Compiler Construction

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

Summer Semester 2016

Thomas Noll
Software Modeling and Verification Group
RWTH Aachen University

https://moves.rwth-aachen.de/teaching/ss-16/cc/

$LL(k)$ Grammars I

$LL(k)$: reading of input from Left to right with k-lookahead, computing a Leftmost 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

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

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).

$LL(k)$ Grammars II

Remarks:

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

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

  is determined by the next k symbols following w.
- Corresponding computations of $NTA(G)$:

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

  where $\pi_i = A \to \beta$ and $\pi_j = A \to \gamma$
- 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 $\beta\alpha \Rightarrow_l^* x$ / $\gamma\alpha \Rightarrow_l^* y$)?

$LL(k)$ Grammars III

Lemma (Characterisation of $LL(k)$)

$G \in LL(k)$ iff for all leftmost derivations of the form

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

such that $\beta \neq \gamma$, it follows that $\mathrm{first}_k(\beta\alpha) \cap \mathrm{first}_k(\gamma\alpha) = \emptyset$.

Proof.

omitted □

Remarks:

- If $G \in LL(k)$, then the A-production is determined by the lookahead sets $\mathrm{first}_k(\beta\alpha)$ (for every $A \to \beta \in P$).
- **Problem:** still infinitely many right contexts α to be considered (if β [or γ] "too short", i.e., $\mathrm{first}_k(\beta\alpha) \neq \mathrm{first}_k(\beta)$).
- **Idea:** α derives to "everything that follows A".

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: $\mathrm{fi} := \mathrm{first}_1$, $\mathrm{fo} := \mathrm{follow}_1$, $\Sigma_\varepsilon := \Sigma \cup \{\varepsilon\}$

Corollary

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

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

2. *For every* $A \in N$,

$$\mathrm{fo}(A) = \{x \in \mathrm{fi}(\alpha) \mid \textit{ex. } w \in \Sigma^*, \alpha \in X^* : S \Rightarrow_l^* wA\alpha\} \subseteq \Sigma_\varepsilon.$$

Recap: $LL(k)$ Grammars

Lookahead Sets

Definition (Lookahead set)

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

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

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

Corollary

1. *For all* $a \in \Sigma$,
$$a \in \mathrm{la}(A \to \beta) \textit{ iff } a \in \mathrm{fi}(\beta) \textit{ or } (\beta \Rightarrow^* \varepsilon \textit{ and } a \in \mathrm{fo}(A))$$
2. $\varepsilon \in \mathrm{la}(A \to \beta)$ *iff* $\beta \Rightarrow^* \varepsilon$ *and* $\varepsilon \in \mathrm{fo}(A)$

Recap: $LL(k)$ Grammars

Characterisation of $LL(1)$

Theorem (Characterisation of $LL(1)$)

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

$$\mathrm{la}(A \to \beta) \cap \mathrm{la}(A \to \gamma) = \emptyset.$$

Proof.

on the board □

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

Computing Lookahead Sets I

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

Lemma 7.1 (Computation of fi/fo)

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

1. $\mathrm{fi}(Y)$ *for* $Y \in X$*:*
   - $Y \in \Sigma \implies \mathrm{fi}(Y) = \{Y\}$
   - $Y \to A_1 \ldots A_k Z\alpha \in P, k \in \mathbb{N}, Z \in X, \varepsilon \in \mathrm{fi}(A_1) \cap \ldots \cap \mathrm{fi}(A_k),$ $a \in \mathrm{fi}(Z) \implies a \in \mathrm{fi}(Y)$
   - $Y \to A_1 \ldots A_k \in P, k \in \mathbb{N}, \varepsilon \in \mathrm{fi}(A_1) \cap \ldots \cap \mathrm{fi}(A_k) \implies \varepsilon \in \mathrm{fi}(Y)$
2. $\mathrm{fi}(Y_1 \ldots Y_n)$ *for* $n \in \mathbb{N}, Y_i \in X$*:*
   - $\varepsilon \in \mathrm{fi}(Y_1) \cap \ldots \cap \mathrm{fi}(Y_{k-1}), a \in \mathrm{fi}(Y_k), k \in [n] \implies a \in \mathrm{fi}(Y_1 \ldots Y_n)$
   - $\varepsilon \in \mathrm{fi}(Y_1) \cap \ldots \cap \mathrm{fi}(Y_n) \implies \varepsilon \in \mathrm{fi}(Y_1 \ldots Y_n)$
3. $\mathrm{fo}(A)$ *for* $A \in N$*:*
   - $\varepsilon \in \mathrm{fo}(S)$
   - $A \to \alpha B \beta \in P, a \in \mathrm{fi}(\beta) \implies a \in \mathrm{fo}(B)$
   - $A \to \alpha B \beta \in P, \varepsilon \in \mathrm{fi}(\beta), x \in \mathrm{fo}(A) \implies x \in \mathrm{fo}(B)$

Computing Lookahead Sets II

Corollary 7.2

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

Example 7.3

Grammar for arithmetic expressions (cf. Example 5.11):

$$G_{AE}: \quad \begin{aligned} E &\to E{+}T \mid T \\ T &\to T{*}F \mid F \\ F &\to (E) \mid \mathtt{a} \mid \mathtt{b} \end{aligned}$$

- $F \to \mathtt{a} \in P \implies \mathtt{a} \in \mathrm{fi}(F)$
- $T \to F \in P, \mathtt{a} \in \mathrm{fi}(F) \implies \mathtt{a} \in \mathrm{fi}(T)$
- $\mathtt{a} \in \mathrm{fi}(T)$
  $\implies \mathrm{la}(T \to T{*}F) = \mathrm{fi}(T{*}F \cdot \mathrm{fo}(T)) \ni \mathtt{a}$
- $\mathtt{a} \in \mathrm{fi}(F)$
  $\implies \mathrm{la}(T \to F) = \mathrm{fi}(F \cdot \mathrm{fo}(T)) \ni \mathtt{a}$

$\Rightarrow \mathtt{a} \in \mathrm{la}(T \to T{*}F) \cap \mathrm{la}(T \to F) \neq \emptyset$

$\Rightarrow G_{AE} \notin LL(1)$

Computing Lookahead Sets

Fixing the Problem (general methods later)

Example 7.4 (continuing Example 7.3)

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

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

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

Remember:

- $\varepsilon \in \mathrm{fo}(S)$
- $A \to \alpha B \beta \in P, a \in \mathrm{fi}(\beta)$
  $\implies a \in \mathrm{fo}(B)$
- $A \to \alpha B \beta \in P, \varepsilon \in \mathrm{fi}(\beta), x \in \mathrm{fo}(A)$
  $\implies x \in \mathrm{fo}(B)$

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

Deterministic Top-Down Parsing

Approach: given $G \in CFG_\Sigma$,

1. Verify that $G \in LL(1)$ by computing the lookahead sets and checking alternatives for disjointness
2. Start with nondeterministic top-down parsing automaton $NTA(G)$
3. Use 1-symbol lookahead to control the choice of expanding productions:
   - $(aw, A\alpha, z) \vdash (aw, \beta\alpha, zi)$
     if $\pi_i = A \to \beta$ and $a \in \mathrm{la}(\pi_i)$
   - $(\varepsilon, A\alpha, z) \vdash (\varepsilon, \beta\alpha, zi)$
     if $\pi_i = A \to \beta$ and $\varepsilon \in \mathrm{la}(\pi_i)$
   - [matching steps as before: $(aw, a\alpha, z) \vdash (w, \alpha, z)$]

   $\Longrightarrow$ deterministic top-down parsing automaton $\mathrm{DTA}(G)$

Remarks:

- $\mathrm{DTA}(G)$ is actually not a pushdown automaton (a is read 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 $a \notin \bigcup_{A \to \beta \in P} \mathrm{la}(A \to \beta)$)

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, $\mathrm{DTA}(G)$, 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 $\mathrm{act} : \Sigma_\varepsilon \times X_\varepsilon \to \{(\alpha, i) \mid \pi_i = A \to \alpha\} \cup \{\mathrm{pop}, \mathrm{accept}, \mathrm{error}\}$

$$\begin{aligned} \text{with } \mathrm{act}(x, A) &:= (\alpha, i) \text{ if } \pi_i = A \to \alpha \text{ and } x \in \mathrm{la}(\pi_i) \\ \mathrm{act}(a, a) &:= \mathrm{pop} \\ \mathrm{act}(\varepsilon, \varepsilon) &:= \mathrm{accept} \\ \mathrm{act}(x, y) &:= \mathrm{error} \qquad \text{otherwise} \end{aligned}$$

- Transitions for $x \in \Sigma_\varepsilon$, $w \in \Sigma^*$, $Y \in X$, $\beta \in X^*$, and $z \in [p]^*$:

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

The Deterministic Top-Down Automaton II

Example 7.6 (cf. Example 7.4)

G'_{AE} :

$$
\begin{array}{lll}
E & \to TE' & (1) \\
E' & \to +TE' \mid \varepsilon & (2,3) \\
T & \to FT' & (4) \\
T' & \to *FT' \mid \varepsilon & (5,6) \\
F & \to (E) \mid \mathtt{a} \mid \mathtt{b} & (7,8,9)
\end{array}
$$

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

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

act	E	E'	T	T'	F	a	b	(	)	*	+	ε
a	$(TE', 1)$		$(FT', 4)$		$(\mathtt{a}, 8)$	pop						
b	$(TE', 1)$		$(FT', 4)$		$(\mathtt{b}, 9)$		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

The Deterministic Top-Down Automaton III

Example 7.6 (continued)

act	E	E'	T	T'	F	a	b	(	)	*	+	ε
a	$(TE', 1)$		$(FT', 4)$		$(\mathtt{a}, 8)$	pop						
b	$(TE', 1)$		$(FT', 4)$		$(\mathtt{b}, 9)$		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

Leftmost analysis of `(a)*b`:

$$
\begin{array}{ll}
 & ((\mathtt{a})*\mathtt{b}, E, \varepsilon) \\
\vdash & ((\mathtt{a})*\mathtt{b}, TE', 1) \\
\vdash & ((\mathtt{a})*\mathtt{b}, FT'E', 14) \\
\vdash & ((\mathtt{a})*\mathtt{b}, (E)T'E', 147) \\
\vdash & (\ \mathtt{a})*\mathtt{b}, E)T'E', 147) \\
\vdash & (\ \mathtt{a})*\mathtt{b}, TE')T'E', 1471) \\
\vdash & (\ \mathtt{a})*\mathtt{b}, FT'E')T'E', 14714) \\
\vdash & (\ \mathtt{a})*\mathtt{b}, \mathtt{a}T'E')T'E', 147148) \\
\vdash & (\ \ )*\mathtt{b}, T'E')T'E', 147148)
\end{array}
\qquad
\begin{array}{ll}
\vdash & (\ )*\mathtt{b}, E')T'E', 1471486) \\
\vdash & (\ )*\mathtt{b},)T'E', 14714863) \\
\vdash & (\ *\mathtt{b}, T'E', 14714863) \\
\vdash & (\ *\mathtt{b}, *FT'E', 147148635) \\
\vdash & (\ \ \mathtt{b}, FT'E', 147148635) \\
\vdash & (\ \ \mathtt{b}, \mathtt{b}T'E', 1471486359) \\
\vdash & (\ \ \varepsilon, T'E', 1471486359) \\
\vdash & (\ \ \varepsilon, E', 14714863596) \\
\vdash & (\ \ \varepsilon, \varepsilon, 147148635963)
\end{array}
$$

Transformation to $LL(1)$

Assume that $G = \langle N, \Sigma, P, S \rangle \in CFG_\Sigma \setminus LL(1)$
(i.e., there exist $A \to \beta \mid \gamma \in P$ such that $\mathrm{la}(A \to \beta) \cap \mathrm{la}(A \to \gamma) \neq \emptyset$)

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. Removal of left recursion
2. Left factorization

(used in parser-generating systems such as ANTLR)

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 $A \Rightarrow_l^+ A\beta$.

Example 7.9

The grammar (cf. Example 5.11) G_{AE} :

$$\begin{aligned} E &\to E{+}T \mid T \\ T &\to T{*}F \mid F \\ F &\to (E) \mid \mathtt{a} \mid \mathtt{b} \end{aligned}$$

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

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 $A \Rightarrow_l^+ A\beta$. Together with the reducedness of G this implies that $S \Rightarrow_l^* vA\alpha \Rightarrow_l^+ vA\beta\alpha \Rightarrow_l^+ vw$ for some $v, w \in \Sigma^*$ and $\alpha \in X^*$.
The corresponding computation of $\mathrm{DTA}(G)$ (Definition 7.5) starts with $(vw, S, \varepsilon) \vdash^* (w, A\alpha, \ldots) \vdash^+ (w, A\beta\alpha, \ldots)$.
But in the last state the behaviour of $\mathrm{DTA}(G)$ is determined by the same input ($\mathrm{fi}(w)$) and stack symbol (A). Thus it enters a loop of the form

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

and will never recognize w. ↯ □

Removing Direct Left Recursion

Direct left recursion occurs in productions of the form

$$A \to A\alpha_1 \mid \ldots \mid A\alpha_m \mid \beta_1 \mid \ldots \mid \beta_n \qquad \text{where } \alpha_i \neq \varepsilon \text{ and } \beta_j \neq A\ldots$$

Transformation: replacement by right recursion

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

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

Example 7.11

G_{AE} : $E \to E{+}T \mid T$, $T \to T{*}F \mid F$, $F \to (E) \mid \mathtt{a} \mid \mathtt{b}$ is transformed into G'_{AE} :

$$\begin{aligned} E &\to TE' \\ E' &\to +TE' \mid \varepsilon \\ T &\to FT' \\ T' &\to *FT' \mid \varepsilon \\ F &\to (E) \mid \mathtt{a} \mid \mathtt{b} \end{aligned}$$

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

Removing Indirect Left Recursion

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

$$\begin{aligned} A &\to A_1\alpha_1 \mid \dots \\ A_1 &\to A_2\alpha_2 \mid \dots \\ &\vdots \\ A_{n-1} &\to A_n\alpha_n \mid \dots \\ A_n &\to A\beta \mid \dots \end{aligned}$$

Transformation: into Greibach Normal Form with productions of the form

$A \to aB_1 \dots B_n$ (where $n \in \mathbb{N}$ and each $B_i \neq S$) or
$S \to \varepsilon$

(cf. *Formale Systeme, Automaten, Prozesse*)

Left Factorization

Applies to productions of the form

$$A \to \alpha\beta \mid \alpha\gamma$$

which are problematic if α "at least as long as lookahead".

Transformation: delaying the decision by left factorization

$$A \to \alpha A' \qquad A' \to \beta \mid \gamma$$

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

Example 7.12

$$\begin{aligned} \textit{Statement} \to\ & \text{if } \textit{Condition} \text{ then } \textit{Statement} \text{ else } \textit{Statement} \text{ fi} \\ \mid\ & \text{if } \textit{Condition} \text{ then } \textit{Statement} \text{ fi} \end{aligned}$$

is transformed into

$$\begin{aligned} \textit{Statement} &\to \text{if } \textit{Condition} \text{ then } \textit{Statement}\ S' \\ S' &\to \text{else } \textit{Statement} \text{ fi} \mid \text{fi} \end{aligned}$$

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., $A \to \alpha \in P \implies \alpha \neq \varepsilon$)
- General case: see O. Mayer: *Syntaxanalyse*, p. 211ff

Lemma 7.13

Let $G = \langle N, \Sigma, P, S \rangle \in LL(1)$ *be* ε*-free. If*

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

in $\mathrm{DTA}(G)$*, then*

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

The Complexity of $LL(1)$ Parsing II

Proof.

Let $(w, S, \varepsilon) \vdash^n (\varepsilon, \varepsilon, z)$ in $\mathrm{DTA}(G)$. 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

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

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

Idea: avoid explicit use of pushdown store (as in $\mathrm{DTA}(G)$) 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 7.14 (Arithmetic expressions; cf. Example 7.11)

```
proc main();
  token := next(); E()
proc E();    (* E → T E' *)
  if token in {'(','a','b'} then print(1); T(); E'()
  else print(error); stop fi
proc E'();   (* E' → + T E' | ε *)
  if token = '+' then print(2); token := next(); T(); E'()
  elsif token in {EOF, ')'} then print(3)
  else print(error); stop fi
proc T();    (* T → F T' *)
  if token in {'(','a','b'} then print(4); F(); T'()
  else print(error); stop fi
proc T'();   (* T' → * F T' | ε *)
  if token = '*' then print(5); token := next(); F(); T'()
  elsif token in {'+',EOF,')'} then print(6)
  else print(error); stop fi
proc F();    (* F → ( E ) | a | 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
```