Shift-Reduce Parser

Observation:
- The sequence of reductions corresponds to a reverse rightmost-derivation for the input
- To prove correctness, we have to prove:
  \[(e, w) \Rightarrow^* (A, e)\]
  \[\text{iff}\]
  \[A \Rightarrow^* w\]
- The shift-reduce pushdown automaton \(M^R_D\) 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 \(M^R_D\)

Input:
- counter * 2 + 10

Pushdown:
- \(q_0\)

Bottom-up Analysis: Viable Prefix

\(\alpha \gamma\) is viable for \([B \rightarrow \gamma \bullet]\)

\(S \Rightarrow^* \alpha B v\)

... with \(\alpha = \alpha_1 \ldots \alpha_m\)
Bottom-up Analysis: Admissible Items

The item \([B \rightarrow \gamma \bullet \beta]\) is called admissible for \(\alpha'\) iff

\[S \rightarrow \gamma \alpha B\]

with \(\alpha' = \alpha \gamma\).

... with \(\alpha = \alpha_1 \ldots \alpha_m\)

---

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: \([S \rightarrow \bullet \epsilon]\)
Final states: \(\{(B \rightarrow \gamma \bullet) \mid B \rightarrow \gamma \in P\}\)

Transitions:
1. \((\{A \rightarrow \alpha \bullet X \beta\}, X, \{A \rightarrow \alpha X \bullet \beta\}), X \in (N \cup T), A \rightarrow \alpha X \beta \in P;\)
2. \((\{A \rightarrow \alpha \bullet B \beta\}, B, \{B \rightarrow \bullet \gamma\}), A \rightarrow \alpha B \beta, B \rightarrow \gamma \in P;\)

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

---

Reverse Rightmost Derivations in Shift-Reduce-Parsers

Idea: Observe reverse rightmost derivations of \(M^{R\|}\)

Input:

\[+40\]

Pushdown:

\(q(T + P)\)

---

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: \([S \rightarrow \bullet \epsilon]\)
Final states: \(\{(B \rightarrow \gamma \bullet) \mid B \rightarrow \gamma \in P\}\)

Transitions:
1. \((\{A \rightarrow \alpha \bullet X \beta\}, X, \{A \rightarrow \alpha X \bullet \beta\}), X \in (N \cup T), A \rightarrow \alpha X \beta \in P;\)
2. \((\{A \rightarrow \alpha \bullet B \beta\}, B, \{B \rightarrow \bullet \gamma\}), A \rightarrow \alpha B \beta, B \rightarrow \gamma \in P;\)

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

For example:

\[
E \rightarrow E + T \mid T \\
T \rightarrow T + F \mid F \\
F \rightarrow (E) \mid \text{int}
\]

Observation:

The set of viable prefixes from \((\mathcal{N} \cup \mathcal{T})^*\) 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: \(\{[B \rightarrow \gamma \bullet] \mid B \rightarrow \gamma \in \mathcal{F}\}\)

Transitions:

1. \([(A \rightarrow \alpha \bullet X \beta), X, ([A \rightarrow \alpha X \bullet \beta]), X \in (\mathcal{N} \cup \mathcal{T}), A \rightarrow \alpha X \beta \in \mathcal{P};\]
2. \([(A \rightarrow \alpha \bullet B \beta \bullet \epsilon, [B \rightarrow \bullet \gamma]), A \rightarrow \alpha B \beta \bullet, B \rightarrow \gamma \in \mathcal{F};\]

The automaton \(\epsilon(G)\) is called characteristic automaton for \(G\).
Canonical LR(0)-Automaton

The canonical LR(0)-automaton $LR(G)$ is created from $G$ by:
1. performing arbitrarily many ε-transitions after every consuming transition
2. performing the powerset construction

... for example:

Canonical LR(0)-Automaton

Example:

$$\delta(q_0, \epsilon) = \{(E \to \epsilon), (E \to E + T), (E \to \epsilon + T), (E \to \epsilon + F), (E \to \epsilon + \text{int})\}$$

Therefore we determine:

$$\delta(q_0, E) = q_1$$

Canonical LR(0)-Automaton

$$\delta(q_0, \epsilon) = \{(E \to \epsilon), (E \to E + T), (E \to \epsilon + T), (E \to \epsilon + F), (E \to \epsilon + \text{int})\}$$

$$\delta(q_1, E) = \{(E \to E + T), (E \to \epsilon + T), (E \to \epsilon + F), (E \to \epsilon + \text{int})\}$$

$$\delta(q_0, \epsilon + T) = \{(T \to T + F), (T \to \epsilon + T), (T \to \epsilon + F), (T \to \epsilon + \text{int})\}$$

$$\delta(q_1, \epsilon + T) = \{(T \to T + F), (T \to \epsilon + T), (T \to \epsilon + F), (T \to \epsilon + \text{int})\}$$

$$\delta(q_0, \epsilon + F) = \{(F \to T + F), (F \to \epsilon + F), (F \to \epsilon + \text{int})\}$$

$$\delta(q_1, \epsilon + F) = \{(F \to T + F), (F \to \epsilon + F), (F \to \epsilon + \text{int})\}$$

$$\delta(q_0, \epsilon + \text{int}) = \{(\text{int} \to \text{int}), (\text{int} \to \epsilon + \text{int})\}$$

$$\delta(q_1, \epsilon + \text{int}) = \{(\text{int} \to \text{int}), (\text{int} \to \epsilon + \text{int})\}$$
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:

![Diagram of an LR(0)-Automaton]

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)

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

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

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 \( [A \rightarrow \gamma \bullet] \) 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

The construction of the LR(0)-parser:

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

Transitions:
- **Shift**: \((p, a, p', q)\) if \( q = \delta(p, a) \neq \emptyset \)
- **Reduce**: \((p q_0 \ldots q_n, \alpha, p, q)\) if \( [A \rightarrow X_1 \ldots X_m \bullet] \in Q_n, q = \delta(p, \alpha) \)
- **Finish**: \((p, \text{f}, \emptyset)\) if \( [S \rightarrow S^*] \in \emptyset \)

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

LR(0)-Parser

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

We identify two reasons:

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

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

Those states are called LR(0)-unsuited.

LR(0)-Parser

Correctness:

we show:

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

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 \( w \).

Revisiting the Conflicts of the LR(0)-Automaton

What differentiates the particular Reductions and Shifts?

Input:
* 2 + 40

Pushdown:
\( ( q_0 T ) \)

E 0 T 1
E 1 T 0 F 2
T 1 T 1 T 2
F 2
int

E \rightarrow E + F | T
T \rightarrow T + T | F
F \rightarrow ( E ) | \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 $\text{First}_k(w) = \text{First}_k(z)$ with:

$$
\begin{align*}
S & \rightarrow \alpha A w \quad \rightarrow \quad \alpha \beta w \\
S & \rightarrow \beta A' w' \quad \rightarrow \quad \alpha \beta x
\end{align*}
$$

follows: $\alpha = \alpha' \land A = A' \land w' = x$.

**for example:**

1. $S \rightarrow A \mid B \quad A \rightarrow aAb \mid 0 \quad B \rightarrow aBbb \mid 1$

2. $a b^n c A b^n b^m c$

**LR(k)-Grammars**

**for example:**

3. $S \rightarrow aAc \quad A \rightarrow b b A \mid b \quad \ldots$ is not $LR(0)$, but $LR(1)$:

   - Let $S \rightarrow _k \alpha X w \rightarrow \alpha \beta w$ with $\{y\} = \text{First}_k(w)$ then $\alpha \beta y$ is one of these forms:

   - $a b^n b c \ , \ a b^n b A c \ , \ a A c$

4. $S \rightarrow a Ac \quad A \rightarrow b Ab \mid b \quad \ldots$ is not $LR(k)$ for any $k \geq 0$:

   - Consider the rightmost derivations:

   $$
   S \rightarrow _k a b^n A b^n c \rightarrow a b^n b b^n c \quad 1 \leq n
   $$