sie schließlich den mit dem Startsymbol der Grammatik markierten Knoten als Wurzelknoten des gesamten Syntaxbaums erzeugen und einbauen. Sie verwenden bei der Analyse einen Keller, auf dem zwei Operationen möglich sind,

- Kellern des nächsten Eingabesymbols (shift), und
- Lokalisieren einer rechten Produktionsseite am oberen Kellerende und Ersetzen durch das Nichtterminal der entsprechenden linken Seite (reduce).

Von diesen beiden Operationen haben sie den Namen **shift-reduce-Parser**. Mit der Festlegung, daß nur am oberen Kellerende reduziert werden darf, ist festgelegt, daß ein solcher shift-reduce-Parser ein Rechtsparser ist, d.h. als Ergebnis einer erfolgreichen Analyse eine Rechtsableitung in gespiegelter Reihenfolge liefert. Wie macht man sich dies klar?

Ein shift-reduce-Parser darf niemals eine „gebotene" Reduktion verpassen, d.h. durch ein eingelesenes Symbol im Keller „zudecken". Dabei soll eine Reduktion geboten heißen, wenn sie gemacht werden muß, um eine Rechtsableitung bis zum Startsymbol zu erhalten. Ist eine rechte Seite erst einmal zugedeckt, wird sie nicht mehr am oberen Ende des Kellers auftauchen, es sei denn man läßt zu, daß vorgenommene Reduktionen und Leseaktionen rückgängig gemacht werden. Die zugedeckte, gebotene Reduktion kann nicht mehr durchgeführt werden. Eine rechte Seite am oberen Kellerende, die reduziert werden muß, um eine Ableitung zu erhalten, werden wir **Griff** (handle) nennen.

Nicht alle Vorkommen von rechten Seiten am oberen Kellerende sind jedoch Griffe. Manchmal würden Reduktionen am oberen Kellerende in Sackgassen, d.h. nicht zu einer Rechtsableitung, führen.

Beispiel 8.4.1
G_0 sei wieder unsere Ausdrucksgrammatik mit den Produktionen:

$$\begin{array}{lcl} S & \rightarrow & E \\ E & \rightarrow & E + T \mid T \\ T & \rightarrow & T * F \mid F \\ F & \rightarrow & (E) \mid \mathbf{id} \end{array}$$

Tabelle 8.5 zeigt eine erfolgreiche bottom up-Analyse des Satzes $\mathbf{id} * \mathbf{id}$ von G_0. Dabei sind in einem Schritt das Verpassen einer gebotenen Reduktion und in zwei Schritten auch die anderen Reduktionsmöglichkeiten angegeben; diese würden jedoch in Sackgassen führen, d.h. nicht zu Rechtssatzformen. □

8.4.2 Bottom up-Analyse mit Zurücksetzen in Prolog

Ähnlich wie bei der top down-Syntaxanalyse wollen wir auch hier Prolog dazu benutzen, eine sehr ineffiziente Art der bottom up-Syntaxanalyse vorzuführen, die Analyse mit Zurücksetzen (bottom up backtrack parsing). Der unten angegebene bottom up-Parser in Prolog versucht, eine Ableitung durch Ausprobieren zu finden. Dabei hat er durch die Anordnung der Klauseln für *parse* eine Präferenz

Tabelle 8.5: Eine erfolgreiche Analyse des Satzes **id**∗**id** mit den möglichen Sackgassen.

Keller	Eingabe	Kommentar
	id ∗ **id**	
id	∗**id**	
F	∗**id**	Lesen von ∗ würde gebotene Reduktion verpassen
T	∗**id**	Reduktion von T zu E führte in eine Sackgasse
T∗	**id**	
T ∗ **id**		
$T * F$		Reduktion von F zu T führte in eine Sackgasse
T		
E		
S		akzeptiert

für die Reduktion, d.h. er reduziert immer soviel wie möglich, bevor er liest. Das Prädikat *parse* hat zwei Argumente, die restliche Eingabe und den Analysekeller. Beide werden durch Listen dargestellt. Da die Kelleroperationen effizient nur auf den Listenanfang realisiert werden können, spiegeln wir den Kellerinhalt; das oberste Kellersymbol befindet sich also am Listenanfang. Der Parser zerfällt wieder in den Parser-Treiber, der unabhängig von der Grammatik ist, – das ist das Prädikat *parse*– und die Darstellung der Produktionsregeln – das sind die Klauseln für das Prädikat *reduce*. Es folgt der vollständige Parser für die Grammatik für arithmetische Ausdrücke.

```
% Parsertreiber

% Akzeptieren
parse([$], [n(s) ]).

% Reduktion
parse(Input, Stack) :-  reduce(Stack, Newstack),
                        parse(Input, Newstack).
% Lesen
parse([NextSym | Rest], Stack) :- parse(Rest, [NextSym | Stack]).

% Produktionsregeln

reduce( [n(expr) | R], [n(s) | R]).
reduce( [n(term), t('+'), n(expr) | R ], [n(expr) | R]).
reduce( [n(term) | R], [n(expr) | R]).
reduce( [n(factor), t('*'), n(term) | R], [n(term) | R]).
reduce( [n(factor) | R], [n(term) | R]).
reduce( [t(')'), n(expr), t('(') | R], [n(factor) | R]).
reduce( [t(id) | R], [n(factor) | R]).
```

Angesetzt auf die Anfrage

```
?- parse ([t('('), t(id), t(')'), $], []).
```

führt er die folgende Berechnung durch:

```
C-Prolog version 1.5

| ?- parse([t('('), t(id), t(')'), $], []).
   (1) 1 Call: parse([t(('(),t(id),t(')'),$],[])
   (2) 2 Call: reduce([],_65637)
   (2) 2 Fail: reduce([],_65637)
   (1) 1 Back to: parse([t(('(),t(id),t(')'),$],[])
   (3) 3 Call: parse([t(id),t(')'),$],[t(('(')])
   (4) 4 Call: reduce([t(('(')],_65645)
   (4) 4 Fail: reduce([t(('(')],_65645)
   (3) 3 Back to: parse([t(id),t(')'),$],[t(('(')])
   (5) 5 Call: parse([t(')'),$],[t(id),t(('(')])
   (6) 6 Call: reduce([t(id),t(('(')],_65653)
   (6) 6 Exit: reduce([t(id),t(('(')],[n(factor),t(('(')])
   (7) 6 Call: parse([t(')'),$],[n(factor),t(('(')])
   (8) 7 Call: reduce([n(factor),t(('(')],_65672)
   (8) 7 Exit: reduce([n(factor),t(('(')],[n(term),t(('(')])
   (9) 7 Call: parse([t(')'),$],[n(term),t(('(')])
   (10) 8 Call: reduce([n(term),t(('(')],_65691)
   (10) 8 Exit: reduce([n(term),t(('(')],[n(expr),t(('(')])
   (11) 8 Call: parse([t(')'),$],[n(expr),t(('(')])
   (12) 9 Call: reduce([n(expr),t(('(')],_65710)
   (12) 9 Exit: reduce([n(expr),t(('(')],[n(s),t(('(')])
   (13) 9 Call: parse([t(')'),$],[n(s),t(('(')])
   (14) 10 Call: reduce([n(s),t(('(')],_65729)
   (14) 10 Fail: reduce([n(s),t(('(')],_65729)
   (13) 9 Back to: parse([t(')'),$],[n(s),t(('(')])
   (15) 11 Call: parse([$],[t(')'),n(s),t(('(')])
   (16) 12 Call: reduce([t(')'),n(s),t(('(')],_65737)
   (16) 12 Fail: reduce([t(')'),n(s),t(('(')],_65737)
   (15) 11 Back to: parse([$],[t(')'),n(s),t(('(')])
   (17) 13 Call: parse([],[$,t(')'),n(s),t(('(')])
   (18) 14 Call: reduce([$,t(')'),n(s),t(('(')],_65745)
   (18) 14 Fail: reduce([$,t(')'),n(s),t(('(')],_65745)
   (17) 13 Back to: parse([],[$,t(')'),n(s),t(('(')])
   (17) 13 Fail: parse([],[$,t(')'),n(s),t(('(')])
   (15) 11 Fail: parse([$],[t(')'),n(s),t(('(')])
   (13) 9 Fail: parse([t(')'),$],[n(s),t(('(')])
   (12) 9 Back to: reduce([n(expr),t(('(')],_65710)
   (12) 9 Fail: reduce([n(expr),t(('(')],_65710)
```



```
   (11) 8 Back to: parse([t(')'),$],[n(expr),t(('(')])
   (19) 15 Call: parse([$],[t(')'),n(expr),t(('(')])
   (20) 16 Call: reduce([t(')'),n(expr),t(('(')],_65718)
   (20) 16 Exit: reduce([t(')'),n(expr),t(('(')],[n(factor)])
   (21) 16 Call: parse([$],[n(factor)])
   (22) 17 Call: reduce([n(factor)],_65737)
   (22) 17 Exit: reduce([n(factor)],[n(term)])
   (23) 17 Call: parse([$],[n(term)])
   (24) 18 Call: reduce([n(term)],_65756)
   (24) 18 Exit: reduce([n(term)],[n(expr)])
   (25) 18 Call: parse([$],[n(expr)])
   (26) 19 Call: reduce([n(expr)],_65775)
   (26) 19 Exit: reduce([n(expr)],[n(s)])
   (27) 19 Call: parse([$],[n(s)])
   (27) 19 Exit: parse([$],[n(s)])
```

... weitere erfolgreiche Exits

```
   (3) 3 Exit: parse([t(id),t(')'),$],[t(('(')])
   (1) 1 Exit: parse([t(('(),t(id),t(')'),$],[])

yes
```

8.4.3 LR(k)-Analysatoren

In diesem Abschnitt wird das mächtigste deterministisch bottom up-arbeitende Syntaxanalyseverfahren behandelt, die LR(k)-Analyse. Dabei steht L dafür, daß die Analysatoren dieser Klasse ihre Eingabe von links nach rechts lesen, und R kennzeichnet sie als Rechtsparser; k gibt an, wieviele Symbole sie in der Eingabe vorausschauen dürfen, um Entscheidungen zu treffen.

Wir gehen von dem Item-Kellerautomaten K_G für eine kontextfreie Grammatik G aus, um einen LR(k)-Analysator für G zu finden. Im LL(k)-Ansatz versucht man, aus der Grammatik für die Expansionsübergänge von K_G solche Vorausschauworte zu berechnen, die eine eindeutige Auswahl der zu wählenden Alternative gestatten. LR(k)-Analysatoren versuchen, solch frühzeitige Entscheidungen zu vermeiden und, solange das realisierbar ist, alle Möglichkeiten parallel zu verfolgen. Erst wenn Entscheidungsbedarf gegeben ist, also wenn sowohl Weiterlesen als auch Reduzieren oder Reduktionen mittels mehrerer Produktionen möglich sind, wird aufgrund der Vorausschau von k Symbolen entschieden, ob gelesen oder reduziert bzw. mittels welcher Produktion reduziert wird.

Nach der mehr didaktisch motivierten Herleitung des LR(0)-Analysators und einer praktikablen Methode seiner Konstruktion in diesem Abschnitt wird im Abschnitt 8.4.4 der sogenannte kanonische LR(k)-Parser mit seinen Eigenschaften vorgestellt. Im Abschnitt 8.4.5 werden schwächere, aber für die Praxis meist ausreichend starke Varianten der LR(k)-Analyse eingeführt. Zum Abschluß wird ein Fehlerbehandlungsverfahren für die LR(k)-Analysemethode erläutert.

Der charakteristische endliche Automat zum Item-Kellerautomaten

An Stelle der tabellarischen Darstellung der Übergangsrelation von K_G – wie in Tabelle 8.1 – kann man K_G auch durch einen nichtdeterministischen endlichen Automaten, seinen charakteristischen endlichen Automaten, darstellen. Da diese Darstellung natürlich die gleiche Information enthalten soll, muß man mit einigen Zuständen bzw. Übergängen dieses NEA Kelleroperationen verbinden.

Definition 8.4.1 (charakteristischer endlicher Automat)
Sei G eine kontextfreie Grammatik. Den folgenden nichtdeterministischen endlichen Automaten $char(K_G) = (Q_c, V_c, \Delta_c, q_c, F_c)$ nennen wir den **charakteristischen endlichen Automaten** zu K_G, wenn gilt:
$Q_c = It_G$; seine Zustände sind die Items der Grammatik;
$V_c = V_T \cup V_N$; sein Eingabealphabet besteht aus den Nichtterminal- und Terminalsymbolen;
$q_c = [S' \to .S]$; der Startzustand ergibt sich aus der hinzugefügten Produktion $S' \to S$;
$F_c = \{[X \to \alpha.] \mid X \to \alpha \in P\}$; Endzustände sind die vollständigen Items;
$\Delta_c = \{([X \to \alpha.Y\beta], Y, [X \to \alpha Y.\beta]) \mid X \to \alpha Y\beta \in P \text{ und } Y \in V_N \cup V_T\}$
$\cup \{([X \to \alpha.Y\beta], \varepsilon, [Y \to .\gamma]) \mid X \to \alpha Y\beta \in P \text{ und } Y \to \gamma \in P\}$;

char(K_G*)* kennt Übergänge innerhalb einer Produktion und zwar unter Terminalen wie Nichtterminalen und Übergänge, die zu den Expansionsübergängen von K_G korrespondieren. □

Beispiel 8.4.2
Sei G_0 wieder die Grammatik für arithmetische Ausdrücke mit den Produktionen

$$\begin{array}{lcl} S & \to & E \\ E & \to & E+T \mid T \\ T & \to & T*F \mid F \\ F & \to & (E) \mid \mathbf{id} \end{array}$$

Der charakteristische endliche Automat zu K_{G_0} ist in Abb. 8.18 durch sein Übergangsdiagramm dargestellt. □

Wie können wir einen Kellerautomaten durch einen endlichen Automaten vollständig beschreiben? Das geht im allgemeinen natürlich nicht. Aber für K_G funktioniert es aufgrund seiner besonderen Struktur.

Mit jedem ε-Übergang in *char(*K_G*)* verbinden wir eine Keller-Operation in K_G, die den neuen Zustand von *char(*K_G*)* auf den Keller von K_G drückt.
Die Übergänge von *char(*K_G*)* unter Terminalsymbolen entsprechen genau den Leseübergängen, die K_G auf seinem aktuellen Zustand, d.h. seinem obersten Kelleritem ausführt. Gelangt *char(*K_G*)* in einen Endzustand $[X \to \alpha.]$, so entspricht das folgenden Aktionen in K_G: K_G entfernt das bei ihm oben auf dem Keller befindliche Item $[X \to \alpha.]$ vom Keller und macht aus dem neuen obersten Kelleritem einen Übergang unter X, wie es Δ_c beschreibt. Das Ergebnis ist der neue Zustand von *char(*K_G*)* (und von K_G). Damit haben wir die Arbeitsweise

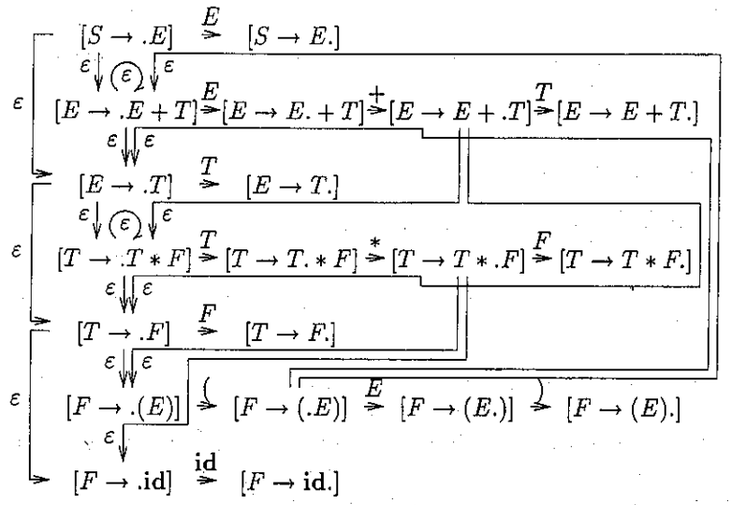


Abb. 8.18: Der charakteristische endliche Automat *char(*K_{G_0}*)* für die Grammatik G_0.

des Kellerautomaten K_G durch die Angabe des endlichen Automaten *char(*K_G*)* beschrieben. *char(*K_G*)* ist eine kompaktere Darstellung von K_G, weil man einige Redundanzen in der Beschreibung von K_G vermeidet. Etwas scharfes Hinsehen macht es offenkundig, daß wegen der Konstruktion von K_G alle (R)-Übergänge Redundanzen enthalten:

- Ein (R)-Übergang, besagt die Definition von δ, ist nur dann möglich, wenn das obere Kellerende die Form $[Y \to \beta.X\gamma][X \to \alpha.]$ hat, d.h. wenn das zweitoberste Item einen Übergang unter der linken Seite des obersten Items erlaubt. Nach Konstruktion ist das aber immer der Fall.

- Was geschieht in einem (R)-Übergang? Der aktuelle Zustand $[X \to \alpha.]$ wird vergessen, und das oberste Kelleritem wird, nachdem der Punkt über das X geschoben wurde, als neuer aktueller Zustand installiert. Das wird getan unabhängig davon, welches spezielle Item (allerdings mit X hinter dem Punkt) oben auf dem Keller steht.

Diese beiden Arten von Redundanz eliminiert *char(*K_G*)*. Zusätzlich hat der charakteristische endliche Automat zu einem Item-Kellerautomaten aber noch andere interessante Eigenschaften. Betrachtet man alle seine Zustände als Endzustände, so akzeptiert er eine für die LR-Analyse äußerst interessante Sprache.

Definition 8.4.2 (Griff, zuverlässiges Präfix)
Sei $S' \underset{rm}{\overset{*}{\Longrightarrow}} \beta X u \underset{rm}{\Longrightarrow} \beta\alpha u$ eine Rechtsableitung zu einer kontextfreien Grammatik G. α heißt dann der **Griff** der Rechtssatzform $\beta\alpha u$. Jedes Präfix von $\beta\alpha$ heißt ein **zuverlässiges Präfix** von G. □

Beispiel 8.4.3
In der Grammatik G_0 sind

Rechtssatzform	Griff	zuverl. Präfix	Begründung
$E + F$	F	E, $E+$, $E + F$	$S \underset{rm}{\Longrightarrow} E \underset{rm}{\Longrightarrow} E + T \underset{rm}{\Longrightarrow} E + F$
$T *$ **id**	**id**	T, $T*$, $T *$ **id**	$S \underset{rm}{\overset{3}{\Longrightarrow}} T * F \underset{rm}{\Longrightarrow} T *$ **id**

□

In einer nicht mehrdeutigen Grammatik ist der Griff einer Rechtssatzform das eindeutig bestimmte Teilwort, welches in einem bottom up-Analysator im nächsten Reduktionsschritt durch ein Nichtterminal ersetzt werden muß, um zu einer Rechtsableitung zu kommen. Ein zuverlässiges Präfix ist der Anfang einer Rechtssatzform, der sich nicht echt über den Griff hinaus erstreckt. In einem zuverlässigen Präfix ist also eine Reduktion höchstens am Ende möglich. Bis auf diese (evtl. mögliche) Reduktion am Ende des zuverlässigen Präfixes ist es also soweit wie möglich reduziert.

Definition 8.4.3 (gültiges Item)
Ein Item $[X \to \alpha.\beta]$ heißt **gültig** für das zuverlässige Präfix $\gamma\alpha$, wenn es eine Rechtsableitung $S' \underset{rm}{\overset{*}{\Longrightarrow}} \gamma X w \underset{rm}{\Longrightarrow} \gamma\alpha\beta w$ gibt. □

Beispiel 8.4.4
Wir geben zwei zuverlässige Präfixe von G_0 und einige für sie gültige Items an. Um zu zeigen, daß diese Items gültig gemäß Definition 8.4.3 sind, wird jeweils eine Rechtssatzform und die entsprechenden Bindungen der Variablen aus Definition 8.4.3 angegeben.

zuverl. Präfix	gültige Items	Begründung	γ	w	X	α	β
$E+$	$[E \to E + .T]$	$S \underset{rm}{\Longrightarrow} E \underset{rm}{\Longrightarrow} E + T$	ε	ε	E	$E+$	T
	$[T \to .F]$	$S \underset{rm}{\overset{*}{\Longrightarrow}} E + T \underset{rm}{\Longrightarrow} E + F$	$E+$	ε	T	ε	F
	$[F \to .\mathbf{id}]$	$S \underset{rm}{\overset{*}{\Longrightarrow}} E + F \underset{rm}{\Longrightarrow} E +$ **id**	$E+$	ε	F	ε	**id**
$(E + ($	$[F \to (.E)]$	$S \underset{rm}{\overset{*}{\Longrightarrow}} (E + F)$ $\underset{rm}{\Longrightarrow} (E + (E))$	$(E+$	)	F	(	$E)$

□

Ist beim Versuch, eine Rechtsableitung für ein Wort zu erstellen, das bisher gelesene Präfix x des Wortes zu einem zuverlässigen Präfix $\gamma\alpha$ reduziert worden, so beschreibt jedes für $\gamma\alpha$ gültige Item $[X \to \alpha.\beta]$ eine mögliche Interpretation

der Analysesituation. Es gibt also eine Rechtsableitung, in der $\gamma\alpha$ Präfix einer Rechtssatzform und $X \to \alpha\beta$ eine der möglichen gerade „bearbeiteten" Produktionen ist. Alle solchen Produktionen sind Kandidaten für spätere Reduktionen.

Betrachten wir die Rechtsableitung $S' \underset{rm}{\overset{*}{\Longrightarrow}} \gamma X w \underset{rm}{\Longrightarrow} \gamma\alpha\beta w$. Da sie als Rechtsableitung fortgesetzt werden soll, muß eine Reihe von Schritten folgen, die β zu einem Terminalwort v ableiten, darauf eine Reihe von Schritten, die α zu einem Terminalwort u ableiten, also $S' \underset{rm}{\overset{*}{\Longrightarrow}} \gamma X w \underset{rm}{\Longrightarrow} \gamma\alpha\beta w \underset{rm}{\overset{*}{\Longrightarrow}} \gamma\alpha v w \underset{rm}{\overset{*}{\Longrightarrow}} \gamma u v w$. Das gültige Item $[X \to \alpha.\beta]$ für das zuverlässige Präfix $\gamma\alpha$ beschreibt die Analysesituation, in der die Reduktion von u nach α bereits geschehen ist, während die Reduktion von v nach β noch nicht begonnen hat. Ein mögliches „Fernziel" in dieser Situation ist die Anwendung der Produktion $X \to \alpha\beta$.

Satz 8.4.1
Zu jedem zuverlässigen Präfix gibt es mindestens ein gültiges Item.

Wir kommen zurück zu der Frage, welche Sprache der charakteristische endliche Automat von K_G akzeptiert. Satz 8.4.2 besagt, daß er unter einem zuverlässigen Präfix in einen Zustand übergeht, der ein gültiges Item für dieses Präfix ist. Endzustände, das sind vollständige Items, sind nur gültig für maximal lange zulässige Präfixe, d.h. Präfixe mit einer möglichen Reduktion am Ende.

Satz 8.4.2
Ist $\gamma \in (V_T \cup V_N)^*$ und $q \in Q_c$, dann gilt $(q_c, \gamma) \overset{*}{\vdash}_{char(K_G)} (q, \varepsilon)$ genau dann, wenn γ ein zuverlässiges Präfix und q ein gültiges Item für γ ist.

Beweis:
„ $\Rightarrow$ "
Wir führen den Beweis durch Induktion über die Länge der Berechnung.
$(q_c, \gamma) \overset{0}{\vdash}_{char(K_G)} (q_c, \varepsilon)$ gilt nur für $\gamma = \varepsilon$. ε ist offensichtlich ein zuverlässiges Präfix, und $q_c = [S' \to .S]$ ist ein gültiges Item für ε.
Sei nun vorausgesetzt, daß die Behauptung für alle Berechnungen der Länge kleiner als n gelte. Wir nehmen jetzt an, daß $(q_c, \gamma) \overset{n}{\vdash}_{char(K_G)} (q, \varepsilon)$.
Dann müssen wir zwei Fälle betrachten.

1. Fall: $(q_c, \gamma) \overset{n-1}{\vdash}_{char(K_G)} (p, \varepsilon) \vdash_{char(K_G)} (q, \varepsilon)$; d.h. γ wurde schon nach spätestens $n-1$ Schritten konsumiert, und der letzte Schritt ist ein ε-Übergang. Dann muß gelten $p = [Y \to \alpha.X\beta]$ und $q = [X \to .\delta]$. Nach Induktionsvoraussetzung ist γ ein zuverlässiges Präfix und p ein gültiges Item für γ. Dann ist trivialerweise auch q ein gültiges Item für γ.

2. Fall: $\gamma = \gamma' X$ und $(q_c, \gamma' X) \overset{n-1}{\vdash}_{char(K_G)} (p, X) \vdash_{char(K_G)} (q, \varepsilon)$. Nach Induktionsvoraussetzung ist γ' ein zuverlässiges Präfix und p gültig für γ', da $(q_c, \gamma') \overset{n-1}{\vdash} (p, \varepsilon)$. Dann hat p die Form $[Y \to \alpha.X\beta]$ und q die Form $[Y \to \alpha X.\beta]$. Also gibt es

eine Rechtsableitung $S \underset{rm}{\overset{*}{\Longrightarrow}} \delta Y u \underset{rm}{\Longrightarrow} \delta\alpha X\beta u$ mit $\gamma' = \delta\alpha$. Dann ist $\gamma = \gamma' X$ ein zuverlässiges Präfix und q ein gültiges Item für γ.

" $\Leftarrow$ "
Induktion über die Länge von γ.
$\gamma = \varepsilon$: Alle gültigen Items für das zuverlässige Präfix ε haben die Form $q = [X \to .\alpha]$ mit $S' \overset{*}{\Longrightarrow} X\beta$. Nach Konstruktion von $char(K_G)$ gilt dann:
$(q_c, \varepsilon) \overset{*}{\vdash}_{char(K_G)} (q, \varepsilon)$.
Induktionsvoraussetzung: Für alle zuverlässigen Präfixe γ der Länge kleiner als n und alle für sie gültigen Items q gelte die Behauptung. Wir betrachten einen beliebigen zuverlässigen Präfix γX der Länge n und ein beliebiges gültiges Item q für γX. Sei $q = [A \to \beta_1.\beta_2]$. Es gibt eine Rechtsableitung $S \underset{rm}{\overset{*}{\Longrightarrow}} \alpha Aw \underset{rm}{\Longrightarrow} \alpha\beta_1\beta_2 w$ und $\gamma X = \alpha\beta_1$. Dann haben wir zwei Fälle zu betrachten.

1. Fall: $\beta_1 = \gamma' X$ d.h. $\gamma = \alpha\gamma'$. Dann ist $[A \to \gamma'.X\beta_2]$ gültig für das zuverlässige Präfix γ. Nach Konstruktion von $char(K_G)$ und Induktionsvoraussetzung gilt dann
$(q_c, \gamma X) \overset{*}{\vdash}_{char(K_G)} ([A \to \gamma'.X\beta_2], X) \vdash_{char(K_G)} ([A \to \gamma' X.\beta_2], \varepsilon)$.

2. Fall: $\beta_1 = \varepsilon, \alpha = \gamma X$. In der Rechtsableitung $S' \underset{rm}{\overset{*}{\Longrightarrow}} \alpha Aw$ betrachten wir den Schritt, in dem dieses Vorkommen von X eingeführt wurde. Dieser hatte die Form $S' \underset{rm}{\overset{*}{\Longrightarrow}} \mu By \underset{rm}{\Longrightarrow} \mu\nu X\rho y$ mit $\mu\nu = \gamma$. Jeder spätere Schritt in $\mu\nu X\rho y \underset{rm}{\overset{*}{\Longrightarrow}} \alpha Aw$ ersetzt nur Nichtterminale links von X. Dann ist $[B \to \nu.X\rho]$ ein gültiges Item für das zuverlässige Präfix $\gamma = \mu\nu$. Nach Induktionsvoraussetzung und Konstruktion von $char(K_G)$ gilt:
$(q_c, \gamma X) \overset{*}{\vdash}_{char(K_G)} ([B \to \nu.X\rho], X) \vdash_{char(K_G)} ([B \to \nu X.\rho], \varepsilon) \overset{*}{\vdash} ([A \to \beta_1.\beta_2], \varepsilon)$,
wobei in den letzten Schritten Expansionsübergänge gemacht wurden. □

Korollar 8.4.2.1
Die Sprache der zuverlässigen Präfixe einer kontextfreien Grammatik ist regulär.

Beweis:
Faßt man alle Zustände in $char(K_G)$ als Endzustände auf, so erhält man einen endlichen Automaten, der nach Satz 8.4.2 die Sprache der zuverlässigen Präfixe akzeptiert. □

LR-DEA(G)

In Kapitel 7 wurde schon ein Verfahren vorgestellt, nämlich der Algorithmus **NEA → DEA**, welches aus einem nichtdeterministischen endlichen Automaten einen äquivalenten deterministischen endlichen Automaten erzeugt. Dieser deterministische endliche Automat verfolgt alle Pfade parallel, die der nichtdeterministische für eine Eingabe durchlaufen könnte. Seine Zustände sind Mengen von

Zuständen des nichtdeterministischen Automaten. Diese sogenannte Teilmengenkonstruktion wenden wir jetzt auf den charakteristischen endlichen Automaten $char(K_G)$ des Item-Kellerautomaten K_G einer kontextfreien Grammatik G an.

Definition 8.4.4 (LR-DEA)
Der sich aus der Anwendung des Algorithmus **NEA → DEA** auf $char(K_G)$ ergebende deterministische endliche Automat $(Q_d, V_N \cup V_T, \delta_d, q_d, F_d)$ wird **LR-DEA(G)** genannt. □

Beispiel 8.4.5
Der LR-DEA(G_0), wobei G_0 die Grammatik aus Beispiel 8.2.2 ist, ergibt sich durch Anwendung des Algorithmus **NEA → DEA** auf $char(K_{G_0})$, welcher in Abbildung 8.18 dargestellt ist. LR-DEA(G_0) ist in Abbildung 8.19 angegeben. □

Nun möchten wir einige interessante Eigenschaften des LR-DEA(G) zu einer kontextfreien Grammatik G auflisten.

Satz 8.4.3
Sei γ ein zuverlässiges Präfix, und sei $p(\gamma) \in Q_d$ der eindeutig bestimmte Zustand, in den LR-DEA(G) unter Lesen von γ aus dem Anfangszustand übergeht, d.h. $(q_d, \gamma) \overset{*}{\vdash}_{LR-DEA(G)} (p(\gamma), \varepsilon)$. Dann gelten die folgenden Behauptungen:

(a) $p(\varepsilon) = q_d$

(b) $p(\gamma) = \{q \in Q_c \mid (q_c, \gamma) \overset{*}{\vdash}_{char(K_G)} (q, \varepsilon)\}$

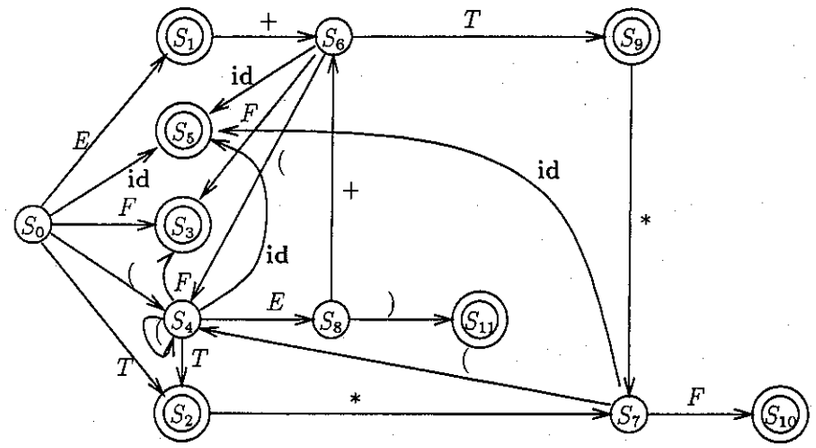
(c) $p(\gamma) = \{i \in It_G \mid i \text{ gültig für } \gamma\}$

(d) *Sei Γ die (i.a. unendliche) Menge aller zuverlässigen Präfixe von G. Dann definiert die Abbildung $p : \Gamma \to Q_d$ eine endliche Partition auf Γ.*

(e) *$L(LR\text{-}DEA(G))$ ist die Menge der zuverlässigen Präfixe von G, die mit einem Griff enden.* □

Beweis:
zu (a): Dies folgt aus der Konstruktion von $q_d = \{q \in Q_c \mid (q_0, \varepsilon) \overset{*}{\vdash}_{char(K_G)} (q, \varepsilon)\}$
zu (b): Dies ist eine Eigenschaft der Teilmengenkonstruktion. Sie wird über Induktion bewiesen.
zu (c): Diese Behauptung folgt aus (b) zusammen mit Satz 8.4.2.
zu (d): trivial.
zu (e): Die Endzustände von LR-DEA(G) enthalten mindestens ein vollständiges Item. Jedes solche vollständige Item bestimmt einen Griff am Ende des Präfixes. □

Rufen wir uns ins Gedächtnis zurück, was es heißt, daß ein Item gültig für ein zuverlässiges Präfix ist. Zuverlässige Präfixe sind Präfixe von Rechtssatzformen, wie sie während der Reduktion eines Eingabewortes auftreten. Wenn in ihnen eine Reduktion möglich ist, die wieder zu einer Rechtssatzform führt, dann nur am äußersten rechten Ende. Ein für solch ein zuverlässiges Präfix gültiges Item beschreibt eine mögliche Sicht der aktuellen Analysesituation.



$S_0 = \{ [S \to .E], [E \to .E+T], [E \to .T], [T \to .T*F], [T \to .F], [F \to .(E)], [F \to .\mathbf{id}] \}$

$S_1 = \{ [S \to E.], [E \to E.+T] \}$

$S_2 = \{ [E \to T.], [T \to T.*F] \}$

$S_3 = \{ [T \to F.] \}$

$S_4 = \{ [F \to (.E)], [E \to .E+T], [E \to .T], [T \to .T*F] [T \to .F] [F \to .(E)] [F \to .\mathbf{id}] \}$

$S_5 = \{ [F \to \mathbf{id}.] \}$

$S_6 = \{ [E \to E+.T], [T \to .T*F], [T \to .F], [F \to .(E)], [F \to .\mathbf{id}] \}$

$S_7 = \{ [T \to T*.F], [F \to .(E)], [F \to .\mathbf{id}] \}$

$S_8 = \{ [F \to (E.)], [E \to E.+T] \}$

$S_9 = \{ [E \to E+T.], [T \to T.*F] \}$

$S_{10} = \{ [T \to T*F.] \}$

$S_{11} = \{ [F \to (E).] \}$

Abb. 8.19: Der LR-DEA(G_0), der sich aus dem $char(K_{G_0})$ in Abbildung 8.18 ergibt

Satz 8.4.3 besagt jetzt gerade, daß die Zustände des LR-DEA(G) so konstruiert sind, daß sie die Menge der zuverlässigen Präfixe in endlich viele disjunkte Teilmengen zerlegen, und daß der einem zuverlässigen Präfix γ zugeordnete Zustand $p(\gamma)$ aus allen für γ gültigen Items besteht, d.h. allen möglichen Beschreibungen der aktuellen Analysesituation.

Beispiel 8.4.6
$\gamma = E+F$ ist ein zuverlässiges Präfix von G_0. Dem Zustand $p(\gamma) = S_3$ sind auch die folgenden zuverlässigen Präfixe zugeordnet:
$F,\ (F,\ ((F,\ (((F, \ldots$
$T*(F,\ T*((F,\ T*(((F, \ldots$
$E+F,\ E+(F,\ E+((F, \ldots$ □

Beispiel 8.4.7 Man betrachte Zustand S_6 im LR-DEA(G_0). Er enthält genau alle gültigen Items für das zuverlässige Präfix $E+$, nämlich die Items $[E \to E+.T], [T \to .T*F], [T \to .F], [F \to .\mathbf{id}], [F \to .(E)]$.
Denn $E+$ ist Präfix der Rechtssatzform $E+T$;

$$S \underset{rm}{\Longrightarrow} E \underset{rm}{\Longrightarrow} \underset{\uparrow}{E+T} \underset{rm}{\Longrightarrow} \underset{\uparrow}{E+F} \underset{rm}{\Longrightarrow} \underset{\uparrow}{E+\mathbf{id}}$$

also sind z.B. $[E \to E+.T]$ $[T \to .F]$ $[F \to .\mathbf{id}]$

gültig. □

Der LR-DEA(G) zu einer kontextfreien Grammatik G akzeptiert als deterministischer endlicher Automat, wie Satz 8.4.3(e) besagt, eine Sprache von zuverlässigen Präfixen von G. Wir können ihn aber wieder als Beschreibung eines Kellerautomaten auffassen und zwar folgendermaßen: $K_0 = (\Gamma, V_T, \Delta, q_0, \{q_f\})$ sei der folgende Kellerautomat.

Γ, das Kelleralphabet, ist die Menge Q_d der Zustände des LR-DEA(G). Jeder Zustand besteht aus einer Menge von kontextfreien Items von G.

$q_0 = q_d$ ist der Anfangszustand; mit ihm wird der Keller von K_0 initialisiert.

q_f ist der Endzustand des LR-DEA(G), der das Item $[S' \to S.]$ enthält.

$\Delta \subseteq \Gamma^* \times (V_T \cup \{\varepsilon\}) \times \Gamma^*$ ist die Übergangsrelation. Sie ist definiert durch:

(Lesen)
$(q, a, q\delta_d(q,a)) \in \Delta$, falls $\delta_d(q,a)$ definiert ist. In diesem Übergang werden das nächste Eingabesymbol a gelesen und der Nachfolgezustand von q unter a gekellert. Er ist nur dann möglich, wenn mindestens ein Item der Form $[X \to \cdots .a \cdots]$ in q vorhanden ist.
(Reduzieren)
$(qq_1 \ldots q_n, \varepsilon, q\delta_d(q,X)) \in \Delta$, falls $[X \to \alpha.] \in q_n$, $|\alpha| = n$. Das vollständige Item $[X \to \alpha.]$ in dem obersten Kellereintrag signalisiert eine mögliche Reduktion. Daraufhin werden soviele Einträge aus dem Keller entfernt, wie die rechte Seite

lang ist. Danach wird der X-Nachfolger des neuen obersten Kellereintrags gekellert. In Abbildung 8.20 wird ein Ausschnitt aus dem Übergangsdiagramm des LR-DEA(G) gezeigt, der diese Situation widerspiegelt. Dem α-Weg im Übergangsdiagramm entsprechen $|\alpha|$ Einträge oben auf dem Keller. Diese Einträge werden bei der Reduktion entfernt. Der darunterliegende neue oberste Zustand hat einen Übergang unter X, der jetzt durchlaufen wird.

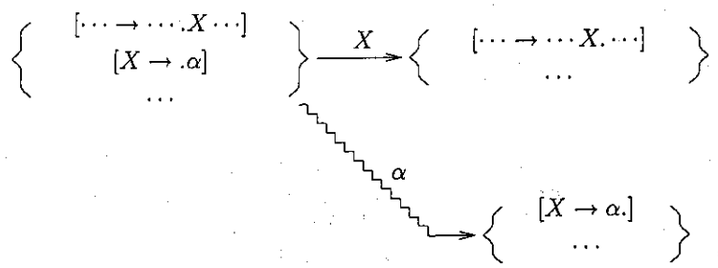


Abb. 8.20: Ausschnitt aus dem Übergangsdiagramm des LR-DEA(G)

Der Sonderfall $[X \to \varepsilon.]$ verdient Beachtung. Nach obiger Darstellung wären bei einer Reduktion $|\varepsilon| = 0$ oberste Kellereinträge zu entfernen, und aus dem neuen (wie alten) aktuellen Zustand wäre ein Übergang unter X zu machen. Dies ist auch genau so; denn mit jedem Item $[\cdots \to \cdots .X \cdots]$ ist auch das vollständige Item $[X \to \varepsilon.]$ im gleichen Zustand. Aus diesem Zustand ist ein Übergang unter X möglich.

Der Kellerinhalt, also die Folge der gekellerten Zustände des LR-DEA(G), bestimmt eindeutig ein zugehöriges zuverlässiges Präfix; denn nach der Konstruktion des LR-DEA(G) hat jeder seiner Zustände genau ein „Eingangssymbol", d.h. ein Symbol unter dem alle Übergänge in ihn erfolgen. Also kann man zu einem Kellerinhalt $q_0, \ldots, q_n$ mit $q_0 = q_d$ eindeutig ein Wort $\alpha = X_1 \ldots X_n \in (V_N \cup V_T)^*$ ablesen, so daß $\delta_d(q_i, X_{i+1}) = q_{i+1}$ ist. α ist ein zuverlässiges Präfix und q_n ist der α zugeordnete Zustand, der alle für α gültigen Items enthält.

Wann ist der so beschriebene Kellerautomat K_0 nichtdeterministisch? Offensichtlich, wenn ein Zustand q

- sowohl einen Leseübergang unter einem Symbol $a \in V_T$ als auch einen Reduzierübergang hat (**shift-reduce-Konflikt**), oder
- zwei verschiedene Reduzierübergänge (gemäß zweier verschiedener Produktionen) hat (**reduce-reduce-Konflikt**).

Im ersten Fall gibt es mindestens ein Lese-Item $[X \to \alpha.a\beta]$ und mindestens ein vollständiges Item $[Y \to \gamma.]$ in q, im zweiten zwei verschiedene vollständige Items $[Y \to \alpha.]$, $[Z \to \beta.]$.

Definition 8.4.5 (ungeeignete Zustände)
Sei $(Q_d, V_N \cup V_T, \Delta, q_d, \{q_f\})$ der LR-DEA(G) zu einer kontextfreien Grammatik G. Ein Zustand $q \in Q_d$ heißt **ungeeignet**, wenn er einen shift-reduce- oder einen reduce-reduce-Konflikt enthält. □

Ein geeigneter Zustand enthält also entweder nur unvollständige Items, oder er besteht aus genau einem vollständigen Item.

Ungeeignete Zustände machen einen LR-DEA(G) also nichtdeterministisch. Wir werden im folgenden Parser entwickeln, die durch Vorausschauen in die restliche Eingabe die in ungeeigneten Zuständen zu wählende Aktion deterministisch bestimmen.

Beispiel 8.4.8
Der LR-DEA(G) in Abbildung 8.19 hat drei ungeeignete Zustände, nämlich die Zustände S_1, S_2 und S_9. Im Zustand S_1 kann man E zu S reduzieren (vollständiges Item $[S \to E.]$) oder ein "+" lesen (shift-Item $[E \to E. + T]$); in S_2 kann man T zu E reduzieren (vollständiges Item $[E \to T.]$) oder ein "$*$" lesen (shift-Item $[T \to T. * F]$); in S_9 schließlich kann der Parser $E + T$ zu E reduzieren (vollständiges Item $[E \to E + T.]$) oder ein "$*$" lesen (shift-Item $[T \to T. * F]$). □

Direkte Konstruktion des LR-DEA(G)

Der LR-DEA(G) zu einer kontextfreien Grammatik G muß nicht über den Itemkellerautomaten K_G, dessen charakteristischen endlichen Automaten $char(K_G)$ und die Teilmengenkonstruktion erstellt werden. Er läßt sich aus G mithilfe des folgenden Algorithmus direkt erzeugen:

Algorithmus LR-DEA:
Eingabe: kontextfreie Grammatik $G = (V'_N, V_T, P', S')$
Ausgabe: LR-DEA(G) $= (Q_d, V_N \cup V_T, q_d, \delta_d, F_d)$
Methode: Die Zustände und Übergänge des LR-DEA(G) werden schrittweise mithilfe der folgenden drei Hilfsfunktionen *Start, Abschluss* und *Nachf* konstruiert.

var q, q': **set of item**;
Q_q: **set of set of item**;
δ_d: **set of item** $\times (V_N \cup V_T) \to$ **set of item**;

function *Start:* **set of item**;
return($\{[S' \to .S]\}$);
($*$ wenn S' das neue und S das alte Startsymbol von G sind $*$)

function *Abschluss*(s : **set of item**) : **set of item**;
($*$ entspricht den ε-Folgezuständen aus Def. 7.2.7 $*$)

```
begin
  q := S;
  while exist. [X → α.Yβ] in q and Y → γ in P
        and [Y → .γ] not in q do
    füge [Y → .γ] zu q hinzu
  od;
  return(q)
end;

function Nachf(s : set of item, Y : V_N ∪ V_T) : set of item;
    (* entspricht den (L)-Übergängen in K_G *)
  return({[X → αY.β] | [X → α.Yβ] ∈ s});

begin
  Q_d := {Abschluss(Start)};
  δ_d := ∅;
  foreach q in Q_d and X in V_N ∪ V_T do[2]
    let q' = Abschluss(Nachf(q, X)) in
      if q' ≠ ∅
      then
        if q' not in Q_d
        then Q_d := Q_d ∪ {q'}
        fi;
        δ_d := δ_d ∪ {q --X--> q'}
      fi
    tel
  od
end
```

8.4.4 LR(k): Definition, Eigenschaften, Beispiele

Sei $S' = \alpha_0 \underset{rm}{\Longrightarrow} \alpha_1 \underset{rm}{\Longrightarrow} \alpha_2 \cdots \underset{rm}{\Longrightarrow} \alpha_m = v$ eine beliebige Rechtsableitung zu einer kontextfreien Grammatik G. Wir werden G eine LR(k)-Grammatik nennen, wenn in jeder solchen Rechtsableitung und jeder darin auftretenden Rechtssatzform α_i

- der Griff lokalisiert werden kann, und
- die anzuwendende Produktion bestimmt werden kann,

indem man α_i von links bis höchstens k Symbole hinter dem Griff betrachtet. In einer LR(k)-Grammatik ist also die Aufteilung von α_i in $\gamma\beta w$ und die Bestimmung von $X \to \beta$, so daß $\alpha_{i-1} = \gamma X w$ ist, eindeutig durch $\gamma\beta$ und $k : w$ bestimmt.

[2] Die Semantik der foreach-Anweisung sei so, daß jedes Element der (sich dynamisch erweiternden) Menge Q_d genau einmal mit jedem Element von $V_N \cup V_T$ kombiniert wird.

Definition 8.4.6 (LR(k)-Grammatik)
Sei $G' = (V'_N, V_T, P', S')$ die um das neue Startsymbol S' und die zusätzliche Produktion $S' \to S$ erweiterte kontextfreie Grammatik zu einer kontextfreien Grammatik $G = (V_N, V_T, P, S)$. G' heißt **LR(k)-Grammatik**, wenn aus

$S' \underset{rm}{\overset{*}{\Longrightarrow}} \alpha X w \underset{rm}{\Longrightarrow} \alpha\beta w$ und

$S' \underset{rm}{\overset{*}{\Longrightarrow}} \gamma Y x \underset{rm}{\Longrightarrow} \alpha\beta y$ und

$k : w = k : y$ folgt, daß

$\alpha = \gamma$ und $X = Y$ und $x = y$. □

Beispiel 8.4.9
Sei G die Grammatik mit den Produktionen

$S \to A \mid B$
$A \to aAb \mid 0$
$B \to aBbb \mid 1$

$L(G) = \{a^n 0 b^n \mid n \geq 0\} \cup \{a^n 1 b^{2n} \mid n \geq 0\}$. Wir wissen schon, daß G nicht LL(k) für beliebiges k ist.
G ist LR(0)-Grammatik. Die Rechtssatzformen von G haben die Formen (der Griff ist jeweils unterstrichen): $S, \underline{A}, \underline{B}, a^n\underline{aAb}b^n$, $a^n\underline{aBbb}b^{2n}$, $a^n\underline{a0}bb^n$, $a^n\underline{a1}bbb^{2n}$. Nur im Falle der Rechtssatzformen a^naAbb^n und a^naBbbb^{2n} gäbe es jeweils zwei verschiedene mögliche Reduktionen. Man könnte a^naAbb^n zu a^nAb^n und zu a^naSbb^n reduzieren. Die erste gehört zu der Rechtsableitung $S \underset{rm}{\overset{*}{\Longrightarrow}} a^nAb^n \underset{rm}{\Longrightarrow} a^naAbb^n$, die zweite gehört nicht zu einer Rechtsableitung. Aus dem Präfix a^n von a^nAb^n ergibt sich eindeutig, ob A Griff ist, nämlich im Fall $n = 0$, oder ob aAb Griff ist, nämlich im Fall $n > 0$. Die Rechtssatzformen a^nBb^{2n} werden analog behandelt. □

Beispiel 8.4.10
Die Grammatik G_1 mit den Produktionen

$S \to aAc$
$A \to Abb \mid b$

und der Sprache $L(G_1) = \{ab^{2n+1}c \mid n \geq 0\}$ ist LR(0)-Grammatik. In einer Rechtssatzform $aAbbb^{2n}c$ gibt es nur die Reduktion zu $aAb^{2n}c$ als Teil einer Rechtsableitung. Der Präfix $aAbb$ bestimmt dies eindeutig. Für die Rechtssatzform $abb^{2n}c$ gilt, b ist der Griff, und der Präfix ab bestimmt dies eindeutig. □

Beispiel 8.4.11
Die Grammatik G_2 mit den Produktionen

$S \to aAc$
$A \to bbA \mid b$

mit der Sprache $L(G_2) = L(G_1)$ ist LR(1)-Grammatik. Die kritischen Rechtssatzformen haben die Form ab^nw. Falls $1 : w = b$, so liegt der Griff in w; falls $1 : w = c$, so bildet das letzte b in b^n den Griff. □

Beispiel 8.4.12
Die Grammatik G_3 mit den Produktionen
$S \to aAc$
$A \to bAb \mid b$
und mit $L(G_3) = L(G_1)$ ist nicht LR(k)-Grammatik für beliebiges k. Denn sei k beliebig, aber fest gewählt. Man betrachte die zwei Rechtsableitungen
$S \underset{rm}{\overset{*}{\Longrightarrow}} ab^nAb^nc \underset{rm}{\Longrightarrow} ab^nbb^nc$
$S \underset{rm}{\overset{*}{\Longrightarrow}} ab^{n+1}Ab^{n+1}c \underset{rm}{\Longrightarrow} ab^{n+1}bb^{n+1}c \quad \text{mit } n \geq k$
Hier sind mit den Bezeichnungen aus Definition 8.4.6: $\alpha = ab^n, \beta = b, \gamma = ab^{n+1}, w = b^nc, y = b^{n+2}c$. Es gilt $k : w = k : y = b^k$. Aus $\alpha \neq \gamma$ folgt, daß G_3 keine LR(k)-Grammatik ist. □

Satz 8.4.4
Eine kontextfreie Grammatik G ist genau dann eine LR(0)-Grammatik, wenn LR-DEA(G) keine ungeeigneten Zustände hat.

Beweis:
" $\Rightarrow$ "
Sei G eine LR(0)-Grammatik;
Annahme: LR-DEA(G) hat einen ungeeigneten Zustand p.

Fall 1: p hat einen reduce-reduce-Konflikt; dann hat p mindestens zwei verschiedene reduce-Items $[X \to \beta.]$, $[Y \to \delta.]$. p zugeordnet ist eine nichtleere Menge von zuverlässigen Präfixen. Sei γ ein solches zuverlässiges Präfix. Beide reduce-Items sind gültig für $\gamma\beta$; d.h. es gibt zwei verschiedene Rechtsableitungen
$S' \underset{rm}{\overset{*}{\Longrightarrow}} \gamma Xw \underset{rm}{\Longrightarrow} \gamma\beta w$ und
$S' \underset{rm}{\overset{*}{\Longrightarrow}} \nu Yy \underset{rm}{\Longrightarrow} \nu\delta y$ mit $\nu\delta = \gamma\beta$.
Das ist aber ein Widerspruch zur LR(0)-Eigenschaft.

Fall 2: p hat einen shift-reduce-Konflikt; der Widerspruch ergibt sich analog.

" $\Leftarrow$ "
LR-DEA(G) hat keine ungeeigneten Zustände.
Man betrachte die beiden Rechtsableitungen
$S' \underset{rm}{\overset{*}{\Longrightarrow}} \alpha Xw \underset{rm}{\Longrightarrow} \alpha\beta w$
$S' \underset{rm}{\overset{*}{\Longrightarrow}} \gamma Yx \underset{rm}{\Longrightarrow} \alpha\beta y$. Zu zeigen ist, daß $\alpha = \gamma$, $X = Y$, $x = y$ gelten.
LR-DEA(G) kommt nach Lesen von $\alpha\beta$ in einen Zustand p. p ist nicht ungeeignet. Damit ist $p = \{[X \to \beta.]\}$. p enthält alle für $\alpha\beta$ gültigen Items. Also ist $\alpha = \gamma$, $X = Y$ und $x = y$. □

Damit haben wir folgende Zusammenhänge erkannt. Ausgehend von einer kontextfreien Grammatik G konnten wir ihren LR-DEA(G) konstruieren, entweder direkt oder auf dem Umweg über den Item-Kellerautomaten K_G. Dieser

LR-DEA(G) beschreibt eindeutig das Verhalten eines Kellerautomaten K_0. K_0 ist deterministisch, wenn LR-DEA(G) keine ungeeigneten Zustände enthält. Der Satz 8.4.4 besagt, daß dies der Fall ist, wenn die Ausgangsgrammatik G eine LR(0)-Grammatik ist. Für den in der Praxis selten vorkommenden Fall der LR(0)-Grammatiken haben wir damit ein Parsergenerierungsverfahren kennengelernt.

Wir wenden uns nun den relevanteren Fällen $k > 0$ zu. Im LR(0)-Parser gibt jeweils der aktuelle Zustand, eine Menge von kontextfreien Items, die nächste Aktion an, also Lesen oder Reduzieren. LR(k)-Parser haben ebenfalls Zustände, die aus Mengen von Items bestehen, allerdings aus LR(k)-Items, das sind kontextfreie Items erweitert um eine Menge von Wörtern der Länge k.

Definition 8.4.7 (LR(k)-Item)
Sei G' eine erweiterte kontextfreie Grammatik. $[X \to \alpha_1.\alpha_2, L]$ heißt **LR(k)-Item** von G', wenn $X \to \alpha_1\alpha_2 \in P$ und $L \subseteq V_{T\#}^{\leq k}$ ist. $[X \to \alpha_1.\alpha_2]$ ist der **Kern** des LR(k)-Items $[X \to \alpha_1.\alpha_2, L]$, L seine **Vorausschaumenge**. Dieses LR(k)-Item ist **gültig** für ein zuverlässiges Präfix $\alpha\alpha_1$, wenn es für alle $u \in L$ eine Rechtsableitung $S'\# \underset{rm}{\overset{*}{\Longrightarrow}} \alpha Xw \underset{rm}{\Longrightarrow} \alpha\alpha_1\alpha_2 w$ gibt mit $u = k : w$. □

Somit können wir die bisher betrachteten kontextfreien Items als LR(0)-Items betrachten, wenn wir $[X \to \alpha_1.\alpha_2, \{\varepsilon\}]$ mit $[X \to \alpha_1.\alpha_2]$ identifizieren.

Beispiel 8.4.13
Betrachten wir wieder die Grammatik G_0.
(1) $[E \to E+.T, \{), +\}]$ ist ein gültiges LR(1)-Item für ($E+$ in G_0
(2) $[E \to T., \{*\}]$ ist kein gültiges LR(1)-Item für irgendein zuverlässiges Präfix
denn:
(1) $S' \underset{rm}{\overset{*}{\Longrightarrow}} (E) \underset{rm}{\Longrightarrow} (E+T) \underset{rm}{\overset{*}{\Longrightarrow}} (E+T+\mathbf{id})$ wobei $\alpha = (, \alpha_1 = E+, \alpha_2 = T, u = +, w = +\mathbf{id})$
(2) In keiner Rechtssatzform kann das Teilwort $E*$ auftreten. □

Satz 8.4.5
Eine kontextfreie Grammatik G ist genau dann eine LR(k)-Grammatik , wenn gilt:
Sei $\alpha\beta$ ein zuverlässiges Präfix. Wenn das LR(k)-Item $[X \to \beta., L_1]$ für $\alpha\beta$ gültig ist, dann gibt es kein anderes für $\alpha\beta$ gültiges LR(k)-Item $[Y \to \beta_1.\beta_2, L_2]$, so daß $L_1 \cap FIRST_k(\beta_2 L_2) \neq \emptyset$. □

Bemerken Sie, daß das Item $[Y \to \beta_1.\beta_2]$ für das zuverlässige Präfix $\alpha\beta$ gültig sein kann, ohne daß $\beta_1 = \beta$ ist. β_1 kann ein echter Suffix von β sein oder sich vom Ende von β bis in α hinein erstrecken. Diesen beiden Fällen entsprechen jeweils andere Zerlegungen von $\alpha\beta$.

8.4.5 LR(k)-Parser

Die LR(k)-Definition besagt Folgendes: Hat man beim Lesen einer Rechtssatzform einen Kandidaten für eine Reduktion gefunden, so kann man mithilfe des dazugehörigen zuverlässigen Präfixes und zusätzlich der k nächsten Symbole der Eingabe entscheiden, ob dieser Kandidat tatsächlich der Griff ist oder nicht und, wenn ja, wozu er reduziert werden muß. Wenn wir alle Kombinationen aus zuverlässigen Präfixen und Wörtern der Länge k tabellieren wollten, hätten wir Schwierigkeiten, weil es i.a. unendlich viele zuverlässige Präfixe gibt. Satz 8.4.3 besagt jedoch, daß die Zustände des LR-DEA(G) eine endliche Partition auf dieser Menge der zuverlässigen Präfixe induzieren. Jedes zuverlässige Präfix γ gehört genau zu einem Zustand des LR-DEA(G), nämlich dem, in den der LR-DEA(G) übergeht, wenn er, beginnend in seinem Anfangszustand, γ liest. Dieser zu γ gehörende Zustand zusammen mit den nächsten k Symbolen reicht aus, um die Entscheidung zu treffen, ob weitergelesen oder reduziert werden soll. Also wird eine Tabelle eines LR(k)-Parsers, die sogenannte **action-Tabelle** für jede Kombination von Zustand und k Eingabesymbolen eine der folgenden Aktionen festlegen:

shift:	lies das nächste Eingabesymbol;
reduce ($X \to \alpha$):	reduziere mittels der Produktion $X \to \alpha$;
error:	melde Fehler und
accept:	melde erfolgreiches Ende des Parserlaufs.

Eine zweite Tabelle, genannt **goto-Tabelle**, enthält die Darstellung der Übergangsfunktion des LR-DEA(G). Sie beschreibt also die Übergänge zwischen Zuständen unter Terminalen und Nichtterminalen der Grammatik. Sie wird konsultiert, wenn eine *shift*-Aktion oder eine *reduce*-Aktion passiert ist, um den neuen aktuellen Zustand zu berechnen. Bei einem *shift* bestimmt sie den Übergang aus dem aktuellen Zustand unter dem gelesenen Symbol; bei einer Reduktion mittels $X \to \alpha$ den Übergang unter X aus dem Zustand, der nach Entfernen der zu α gehörenden Kellerzustände oben auf dem Keller liegt. Diese beiden Tabellen sind in Abbildung 8.21 dargestellt.

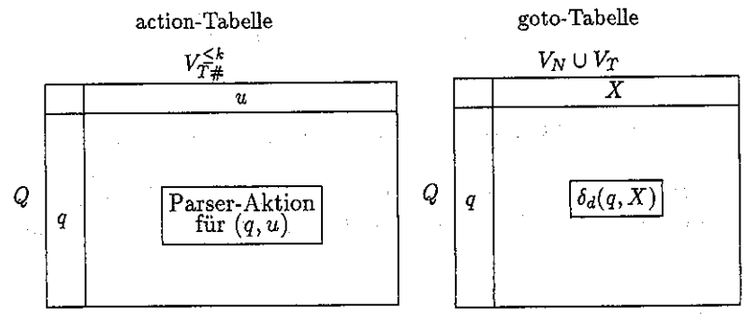


Abb. 8.21: LR(k)-Parser-Kontrolle, bestehend aus action-Tabelle und goto-Tabelle. Die Zustandsmenge des LR(k)-Parsers sei Q.

Man sieht, daß der aktuelle Zustand, der oberste im Keller, nicht nur alle wesentliche Information über das augenblickliche zuverlässige Präfix enthält, sondern auch, daß man diesen Zustand nicht bei jedem Wechsel von einem aktuellen zuverlässigen Präfix γ zum nächsten, γ', durch eine Analyse von γ' durch den LR-DEA(G) neu berechnen muß. Der neue Zustand für γ' ergibt sich durch lokale Berechnungen am oberen Kellerende.

Ein LR(k)-Parser für eine LR(k)-Grammatik G besteht aus den beiden Tabellen, die auf die anschließend beschriebene Weise aus G erzeugt werden, und einem Programm, welches diese Tabellen interpretiert. Dieses Programm ist von G unabhängig, kann also für alle LR(k)-Grammatiken benutzt werden. Es heißt hier **LR(k)-Parser**.

Algorithmus LR(k)-PARSER:

```
type  state = set of item;
var   lookahead: seq of symbol;
                    (* die nächsten k lexikalisch analysierten,
                       aber noch nicht konsumierten Eingabesymbole *)
      S : stack of state; proc scan;
      (* analysiert ein weiteres Symbol lexikalisch,
         fügt es hinten an lookahead an *)
proc  acc;
      (* melde erfolgreiches Ende der syntaktischen Analyse; halte *)
proc  err(meldung: string);
      (* melde Fehler; halte *)
scan^k;
push(S, q0);
forever do
   case action[top(S), lookahead] of
      shift: begin   push(S, goto[top(S), hd(lookahead)]);
                     lookahead := tl(lookahead);
                     scan
             end;
      reduce (X → α) : begin   pop^|α|(S); push(S, goto[top(S), X]);
                               output("X → α")
                       end;
      accept:  acc;
      error:   err("...");
   end case
od
```

Die Konstruktion von LR(k)-Parsern

Im folgenden werden wir drei verschiedene Arten kennenlernen, aus einer kontextfreien Grammatik einen LR(k)-Parser für die von ihr definierte Sprache zu