

# Automaten und Formale Sprachen

SoSe 2007 in Trier

Henning Fernau

Universität Trier

fernau@uni-trier.de

# Automaten und Formale Sprachen

Gesamtübersicht

- Organisatorisches
- Einführung
- Endliche Automaten und reguläre Sprachen
- **Kontextfreie Grammatiken und kontextfreie Sprachen**
- Chomsky-Hierarchie

# Kontextfreie Grammatiken und kontextfreie Sprachen

1. Automaten mit unendlichem Speicher
2. Kontextfreie Grammatiken
3. Normalformen
4. Nichtkontextfreie Sprachen
5. **Algorithmen für kontextfreie Grammatiken: Parsing**

## **Compiler(auf)bau:** Eine Übersicht der Phasen

1. Scanner: lexikalische Analyse (endliche Automaten mit Ausgabe)
2. Parser: syntaktische Analyse (kontextfreie Sprachen)
3. semantische Analyse
4. Codegenerierung
5. Optimierung

## Parsing

Linksableitung: Tiefensuche mit Linksabstieg durch Syntaxbaum

Rechtsableitung: Tiefensuche mit Rechtsabstieg durch Syntaxbaum

Ein *Linksparser* für Grammatik  $G(\Sigma, Q, R, s)$  liefert zu einem Wort  $w \in L(G)$  eine Linksableitung von  $w$ , beschrieben durch ein Wort über  $R$ , und NEIN, falls  $w \notin L(G)$ .

Analog: *Rechtsparser* liefert Rechtsableitung.

Hinweis: Es gibt *mehrdeutige* kfG, d.h., kfG, bei denen (manche) Wörter mehr als eine Linksableitung besitzen; dann liefert ein Linksparser irgendeine solche Linksableitung.

## Prädikative (Links-)Parser

Erinnerung: KfG  $\rightarrow$  Kellerautomat Konstruktion

Sei  $G = (\Sigma, N, R, S)$  eine kfG. Betrachte folgende Transitionen:

- $((s, \lambda, \lambda), (f, S))$ ,
- für jede Regel  $C \rightarrow w$ :  $((f, \lambda, C), (f, w))$ ,
- für jedes Terminalzeichen  $a$ :  $((f, a, a), (f, \lambda))$ .

$f$  ist der einzige Endzustand und  $s$  der Startzustand.

Hieraus **Linksparser**:

Der Linksparser simuliert im Wesentlichen den beschriebenen Kellerautomaten (und ist daher im Allgemeinen nichtdeterministisch):

Simuliert der Kellerautomat die kfG, so gibt der Parser die entsprechende Regel aus (*Produktionsschritt*).

Schritte der Form  $((f, a, a), (f, \lambda))$  heißen *Leseschritte (Shifts)* und führen zu keiner Ausgabe.

Im Wesentlichen kann der Linksparser die Zustandsinformation des Kellerautomaten ignorieren (Kellerautomaten-Normalformen!).

## Die Konstruktion am Beispiel

$G = (\{a, b\}, \{A, S\}, R, S)$  mit den Regeln  $r_1 = S \rightarrow ASb$ ,  $r_2 = A \rightarrow a$  und  $r_3 = S \rightarrow \lambda$ .

Der entsprechende Kellerautomat hat folgende Regeln:

$((s, \lambda, \lambda), (f, S))$ ,  $((f, \lambda, S), (f, aSb))$ ,  $((f, \lambda, A), (f, a))$ ,  $((f, \lambda, S), (f, \lambda))$ ,  
 $((f, a, a), (f, \lambda))$ ,  $((f, b, b), (f, \lambda))$ .

Die Ableitung  $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$  wird wie folgt simuliert:

1. Initialisierungsphase:

$(s, aabb, \lambda) \vdash (f, aabb, S)$

2. Simulieren der kfG (Produktionsschritt) und

3. Überprüfen der Eingabe (Leseschritt):

$(f, aabb, S) \vdash (f, aabb, ASb) \vdash (f, aabb, aSb) \vdash (f, abb, Sb) \vdash (f, abb, ASbb)$   
 $\vdash (f, abb, aSbb) \vdash (f, bb, Sbb) \vdash (f, bb, bb) \vdash (f, b, b) \vdash (f, \lambda, \lambda)$

## Die Arbeit des entsprechenden Linksparsers

Resteingabe	Kellerinhalt	Ausgabe
aabb	S	(Anfang)
aabb	ASb	r <sub>1</sub>
aabb	aSb	r <sub>2</sub>
abb	Sb	(Leseschritt)
abb	ASbb	r <sub>1</sub>
abb	aSbb	r <sub>2</sub>
bb	Sbb	(Leseschritt)
bb	bb	r <sub>3</sub>
b	b	(Leseschritt)
λ	λ	(Leseschritt und Akzeptieren)

Erhaltene Linksableitung:

$$S \Rightarrow^{r_1} ASB \Rightarrow^{r_2} aSb \Rightarrow^{r_1} aASbb \Rightarrow^{r_2} aaSbb \Rightarrow^{r_3} aabb$$

## Konflikte bei Linksparsern

Solange ein Terminalzeichen auf dem Keller liegt, muss ein Linksparser einen Leseschritt durchführen (stimmt das oberste Kellerzeichen nicht mit dem aktuellen Eingabezeichen überein, so NEIN).

Bei den Produktionsschritten kann es jedoch zu *Konflikten* kommen: Welche Produktion (Regel) ist anzuwenden (zu simulieren) und damit auszugeben ?

*Idee*: Möglicherweise hilft die Resteingabe weiter. . .

Im Beispiel: Konflikt zwischen  $r_1$  und  $r_3$ ;

aktuelles Eingabezeichen  $x = a \rightsquigarrow r_1, x = b \rightsquigarrow r_3$ ;

evtl. Sonderfall: Resteingabe ist leer; Konvention: man liest dann das *Eingabeendezeichen* \$.

## Greibach-Normalform

Eine kfG  $G = (\Sigma, N, R, S)$  ist in *Greibach-Normalform* gdw.  
 $R \subseteq (N \times \Sigma(N \setminus \{S\})^*) \cup (S \times \{\lambda\})$ .

Hinweis: Umformung Kellerautomat  $\rightarrow$  kfG liefert "fast" Greibach NF.

**Satz**: Jede kontextfreie Sprache besitzt eine sie erzeugende kfG in Greibach NF.  
(ohne Beweis)

Eine kfG  $G = (\Sigma, N, R, S)$  mit  $R \subseteq (N \times \Sigma(N \cup \Sigma)^*)$  heißt *simpel* oder *s-Grammatik* gdw.  $\forall a \in \Sigma : \#\{A \rightarrow a\beta \in R\} \leq 1$ .

**Einfach**: zu einer s-Grammatik eine äquivalente kfG in Greibach-NF angeben.

**Lemma**: Linksparser für s-Grammatiken arbeiten deterministisch.

**Ein Beispiel** für eine “sinnvolle” s-Grammatik mit  $\Sigma = \{\beta, \varepsilon, \alpha, ;\}$ :

$$S \rightarrow \beta C$$

$$C \rightarrow \varepsilon, C \rightarrow \alpha; C, C \rightarrow \beta C; C.$$

$\beta$ : “begin”;  $\varepsilon$ : “end”

Resteingabe	Kellerinhalt	Ausgabe
$\beta \dots$	S	$S \rightarrow \beta C$
$\varepsilon \dots$	C	$C \rightarrow \varepsilon$
$\alpha \dots$	C	$C \rightarrow \alpha; C$
$\beta \dots$	C	$C \rightarrow \beta C; C$

## Eine Ableitung in der Blockverschachtelungsgrammatik:

$$S \Rightarrow \beta C \Rightarrow \beta \beta C; C \Rightarrow \beta \beta \alpha; C; C \Rightarrow^2 \beta \beta \alpha; \varepsilon; \varepsilon$$

Arbeit des Linksparsers (deterministisch!):

Resteingabe	Kellerinhalt	Ausgabe
$\beta \beta \alpha; \varepsilon; \varepsilon$	$S$	(Anfang)
$\beta \beta \alpha; \varepsilon; \varepsilon$	$\beta C$	$S \rightarrow \beta C$
$\beta \alpha; \varepsilon; \varepsilon$	$C$	(Leseschritt)
$\beta \alpha; \varepsilon; \varepsilon$	$\beta C; C$	$C \rightarrow \beta C; C$
$\alpha; \varepsilon; \varepsilon$	$C; C$	(Leseschritt)
$\alpha; \varepsilon; \varepsilon$	$\alpha; C; C$	$C \rightarrow \alpha; C$
$\varepsilon; \varepsilon$	$C; C$	(Leseschritte)
$\varepsilon; \varepsilon$	$\varepsilon; C$	$C \rightarrow \varepsilon$
$\varepsilon$	$C$	(Leseschritte)
$\lambda$	$\lambda$	$C \rightarrow \varepsilon$ (Akzeptanz)

**Linksparser** heißen auch *Top-Down-Parser*

Der Ableitungsbaum wird von oben nach unten durchforstet.

**Problem:** Regelanwendungskonflikte.

**Mögliche Lösung:** Verzeichne zu jeder Regel  $A \rightarrow w$  die Menge der Eingabezeichen (oder allgemeiner Eingabewörter einer vorgegebenen Maximallänge  $k$ ), die als Präfix der Länge  $k$  für ein aus  $w$  ableitbares Terminalwort vorkommen können.

Tatsächlich wird die Situation noch verkompliziert durch mögliche  $\lambda$ -Regel; das betrachten wir hier nicht.

Auf dieser Idee fußend werden *LL(k)-Grammatiken* definiert.

**Eine Beispiel**-Grammatik mit  $\Sigma = \{\beta, \varepsilon, \alpha, ;, \iota, \theta, \eta, c, \$\}$ :

$S \rightarrow B (\beta), S \rightarrow I (\iota), S \rightarrow \alpha; S (\alpha), S \rightarrow \lambda (\$, \varepsilon, \theta)$

$B \rightarrow \beta S \varepsilon S$

$I \rightarrow \iota c \theta S \eta S \varepsilon S$

$\beta$ : “begin”;  $\varepsilon$ : “end”

$\iota$ : “if”;  $\theta$ : “then”;  $\eta$ : “else”

Hinweis: Also doch  $\lambda$ -Regeln !

## Ein Linksparser bei der Arbeit

$S \rightarrow B (\beta),$

$S \rightarrow I (\iota),$

$S \rightarrow a; S (a),$

$S \rightarrow \lambda (\$, \epsilon, \theta)$

$B \rightarrow \beta S \epsilon S$

$I \rightarrow \iota \theta S \eta S \epsilon S$

Resteingabe	Kellerinhalt	Ausgabe
$a; \iota \theta \beta a; \epsilon; \eta \epsilon \$$	$S$	(Anfang)
$a; \iota \theta \beta a; \epsilon; \eta \epsilon \$$	$a; S$	$S \rightarrow a; S$
$\iota \theta \beta a; \epsilon; \eta \epsilon \$$	$S$	(zwei Leseschritte)
$\iota \theta \beta a; \epsilon; \eta \epsilon \$$	$I$	$S \rightarrow I$
$\iota \theta \beta a; \epsilon; \eta \epsilon \$$	$\iota \theta S \eta S \epsilon S$	$I \rightarrow \iota \theta S \eta S \epsilon S$
$\beta a; \epsilon; \eta \epsilon \$$	$S \eta S \epsilon S$	(drei Leseschritte)
$\beta a; \epsilon; \eta \epsilon \$$	$B \eta S \epsilon S$	$S \rightarrow B$
$\beta a; \epsilon; \eta \epsilon \$$	$\beta S \epsilon S \eta S \epsilon S$	$B \rightarrow \beta S \epsilon S$
$a; \epsilon; \eta \epsilon \$$	$S \epsilon S \eta S \epsilon S$	(Leseschritt)
$a; \epsilon; \eta \epsilon \$$	$a; S \epsilon S \eta S \epsilon S$	$S \rightarrow a; S$
$\epsilon; \eta \epsilon \$$	$S \epsilon S \eta S \epsilon S$	(zwei Leseschritte)
$\epsilon; \eta \epsilon \$$	$\epsilon S \eta S \epsilon S$	$S \rightarrow \lambda$
$\eta \epsilon \$$	$S \eta S \epsilon S$	(Leseschritt)
$\eta \epsilon \$$	$\eta S \epsilon S$	$S \rightarrow \lambda$
$\epsilon \$$	$S \epsilon S$	(Leseschritt)
$\epsilon \$$	$\epsilon S$	$S \rightarrow \lambda$
$\$$	$S$	(Leseschritt)
$\$$		$S \rightarrow \lambda$
$\$$		(Akzeptanz)

## Parsen mit rekursivem Abstieg (für LL(1))

- Für jedes oberste Kellerzeichen  $X$  Extra-Prozedur  $P_X(\alpha u, K)$ , wobei  $\alpha u$  die restliche Eingabe ist, also  $\alpha$  das aktuelle Eingabezeichen, und  $K$  der Keller (ohne oberstes Zeichen  $X$ ).
- $P_b(\alpha u, K)$  für Terminalzeichen  $b$ : (1) liefert Fehlermeldung, falls  $b \neq \alpha$ ; (2) falls  $b = \alpha$ , wird das oberste Kellerzeichen entfernt; gilt nun (2A)  $\alpha = \$$  und  $u = K = \lambda$ , so akzeptiere; andernfalls ist (2B)  $K = \lambda$  ein Fehlerfall; für (2C)  $K \neq \lambda$  sei  $K = X'K'$  und rekursiv rufe  $P_{X'}(u, K')$  auf.
- In  $P_X(\alpha u, K)$  wird für ein Nichtterminalzeichen  $X$  nach verschiedenen Möglichkeiten für  $\alpha$  rekursiv verzweigt.  
(Hier könnte man auch noch tiefer in der Ausgabe vorausschauen, wenn LL(k) gefordert.)  
Das heißt, die betreffende Regel  $X \rightarrow w$  wird ausgeführt und der neue Keller erfüllt  $X'K' = wK$  für ein Zeichen  $X'$ , und rekursiv wird  $P_{X'}(\alpha u, K')$  aufgerufen.

Beobachte den Rekursionskeller!

Gefahr durch Linksrekursion in der Grammatik, d.h.:  $X \xRightarrow{*} X\alpha$ .

## Rechtsparser: bottom-up Simulation von Ableitungsbäumen

Sei  $G = (\Sigma, N, R, S)$  eine kfG.

Betrachte folgende Transitionen (Kellerautomat hat nur einen Zustand):

- für jede Regel  $C \rightarrow w$ :  $((f, \lambda, w), (f, C))$  (Reduktionsschritt),
- für jedes Terminalzeichen  $a$ :  $((f, a, \lambda), (f, a))$  (Leseschritt).

Der Kellerautomat akzeptiert bei leerer Eingabe und nur  $S$  auf dem Keller.

Es ist hier einfacher, beim Keller das oberste Zeichen “rechts” anzunehmen.

## Die Konstruktion am Beispiel

$G = (\{a, b\}, \{A, S\}, R, S)$  mit den Regeln  
 $r_1 = S \rightarrow ASb$ ,  $r_2 = A \rightarrow a$  und  $r_3 = S \rightarrow \lambda$ .

Der entsprechende Kellerautomat hat folgende Regeln:

$((f, \lambda, aSb), (f, S))$ ,  $((f, \lambda, a), (f, A))$ ,  $((f, \lambda, \lambda), (f, S))$ ,  
 $((f, a, \lambda), (f, a))$ ,  $((f, b, \lambda), (f, b))$ .

Die Ableitung  $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$  wird wie folgt simuliert:

**1. Simulieren der kfG (Reduktionsschritt)** und **2. Leseschritt:**

$(f, aabb, \lambda) \vdash (f, abb, a) \vdash (f, abb, A) \vdash (f, bb, Aa) \vdash (f, bb, AA)$   
 $\vdash (f, bb, AAS) \vdash (f, b, AASb) \vdash (f, b, AS) \vdash (f, \lambda, ASb) \vdash (f, \lambda, S)$

## Die Arbeit des entsprechenden Rechtsparsers

Resteingabe	Kellerinhalt	Ausgabe
aabb		(Anfang)
abb	a	(Leseschritt)
abb	A	r <sub>2</sub>
bb	Aa	(Leseschritt)
bb	AA	r <sub>2</sub>
bb	AAS	r <sub>3</sub>
b	AASb	(Leseschritt)
b	AS	r <sub>1</sub>
λ	ASb	(Leseschritt)
λ	S	(r <sub>1</sub> und Akzeptieren)

Erhaltene Rechtsableitung: (ACHTUNG: Rückwärtslesen der Parserausgabe!)

$S \Rightarrow^{r_1} ASb \Rightarrow^{r_1} AASbb \Rightarrow^{r_3} AAbb \Rightarrow^{r_2} Aabb \Rightarrow^{r_2} aabb$

## Konflikte bei Rechtsparsern

Es gibt zwei Arten von *Konflikten*:

Welche Produktion (Regel) unter mehreren ist anzuwenden (reduzierend zu simulieren) und damit auszugeben ? Reduktionsschritt oder (weiterer) Leseschritt ?

*Idee* wiederum: Möglicherweise hilft die Resteingabe weiter...

Im Beispiel: Konflikt zwischen  $r_1$  und  $r_3$ ;  
aktuelles Eingabezeichen  $x = \$$  oder  $x = b \rightsquigarrow r_1, x = b \rightsquigarrow r_3$ ;  
Hilft hier wohl nicht ...

*Mitteilung*: Schon LR(1)-Grammatiken kennzeichnen die deterministisch kontextfreien Sprachen und sind somit beschreibungsmächtiger als LL(k)-Grammatiken für beliebige k.  
Die entsprechende (kompliziertere) Theorie wird in der Vorlesung *Compilerbau* erläutert.