

Compiler Construction

Lecture 8: Syntax Analysis IV

(Transformation to $LL(1)$ & Bottom-Up Parsing)

Thomas Noll

Lehrstuhl für Informatik 2
(Software Modeling and Verification)

RWTH Aachen University

noll@cs.rwth-aachen.de

<http://www-i2.informatik.rwth-aachen.de/i2/cc12/>

Summer Semester 2012

- 1 Repetition: $LL(1)$ Parsing
- 2 Transformation to $LL(1)$
- 3 The Complexity of $LL(1)$ Parsing
- 4 Recursive-Descent Parsing
- 5 Bottom-Up Parsing
- 6 Nondeterministic Bottom-Up Parsing

Characterization of $LL(1)$

Definition (Lookahead set)

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

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

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

Theorem (Characterization of $LL(1)$)

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

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

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 $\pi_i = A \rightarrow \beta$ and $a \in la(\pi_i)$
 - $(\varepsilon, A\alpha, z) \vdash (\varepsilon, \beta\alpha, zi)$
if $\pi_i = A \rightarrow \beta$ and $\varepsilon \in la(\pi_i)$
 - [matching steps as before: $(aw, a\alpha, z) \vdash (w, \alpha, z)$] $\Rightarrow$ **deterministic top-down parsing automaton** $DTA(G)$

Remarks:

- $DTA(G)$ 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 $a \notin \bigcup_{A \rightarrow \beta \in P} la(A \rightarrow \beta)$)

- 1 Repetition: $LL(1)$ Parsing
- 2 Transformation to $LL(1)$
- 3 The Complexity of $LL(1)$ Parsing
- 4 Recursive-Descent Parsing
- 5 Bottom-Up Parsing
- 6 Nondeterministic Bottom-Up Parsing

Transformation to $LL(1)$

Assume that $G = \langle N, \Sigma, P, S \rangle \in CFG_{\Sigma} \setminus LL(1)$

(i.e., there exist $A \rightarrow \beta \mid \gamma \in P$ such that $la(A \rightarrow \beta) \cap la(A \rightarrow \gamma) \neq \emptyset$)

Two **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)

Remarks:

- 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; details later).

Definition 8.1 (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 8.2

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 8.3

The grammar (cf. Example 5.10)

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

is left recursive, and in Example 7.4 it was shown that $G_{AE} \notin LL(1)$

Lemma 8.4

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 $DTA(G)$ (Def. 7.6) starts with $(vw, S, \varepsilon) \vdash^* (w, A\alpha, \dots) \vdash^+ (w, A\beta\alpha, \dots)$.

But in the last state the behaviour of $DTA(G)$ is determined by the same input ($\text{fi}(w)$) and stack symbol (A). Thus it enters a 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 . Contradiction □

Removing 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 8.5

$$\begin{aligned} G_{AE} : \quad & E \rightarrow E+T \mid T \\ & T \rightarrow T*F \mid F \\ & F \rightarrow (E) \mid a \mid b \end{aligned} \quad \text{is transformed into}$$

$$\begin{aligned} G'_{AE} : \quad & E \rightarrow TE' \\ & E' \rightarrow +TE' \mid \varepsilon \\ & T \rightarrow FT' \\ & T' \rightarrow *FT' \mid \varepsilon \\ & F \rightarrow (E) \mid a \mid b \end{aligned} \quad \text{with } G'_{AE} \in LL(1) \text{ (see Example 7.5).}$$

Removing Indirect Left Recursion

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

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

Transformation: into Greibach Normal Form with productions of the form

$$\begin{aligned} A &\rightarrow aB_1 \dots B_n \quad (\text{where } n \in \mathbb{N} \text{ and each } B_i \neq S) \text{ or} \\ S &\rightarrow \varepsilon \end{aligned}$$

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

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**

$$\begin{aligned} A &\rightarrow \alpha A' \\ A' &\rightarrow \beta \mid \gamma \end{aligned}$$

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

Example 8.6

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

- 1 Repetition: $LL(1)$ Parsing
- 2 Transformation to $LL(1)$
- 3 The Complexity of $LL(1)$ Parsing**
- 4 Recursive-Descent Parsing
- 5 Bottom-Up Parsing
- 6 Nondeterministic Bottom-Up 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., $A \rightarrow \alpha \in P \implies \alpha \neq \varepsilon$)
- General case: see O. Mayer: *Syntaxanalyse*, p. 211ff

Lemma 8.7

Let $G = \langle N, \Sigma, P, S \rangle \in LL(1)$ be ε -free. 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 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 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 8.4, G is not left recursive), and hence $k \leq |N|$.
- 4 Altogether: $n \leq (|w| + 1) \cdot (|N| + 1)$.



- 1 Repetition: $LL(1)$ Parsing
- 2 Transformation to $LL(1)$
- 3 The Complexity of $LL(1)$ Parsing
- 4 Recursive-Descent Parsing
- 5 Bottom-Up Parsing
- 6 Nondeterministic Bottom-Up 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()**

Example 8.8 (Arithmetic expressions; cf. Example 8.5)

```
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'();  (*  $E' \rightarrow + T E' \mid \epsilon$  *)
  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'();  (*  $T' \rightarrow * F T' \mid \epsilon$  *)
  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
```

- 1 Repetition: $LL(1)$ Parsing
- 2 Transformation to $LL(1)$
- 3 The Complexity of $LL(1)$ Parsing
- 4 Recursive-Descent Parsing
- 5 Bottom-Up Parsing**
- 6 Nondeterministic Bottom-Up Parsing

Repetition: Top-Down Parsing

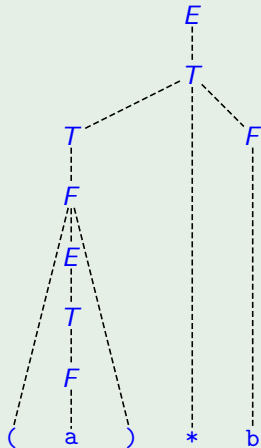
Example 8.9

Grammar for
arithmetic expressions:

$$\begin{aligned} G_{AE} : \quad E &\rightarrow E+T \mid T & (1, 2) \\ T &\rightarrow T*F \mid F & (3, 4) \\ F &\rightarrow (E) \mid a \mid b & (5, 6, 7) \end{aligned}$$

Leftmost analysis of $(a)*b$:

2 3 4 5 2 4 6 7



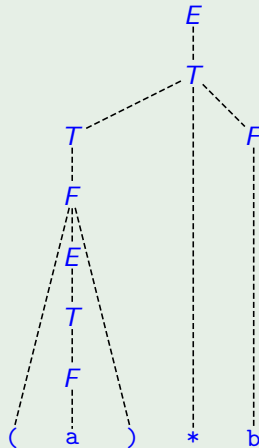
Example 8.10

Grammar for
arithmetic expressions:

$$\begin{aligned} G_{AE} : \quad E &\rightarrow E+T \mid T & (1, 2) \\ T &\rightarrow T*F \mid F & (3, 4) \\ F &\rightarrow (E) \mid a \mid b & (5, 6, 7) \end{aligned}$$

Reversed rightmost analysis
of $(a)*b$:

6 4 2 5 4 7 3 2



Approach:

- 1 Given $G \in CFG_{\Sigma}$, construct a **nondeterministic bottom-up parsing automaton** (NBA) which accepts $L(G)$ and which additionally computes corresponding (reversed) rightmost analyses
 - input alphabet: Σ
 - pushdown alphabet: X
 - output alphabet: $[p]$ (where $p := |P|$)
 - state set: omitted
 - transitions:
 - shift**: shifting input symbols onto the pushdown
 - reduce**: replacing the right-hand side of a production by its left-hand side (= inverse expansion steps)
- 2 Remove nondeterminism by allowing **lookahead** on the input:
 $G \in LR(k)$ iff $L(G)$ recognizable by deterministic bottom-up parsing automaton with lookahead of k symbols

- 1 Repetition: $LL(1)$ Parsing
- 2 Transformation to $LL(1)$
- 3 The Complexity of $LL(1)$ Parsing
- 4 Recursive-Descent Parsing
- 5 Bottom-Up Parsing
- 6 Nondeterministic Bottom-Up Parsing

Definition 8.11 (Nondeterministic bottom-up parsing automaton)

Let $G = \langle N, \Sigma, P, S \rangle \in CFG_{\Sigma}$. The **nondeterministic bottom-up parsing automaton** of G , $NBA(G)$, is defined by the following components.

- **Input alphabet:** Σ
- **Pushdown alphabet:** X
- **Output alphabet:** $[p]$
- **Configurations:** $\Sigma^* \times X^* \times [p]^*$ (top of pushdown to the right)
- **Transitions** for $w \in \Sigma^*$, $\alpha \in X^*$, and $z \in [p]^*$:
 - shifting steps: $(aw, \alpha, z) \vdash (w, \alpha a, z)$ if $a \in \Sigma$
 - reduction steps: $(w, \alpha\beta, z) \vdash (w, \alpha A, zi)$ if $\pi_i = A \rightarrow \beta$
- **Initial configuration** for $w \in \Sigma^*$: $(w, \varepsilon, \varepsilon)$
- **Final configurations:** $\{\varepsilon\} \times \{S\} \times [p]^*$

Example 8.12

Grammar for
arithmetic expressions
(cf. Example 8.10):

$$\begin{aligned}
 G_{AE} : E &\rightarrow E+T \mid T & (1, 2) \\
 T &\rightarrow T*F \mid F & (3, 4) \\
 F &\rightarrow (E) \mid a \mid b & (5, 6, 7)
 \end{aligned}$$

Bottom-up parsing of $(a)*b$:

$$\begin{aligned}
 &((a)*b, \varepsilon, \varepsilon) \\
 \vdash & (a)*b, (, \varepsilon) \\
 \vdash & ()*b, (a, \varepsilon) \\
 \vdash & ()*b, (F, 6) \\
 \vdash & ()*b, (T, 64) \\
 \vdash & ()*b, (E, 642) \\
 \vdash & (*b, (E), 642) \\
 \vdash & (*b, F, 6425) \\
 \vdash & (*b, T, 64254) \\
 \vdash & (b, T*, 64254) \\
 \vdash & (\varepsilon, T*b, 64254) \\
 \vdash & (\varepsilon, T*F, 642547) \\
 \vdash & (\varepsilon, T, 6425473) \\
 \vdash & (\varepsilon, E, 64254732)
 \end{aligned}$$