# Chapter 19

## LL(k) Grammars

# LL(k) Parsers

- Can be developed using PDAs for parsing CFGs by converting the *machines* directly into *program statements*

- Describe the parsing strategy:

  i) the input string is scanned in a left-to-right manner

  ii) the parsers generate a leftmost derivation, and

  iii) a *deterministic top-down* parsing using a *k*-symbol lookahead, attempting to construct a leftmost derivation of an *input string*

  - The lookahead principle can be used to construct programs that overcome the *non-determinism* found in some PDA.

# The Lookahead principle

- Converting the *non-deterministic transitions* into the *deterministic program segments*
- *Predicts* which one of the several production rules (in an *unambiguous* CFG) should be used to process the remaining input symbols
- Example. Consider a derivation of the string *acbb* using *G*:

$S \rightarrow aS \mid cA$ $\quad A \rightarrow bA \mid cB \mid \lambda$ $\quad B \rightarrow cB \mid a \mid \lambda$

  - Comparing the lookahead (input) symbol with the terminal symbol in each of the appropriate production rules permits the deterministic construction of each derivation in *G*

| Prefix Generated | Lookahead Symbol | Production Rule | Derivation |
|---|---|---|---|
| $\lambda$ | *a* | $S \rightarrow aS$ | $S \Rightarrow aS$ |
| *a* | *c* | $S \rightarrow cA$ | $\Rightarrow acA$ |
| *ac* | *b* | $A \rightarrow bA$ | $\Rightarrow acbA$ |
| *acb* | *b* | $A \rightarrow bA$ | $\Rightarrow acbbA$ |
| *acbb* | $\lambda$ | $A \rightarrow \lambda$ | $\Rightarrow acbb$ |

# Lookahead Strings and Lookahead Sets

- Let $p$ be a terminal string. An intermediate step in a derivation of $p$ has the form $S \overset{*}{\Rightarrow} uAv$, where $p = ux$. The string $x$ is called a **lookahead string** for the variable $A$. The **lookahead set** of $A$ consists of all *lookahead strings* for $A$.

- Defn. 19.1.1. Let $G = (V, \Sigma, P, S)$ be a CFG and $A \in V$

  i) The **lookahead set** of the *variable* $A$, LA($A$), is defined by

  $$LA(A) = \{ x \mid S \overset{*}{\Rightarrow} uAv \overset{*}{\Rightarrow} ux \in \Sigma^* \}$$

  ii) For each rule $A \to w$ in $P$, the **lookahead set** of the *rule* $A \to w$ is defined by

  $$LA(A \to w) = \{ x \mid wv \overset{*}{\Rightarrow} x, \text{ where } x \in \Sigma^* \wedge S \overset{*}{\Rightarrow} uAv \}$$

  Note: $LA(A \to w) \subseteq LA(A)$ such that $LA(A \to w)$ dictates the derivations $Av \overset{*}{\Rightarrow} x$, which are initiated with the rule $A \to w$

# Lookahead Strings and Lookahead Sets

- Example 19.1.4.

| Grammar | Rule | # of lookahead symbols to be considered |
|---|---|---|
| $G_1$: | $S \rightarrow aSc \mid aabc$ | 3: *aaa…, aab…* |
| $G_2$: | $S \rightarrow aA$<br>$A \rightarrow Sc \mid abc$ | 2 (for $A$): *aa…, ab…* |
| $G_3$: | $S \rightarrow aaAc$<br>$A \rightarrow aAc \mid b$ | 1 (for $A$): *a…, b…* |

- Example 19.1.2. $G_2$: $S \rightarrow ABCabcd$, $A \rightarrow a \mid \lambda$, $B \rightarrow b \mid \lambda$, $C \rightarrow c \mid \lambda$

  LA($S$) = { *abcabcd*, *ababcd*, *acabcd*, *bcabcd*, *aabcd*, *babcd*, *cabcd*, *abcd* }

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

  LA($A \rightarrow \lambda$) = { *bcabcd*, *babcd*, *cabcd*, *abcd* }

  LA($B \rightarrow b$) = { *bcabcd*, *babcd* }

  LA($B \rightarrow \lambda$) = { *cabcd*, *abcd* }

  LA($C \rightarrow c$) = { *cabcd* }

  LA($C \rightarrow \lambda$) = { *abcd* }

# Lookahead Sets in CFGs

- Example 19.1.1. Given the following grammar $G_1$:

$$S \rightarrow Aabd \mid cAbcd$$
$$A \rightarrow a \mid b \mid \lambda$$

LA($S$) = { *aabd*, *babd*, *abd*, *cabcd*, *cbbcd*, *cbcd* }

LA($S \rightarrow Aabd$) = { *aabd*, *babd*, *abd* } /* 1st symbol: { a, b } */

LA($S \rightarrow cAbcd$) = { *cabcd*, *cbbcd*, *cbcd* } /* 1st symbol: { c } */

- We can select the appropriate $S$ rule above using the 1st symbol of the LA strings.

LA($A \rightarrow a$) = { *aabd*, *abcd* } /* 2nd symbol: { *aa*, *ab* }; 3rd symbol: { *aab*, ***abc*** } */

LA($A \rightarrow b$) = { *babd*, *bbcd* } /* 2nd symbol: { *ba*, *bb* }; 3rd symbol: { *bab*, *bbc* } */

LA($A \rightarrow \lambda$) = { *abd*, *bcd* } /* 2nd symbol: { *ab*, *bc* }; 3rd symbol: { ***abd***, *bcd* } */

- The 3rd symbol of the LA strings provides sufficient information to discriminate which one of the $A$ rules to use.

# Lookahead Strings and Lookahead Sets

- Example 19.1.2. Given the following grammar $G_2$:

$$S \to ABCabcd, \quad A \to a \mid \lambda, \quad B \to b \mid \lambda, \quad C \to c \mid \lambda$$

LA($S$) = { *abcabcd*, *ababcd*, *acabcd*, *bcabcd*, *aabcd*, *babcd*, *cabcd*, *abcd* }

  - No lookahead symbol is required in selecting the only ***S*** rule

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

$A \to \lambda$ LA($A \to \lambda$) = { *bcabcd*, *babcd*, *cabcd*, *abcd* }

  - The 4th lookahead symbol is required in selecting the ***A*** rule

$B \to b$ LA($B \to b$) = { *bcabcd*, *babcd* }

$B \to \lambda$ LA($B \to \lambda$) = { *cabcd*, *abcd* }

  - The 1st lookahead symbol is required in selecting the ***B*** rule

$C \to c$ LA($C \to c$) = { *cabcd* }

$C \to \lambda$ LA($C \to \lambda$) = { *abcd* }

  - The 1st lookahead symbol is required in selecting the ***C*** rule

# FIRST, FOLLOW, and Lookahead Sets

- The lookahead set $LA_k(A)$ contains prefixes of length up to $k$ of strings that can be derived from the variable $A$ (and after)
- If variable $A$ derives strings of length < $k$, the remainder of the lookahead strings comes from derivations that follow $A$ in the production rules of the grammar.
- $FIRST_k(A)$ contains *prefixes* of length up to $k$ of terminal symbols (directly) derivable from $A$.
- $FOLLOW_k(A)$ contains *prefixes* of length up to $k$ of terminal symbols that can follow the strings derivable from $A$.
- Defn. 19.2.1. Let $G$ be a CFG. For every string $u \in (V \cup \Sigma)^*$ and $k > 0$, the set $FIRST_k(u)$ is defined by

$$FIRST_k(u) = trunc_k( \{ x \mid u \overset{*}{\Rightarrow} x, x \in \Sigma^* \} )$$

where

$$trunc_k(X) = \{ u \mid u \in X \text{ w/ length}(u) \leq k \text{ or } uv \in X \text{ w/ length}(u) = k \}$$

# FIRST, FOLLOW, and Lookahead Sets

- Defn. 19.2.3. Let $G$ be a CFG. For every $A \in V$ and $k > 0$, the set $\text{FOLLOW}_k(A)$ is defined by

$$\text{FOLLOW}_k(A) = \{ x \mid S \overset{*}{\Rightarrow} uAv \text{ and } x \in \text{FIRST}_k(v) \}$$

- Example 19.2.1. Given $G_2$ (in Example 19.1.2),

$$S \to ABCabcd, \quad A \to a \mid \lambda, \quad B \to b \mid \lambda, \quad C \to c \mid \lambda$$

where $ABC \Rightarrow \{ abc, ab, ac, bc, a, b, c, \lambda \}$

$\text{FIRST}_1(ABC) = \{ a, b, c, \lambda \}$

$\text{FIRST}_2(ABC) = \{ ab, ac, bc, a, b, c, \lambda \}$

$\text{FIRST}_3(S) = \{ abc, aba, aca, bca, aab, bab, cab \}$

- Example 19.2.2.

$\text{FOLLOW}_1(S) = \{ \lambda \}$

$\text{FOLLOW}_1(A) = \{ b, c, a \}$

$\text{FOLLOW}_1(B) = \{ c, a \}$

$\text{FOLLOW}_1(C) = \{ a \}$

$\text{FOLLOW}_2(S) = \{ \lambda \}$

$\text{FOLLOW}_2(A) = \{ bc, ba, ca, ab \}$

$\text{FOLLOW}_2(B) = \{ ca, ab \}$

$\text{FOLLOW}_2(C) = \{ ab \}$

# FIRST, FOLLOW and Lookahead Sets

- Lemma 19.2.2. For every $k > 0$,

  1. $\text{FIRST}_k(\lambda) = \{ \lambda \}$
  2. $\text{FIRST}_k(a) = \{ a \}$
  3. $\text{FIRST}_k(au) = \{ av \mid v \in \text{FIRST}_{k-1}(u) \}$
  4. $\text{FIRST}_k(uv) = \text{trunc}_k(\text{FIRST}_k(u)\ \text{FIRST}_k(v))$
  5. If $A \rightarrow w \in G$, then $\text{FIRST}_k(w) \subseteq \text{FIRST}_k(A)$

- Lemma 19.2.4. For every $k > 0$,

  1. $\text{FOLLOW}_k(S)$ contains $\lambda$, where $S$ is the start symbol of $G$
  2. If $A \rightarrow uB \in G$, then $\text{FOLLOW}_k(A) \subseteq \text{FOLLOW}_k(B)$,

     i.e., any string that follows $A$ can also follow $B$
  3. If $A \rightarrow uBv \in G$, then $\text{trunc}_k(\text{FIRST}_k(v)\ \text{FOLLOW}_k(A)) \subseteq \text{FOLLOW}_k(B)$

     i.e., the strings that follow $B$ include those generated by $v$ concatenated with all terminal strings that follow $A$

- Example: Given $S \rightarrow aSc \mid bSc \mid \lambda$

  $\text{FIRST}_1(S) = \{ a, b, \lambda \}$ $\quad\quad$ $\text{FIRST}_2(S) = \{ aa, ab, ac, ba, bb, bc, \lambda \}$

  $\text{FOLLOW}_1(S) = \{ c, \lambda \}$ $\quad\quad$ $\text{FOLLOW}_2(S) = \{ c, cc, \lambda \}$

# LL(*K*) Grammars

- Theorem 19.2.5. Let $G$ be a CFG. For every $k > 0$, $A \in V$, and rule $A \to w = u_1 u_2 \ldots u_n$ in $P$,

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

  ii) $LA_k(A \to w) = trunc_k(FIRST_k(w)\ FOLLOW_k(A))$
  $= trunc_k(FIRST_k(u_1) \ldots FIRST_k(u_n)\ FOLLOW_k(A))$

# 19.4. Construction of FIRST$_k$ Sets

- Algorithm 19.4.1 Construction of FIRST$_k$ Sets
  - Input: a CFG $G = (V, \Sigma, P, S)$

1. For each $a \in \Sigma$, **do F'**$(a)$ := $\{ a \}$
2. For each $A \in V$, **do F**$(A)$ := $\begin{cases} \{ \lambda \} & \text{if } A \to \lambda \in P \\ \phi & \text{otherwise} \end{cases}$
3. Repeat
   - 3.1 for each $A \in V$, **do F'**$(A)$ := **F**$(A)$
   - 3.2 for each rule $A \to u_1 u_2 \ldots u_n$ with $n > 0$ **do**

     **F($A$) := F($A$) $\cup$ trunc$_k$(F'($u_1$)F'($u_2$) … F'($u_n$))**

   UNTIL **F**$(A)$ = **F'**$(A)$, $\forall_{A \in V}$
4. FIRST$_k(A)$ = **F**$(A)$

# 19.4. Construction of FIRST$_k$ Sets

- Example 19.4.1 Construct the **FIRST$_2$** sets for the variables of

$$S \rightarrow A\#\#$$
$$A \rightarrow aAd \mid BC$$
$$B \rightarrow bBc \mid \lambda$$
$$C \rightarrow acC \mid ad$$

$$F'(a) = a \quad F'(b) = b \quad F'(c) = c \quad F'(d) = d \quad F'(\#) = \#$$

$$F(S) = \phi \quad F(A) = \phi \quad F(B) = \{ \lambda \} \quad F(C) = \phi$$

$F(S) := F(S) \cup trunc_2(F'(A) \{ \# \} \{ \# \})$

$F(A) := F(A) \cup trunc_2(\{ a \} F'(A) \{ d \}) \cup trunc_2(F'(B) F'(C))$

$F(B) := F(B) \cup trunc_2(\{ b \} F'(B) \{ c \})$

$F(C) := F(C) \cup trunc_2(\{ a \} \{ c \} F'(C)) \cup trunc_2(\{ a \} \{ d \})$

| | $F(S)$ | $F(A)$ | $F(B)$ | $F(C)$ |
|---|---|---|---|---|
| 0 | $\phi$ | $\phi$ | $\{ \lambda \}$ | $\phi$ |
| 1 | $\phi$ | $\phi$ | $\{ \lambda, bc \}$ | $\{ ad \}$ |
| 2 | $\phi$ | $\{ ad, bc \}$ | $\{ \lambda, bc, bb \}$ | $\{ ad, ac \}$ |
| 3 | $\{ ad, bc \}$ | $\{ ad, bc, aa, ab, ac, bb \}$ | $\{ \lambda, bc, bb \}$ | $\{ad, ac \}$ |
| 4 | $\{ ad, bc, aa, ab, ac, bb \}$ | $\{ ad, bc, aa, ab, ac, bb \}$ | $\{ \lambda, bc, bb \}$ | $\{ ad, ac \}$ |
| 5 | $\{ ad, bc, aa, ab, ac, bb \}$ | $\{ ad, bc, aa, ab, ac, bb \}$ | $\{ \lambda, bc, bb \}$ | $\{ ad, ac \}$ |

# 19.4. Construction of $FOLLOW_k$ Sets

- Algorithm 19.5.1 Construction of $FOLLOW_k$ Sets
  - Input: a CFG $G = (V, \Sigma, P, S)$, $FIRST_k(A)$ for every $A \in V$

1. $FL(S) := \{ \lambda \}$
2. **for** each $A \in V - \{ S \}$ , **do** $FL(A) := \phi$
3. **repeat**
   - 3.1 **for** each $A \in V$, **do** $FL'(A) := FL(A)$
   - 3.2 **for** each rule $A \rightarrow w = u_1 u_2 \ldots u_n$ with $w \notin \Sigma^*$ **do**
     - 3.2.1. $L := FL'(A)$
     - 3.2.2. **if** $u_n \in V$, **then** $FL(u_n) := FL(u_n) \cup L$
     - 3.2.3. **for** $i := n - 1$ to 1 **do**
       - 3.2.3.1. $L := trunc_k(FIRST_k(u_{i+1})\ L)$
       - 3.2.3.2. **if** $u_i \in V$, **then** $FL(u_i) := FL(u_i) \cup L$

   **until** $FL(A) = FL'(A)$, $\forall_{A \in V}$
4. $FOLLOW_k(A) = FL(A)$

# 19.5. Construction of FOLLOW$_k$ Sets

- Example 19.5.1 Construct the **FOLLOW$_2$** sets for the variables of

| Rule | Assignments |
|---|---|
| $S \to A\#\#$ | $*$ $FL(A) := FL(A) \cup trunc_2(\{\ \#\ \}\ \{\ \#\ \}\ FL'(S))$ |
| $A \to aAd$ | $FL(A) := FL(A) \cup trunc_2(\{\ d\ \}\ FL'(A))$ |
| $A \to BC$ | $FL(C) := FL(C) \cup FL'(A)$ |
| | $FL(B) := \sout{FL(B) \cup trunc_2(\text{FIRST}_2(C)\ FL'(A))}$ |
| | $*$ $:= FL(B) \cup trunc_2(\{\ ad, ac\ \}\ FL'(A))$ |
| $B \to bBc$ | $FL(B) := FL(B) \cup trunc_2(\{\ c\ \}\ FL'(B))$ |
| $C \to acC \mid ad$ | $\sout{FL(C) := FL(C) \cup FL'(C)}$ |

| | **FL**($S$) | **FL**($A$) | **FL**($B$) | **FL**($C$) |
|---|---|---|---|---|
| 0 | { λ } | ϕ | ϕ | ϕ |
| 1 | { λ } | { ## } | ϕ | ϕ |
| 2 | { λ } | { ##, d# } | { ad, ac } | { ## } |
| 3 | { λ } | { ##, d#, dd } | { ad, ac, ca } | { ##, d# } |
| 4 | { λ } | { ##, d#, dd } | { ad, ac, ca, cc } | { ##, d#, dd } |
| 5 | { λ } | { ##, d#, dd } | { ad, ac, ca, cc } | { ##, d#, dd } |

# 19.5 Construction of $LA_k$ Sets

- Example 19.5.2 Construct the **$LA_2$** sets for the rules of

$$LA_2(S \to A\#\#) = \{ ad, bc, aa, ab, bb, ac \}$$
$$LA_2(A \to aAd) = \{ aa, ab \}$$
$$LA_2(A \to BC) = \{ ad, ac, bc, bb \}$$
$$LA_2(B \to bBc) = \{ bc, bb \}$$
$$LA_2(B \to \lambda) = \{ ad, ac, ca, cc \}$$
$$LA_2(C \to acC) = \{ ac \}$$
$$LA_2(C \to ad) = \{ ad \}$$

| $FIRST_2(S)$ | $FIRST_2(A)$ | $FIRST_2(B)$ | $FIRST_2(C)$ |
|---|---|---|---|
| { *ad*, *bc*, *aa*, *ab*, *bb*, *ac* } | { *ad*, *bc*, *aa*, *ab*, *bb*, *ac* } | { λ, *bc*, *bb* } | { *ad*, *ac* } |

| $FOLLOW_2(S)$ | $FOLLOW_2(A)$ | $FOLLOW_2(B)$ | $FOLLOW_2(C)$ |
|---|---|---|---|
| { λ } | { ##, *d*#, *dd* } | { *ad*, *ac*, ca, *cc* } | { ##, *d*#, *dd* } |

# 19.3 Strong LL(*K*) Grammars

- In strong LL(*k*) grammars
  - $\forall A \in V$, $LA_k(A)$ is partitioned by $LA_k(A \rightarrow w_i)$, $i \geq 1$
  - An endmarker $\#^k$ is attached to the *end* of each string in the grammar to guarantee that every LA string contains exactly *k* symbols

- Definition 19.3.1 Let $G = (V, \sum, P, S)$ be a CFG w/ endmarker $\#^k$. *G* is *strong* LL(*k*) if there are two leftmost derivations

$$S \overset{*}{\Rightarrow} u_1 A v_1 \overset{*}{\Rightarrow} u_1 x v_1 \overset{*}{\Rightarrow} u_1 z w_1$$
$$S \overset{*}{\Rightarrow} u_2 A v_2 \overset{*}{\Rightarrow} u_2 y v_2 \overset{*}{\Rightarrow} u_2 z w_2$$

  where $u_i, w_i, z \in \sum^*$ ($i$ = 1 or 2) and $length(z) = k$, then $x = y$.

- Theorem 19.3.2 A grammar *G* is *strong* LL(*k*) if and only if $\forall_i$, $LA_k(A \rightarrow w_i)$ partition LA(*A*) for each variable $A \in V$.

# 19.6 A Strong LL(1) Grammar

- Given the following grammar *G*:

$$S \rightarrow A\#$$
$$A \rightarrow TB$$
$$B \rightarrow Z \mid \lambda$$
$$Z \rightarrow +TY$$
$$Y \rightarrow Z \mid \lambda$$
$$T \rightarrow b \mid (A)$$

*G* is a *strong* LL(1) since the $LA_1$ sets for the rules are *disjoint*

$LA_1(S \rightarrow A\#) = \{ b, ( \}$

$LA_1(A \rightarrow TB) = \{ b, ( \}$

$LA_1(B \rightarrow Z) = \{ + \}$

$LA_1(B \rightarrow \lambda) = \{ \#, ) \}$

$LA_1(Z \rightarrow +TY) = \{ + \}$

$LA_1(Y \rightarrow Z) = \{ + \}$

$LA_1(Y \rightarrow \lambda) = \{ \#, ) \}$

$LA_1(T \rightarrow b) = \{ b \}$

$LA_1(T \rightarrow (A)) = \{ ( \}$

# 19.7 A Strong LL($k$) Parser

- Example 19.7.1

Input String:
$p = (b+b)\#$

$LA_1(S \to A\#) = \{ b, ( \}$
$LA_1(A \to TB) = \{ b, ( \}$
$LA_1(Y \to Z) = \{ + \}$
$LA_1(Y \to \lambda) = \{ \#, ) \}$
$LA_1(Z \to +TY) = \{ + \}$
$LA_1(B \to Z) = \{ + \}$
$LA_1(B \to \lambda) = \{ \#, ) \}$
$LA_1(T \to b) = \{ b \}$
$LA_1(T \to (A)) = \{ ( \}$

| $u$ | $A$ | $V$ | LA | Rule | Derivation |
|---|---|---|---|---|---|
| $\lambda$ | $S$ | $\lambda$ | ( | $S \to A\#$ | $S \Rightarrow A\#$ |
| $\lambda$ | $A$ | $\#$ | ( | $A \to TB$ | $\Rightarrow TB\#$ |
| $\lambda$ | $T$ | $B\#$ | ( | $T \to (A)$ | $\Rightarrow (A)B\#$ |
| $($ | $A$ | $)B\#$ | $b$ | $A \to TB$ | $\Rightarrow (TB)B\#$ |
| $($ | $T$ | $B)B\#$ | $b$ | $T \to b$ | $\Rightarrow (bB)B\#$ |
| $(b$ | $B$ | $)B\#$ | + | $B \to Z$ | $\Rightarrow (bZ)B\#$ |
| $(b$ | $Z$ | $)B\#$ | + | $Z \to +TY$ | $\Rightarrow (b+TY)B\#$ |
| $(b+$ | $T$ | $Y)B\#$ | $b$ | $T \to b$ | $\Rightarrow (b+bY)B\#$ |
| $(b+b$ | $Y$ | $)B\#$ | ) | $Y \to \lambda$ | $\Rightarrow (b+b)B\#$ |
| $(b+b)$ | $B$ | $\#$ | $\#$ | $B \to \lambda$ | $\Rightarrow (b+b)\#$ |

# 19.8 LL(*K*) Grammars

- Definition 19.8.1 Let $G = (V, \sum, P, S)$ be a CFG w/ endmarker $\#^k$. $G$ is LL($k$) if whenever there are two leftmost derivations

$$S \overset{*}{\Rightarrow} uAv \overset{*}{\Rightarrow} uxv \overset{*}{\Rightarrow} uzw_1$$
$$S \overset{*}{\Rightarrow} uAv \overset{*}{\Rightarrow} uyv \overset{*}{\Rightarrow} uzw_2$$

  where $u, w_i, z \in \sum^*$ ($i$ = 1 or 2) and $length(z) = k$, then $x = y$.

- Theorem 19.8.2 Let $G = (V, \sum, P, S)$ be a CFG w/ endmarker $\#^k$ & *$uAv$ a sentential form of $G$*.

  1) The lookahead set of the sentential form $uAv$ is defined by $LA_k(uAv) = FIRST_k(Av)$.

  2) The lookahead set for the sentential form $uAv$ & rule $A \rightarrow w$ is defined by $LA_k(uAv, A \rightarrow w)$.

# Lookahead Sets in CFGs

- Example 19.8.1. Given the LA sets of grammar $G_1$:

$LA(S) = \{ aabd, babd, abd, cabcd, cbbcd, cbcd \}$

$LA(S \rightarrow Aabd) = \{ aabd, babd, abd \}$ /* 1st symbol: {a, b} */

$LA(S \rightarrow cAbcd) = \{ cabcd, cbbcd, cbcd \}$ /* 1st symbol: { c } */

$LA(A \rightarrow a) = \{ aabd, abcd \}$ /* 2nd symbol: {*aa*, *ab*}; 3rd symbol: {*aab*, *abc*} */

$LA(A \rightarrow b) = \{ babd, bbcd \}$ /* 2nd symbol: {*ba*, *bb*}; 3rd symbol: {*bab*, *bbc*} */

$LA(A \rightarrow \lambda) = \{ abd, bcd \}$ /* 2nd symbol: {*ab*, *bc*}; 3rd symbol: {*abd*, *bcd*} */

$G_1$ is not *strong LL(2),* but it is *strong LL(3)* since

$LA_2(S, S \rightarrow Aabd) = \{ aa, ba, ab \}$

$LA_2(S, S \rightarrow cAbcd) = \{ ca, cb \}$

$LA_2(cAbcd, A \rightarrow a) = \{ ab \}$

$LA_2(cAbcd, A \rightarrow b) = \{ bb \}$

$LA_2(cAbcd, A \rightarrow \lambda) = \{ bc \}$

$LA_2(Aabd, A \rightarrow a) = \{ aa \}$

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

$LA_2(Aabd, A \rightarrow \lambda) = \{ ab \}$

# 19.7 A Strong LL($k$) Parser

- Algorithm 19.7.1 Deterministic Parser for a Strong LL($k$) Grammar

**Input**: A strong LL($k$) grammar $G = (V, \Sigma, P, S)$, $p \in \Sigma^*$, $LA_k(A \rightarrow w)$, $\forall A \rightarrow w \in P$.

**Output**: $p \in L(G)$ or $p \notin L(G)$.

1. $q := S$
2. **repeat**
   2.0. Let $q = uAv$, where $A$ is the leftmost variable in $q$.
   Let $p = uyz$, where length($y$) = $k$.
   2.1. **If** $y \in LA_k(A \rightarrow w)$ in $P$, **then** $q := uwv$.
   **until** $q = p$ or $y \notin LA_k(A \rightarrow w)$, $\forall A$ rules in $P$.
3. **If** $q = p$, then
   *accept*
   **else**
   *reject*

# Lookahead Sets in CFGs

- Example 19.8.2. Given the LA sets of grammar *G*:

$$S \rightarrow aBAd \mid bBbAd$$
$$A \rightarrow abA \mid c$$
$$B \rightarrow ab \mid a$$

Consider $LA_3(B)$:

$$LA_3(aBAd, B \rightarrow ab) = \{ aba, abc \}$$
$$LA_3(aBAd, B \rightarrow a) = \{ aab, acd \}$$
$$LA_3(bBbAd, B \rightarrow ab) = \{ abb \}$$
$$LA_3(bBbAd, B \rightarrow a) = \{ aba, abc \}$$

*G* is not *strong LL*(*k*), for any $k \geq 1$, since

$$LA_3(B \rightarrow ab) = ab(ab)^*cd \cup abb(ab)^*cd$$
$$LA_3(B \rightarrow a) = a(ab)^*cd \cup ab(ab)^*cd$$

# 19.8 LL($k$) Parser

- Algorithm 19.8.3 Deterministic Parser for an LL($k$) Grammar.

**Input**: An LL($k$) grammar $G = (V, \Sigma, P, S)$, $p \in \Sigma^*$, $FIRST_k(A)$, $\forall A \in V$

**Output**: $p \in L(G)$ or $p \notin L(G)$.

1. $q := S$
2. **Repeat**
   - 2.0. Let $q = uAv$, where $A$ is the leftmost variable in $q$.
     Let $p = uyz$, where length($y$) = $k$.
   - 2.1. **For** each rule $A \rightarrow w$, construct $LA_k(uAv, A \rightarrow w)$
   - 2.2. **If** $y \in LA_k(uAv, A \rightarrow w)$ in $P$, **then** $q := uwv$.

   **Until** $q = p$ or $y \notin LA_k(uAv, A \rightarrow w)$, $\forall A$ rules in $P$.
3. **If** $q = p$, then
   *accept*
   **else**
   *reject*

# LR(*K*) Grammars

- A deterministic bottom-up parser can be adopted in an attempt to reduce the *input string* to the *start symbol* of a grammar
- Read the input string from *left to right* while constructing a *rightmost derivation* of the input string using a lookahead system involving *k* symbols
- *Process* (of recognizing input strings of a CFG *G*):
  - Step 1. Transfers symbols from its input to a stack till the uppermost stack symbols match the R.H.S. of some production rule *R*
  - Step 2. Replace these symbols with the L.H.S. of *R*
  - Step 3. Repeat steps 1 and 2 till the top stack symbol is the grammar's *start symbol* or halt (i.e., the input string cannot be derived from *G*)

# LR(*K*) Grammars

- Constructing a PDA from a CFG *G* that behaves as a *LR*(*k*) parser:
  - Step 1. Create *states* $q_0$ (initial), $q_f$ (final), $q_1$ and $q_2$
  - Step 2. Create transitions $\delta(q_0, \lambda, \lambda) = \{ [q_1, \#] \}$ and $\delta(q_2, \lambda, \#) = \{ [q_f, \lambda] \}$
  - Step 3. For each terminal symbol $x \in \Sigma$,
    - Create the transition $\delta(q_1, x, \lambda) = \{ [q_1, X] \}$, where $X \in \Gamma$, a shift
  - Step 4. For each production rule $N \rightarrow w$ in *P*, where $w \in (V \cup \Sigma)^*$
    - Create the transition $\delta(q_1, \lambda, w) = \{ [q_1, N] \}$, a reduce
  - Step 5. Create the transition $\delta(q1, \lambda, S) = \{ [q2, \lambda] \}$, where *S* is the start symbol in *G*

# LR(*K*) Grammars

- Example: Let *G* be the CFG

$$S \rightarrow zMNz$$

$$M \rightarrow aMa \mid z$$

$$N \rightarrow bNb \mid z$$

A left-to-right, rightmost derivation of the string *zazabzbz* is:

$$\begin{aligned} S &\Rightarrow z\text{MN}z \\ &\Rightarrow z\text{M}b\text{N}bz \\ &\Rightarrow zMbzbz \\ &\Rightarrow za\text{M}abzbz \\ &\Rightarrow zazabzbz \end{aligned}$$