z \in \Sigma^*$ and $length(z) = k$ then $x = y$.

ALL this says is: only 1 rule can generate z since it is part of the k -step lookahead. If there is more than 1 way to get z from A , G is NOT $LL(k)$

If G is strong $LL(K)$ for some K , then G is unambiguous.

...since only 1 rule can apply given a K lookahead, there is only 1 parse.

If G has a left recursive variable, then G is not strong $LL(K)$ for any K .

Let $A \rightarrow x$ where x is of any length. Both $A \rightarrow Ay$ and $A \rightarrow x$ have the same K -lookahead terminals in the leftmost position.

The same argument holds if the recursion is indirect:

$$A \Rightarrow By \Rightarrow \dots \Rightarrow Awy$$
$$A \Rightarrow x$$

A strong $LL(1)$ grammer

AE:

$$S \rightarrow A\#$$

$$A \rightarrow T$$

$$A \rightarrow A + T$$

$$T \rightarrow b$$

$$T \rightarrow (A)$$

We add an end of parse marker to the grammer: #.

We remove leftmost recursion. AE_1 :

$$S \rightarrow A^\#$$

$$A \rightarrow T$$

$$A \rightarrow TZ$$

$$Z \rightarrow +T$$

$$Z \rightarrow +TZ$$

$$T \rightarrow b$$

$$T \rightarrow (A)$$

still not $LL(1)$ because $\left\{ \begin{array}{l} Z \rightarrow +T \\ Z \rightarrow +TZ \end{array} \right.$ and $\left\{ \begin{array}{l} A \rightarrow T \\ A \rightarrow TZ \end{array} \right.$

left factor these rules.

AE_2 :

$$S \rightarrow A^\#$$

$$A \rightarrow TB$$

$$B \rightarrow Z$$

$$B \rightarrow \lambda$$

$$Z \rightarrow +TY$$

$$Y \rightarrow Z$$

$$Y \rightarrow \lambda$$

$$T \rightarrow b$$

$$T \rightarrow (A)$$

AE_2 is now strong $LL(1)$

$$\begin{aligned}
LA_1 (S \rightarrow A\#) &= \{b, (\} \\
LA_1 (A \rightarrow TB) &= \{b, (\} \\
LA_1 (B \rightarrow Z) &= \{+\} \\
LA_1 (B \rightarrow \lambda) &= \{\#,)\} \\
LA_1 (Z \rightarrow +TY) &= \{+\} \\
LA_1 (Z \rightarrow \lambda) &= \{\#,)\} \\
LA_1 (Y \rightarrow Z) &= \{+\} \\
LA_1 (Y \rightarrow \lambda) &= \{\#,)\} \\
LA_1 (T \rightarrow b) &= \{b\} \\
LA_1 (T \rightarrow (A)) &= \{(
\end{aligned}$$

Since the rules are disjoint the grammar is strong $LL(1)$. parsing is now simple since we can look ahead 1 symbol and know what to do.

16.8 LL(K) Grammars that are not strong LL(K)

A grammar that is $LL(K)$ but not strong $LL(K)$ the correct parse rule can be selected by looking at local context.

G is $LL(K)$ if when there are 2 leftmost derivations

$$S \xRightarrow{*} uAv \Rightarrow uxv \xRightarrow{*} uz w_1$$

$$S \xRightarrow{*} uAv \Rightarrow uyv \xRightarrow{*} uz w_2$$

when $u w_i \in \Sigma^*$ and $length(z) = k$ then $x = y$.

In the same context the choice of rule is determined.

E.G.

$$S \rightarrow A a b d \mid c A b c d$$

$$A \rightarrow a \mid b \mid \lambda$$

contexts of A: $AAbd$, $aAbcd$

$$LA_2(S, S \rightarrow A a b d) = \{a a, b a, a b\}$$

$$LA_2(S, S \rightarrow c A b c d) = \{c a, c b\}$$

$$LA_2(A a b d, A \rightarrow a) = \{a a\}$$

$$LA_2(A a b d, A \rightarrow b) = \{ba\}$$

$$LA_2(A a b d, A \rightarrow \lambda) = \{a b\}$$

$$LA_2(c a b c d, A \rightarrow a) = \{a b\}$$

$$LA_2(c a b c d, A \rightarrow b) = \{b b\}$$

$$LA_2(c a b c d, A \rightarrow \lambda) = \{b c\}$$