Chapter 4:
Bottom-up Analysis

**Attention:**
Many grammars are not $LL(k)$!

A reason for that is:

**Definition**
Grammar $G$ is called left-recursive, if

$$A \rightarrow^{+} AB$$

for an $A \in N$, $B \in (T \cup N)^*$

**Example:**

$$
\begin{align*}
E & \rightarrow F + T \\
F & \rightarrow T * F \\
F & \rightarrow (E) \\
T & \rightarrow \text{name} \\
\end{align*}
$$

... is left-recursive
Theorem:
Let a grammar $G$ be reduced and left-recursive, then $G$ is not $LL(k)$ for any $k$.

Proof:
Let $A \rightarrow A \beta | \alpha \in P$ and $A$ be reachable from $S$

Assumption: $G$ is $LL(k)$

First™($\alpha \beta^n \gamma$) \cap First™($\alpha \beta^{n+1} \gamma$) = \emptyset

Theorem:
Let a grammar $G$ be reduced and left-recursive, then $G$ is not $LL(k)$ for any $k$.

Proof:
Let $A \rightarrow A \beta | \alpha \in P$ and $A$ be reachable from $S$

Assumption: $G$ is $LL(k)$

First™($\alpha \beta^n \gamma$) \cap First™($\alpha \beta^{n+1} \gamma$) = \emptyset
**Bottom-up Analysis**

**Theorem:**
Let a grammar $G$ be reduced and left-recursive, then $G$ is not $LL(k)$ for any $k$.

**Proof:**
Let $A \to A \beta | \alpha \in P$ and $A$ be reachable from $S$.

**Assumption:** $G$ is $LL(k)$.

$\Rightarrow \text{First}_k(\alpha, \beta^{n+1} \gamma) = \emptyset$.

**Bottom-up Analysis**

**Theorem:**
Let a grammar $G$ be reduced and left-recursive, then $G$ is not $LL(k)$ for any $k$.

**Proof:**
Let $A \to A \beta | \alpha \in P$ and $A$ be reachable from $S$.

**Assumption:** $G$ is $LL(k)$.

$\Rightarrow \text{First}_k(\alpha, \beta^{n} \gamma) \cap \text{First}_k(\alpha, \beta^{n+1} \gamma) = \emptyset$.

**Shift-Reduce Parser**

**Idea:**
We *delay* the decision whether to reduce until we know, whether the input matches the right-hand-side of a rule!

**Construction:**
Shift-Reduce parser $M_G^R$.
- The input is shifted successively to the pushdown.
- Is there a complete right-hand side (a handle) atop the pushdown, it is replaced (reduced) by the corresponding left-hand side.

Donald Knuth
Shift-Reduce Parser

Example:

\[ S \rightarrow AB \]
\[ A \rightarrow a \]
\[ B \rightarrow b \]

The pushdown automaton:

<table>
<thead>
<tr>
<th>States:</th>
<th>( q_0, f, a, b, A, B, S );</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start state:</td>
<td>( q_0 )</td>
</tr>
<tr>
<td>End state:</td>
<td>( f )</td>
</tr>
</tbody>
</table>

\[
\begin{array}{c|c|c}
  q_0 & a & q_0 a \\
  a & e & A \\
  b & e & B \\
  AB & e & S \\
  q_0 S & e & f \\
\end{array}
\]

Shift-Reduce Parser

Construction:
In general, we create an automaton \( M_R = (Q, T, \delta, q_0, F) \) with:

- \( Q = T \cup N \cup \{q_0, f\} \) \( (q_0, f) \) fresh;
- \( F = \{f\} \);
- Transitions:

\[
\delta = \{(q, x, q x) | q \in Q, x \in T \} \cup \quad \text{// Shift-transitions}
\{(q, \epsilon, q A) | q \in Q, A \rightarrow \alpha \in P \} \cup \quad \text{// Reduce-transitions}
\{(q_0 S, \epsilon, f)\} \quad \text{// finish}
\]

Example-computation:

\( (q_0, a b) \vdash (q_0 a, b) \vdash (q_0 A, b) \)
\( \vdash (q_0 A b, \epsilon) \vdash (q_0 A B, \epsilon) \)
\( \vdash (q_0 S, \epsilon) \vdash (f, \epsilon) \)

Shift-Reduce Parser

Observation:

- The sequence of reductions corresponds to a reverse rightmost-derivation for the input
- To prove correctness, we have to prove:

\[
(\epsilon, w) \vdash (A, \epsilon) \iff A \rightarrow^* w
\]

- The shift-reduce pushdown automaton \( M_R \) is in general also non-deterministic
- For a deterministic parsing algorithm, we have to identify computation-states for reduction

\[ \Rightarrow \text{LR-Parsing} \]
Reverse Rightmost Derivations in Shift-Reduce-Parsers

**Idea:** Observe reverse rightmost-derivations of $M_G^R$!

**Input:**

- counter $\ast 2 + 40$

**Pushdown:**

- $(q_0)$

- $T 1 \ast F 2 \text{ int}$

- $F 1 \text{ name}$

---

Reverse Rightmost Derivations in Shift-Reduce-Parsers

**Idea:** Observe reverse rightmost-derivations of $M_G^R$!

**Input:** $\ast 2 + 40$

**Pushdown:**

- $(q_0 \text{name})$

- $T 1 \ast F 2 \text{ int}$

- $F 1 \text{name}$

---

Reverse Rightmost Derivations in Shift-Reduce-Parsers

**Idea:** Observe reverse rightmost-derivations of $M_G^R$!

**Input:** $\ast 2 + 40$

**Pushdown:**

- $(q_0 F)$

- $T 1 \ast F 2 \text{ int}$

- $F 1 \text{name}$

---

Reverse Rightmost Derivations in Shift-Reduce-Parsers

**Idea:** Observe reverse rightmost-derivations of $M_G^R$!

**Input:** $\ast 2 + 40$

**Pushdown:**

- $(q_0 T)$

- $T 1 \ast F 2 \text{ int}$

- $F 1 \text{name}$
Reverse Rightmost Derivations in Shift-Reduce-Parsers

**Idea:** Observe reverse rightmost-derivations of $M_G^R$!

**Input:**

```
+ 40
```

**Pushdown:**

```
(q_0 T + int)
```

---

Reverse Rightmost Derivations in Shift-Reduce-Parsers

**Idea:** Observe reverse rightmost-derivations of $M_G^R$!

**Input:**

```
+ 40
```

**Pushdown:**

```
(q_0 T + F)
```

---

Reverse Rightmost Derivations in Shift-Reduce-Parsers

**Idea:** Observe reverse rightmost-derivations of $M_G^R$!

**Input:**

```
( f )
```

---

Reverse Rightmost Derivations in Shift-Reduce-Parsers

**Idea:** Observe reverse rightmost-derivations of $M_G^R$!

**Input:**

```
+ 40
```

**Pushdown:**

```
(q_0 T + F)
```

---

**Generic Observation:**

In a sequence of configurations of $M_G^R$

```
(q_0 \alpha \gamma, \nu) \vdash (q_0 \alpha B, \nu) \vdash^* (q_0 S, \epsilon)
```

we call $\alpha \gamma$ a viable prefix for the complete item $[B \rightarrow \gamma \epsilon]$.  

---
Reverse Rightmost Derivations in Shift-Reduce-Parsers

Idea: Observe reverse rightmost-derivations of $M_G^R$.

Input:

Pushdown:

($\alpha, E, F$) \[ E \rightarrow 0 \]

name

F 1

T 1

F 2

T 1

1 1

4 1

2 1

Generic Observation:

In a sequence of configurations of $M_G^R$

$(q_0, \alpha, \gamma, v) \rightarrow (q_0, \alpha, B, v) \rightarrow^*(q_0, S, \epsilon)$

we call $\alpha \gamma$ a viable prefix for the complete item $[B \rightarrow \gamma]$.

Bottom-up Analysis: Viable Prefix

$\alpha \gamma$ is viable for $[B \rightarrow \gamma]$ iff $S \rightarrow_i^* \alpha B v$

... with $\alpha = \alpha_1 \ldots \alpha_m$

Conversely, for an arbitrary valid word $\alpha'$ we can determine the set of all later on possibly matching rules ...
Characteristic Automaton

Observation:
The set of viable prefixes from \((N \cup T)^*\) for (admissible) items can be computed from the content of the shift-reduce parser’s pushdown with the help of a finite automaton:

States: Items
Start state: \([S \rightarrow \bullet S]\)
Final states: \(\{[B \rightarrow \gamma \bullet] \mid B \rightarrow \gamma \in P\}\)
Transitions:
1. \(\left( [A \rightarrow \alpha \bullet X \beta], X, [A \rightarrow \alpha X \bullet \beta] \right), \quad X \in (N \cup T), A \rightarrow \alpha X \beta \in P; \)
2. \(\left( [A \rightarrow \alpha \bullet B \beta], e, [B \rightarrow \bullet \gamma] \right), \quad A \rightarrow \alpha B \beta, \quad B \rightarrow \gamma \in P;\)

The automaton \(e(G)\) is called characteristic automaton for \(G\).

Reverse Rightmost Derivations in Shift-Reduce-Parsers

Idea: Observe reverse rightmost-derivations of \(M_G^R\)!

Input:

\[ + 40 \]

Pushdown:
\( (q_0, E \bullet L_r) \)

Generic Observation:
In a sequence of configurations of \(M_G^R\)
\[(q_0, \alpha \gamma, v) \vdash (q_0, \alpha B, v) \vdash^* (q_0, S, e)\]
we call \(\alpha \gamma\) a viable prefix for the complete item \([B \rightarrow \gamma \bullet]\).

Characteristics Automaton

For example:
\[ E \rightarrow E + T \quad | \quad T \]
\[ T \rightarrow T \ast F \quad | \quad F \]
\[ F \rightarrow (E) \quad | \quad \text{int} \]

Diagram:

\[ S \rightarrow \bullet E \quad \rightarrow \quad E \rightarrow E + T \quad \rightarrow \quad T \rightarrow T \ast F \quad \rightarrow \quad F \rightarrow (E) \quad \rightarrow \quad \text{int} \]
Reverse Rightmost Derivations in Shift-Reduce-Parsers

Idea: Observe reverse rightmost-derivations of $M_C^R$.

Input:

Pushdown:

( $q_0 E + P_1$ )

Generic Observation:
In a sequence of configurations of $M_C^R$

$$(q_0 \alpha \gamma, \nu) \vdash (q_0 \alpha B, \nu) \vdash^* (q_0 S, \epsilon)$$

we call $\alpha \gamma$ a viable prefix for the complete item $[B \rightarrow \gamma \bullet]$.

Characteristic Automaton

For example:

$E \rightarrow E + T \mid T$
$T \rightarrow T * F \mid F$
$F \rightarrow (E) \mid \text{int}$

Canonical LR(0)-Automaton

The canonical $LR(0)$-automaton $LR(G)$ is created from $\epsilon(G)$ by:

1. performing arbitrarily many $\epsilon$-transitions after every consuming transition
2. performing the powerset construction

... for example:
Canonical LR(0)-Automaton

Example:

\[
\begin{align*}
E & \to E + T \quad | \quad T \\
T & \to T * F \\
F & \to \text{int} \\
\end{align*}
\]

Therefore we determine:

\[
\begin{align*}
q_0 & = \delta(q_0, E) = \{ E \to E + T \} \\
q_1 & = \delta(q_0, T) = \{ T \to T * F \} \\
q_2 & = \delta(q_0, \text{int}) = \{ F \to \text{int} \}
\end{align*}
\]

Canonical LR(0)-Automaton

Example:

\[
\begin{align*}
E & \to E + T \quad | \quad T \\
T & \to T * F \\
F & \to (E) \\
\end{align*}
\]

Therefore we determine:

\[
\begin{align*}
q_0 & = \{ [S' \to E], \quad \} \\
q_1 & = \delta(q_0, E) = \{ [E \to E + T], \quad \} \\
q_2 & = \delta(q_0, T) = \{ [T \to T * F], \quad \} \\
q_3 & = \delta(q_0, F) = \{ [F \to \text{int}], \quad \}
\end{align*}
\]

Canonical LR(0)-Automaton

Example:

\[
\begin{align*}
E & \to E + T \quad | \quad T \\
T & \to T * F \\
F & \to \text{int} \\
\end{align*}
\]

Therefore we determine:

\[
\begin{align*}
q_0 & = \delta(q_0, E) = \{ [E \to E \cdot E], \quad \} \\
q_1 & = \delta(q_0, T) = \{ [T \to T \cdot E], \quad \} \\
q_2 & = \delta(q_0, \text{int}) = \{ [F \to \text{int}], \quad \}
\end{align*}
\]

Canonical LR(0)-Automaton

Example:

\[
\begin{align*}
E & \to E + T \quad | \quad T \\
T & \to T * F \\
F & \to \text{int} \\
\end{align*}
\]

Therefore we determine:

\[
\begin{align*}
q_0 & = \delta(q_0, E) = \{ [E \to E + T], \quad \} \\
q_1 & = \delta(q_0, T) = \{ [T \to T * F], \quad \} \\
q_2 & = \delta(q_0, \text{int}) = \{ [F \to \text{int}], \quad \}
\end{align*}
\]
Canonical LR(0)-Automaton

Example:

\[
E \rightarrow E + T \quad | \quad T \\
T \rightarrow T \cdot F \quad | \quad F \\
F \rightarrow (E) \quad | \quad \text{int}
\]

Therefore we determine:

\[
q_0 = \{ [S' \rightarrow \bullet E] \} \quad q_1 = \delta(q_0, E) = \{ [S' \rightarrow E \bullet], [E \rightarrow E \bullet + T] \} \\
q_2 = \delta(q_0, E) = \{ [E \rightarrow E \bullet], [T \rightarrow T \bullet + F] \} \quad q_3 = \delta(q_0, F) = \{ [T \rightarrow F \bullet] \} \\
q_4 = \delta(q_0, \text{int}) = \{ [F \rightarrow \text{int} \bullet] \}
\]

Canonical LR(0)-Automaton

Observation:

The canonical LR(0)-automaton can be created directly from the grammar.

Therefore we need a helper function \( \delta^*_{\epsilon} \) (\( \epsilon \)-closure)

\[
\delta^*_{\epsilon}(q) = q \cup \{ [B \rightarrow \bullet \gamma] \mid \exists [A \rightarrow \alpha \bullet B' \beta'] \in q, \beta' \in (N \cup T)^* \cdot B' \rightarrow \beta \}
\]

We define:

\[
\begin{align*}
\text{States:} & \quad \text{Sets of items;} \\
\text{Start state:} & \quad \delta^*_{\epsilon} \{ [S' \rightarrow \bullet S] \} \\
\text{Final states:} & \quad \{ q \mid \exists A \rightarrow \alpha \in P : [A \rightarrow \alpha \bullet] \in q \} \\
\text{Transitions:} & \quad \delta(q, X) = \delta^*_{\epsilon} \{ [A \rightarrow \alpha X \bullet \beta] \mid [A \rightarrow \alpha \bullet X \beta] \in q \}
\end{align*}
\]

LR(0)-Parser

Idea for a parser:

- The parser manages a viable prefix \( \alpha = X_1 \ldots X_m \) on the pushdown and uses LR(G), to identify reduction spots.
- It can reduce with \( A \rightarrow \gamma \), if \( [A \rightarrow \gamma \bullet] \) is admissible for \( \alpha \)

Optimization:

We push the states instead of the \( X_i \) in order not to process the pushdown’s content with the automaton anew all the time. Reduction with \( A \rightarrow \gamma \) leads to popping the uppermost \( |\gamma| \) states and continue with the state on top of the stack and input \( A \).

Attention:

This parser is only deterministic, if each final state of the canonical LR(0)-automaton is conflict free.
Characteristic Automaton

For example:

\[ E \rightarrow E + T \quad | \quad T \]
\[ T \rightarrow T * F \quad | \quad F \]
\[ F \rightarrow (E) \quad | \quad \text{int} \]

Canonical LR(0)-Automaton

The canonical LR(0)-automaton LR(G) is created from \( \epsilon(G) \) by:
- performing arbitrarily many \( \epsilon \)-transitions after every consuming transition
- performing the powerset construction

... for example:

LR(0)-Parser

Idea for a parser:
- The parser manages a viable prefix \( \alpha = X_1 \ldots X_n \) on the pushdown and uses LR(G), to identify reduction spots.
- It can reduce with \( A \rightarrow \gamma \), if \( [A \rightarrow \gamma] \) is admissible for \( \alpha \)

Optimization:

We push the states instead of the \( X_i \) in order not to process the pushdown's content with the automaton anew all the time.
Reduction with \( A \rightarrow \gamma \) leads to popping the uppermost \( |\gamma| \) states and continue with the state on top of the stack and input \( A \).

Attention:

This parser is only deterministic, if each final state of the canonical LR(0)-automaton is conflict free.

LR(0)-Parser

... for example:

\[ q_1 = \{ [S' \rightarrow E \bullet], [E \rightarrow E \bullet + T] \} \]
\[ q_2 = \{ [E \rightarrow T \bullet], [T \rightarrow T \bullet + F] \} \]
\[ q_3 = \{ [T \rightarrow F \bullet] \} \]
\[ q_4 = \{ [F \rightarrow \text{int} \bullet] \} \]
\[ q_9 = \{ [E \rightarrow E + T \bullet], [T \rightarrow T \bullet + F] \} \]
\[ q_{10} = \{ [T \rightarrow T \bullet + F] \} \]
\[ q_{11} = \{ [F \rightarrow (E) \bullet] \} \]

The final states \( q_1, q_2, q_9 \) contain more then one admissible item

\( \Rightarrow \) non deterministic!
LR(0)-Parser

The construction of the LR(0)-parser:

States: \( Q \cup \{ f \} \)  \( (\text{f fresh}) \)
Start state: \( q_0 \)
Final state: \( f \)

Transitions:

- **Shift:** \( (p, a, pq) \) if \( q = \delta(p, a) \neq \emptyset \)
- **Reduce:** \( (pq_1 \ldots q_m, \epsilon, pq) \) if \( \left[ A \rightarrow X_1 \ldots X_m \bullet \right] \in q_m, \)
  \( q = \delta(p, A) \)
- **Finish:** \( (q_0 p, \epsilon, f) \) if \( \left[ S' \rightarrow S \bullet \right] \in p \)

with \( LR(G) = (Q, T, \delta, q_0, F) \).

Correctness:

we show:

The accepting computations of an LR(0)-parser are one-to-one related to those of a shift-reduce parser \( M^R \).

we conclude:

- The accepted language is exactly \( L(G) \)
- The sequence of reductions of an accepting computation for a word \( w \in T \) yields a reverse rightmost derivation of \( G \) for \( w \)

Attention:

Unluckily, the LR(0)-parser is in general non-deterministic.

We identify two reasons:

- **Reduce-Reduce-Conflict:** \( \left[ A \rightarrow \gamma \bullet \right], \left[ A' \rightarrow \gamma' \bullet \right] \in q \) with \( A \neq A' \lor \gamma \neq \gamma' \)

- **Shift-Reduce-Conflict:** \( \left[ A \rightarrow \gamma \bullet \right], \left[ A' \rightarrow \alpha \bullet \beta \right] \in q \) with \( \alpha \in T \)
  \( \text{for a state } q \in Q \).

Those states are called LR(0)-unsuited.