Script generated by TTT

Title: Petter: Compilerbau (23.05.2019)

Date: Thu May 23 14:12:33 CEST 2019

Duration: 93:45 min

Pages: 31

Chapter 4:
Bottom-up Analysis

Shift-Reduce Parser

Donald Knuth

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

Shift-Reduce Parser

Construction:

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

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

$$\begin{array}{lll} \delta = & \{(q, x, q\,x) \mid q \in Q, x \in T\} \cup & // \text{ Shift-transitions} \\ & \{(q\,\alpha, \epsilon, q\,A) \mid q \in Q, A \to \alpha \in P\} \cup & // \text{ Reduce-transitions} \\ & \{(q_0\,S, \epsilon, f)\} & // \text{ finish} \end{array}$$

Example-computation:

$$\begin{array}{lll} (q_0, & a\,b) & \vdash \quad (q_0\ a, \quad b) \quad \vdash \quad (q_0\ A, \quad b) \\ & & \vdash \quad (q_0\ A\ b, \quad \epsilon) \quad \vdash \quad (q_0\ A B, \quad \epsilon) \\ & & \vdash \quad (q_0\ S, \quad \epsilon) \quad \vdash \quad (f, \quad \epsilon) \end{array}$$

Shift-Reduce Parser

Construction:

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

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

$$\begin{array}{rcll} \delta & = & \{(q, x, q\,x) \mid q \in Q, x \in T\} \cup & // \text{ Shift-transitions} \\ & & \{(q\,\alpha, \epsilon, q\,A) \mid q \in Q, A \to \alpha \in P\} \cup & // \text{ Reduce-transitions} \\ & & \{(q_0\,S, \epsilon, f)\} & // \text{ finish} \end{array}$$

Example-computation:

$$\begin{array}{rcl} (q_0, \quad a\,b) & \vdash & (q_0\ a, \quad b) \\ & \vdash & (q_0\ A\ b, \quad \epsilon) \\ & \vdash & (q_0\ S, \quad \epsilon) \end{array} \quad \begin{array}{rcl} \vdash & (q_0\ A, & b) \\ \vdash & (q_0\ A\,B, & \epsilon) \\ \vdash & (f, & \epsilon) \end{array}$$

Shift-Reduce Parser

Observation:

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

$$(\epsilon, w) \vdash^* (A, \epsilon) \quad \text{iff} \quad A \to^* w$$

- The shift-reduce pushdown automaton M_G^R is in general also non-deterministic
- For a deterministic parsing-algorithm, we have to identify computation-states for reduction

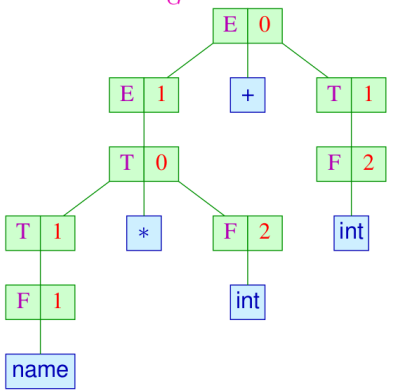
$\Longrightarrow$ LR-Parsing

Reverse Rightmost Derivations in Shift-Reduce-Parsers

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

Input:
counter $* 2 + 40$

Pushdown:
(q_0)



Code Synthesis

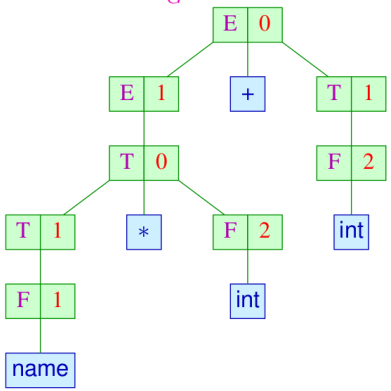
Chapter 4:
Functions

Reverse Rightmost Derivations in Shift-Reduce-Parsers

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

Input:
counter $* 2 + 40$

Pushdown:
(q_0)



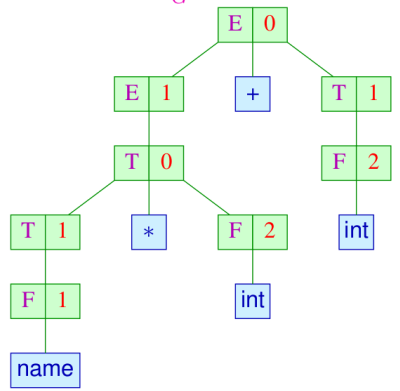
$$
\begin{array}{lcl}
E & \to & E{+}T\ {}^{0} \mid T\ {}^{1} \\
T & \to & T{*}F\ {}^{0} \mid F\ {}^{1} \\
F & \to & (\ E\ )\ {}^{0} \mid \text{name}\ {}^{1} \mid \text{int}\ {}^{2}
\end{array}
$$

Reverse Rightmost Derivations in Shift-Reduce-Parsers

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

Input:
counter $* 2 + 40$

Pushdown:
(q_0)



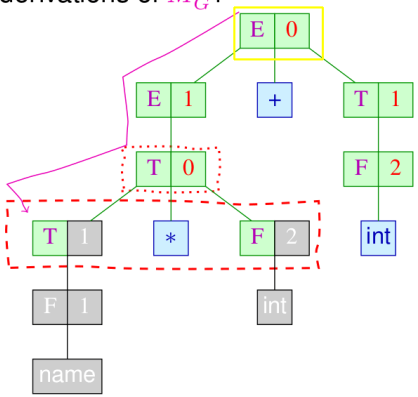
$$
\begin{array}{lcl}
E & \to & E{+}T\ {}^{0} \mid T\ {}^{1} \\
T & \to & T{*}F\ {}^{0} \mid F\ {}^{1} \\
F & \to & (\ E\ )\ {}^{0} \mid \text{name}\ {}^{1} \mid \text{int}\ {}^{2}
\end{array}
$$

Reverse Rightmost Derivations in Shift-Reduce-Parsers

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

Input:
$+ 40$

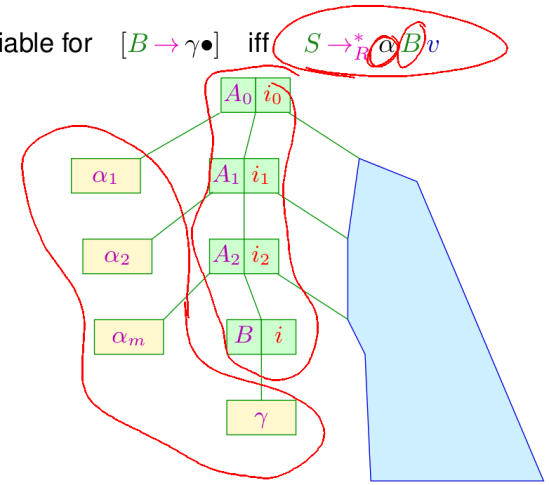
Pushdown:
(q_0 $T * F$)



$$
\begin{array}{lcl}
E & \to & E{+}T\ {}^{0} \mid T\ {}^{1} \\
T & \to & T{*}F\ {}^{0} \mid F\ {}^{1} \\
F & \to & (\ E\ )\ {}^{0} \mid \text{name}\ {}^{1} \mid \text{int}\ {}^{2}
\end{array}
$$

Bottom-up Analysis: Viable Prefix

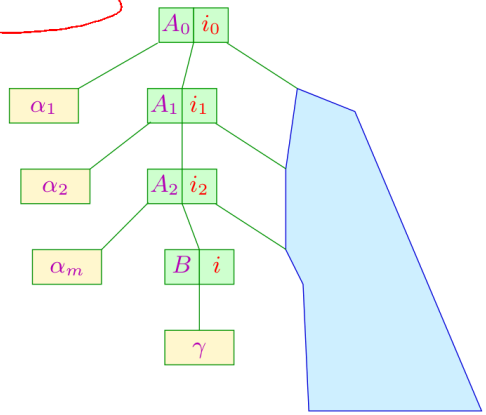
$\alpha\gamma$ is viable for $[B \to \gamma\bullet]$ iff $S \to_R^* \alpha B v$



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

Bottom-up Analysis: Viable Prefix

$\alpha\,\gamma$ is viable for $[B \to \gamma\bullet]$ iff $S \to^*_R \alpha\, B\, v$



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

Conversely, for an arbitrary valid word α' 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: $[S' \to \bullet\, S]$

Final states: $\{[B \to \gamma\bullet] \mid B \to \gamma \in P\}$

Transitions:

(1) $([A \to \alpha \bullet X\, \beta], X, [A \to \alpha\, X \bullet \beta]), \quad X \in (N \cup T), A \to \alpha\, X\, \beta \in P;$

(2) $([A \to \alpha \bullet B\, \beta], \epsilon, [B \to \bullet\, \gamma]), \quad A \to \alpha\, B\, \beta\,,\ B \to \gamma \in P;$

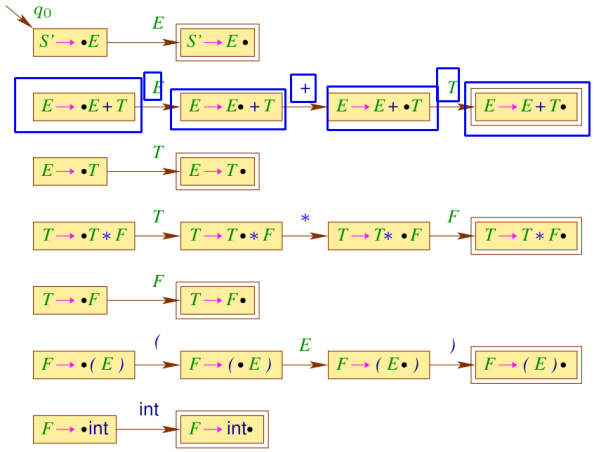
The automaton $c(G)$ is called characteristic automaton for G.

Characteristic Automaton

For example:

$$\begin{array}{lcll} E & \to & E+T & \mid T \\ T & \to & T*F & \mid F \\ F & \to & (\ E\ ) & \mid \text{int} \end{array}$$

Transitions (1)

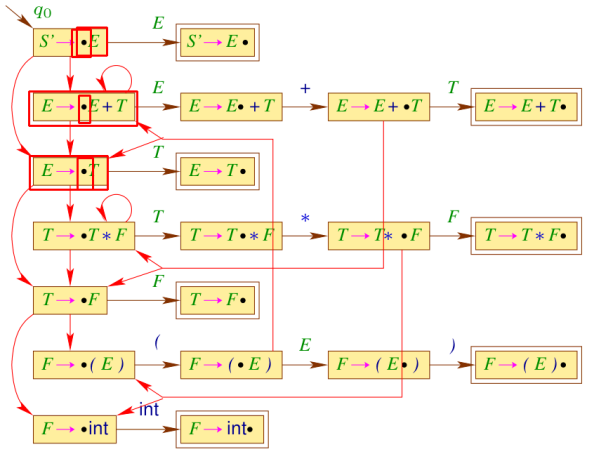


Characteristic Automaton

For example:

$$\begin{array}{lcll} E & \to & E+T & \mid T \\ T & \to & T*F & \mid F \\ F & \to & (\ E\ ) & \mid \text{int} \end{array}$$

Transitions (2)

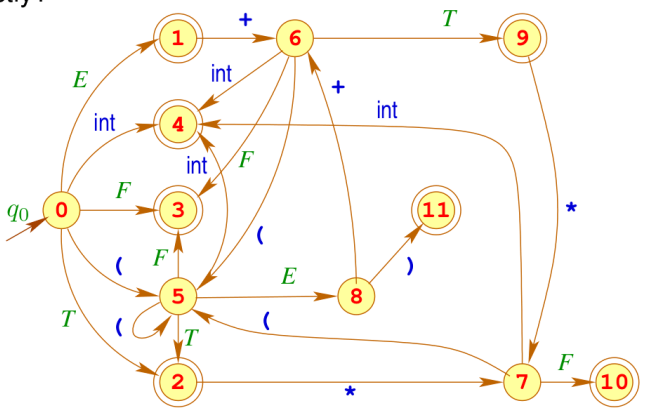


Canonical LR(0)-Automaton

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

1. performing arbitrarily many ϵ-transitions after every consuming transition
2. performing the powerset construction
3. Idea: or rather apply characteristic automaton construction to powersets directly?

... for example:

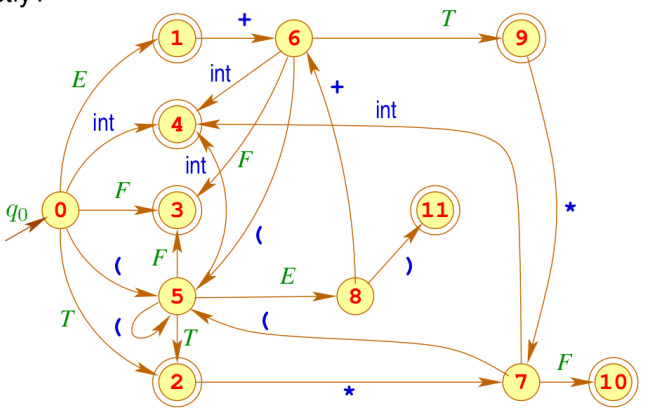


Canonical LR(0)-Automaton

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

1. performing arbitrarily many ϵ-transitions after every consuming transition
2. performing the powerset construction
3. Idea: or rather apply characteristic automaton construction to powersets directly?

... for example:



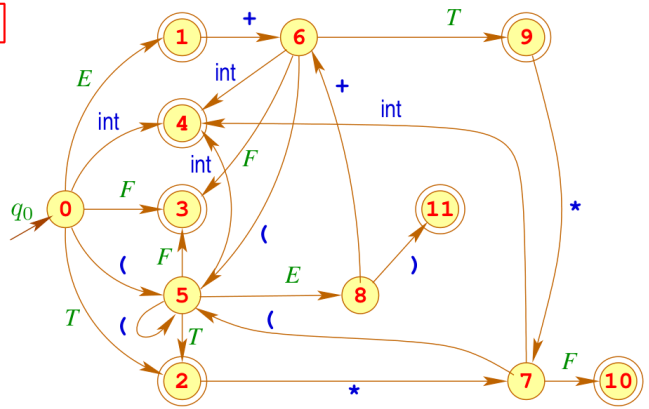
124 / 287

Canonical LR(0)-Automaton

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

1. performing arbitrarily many ϵ-transitions after every consuming transition
2. performing the powerset construction

... for example:

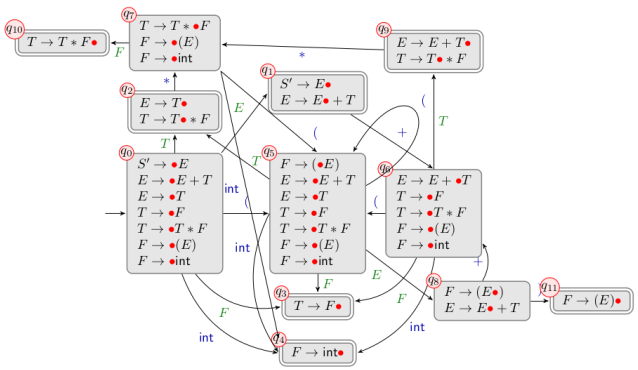


124 / 287

Canonical LR(0)-Automaton

For example:

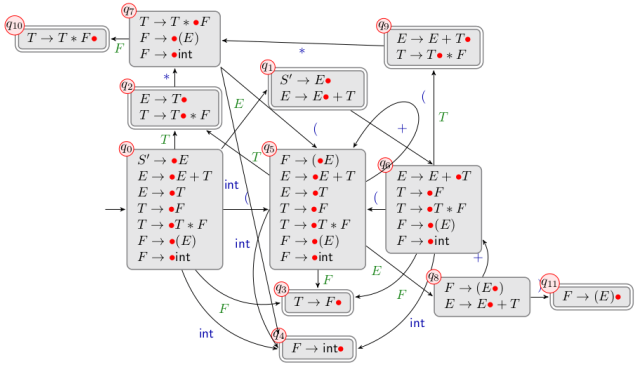
$$
\begin{array}{lcl}
S' & \rightarrow & E \\
E & \rightarrow & E + T \mid T \\
T & \rightarrow & T * F \mid F \\
F & \rightarrow & (E) \mid \text{int}
\end{array}
$$



Canonical LR(0)-Automaton

For example:

$$\begin{array}{lcl} S' & \to & E \\ E & \to & E + T \mid T \\ T & \to & T * F \mid F \\ F & \to & (E) \mid \text{int} \end{array}$$



Canonical LR(0)-Automaton

Observation:

The canonical $LR(0)$-automaton can be created directly from the grammar.
For this we need a helper function δ^*_ϵ (ϵ-closure)

$$\delta^*_\epsilon(q) = q \cup \{[B \to \bullet\, \gamma] \mid B \to \gamma \in P, \; [A \to \alpha \bullet B'\, \beta'] \in q, \; B' \to^* B\, \beta\}$$

We define:

States: Sets of items;

Start state: $\delta^*_\epsilon\{[S' \to \bullet\, S]\}$

Final states: $\{q \mid [A \to \alpha \bullet] \in q\}$

Transitions: $\delta(q, X) = \delta^*_\epsilon\{[A \to \alpha\, X \bullet \beta] \mid [A \to \alpha \bullet X \beta] \in q\}$

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 $A \to \gamma$, if $[A \to \gamma \bullet]$ is admissible for α

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 $A \to \gamma$ leads to popping the uppermost $|\gamma|$ states and continue with the state on top of the stack and input A.

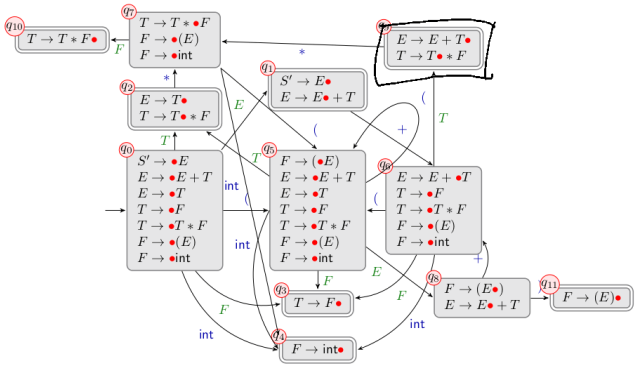
Attention:

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

Canonical LR(0)-Automaton

For example:

$$\begin{array}{lcl} S' & \to & E \\ E & \to & E + T \mid T \\ T & \to & T * F \mid F \\ F & \to & (E) \mid \text{int} \end{array}$$



... for example:

$$
\begin{array}{lll}
q_1 = \{[S' \to E\,\bullet], & & \\
\quad [E \to E\,\bullet + T]\} & & \\
q_2 = \{[E \to T\,\bullet], & q_9 = & \{[E \to E + T\,\bullet], \\
\quad [T \to T\,\bullet * F]\} & & \quad [T \to T\,\bullet * F]\} \\
q_3 = \{[T \to F\,\bullet]\} & q_{10} = & \{[T \to T * F\,\bullet]\} \\
q_4 = \{[F \to \mathsf{int}\,\bullet]\} & q_{11} = & \{[F \to (\,E\,)\,\bullet]\}
\end{array}
$$

The final states q_1, q_2, q_9 contain more than one admissible item

$\Rightarrow$ non deterministic!

Attention:

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

We identify two reasons:

Reduce-Reduce-Conflict:

$[A \to \gamma\,\bullet]\,,\ [A' \to \gamma'\,\bullet] \ \in\ q$ with $A \neq A' \vee \gamma \neq \gamma'$

Shift-Reduce-Conflict:

$[A \to \gamma\,\bullet]\,,\ [A' \to \alpha\,\bullet\,a\,\beta] \ \in\ q$ with $a \in T$

for a state $q \in Q$.

Those states are called $LR(0)$-unsuited.

The construction of the $LR(0)$-parser:

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

Start state: q_0

Final state: f

Transitions:

Shift:	$(p, a, p\,q)$	if	$q = \delta(p, a) \neq \emptyset$
Reduce:	$(p\,q_1 \ldots q_m, \epsilon, p\,q)$	if	$[A \to X_1 \ldots X_m\,\bullet] \in q_m$, $q = \delta(p, A)$
Finish:	$(q_0\,p, \epsilon, f)$	if	$[S' \to S\bullet] \in p$

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_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

LR(0)-Parser

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

We identify two reasons:

Reduce-Reduce-Conflict:

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

Shift-Reduce-Conflict:

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

for a state $q \in Q$.

Those states are called $LR(0)$-unsuited.

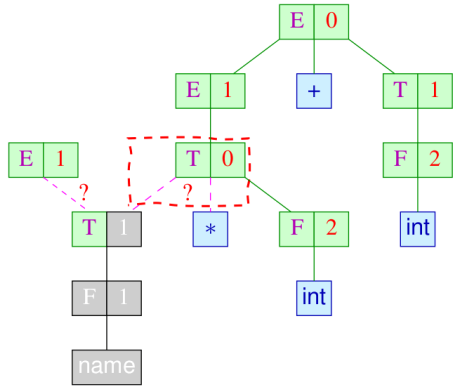
Revisiting the Conflicts of the LR(0)-Automaton

Idea: Matching lookahead with *right context* matters!

Input:

Pushdown:

(q_0 T)



LR(k)-Grammars

Idea: Consider k-lookahead in conflict situations.

Definition:
The reduced contextfree grammar G is called $LR(k)$-grammar, if for $\text{First}_{|\alpha\beta|+k}(\alpha\,\beta\,w) = \text{First}_{|\alpha\beta|+k}(\alpha'\,\beta'\,w')$ with:

$$\left.\begin{array}{llll} S & \to_R^* & \alpha\,A\,w & \to \quad \alpha\,\beta\,w \\ S & \to_R^* & \alpha'\,A'\,w' & \to \quad \alpha'\,\beta'\,w' \end{array}\right\} \text{follows: } \alpha = \alpha' \wedge \beta = \beta' \wedge A = A'$$

Strategy for testing Grammars for $LR(k)$-property

1. Focus iteratively on all rightmost derivations $S \to_R^* \alpha\,X\,w \to \alpha\,\beta\,w$
2. Iterate over $k \geq 0$
   1. For each $\gamma = \text{First}_{|\alpha\beta|+k}(\alpha\,\beta w)$ check if there exists a differently right-derivable $\alpha'\beta'w'$ for which $\gamma = \text{First}_{|\alpha\beta|+k}(\alpha'\,\beta' w')$
   2. if there is none, we have found no objection against k, being enough lookahead to disambiguate $\alpha\beta w$ from other rightmost derivations

LR(k)-Grammars

for example:

(1) $S \to A \mid B \qquad A \to a\,A\,b \mid 0 \qquad B \to a\,B\,b\,b \mid 1$

$a\,b^{n+1}\,b\,b^{n+1}\,c$