



Compiler Construction

Lecture 7: Syntax Analysis III ($LL(1)$ Parsing)

Summer Semester 2017

Thomas Noll

Software Modeling and Verification Group

RWTH Aachen University

<https://moves.rwth-aachen.de/teaching/ss-17/cc/>

Recap: $LL(k)$ Grammars

$LL(k)$ Grammars I

$LL(k)$: reading of input from **L**eft to right with k -lookahead, computing a **L**eftmost analysis

Definition ($LL(k)$ grammar)

Let $G = \langle N, \Sigma, P, S \rangle \in CFG_{\Sigma}$ and $k \in \mathbb{N}$. Then G has the $LL(k)$ property (notation: $G \in LL(k)$) if for all leftmost derivations of the form

$$S \Rightarrow_i^* wA\alpha \begin{cases} \Rightarrow_i w\beta\alpha \Rightarrow_i^* wx \\ \Rightarrow_i w\gamma\alpha \Rightarrow_i^* wy \end{cases}$$

such that $\beta \neq \gamma$, it follows that $\text{first}_k(x) \neq \text{first}_k(y)$
(i.e., different productions must not yield the same lookahead).

Recap: $LL(k)$ Grammars

$LL(k)$ Grammars II

Remarks:

- If $G \in LL(k)$, then the leftmost derivation step for $wA\alpha$ in

$$S \Rightarrow_i^* wA\alpha \begin{cases} \Rightarrow_l w\beta\alpha \Rightarrow_i^* wx \\ \Rightarrow_l w\gamma\alpha \Rightarrow_i^* wy \end{cases}$$

is **determined by the next k symbols** following w .

- Corresponding **computations of $NTA(G)$** :

$$\begin{array}{l} (wx, S, \varepsilon) \vdash^{|w|+|z|} (x, A\alpha, z) \stackrel{(*)}{\vdash} (x, \beta\alpha, zi) \vdash^{|x|+|z'|} (\varepsilon, \varepsilon, ziz') \\ (wy, S, \varepsilon) \vdash^{|w|+|z|} (y, A\alpha, z) \stackrel{(*)}{\vdash} (y, \gamma\alpha, zj) \vdash^{|y|+|z''|} (\varepsilon, \varepsilon, zjz'') \end{array}$$

where $\pi_i = A \rightarrow \beta$ and $\pi_j = A \rightarrow \gamma$

- **Deterministic decision** in $(*)$ possible if $\text{first}_k(x) \neq \text{first}_k(y)$
- **Problem:** how to **determine the A -production** from the lookahead (potentially infinitely many derivations $\beta\alpha \Rightarrow_i^* x$ and $\gamma\alpha \Rightarrow_i^* y$)?

Recap: $LL(k)$ Grammars

The Case $k = 1$

Motivation:

- $k = 1$ sufficient to resolve nondeterminism in “most” practical applications
- Implementation of $LL(k)$ parsers for $k > 1$ rather involved
(cf. **ANTLR** [ANother Tool for Language Recognition; formerly PCCTS] at <http://www.antlr.org/>)

Abbreviations: $fi := first_1$, $fo := follow_1$, $\Sigma_\epsilon := \Sigma \cup \{\epsilon\}$

Corollary

1. For every $\alpha \in X^*$,

$$fi(\alpha) = \{a \in \Sigma \mid \text{ex. } w \in \Sigma^* : \alpha \Rightarrow^* aw\} \cup \{\epsilon \mid \alpha \Rightarrow^* \epsilon\} \subseteq \Sigma_\epsilon$$

2. For every $A \in N$,

$$fo(A) = \{x \in fi(\alpha) \mid \text{ex. } w \in \Sigma^*, \alpha \in X^* : S \Rightarrow_j^* wA\alpha\} \subseteq \Sigma_\epsilon.$$

Recap: $LL(k)$ Grammars

Lookahead Sets

Definition (Lookahead set)

Given $\pi = A \rightarrow \beta \in P$,

$$\text{la}(\pi) := \text{fi}(\beta \cdot \text{fo}(A)) \subseteq \Sigma_\varepsilon$$

is called the **lookahead set** of π (where $\text{fi}(\Gamma) := \bigcup_{\gamma \in \Gamma} \text{fi}(\gamma)$).

Corollary

1. For all $a \in \Sigma$,

$$a \in \text{la}(A \rightarrow \beta) \quad \text{iff} \quad a \in \text{fi}(\beta) \text{ or } (\beta \Rightarrow^* \varepsilon \text{ and } a \in \text{fo}(A))$$

2. $\varepsilon \in \text{la}(A \rightarrow \beta)$ iff $\beta \Rightarrow^* \varepsilon$ and $\varepsilon \in \text{fo}(A)$

Recap: $LL(k)$ Grammars

Characterisation of $LL(1)$

Theorem (Characterisation of $LL(1)$)

$G \in LL(1)$ iff for all pairs of rules $A \rightarrow \beta \mid \gamma \in P$ (where $\beta \neq \gamma$):

$$\text{la}(A \rightarrow \beta) \cap \text{la}(A \rightarrow \gamma) = \emptyset.$$

Proof.

on the board



Remark: the above theorem generally does not hold if $k > 1$ (cf. exercises)

Computing Lookahead Sets

Computing Lookahead Sets I

(see Waite/Goos: *Compiler Construction*, p. 164f)

Lemma 7.1 (Computation of fi/fo)

$\text{fi}(\alpha) \subseteq \Sigma_\varepsilon$ (for $\alpha \in X^*$) and $\text{fo}(A) \subseteq \Sigma_\varepsilon$ (for $A \in N$) are the least sets such that:

1. $\text{fi}(Y)$ for $Y \in X$:

$$- Y \in \Sigma \implies \text{fi}(Y) = \{Y\}$$

$$- Y \in N, Y \rightarrow A_1 \dots A_k Z \alpha \in P \ (k \in \mathbb{N}), Z \in X, \varepsilon \in \text{fi}(A_1) \cap \dots \cap \text{fi}(A_k), a \in \text{fi}(Z)$$

$$\implies a \in \text{fi}(Y)$$

$$- Y \in N, Y \rightarrow A_1 \dots A_k \in P \ (k \in \mathbb{N}), \varepsilon \in \text{fi}(A_1) \cap \dots \cap \text{fi}(A_k) \implies \varepsilon \in \text{fi}(Y)$$

2. $\text{fi}(Y_1 \dots Y_n)$ for $n \in \mathbb{N}$, $Y_i \in X$:

$$- \varepsilon \in \text{fi}(Y_1) \cap \dots \cap \text{fi}(Y_{k-1}), a \in \text{fi}(Y_k) \ (k \in [n]) \implies a \in \text{fi}(Y_1 \dots Y_n)$$

$$- \varepsilon \in \text{fi}(Y_1) \cap \dots \cap \text{fi}(Y_n) \implies \varepsilon \in \text{fi}(Y_1 \dots Y_n)$$

3. $\text{fo}(A)$ for $A \in N$:

$$- \varepsilon \in \text{fo}(S)$$

$$- A \rightarrow \alpha B \beta \in P, a \in \text{fi}(\beta) \implies a \in \text{fo}(B)$$

$$- A \rightarrow \alpha B \beta \in P, \varepsilon \in \text{fi}(\beta), x \in \text{fo}(A) \implies x \in \text{fo}(B)$$

Computing Lookahead Sets

Computing Lookahead Sets II

Corollary 7.2

1. $A \rightarrow a\alpha \in P \implies a \in \text{fi}(A)$
2. $A \rightarrow B\alpha \in P, a \in \text{fi}(B) \implies a \in \text{fi}(A)$
3. $A \rightarrow \varepsilon \in P \implies \varepsilon \in \text{fi}(A)$
4. $a \in \text{fi}(A) \implies a \in \text{fi}(A\alpha)$

Example 7.3

Grammar for
arithmetic expressions
(cf. Example 5.11):

$$G_{AE} : \begin{array}{l} E \rightarrow E+T \mid T \\ T \rightarrow T*F \mid F \\ F \rightarrow (E) \mid a \mid b \end{array}$$

- $F \rightarrow a \in P \implies a \in \text{fi}(F)$
 - $T \rightarrow F \in P, a \in \text{fi}(F) \implies a \in \text{fi}(T)$
 - $a \in \text{fi}(T)$
 $\implies \text{la}(T \rightarrow T*F) = \text{fi}(T*F \cdot \text{fo}(T)) \ni a$
 - $a \in \text{fi}(F)$
 $\implies \text{la}(T \rightarrow F) = \text{fi}(F \cdot \text{fo}(T)) \ni a$
- $\implies a \in \text{la}(T \rightarrow T*F) \cap \text{la}(T \rightarrow F) \neq \emptyset$
 $\implies G_{AE} \notin LL(1)$

Computing Lookahead Sets

Fixing the Problem (general methods later)

Example 7.4 (continuing Example 7.3)

Restructuring (such that $L(G'_{AE}) = L(G_{AE})$):

$$\begin{aligned}
 G'_{AE} : \quad & E \rightarrow TE' \\
 & E' \rightarrow +TE' \mid \varepsilon \\
 & T \rightarrow FT' \\
 & T' \rightarrow *FT' \mid \varepsilon \\
 & F \rightarrow (E) \mid a \mid b
 \end{aligned}$$

$A \in N$	$fi(A)$	$fo(A)$
E	$\{ (, a, b \}$	$\{ \varepsilon,) \}$
E'	$\{ +, \varepsilon \}$	$\{ \varepsilon,) \}$
T	$\{ (, a, b \}$	$\{ +, \varepsilon,) \}$
T'	$\{ *, \varepsilon \}$	$\{ +, \varepsilon,) \}$
F	$\{ (, a, b \}$	$\{ *, +, \varepsilon,) \}$

Remember:

- $\varepsilon \in fo(S)$
- $A \rightarrow \alpha B \beta \in P, a \in fi(\beta)$
 $\implies a \in fo(B)$
- $A \rightarrow \alpha B \beta \in P, \varepsilon \in fi(\beta), x \in fo(A)$
 $\implies x \in fo(B)$

$A \rightarrow \beta \in P$	$la(A \rightarrow \beta) = fi(\beta \cdot fo(A))$
$E \rightarrow TE'$	$\{ (, a, b \}$
$E' \rightarrow +TE'$	$\{ + \}$

Deterministic Top-Down Parsing

The Approach

Deterministic top-down parsing

Given $G \in CFG_{\Sigma}$,

1. verify $G \in LL(1)$ by computing lookahead sets and checking alternatives for disjointness
2. start with nondeterministic top-down parsing automaton $NTA(G)$
3. use **1-symbol lookahead** to trigger the choice of expanding productions:
 - $(aw, A\alpha, z) \vdash (aw, \beta\alpha, zi)$ if $\pi_i = A \rightarrow \beta$ and $a \in \text{la}(\pi_i)$
 - $(\varepsilon, A\alpha, z) \vdash (\varepsilon, \beta\alpha, zi)$ if $\pi_i = A \rightarrow \beta$ and $\varepsilon \in \text{la}(\pi_i)$
 - [matching steps as before: $(aw, a\alpha, z) \vdash (w, \alpha, z)$] \implies **deterministic top-down parsing automaton** $DTA(G)$

Remarks:

- $DTA(G)$ is actually **not a pushdown automaton** (a is inspected but not consumed).
But: can be simulated using the finite control.
- Advantage of using lookahead is **twofold**:
 - removal of nondeterminism
 - earlier detection of syntax errors (in configurations $(aw, A\alpha, z)$ where $a \notin \bigcup_{A \rightarrow \beta \in P} \text{la}(A \rightarrow \beta)$)

Deterministic Top-Down Parsing

The Deterministic Top-Down Automaton I

Definition 7.5 (Deterministic top-down parsing automaton)

Let $G = \langle N, \Sigma, P, S \rangle \in LL(1)$. The **deterministic top-down parsing automaton** of G , $DTA(G)$, is defined by the following components.

- **Input alphabet** Σ , **pushdown alphabet** X , **output alphabet** $[p]$
- **Configurations** $\Sigma^* \times X^* \times [p]^*$, **initial configuration** (w, S, ε) , **final configurations** $\{\varepsilon\} \times \{\varepsilon\} \times [p]^*$ (as $NTA(G)$)
- **Action function** $\text{act} : \Sigma_\varepsilon \times X_\varepsilon \rightarrow \{(\alpha, i) \mid \pi_i = A \rightarrow \alpha\} \cup \{\text{pop}, \text{accept}, \text{error}\}$
with $\text{act}(x, A) := (\alpha, i)$ if $\pi_i = A \rightarrow \alpha$ and $x \in \text{la}(\pi_i)$
 $\text{act}(a, a) := \text{pop}$
 $\text{act}(\varepsilon, \varepsilon) := \text{accept}$
 $\text{act}(x, y) := \text{error}$ otherwise
- **Transitions** for $x \in \Sigma_\varepsilon$, $w \in \Sigma^*$, $Y \in X$, $\beta \in X^*$, and $z \in [p]^*$:

$$(xw, Y\beta, z) \vdash \begin{cases} (xw, \alpha\beta, zi) & \text{if } \text{act}(x, Y) = (\alpha, i) \\ (w, \beta, z) & \text{if } \text{act}(x, Y) = \text{pop} \end{cases}$$

Deterministic Top-Down Parsing

The Deterministic Top-Down Automaton II

Example 7.6 (cf. Example 7.4)

$$\begin{aligned}
 G'_{AE} : E &\rightarrow TE' && (1) \\
 E' &\rightarrow +TE' \mid \varepsilon && (2, 3) \\
 T &\rightarrow FT' && (4) \\
 T' &\rightarrow *FT' \mid \varepsilon && (5, 6) \\
 F &\rightarrow (E) \mid a \mid b && (7, 8, 9)
 \end{aligned}$$

$A \rightarrow \beta \in P$	$la(A \rightarrow \beta)$
$E \rightarrow TE'$	$\{ (, a, b \}$
$E' \rightarrow +TE'$	$\{ + \}$
$E' \rightarrow \varepsilon$	$\{ \varepsilon,) \}$
$T \rightarrow FT'$	$\{ (, a, b \}$
$T' \rightarrow *FT'$	$\{ * \}$
$T' \rightarrow \varepsilon$	$\{ +, \varepsilon,) \}$
$F \rightarrow (E)$	$\{ (\}$
$F \rightarrow a$	$\{ a \}$
$F \rightarrow b$	$\{ b \}$

act : $\Sigma_\varepsilon \times X_\varepsilon \rightarrow \{(\alpha, i) \mid \pi_i = A \rightarrow \alpha\} \cup \{\text{pop, accept, error}\}$ (empty = error)

act	E	E'	T	T'	F	a	b	()	*	+	ε
a	$(TE', 1)$		$(FT', 4)$		$(a, 8)$	pop						
b	$(TE', 1)$		$(FT', 4)$		$(b, 9)$	pop						
($(TE', 1)$		$(FT', 4)$		$((E), 7)$			pop				
)		$(\varepsilon, 3)$		$(\varepsilon, 6)$					pop			
*				$(*FT', 5)$						pop		
+		$(+TE', 2)$		$(\varepsilon, 6)$							pop	
ε		$(\varepsilon, 3)$		$(\varepsilon, 6)$								accept

Deterministic Top-Down Parsing

The Deterministic Top-Down Automaton III

Example 7.6 (continued)

act	E	E'	T	T'	F	a	b	()	*	+	ϵ
a	$(TE', 1)$		$(FT', 4)$		$(a, 8)$	pop						
b	$(TE', 1)$		$(FT', 4)$		$(b, 9)$	pop						
($(TE', 1)$		$(FT', 4)$		$((E), 7)$		pop					
)		$(\epsilon, 3)$		$(\epsilon, 6)$				pop				
*				$(*FT', 5)$					pop			
+	$(+TE', 2)$			$(\epsilon, 6)$						pop		
ϵ		$(\epsilon, 3)$		$(\epsilon, 6)$								accept

Leftmost analysis of $(a)*b$:

$((a)*b, E, \epsilon)$	$\vdash (()*b, E')T'E', 1471486)$
$\vdash ((a)*b, TE', 1)$	$\vdash (()*b,)T'E', 14714863)$
$\vdash ((a)*b, FT'E', 14)$	$\vdash (*b, T'E', 14714863)$
$\vdash ((a)*b, (E)T'E', 147)$	$\vdash (*b, *FT'E', 147148635)$
$\vdash (a)*b, E)T'E', 147)$	$\vdash (b, FT'E', 147148635)$
$\vdash (a)*b, TE')T'E', 1471)$	$\vdash (b, bT'E', 1471486359)$
$\vdash (a)*b, FT'E')T'E', 14714)$	$\vdash (\epsilon, T'E', 1471486359)$
$\vdash (a)*b, aT'E')T'E', 147148)$	$\vdash (\epsilon, E', 14714863596)$
$\vdash ()*b, T'E')T'E', 147148)$	$\vdash (\epsilon, \epsilon, 147148635963)$

Transformation to $LL(1)$

Transformation to $LL(1)$

Goal: transformation of $G = \langle N, \Sigma, P, S \rangle \in CFG_{\Sigma} \setminus LL(1)$

(i.e., $la(A \rightarrow \beta) \cap la(A \rightarrow \gamma) \neq \emptyset$ for some $A \rightarrow \beta \mid \gamma \in P$) to $G' \in LL(1)$ with $L(G') = L(G)$

Problems

- Transformations generally **preserve the semantics** (= generated language) of CFGs but **not the syntactic structure** of words (different syntax trees).
- Transformations **cannot always yield an $LL(1)$ grammar** since not every context-free language is generated by an LL grammar. Example of $L \notin \bigcup_{k \in \mathbb{N}} L(LL(k))$:
 $\{a^n 0 b^n \mid n \geq 1\} \cup \{a^n 1 b^{2n} \mid n \geq 1\}$ (unbounded number of as required for decision)
- Even worse: given an arbitrary CFG it is **undecidable** whether there is $G \in LL(k)$ (for some k) which generates the same language (cf. D.J. Rosenkrantz, R.E. Stearns: *Properties of Deterministic Top Down Grammars*, Information and Control 17:226–256, 1970)

Heuristics for transforming G into $G' \in LL(1)$

1. Elimination of left recursion
2. Left factorization

(used in parser-generating systems such as ANTLR)

Transformation to $LL(1)$

Left Recursion I

Definition 7.7 (Left recursion)

A grammar $G = \langle N, \Sigma, P, S \rangle \in CFG_{\Sigma}$ is called **left recursive** if there exist $A \in N$ and $\alpha \in X^*$ such that $A \Rightarrow^+ A\alpha$.

Corollary 7.8

If $G \in CFG_{\Sigma}$ is left recursive with $A \Rightarrow^+ A\alpha$, then there exists $\beta \in X^$ such that $A \Rightarrow_i^+ A\beta$.*

Example 7.9

The grammar (cf. Example 5.11) $G_{AE} : \begin{array}{l} E \rightarrow E+T \mid T \\ T \rightarrow T*F \mid F \\ F \rightarrow (E) \mid a \mid b \end{array}$

is left recursive, and in Example 7.3 it was shown that $G_{AE} \notin LL(1)$

Transformation to $LL(1)$

Left Recursion II

Lemma 7.10

If $G \in CFG_{\Sigma}$ is left recursive, then $G \notin \bigcup_{k \in \mathbb{N}} LL(k)$.

Proof.

(for $k = 1$) Assume that $G \in LL(1)$ is left recursive with $A \Rightarrow_i^+ A\beta$. Together with the reducedness of G this implies that $S \Rightarrow_i^* vA\alpha \Rightarrow_i^+ vA\beta\alpha \Rightarrow_i^+ vw$ for some $v, w \in \Sigma^*$ and $\alpha \in X^*$.

The corresponding computation of $DTA(G)$ (Definition 7.5) starts with $(vw, S, \varepsilon) \vdash^* (w, A\alpha, \dots) \vdash^+ (w, A\beta\alpha, \dots)$.

But in the last configuration the behaviour of $DTA(G)$ is determined by the same input ($\text{fi}(w)$) and stack symbol (A). Thus it enters an infinite **loop** of the form

$$(w, A\alpha, \dots) \vdash^+ (w, A\beta\alpha, \dots) \vdash^+ (w, A\beta\beta\alpha, \dots) \vdash^+ \dots$$

and will never recognize w . ⚡



Transformation to $LL(1)$

Eliminating Direct Left Recursion

Direct left recursion occurs in productions of the form

$$A \rightarrow A\alpha_1 \mid \dots \mid A\alpha_m \mid \beta_1 \mid \dots \mid \beta_n \quad \text{where } \alpha_i \neq \varepsilon \text{ and } \beta_j \neq A\dots$$

Transformation: replacement by **right recursion**

$$\begin{aligned} A &\rightarrow \beta_1 A' \mid \dots \mid \beta_n A' \\ A' &\rightarrow \alpha_1 A' \mid \dots \mid \alpha_m A' \mid \varepsilon \end{aligned}$$

(with a new $A' \in N$) which preserves $L(G)$.

Example 7.11

$$\begin{array}{l} G_{AE} : E \rightarrow E+T \mid T \\ \quad T \rightarrow T*F \mid F \\ \quad F \rightarrow (E) \mid a \mid b \end{array} \quad \text{is transformed into} \quad \begin{array}{l} G'_{AE} : E \rightarrow TE' \\ \quad E' \rightarrow +TE' \mid \varepsilon \\ \quad T \rightarrow FT' \\ \quad T' \rightarrow *FT' \mid \varepsilon \\ \quad F \rightarrow (E) \mid a \mid b \end{array}$$

with $G'_{AE} \in LL(1)$ (see Example 7.4).

Transformation to $LL(1)$

Eliminating Indirect Left Recursion I

Indirect left recursion occurs in productions of the form ($k \geq 1$)

$$A \rightarrow A_1\alpha_1 \quad A_1 \rightarrow A_2\alpha_2 \quad \dots \quad A_{k-1} \rightarrow A_k\alpha_k \quad A_k \rightarrow A\beta$$

Systematic elimination of left recursion from a grammar without cycles ($A \Rightarrow^+ A$) and ε -productions ($A \rightarrow \varepsilon$) (which can both be eliminated systematically):

Algorithm 7.12 (Elimination of left recursion)

Input: $G = \langle N, \Sigma, P, S \rangle \in CFG_\Sigma$ with $N = \{B_1, \dots, B_n\}$

Procedure: **for** $i := 1$ **to** n **do**

for $j := 1$ **to** $i - 1$ **do**

 replace each $B_i \rightarrow B_j\gamma$ by $B_i \rightarrow \delta_1\gamma \mid \dots \mid \delta_k\gamma$,

 where $B_j \rightarrow \delta_1 \mid \dots \mid \delta_k$ are all current B_j -productions

end

 eliminate direct left recursion among B_i -productions

end

Output: equivalent grammar G' without left recursion (possibly with ε -productions)

Transformation to $LL(1)$

Eliminating Indirect Left Recursion II

Example 7.13

1. Original grammar ($B_1 = S, B_2 = A$):

$$S \rightarrow Sa \mid Ab \mid c$$
$$A \rightarrow Ad \mid Se \mid f$$

2. $i = 1$: After B_1 -substitution (no j): After removal of direct left B_1 -recursion:

$$S \rightarrow Sa \mid Ab \mid c$$
$$A \rightarrow Ad \mid Se \mid f$$
$$S \rightarrow AbS' \mid cS'$$
$$S' \rightarrow aS' \mid \varepsilon$$
$$A \rightarrow Ad \mid Se \mid f$$

3. $i = 2$: After B_2 -substitution ($j = 1$): After removal of direct left B_2 -recursion:

$$S \rightarrow AbS' \mid cS'$$
$$S' \rightarrow aS' \mid \varepsilon$$
$$A \rightarrow Ad \mid AbS'e \mid cS'e \mid f$$
$$S \rightarrow AbS' \mid cS'$$
$$S' \rightarrow aS' \mid \varepsilon$$
$$A \rightarrow cS'eA' \mid fA'$$
$$A' \rightarrow dA' \mid bS'eA' \mid \varepsilon$$

(Remark: $G' \notin LL(1)$ since $\text{la}(S \rightarrow AbS') \cap \text{la}(S \rightarrow cS') = \{c\} \neq \emptyset$)

Transformation to $LL(1)$

Left Factorization

Applies to productions of the form

$$A \rightarrow \alpha\beta \mid \alpha\gamma$$

which are problematic if α “at least as long as lookahead”.

Transformation: delaying the decision by **left factorization**

$$A \rightarrow \alpha A' \quad A' \rightarrow \beta \mid \gamma$$

(with a new $A' \in N$) which preserves $L(G)$.

Example 7.14

Statement \rightarrow if *Condition* then *Statement* else *Statement* fi
| if *Condition* then *Statement* fi

is transformed into

Statement \rightarrow if *Condition* then *Statement* *S'*
S' \rightarrow else *Statement* fi | fi

The Complexity of $LL(1)$ Parsing

The Complexity of $LL(1)$ Parsing I

- $LL(1)$ parsing has time (and hence space) **complexity** $\mathcal{O}(|w|)$ (where $w \in \Sigma^*$ is the input word)
- Here: proof for **ε -free grammars** (i.e., without ε -productions)
- General case: see O. Mayer: *Syntaxanalyse*, p. 211ff

Lemma 7.15

Let $G = \langle N, \Sigma, P, S \rangle \in LL(1)$ be without ε -productions. If

$$(w, S, \varepsilon) \vdash^n (\varepsilon, \varepsilon, z)$$

in $DTA(G)$, then

$$n \leq (|w| + 1) \cdot (|N| + 1).$$

The Complexity of $LL(1)$ Parsing

The Complexity of $LL(1)$ Parsing II

Proof.

Let $(w, S, \varepsilon) \vdash^n (\varepsilon, \varepsilon, z)$ in $DTA(G)$. To show: $n \leq (|w| + 1) \cdot (|N| + 1)$

1. Clear: the computation involves $|w|$ matching steps.
2. Since G is ε -free, every matching step is preceded (and followed) by $k \geq 0$ expansion steps of the form

$$\begin{aligned} (av, A_1\alpha_1, \dots) &\vdash (av, A_2\alpha_2\alpha_1, \dots) \\ &\vdots \\ &\vdash (av, A_k\alpha_k \dots \alpha_1, \dots) \\ &\vdash (av, a\alpha_{k+1} \dots \alpha_1, \dots) \end{aligned}$$

where $A_i \rightarrow A_{i+1}\alpha_{i+1}$ for each $i \in [k - 1]$ and $A_k \rightarrow a\alpha_{k+1}$.

3. This implies that $A_i \neq A_j$ for $i \neq j$ (by Lemma 7.10, G is not left recursive), and hence $k \leq |N|$.
4. Altogether: $n \leq (|w| + 1) \cdot (|N| + 1)$.



Recursive-Descent Parsing

Recursive-Descent Parsing I

Idea: avoid explicit use of pushdown store (as in $DTA(G)$) by employing **recursive procedures** (with implicit runtime stack)

Advantage: simple implementation

Ingredients: • variable `token` for current token

- function `next()` for invoking the scanner
- procedure `print(i)` for displaying the leftmost analysis (or errors)

Method: to every $A \in N$ we assign a procedure $A()$ which

- tests `token` with regard to the lookahead sets of the A -productions,
- prints the corresponding rule number and
- evaluates the corresponding right-hand side as follows:
 - for $a \in \Sigma$: match `token`; call `next()`
 - for $A \in N$: call $A()$

Recursive-Descent Parsing II

Example 7.16 (Arithmetic expressions; cf. Example 7.11)

```
proc main();
  token := next(); E()
proc E();   (*  $E \rightarrow T E'$  *)
  if token in {'(', 'a', 'b'} then print(1); T(); E'()
  else print(error); stop fi
proc E'();  (*  $E' \rightarrow + T E' \mid \varepsilon$  *)
  if token = '+' then print(2); token := next(); T(); E'()
  elsif token in {EOF, ')'} then print(3)
  else print(error); stop fi
proc T();   (*  $T \rightarrow F T'$  *)
  if token in {'(', 'a', 'b'} then print(4); F(); T'()
  else print(error); stop fi
proc T'();  (*  $T' \rightarrow * F T' \mid \varepsilon$  *)
  if token = '*' then print(5); token := next(); F(); T'()
  elsif token in {'+', EOF, ')'} then print(6)
  else print(error); stop fi
proc F();   (*  $F \rightarrow ( E ) \mid a \mid b$  *)
  if token = '(' then print(7); token := next(); E();
    if token = ')' then token := next() else print(error); stop fi
  elsif token = 'a' then print(8); token := next()
  elsif token = 'b' then print(9); token := next()
  else print(error); stop fi
```