Proseminar Syntaktische Analyse

Universität des Saarlandes

Wintersemester 1999/2000

Dozenten:
Reinhard Wilhelm
Andreas Kerren

LR(k)-Syntaxanalyse, Teil 2

Stephan Wilhelm

25. April 2000

Inhaltsverzeichnis

1 Motivation **1**

1.1 Größe des LR(k)-Parsers . . . 1

1.2 Beispiel . . . 1

1.3 Warum wachsen LR(k)-Parser mit k? . . . 3

2 LALR(k) **3**

2.1 Konstruktion aus dem LR(k)-Automaten . . . 3

2.2 LALR(k)-Parser . . . 4

2.3 LALR(k)-Grammatiken . . . 5

2.4 Beispiel . . . 6

3 SLR(k) **6**

3.1 Konstruktion aus dem LR(0)-Automaten . . . 6

3.2 Beispiel . . . 7

3.3 SLR(k)-Parser . . . 7

3.4 SLR(k)-Grammatiken . . . 8

3.5 Beispiel: Fehlererkennung . . . 10

4 Vergleich der Verfahren **11**

4.1 Unterschiede zwischen LALR(k)- und SLR(k)-Parsern . . . 11

4.2 Ergebnisse . . . 12

5 Transformation von Grammatiken **12**

5.1 Motivation . . . 12

5.2 Transformation von LR(k)- in SLR(k)-Grammatiken . . . 12

5.2.1 Vorgehensweise . . . 13

5.2.2 Ergebnisse . . . 13

5.3 Transformation von LR(k)- in LR(1)-Grammatiken . . . 14

5.3.1 Vorgehensweise . . . 14

5.4 Schlußfolgerungen . . . 15

Zusammenfassung

Diese Ausarbeitung entstand im Rahmen des Proseminars "Syntaktische Analyse von Programmiersprachen" im Wintersemester 1999/2000 am Lehrstuhl von Prof. Wilhelm an der Universität des Saarlandes.

Die folgenden Ausführungen schließen an die Seminarausarbeitung "LR(k)-Syntaxanalyse" von Claudia Bieg an, in welcher der kanonische LR(k)-Parser eingeführt wurde. Im Folgenden werden zuerst zwei Vereinfachungen des LR-Konzepts besprochen, nämlich die LALR(k)- und die SLR(k)-Syntaxanalyse. Im Anschluß daran folgt eine Abhandlung über Grammatiktransformationen, mit deren Hilfe die Äquivalenz der beiden Konzepte zur kanonischen LR(k)-Syntaxanalyse gezeigt wird.

1 Motivation

1.1 Größe des LR(k)-Parsers

Wie aus dem vorangegangenen Artikel bereits bekannt ist, ist der kanonische LR(k)-Parser das mächtigste deterministische Verfahren zur syntaktischen Analyse von kontextfreien Grammatiken. In der Praxis stellt sich jedoch heraus, daß dieses Verfahren für die Grammatiken üblicher Programmiersprachen sehr große Aktionstabellen erzeugt. Den Grund hierfür erläutert der folgende Satz.

Satz 1

Die Größe des kanonischen LR(k)-Parsers für eine Grammatik $G = (V, T, P, S)$ ist

$$O(2^{|T|^k |G| + k \log |T| + \log |G|})$$

Beweisskizze

Die Anzahl der Item-Kerne der Form $A \rightarrow \alpha \cdot \beta$ in G ist maximal $|G|$ und die Anzahl der Lookahead-Strings in $k : T^*$ höchstens $(|T| + 1)^k$. Damit ist die Anzahl der k-Items von G höchstens $(|T| + 1)^k |G|$.

Für die \$-erweiterte Grammatik G' folgt dann, daß die Anzahl der LR(k)-Äquivalenzklassen $[\delta]_k$ in $[G']_k$ durch $2^{(|T|+1)^k} |G'|$ beschränkt ist.

Damit ist die Summe über die Länge aller Reduktionen (ra) der Form

$$[\delta]_k [\delta X_1]_k \ldots [\delta X_1 \ldots X_n]_k | y \rightarrow [\delta]_k [\delta A]_k | y$$

gleich

$$O(2^{(|T|+1)^k |G'|} |G| (|T| + 1)^k) \ .$$

Damit ist obiger Satz bewiesen. ■

Man sieht an diesem Term, daß die Größe von G insbesondere von der Länge des Lookaheads k abhängig ist, da dieser doppelt im Exponenten steht.

1.2 Beispiel

Gegeben ist die Grammatik $G_{ab\epsilon}$ mit den Regeln $S \rightarrow aA|bB, A \rightarrow \epsilon|cAd, B \rightarrow, \epsilon|cBd$.

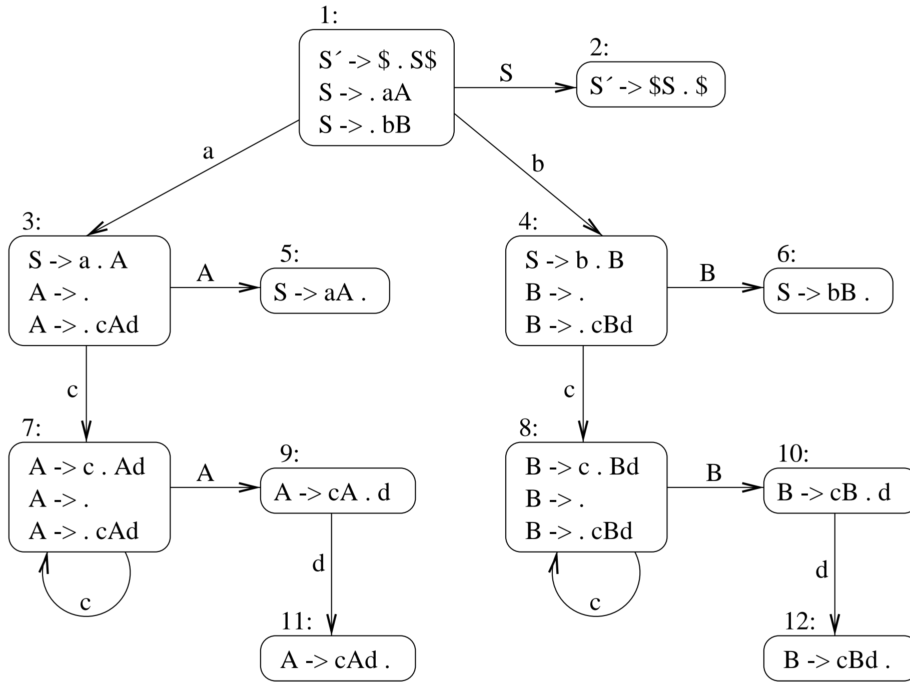


Abbildung 1: LR(0)-Automat für $G_{ab\epsilon}$

Für beliebiges $k, n \geq 0$ sehen die Klassen gültiger Items wie folgt aus:

$$
\begin{array}{lll}
\text{VALID}_k(\$ac^{n+1}) & = \{ & [A \to c \cdot Ad, k : d^n\$], [A \to \cdot, k : d^{n+1}\$], \\
 & & [A \to \cdot cAd, k : d^{n+1}\$] \}, \\
\text{VALID}_k(\$ac^{n+1}A) & = \{ & [A \to cA \cdot d, k : d^n\$] \}, \\
\text{VALID}_k(\$ac^{n+1}Ad) & = \{ & [A \to cAd \cdot, k : d^n\$] \}, \\
\text{VALID}_k(\$bc^{n+1}) & = \{ & [B \to c \cdot Bd, k : d^n\$], [B \to \cdot, k : d^{n+1}\$], \\
 & & [B \to \cdot cBd, k : d^{n+1}\$] \}, \\
\text{VALID}_k(\$bc^{n+1}B) & = \{ & [B \to cB \cdot d, k : d^n\$] \}, \\
\text{VALID}_k(\$bc^{n+1}Bd) & = \{ & [B \to cBd \cdot, k : d^n\$] \},
\end{array}
$$

Für festes k sind diese Zustände unterschiedlich, genau dann wenn $n < k$.
Für alle $n \geq k$ gilt dann:

$$
\begin{array}{lll}
\text{VALID}_k(\$ac^{n+1}) & = & \text{VALID}_k(\$ac^{k+1}) \\
\text{VALID}_k(\$ac^{n+1}A) & = & \text{VALID}_k(\$ac^{k+1}A) \\
\text{VALID}_k(\$ac^{n+1}Ad) & = & \text{VALID}_k(\$ac^{k+1}Ad) \\
\text{VALID}_k(\$bc^{n+1}) & = & \text{VALID}_k(\$bc^{k+1}) \\
\text{VALID}_k(\$bc^{n+1}B) & = & \text{VALID}_k(\$bc^{k+1}B) \\
\text{VALID}_k(\$bc^{n+1}Bd) & = & \text{VALID}_k(\$bc^{k+1}Bd)
\end{array}
$$

Schließlich kommen noch die von n unabhängigen Zustände hinzu:

$$
\begin{array}{ll}
\text{VALID}_k(\$), & \text{VALID}_k(\$S), \\
\text{VALID}_k(\$a), & \text{VALID}_k(\$aA), \\
\text{VALID}_k(\$b), & \text{VALID}_k(\$bB).
\end{array}
$$

Damit beträgt die Anzahl der Zustände der kanonischen LR(k)-Maschine insgesamt

$$6(k+1)+6=6k+12\ .$$

In diesem Beispiel wächst die LR(k)-Maschine linear in k. Damit wächst auch die Anzahl der Aktionen des LR(k)-Parsers mit größerem Lookahead k. Für andere Grammatiken kann die Zunahme der Aktionen exponentiell in k werden.

1.3 Warum wachsen LR(k)-Parser mit k?

Um sich den Grund für die Zunahme der Aktionen mit k zu veranschaulichen, muß man sich an die Konstruktion des kanonischen LR(k)-Parsers erinnern. Bei einem Lookahead von 0 entsprechen die Zustände des LR(0)-Automaten den Äquivalenzklassen $[\gamma]_0$ für G. Ein Lookahead größer 0 führt nun zur Partitionierung dieser Äquivalenzklasse $[\gamma]_0$ in eine oder mehrere LR(k)-Äquivalenzklassen $[\gamma_1]_k, \ldots, [\gamma_n]_k$.

2 LALR(k)

Aus den vorangegangenen Ausführungen ergibt sich, daß man die Anzahl der Aktionen eines LR(k)-Parsers dadurch verringern kann, daß man die Anzahl der LR(k)-Äquivalenzklassen verringert. Um dies zu erreichen konstruiert man aus einem gegebenen LR(k)-Automaten den sogenannten *Lookahead LR(k)-Automaten*. Der folgende Abschnitt erläutert einen Algorithmus, der es gestattet, Äquivalenzklassen von G unter gewissen Umständen zu vereinigen.

2.1 Konstruktion aus dem LR(k)-Automaten

1. Ausgangspunkt ist ein bereits konstruierter LR(k)-Automat für eine Grammatik G.
2. In diesem Automaten werden nun alle Zustände $q_1, \ldots, q_n$ mit der gleichen Menge von Item-Kernen zu einem neuen Zustand q' vereinigt und dem Automaten hinzugefügt.
3. Die Zustände, die vereinigt wurden, werden aus dem Automaten entfernt.

Abbildung 2 veranschaulicht die Vorgehensweise an einem Beispiel. Abbildung 8 enthält die vollständigen Automaten (siehe letzte Seite).

Der somit erzeugte Automat heißt *Lookahead LR(k)-Automat*, falls der gegebene LR(k)-Automat konfliktfrei ist und durch die Vereinigung von Zuständen keine neuen Konflikte entstehen. Mit dem Algorithmus zur Erzeugung des LR(k)-Parsers aus einem LR(k)-Automaten kann aus dem LALR(k)-Automaten ein LALR(k)-Parser erzeugt werden. Aufgrund der Konstruktion ist der LALR(k)-Automat *isomorph* zum LR(0)-Automaten.

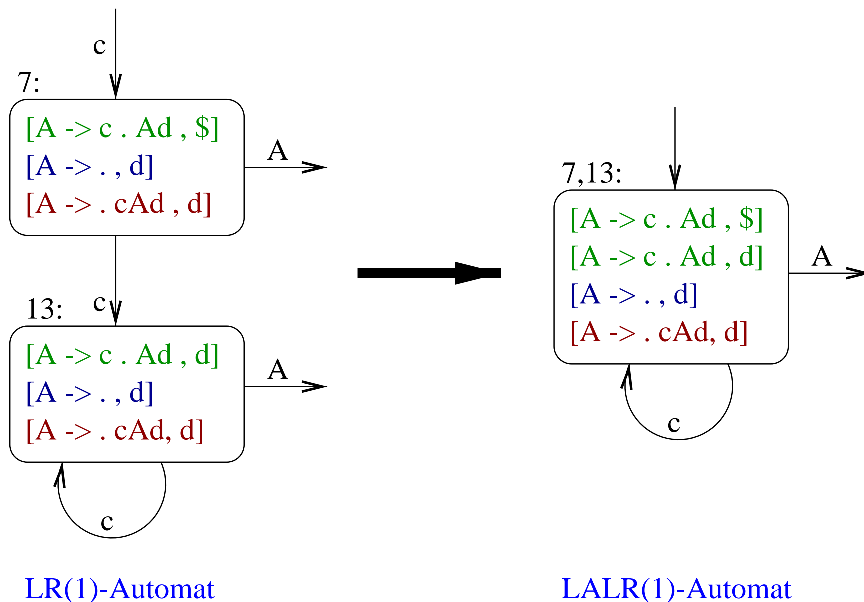


Abbildung 2: Konstruktion des LALR(1)-Automat aus LR(1) (Ausschnitt).

Es bleibt anzumerken, daß das hier vorgestellte Verfahren ziemlich *ineffizent* ist. Ein effizienteres Verfahren ist die Konstruktion aus dem LR(0)-Automaten für G. Dabei sucht man geeignete Lookahead-Strings der Länge k für die 0-Items aus dem LR(0)-Automaten. Diese Vorgehensweise kann analog zur Konstruktion des SLR(k)-Automaten beschrieben werden. Eine formale Beschreibung findet sich in Abschnitt 4.

Die Größe des LALR(k)-Parsers liegt in $O(2^{|G|+klog|T|+log|G|})$. Damit ist der LALR($k$)-Parser deutlich kleiner als der korrespondierende LR(k)-Parser (Beweis analog zu LR(k)).

2.2 LALR(k)-Parser

Definition

Sei $G = (V, T, P, S)$ eine Grammatik und G' die \$-erweiterte Grammatik für G, $k \in N$.

Der LALR(k)-Parser für G ist dann ein Kellerautomat mit Ausgabe mit Kelleralphabet $[G']_0$, Eingabealalphabet T, Startinhalt $[\$]_0$ des Kellers, Menge der Endinhalte $\{[\$]_0\ [\$S]_0\}$ des Kellers, Menge der Aktionen bestehend aus allen LALR(k) reduce- und shift-Aktionen für G und Ausgabeeffekt τ, wobei τ wie üblich alle reduce-Aktionen durch Regel r nach r und alle shift Aktionen nach ϵ überführt.

Definition: LALR(k)-Aktionen für G

Eine Regel der Form

$$(ra) \qquad [\delta]_0[\delta X_1]_0 \dots [\delta X_1 \dots X_n]_0 | y \rightarrow [\delta]_0[\delta A]_0 | y$$

heißt LALR(k) *reduce action* durch die Vorschrift $A \rightarrow X_1 \dots X_n$ mit Lookahead y, falls $\delta \in \$V^*$, $X_1, \dots, X_n$ Symbole aus $V (n \geq 0)$, $A \rightarrow X_1 \dots X_n$ eine Regel aus P und $y \in k : T^*\$$, so daß $[A \rightarrow X_1 \dots X_n \cdot, y] \in \text{VALID}_k(\gamma)$ und $[\gamma]_0 = [\delta X_1 \dots X_n]_0$ für $\gamma \in \$V^*$.

Das heißt $[A \rightarrow X_1 \dots X_n \cdot, y]$ ist LR(k)-gültig für ein zuverlässiges Präfix, das LR(0)-äquivalent zu $\delta X_1 \dots X_n$ ist.

Eine Regel der Form

$$(sa) \qquad [\delta]_0 | ay \rightarrow [\delta]_0[\delta a]_0 | y$$

heißt LALR(k) *shift action* auf einem Terminal a und Lookahead ay, falls $\delta \in \$V^*$, a ein Terminal $\in T$ und $y \in max\{k-1, 0\} : T^*\$$, so daß $[A \rightarrow \alpha \cdot a\beta, z] \in \text{VALID}_k(\gamma)$, $[\gamma]_0 = [\delta]_0$, und $y \in FIRST_{max\{k-1,0\}}(\beta z)$, für Regeln $A \rightarrow \alpha a \beta$, $z \in k : T^*\$$ und $\gamma \in \$V^*$.

Das heißt, für ein zuverlässiges Präfix γ, das LR(0)-äquivalent zu δ ist, enthält $\text{VALID}_k(\gamma)$ ein Item der Form $[A \rightarrow \alpha \cdot a\beta, z]$.

2.3 LALR(k)-Grammatiken

Definition

Eine Grammatik $G = (V, T, P, S)$ ist LALR(k) genau dann, wenn ihr LALR(k)-Parser deterministisch ist und $S \Rightarrow^+ S$ unmöglich in G.

Satz 2: Determinismus des LALR(k)-Parsers

Der LALR(k)-Parser einer Grammatik G ist deterministisch genau dann wenn G deterministisch ist und kein reduce-reduce- oder shift-reduce-Konflikt für zwei Items in einem Zustand des LALR(k)-Automaten für G auftritt.

Beweis analog zum LR(k)-Parser.

Satz 3: Klasse der LALR(k)-Grammatiken

Die Klasse der LALR(0)-Grammatiken stimmt mit der Klasse der LR(0)-Grammatiken überein. Für $k \geq 0$ ist die Klasse der LALR(k)-Grammatiken in der Klasse der LR(k)-Grammatiken enthalten.

Beweisskizze

$k = 0$:
Der Beweis folgt direkt aus der Konstruktion des LALR(k)-Automaten aus dem LR(k)-Automaten (Äquivalenz der Automaten). Analoges gilt für die Konstruktion aus dem LR(0)-Automaten.

$k > 0$:
Da durch die Vereinigung von Zuständen die Zahl von Item-Paaren, die einen Konflikt zeigen nur vergrößert werden kann, folgt daß die Klasse der LALR(k)-Grammatiken in der Klasse der LR(k)-Grammatiken enthalten ist. Andererseits kann die Vereinigung von Zuständen die Entstehung neuer Konflikte bewirken, die im ursprünglichen Automaten nicht vorhanden waren. Daraus folgt, daß nicht alle LR(k)-Grammatiken auch LALR(k)-Grammatiken sind. ■

2.4 Beispiel

Der LR(1)-Automat für die Grammatik $S \to aAa|bAb|aBb|bBa$, $A \to c$, $B \to c$ besitzt unter anderem die beiden in Abbildung 3 dargestellten kerngleichen Zustände:

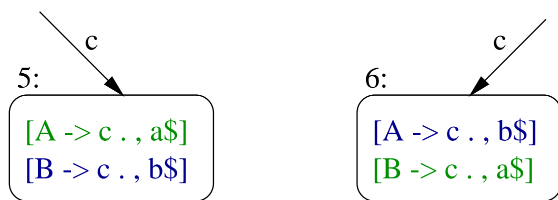


Abbildung 3: Ausschnitt aus dem LR(1)-Automaten für obige Grammatik

Bei der Konstruktion des LALR(1)-Parsers werden nun die Zustände q_5 und q_6 zu einem neuen Zustand $q_{5,6}$ vereinigt. Aus diesem Zustand kann der Parser aber nicht mehr eindeutig reduzieren. Unter den Lookahead-Symbolen a und b sind jeweils Reduktionen nach A, bzw. B möglich. Damit enthält dieser Zustand nun zwei reduce-reduce-Konflikte, die in den beiden ursprünglichen Zuständen nicht vorhanden waren. Somit ist diese Grammatik zwar eine LR(1)-Grammatik, aber keine LALR(1)-Grammatik.

3 SLR(k)

3.1 Konstruktion aus dem LR(0)-Automaten

1. Ausgangspunkt ist ein bereits konstruierter LR(0)-Automat für G.

2. In diesem ersetzt man in jedem Zustand q alle Items $[A \to \alpha \cdot \beta]$ durch die folgende Menge von k-Items:
$$[A \to \alpha \cdot \beta, y_1], \ldots, [A \to \alpha \cdot \beta, y_n]\,,$$
wobei $\{y_1, \ldots, y_n\} = FOLLOW_k(A)$.

Der durch diese Konstruktion entstehende Automat heißt *Simple LR(k)-Automat.* Seine Menge von Lookahead-Strings ist unabhängig vom Kontext (Zustand), da einfach alle möglichen Lookaheads hinzugefügt werden. Dies entspricht gerade $FOLLOW_k(A)$ für jeden Zustand A im Automaten. Da die Menge der Lookahead-Strings des SLR(k)-Parsers maximal ist, ist sie auch eine Obermenge der Menge der Lookahead-Strings des LALR(k)-Parsers. Desweiteren kann auch beim SLR(k)-Automaten der Algorithmus zur Erzeugung des LR(k)-Parsers verwendet werden, um einen SLR(k)-Parser zur erzeugen.

3.2 Beispiel

Betrachte die in Abbildung 4 dargestellten SLR(1)-Parser und LALR(1)-Parser für die Grammatik $G_{ab\epsilon}$.

Für SLR(1) gilt:

$$\begin{array}{lclcl} FOLLOW_1(S) & = & \{\$\} & & \\ FOLLOW_1(A) & = & \{\$, d\} & = & FOLLOW_1(B). \end{array}$$

Aus der Konstruktion der Aktionstabelle ergibt sich, daß der SLR(1)-Parser für diese Grammatik 4 Aktionen mehr besitzt als der korrespondierende LALR(1)-Parser.

3.3 SLR(k)-Parser

Definition

Sei $G = (V, T, P, S)$ eine Grammatik und G' die \$-erweiterte Grammatik für G und $k \in N$.

Der SLR(k)-Parser für G ist dann ein Kellerautomat mit Ausgabe mit Kelleralphabet $[G']_0$, Eingabealalphabet T, Startinhalt $[\$]_0$ des Kellers, Menge der Endinhalte $\{[\$]_0\ [\$S]_0\}$ des Kellers, Menge der Aktionen bestehend aus allen SLR(k) reduce- und shift-Aktionen für G und dem Ausgabeeffekt τ, wobei τ wie üblich alle reduce-Aktionen durch Regel r nach r und alle shift-Aktionen nach ϵ überführt.

Definition: SLR(k)-Aktionen für G

Eine Regel der Form

$$(ra) \qquad [\delta]_0[\delta X_1]_0 \ldots [\delta X_1 \ldots X_n]_0|y \to [\delta]_0[\delta A]_0|y$$

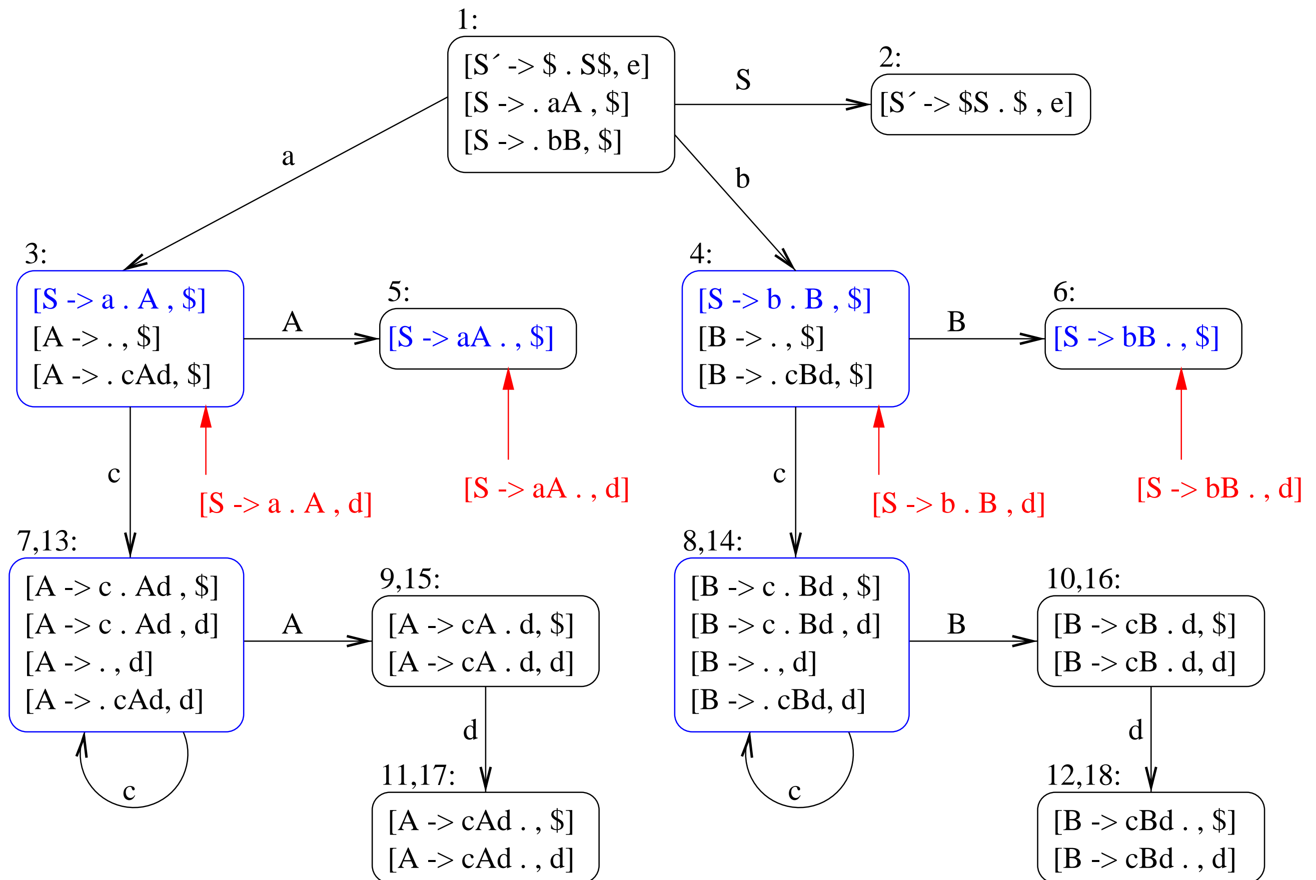


Abbildung 4: LALR(1)-Automat für $G_{a,b,\epsilon}$ mit SLR(1)-Items

heißt SLR(k) *reduce action* durch die Vorschrift $A \to X_1 \dots X_n$ mit Lookahead y, falls $\delta \in \$V^*$, $X_1, \dots, X_n$ Symbole aus $V (n \geq 0)$, $A \to X_1 \dots X_n$ eine Regel aus P und $y \in k : T^*\$$, so daß $[A \to X_1 \dots X_n \cdot] \in \text{VALID}_0(\delta X_1 \dots X_n)$ und $y \in FOLLOW_k(A)$.

Eine Regel der Form

$$(sa) \qquad [\delta]_0|ay \to [\delta]_0[\delta a]_0|y$$

heißt SLR(k) *shift action* auf einem Terminal a und Lookahead ay, falls $\delta \in \$V^*$, a ein Terminal $\in T$ und $y \in max\{k-1, 0\} : T^*\$$, so daß $[A \to \alpha \cdot a\beta] \in \text{VALID}_0(\delta)$ und $y \in FIRST_{max\{k-1,0\}}(\beta FOLLOW_k(A))$ für Regeln $A \to \alpha a\beta$.

3.4 SLR(k)-Grammatiken

Definition

Eine Grammatik $G = (V, T, P, S)$ ist SLR(k) genau dann, wenn ihr SLR(k)-Parser deterministisch ist und $S \Rightarrow^+ S$ unmöglich in G.

Satz4 : Determinismus des SLR(k)-Parsers

Der SLR(k)-Parser einer Grammatik G ist deterministisch genau dann, wenn die folgenden Aussagen für alle Zustände q des LR(0)-Automaten für die \$-erweiterte Grammatik G' wahr sind:

1. Falls q ein Paar verschiedener Items $[A_1 \to \omega_1 \cdot]$, $[A_2 \to \omega_2 \cdot]$ enthält, dann ist
$$FOLLOW_k(A_1) \cap FOLLOW_k(A_2) = \emptyset\,.$$
2. Falls q ein Itempaar $[A \to \alpha \cdot a\beta]$, $[B \to \omega \cdot]$ enthält, wobei a ein Terminal, dann ist
$$FIRST_k(a\beta FOLLOW_k(A)) \cap FOLLOW_k(B) = \emptyset\,.$$

Beweisskizze

Angenommen der SLR(k)-Parser besitzt zwei verschiedene reduce-Aktionen für dieselbe Konfiguration (reduce-reduce Konflikt). Dann gibt es in einem Zustand zwei Items $[A_1 \to w_1, y]$, $[A_2 \to w_2, y]$ mit gemeinsamen Lookahead y. y liegt dann sowohl in $FOLLOW_k(A_1)$ als auch in $FOLLOW_k(A_2)$ womit der Schnitt beider Mengen nicht leer ist und folglich Forderung (1) nicht gilt.

Angenommen der SLR(k)-Parser besitzt eine shift- und eine reduce-Aktion für dieselbe Konfiguration (shift-reduce Konflikt). Dann besitzen beide Items $[A \to \alpha \cdot a\beta, y]$, $[B \to w\cdot, y]$ einen gemeinsamen Lookahead-String y. y liegt dann sowohl in $FIRST_k(a\beta FOLLOW_k(A))$ als auch in $FOLLOW_k(B)$ womit der Schnitt beider Mengen nicht leer ist und folglich Forderung (2) nicht gilt. ■

Satz 5: Klasse der SLR(k)-Grammatiken

Die Klasse der SLR(0)-Grammatiken stimmt mit der Klasse der LALR(0)-Grammatiken überein. Für $k \geq 0$ ist die Klasse der SLR(k)-Grammatiken in der Klasse der LALR(k)-Grammatiken enthalten.

Beweisskizze

$k = 0$:
Der Beweis folgt direkt aus der Konstruktion des SLR(k)-Automaten aus dem LR(0)-Automaten (Äquivalenz der Automaten).

$k > 0$:
Der SLR(k)-Parser enthält alle Aktionen des LALR(k)-Parsers und eventuell mehr. Zusätzliche Aktionen können die Klasse der akzeptierten Sprachen aber nur verkleinern. Damit

ist auch die Klasse der zugehörigen Grammatiken kleiner oder gleich der des LALR(k)-Parser . Somit ist die Klasse der SLR(k)-Grammatiken eine Teilmenge der Klasse der LALR(k)-Grammatiken.

Offenbar sind nicht alle Strings aus $FOLLOW_k(A)$ in jedem Kontext gültige Nachfolger von A. Damit kann der SLR(k)-Automat für eine LALR(k)-Sprache in einen Zustand gelangen, aus dem er den Rest der Eingabe nicht mehr akzeptieren kann. Somit ist er kein gültiger Parser für die zugehörige LALR(k)-Grammatik und es folgt, daß nicht alle LALR(k)-Grammatiken auch SLR(k)-Grammatiken sind. ■

3.5 Beispiel: Fehlererkennung

Gegeben sei eine SLR(1)-Grammatik G mit den Regeln $S \to aA|Ac|Bd$, $A \to B$, $B \to b$. Abbildung 5 zeigt den Automaten für die entsprechenden Parser. Interessant ist nun der Vergleich des Verhaltens der drei Parser bei Eingabe des Wortes abc, welches nicht in $L(G)$ liegt.

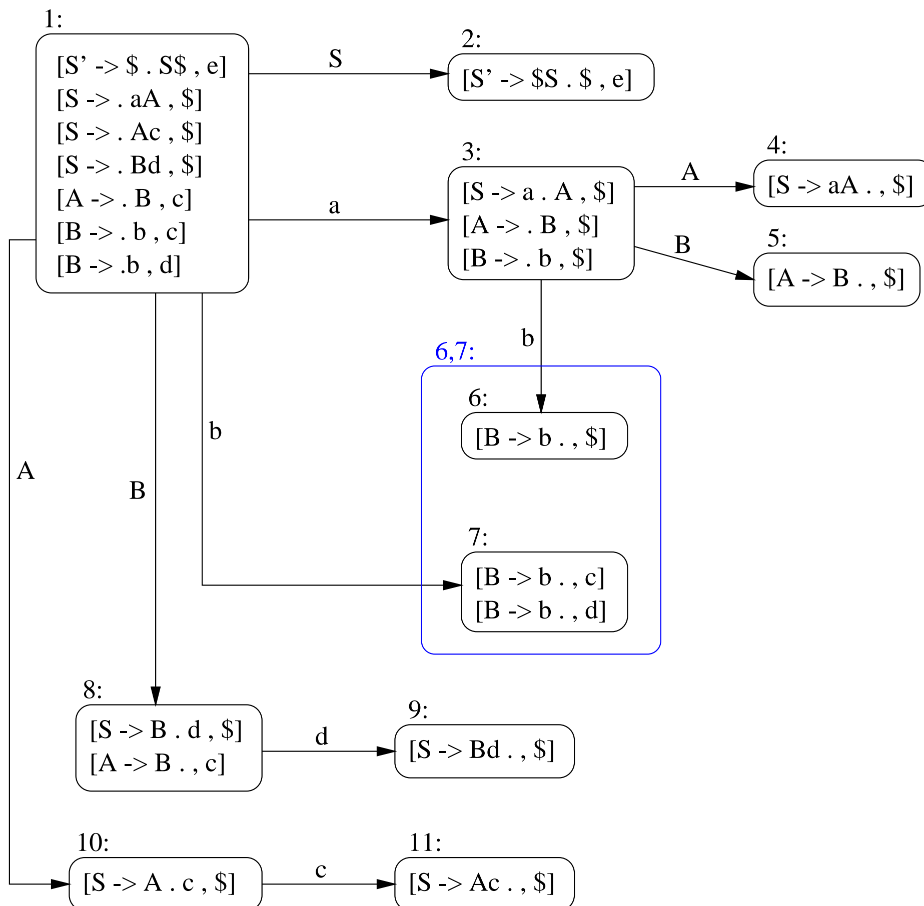


Abbildung 5: LR(1)-Automat und LALR(1)-Automat für G

Der LR(1)-Parser entdeckt den Fehler sobald er das Eingabesymbol c eingelesen hat. Er durchläuft damit folgende Folge von Konfigurationen: $\$q_1|abc\$ \Rightarrow \$q_1q_3|bc\$ \Rightarrow \$q_1q_3q_6|c\$$

Der LALR(1)-Parser führt bei Eingabesymbol c noch die Reduktion $B \rightarrow b$ durch bevor er den Fehler erkennt. Die Folge der durchlaufenen Konfigurationen lautet damit: $\$q_1|abc\$ \Rightarrow \$q_1q_3|bc\$ \Rightarrow \$q_1q_3q_{6,7}|c\$ \Rightarrow \$q_1q_3q_5|c$

Der SLR(1)-Parser schließlich führt zusätzlich noch die Reduktion $A \rightarrow B$ durch, da $c \in FOLLOW_1(A)$ enthalten ist. Also durchläuft er die folgenden Konfigurationen: $\$q_1|abc\$ \Rightarrow \$q_1q_3|bc\$ \Rightarrow \$q_1q_3q_{6,7}|c\$ \Rightarrow \$q_1q_3q_5|c \Rightarrow \$q_1q_3q_4|c$

Dieser Unterschied im Verhalten der drei Parser ist in der Praxis von geringer Bedeutung. Da keiner der Parser bei fehlerhafter Eingabe eine shift-Aktion durchführt, kann der Fehler in allen Fällen lokalisiert werden. Desweiteren sind LR(k)-, LALR(k)- und SLR(k)-Parser bei Eingabe einer SLR(k)-Grammatik bis auf das Verhalten bei fehlerhafter Eingabe äquivalent zueinander.

4 Vergleich der Verfahren

4.1 Unterschiede zwischen LALR(k)- und SLR(k)-Parsern

In den vorangegangenen Abschnitten wurden LALR(k)- und SLR(k)-Parser aus verschiedenen LR(k)- bzw. LR(0)-Parsern entwickelt. Es wurde jedoch bereits darauf hingewiesen, daß sich LALR(k)-Parser ebenfalls aus LR(0)-Parsern entwickeln lassen. Da in beiden Fällen die resultierenden Parser isomorph zum Ausgangsparser sind unterscheiden sie sich offenbar nur in der Konstruktion der Menge von Lookaheadstrings (Vorausschaumenge). Die Menge der zu einem Item $[X \rightarrow \alpha.\beta]$ im Zustand q hinzuzufügenden Vorausschaumenge heißt $LA(q, [X \rightarrow \alpha.\beta])$. Ihre Definition für den SLR(k)-Parser lautet:

$$LA_S(q, [X \rightarrow \alpha.]) = \{a \in T \cup \{\$\} | S'\$ \Rightarrow^* \beta X a \gamma\} = FOLLOW_k(X)$$

für alle q mit $[X \rightarrow \alpha.] \in q$. Das heißt jedem reduce-Item $[X \rightarrow \alpha.]$ wird unabhängig vom Kontext (das heißt in *allen* Zuständen) die Menge $FOLLOW_k(X)$ als Vorausschaumenge zugeordnet.

Für den LALR(k)-Parser wird zu jedem Vorkommen eines Items $[X \rightarrow \alpha.]$ in einem Zustand q eine *von q abhängige Vorausschaumenge* konstruiert. Diese Menge ist höchstens so groß wie die Vorausschaumenge eines SLR(k)-Parsers, in der Regel jedoch kleiner.

$$LA_L(q, [X \rightarrow \alpha]) = \{a \in T \cup \{\$\} | S'\$ \Rightarrow^*_{rm} \beta X a w \text{ und } \delta^*_d(q_d, \beta\alpha\ ) = q\}$$

Dabei ist δ_d ist die Übergangsfunktion des LR(0)-Automaten für G. Nach dieser Definition kommen nur solche Terminale in die Vorausschaumenge, die auf X in einer Rechtsatzform $\beta X a w$ folgen können, sodaß der Automat bei Eingabe von $\beta\alpha$ in den Zustand q kommt.

4.2 Ergebnisse

Ein wichtiges Ergebnis der vorangegangenen Ausführungen ist die Erkenntnis, daß die Klasse der SLR(k)-Grammatiken eine Teilmenge der Klasse der LALR(k)-Grammatiken ist, welche wiederum eine Teilmenge der Klasse der LR(k)-Grammatiken ist. Dies bedeutet, daß es genügt eine Grammatik mit SLR(k)-Eigenschaft zu finden, um sie mit einem der Verfahren zu parsen. Glücklicherweise kann man beliebige kontextfreie Grammatiken in deterministischer, polynomieller Zeit in der Größe von $|G|$ auf SLR(k)-Eigenschaft testen, wohingegen der Test auf LALR(k)-Eigenschaft PSPACE-vollständig ist (für $k > 0$). Dies hängt auch damit zusammen, daß der SLR(k)-Parser einfacher zu konstruieren ist als der LALR(k)-Parser, da der Lookahead unabhängig vom Kontext ist. Mit der Erkenntnis, daß sich alle drei Parser für eine SLR(k)-Grammatik nur in der Erkennung fehlerhafter Eingabestrings unterscheiden, läßt sich zu dieser Grammatik ein LALR(k)-Parser konstruieren, um die erzeugte Sprache effizent zu parsen.

5 Transformation von Grammatiken

5.1 Motivation

Nach den vorangegangenen Ausführungen bleibt noch das Problem des Auffindens einer geeigneten Grammatik. Eventuell steht man auch vor dem Problem, daß eine bestimmte Grammatik zu parsen ist, man aber zu dieser Grammatik keinen Parser konstruieren kann. Um die Grammatik trotzdem parsen zu können, versucht man nun eine geeignete *Transformation* zu finden, um diese Grammatik in eine äquivalente Grammatik zu transformieren zu welcher ein Parser konstruiert werden kann. Die Transformation muß die folgende Eigenschaft erfüllen:

> Aus dem Parsen eines Satzes der transformierten Grammatik, läßt sich das Parsen dieses Satzes in der Originalgrammatik rekonstruieren.

Im Folgenden werden nun zwei verschiedene Transformationen vorgestellt, die von allgemeinerem Interesse sind.

5.2 Transformation von LR(k)- in SLR(k)-Grammatiken

Wir wollen nun zeigen, daß jede LR(k)-Sprache durch eine SLR(k)-Grammatik erzeugt werden kann. Dazu suche man eine geeignete Grammatiktransformation, welche eine Grammatik G in eine transformierte Grammatik $T_k(G)$ überführt, wobei $T_k(G)$ SLR(k)-Eigenschaft besitzt genau dann wenn G LR(k)-Eigenschaft aufweist.

5.2.1 Vorgehensweise

Gegeben ist eine Grammatik G. Die gesuchte Transformation T_k funktioniert folgendermaßen:

- Ersetze jedes Nichtterminal A in G durch $([\gamma]_k, A)$, wobei γ zuverlässiges Präfix für G ist.
- $([\gamma]_k, A)$ generiert dieselbe Sprache wie A, kann aber in Rechtsableitungen nur in Kontexten äquivalent zu γ benutzt werden.
- $FOLLOW_k(([\gamma]_k, A))$ enthält dann nur Strings aus $FOLLOW_k(A)$, die legale Nachfolger von A in einem zu γ äquivalenten Kontext sind.

Dabei erzeugt T_k eine zu G strukturäquivalente Grammatik.

Definition: Strukturäquivalenz

Zwei Grammatiken G_1 und G_2 heißen *strukturäquivalent* genau dann wenn

1. $L(G_1) = L(G_2)$ und
2. die Ableitungsbäume für einen Satz w in G_1 und G_2 sind bis auf die Benennung von Nichtterminalknoten identisch.

Somit gelten für die Transformation T_k die folgenden Aussagen:

1. Falls $T_k(G)$ SLR(k)-Eigenschaft aufweist, besitzt G LR(k)-Eigenschaft.
2. Falls $T_k(G)$ *nicht* SLR(k)-Eigenschaft besitzt, ist G keine LR(k)-Grammatik.

5.2.2 Ergebnisse

Die Ausführungen dieses Abschnitts lassen nun folgende Schlußfolgerungen zu:

Sei k eine natürliche Zahl. Dann kann jede Grammatik G in eine strukturäquivalente Grammatik mit SLR(k)-Eigenschaft transformiert werden genau dann wenn G LR(k)-Eigenschaft besitzt. Desweiteren gilt für festes $k \geq 0$, daß die Familien der LR(k)-Sprachen, LALR(k)-Sprachen und SLR(k)-Sprachen äquivalent sind. Dies bedeutet für die Praxis, daß einfach jede LR(k)-Sprache durch eine SLR(k)-Grammatik erzeugt werden kann.

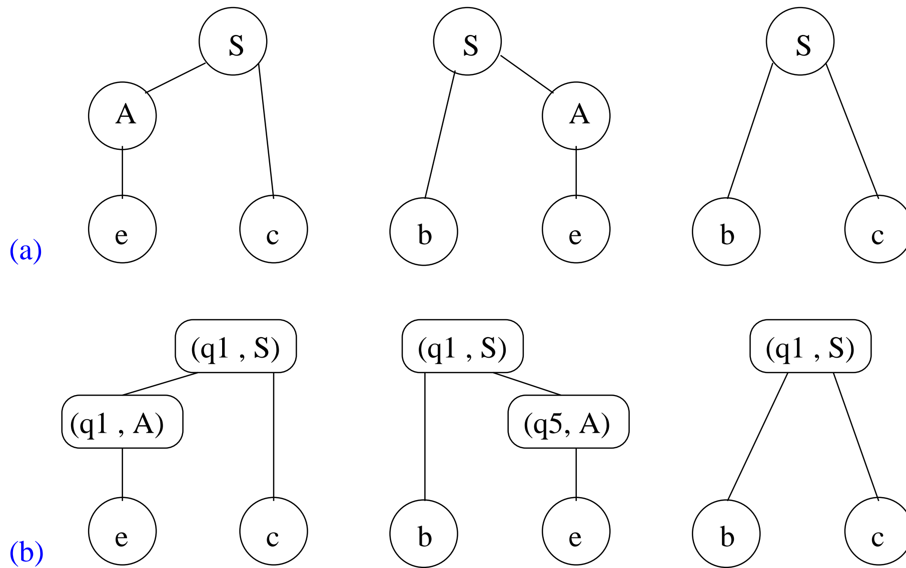


Abbildung 6: Ableitungsbäume für Sätze aus den strukturäquivalenten Grammatiken G_{nslr} (a) und $T_1(G_{nslr})$ (b)

5.3 Transformation von LR(k)- in LR(1)-Grammatiken

Wie aus Abschnitt 1 bereits bekannt ist, wächst die Zustandsmenge des kanonischen LR(k)-Parsers mit der Länge des Lookaheads k. Gleiches gilt auch für LALR(k)-Parser, wenngleich auch diese deutlich kleiner sind als ihre korrespondierenden LR(k)-Parser. Mittels einer Grammatiktransformation läßt sich jedoch zeigen, daß für alle $k \geq 1$ die Familien der LR(k)-Sprachen und der LR(1)-Sprachen äquivalent sind.

- Zu jeder LR(k)-Grammatik gibt es eine äquivalente LR(1)-Grammatik.
- Jede LR(k)-Grammatik kann in eine LR(1)-Grammatik transformiert werden, welche die Originalgrammatik "überdeckt".

5.3.1 Vorgehensweise

Gegeben ist eine Grammatik G, welche in eine Grammatik $T_{k,1}(G)$ transformiert werden soll. Dabei besitzt $T_{k,1}(G)$ genau dann LR(1)-Eigenschaft, wenn G LR($k+1$)-Eigenschaft besitzt. Dazu ersetzt man alle Nichtterminale A aus G durch (x, A, y), wobei $y \in FOLLOW_k$ (A) und $x \in FIRST_k(Ay)$. (x, A, y) erzeugt nun die Sprache

$$\{z \in T^* | xz = vy, A \Rightarrow^* v, \text{ und } k : vy = x\}$$

Diese Transformation entspricht einem *Shift* der Ableitungsbäume in G um k Symbole nach rechts. Das bedeutet für den Parser, daß Reduktionen zurückgestellt werden bis ein Lookahead der Größe 1 ausreicht um die nächste Parsing-Aktion eindeutig zu bestimmen.

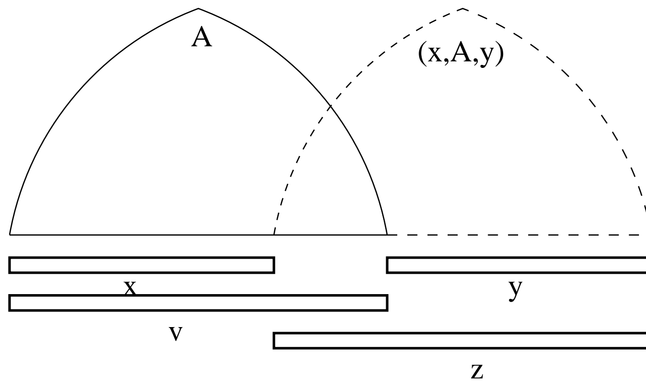


Abbildung 7: Ableitungsbäume in G und $T_k(G)$ (gestrichelt dargestellt) mit $vy = xz$, $A \Rightarrow^* v, (x, A, y) \Rightarrow^* z$, $k : vy = x$ und $|v| > k$

Abbildung 7 verdeutlich den Shift des Ableitungsbaums. Der Ableitungsbaum des Präfixes v in vy wird um k Symbole nach rechts verschoben, so daß er ein Ableitungsbaum des Suffixes z in $vy = xz$ wird.

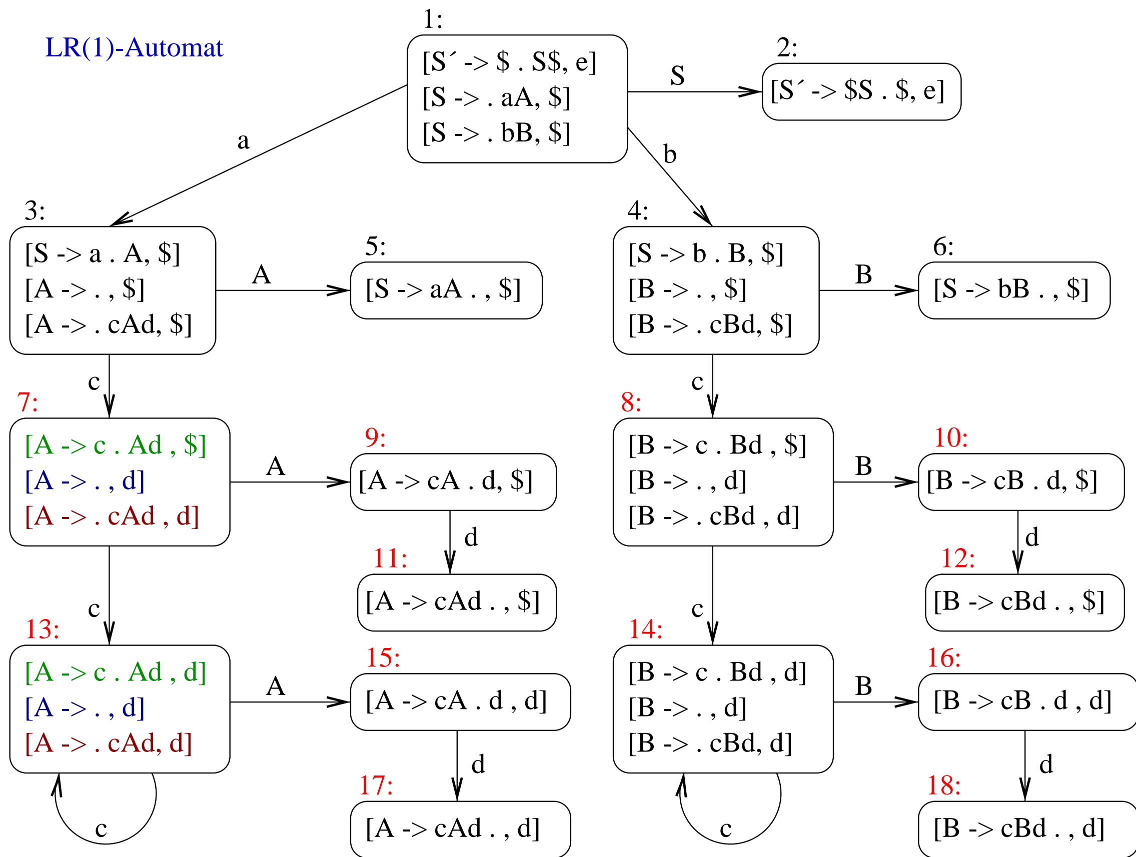
5.4 Schlußfolgerungen

Das Ergebnis des vorangegangenen Abschnitts besagt nun, daß man beim Parsen auf Lookahead-Strings der Länge $k > 1$ verzichten kann, da jede LR(k)-Grammatik in eine LR(1)-Grammatik transformiert werden kann. Da weiterhin jede LR(1)-Grammatik in eine äquivalente SLR(1)-Grammatik transformiert werden kann folgt schließlich, daß jede deterministische Sprache bereits durch einen SLR(1)-Parser geparsed werden kann, folglich auch durch einen LALR(1)-, bzw. LR(1)-Parser.

Literatur

[1] Soisalou, Soininen: *Parsing Theory*

[2] R. Wilhelm, D. Maurer: *Übersetzerbau*, Springer Verlag, 2. Auflage (1997)



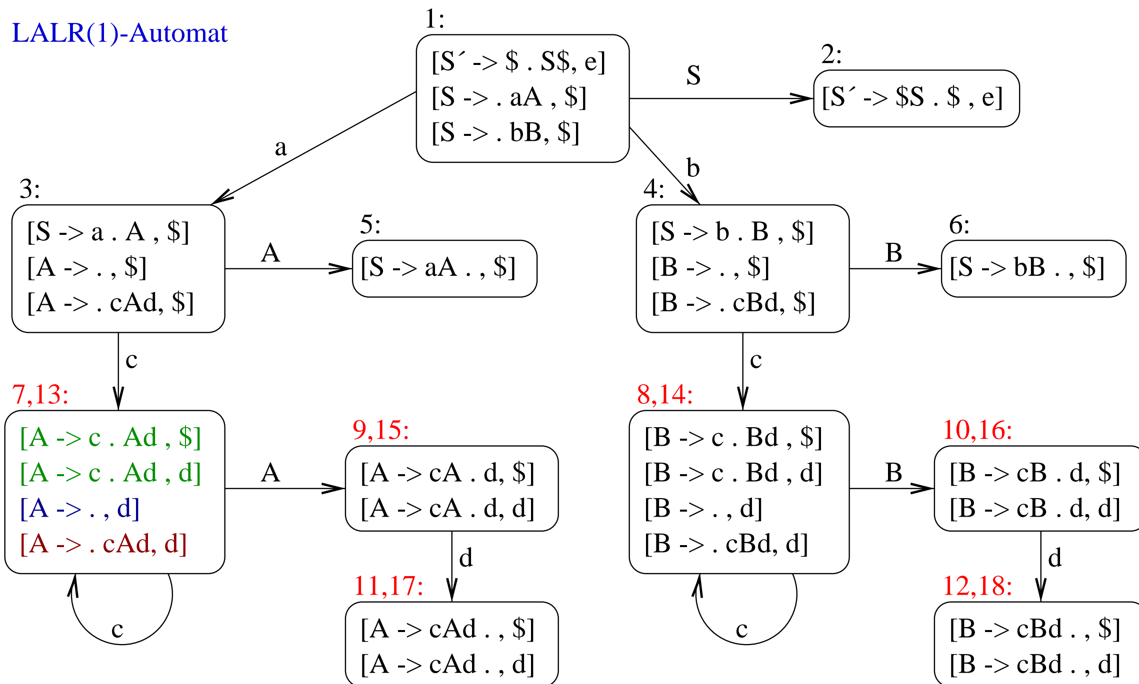


Abbildung 8: Konstruktion des LALR(1)-Automat für $G_{ab\epsilon}$