Reverse Rightmost Derivations in Shift-Reduce-Parsers

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

**Input:**

```
* 2 + 40
```

**Pushdown:**

```
( q₀ name )
```

---

Reverse Rightmost Derivations in Shift-Reduce-Parsers

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

**Input:**

```
* 2 + 40
```

**Pushdown:**

```
( q₀ f )
```

---

Reverse Rightmost Derivations in Shift-Reduce-Parsers

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

**Input:**

```
* 2 + 40
```

**Pushdown:**

```
( q₀ name )
```

---

Reverse Rightmost Derivations in Shift-Reduce-Parsers

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

**Input:**

```
* 2 + 40
```

**Pushdown:**

```
( q₀ name )
```
Reverse Rightmost Derivations in Shift-Reduce-Parsers

**Idea:** Observe reverse rightmost-derivations of $M_C$!

**Input:**
\[ \ast \ 2 + 40 \]

**Pushdown:**
( $q_0 \ F$ )

---

Reverse Rightmost Derivations in Shift-Reduce-Parsers

**Idea:** Observe reverse rightmost-derivations of $M_C$!

**Input:**
\[ 2 + 40 \]

**Pushdown:**
( $q_0 \ F$ )

---

Reverse Rightmost Derivations in Shift-Reduce-Parsers

**Idea:** Observe reverse rightmost-derivations of $M_C$!

**Input:**
\[ + \ 40 \]

**Pushdown:**
( $q_0 \ T$ )

---

Reverse Rightmost Derivations in Shift-Reduce-Parsers

**Idea:** Observe reverse rightmost-derivations of $M_C$!

**Input:**
\[ + \ 40 \]

**Pushdown:**
( $q_0 \ E$ )
Reverse Rightmost Derivations in Shift-Reduce-Parsers

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

Input: + 40

Pushdown: ($q_0 E$)

Reverse Rightmost Derivations in Shift-Reduce-Parsers

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

Input: * 2 + 40

Pushdown: ($q_0 F$)
Reverse Rightmost Derivations in Shift-Reduce-Parsers

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

Input: 

$$q_0 E + T$$

Pushdown: 

$$(q_0 E + T)$$

Generic Observation: 

In a sequence of configurations of $M_G^R$

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

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

Reverse Rightmost Derivations in Shift-Reduce-Parsers

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

Input: 

$$+ 40$$

Pushdown: 

$$(q_0 E + T)$$

Generic Observation: 

In a sequence of configurations of $M_G^R$

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

we call $\alpha \gamma$ a viable prefix for the complete item $[B \rightarrow \gamma]$.
Bottom-up Analysis: Viable Prefix

\[ \alpha \gamma \text{ is viable for } [B \rightarrow \gamma \bullet] \iff S \rightarrow^*_R \alpha B \gamma \]

\[ \begin{array}{c}
\alpha_1 \\
\alpha_2 \\
\vdots \\
\alpha_m
\end{array} \]

\[ \begin{array}{c}
A_1 \\
A_2 \\
B \\
i
\end{array} \]

\[ \alpha = \alpha_1 \ldots \alpha_m \]

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:

LR(0)-Parser

Example:

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

Therefore we determine:

\[ \begin{array}{c}
q_1 = \{ [S' \rightarrow E \bullet], [E \rightarrow E + T \bullet] \} \\
q_2 = \{ [E \rightarrow T \bullet], [T \rightarrow T + F \bullet] \} \\
q_3 = \{ [T \rightarrow F \bullet] \} \\
q_4 = \{ [F \rightarrow \text{int} \bullet] \} \\
q_5 = \{ [E \rightarrow E + T \bullet], [T \rightarrow T * F \bullet] \} \\
q_6 = \{ [E \rightarrow E \bullet] \} \\
q_7 = \{ [T \rightarrow T + F \bullet] \} \\
q_8 = \{ [T \rightarrow F \bullet] \} \\
q_9 = \{ [T \rightarrow T * F \bullet] \} \\
q_{10} = \{ [T \rightarrow F \bullet] \} \\
q_{11} = \{ [F \rightarrow ( E ) \bullet] \}
\end{array} \]

The final states \( q_1, q_2, q_9 \) contain more than one admissible item
\[ \Rightarrow \text{non deterministic!} \]
**LR(0)-Parser**

... for example:

\[ q_1 = \{(S \to E \bullet), \} \]

\[ q_2 = \{[E \to E \bullet + T], \} \]

\[ q_3 = \{[T \to T \bullet * F], \} \]

\[ q_4 = \{[F \to \mathtt{int} \bullet], \} \]

The final states \( q_1, q_2, q_3 \) contain more than one admissible item \( \Rightarrow \) non deterministic!

**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 \( M_G^R \).

we conclude:

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

**Canonical LR(0)-Automaton**

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

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

... for example:

**Revisiting the Conflicts of the LR(0)-Automaton**

What differentiates the particular Reductions and Shifts?

Input:

\[ * \ 2 + 40 \]

Pushdown:

( \( q_0 \ T \) )
Revisiting the Conflicts of the LR(0)-Automaton

What differentiates the particular Reductions and Shifts?

Input:

\[ + \ 40 \]

Pushdown:

\[ (q_0 \ T) \]

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

Revisiting the Conflicts of the LR(0)-Automaton

What differentiates the particular Reductions and Shifts?

Input:

\[ * 2 + 40 \]

Pushdown:

\[ (q_0 \ E) \]

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

Revisiting the Conflicts of the LR(0)-Automaton

What differentiates the particular Reductions and Shifts?

Input:

\[ + 40 \]

Pushdown:

\[ (q_0 \ T) \]

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

Revisiting the Conflicts of the LR(0)-Automaton

Idea: Matching lookahead with right context matters!

Input:

\[ * 2 + 40 \]

Pushdown:

\[ (q_0 \ E) \]

\[
E \rightarrow E + T \quad | \quad T \\
T \rightarrow T * F \quad | \quad F \\
F \rightarrow (E) \quad | \quad \text{int}
\]
Revisiting the Conflicts of the LR(0)-Automaton

Idea: Matching lookahead with right context matters!

Input:

Pushdown:

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:

\[
\begin{aligned}
S &\rightarrow \alpha Aw \\
S &\rightarrow \alpha A'w' \\
\end{aligned}
\]

follows: \( \alpha = \alpha' \wedge A = A' \wedge w' = x \)

LR(k)-Grammars

for example:

\[
\begin{aligned}
A &\rightarrow aAb \mid 0 \\
B &\rightarrow aBbb \mid 1 \\
\end{aligned}
\]

\[
\begin{aligned}
a &\rightarrow A & ab &\rightarrow A & 0 &\rightarrow aA \\
&\rightarrow 0 &\rightarrow 0 &\rightarrow \alpha &\beta &\rightarrow \alpha &\beta \end{aligned}
\]
LR(k)-Grammars

for example:

(1) \[ S \to A \mid B \quad A \to aA b \mid 0 \quad B \to aB b b \mid 1 \]
... is not \( LL(k) \) for any \( k \) — but \( LR(0) \):

Let \( S \to a^a \alpha X w \to \alpha \beta w \). Then \( \alpha \beta \) is of one of these forms:

\[
A, B, a^a aA b, a^a aB b b, a^a 0, a^a 1 \quad (n \geq 0)
\]

(2) \[ S \to aA c \quad A \to a b b \mid b \]

LR(k)-Grammars

for example:

(1) \[ S \to A \mid B \quad A \to aA b \mid 0 \quad B \to aB b b \mid 1 \]
... is not \( LL(k) \) for any \( k \) — but \( LR(0) \):

Let \( S \to a^a \alpha X w \to \alpha \beta w \). Then \( \alpha \beta \) is of one of these forms:

\[
A, B, a^a aA b, a^a aB b b, a^a 0, a^a 1 \quad (n \geq 0)
\]

LR(k)-Grammars

for example:

(1) \[ S \rightarrow A \mid B \quad A \rightarrow aAa \mid 0 \quad B \rightarrow aBbb \mid 1 \]
    ... is not \( LL(k) \) for any \( k \) — but \( LR(0) \):

Let \( S \rightarrow \alpha Xw \rightarrow \alpha \beta w \). Then \( \alpha \beta \) is of one of these forms:

\[ A, B, a^n aAa, a^n aBbb, a^n 0, a^n 1 \quad (n \geq 0) \]

(2) \[ S \rightarrow aAc \quad A \rightarrow bbb \mid b \]
    ... is also not \( LL(k) \) for any \( k \):

Let \( S \rightarrow \alpha Xw \rightarrow \alpha \beta w \). Then \( \alpha \beta \) is of one of these forms:

\[ a b, aAa, AAc \]

LR(k)-Grammars

for example:

(3) \[ S \rightarrow aAc \quad A \rightarrow bba \mid b \]
    Let \( S \rightarrow \alpha Xw \rightarrow \alpha \beta w \) with \( \{y\} = \text{First}_k(w) \), then \( \alpha \beta y \) is of one of these forms:

\[ aAc, (bb)^nA, a(nbb) \]

LR(k)-Grammars

for example:

(3) \[ S \rightarrow aAc \quad A \rightarrow bba \mid b \]
    Let \( S \rightarrow \alpha Xw \rightarrow \alpha \beta w \) with \( \{y\} = \text{First}_k(w) \), then \( \alpha \beta y \) is of one of these forms:

\[ a b^n b c, a b^n b bAc, aAc \]
for example:

(3) \[ S \rightarrow a A c \quad A \rightarrow b b A \mid b \] ... is not LR(0), but LR(1):

Let \[ S \rightarrow^*_R \alpha X w \rightarrow \alpha \beta w \] with \( \{ y \} = \text{First}_k(w) \) then

\( \alpha \beta y \) is of one of these forms:

\[ a b^{2n} b c , \ a b^{2n} b b A c , \ a A c \]

(4) \[ S \rightarrow a A c \quad A \rightarrow b A b \]

LR(1)-Parsing

Idea: Let's equip items with 1-lookahead

Definition LR(1)-Item

An LR(1)-item is a pair \([B \rightarrow \alpha \bullet \beta, \nu]\) with

\[ x \in \text{Follow}_1(B) = \bigcup \{ \text{First}_1(\nu) \mid S \rightarrow^* \nu B \nu \} \]
Admissible LR(1)-Items

The item \([B \rightarrow \alpha \cdot \beta, x]\) is \textit{admissible} for \(\gamma \) if:
\[
S \rightarrow_R^* \gamma B w \quad \text{with} \quad \{x\} = \text{First}_1(w)
\]

The Canonical LR(1)-Automaton

The canonical LR(1)-automaton \(LR(G, 1)\) is created from \(c(G, 1)\), by performing arbitrarily many \(\epsilon\)-transitions and then making the resulting automaton deterministic ...

But again, it can be constructed directly from the grammar; analogously to \(LR(0)\), we need the \(\epsilon\)-closure \(\delta^*_\epsilon\) as a helper function:
\[
\delta^*_\epsilon(q) = q \cup \{[A \rightarrow \alpha \cdot B \beta', x'] \in q, \beta' \in (N \cup T)^*: \gamma \rightarrow^* \gamma \beta \land x \in \text{First}_1(\gamma \beta') \cup \{x'\}\}
\]

Then, we define:

\textbf{States:} Sets of LR(1)-items;
\textbf{Start state:} \(q_0 = [\epsilon S, \epsilon]\);
\textbf{Final states:} \(\{q \mid \exists A \rightarrow \alpha \in P: [A \rightarrow \alpha \cdot \epsilon, x] \in q\}\)
\textbf{Transitions:} \(\delta(q, X) = \delta^*_\epsilon([A \rightarrow \alpha X \cdot \beta, x] \cup [A \rightarrow \alpha \cdot X \beta, x] \in q}\)

The Characteristic LR(1)-Automaton

The set of admissible LR(1)-items for viable prefixes is again computed with the help of the finite automaton \(c(G, 1)\).

The automaton \(c(G, 1)\):

\textbf{States:} LR(1)-items
\textbf{Start state:} \([S' \rightarrow \epsilon, \epsilon]\)
\textbf{Final states:} \(\{[B \rightarrow \gamma \cdot x, \epsilon] \mid B \rightarrow \epsilon \in F, x \in \text{First}_1(\beta) \cup \{x\}\}\)
\textbf{Transitions:}
\begin{align*}
(1) & \quad ([A \rightarrow \alpha \cdot X \beta, x], X, [A \rightarrow \alpha \cdot X \cdot \beta, x]), \quad X \in (N \cup T) \\
(2) & \quad ([A \rightarrow \alpha \cdot B \beta, \epsilon], [B \rightarrow \epsilon \gamma, X']), \\
& \quad A \rightarrow \alpha B \beta, \quad B \rightarrow \epsilon \gamma \in P, x' \in \text{First}_1(\beta) \cup \{x\};
\end{align*}

This automaton works like \(c(G)\) — but additionally manages a 1-prefix from \(\text{Follow}_1\) of the left-hand sides.

The Canonical LR(1)-Automaton

For example:

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

\[
\text{First}_1(S') = \text{First}_1(E) = \text{First}_1(T) = \text{First}_1(F) = \text{name, int, (}
\]

\[
q_0 = \{(E, \{\})\}, \quad q_3 = \delta(q_0, F) = \{[T \rightarrow F \cdot \epsilon]\}
\]
\[
q_1 = \delta(q_0, E) = \{(E, \{\})\}, \quad q_4 = \delta(q_3, \text{int}) = \{[E \rightarrow \text{int} \cdot \epsilon]\}
\]
\[
q_2 = \delta(q_0, T) = \{(T, \{\})\}, \quad q_5 = \delta(q_3, \text{int}) = \{[T \rightarrow T \cdot F \cdot \epsilon]\}
\]
The Canonical LR(1)-Automaton

For example:

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

\[
\delta(q_0, E) = \{(\epsilon)\} \\
\delta(q_1, T) = \{(\epsilon)\} \\
\delta(q_2, T) = \{(\epsilon)\}
\]

\[
\begin{align*}
\text{First}_1(S') &= \text{First}_1(E) = \text{First}_1(T) = \text{First}_1(F) = \text{name, int, (}
\end{align*}
\]
The Canonical LR(1)-Automaton

For example:

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

First \( S' \) = First \( E \) = First \( T \) = First \( F \) = name, int, (

\[
\begin{align*}
q_s & = \delta(q_s, () ) = \{ \begin{array}{c} 
E \rightarrow E + T
\end{array} \}, \\
q_3 & = \delta(q_3, +) = \{ \begin{array}{c} 
T \rightarrow T * F \\
F \rightarrow (E) \\
F \rightarrow \text{int}
\end{array} \}, \\
q_{10} & = \delta(q_{10}, F) = \{ \begin{array}{c} 
T \rightarrow T * F \\
F \rightarrow (E) \\
F \rightarrow \text{int}
\end{array} \}, \\
q_{11} & = \delta(q_{11}, ()) = \{ \begin{array}{c} 
F \rightarrow (E) \\
F \rightarrow \text{int}
\end{array} \}
\end{align*}
\]

First \( S' \) = First \( E \) = First \( T \) = First \( F \) = name, int, (

\[
\begin{align*}
q_s & = \delta(q_s, () ) = \{ \begin{array}{c} 
E \rightarrow (E), (\ ), +, )
\end{array} \}, \\
q_7 & = \delta(q_7, +) = \{ \begin{array}{c} 
T \rightarrow T * F \\
F \rightarrow (E) \\
F \rightarrow \text{int}
\end{array} \}, \\
q_{10} & = \delta(q_{10}, F) = \{ \begin{array}{c} 
T \rightarrow T * F \\
F \rightarrow (E) \\
F \rightarrow \text{int}
\end{array} \}, \\
q_{11} & = \delta(q_{11}, ()) = \{ \begin{array}{c} 
F \rightarrow (E) \\
F \rightarrow \text{int}
\end{array} \}
\end{align*}
\]
The Canonical LR(1)-Automaton

Discussion:
- In the example, the number of states was almost doubled... and it can become even worse.
- The conflicts in states $q_1, q_2, q_3$ are now resolved!
  e.g. we have for:

\[
q_3 = \{[E \to E + T \cdot, \{\epsilon, +\}],

T \to T \cdot F, \{\epsilon, +, *\}\}\}
\]

with:

\[
\{\epsilon, +\} \cap (\text{First}\{F\} \cap \{\epsilon, +, *\}) = \{\epsilon, +\} \cap \{\epsilon, +\} = \emptyset
\]

The LR(1)-Parser

The construction of the LR(1)-parser:

States: $Q \cup \{f\}$  ($f$ fresh)

Start state: $q_0$

Final state: $f$

Transitions:

Shift: $(p, a, p) \quad \text{if} \quad q = \text{goto}[q, a],
\quad s = \text{action}[p, w]$  

Reduce: $(p q_1 \ldots q_{|\beta|}, \epsilon, p) \quad \text{if} \quad [A \to \beta \cdot] \in q_{|\beta|},
\quad q = \text{goto}(p, A),
\quad [A \to \beta \cdot] = \text{action}[q_{|\beta|}, w]$  

Finish: $(q_0 p, \epsilon, f) \quad \text{if} \quad [S' \to S \cdot] \in p$  

with $LR(G, 1) = (Q, T, \delta, q_0, F)$.
The LR(1)-Parser

The construction of the LR(1)-parser:

States: \( Q \cup \{f\} \quad (f \text{ fresh}) \)
Start state: \( q_0 \)
Final state: \( f \)
Transitions:
Shift: \( (p, a, p' q) \quad \text{if} \quad q = \text{goto}[q, a], \)
\( s = \text{action}[p, w] \)
Reduce: \( (q_1 \ldots q_{|\beta|}, \epsilon, p q) \quad \text{if} \quad [A \rightarrow \beta \bullet] \in q_{|\beta|}, \)
\( q = \text{goto}(p, A), \)
\( [A \rightarrow \beta \bullet] = \text{action}[q_{|\beta|}, w] \)
Finish: \( (q_0, p, \epsilon, f) \quad \text{if} \quad [S' \rightarrow S \bullet] \in P \)

with \( LR(G, 1) = (Q, T, \delta, q_0, F) \).

The Canonical LR(1)-Automaton

In general:

We identify two conflicts:

Reduce-Reduce-Conflict:
\[ [A \rightarrow \gamma \bullet, x], \quad [A' \rightarrow \gamma' \bullet, y] \in q \quad \text{with} \quad A \neq A' \vee \gamma \neq \gamma' \]

Shift-Reduce-Conflict:
\[ [A \rightarrow \gamma \bullet, x], \quad [A' \rightarrow \alpha \bullet a \beta, y] \in q \quad \text{with} \quad a \in T \text{ und } x \in \{a\} \circ_k \text{First}_k(\beta) \circ_k \{y\} . \]

for a state \( q \in Q \).

Such states are now called LR(k)-unsuited

Special LR(k)-Subclasses

Theorem:
A reduced contextfree grammar \( G \) is called \( LR(k) \) iff the canonical \( LR(k) \)-automaton \( LR(G, k) \) has no \( LR(k) \)-unsuited states.

Discussion:

- Our example apparently is \( LR(1) \)
- In general, the canonical \( LR(k) \)-automaton has much more states then \( LR(G) = LR(G, 0) \)
- Therefore in practice, subclasses of \( LR(k) \)-grammars are often considered, which only use \( LR(G) \)...
Lexical and Syntactical Analysis:

From Regular Expressions to Finite Automata

From Finite Automata to Scanners

Parsing Methods

Discussion:
- All context-free languages, that can be parsed with a deterministic pushdown automaton, can be characterized with an LR(1)-grammar.
- LR(0)-grammars describe all prefixfree deterministic context-free languages.
- The language-classes of LL(k)-grammars form a hierarchy within the deterministic context-free languages.
Lexical and Syntactical Analysis:

From characteristic to canonical Automata:

From Shift-Reduce-Parsers to LR(1)-Parsers: