Compiler Construction

Lecture 8: Syntax Analysis IV ($LL(1)$ and Bottom-Up Parsing)

Winter Semester 2018/19

Thomas Noll
Software Modeling and Verification Group
RWTH Aachen University

https://moves.rwth-aachen.de/teaching/ws-1819/cc/

Recap: $LL(1)$ Parsing

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.

later □

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

The Deterministic Top-Down Automaton

Definition (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 $\mathrm{act} : \Sigma_\varepsilon \times X_\varepsilon \to \{(\alpha, i) \mid \pi_i = A \to \alpha\} \cup \{\mathrm{pop}, \mathrm{accept}, \mathrm{error}\}$
  with $\mathrm{act}(x, A) := (\alpha, i)$ if $\pi_i = A \to \alpha$ and $x \in \mathrm{la}(\pi_i)$
  $\mathrm{act}(a, a) := \mathrm{pop}$
  $\mathrm{act}(\varepsilon, \varepsilon) := \mathrm{accept}$
  $\mathrm{act}(x, y) := \mathrm{error}$ otherwise
- Transitions for $x \in \Sigma_\varepsilon$, $w \in \Sigma^*$ (with $x \neq \varepsilon$ if $w \neq \varepsilon$), $Y \in X$, $\beta \in X^*$, and $z \in [p]^*$:

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

Transformation to $LL(1)$

Goal: transformation of $G = \langle N, \Sigma, P, S \rangle \in CFG_\Sigma \setminus LL(1)$
(i.e., $\mathrm{la}(A \to \beta) \cap \mathrm{la}(A \to \gamma) \neq \emptyset$ for some $A \to \beta \mid \gamma \in P$) 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 a's 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`)

Left Recursion I

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 $A \Rightarrow_l^+ A\beta$.

Example 8.3

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 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 $\mathrm{DTA}(G)$ (Definition 7.5) starts with $(vw, S, \varepsilon) \vdash^* (w, A\alpha, \ldots) \vdash^+ (w, A\beta\alpha, \ldots)$.
But in the last configuration the behaviour of $\mathrm{DTA}(G)$ is determined by the same input ($\mathrm{fi}(w)$) and stack symbol (A). Thus it enters an infinite 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. ↯ □

Transformation to $LL(1)$

Eliminating 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 new $A' \in N$) which preserves $L(G)$

Example 8.5

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

is 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).

Eliminating Indirect Left Recursion I

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

$$A \to A_1\alpha_1 \qquad A_1 \to A_2\alpha_2 \qquad \ldots \qquad A_{k-1} \to A_k\alpha_k \qquad A_k \to A\beta$$

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

Algorithm 8.6 (Elimination of left recursion)

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

Procedure: **for** $i := 1$ **to** n **do**
  for $j := 1$ **to** $i - 1$ **do** *% remove references to "previous" nonterminals*
    replace each $B_i \to B_j\gamma$ *by* $B_i \to \delta_1\gamma \mid \ldots \mid \delta_k\gamma$,
    where $B_j \to \delta_1 \mid \ldots \mid \delta_k$ *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)*

Eliminating Indirect Left Recursion II

Example 8.7

1. Original grammar ($n = 2, B_1 = S, B_2 = A$): $G: S \to Sa \mid Ab \mid c$, $A \to Ad \mid Se \mid f$
2. $i = 1$: After B_1-substitution (no $j < i$): $S \to Sa \mid Ab \mid c$, $A \to Ad \mid Se \mid f$. After removal of direct left B_1-recursion: $S \to AbS' \mid cS'$, $S' \to aS' \mid \varepsilon$, $A \to Ad \mid Se \mid f$
3. $i = 2$: After B_2-substitution ($j = 1$): $S \to AbS' \mid cS'$, $S' \to aS' \mid \varepsilon$, $A \to Ad \mid AbS'e \mid cS'e \mid f$. After removal of direct left B_2-recursion: $G': S \to AbS' \mid cS'$, $S' \to aS' \mid \varepsilon$, $A \to cS'eA' \mid fA'$, $A' \to dA' \mid bS'eA' \mid \varepsilon$

(Observation: $G' \notin LL(1)$ since $\mathrm{la}(S \to AbS') \cap \mathrm{la}(S \to cS') = \{c\} \neq \emptyset$)

Remark: after applying Algorithm 8.6, never $G' \in LL(1)$ (cf. exercises)

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 8.8

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

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 $\mathcal{O}(|w|)$ (for input word $w \in \Sigma^*$)
- Here: proof for ε-free grammars (i.e., without ε-productions)
- General case: see O. Mayer: *Syntaxanalyse*, p. 211ff

Lemma 8.9

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 $\mathrm{DTA}(G)$*, 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 8.4, 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 8.10 (Arithmetic expressions; cf. Example 8.5)

```
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
```

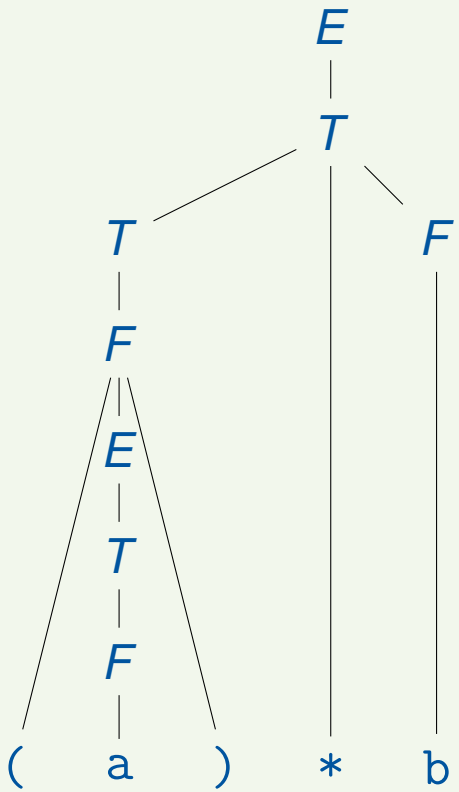
Recap: Top-Down Parsing

Example 8.11

Grammar for arithmetic expressions:

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

Leftmost analysis of `(a)*b`:
2 3 4 5 2 4 6 7



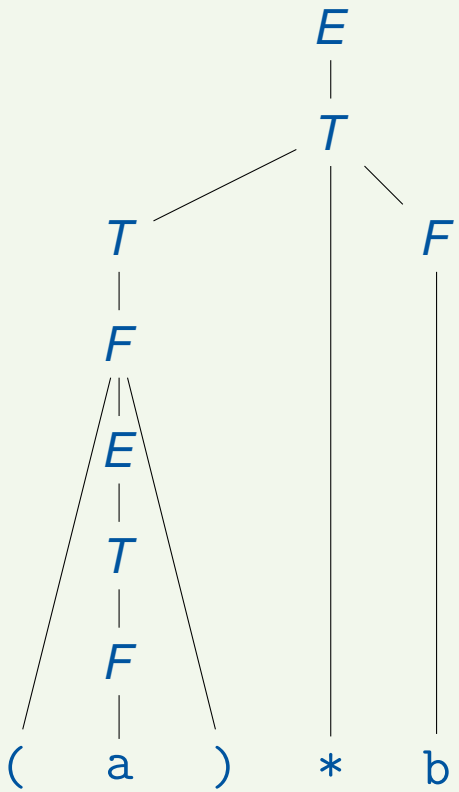
Bottom-Up Parsing I

Example 8.12

Grammar for
arithmetic expressions:

$$G_{AE}: \quad E \to E{+}T \mid T \quad (1,2)$$
$$T \to T{*}F \mid F \quad (3,4)$$
$$F \to (E) \mid \texttt{a} \mid \texttt{b} \quad (5,6,7)$$

Reversed rightmost analysis
of `(a)*b`:
6 4 2 5 4 7 3 2



Bottom-Up Parsing II

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 step)
2. Remove nondeterminism by allowing lookahead on the input:
   $G \in LR(k)$ iff $L(G)$ recognisable by deterministic bottom-up parsing automaton with lookahead of k symbols

The Nondeterministic Bottom-Up Automaton I

Definition 8.13 (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 $\pi_i = A \to \beta$
- Initial configuration for $w \in \Sigma^*$: $(w, \varepsilon, \varepsilon)$
- Final configurations: $\{\varepsilon\} \times \{S\} \times [p]^*$

The Nondeterministic Bottom-Up Automaton II

Example 8.14

Grammar for arithmetic expressions (cf. Example 8.12):

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

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

$$
\begin{array}{ll}
 & (\texttt{(a)*b}, \varepsilon\ , \varepsilon) \\
\vdash & (\ \texttt{a)*b}, (\ , \varepsilon) \\
\vdash & (\ \ \texttt{)*b}, (\texttt{a}, \varepsilon)
\end{array}
$$

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

Correctness of $NBA(G)$

Theorem 8.15 (Correctness of $NBA(G)$)

Let $G = \langle N, \Sigma, P, S \rangle \in CFG_\Sigma$ and $NBA(G)$ as before. Then, for every $w \in \Sigma^*$ and $z \in [p]^*$,

$$(w, \varepsilon, \varepsilon) \vdash^* (\varepsilon, S, z) \quad \textit{iff} \quad \overleftarrow{z} \textit{ is a rightmost analysis of } w$$

Proof.

similar to the top-down case (Theorem 6.1) □