**LR(k)-Grammars**

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

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

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

**Strategy** for testing Grammars for LR($k$)-property
- Focus iteratively on all rightmost derivations $S \rightarrow^* R_\alpha X \rightarrow \alpha \beta w$
- Identify handles $\alpha \beta$ in s. forms $\alpha \beta w \; (w \in T^*, \alpha, \beta \in (N \cup T)^*)$
- Determine minimal $k$, such that $\text{First}_k(w)$ associates $\beta$ with a unique $X \rightarrow \beta$ for non-prefixing $\alpha \beta$s

**for example:**

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

```
A  B  aAb  aBbb
aAAb  aAb  bbb
  aAAb
  aAAb
  aAAb
  aAb
  aB
  aB
  aB
  a
```
LR(k)-Grammars

for example:

(1) \[ S \rightarrow A \mid B \quad A \rightarrow aAb \mid 0 \quad B \rightarrow aBbb \mid 1 \]
    
    ... is not LL(k) for any \( k \) — but LR(0):
    
    Let \( S \rightarrow_R \alpha Xw \rightarrow \alpha \beta w \). Then \( \alpha \beta \) is of one of these forms:
    
    \[ A, B, a^n aAb, a^n aBbb, a^n 0, a^n 1 \quad (n \geq 0) \]

(2) \[ S \rightarrow aAc \quad A \rightarrow Abb \mid b \]
    
    \[ a^2 \]
    
    \[ a^2 \]
    
    \[ a^2 \]
    
    \[ a^2 \]

LR(k)-Grammars

for example:

(1) \[ S \rightarrow A \mid B \quad A \rightarrow aAb \mid 0 \quad B \rightarrow aBbb \mid 1 \]
    
    ... is not LL(k) for any \( k \) — but LR(0):
    
    Let \( S \rightarrow_R \alpha Xw \rightarrow \alpha \beta w \). Then \( \alpha \beta \) is of one of these forms:
    
    \[ A, B, a^n aAb, a^n aBbb, a^n 0, a^n 1 \quad (n \geq 0) \]

(2) \[ S \rightarrow aAc \quad A \rightarrow Abb \mid b \]
    
    \[ a^2 \]
    
    \[ a^2 \]
    
    \[ a^2 \]
    
    \[ a^2 \]

LR(k)-Grammars

for example:

(1) \[ S \rightarrow A \mid B \quad A \rightarrow aAb \mid 0 \quad B \rightarrow aBbb \mid 1 \]
    
    ... is not LL(k) for any \( k \) — but LR(0):
    
    Let \( S \rightarrow_R \alpha Xw \rightarrow \alpha \beta w \). Then \( \alpha \beta \) is of one of these forms:
    
    \[ A, B, a^n aAb, a^n aBbb, a^n 0, a^n 1 \quad (n \geq 0) \]

(2) \[ S \rightarrow aAc \quad A \rightarrow Abb \mid b \]
    
    \[ a^2 \]
    
    \[ a^2 \]
    
    \[ a^2 \]
    
    \[ a^2 \]
LR(k)-Grammars

for example:

(3) \( S \to aAc \quad A \to b b A \mid b \)

\[
\begin{align*}
&\ aA_c \\
&\ a b A c \\
&\ a b A_c \\
&\ a b c
\end{align*}
\]

is of one of these forms:

\( a b^{2n} b c, a b^{2n} b b A c, a A c \)

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

\( a b^{2n} b c, a b^{2n} b b A c, a A c \)
LR(k)-Grammars

for example:

(3) \( S \rightarrow \alpha A c \quad A \rightarrow b b A \mid b \quad \ldots \text{is not } LR(0) \), but \( LR(1) \):
Let \( S \rightarrow \alpha X w \rightarrow \alpha \beta w \) with \( \{ y \} = \text{First}_k(w) \) then
\[ \alpha \beta y \quad \text{is of one of these forms:} \]
\[ a b^{2n} b c, \quad a b^{2n} b b A c, \quad a A c \]

(4) \( S \rightarrow a A c \quad A \rightarrow b A b \mid b \quad \ldots \text{is not } LR(0), \text{ but } LR(1): \)
Consider the rightmost derivations:
\[ S \rightarrow \underbrace{a b^n A b^n c}_{R} \rightarrow a b^n b b^n c \]

LR(1)-Parsing

Idea: Let’s equip items with 1-lookahead

\textbf{Definition LR(1)-Item}
An \( LR(1) \)-item is a pair \( \boxed{B \rightarrow \alpha \cdot \beta \cdot x} \) with
\[ x \in \text{Follow}_1(B) = \bigcup \{ \text{First}_1(v) \mid S \rightarrow \gamma w \} \]

Admissible LR(1)-Items

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

... with \( \gamma_0 \ldots \gamma_m = \gamma \)
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 \( c(G, 1) \):

- **States:** \( LR(1) \)-items
- **Start state:** \( [S' \rightarrow \bullet S, \epsilon] \)
- **Final states:** \( \{ [B \rightarrow \gamma \bullet, x] \mid B \rightarrow \gamma \in P, x \in \text{Follow}_1(B) \} \)

**Transitions:**

1. \( (A \rightarrow \alpha \bullet X \beta, x), X, [A \rightarrow \alpha X \bullet \beta, x], \ X \in (N \cup T) \)
2. \( (A \rightarrow \alpha \bullet B \beta, x), \ [B \rightarrow \bullet \gamma, x'], \ A \rightarrow \alpha B \beta, \ B \rightarrow \gamma \in P, x' \in \text{First}_1(\beta) \cup_1 \{ x \}; \)

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

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...
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:

\[
\delta^*_\epsilon(q) = q \cup \{ [C \rightarrow \bullet \gamma, x] \mid \exists [A \rightarrow \alpha \bullet B \beta', x'] \in q, \beta' \in (N \cup T)^* : B \rightarrow^{*} C \beta \land x \in \text{First}_1(\beta \beta') \cap \{ x' \} \}
\]

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 \( c(G, 1) \):

- **States**: LR(1)-items
- **Start state**: \([S' \rightarrow \bullet S, \epsilon]\)
- **Final states**: \( \{ [B \rightarrow \gamma \bullet, x] \mid B \rightarrow \gamma \in P, x \in \text{Follow}_1(B) \} \)
- **Transitions**:  
  1. \( [A \rightarrow \alpha \bullet X \beta, x], [A \rightarrow \alpha X \bullet \beta, x] \), \( X \in (N \cup T) \)
  2. \( [A \rightarrow \alpha \bullet X, x], [B \rightarrow \bullet \gamma, x'] \), \( A \rightarrow \alpha B \beta, B \rightarrow \gamma \in P, x' \in \text{First}_1(\beta) \cap \{ x \} \)

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

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 \{ [C \rightarrow \bullet, \gamma, x] \mid \exists [A \rightarrow \alpha \bullet B \beta', x'] \in q, \beta \in (N \cup T)^* : \exists \gamma \in \text{First}_1(\beta \beta') \cup \{ \epsilon \} \}

Then, we define:

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

For example:

$E \rightarrow E + T \quad | \quad T$

$T \rightarrow T \star F \quad | \quad F$

$F \rightarrow (E) \quad | \quad \text{int}$

Then, we define:

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

For example:

$q_0 = \{ [S' \rightarrow E, \bullet, \gamma] \}$

$q_3 = \delta(q_0, E) = \{ [T \rightarrow F \bullet, \{ \epsilon, +, * \}] \}$

$q_4 = \delta(q_0, F) = \{ [F \rightarrow \text{int}, \{ \epsilon, +, * \}] \}$

$q_5 = \delta(q_0, \text{int}) = \{ [F \rightarrow \text{int}, \{ \epsilon, +, * \}] \}$

$q_1 = \delta(q_0, E) = \{ [E \rightarrow E + T, \{ \epsilon, + \}] \}$

$q_2 = \delta(q_0, T) = \{ [T \rightarrow T \star F, \{ \epsilon, + \}] \}$
The Canonical LR(1)-Automaton

For example:

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

First\(_1\)(S\(^\prime\)) = First\(_1\)(E) = First\(_1\)(T) = First\(_1\)(F) = name, int, ( 

\[
\begin{align*}
q_0 &= (\text{|$S'$| to } E, \{e\}, [E \rightarrow E + T, \{e, +\}], [E \rightarrow E + T, \{e, +, +\}]) \\
q_3 &= \delta(q_0, F) = ([T \rightarrow F \cdot, \{e, +, +\}]) \\
q_4 &= \delta(q_3, \text{int}) = ([F \rightarrow \text{int}, \{e, +, +\}]) \\
\end{align*}
\]

\[
\begin{align*}
q_1 &= \delta(q_0, E) = ([S' \rightarrow F \cdot, \{e\}, [E \rightarrow E + T, \{e, +\}]) \\
q_2 &= \delta(q_0, T) = ([E \rightarrow E + T, \{e, +\}], [T \rightarrow T \cdot F, \{e, +, +\}])
\end{align*}
\]
Discussion:
- In the example, the number of states was almost doubled ... and it can become even worse.
- The conflicts in states $q_1, q_2, q_0$ are now resolved!
  e.g. we have for:

$$q_f = \{ [E \rightarrow E + T], [T \rightarrow T \cdot F], \{ \epsilon, +, * \} \}$$

with:

$$\{ \epsilon, + \} \cap \text{First}_1(*, F) \cap \{ \epsilon, +, * \} = \{ \epsilon, + \} \cap \{ * \} = \emptyset$$

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: $(p, a, p[q])$ if $q = \text{goto}[q][a, \epsilon]$  
Reduce: $[q_1, \ldots, q_i, \beta, \epsilon, p, q]$ if $A \rightarrow \beta \in \text{q}[\beta]$, $q = \text{goto}[q, A]$  
Finish: $[q_0, p, \epsilon]$ if $S \rightarrow S \in p$

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

The LCIR(1)-Parser:

The action-table encodes the transitions:

$$\text{goto}[q, X] = \delta(q, X) \in Q$$

The goto-table encodes the transitions:

The action-table describes for every state $q$ and possible lookahead $w$ the necessary action.
The LR(1)-Parser:

Possible actions are:
- **Shift**
- **Reduce** \( \langle A \rightarrow \gamma \rangle \)
- **Error**

... for example:

\[
\begin{align*}
E & \rightarrow E + T^0 & T^1 \\
T & \rightarrow T * F^0 & F^1 \\
F & \rightarrow \langle E \rangle^0 & \text{int}^1
\end{align*}
\]

<table>
<thead>
<tr>
<th>action</th>
<th>( \epsilon ) int (()</th>
<th>( )</th>
<th>( )</th>
<th>( )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_1 )</td>
<td>( S', 0 )</td>
<td>( E, 1 )</td>
<td>( s )</td>
<td></td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( E, 1 )</td>
<td>( E, 1 )</td>
<td>( s )</td>
<td></td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( T, 1 )</td>
<td>( T, 1 )</td>
<td>( T, 1 )</td>
<td></td>
</tr>
<tr>
<td>( q_4 )</td>
<td>( F, 1 )</td>
<td>( F, 1 )</td>
<td>( F, 1 )</td>
<td></td>
</tr>
<tr>
<td>( q_5 )</td>
<td>( E, 0 )</td>
<td>( E, 0 )</td>
<td>( s )</td>
<td></td>
</tr>
<tr>
<td>( q_6 )</td>
<td>( T, 0 )</td>
<td>( T, 0 )</td>
<td>( T, 0 )</td>
<td></td>
</tr>
<tr>
<td>( q_7 )</td>
<td>( F, 0 )</td>
<td>( F, 0 )</td>
<td>( F, 0 )</td>
<td></td>
</tr>
</tbody>
</table>

The Canonical LR(1)-Automaton

In general: We identify two conflicts:

**Reduce-Reduce-Conflict:**

\[
\langle A \rightarrow \gamma \bullet \rangle, \quad \langle A' \rightarrow \gamma' \bullet \rangle \in q \quad \text{with} \quad A \neq A' \vee \gamma \neq \gamma'
\]

**Shift-Reduce-Conflict:**

\[
\langle A \rightarrow \gamma \bullet \rangle, \quad \langle A' \rightarrow \alpha \bullet \rangle \in q \quad \text{with} \quad a \in T \quad \text{and} \quad x \in \{a\} \quad \text{for a state} \quad q \in Q.
\]

Such states are now called \( LR(1) \)-unsuited

The Canonical LR(1)-Automaton

In general: We identify two conflicts:

**Reduce-Reduce-Conflict:**

\[
\langle A \rightarrow \gamma \bullet \rangle, \quad \langle A' \rightarrow \gamma' \bullet \rangle \in q \quad \text{with} \quad A \neq A' \vee \gamma \neq \gamma'
\]

**Shift-Reduce-Conflict:**

\[
\langle A \rightarrow \gamma \bullet \rangle, \quad \langle A' \rightarrow \alpha \bullet \rangle \in q \quad \text{with} \quad a \in T \quad \text{and} \quad x \in \{a\} \quad \text{for a state} \quad q \in Q.
\]

Such states are now called \( LR(1) \)-unsuited

Special LR(k)-Subclasses

**Theorem:**

A reduced context-free grammar \( G \) is called \( LR(k) \) if the canonical \( LR(k) \)-automaton \( LR(G, k) \) has no \( LR(k) \)-unsuited states.

Such states are now called \( LR(k) \)-unsuited
**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 $LR(G)$ ...

**Parsing Methods**

```
deterministic languages
  = LR(1) = ... = LR(k)
```

- LALR(k)
- SLR(k)
- LR(0)
- regular languages
- LL(1)...

---

**Lexical and Syntactical Analysis:**

**Concept of specification and implementation:**

```
0 | [1-9][0-9]*  
```

```
E→E(op)E
```

**Generator**
Lexical and Syntactical Analysis:

From Regular Expressions to Finite Automata

From Finite Automata to Scanners

Lexical and Syntactical Analysis:

Computation of lookahead sets:

From Item-Pushdown Automata to LL(1)-Parsers:

Mini-Projects

1. parse Regular Expressions with PEGs and Recursive Descent Parsing
2. parse Regular Expressions with CUP
3. parse Regular Expressions with ANTLR
4. generate NFA-Transition table from Regex Trees via Thompson Construction
5. generate NFA-Transition table from Regex Trees via Berry-Sethi Construction
6. generate NFA-Transition table from Regex Trees via Antimirov Automaton Construction
7. generate NFA-Transition table from Regex Trees via Follow Automaton Construction
8. implement flex based on Regex -> NFA Module
9. extend simpleC by ellipses, enums and unions
10. extend simpleC by typecasts and type checking
11. parsing BNF grammars with CUP and computing First/Follow
12. parsing BNF grammars with ANTLR and computing First/Follow
13. parsing BNF grammars with JavaCC and computing the canonical-LR(0) Automaton
14. given each symbols First/Follow-Set construct the SLR(1) Automaton
15. computing example traces for reaching S/R and R/R conflicts in the LR(0) Automaton
16. transforming (in-)direct left-recursive grammars into (potentially) right recursive grammars
17. semi-deciding k-Ambiguity
18. C4Script (Generating LLVM from a script language)