Script generated by TTT

Title: Simon: Compilerbau (16.05.2012)

Date: Wed May 16 14:13:12 CEST 2012

Duration: 79:30 min

Pages: 34

Kapitel 4:
Bottom-up Analyse

Bottom-up Analyse

Achtung:
Viele Grammatiken sind nicht $LL(k)$!

Eine Grund dafür ist:

Definition
Die Grammatik G heißt links-rekursiv, falls

$$A \to^+ A\,\beta \qquad \text{für ein} \quad A \in N,\ \beta \in (T \cup N)^*$$

Bottom-up Analyse

Achtung:
Viele Grammatiken sind nicht $LL(k)$!

Eine Grund dafür ist:

Definition
Die Grammatik G heißt links-rekursiv, falls

$$A \to^+ A\,\beta \qquad \text{für ein} \quad A \in N,\ \beta \in (T \cup N)^*$$

Beispiel:

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

... ist links-rekursiv

Bottom-up Analyse

Satz:

Ist die Grammatik G reduziert und links-rekursiv, dann ist G nicht $LL(k)$ für jedes k.

Beweis:

Sei $A \to A\,\beta \mid \alpha \quad \in P$
und A erreichbar von S

Annahme: G ist $LL(k)$

Bottom-up Analyse

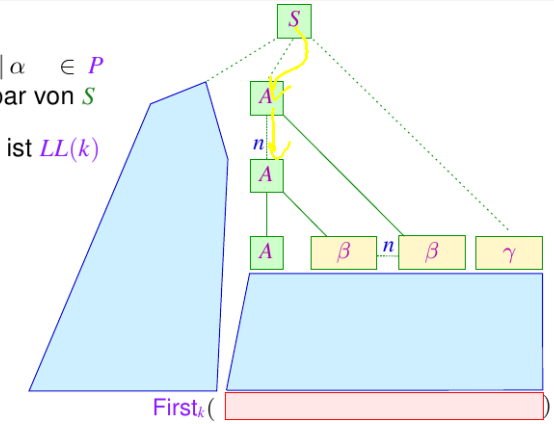
Satz:

Ist die Grammatik G reduziert und links-rekursiv, dann ist G nicht $LL(k)$ für jedes k.

Beweis:

Sei $A \to A\,\beta \mid \alpha \quad \in P$
und A erreichbar von S

Annahme: G ist $LL(k)$



Bottom-up Analyse

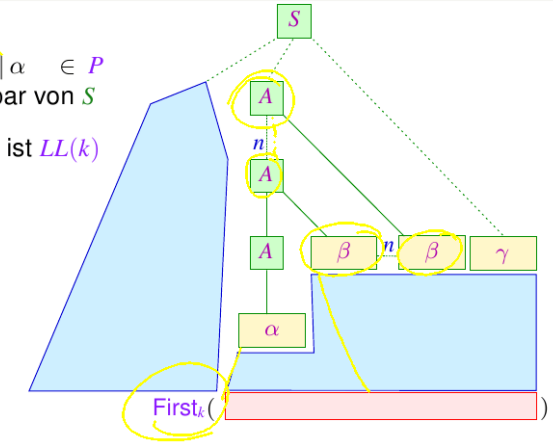
Satz:

Ist die Grammatik G reduziert und links-rekursiv, dann ist G nicht $LL(k)$ für jedes k.

Beweis:

Sei $A \to A\,\beta \mid \alpha \quad \in P$
und A erreichbar von S

Annahme: G ist $LL(k)$



Bottom-up Analyse

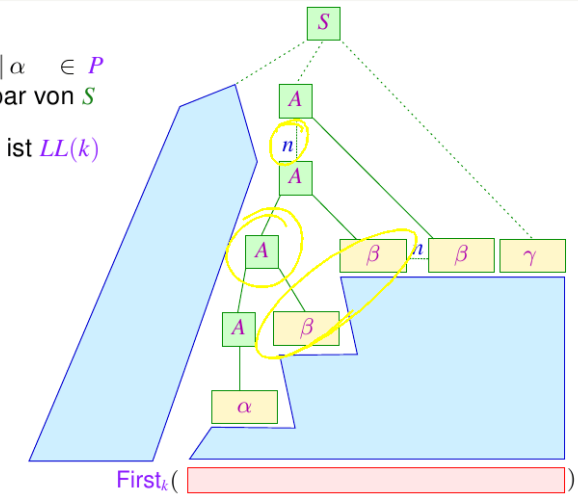
Satz:

Ist die Grammatik G reduziert und links-rekursiv, dann ist G nicht $LL(k)$ für jedes k.

Beweis:

Sei $A \to A\,\beta \mid \alpha \quad \in P$
und A erreichbar von S

Annahme: G ist $LL(k)$



Bottom-up Analyse

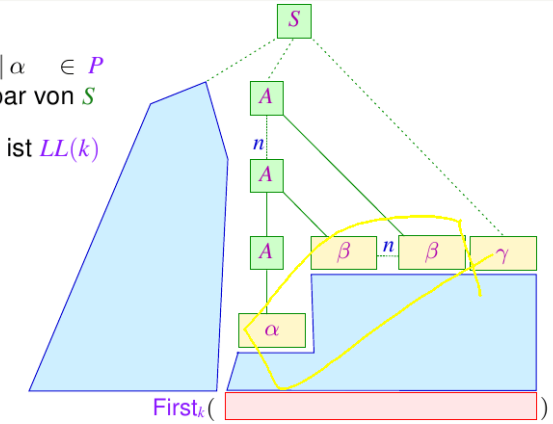
Satz:

Ist die Grammatik G reduziert und links-rekursiv, dann ist G nicht $LL(k)$ für jedes k.

Beweis:

Sei $A \to A\,\beta \mid \alpha \quad \in P$
und A erreichbar von S

Annahme: G ist $LL(k)$



Bottom-up Analyse

Satz:

Ist die Grammatik G reduziert und links-rekursiv, dann ist G nicht $LL(k)$ für jedes k.

Beweis:

Sei $A \to A\,\beta \mid \alpha \quad \in P$
und A erreichbar von S

Annahme: G ist $LL(k)$

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

Bottom-up Analyse

Satz:

Ist die Grammatik G reduziert und links-rekursiv, dann ist G nicht $LL(k)$ für jedes k.

Beweis:

Sei $A \to A\,\beta \mid \alpha \quad \in P$
und A erreichbar von S

Annahme: G ist $LL(k)$

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

Fall 1: $\beta \to^* \epsilon$ — Widerspruch !!!

Fall 2: $\beta \to^* w \neq \epsilon \Longrightarrow \text{First}_k(\alpha\,\beta^k\,\gamma) \cap \text{First}_k(\alpha\,\beta^{k+1}\,\gamma) \neq \emptyset$

Shift-Reduce-Parser

Donald Knuth

Idee:

Wir verzögern die Entscheidung ob wir reduzieren bis wir wissen, ob die Eingabe einer rechten Seite entspricht!

Konstruktion: Shift-Reduce-Parser M_G^R

- Die Eingabe wird sukzessive auf den Keller geschoben.
- Liegt oben auf dem Keller eine vollständige rechte Seite (ein Handle) vor, wird dieses durch die zugehörige linke Seite ersetzt (reduziert)

Shift-Reduce-Parser

Beispiel:

$$\begin{array}{lcl} S & \to & A\,B \\ A & \to & a \\ B & \to & b \end{array}$$

Der Kellerautomat:

Zustände: $q_0, f, a, b, A, B, S;$
Anfangszustand: q_0
Endzustand: f

q_0	a	$q_0\,a$
a	ϵ	A
A	b	$A\,b$
b	ϵ	B
$A\,B$	ϵ	S
$q_0\,S$	ϵ	f

Shift-Reduce-Parser

Konstruktion: Allgemein konstruieren wir einen Automaten $M_G^R = (Q, T, \delta, q_0, F)$ mit:

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

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

Shift-Reduce-Parser

Konstruktion: Allgemein konstruieren wir einen Automaten $M_G^R = (Q, T, \delta, q_0, F)$ mit:

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

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

Beispiel-Berechnung:

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

Shift-Reduce-Parser

Offenbar gilt:

- Die Folge der Reduktionen entspricht einer reversen Rechtsableitung für die Eingabe
- Zur Korrektheit zeigt man, dass gilt:

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

- Der Shift-Reduce-Kellerautomat M_G^R ist i.a. auch nicht-deterministisch
- Um ein deterministisches Parse-Verfahren zu erhalten, muss man die Reduktionsstellen identifizieren

$\Longrightarrow$ LR-Parsing

Bottom-up Analyse

Idee: Wir rekonstruieren reverse Rechtsableitungen!

Dazu versuchen wir, für den Shift-Reduce-Parser M_G^R die Reduktionsstellen zu identifizieren ...

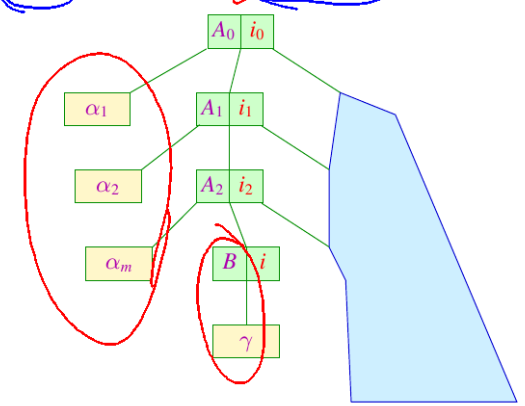
Betrachte eine Berechnung dieses Kellerautomaten:

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

$\alpha\,\gamma$ nennen wir zuverlässiges Präfix für das vollständige Item $[B \to \gamma\bullet]$.

Bottom-up Analyse

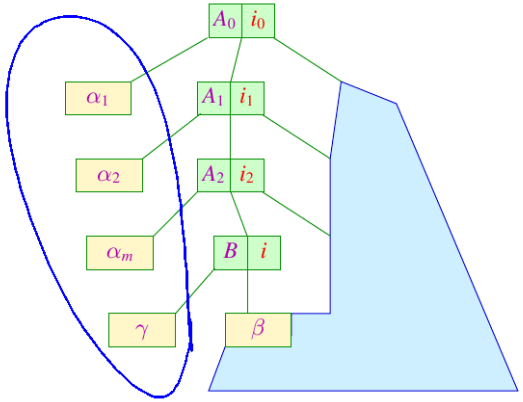
Dann ist $\alpha\,\gamma$ zuverlässig für $[B \to \gamma\bullet]$ gdw. $S \to_R^* \alpha\,B\,v$



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

Bottom-up Analyse

Das Item $[B \to \gamma \bullet \beta]$ heißt gültig für α' gdw. $S \to_R^* \alpha\,B\,v$ mit $\alpha' = \alpha\,\gamma$:



... wobei $\alpha = \alpha_1 \ldots \alpha_m$:-)

Charakteristischer Automat

Beobachtung:

Die Menge der zuverlässigen Präfixe aus $(N \cup T)^*$ für (vollständige) Items kann mithilfe eines endlichen Automaten, aus dem Kellerinhalt des Shift-Reduce-Parsers berechnet werden:

Zustände: Items

Anfangszustand: $[S' \to \bullet\, S]$

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

Übergänge:

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

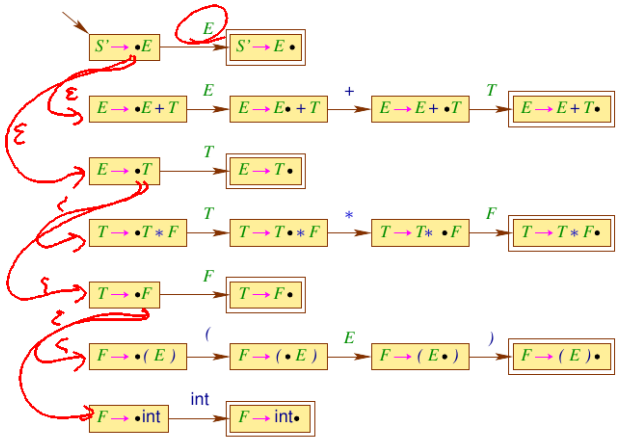
(2) $([A \to \alpha \bullet B\,\beta], \epsilon, [B \to \bullet\,\gamma])$, $A \to \alpha\,B\,\beta$, $B \to \gamma \in P$;

Den Automaten $c(G)$ nennen wir charakteristischen Automaten für G.

Charakteristischer Automat

im Beispiel:

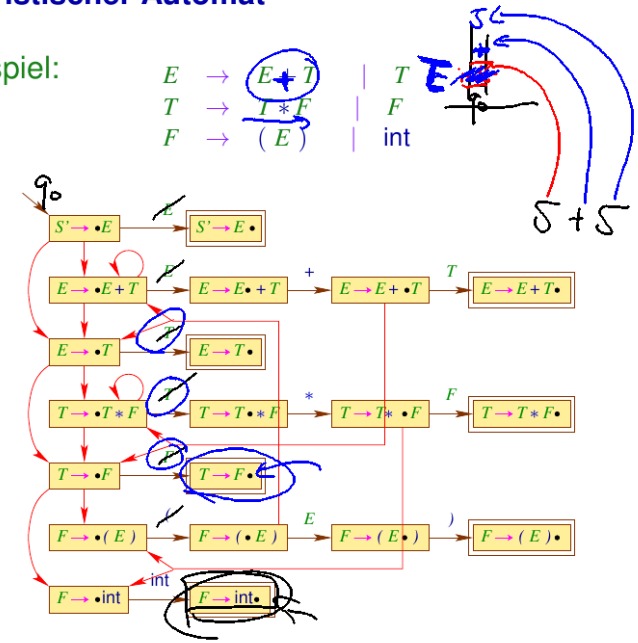
$$
\begin{array}{lcll}
E & \to & E+T & \mid T \\
T & \to & T*F & \mid F \\
F & \to & (\,E\,) & \mid \text{int}
\end{array}
$$



Charakteristischer Automat

im Beispiel:

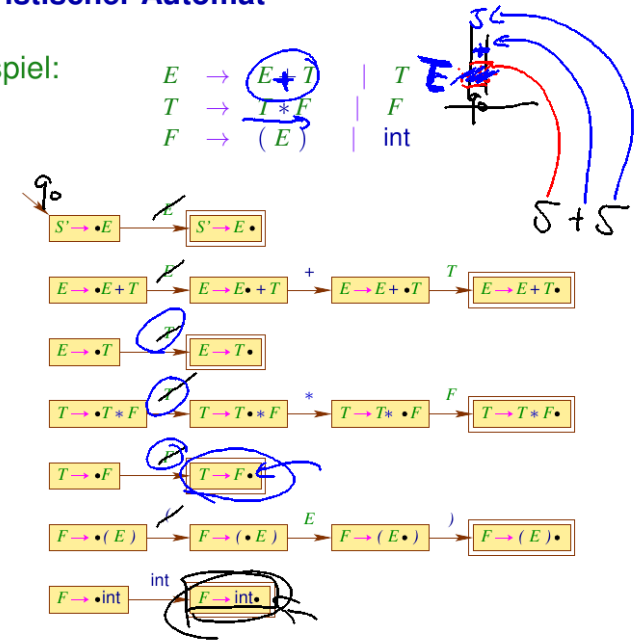
$$
\begin{array}{lcll}
E & \to & E+T & \mid T \\
T & \to & T*F & \mid F \\
F & \to & (\,E\,) & \mid \text{int}
\end{array}
$$



Charakteristischer Automat

im Beispiel:

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

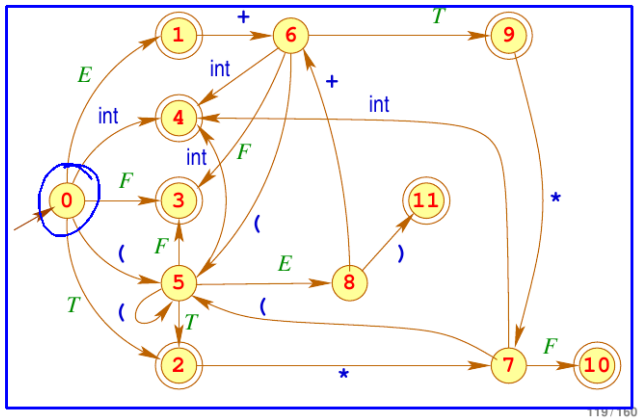


Kanonischer LR(0)-Automat

Den kanonischen $LR(0)$-Automaten $LR(G)$ erhalten wir aus $c(G)$, indem wir:

1. nach jedem lesenden Übergang beliebig viele ϵ-Übergänge einschieben
2. die Teilmengenkonstruktion anwenden.

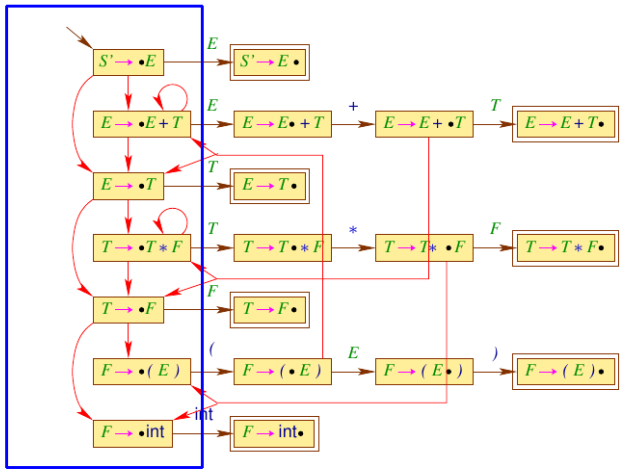
... im Beispiel:



Charakteristischer Automat

im Beispiel:

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



Kanonischer LR(0)-Automat

Dazu konstruieren wir:

$q_0 = \{[S' \to \bullet E],\ [E \to \bullet E + T],\ [E \to \bullet T],\ [T \to \bullet T * F],\ [T \to \bullet F],\ [F \to \bullet (\ E\ )],\ [F \to \bullet \text{int}]\}$

$q_1 = \delta(q_0, E) = \{[S' \to E\bullet],\ [E \to E \bullet + T]\}$

$q_2 = \delta(q_0, T) = \{[E \to T\bullet],\ [T \to T \bullet * F]\}$

$q_3 = \delta(q_0, F) = \{[T \to F\bullet]\}$

$q_4 = \delta(q_0, \text{int}) = \{[F \to \text{int}\bullet]\}$

Kanonischer LR(0)-Automat

$q_5 = \delta(q_0, (\,)) = \{[F \to (\ \bullet E\ )],\ [E \to \bullet E + T],\ [E \to \bullet T],\ [T \to \bullet T * F],\ [T \to \bullet F],\ [F \to \bullet (\ E\ )],\ [F \to \bullet \text{int}]\}$

$q_6 = \delta(q_1, +) = \{[E \to E + \bullet T],\ [T \to \bullet T * F],\ [T \to \bullet F],\ [F \to \bullet (\ E\ )],\ [F \to \bullet \text{int}]\}$

$q_7 = \delta(q_2, *) = \{[T \to T * \bullet F],\ [F \to \bullet (\ E\ )],\ [F \to \bullet \text{int}]\}$

$q_8 = \delta(q_5, E) = \{[F \to (\ E \bullet\ )],\ [E \to E \bullet + T]\}$

$q_9 = \delta(q_6, T) = \{[E \to E + T\bullet],\ [T \to T \bullet * F]\}$

$q_{10} = \delta(q_7, F) = \{[T \to T * F\bullet]\}$

$q_{11} = \delta(q_8,)) = \{[F \to (\ E\ )\bullet]\}$

Kanonischer LR(0)-Automat

Beachte:

Der kanonische $LR(0)$-Automat kann auch direkt aus der Grammatik konstruiert werden.
Man benötigt die Hilfsfunktion: δ^*_ϵ (ϵ-Abschluss)

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

Dann definiert man:

Zustände: Mengen von Items;

Anfangszustand $\delta^*_\epsilon\{[S' \to \bullet S]\}$

Endzustände: $\{q \mid \exists A \to \alpha \in P :\ [A \to \alpha\bullet] \in q\}$

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

Kanonischer LR(0)-Automat

Dazu konstruieren wir:

$$
\begin{array}{lcl}
q_0 & = & \{[S' \to \bullet\, E], \\
 & & \ [E \to \bullet\, E + T], \\
 & & \ [E \to \bullet\, T], \\
 & & \ [T \to \bullet\, T * F]\} \\
 & & \ [T \to \bullet\, F], \\
 & & \ [F \to \bullet\, (\, E\,)\,], \\
 & & \ [F \to \bullet\, \mathsf{int}]\}
\end{array}
\qquad
\begin{array}{lcl}
q_1 & = & \delta(q_0, E) \\
 & & \\
q_2 & = & \delta(q_0, T) \\
 & & \\
q_3 & = & \delta(q_0, F) \\
 & & \\
q_4 & = & \delta(q_0, \mathsf{int})
\end{array}
\qquad
\begin{array}{l}
= \{[S' \to E\bullet], \\
\quad [E \to E \bullet + T]\} \\
= \{[E \to T\bullet], \\
\quad [T \to T \bullet * F]\} \\
= \{[T \to F\bullet]\} \\
= \{[F \to \mathsf{int}\bullet]\}
\end{array}
$$

Kanonischer LR(0)-Automat

Beachte:

Der kanonische $LR(0)$-Automat kann auch direkt aus der Grammatik konstruiert werden.
Man benötigt die Hilfsfunktion: δ^*_ϵ (ϵ-Abschluss)

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

Dann definiert man:

Zustände: Mengen von Items;
Anfangszustand $\delta^*_\epsilon\,\{[S' \to \bullet\, S]\}$
Endzustände: $\{q \mid \exists A \to \alpha \in P : \ [A \to \alpha\bullet] \in q\}$
Übergänge: $\delta(q, X) = \delta^*_\epsilon\{[A \to \alpha\, X \bullet \beta] \mid [A \to \alpha \bullet X\, \beta] \in q\}$

LR(0)-Parser

Idee zu einem Parser:

- Der Parser verwaltet ein zuverlässiges Präfix $\alpha = X_1 \ldots X_m$ auf dem Keller und benutzt $LR(G)$, um Reduktionsstellen zu entdecken.
- Er kann mit einer Regel $A \to \gamma$ reduzieren, falls $[A \to \gamma\bullet]$ für α gültig ist

Optimierung:

Damit der Automat nicht immer wieder neu über den Kellerinhalt laufen muss, kellern wir anstelle der X_i jeweils die Zustände!
Reduktion mit $A \to \gamma$ führt zum Poppen der obersten $|\gamma|$ Zustände und Fortsetzung mit dem Zustand auf dem Keller unter Eingabe A.

Achtung:

Dieser Parser ist nur dann deterministisch, wenn jeder Endzustand des kanonischen $LR(0)$-Automaten keine Konflikte enthält.

LR(0)-Parser

... im Beispiel:

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

Die Endzustände q_1, q_2, q_9 enthalten mehr als ein gültiges Item
$\Rightarrow$ nicht deterministisch!

LR(0)-Parser

Die Konstruktion des $LR(0)$-Parsers:

Zustände: $Q \cup \{f\}$ (f neu)

Anfangszustand: q_0

Endzustand: f

Übergänge:

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

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

LR(0)-Parser

Achtung:

Leider ist der $LR(0)$-Parser im allgemeinen nicht-deterministisch

Wir identifizieren zwei Gründe:

Reduce-Reduce-Konflikt:

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

Shift-Reduce-Konflikt:

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

für einen Zustand $q \in Q$.

Solche Zustände nennen wir $LR(0)$-ungeeignet.

LR(k)-Grammatik

Idee: Benutze k-Vorausschau, um Konflikte zu lösen.

Definition:

Die reduzierte kontextfreie Grammatik G heißt $LR(k)$-Grammatik, falls für $\text{First}_k(w) = \text{First}_k(x)$ aus:

$$\left.\begin{array}{lllll} S & \to_R^* & \alpha A w & \to & \alpha\beta w \\ S & \to_R^* & \alpha' A' w' & \to & \alpha\beta x \end{array}\right\} \text{ folgt: } \alpha = \alpha' \wedge A = A' \wedge w' = x$$