Contents

Chapter 8

Deterministic top-down parsing

8.1 LL(k) grammars and parsers

Top-down parsing approach, according to which a parser begins with the assumption that the unread input string is syntactically correct, and attempts to verify this fact by traversing the supposed parse tree of this string, along with reading the input from left to right.

First studied by Lewis and Stearns [4], followed by Kurki-Suonio [3], by Rosenkrantz and Stearns [5] and by Knuth [2].

A top-down parser attempts to construct a parse tree of an input string, while reading it from left to right. At every point of its computation, the parser's memory configuration is a pair (α, v), where v is the unread portion of the input string uv, which the parser tries to parse according to the sequence of symbols $\alpha = X_1 \ldots X_\ell \in (\Sigma \cup N)^*$. The latter sequence is stored in a stack, so that it can be accessed only from its left side.

In the figure: the configuration of the parser is described by a path in the tree. The read portion u of the input string has already been parsed according to the subtrees to the left of this path (that grey area). The nodes on the right border of the path form the stack contents α. The white area in the tree has not yet been processed by the parser.

At each point of the computation, the parser sees the top symbol of the stack and the first k symbols of the unread input (known as the *look-ahead string*), where $k \geqslant 1$ is a constant. If there is a nonterminal symbol $A \in N$ at the top of the stack, determines a rule $A \to \alpha$ for this symbol, pops this symbol, and pushes the right-hand side of the rule onto the stack:

$$(A\beta, v) \xrightarrow{A \to \alpha} (\alpha\beta, v)$$

The rule is chosen by accessing a look-up table $T_k \colon N \times \Sigma^{\leqslant k} \to R \cup \{-\}$, which contains either a rule to apply, or a marker indicating a syntax error.

If the top symbol of the stack is a symbol from the alphabet, the parser checks that the unread portion of the input begins with the same symbol, and then pops this symbol from the stack and reads it from the input.

$$(a\beta, av) \xrightarrow{\text{READ } a} (\beta, v)$$

Example 8.1. *Consider the following grammar for the language $\{ a^m b^{m+n} c^n \mid m, n \geqslant 0\}$.*

$$\begin{array}{rcl} S & \to & AB \\ A & \to & aAb \mid \varepsilon \\ B & \to & bBc \mid \varepsilon \end{array}$$

A top-down parser recognizes the string aabbbc as follows.

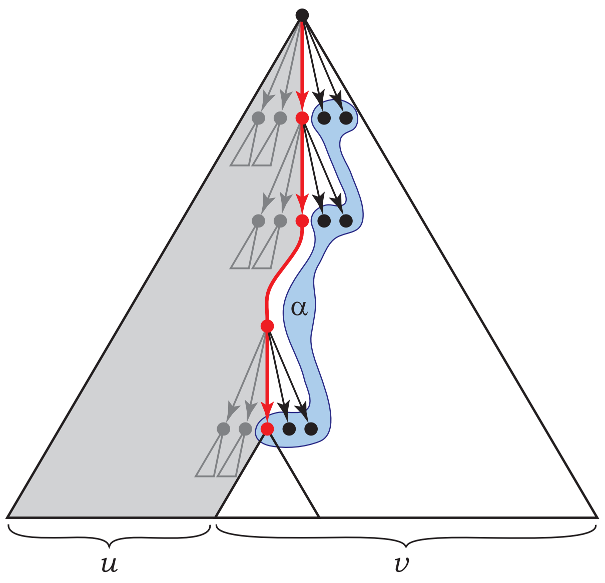

Figure 8.1: The configuration (α, v) of an LL parser demonstrated on a parse tree.

$$(S, aabbbc) \xrightarrow{S\to AB} (AB, aabbbc) \xrightarrow{A\to aAb} (aAbB, aabbbc) \xrightarrow{\text{READ } a}$$
$$(AbB, abbbc) \xrightarrow{A\to aAb} (aAbbB, abbbc) \xrightarrow{\text{READ } a} (AbbB, bbbc) \xrightarrow{A\to\varepsilon} (bbB, bbbc) \xrightarrow{\text{READ } b}$$
$$(bB, bbc) \xrightarrow{\text{READ } b} (B, bc) \xrightarrow{B\to bBc} (bBc, bc) \xrightarrow{\text{READ } b} (Bc, c) \xrightarrow{B\to\varepsilon} (c, c) \xrightarrow{\text{READ } c} (\varepsilon, \varepsilon)$$

It is sufficient to let $k = 1$ and use the following parsing table:

	a	b	c	ε
S	$S \to AB$	$S \to AB$	–	$S \to AB$
A	$A \to aAb$	$A \to \varepsilon$	–	$A \to \varepsilon$
B	–	$B \to bBc$	$B \to \varepsilon$	$B \to \varepsilon$

For a string $w \in \Sigma^*$, denote its first k symbols, with $k \geqslant 0$, by

$$\text{First}_k(w) = \begin{cases} w, & \text{if } |w| \leqslant k \\ \text{first } k \text{ symbols of } w, & \text{if } |w| > k \end{cases}$$

This definition is extended to languages as $\text{First}_k(L) = \{\, \text{First}_k(w) \mid w \in L\}$.

Definition 8.1. *Let $k \geqslant 1$. An ordinary grammar $G = (\Sigma, N, R, S)$ is said to be LL(k), if, for all $A \in N$, $u, u' \in \Sigma^*$ and $v, v' \in \Sigma^*$ with $\text{First}_k(v) = \text{First}_k(v')$, if*

$$S \Longrightarrow^* uA\beta \Longrightarrow u\alpha\beta \Longrightarrow^* uv,$$
$$S \Longrightarrow^* u'A\beta' \Longrightarrow u'\alpha'\beta' \Longrightarrow^* u'v',$$

then $\alpha = \alpha'$.

8.1.1 Look-up table construction

For a grammar $G = (\Sigma, N, R, S)$ construct $\textsc{First}_k(A) = \text{First}_k(L_G(A))$ for all $A \in N$. Let $\textsc{First}_k(a) = \{a\}$.

Definition 8.2. *A string $v \in \Sigma^*$ is said to follow $X \in \Sigma \cup N$, if $S \Longrightarrow^* \gamma X v$ for some $\gamma \in (\Sigma \cup N)^*$.*

For a grammar $G = (\Sigma, N, R, S)$ construct $\textsc{Follow}_k(A) = \text{First}_k(\{ v \mid v \text{ follows } A\})$ for all $A \in N$.

For each rule $A \in X_1 \dots X_\ell \in R$, if $x \in \text{First}_k(\textsc{First}_k(X_1) \dots \textsc{First}_k(X_\ell) \cdot \textsc{Follow}_k(A))$, then let $T_k(A, x) = A \in X_1 \dots X_\ell$. If this definition leads to any contradictory assignments, then the grammar is not LL(k).

Algorithms to construct all this.

Algorithm 8.1 Constructing the sets $\textsc{First}_k$ for an ordinary grammar

Let $G = (\Sigma, N, R, S)$ be an ordinary grammar, let $k > 0$. For all $X \in \Sigma \cup N$, compute the set $\textsc{First}_k(X)$.

```
1: $\textsc{First}_k(A) = \varnothing$ for all $A \in N$
2: $\textsc{First}_k(a) = \{a\}$ for all $a \in \Sigma$
3: while new strings can be added to $\langle \textsc{First}_k(A) \rangle_{A \in N}$ do
4:     for all $A \to X_1 \dots X_\ell \in R$ do
5:         $\textsc{First}_k(A) = \textsc{First}_k(A) \cup \text{First}_k(\textsc{First}_k(X_1) \cdot \ldots \cdot \textsc{First}_k(X_\ell))$
```

Denote $\textsc{First}_k(X_1 \dots X_n) = \text{First}_k(\textsc{First}_k(X_1) \cdot \ldots \cdot \textsc{First}_k(X_n))$.

Algorithm 8.2 Constructing the sets $\textsc{Follow}_k$ for an ordinary grammar

Let $G = (\Sigma, N, R, S)$ be an ordinary grammar, let $k > 0$. For all $A \in N$, compute the set $\textsc{Follow}_k(A)$.

```
1: $\textsc{Follow}_k(S) = \{\varepsilon\}$
2: $\textsc{Follow}_k(A) = \varnothing$ for all $A \in N \setminus \{S\}$
3: while new strings can be added to $\langle \textsc{Follow}_k(A) \rangle_{A \in N}$ do
4:     for all $B \to \beta \in R$ do
5:         for all partitions $\beta = \mu A \nu$, with $A \in N$ and $\mu, \nu \in (\Sigma \cup N)^*$ do
6:             $\textsc{Follow}_k(A) = \textsc{Follow}_k(A) \cup \text{First}_k(\textsc{First}_k(\nu) \cdot \textsc{Follow}_k(B))$
```

Algorithm 8.3 Constructing an LL(k) table for an ordinary grammar

Let $G = (\Sigma, N, R, S)$ be an ordinary grammar, let $k > 0$. Compute $T_k(A, x)$ for all $A \in N$ and $x \in \Sigma^{\leqslant k}$.

```
1: for all $A \to \alpha \in R$ do
2:     for all $x \in \text{First}_k(\textsc{First}_k(\alpha) \cdot \textsc{Follow}_k(A))$ do
3:         if $T_k(A, x)$ is not yet defined then
4:             $T_k(A, x) = (A \to \alpha)$
5:         else
6:             report conflict
```

8.1.2 Recursive descent

A recursive descent parser is a program containing a procedure for every terminal and nonterminal symbol used in the grammar. These procedures have access to the input string $w = a_1 a_2 \dots a_{|w|}$ and to a positive integer p pointing at the current position in this string.

Each procedure $a()$, with $a \in \Sigma$, simply checks that the next input symbol is a, and advances p to the next position.

For each $A \in N$, the procedure $A()$ uses the table T_k to choose one of the rules for A, using the k next symbols of the input. Once a rule $A \to X_1 \dots X_\ell$ is chosen, the subsequent code $X_1(); \dots X_\ell();$ parses the following substring according to this rule, using the corresponding procedures to parse its substrings. Each procedure $X_i()$ advances the position in the input string, so that the next procedure deals with a subsequent substring. In total, $A()$ advances the position in ℓ steps, consuming a substring generated by A.

Easy to program by hand.

8.2 Normal forms for LL(k) grammars

8.2.1 Elimination of null rules

Eliminating null rules is not so easy, because the earlier used method does not preserve the LL property.

Example 8.2. *Consider the following LL(1) grammar, which describes the language* $a^*c\{b, \varepsilon\}$.

$$\begin{aligned} S &\to AB \\ A &\to aA \mid c \\ B &\to b \mid \varepsilon \end{aligned}$$

The standard transformation for eliminating null rules leads to the following grammar, which is not LL(k) for any k.

$$\begin{aligned} S &\to AB \mid A \\ A &\to aA \mid c \\ B &\to b \end{aligned}$$

Indeed, an LL(k) parser would not be able to decide between the rules $S \to AB$ *and* $S \to A$ *on the input* $w = a^{k-1}c$.

Theorem 8.1 (Kurki-Suonio [3]; Rosenkrantz and Stearns [5]). *For every LL(k) grammar there exists and can be effectively constructed an LL(*$k+1$*) grammar without null rules that defines the same language.*

First, transform the grammar so that no rule begins with a nullable symbol.

Lemma 8.1 (Rosenkrantz and Stearns [5]). *For every ordinary grammar* $G = (\Sigma, N, R, S)$, *there exists another grammar* $G' = (\Sigma, N \cup N', R', S')$, *with* $N' = \{ A' \mid A \in N \}$, *which satisfies the following conditions:*

1. *every rule in* R' *is either of the form* $X \to Y\alpha$, *with* $X \in N \cup N'$, $Y \in \Sigma \cup N'$ *and* $\alpha \in (\Sigma \cup N \cup N')^*$, *or of the form* $A \to \varepsilon$, *with* $A \in N$;
2. *for each* $A \in N$, $L_{G'}(A) = L_G(A)$ *and* $L_{G'}(A') = L_G(A') \setminus \{\varepsilon\}$, *and, in particular,* $L(G') = L(G) \setminus \{\varepsilon\}$;
3. *if* G *is LL(k), then so is* G'.

Proof. Let $N_0 = \{ A \mid A \in N,\ \varepsilon \in L_G(A) \}$ be the set of nullable symbols in G,

Consider any rule $A \to \alpha$ in R, and let $B_1 \dots B_m$ denote the largest prefix of α comprised of nullable symbols. Accordingly, this rule can be presented in the following form.

$$A \to B_1 \dots B_m X_1 \dots X_n \qquad (m, n \geqslant 0, B_1, \dots, B_m \in N_0, X_1, \dots, X_n \in (\Sigma \cup N)^*, X_1 \notin N_0)$$

If an LL parser uses this rule in a computation on some input string, it will then process the symbols $B_1, \dots, B_m, X_1, \dots, X_n$ from left to right, deriving ε from zero or more first symbols

$B_1, \ldots, B_{i-1}$, and then either derive a non-empty string from B_i or from X_1 (if $i-1=m$), or generate the empty string (if $i-1=m$ and $n=0$). In the new grammar, the choice of the leftmost symbol in the rule to derive a non-empty substring of the input is made in the rule for A; accordingly, there are the following new rules in R' corresponding to different choices of that symbol.

$$A \to B'_i B_{i+1} \ldots B_m X_1 \ldots X_n \qquad \text{(for } i \in \{1, \ldots, m\}) \tag{8.1a}$$
$$A \to X_1 \ldots X_n \tag{8.1b}$$

If $n=0$, then the latter rule produces the empty string. The rules for the other symbol A' are similar, except that the empty string is never produced.

$$A' \to B'_i B_{i+1} \ldots B_m X_1 \ldots X_n \qquad \text{(for } i \in \{1, \ldots, m\}) \tag{8.1c}$$
$$A' \to X_1 \ldots X_n \qquad \text{(if } n > 0) \tag{8.1d}$$

Proof that the new grammar describes the right language: easy.

Preservation of the LL property (rough sketch). Assume that G is LL(k) and consider the choice between any two rules for A in G'. If these two rules were created from different rules for A in G, then one can choose between these rules in G' in the same way as between their prototypes in G. If these rules were created from the same rule in G, then these rules are $A \to B'_i B_{i+1} \ldots B_m X_1 \ldots X_n$ and $A \to B'_j B_{j+1} \ldots B_m X_1 \ldots X_n$, with $i < j$, and the parser for G' chooses between them by the same condition as the parser for G uses to decide whether to derive the empty string from B'_i. □

Lemma 8.2 (Rosenkrantz and Stearns [5]). *Let $G = (\Sigma, N, R, S)$ be an LL(k) grammar, which has no rules of the form $A \to B\gamma$, with $\varepsilon \in L_G(B)$. Denote its nullable symbols by $N_0 = \{\, A \mid A \in N,\ \varepsilon \in L_G(A)\}$, and let $N_1 = \{\, A \mid A \in N,\ \varepsilon \notin L_G(A)\}$ be the rest of its category symbols. Then there exists an LL(k + 1) grammar $G' = (\Sigma, N', R', S')$, where N' is a finite subset of $\{\, [X\theta] \mid X \in \Sigma \cup N_1,\ \theta \in N_0^*\}$, and for each symbol $[X\theta] \in N'$, $L_{G'}([X\theta]) = L_G(X\theta)$.*

Proof. For every string $X_1\theta_1 \ldots X_n\theta_n \in (\Sigma \cup N)^* \setminus N_0(\Sigma \cup N)^*$, where $X_i \in \Sigma \cup N_1$ and $\theta_i \in N_0^*$, denote $\pi(X_1\theta_1 \ldots X_n\theta_n) = [X_1\theta_1] \ldots [X_n\theta_n] \in (N')^*$.

The initial symbol of the new grammar is $S' = [S]$.

If $[A\theta] \in N'$ and $A \to \alpha \in R$, then there is a rule

$$[A\theta] \to \pi(\alpha\theta)$$

If, for some $a \in \Sigma$ $\theta, \theta' \in N_0^*$, $A \in N_0$, the symbol $[a\theta A\theta']$ is in N', and $A \to \alpha \in R$, with $\alpha \neq \varepsilon$, then there is a rule

$$[a\theta A\theta'] \to a\pi(\alpha\theta')$$

If, for some $a \in \Sigma$ and $\theta \in N_0^*$, the symbol $[a\theta]$ is in N', then there is a rule

$$[a\theta] \to a$$

□

Example 8.3. *Consider the LL(1) grammar from Example 8.2. According to Lemma 8.2, it is transformed to the following LL(2) grammar:*

$$\begin{aligned}
[S] &\to [AB] \\
[AB] &\to [a][AB] \mid [cB] \\
[cB] &\to c[b] \mid c \\
[a] &\to a \\
[b] &\to b
\end{aligned}$$

8.2.2 Greibach normal form

GNF: with all rules of the form $A \to a\alpha$.

Theorem 8.2 (Rosenkrantz and Stearns [5]). *For every LL(k) grammar there exists and can be effectively constructed an LL($k+1$) grammar in the Greibach normal form that describes the same language.*

First, apply Theorem 8.1. If a grammar is LL, there is no left recursion in it. A finite sequence of substitutions.

8.3 Limitations of LL(k) grammars

8.3.1 First method: interchangeable substring ahead

Example 8.4 (Rosenkrantz and Stearns [5]). *The language* $\{\, a^n b^n \mid n \geqslant 0\} \cup \{\, a^n c^n \mid n \geqslant 0\}$ *is not LL(k) for any k.*

Proof. Suppose it is, and let $G = (\Sigma, N, R, S)$ be an LL(k) grammar without null rules that defines this language. For every $n \geqslant 0$, let $\alpha_n \in (\Sigma \cup N)^*$ be the stack contents of the parser on the input $a^{n+k}b^{n+k}$ after consuming the symbols a^n. On the input $a^{n+k}c^{n+k}$, the parser has the same stack contents after consuming a^n, because it cannot yet see whether there are symbols b or c ahead.

Claim 1: $\alpha_m \neq \alpha_n$ for all $m \neq n$.

If $\alpha_m = \alpha_n$, then the parser loses count and will accept the strings $a^{m+k}b^{n+k}$ and $a^{n+k}b^{m+k}$.

Claim 2: there exists n, for which $|\alpha_n| \geqslant k+2$.

Let $\alpha_n = X_1 \ldots X_m$, where $X_i \in \Sigma \cup N$ and $m \geqslant k+2$. It is known that $a^k b^{n+k}, a^k c^{n+k} \in L_G(\alpha_n) = L_G(X_1 \ldots X_m)$, that is, there are partitions $a^k b^{n+k} = x_1 \ldots x_m$ and $a^k c^{n+k} = y_1 \ldots y_m$ with $x_i, y_i \in L_G(X_i)$. Since none of X_i may generate the empty string, $x_i, y_i \neq \varepsilon$, and therefore $x_m = b^\ell$ and $y_m = c^{\ell'}$, where $0 < \ell < n+k$ and $0 < \ell' < n+k$. Then the string $a^k b^{n+k-\ell} c^{\ell'}$ is also in $L_G(\alpha_n)$, and therefore the string $a^{n+k} b^{n+k-\ell} c^{\ell'}$ is generated by the grammar, which is a contradiction. □

8.3.2 Second method: unexpected end

Example 8.5 (Wood [6]). *The language* $a^* \cup \{\, a^n b^n \mid n \geqslant 0\}$ *is not LL(k) for any k.*

Sketch of a proof. Suppose it is. Assume, without loss of generality, that it is described by an LL(k) grammar without null rules. On the one hand, after reading a^n, the parser needs to remember n in the stack, which requires an unbounded number of symbols on the stack. On the other hand, it should always be ready to accept a^k, and therefore cannot have more than k symbols on the stack. □

8.3.3 Disjoint unions of LL(k) languages

Theorem 8.3 (Rosenkrantz and Stearns [5]). *Let* $L_1, \ldots, L_n \subseteq \Sigma^*$ *be pairwise disjoint LL(k) languages, and assume that* $L_1 \cup \ldots \cup L_n$ *is regular. Then all languages* $L_1, \ldots, L_n$ *must be regular.*

Proof. If $L_1 \cup \ldots \cup L_n \neq \Sigma^*$, then $\overline{L_1 \cup \ldots \cup L_n}$ is a non-empty regular language, hence LL(k). Then it can be added to the list. Accordingly, assume that the union of all given languages is Σ^*.

Consider n LL(k) parsers that recognize the corresponding languages $L_1, \ldots, L_n$. The general goal is to prove that there exists some constant $\widehat{m}$ that bounds the number of symbols in the stack of each parser upon reading each input string.

For each string $u \in \Sigma^*$ and for each look-ahead string $x \in \Sigma^{\leqslant k}$, consider the computation of the parser for each L_i on the input ux. Denote its stack contents after reading u by $\alpha^{(i)}_{u,x}$, which may be undefined for some values of i. Consider the ordering of languages $L_1, \ldots, L_n$ according to the number of symbols in $\alpha^{(1)}_{u,x}, \ldots, \alpha^{(n)}_{u,x}$, represented by a permutation $(i_1(u,x), \ldots, i_n(u,x))$ of $(1, \ldots, n)$, where $|\alpha^{(i_1(u,x))}_{u,x}| \leqslant \ldots \leqslant |\alpha^{(i_n(u,x))}_{u,x}|$. If any values are undefined, assume that they are at the end of the list.

The updated goal is to prove that there exist such numbers $m_1, \ldots m_n \geqslant 0$, that for all j, the stack size $|\alpha^{(i_j(u,x))}_{u,x}|$ is bounded by m_j, for all strings $u \in \Sigma^*$ and look-ahead strings $x \in \Sigma^{\leqslant k}$.

Define $m_1 = k$.

Claim: $|\alpha^{(i_1(u,x))}_{u,x}| \leqslant m_1$.

For each u, x, the string ux must belong to some L_i. In order to accept from the configuration $(\alpha^{(i)}_{u,x}, x)$, the stack of the i-th parser may not contain more than $|x|$ symbols. Therefore,

$$\alpha^{(i_1(u,x))}_{u,x} \leqslant \alpha^{(i)}_{u,x} \leqslant |x| \leqslant k$$

For every u, x, consider any continuation that begins with x and is not accepted by the parser number $i_1(u,x)$. Let $v = v(u,x)$ denote the shortest such continuation, with $v \notin L(\alpha^{i_1(u,x)}_{u,x})$. As it is already known that the length of $\alpha^{i_1(u,x)}_{u,x}$ is bounded by m_1, there are only finitely many such languages $L(\alpha^{i_1(u,x)}_{u,x})$, and accordingly, the above definition gives only finitely many different continuations v over all (countably many) values of u and x. Therefore, the maximum length of v is well-defined, and it is used as the second upper bound m_2.

$$m_2 = \max_{\substack{u \in \Sigma^* \\ x \in \Sigma^{\leqslant k}}} \min \big\{\, |v| \;\big|\; \text{First}_k(v) = x,\ v \notin L(\alpha^{i_1(u,x)}_{u,x}) \,\big\}$$

To see that m_2 is the upper bound on the length of $\alpha^{(i_2(u,x))}_{u,x}$ for all u and x, consider any $u \in \Sigma^*$ and $x \in \Sigma^{\leqslant k}$. Let $v \in \Sigma^*$ be any of the shortest strings satisfying $\text{First}_k(v) = x$ and $v \notin L(\alpha^{(i_1(u,x))}_{u,x})$. Then the parser number $i_1(u,x)$ does not accept the string uv, and there should exist some other j-th parser that accepts it, with j in $\{i_2(u,x), \ldots, i_n(u,x)\}$. The acceptance of uv by the j-th parser means that $v \in L(\alpha^{(j)}_{u,x})$, which implies that $|\alpha^{(j)}_{u,x}| \leqslant |v|$, and therefore

$$|\alpha^{(i_2(u,x))}_{u,x}| \leqslant |\alpha^{(j)}_{u,x}| \leqslant |v| \leqslant m_2.$$

Every next upper bound m_j is defined as

$$m_j = \max_{\substack{u \in \Sigma^* \\ x \in \Sigma^{\leqslant k}}} \min \big\{\, |v| \;\big|\; \text{First}_k(v) = x,\ v \notin L(\alpha^{i_1(u,x)}_{u,x}) \cup \ldots \cup L(\alpha^{i_{j-1}(u,x)}_{u,x}) \,\big\}$$

This proves that the size of the stack of each parser is bounded by a constant, and therefore the parser has a finite number of possible stack configurations. Then the parser can be simulated by a finite automaton, which uses its internal states to hold the parser's stack contents. □

Corollary 8.1. *Let $L \subseteq \Sigma^*$ be a non-regular LL(k) language. Then its complement $\overline{L}$ is not LL(k') for any $k' \geqslant 1$.*

Example 8.6 (Rosenkrantz and Stearns [5]). *The language* $L = \{\, a^m b^n \mid 0 \leqslant m \leqslant n\}$ *is defined by the following LL(1) grammar:*

$$\begin{aligned} S &\to AB \\ A &\to aBb \mid \varepsilon \\ B &\to bB \mid \varepsilon \end{aligned}$$

Therefore, the language $L' = \{\, a^m b^n \mid m > n \geqslant 0\}$ *is not LL(k) for any k, because* $L \cup L' = a^* b^*$*.*

Bibliography

[1] J. C. Beatty, “Two iteration theorems for the LL(k) languages”, *Theoretical Computer Science*, 12:2 (1980), 193–228.

[2] D. E. Knuth, “Top-down syntax analysis”, *Acta Informatica*, 1 (1971), 79–110.

[3] R. Kurki-Suonio, “Notes on top-down languages”, *BIT Numerical Mathematics*, 9:3 (1969), 225–238.

[4] P. M. Lewis II, R. E. Stearns, “Syntax-directed transduction”, *Journal of the ACM*, 15:3 (1968), 465–488.

[5] D. J. Rosenkrantz, R. E. Stearns, “Properties of deterministic top-down grammars”, *Information and Control*, 17 (1970), 226–256.

[6] D. Wood, “A further note on top-down deterministic languages”, *Computer Journal*, 14:4 (1971), 396–403.

Index