Syntactic Analysis

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

$$A \rightarrow^+ A \beta$$

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

Example:

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

... 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 $LL(k)$

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

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

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

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

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

**Case 1:** $\beta \Rightarrow^* \epsilon$ — Contradiction !!!
**Case 2:** $\beta \Rightarrow^* w \neq \epsilon \Rightarrow \text{First}_k(\alpha w^k \gamma) \cap \text{First}_k(\alpha w^{k+1} \gamma) \neq \emptyset$

Shift-Reduce Parser

**Example:**

$$
S \rightarrow AB \\
A \rightarrow a \\
B \rightarrow b
$$

The pushdown automaton:

**States:** $q_0, f, a, b, A, B, S$;
**Start state:** $q_0$
**End state:** $f$

Shift-Reduce Parser

**Construction:**
In general, we create an automaton $M^R_G = (Q, T, \delta, q_0, F)$ with:
- $Q = T \cup N \cup \{q_0, f\}$ (fresh);
- $F = \{f\}$
- Transitions:

$$
\delta = \\
\{(a, e, x) | a \in Q, x \in T\} \quad // \quad \text{Shift-transitions} \\
\{(q, a, y, z) | q \in Q, A \rightarrow \alpha \in P\} \quad // \quad \text{Reduce-transitions} \\
\{(q_0, \epsilon, f)\} \quad // \quad \text{finish}
$$
Shift-Reduce Parser

Construction:
In general, we create an automaton $M^R = (Q, T, \delta, q_0, F)$ with:

- $Q = T \cup N \cup \{q_0, f\}$
- $F = \{f\}$
- Transitions:
  $\delta = \{(q, x, q x) \mid q \in Q, x \in T\} \cup \{(q, \alpha, q a) \mid q \in Q, A \rightarrow \alpha \in P\} \cup \{(q_0, S, f)\}$

Example-computation:

$$\begin{align*}
(q_0, a b) & \vdash (q_0, a, b) \vdash (q_0, A, b) \\
& \vdash (q_0, A b, \epsilon) \vdash (q_0, A B, \epsilon) \\
& \vdash (q_0, S, \epsilon) \vdash (f, \epsilon)
\end{align*}$$

Shift-Reduce Parser

Observation:

- The sequence of reductions corresponds to a reverse rightmost-derivation for the input
- To prove correctness, we have to prove:
  $$(\epsilon, w) \vdash^* (A, \epsilon) \text{ iff } A \rightarrow^* w$$
- The shift-reduce pushdown automaton $M^R_G$ is in general also non-deterministic
- For a deterministic parsing algorithm, we have to identify computation-states for reduction

Reverse Rightmost Derivations in Shift-Reduce-Parsers

Idea: Observe reverse rightmost-derivations of $M^R_G$!

Input: $counter + 2 + 40$

Pushdown:

$\begin{align*}
\{q_0\}
\end{align*}$

Reverse Rightmost Derivations in Shift-Reduce-Parsers

Idea: Observe reverse rightmost-derivations of $M^R_G$!

Input: $counter + 2 + 40$

Pushdown:

$\begin{align*}
F1, int
\end{align*}$
Reverse Rightmost Derivations in Shift-Reduce-Parsers

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

Input: $+ 40$

Pushdown: $(q_0, T = R)$

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 $[B \rightarrow \gamma•]$.

Reverse Rightmost Derivations in Shift-Reduce-Parsers

Input: $+ 40$

Pushdown: $(q_0, T = R)$

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 $[B \rightarrow \gamma•]$.

Characterisitic 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: $[S' \rightarrow • S]$
Final states: $\{[B \rightarrow \gamma•] \mid B \rightarrow \gamma \in P\}$

Transitions:
1. $([A \rightarrow \alpha \bullet \beta], X, [A \rightarrow \alpha X \bullet \beta]), \quad X \in (N \cup T), A \rightarrow \alpha X \beta \in P$;
2. $([A \rightarrow \alpha \bullet B \beta], e, [B \rightarrow • \gamma]), \quad A \rightarrow \alpha B \beta, \quad B \rightarrow \gamma \in P$;

The automaton $c(G)$ 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: $(q_0, T = R)$

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 $[B \rightarrow \gamma•]$.
Characteristic Automaton

For example:

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

Reverse Rightmost Derivations in Shift-Reduce-Parsers

Idea: Observe reverse rightmost derivations of \( M_G^{RL} \)

Input:

Pushdown:

\[
(q_0, E, L^R, L^E)
\]

Generic Observation:

In a sequence of configurations of \( M_G^{E} \)

\[
(q_0, \alpha \gamma, v) \vdash (q_0, \alpha B, v) \vdash^* (q_0, S, \epsilon)
\]

we call \( \alpha \gamma \) a viable prefix for the complete item \( [B \to \gamma] \).

Bottom-up Analysis: Admissible Items

The item \( [B \to \gamma \bullet \beta] \) is called admissible for \( \alpha' \) iff \( S \to^* \alpha B \) with \( \alpha' = \alpha \gamma \).
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 stack with the help of a finite automaton:

- **States:** Items
- **Start state:** \([S' \rightarrow \varepsilon S]\)
- **Final states:** \([B \rightarrow \gamma \varepsilon] \mid B \rightarrow \gamma \in P\)

Transitions:

1. \([A \rightarrow \alpha \cdot X \beta, X, A \rightarrow \alpha X \cdot \beta]] \quad X \in (N \cup T), A \rightarrow \alpha X \beta \in P;\]
2. \([A \rightarrow \alpha[B\beta], \varepsilon, B \rightarrow \varepsilon \gamma]] \quad A \rightarrow \alpha \beta \gamma, B \rightarrow \gamma \in P;\]

The automaton \(c(G)\) is called **characteristic automaton** for \(G\).

Canonical LR(0)-Automaton

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

- Performing arbitrarily many \(\epsilon\)-transitions after every consuming transition
- Performing the powerset construction

... for example:

[Diagram of LR(0)-Automaton]

Canonical LR(0)-Automaton

Example:

\[
\begin{align*}
E & \rightarrow E + T \mid T \\
T & \rightarrow T * F \mid F \\
F & \rightarrow (E) \mid \text{int}
\end{align*}
\]

Therefore we determine:

\[
\begin{align*}
[S & \rightarrow E] \cdot \\
[E & \rightarrow E + T] \cdot \\
[T & \rightarrow T * F] \cdot \\
[F & \rightarrow \text{int}]
\end{align*}
\]
Canonical LR(0)-Automaton

Example:

\[
\begin{align*}
E & \rightarrow E + T \\
T & \rightarrow T \ast F \\
F & \rightarrow E \\
\end{align*}
\]

Therefore we determine:

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

Canonical LR(0)-Automaton

\[q_5 = \delta(q_0, \text{int}) = \{F \rightarrow (\bullet E)\}, \quad q_7 = \delta(q_2, \ast) = \{E \rightarrow \bullet E + T\}, \quad q_9 = \delta(q_0, T) = \{T \rightarrow T \ast T\}, \quad q_{10} = \delta(q_7, \text{int}) = \{F \rightarrow \bullet \text{int}\}\]

Canonical LR(0)-Automaton

\[q_8 = \delta(q_0, E) = \{F \rightarrow (\bullet E)\}, \quad q_{10} = \delta(q_7, F) = \{T \rightarrow T \ast T\}, \quad q_{11} = \delta(q_8, \text{int}) = \{F \rightarrow \bullet \text{int}\}\]

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)

\[\delta_{\epsilon}^*(q) = q \cup \{B \rightarrow \bullet \gamma \mid \exists A \rightarrow \alpha \ast B' \beta', q \in \mathcal{Q}, \beta \in (N \cup T)^* : B' \rightarrow^* B\}\]

We define:

States: Sets of items;
Start state: \(\delta_{\epsilon}^* \{S' \rightarrow \bullet S\}\);
Final states: \(\{q \mid \exists A \rightarrow \alpha \in P : [A \rightarrow \alpha \ast \bullet \beta] \in q\}\);
Transitions: \(\delta(q, X) = \delta_{\epsilon}^* \{[A \rightarrow \alpha X \ast \beta] \mid [A \rightarrow \alpha \bullet X \beta] \in q\}\)
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.

Canonical $LR(0)$-Automaton

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

Reverse Rightmost Derivations in Shift-Reduce-Parsers

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

Input: $+ 40$

Pushdown: $(q_0 \gamma F \epsilon F)$

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

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

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

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

We define:
- **States**: Sets of items;
- **Start state**: $\delta_\epsilon^*(\{[S' \rightarrow \bullet S]\})$
- **Final states**: $\{q \mid \exists A \rightarrow \alpha \in P : [A \rightarrow \alpha \bullet] \in q\}$
- **Transitions**: $\delta(q, X) = \delta_\epsilon^*(\{[A \rightarrow \alpha X \bullet \beta] \mid [A \rightarrow \alpha \bullet X \beta] \in q\}$

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.

LR(0)-Parser

... for example:

$$q_1 = \{[S' \rightarrow E \bullet], \quad [E \rightarrow E \bullet + T]\}$$
$$q_2 = \{[E \rightarrow T \bullet], \quad [T \rightarrow T \bullet * F]\}$$
$$q_3 = \{[T \rightarrow F \bullet]\}$$
$$q_4 = \{[F \rightarrow \text{int} \bullet]\}$$

$$q_9 = \{[E \rightarrow E + T \bullet], \quad [T \rightarrow T \bullet * F]\}$$
$$q_{10} = \{[T \rightarrow T \bullet * F]\}$$
$$q_{11} = \{[F \rightarrow \text{int} \bullet]\}$$

The final states $q_1, q_2, q_9$ contain more then one admissible item
$\Rightarrow$ non deterministic!

LR(0)-Parser

Attention:
Unfortunately, the LR(0) parser is in general non-deterministic.

We identify two reasons:

Reduce-Reduce-Conflict:

$$[A \rightarrow \gamma \bullet], \quad [A' \rightarrow \gamma' \bullet] \in q \quad \text{with} \quad A \neq A' \lor \gamma \neq \gamma'$$

Shift-Reduce-Conflict:

$$[A \rightarrow \gamma \bullet], \quad [A' \rightarrow \alpha \bullet a \beta] \in q \quad \text{with} \quad a \in T$$

for a state $q \in Q$.

Those states are called LR(0)-unsuited.
Revisiting the Conflicts of the LR(0)-Automaton

What differentiates the particular Reductions and Shifts?

Input: $2 + 40$

Pushdown: $(q_0, T)$

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

Revisiting the Conflicts of the LR(0)-Automaton

Idea: Matching lookahead with right context matters!

Input: $40$

Pushdown: $(q_0, E)$

$$E \rightarrow E + T \mid T$$
$$T \rightarrow T * F \mid F$$
$$F \rightarrow (E) \mid \text{int}$$
LR(k)-Grammars

**Idea:** Consider $k$-lookahead in conflict situations.

**Definition:**

The reduced contextfree grammar $G$ is called $LR(k)$-grammar, if for $\text{First}_k(w) = \text{First}_k(x)$ with:

$$S \rightarrow^* R \quad \alpha Aw \rightarrow \alpha \beta x$$

$$S \rightarrow^* R \quad \alpha'A'w' \rightarrow \alpha'x \quad \{\text{follows: } \alpha = \alpha' \land A = A' \land w' = x\}$$