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

Ausblick Formale Sprachen

Hauptfragen

- Entscheidbarkeitsfragen (und Komplexität)
- Abschlusseigenschaften
- Charakterisierungen und Normalformen
- Hierarchiefragen

Abschlusseigenschaften

Diese Fragen zielen auf algebraische Eigenschaften von Sprachfamilien.
Eine Sprachklasse $\mathcal{L}$ ist (u.a.) ein Mengensystem.
$\rightsquigarrow$ Abschluss gegen Mengenoperationen ?
Beispielsweise bedeutet *Abschluss unter Vereinigung*:
Wann immer L und M aus $\mathcal{L}$ sind, so ist auch $L \cup M$ aus $\mathcal{L}$.
Allgemein betrachten wir (Scharen von) n-stelligen Mengen-Operatoren O und fragen, ob für beliebige $o \in O$ und beliebige $L_1, \ldots, L_n \in \mathcal{L}$ gilt: $o(L_1, \ldots, L_n) \in \mathcal{L}$, sofern o für die Argumente $L_1, \ldots, L_n$ definiert ist.
Einfache weitere Beispiele: Konkatenation und Kleene-Stern.
Jetzt exemplarisch bei kontextfreien Sprachen!

Satz: $\mathcal{L}_2$ ist unter Vereinigung abgeschlossen.

Beweis: L_1, L_2 seien durch kfGs G_1 bzw. G_2 spezifiziert. Zu zeigen: $L_1 \cup L_2 \in \mathcal{L}_2$.

Genauer sei, für $i = 1, 2$, $G_i = (N_i, \Sigma_i, P_i, S_i)$. O.E. Annahme: $N_1 \cap N_2 = \emptyset$.

Betrachte $G = (N_1 \cup N_2 \cup \{S\}, \Sigma_1 \cup \Sigma_2, P_1 \cup P_2 \cup \{S \to S_1, S \to S_2\}, S)$ mit $S \notin N_1 \cup N_2$.

Beh.: $L(G) = L(G_1) \cup L(G_2)$.

$w \in L(G)$ gdw. es gibt Ableitungsfolge $S \Rightarrow S_i \overset{*}{\Rightarrow}_G w$ ($S \notin N_1 \cup N_2$)

gdw. $S_1 \overset{*}{\Rightarrow}_{G_1} w$ oder $S_2 \overset{*}{\Rightarrow}_{G_2} w$ ($N_1 \cap N_2 = \emptyset$)

gdw. $w \in L(G_1)$ oder $w \in L(G_2)$

gdw. $w \in L(G_1) \cup L(G_2)$.

Satz: $\mathcal{L}_2$ ist unter Konkatenation abgeschlossen.

Beweis: L_1, L_2 seien durch kfGs G_1 bzw. G_2 spezifiziert. Zu zeigen: $L_1 \cdot L_2 \in \mathcal{L}_2$.

Genauer sei, für $i = 1, 2$, $G_i = (N_i, \Sigma_i, P_i, S_i)$. O.E. Annahme: $N_1 \cap N_2 = \emptyset$.

Betrachte $G = (N_1 \cup N_2 \cup \{S\}, \Sigma_1 \cup \Sigma_2, P_1 \cup P_2 \cup \{S \to S_1 S_2\}, S)$ mit $S \notin N_1 \cup N_2$.

Beh.: $L(G) = L(G_1)L(G_2)$.

$w \in L(G)$ gdw. es gibt Ableitungsfolge $S \Rightarrow S_1 S_2 \overset{*}{\Rightarrow}_G w$ ($S \notin N_1 \cup N_2$)

gdw. $S_1 \overset{*}{\Rightarrow}_{G_1} u$ und $S_2 \overset{*}{\Rightarrow}_{G_2} v$ mit $uv = w$ ($N_1 \cap N_2 = \emptyset$)

gdw. $u \in L(G_1)$ und $v \in L(G_2)$

gdw. $w \in L(G_1)L(G_2)$ mit $w = uv$.

Satz: $\mathcal{L}_2$ ist unter Kleene Stern abgeschlossen.

Beweis: L sei durch kfG $G = (N, \Sigma, P, \hat{S})$ spezifiziert.

Betrachte $G' = (N \cup \{\hat{S}\}, \Sigma, P \cup \{\hat{S} \to \hat{S}S, \hat{S} \to \lambda\}, \hat{S})$ mit $\hat{S} \notin N$.

Beh.: $L(G') = L(G)^*$.

$w \in L(G')$ gdw. es gibt Ableitungsfolge $\hat{S} \Rightarrow^k \hat{S}S^k \Rightarrow S^k \overset{*}{\Rightarrow}_G w_1 \dots w_k = w$ ($S \notin N$)

gdw. $S \overset{*}{\Rightarrow}_G w_i$ für $1 \leq i \leq k$

gdw. $w_i \in L(G)$ für $1 \leq i \leq k$

gdw. $w = w_1 \dots w_k \in L(G)^*$.

Satz: $\mathcal{L}_2$ ist unter Durchschnitt mit regulären Sprachen abgeschlossen.

Beweis: L sei durch Kellerautomaten $A = (Q, \Sigma, \Gamma, q_0, \Delta, F)$ akzeptiert.

R sei akzeptiert durch NEA $A' = (Q', \Sigma, \delta, q_0', F')$.

Betrachte Produktkellerautomat $\hat{A} = (\hat{Q}, \Sigma, \Gamma, \hat{q}_0, \hat{\Delta}, \hat{F})$ mit

$\hat{Q} = Q \times Q'$, $\hat{q}_0 = (q_0, q_0')$, $\hat{F} = F \times F'$

Für $a \in \Sigma$: $(((q, q'), a, X), ((p, p'), Y)) \in \hat{\Delta}$ gdw. $((q, a, X), (p, Y)) \in \Delta$ und $(q', a, p') \in \delta$.

Für $a = \lambda$: $(((q, q'), a, X), ((p, p'), Y)) \in \hat{\Delta}$ gdw. $((q, a, X), (p, Y)) \in \Delta$ und $q' = p'$.

Beh.: $L(\hat{A}) = L(A) \cap L(A')$.

Hinweis: Auf Grammatikebene komplizierterer Beweis über Tripelkonstruktion.

Satz: $\mathcal{L}_2$ ist nicht unter Schnittbildung abgeschlossen.

Beweis: Betrachte

$$L = \{a^m b^m c^n | m, n \geq 1\} \quad L' = \{a^m b^n c^n | m, n \geq 1\}$$

kontextfreie Grammatiken dazu:

$$\begin{aligned} G = \quad & (\quad \{S, T, C\}, \{a, b, c\}, \\ & \quad \{S \to TC, T \to ab|aTb, C \to c|cC\}, S\} \quad) \\ G' = \quad & (\quad \{S, T, A\}, \{a, b, c\}, \\ & \quad \{S \to AT, A \to a|aA, T \to bc|bTc\}, S\} \quad) \end{aligned}$$

$L \cap L' = \{a^m b^m c^m | m \geq 1\}$ ist nicht kontextfrei.

De Morgan $\rightsquigarrow$ Folg.: $\mathcal{L}_2$ ist nicht unter Komplement abgeschlossen.

Entscheidbarkeitsfragen

- Wortproblem:
  Zwei Varianten: (1) uniform / allgemein oder (2) fixiert / speziell

- Leerheitsproblem: Ist $L(G)$ leer?

- Endlichkeitsproblem: Ist $L(G)$ endlich?

- Äquivalenz

Der Klassiker Das *Wortproblem*

Zwei Varianten: (1) uniform / allgemein oder (2) fixiert / speziell

(1) Ggb.: (z.B.) Grammatik G und Wort w, Frage: Gilt $w \in \mathrm{L}(\mathrm{G})$?
(2) Gibt es zu vorgelegter Sprache L einen Algorithmus, der entscheidet, ob die Eingabe w in L liegt ?
Anders ausgedrückt fragt (2) danach, ob L *rekursiv* oder *entscheidbar* ist.

CYK kann als uniformer Wortproblemalgorithmus für $\mathcal{L}_2$ aufgefasst werden.

Sind gewisse Abschlusseigenschaften konstruktiv bekannt, insbesondere Schnitt mit einelementigen Sprachen, so lässt sich das allgemeine Wortproblem (und damit auch das spezielle) oft positiv lösen über das Leerheitsproblem.

Wie würde dieser Weg aussehen bei den endlichen Automaten ?

Satz: Für kontextfreie Grammatiken ist das Leerheitsproblem (in Polynomzeit) entscheidbar.

Beweis: Es sei kfG G gegeben.

Konstruiere kfG G' aus G wie folgt:

Ist $A \to \alpha$ Regel in G, so ist $A \to \alpha'$ Regel in G', wobei α' aus α dadurch entsteht, dass jedes Terminalzeichen in α durch λ in α' ersetzt wird.

Also: $\lambda \in L(G')$ gdw. $w \in L(G)$ für irgendein Terminalwort w.

"$\lambda \in L(G')$?" kann durch Konstruktion der Menge N_1 (VL2) leicht entschieden werden.

Satz: Für kontextfreie Grammatiken ist das Endlichkeitsproblem in Exponentialzeit entscheidbar.

Beweis: Überführe zunächst eine vorgelegte Grammatik G in eine (bis auf λ) äquivalente kfG G' in Chomsky-Normalform.

$L(G)$ ist endlich gdw. $L(G')$ ist endlich.

Es sei n die zu G' gehörige Pumpkonstante (Erinnerung: n ist exponentiell in #NT).

Betrachte Menge M alle Wörter der Längen $n \leq k < 2n$:

Beh.: $L(G')$ ist endlich gdw. $M = \emptyset$.

Die komplizierte Richtung $\Leftarrow$ folgt durch Kontraposition und Nullpumpen.

Zusammenfassung Abschlüsse

Abschluss	Schnitt	Vereinigung	Komplement	Konkatenation	Stern
Typ 3	ja	ja	ja	ja	ja
Typ 2 Det.	nein	nein	ja	nein	nein
Typ 2	nein	ja	nein	ja	ja
Typ 1	ja	ja	ja	ja	ja
Typ 0	ja	ja	nein	ja	ja

Hinweis: Erst 1988 wurde von *Immerman* und *Szelepcsényi* gezeigt, dass mit $L \subseteq \Sigma^*$ auch $\Sigma^* \setminus L$ vom Typ 1 ist.

Zusammenfassung Entscheidbarkeit

Entscheidbarkeit	Wortproblem	Aufwand WP	Leerheitspr.	Äquivalenzproblem	Schnittleerheitsproblem
Typ 3	ja	linear bei DEA	ja	ja	ja
Typ 2 Det.	ja	linear	ja	ja	nein
Typ 2	ja	$O(n^3)$ bei CNF	ja	nein	nein
Typ 1	ja	exponentiell	nein	nein	nein
Typ 0	nein	-	nein	nein	nein

Ausblick 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 $G = (N, \Sigma, P, S)$ liefert zu einem Wort $w \in L(G)$ eine Linksableitung von w, beschrieben durch ein Wort über P, 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 $G = (N, \Sigma, P, S)$ 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 $((\mathfrak{f}, \mathfrak{a}, \mathfrak{a}), (\mathfrak{f}, \lambda))$ 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 $r_1 = S \to ASb$, $r_2 = A \to a$ und $r_3 = S \to \lambda$.

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_1
$aabb$	aSb	r_2
abb	Sb	(Leseschritt)
abb	$ASbb$	r_1
abb	$aSbb$	r_2
bb	Sbb	(Leseschritt)
bb	bb	r_3
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 berein, 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* $\$$.

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

$S \to B\ (\beta)$, $S \to I\ (\iota)$, $S \to a;S\ (a)$, $S \to \lambda\ (\$, \varepsilon, \theta)$

$B \to \beta S \varepsilon S$

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

Die blauen Symbole in Klammern bezeichnen den in der *Vorschau (Lookahead)* berücksichtigten Teil der Resteingabe.

β: “begin”; ε: “end”

ι: “if”; θ: “then”; η: “else”

Ein Linksparser bei der Arbeit

$S \to B\ (\beta)$,

$S \to I\ (\iota)$,

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

$S \to \lambda\ (\$, \varepsilon, \theta)$

$B \to \beta S \varepsilon S$

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

Resteingabe	Kellerinhalt	Ausgabe
$a; \iota c \theta \beta a; \varepsilon; \eta \varepsilon \$$	S	(Anfang)
$a; \iota c \theta \beta a; \varepsilon; \eta \varepsilon \$$	$a; S$	$S \to a; S$
$\iota c \theta \beta a; \varepsilon; \eta \varepsilon \$$	S	(zwei Leseschritte)
$\iota c \theta \beta a; \varepsilon \eta \varepsilon \$$	I	$S \to I$
$\iota c \theta \beta a; \varepsilon \eta \varepsilon \$$	$\iota c \theta S \eta S \varepsilon S$	$I \to \iota c \theta S \eta S \varepsilon S$
$\beta a; \varepsilon \eta \varepsilon \$$	$S \eta S \varepsilon S$	(drei Leseschritte)
$\beta a; \varepsilon \eta \varepsilon \$$	$B \eta S \varepsilon S$	$S \to B$
$\beta a; \varepsilon \eta \varepsilon \$$	$\beta S \varepsilon S \eta S \varepsilon S$	$B \to \beta S \varepsilon S$
$a; \varepsilon \eta \varepsilon \$$	$S \varepsilon S \eta S \varepsilon S$	(Leseschritt)
$a; \varepsilon \eta \varepsilon \$$	$a; S \varepsilon S \eta S \varepsilon S$	$S \to a; S$
$\varepsilon \eta \varepsilon \$$	$S \varepsilon S \eta S \varepsilon S$	(zwei Leseschritte)
$\varepsilon; \eta \varepsilon \$$	$\varepsilon S \eta S \varepsilon S$	$S \to \lambda$
$\eta \varepsilon \$$	$S \eta S \varepsilon S$	(Leseschritt)
$\eta \varepsilon \$$	$\eta S \varepsilon S$	$S \to \lambda$
$\varepsilon \$$	$S \varepsilon S$	(Leseschritt)
$\varepsilon \$$	εS	$S \to \lambda$
$\$$	S	(Leseschritt)
$\$$		$S \to \lambda$
$\$$		(Akzeptanz)

Parsen mit rekursivem Abstieg für *LL(1)*

- Für jedes oberste Kellerzeichen X Extra-Prozedur $P_X(au, K)$, wobei
  au restliche Eingabe mit aktuellem Eingabezeichen a und
  K Keller (ohne oberstes Zeichen X).

- $P_b(au, K)$ für Terminalzeichen b:
  (1) liefert Fehlermeldung, falls $b \neq a$;
  (2) falls $b = a$, wird das oberste Kellerzeichen entfernt; gilt nun
  (2A) $a = \$$ und $u = K = \lambda$, so akzeptiere; andernfalls ist
  (2B) $K = \lambda$: Fehlerfall;
  für (2C) $K \neq \lambda$ sei $K = X'K'$ und rufe $P_{X'}(u, K')$ rekursiv auf.

- In $P_X(au, K)$ wird für ein Nichtterminalzeichen X nach verschiedenen Möglichkeiten für a rekursiv verzweigt.
  (Hier könnte man auch noch tiefer in der Ausgabe vorausschauen, wenn LL(k) gefordert.)
  Das heißt, die betreffende Regel $X \to w$ wird ausgeführt und der neue Keller erfüllt $X'K' = wK$ für ein Zeichen X', und rekursiv wird $P_{X'}(au, K')$ aufgerufen.

Beobachte den Rekursionskeller !
Gefahr durch Linksrekursion in der Grammatik, d.h.: $X \stackrel{*}{\Rightarrow} X\alpha$.

Rechtsparser: Bottom-up Simulation von Ableitungsbäumen

Sei $G = (N, \Sigma, P, S)$ eine kfG.
Betrachte folgende Transitionen (Kellerautomat hat nur einen Zustand):

- für jede Regel $C \to w$: $((f, \lambda, w), (f, C))$ (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.
Vergleiche hierzu unseren “Ersetzungssystem”-Ansatz für Kellerautomaten!

Die Konstruktion am Beispiel

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

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 $S \Rightarrow ASb \Rightarrow AASbb \Rightarrow^3 aabb$ 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_2
bb	Aa	(Leseschritt)
bb	AA	r_2
bb	AAS	r_3
b	$AASb$	(Leseschritt)
b	AS	r_1
λ	ASb	(Leseschritt)
λ	S	(r_1 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 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 z.B. $x = b \rightsquigarrow r_1$ oder auch r_3;
Hilft hier wohl nicht . . .

LR(1): Auflösen von Konflikten durch 1-Symbol-Vorschau in der Eingabe.
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.