Grundlagen Theoretischer Informatik 2

WiSe 2011/12 in Trier

Henning Fernau
Universität Trier
fernau@uni-trier.de

Grundlagen Theoretischer Informatik 2 Gesamtübersicht

- Organisatorisches; Einführung
- Ersetzungsverfahren: Grammatiken und Automaten
- Rekursionstheorie-Einführung
- Komplexitätstheorie-Einführung

Organisatorisches

Vorlesungen FR 12.15-13.45 im F59
Vorschlag: 12.25-13.55 ab 2. Semesterwoche

Übungsbetrieb in Form von einer Übungsgruppe
BEGINN: als Sprechstunde in der ersten Semesterwoche
FR 14.00-15.30, F 59

Dozentensprechstunde DO 13-14 in meinem Büro H 410 (4. Stock)

Mitarbeitersprechstunde (Stefan Gulan) DO 13-14 H 413

Tutorensprechstunde (Martin Puppe) DO 13-14 H 407

Chomsky-Normalform für kontextfreie Grammatiken

Eine kontextfreie Grammatik $G = (N, T, P, S)$ ist in Chomsky-Normalform, falls alle ihre Regeln die folgende Form haben:

$$A \to BC \text{ oder } A \to a$$

für $A, B, C \in N$ und $a \in T$.
Kurz: $P \subset (N \times N^2) \cup (N \times T)$

Satz: Zu jeder kontextfreien Grammatik G gibt es eine Grammatik G' in Chomsky-Normalform mit $L(G') = L(G) \backslash \{\lambda\}$.

Hinweis: Mit dem Verfahren aus der vorigen Vorlesung können wir zunächst eine zu G äquivalente λ-freie kfG $\tilde{G}$ konstruieren mit $L(\tilde{G}) = L(G) \backslash \{\lambda\}$.

1. Lange rechte Seiten mit Terminalzeichen

Wir nennen eine Satzform *gemischt*, wenn sie sowohl Terminal- als auch Nichtterminalzeichen enthält.

Lemma: Zu jeder kfG G lässt sich eine äquivalente kfG G' konstruieren, bei welcher rechte Regelseiten mit Terminalzeichen die Länge höchstens Eins haben.

Mit G ist auch G' λ-frei.

Damit enthält G' auch keine gemischten rechten Regelseiten.

1. Führe für jedes Terminalzeichen a ein spezielles Nichtterminal X_a ein.
2. Führe neue Regeln $X_a \to a$ ein.
3. Ersetze a in rechter Regelseite w durch X_a, falls $w \neq a$.

$\Longrightarrow$ Nur noch Regeln der Formen

$$X \to a \text{ mit } a \in T \text{ oder } X \to w \text{ mit } w \in N^*$$

2. Lange rechte Seiten mit Nichtterminalzeichen

Entfernung aller Regeln $X \to B_1 ... B_k$ mit $k \geq 3$

Dazu neue Nichtterminale $C_1, C_2, ..., C_{k-2}$ und neue Regeln:

$$\begin{aligned} X &\to B_1 C_1 \\ C_1 &\to B_2 C_2 \\ &\dots \\ C_{k-2} &\to B_{k-1} B_k \end{aligned}$$

$\Longrightarrow$ Nur noch Regeln der Formen

$$X \to a \text{ mit } a \in T \text{ oder } X \to B_1 B_2 \text{ mit } B_1, B_2 \in N$$

oder

$$X \to B \text{ mit } B \in N$$

Die letztgenannten Kettenregeln zu entfernen können wir bereits (letzte VL)!

Der Weg zur Chomsky-Normalform Zusammenfassung:

0. Eliminiere λ-Regeln.
1. Behandle zu lange rechte Seiten mit Terminalzeichen.
2. Behandle zu lange rechte Seiten mit Nichtterminalzeichen.
3. Entferne Kettenregeln (durch Abkürzungen).

Beachte:
Schritt i zerstört nicht die durch Schritt $j < i$ gewonnenen Eigenschaften!

Ein Beispiel: Gegeben sei $G = (\{S, T\}, \{a, b, c\}, P, S)$ mit

$$P = \{S \to aT|TbT|T, T \to S|c\}$$

1) neue Nichtterminale X_a, X_b, neue Regeln $X_a \to a$, $X_b \to b$

Ersetzung: $S \to aT$durch $S \to X_aT$; $S \to TbT$ durch $S \to TX_bT$

neue Regelmenge:

$$\{S \to X_aT|TX_bT|T, T \to S|c, X_a \to a, X_b \to b\}$$

Ein Beispiel (Forts.):

2) Neues Nichtterminal C

Ersetzung: $S \to TX_bT$durch $S \to TC$ und $C \to X_bT$

neue Regelmenge

$$\{S \to X_aT|TC|T, T \to S|c, X_a \to a, X_b \to b, C \to X_bT\}$$

Ein Beispiel (Forts.):
Schritt 3: Eliminiere die beiden Kettenregeln $S \to T$ und $T \to S$.
Zuerst $S \to T$: füge die beiden Regeln $S \to S$ und $S \to c$ hinzu.
Bei $T \to S$ kommen die Regeln $T \to X_a T$, $T \to T$ und $T \to TC$ hinzu.

Wir haben jetzt die Regelmenge

$$\{S \to X_a T|TC|T|S|c, T \to S|c|X_a T|TC|T, X_a \to a, X_b \to b, C \to X_b T\}$$

Neue Regeln entstehen nicht mehr,
$\leadsto$ Entferne die vier Regeln $S \to S|T$ und $T \to S|T$.
$\leadsto$ Zu G äquivalente Grammatik $G' = (\{S, T, X_a, X_b, C\}, \{a, b, c\}, P', S)$ in Chomsky-Normalform, wobei:

$$P' = \{S \to X_a T|TC|c, T \to c|X_a T|TC, X_a \to a, X_b \to b, C \to X_b T\}$$

Satz: Es gibt einen Algorithmus, der zu jeder vorgelegten kfG $G = (N, T, P, S)$ und jedem $w \in T^*$ entscheidet, ob $w \in L(G)$ gilt.

Beweis: Wird $w = \lambda$ angefragt, so berechnen wir N_1 wie im Beweis I in der vorigen VL. Es gilt: $\lambda \in L(G)$ gdw. $S \in N_1$.
Wird $w \neq \lambda$ angefragt, so überführen wir G in eine Chomky-Normalform kfG G' mit der Eigenschaft $w \in L(G)$ gdw. $w \in L(G')$.
Da G' keine λ-Regeln enthält, kann man "$w \in L(G)$?" durch Berechnen aller aus G' ableitbaren Satzformen der Länge höchstens $\ell(w)$ beantworten.

Cocke, Younger und Kasami haben gezeigt, wie man die Komplexität des beschriebenen Verfahrens durch dynamisches Programmieren erheblich vermindern kann.

Das Verfahren von Cocke, Younger und Kasami (*CYK-Algorithmus*)

Satz: Ist eine kfG G in Chomsky-Normalform fixiert, so lässt sich die Frage "$w \in L(G)$?" in einer Zeit beantworten, die sich durch ein kubisches Polynom in $\ell(w)$ abschätzen lässt.

Beweis: Betrachte $w = a_1 \dots a_n$.
Lege 2–dimensionale dreiecksförmige Tabelle T an mit Nonterminalmengen als Einträgen, so dass $T[i,j] = \{A \in N \mid A \stackrel{*}{\Rightarrow} a_i \dots a_j\}$. T kann man wie folgt "von oben nach unten" berechnen:

$$T[i,i] = \{A \in N \mid A \to a_i \in P\}$$

$$T[i,j] = \{A \in N \mid A \to BC \in P \wedge \exists i \leq k < j : B \in T[i,k], C \in T[k+1,j]\}$$

Die CYK-Tabelle

a_1	a_2	$\ldots$	a_n
$T[1,1]$	$T[2,2]$	$\ldots$	$T[n,n]$
$T[1,2]$	$T[2,3]$	$\ldots$	
$\vdots$	$\vdots$		
$T[1,n-1]$	$T[2,n-1]$		
$T[1,n]$			

$a_1 \ldots a_n \in L(G)$ gdw. das Startsymbol erscheint in $T[1,n]$.

Der CYK-Algorithmus

FOR $j = 1, \ldots, n$

$\quad T[i,i] = \{A \in N \mid A \to a_i \in P\}$

FOR $j = 1, \ldots, n-1$

$\quad$ FOR $i = 1, \ldots, n-j$

$\quad\quad T[i, j+i] = \emptyset$

$\quad\quad$ FOR $k = i, \ldots, j+i-1$

$\quad\quad\quad$ IF $\exists A \to BC \in P : B \in T[i,k], C \in T[k+1, j+i]$

$\quad\quad\quad$ THEN $T[i, j+i] = T[i, j+i] \cup \{A\}$

Ein Beispiel

$S \to AX$, $X \to SB$, $S \to c$, $A \to a$, $B \to b$ erzeugt $L = \{a^n c b^n \mid n \geq 0\}$.

a	a	c	b	b
A	A	S	B	B
		X		
	S			
	X			
S				

Ein **binärer Wurzelbaum** ist gegeben durch ein Tripel $B = (V, \phi, r)$ mit ausgezeichneter *Wurzel* $r \in V$ und einer *Vater-Abbildung* $\phi : V \setminus \{r\} \to V$ mit der Eigenschaft

$$\forall v \in V : \#\underbrace{\{u \in V \mid \phi(u) = v\}}_{\kappa(v):=} \leq 2$$

$\kappa(v)$ liefert also die *Kinder* von v.
Knoten v mit $\kappa(v) = \emptyset$ heißen *Blätter*.

Lemma: Der Ableitungsbaum eines jeden von einer kontextfreien Grammatik in Chomsky-Normalform akzeptierten Wortes ist als binärer Wurzelbaum auffassbar.

Die *Höhe* eines binären Wurzelbaumes $B = (V, \phi, r)$ ist gegeben durch

$$h(B) = \max_{v \in V \setminus \{r\}} \{k \in \mathbb{N} \mid \phi^k(v) = r\}.$$

Lemma: Hat ein binärer Wurzelbaum B mehr als 2^h Blätter, so gilt $h(B) > h$.

Beweis: Wir zeigen die *Kontraposition* per Induktion:
Gilt $h(B) \leq h$, so hat $B = (V, \phi, r)$ höchstens 2^h viele Blätter.
$h = 0$ ist trivial.
Es gelte $h > 0$. Daher gilt $\kappa(r) \neq \emptyset$.
Betrachte $r' \in \kappa(r)$. $V' := \{v \in V \mid \exists k \in \mathbb{N} : \phi^k(v) = r'\}$.
ϕ' sei die Einschränkung von ϕ auf $V' \setminus \{r'\}$.
$B' = (V', \phi', r')$ ist ein binärer Wurzelbaum der Höhe höchstens $h - 1$.
Auf B' ist die Induktionshypothese anwendbar: B' hat höchstens 2^{h-1} viele Blätter.
$\rightsquigarrow$ B hat höchstens $\#\kappa(r) \cdot 2^{h-1} \leq 2^h$ viele Blätter.

Lemma: Ist $G = (N, T, P, S)$ eine kfG in Chomsky-Normalform und ist $w \in L(G)$ mit $\ell(w) > 2^{\#N}$, so gilt für jeden Ableitungsbaum von w bzgl. G, dass es einen Weg von S zu einem Blatt gibt, auf dem mehr als $\#N$ viele Nichtterminalzeichen ersetzt werden.

Beweis: Nach dem vorigen Lemma hat der unterliegende binäre Wurzelbaum die Höhe größer als $\#N$. Es gibt also einen Weg in besagtem Ableitungsbaum von der mit S beschrifteten Wurzel zu einem Blatt, dem mehr als $\#N$ Regelanwendungen entspricht $\rightsquigarrow$ Behauptung.

Folg.: Auf besagtem Weg von der Wurzel zum Blatt im Ableitungsbaum von w finden wir also nach dem Schubfachprinzip zwei Regelanwendungen $A \to v$ und $A \to u$ mit gleicher linker Seite.

Ein Pumping-Lemma für Typ-2

Satz: Zu jeder kfS L gibt es eine Konstante $n > 0$, sodass jedes Wort $w \in L$ mit $\ell(w) \geq n$ als Konkatenation $w = uvxyz$ dargestellt werden kann mit geeigneten u, v, x, y, z mit folgenden Eigenschaften:

1. $\ell(v) > 0$ oder $\ell(y) > 0$;

2. $\ell(vxy) \leq n$;

3. $\forall i \geq 0 : uv^i xy^i z \in L$.

Beweis: Mit L ist auch $L \setminus \{\lambda\}$ vom Typ 2. Sei also o.E. L λ-frei.
L werde durch kfG $G = (N, T, P, S)$ in Chomsky-Normalform beschrieben.
Es sei $n = 2^{\#N}$.
Gibt es kein $w \in L$ mit $\ell(w) \geq n$, so stimmt die Aussage leer.
Sonst wähle ein $w \in L$ mit $\ell(w) \geq n$.
Nach obiger Folgerung gibt es zwei Regelanwendungen $A \to u'$ und $A \to v'$ auf einem Weg in einem Ableitungsbaum von w.
$\rightsquigarrow$ Es gibt Linksableitung

$$S \overset{*}{\Rightarrow} uA\zeta \overset{*}{\Rightarrow} uvA\eta\zeta \overset{*}{\Rightarrow} uvx\eta\zeta \overset{*}{\Rightarrow} uvxy\zeta \overset{*}{\Rightarrow} uvxyz.$$

Wegen Chomsky-NF (keine Kettenregeln oder λ-Regeln) gilt $\ell(v) > 0$ oder $\ell(y) > 0$.
$\ell(vxy) \leq n$ ergibt sich aus dem Beweis der Folgerung sowie wegen Chomsky-NF.
Daher gibt es auch Ableitungen

$$S \overset{*}{\Rightarrow} uA\zeta \overset{*}{\Rightarrow} ux\zeta \overset{*}{\Rightarrow} uxz$$

und allgemeiner

$$S \to uA\zeta \overset{*}{\Rightarrow} uvA\eta\zeta \overset{*}{\Rightarrow} uvvA\eta\eta\zeta \overset{*}{\Rightarrow} uv^iA\eta^i\zeta \overset{*}{\Rightarrow} uv^ixy^iz.$$

Ein Beispiel zur Anwendung des Pumping-Lemmas:

$$L = \{a^k b^k c^k \mid k \in \mathbb{N}\} \notin \mathcal{L}_2$$

Wäre L kontext-frei, so gäbe es Pumping-Konstante n.
Wähle $w = a^n b^n c^n$ als genügend langes Wort.
Diskutiere Zerlegungen $w = uvxyz$.
$\ell(vxy) \leq n \rightsquigarrow vxy$ enthält höchstens a's gefolgt von b's oder b's gefolgt von c's.
Beide Fälle sind analog; diskutiere also den ersten.
"Nullpumpen" liefert ein Wort, in dem alle a- und b-Vorkommen zusammen weniger als die doppelte Anzahl von c's ausmachen: $\lightning$ Wortstruktur.

Eine Anwendung von KfG:
Beschreibung der Syntax (grammatisches Gerüst) natürlicher Sprachen

Satz $\rightarrow$ NP VP
NP $\rightarrow$ Artikel Nomen
VP $\rightarrow$ Verb
Artikel $\rightarrow$ “der”
Artikel $\rightarrow$ “die”
Nomen $\rightarrow$ “Hund”
Nomen $\rightarrow$ “Hunde”
Nomen $\rightarrow$ “Katze”
Verb $\rightarrow$ “beißen”

Satz $\Rightarrow$ NP VP $\Rightarrow^2$ Artikel Nomen Verb $\Rightarrow^3$ “die” “Hunde” “beißen”
Satz $\Rightarrow$ NP VP $\Rightarrow^2$ Artikel Nomen Verb $\Rightarrow^3$ “die” “Katze” “beißen”
Satz $\Rightarrow$ NP VP $\Rightarrow^2$ Artikel Nomen Verb $\Rightarrow^3$ “der” “Katze” “beißen”

Eine Anwendung von KfG: **Beschreibung arithmetischer Ausdrcke**

$\Sigma = \{v, -, +, *, /, (,)\}$, $N = \{E\}$.
Die Regeln seien die folgenden:

$E \to (E)$
$E \to -(E)$
$E \to (E + E)$
$E \to (E - E)$
$E \to (E * E)$
$E \to (E/E)$
$E \to v$

Beispielableitung:

$E \Rightarrow -(E)$
$\Rightarrow -(E + E)$
$\Rightarrow -((E * E) + E)$
$\Rightarrow -((-(E) * E) + E)$
$\Rightarrow -((-(E) * E) + (E - E))$
$\overset{*}{\Rightarrow} -((-(v) * v) + (v - v))$

Hinweis: Der *Scanner*, ein endlicher Automat, produziert idealerweise als Ausgabe die Eingabe für den *Parser* zwecks *syntaktischer Analyse* eines Programmtextes.

“Nebensächlichkeiten” wie Zahlen oder Zahlenvariablen werden in einen Statthalter v bersetzt.

Ein Wort hat viele Ableitungen

Linksableitung:

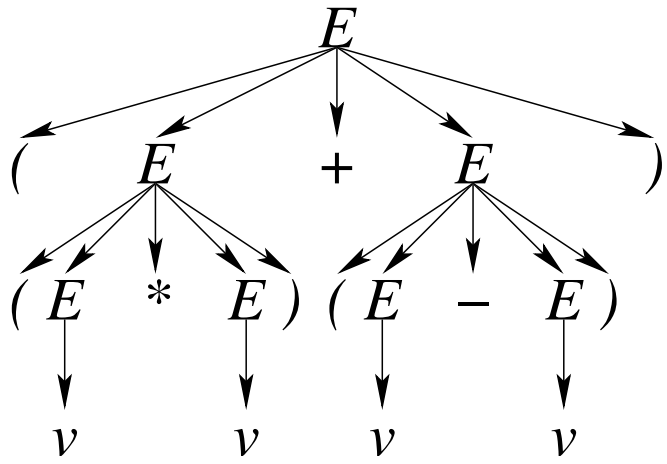
$$
\begin{aligned}
&E \Rightarrow -(E) \\
&\Rightarrow -(E + E) \\
&\Rightarrow -((E * E) + E) \\
&\Rightarrow -((-(E) * E) + E) \\
&\Rightarrow -((-(v) * E) + E) \\
&\Rightarrow -((-(v) * v) + E) \\
&\Rightarrow -((-(v) * v) + (E - E)) \\
&\Rightarrow -((-(v) * v) + (v - E)) \\
&\Rightarrow -((-(v) * v) + (v - v))
\end{aligned}
$$

Rechtsableitung:

$$
\begin{aligned}
&E \Rightarrow -(E) \\
&\Rightarrow -(E + E) \\
&\Rightarrow -(E + (E - E)) \\
&\Rightarrow -(E + (E - v)) \\
&\Rightarrow -(E + (v - v)) \\
&\Rightarrow -((E * E) + (v - v)) \\
&\Rightarrow -((E * v) + (v - v)) \\
&\Rightarrow -((-(E) * v) + (v - v)) \\
&\Rightarrow -((-(v) * v) + (v - v))
\end{aligned}
$$

Ein Syntaxbaum (oder *Ableitungsbaum*) für

$$\mathrm{E} \Rightarrow (\mathrm{E}+\mathrm{E}) \Rightarrow ((\mathrm{E}*\mathrm{E})+\mathrm{E}) \Rightarrow ((\mathrm{E}*\mathrm{E})+(\mathrm{E}-\mathrm{E})) \overset{*}{\Rightarrow} ((v*v)+(v-v))$$



Linksableitung: Tiefensuche mit Linksabstieg durch Syntaxbaum

Rechtsableitung: Tiefensuche mit Rechtsabstieg durch Syntaxbaum

Operatorbaum: ein kompakter Syntaxbaum für das ursprüngliche Beispiel