Formale Sprachen

(parallele und regulierte Ersetzung)

SoSe 2007 in Trier

Henning Fernau
Universität Trier
fernau@informatik.uni-trier.de

Formale Sprachen Gesamtübersicht

- Organisatorisches / Einführung
  Motivation / Erinnerung / Fragestellungen

- Diskussion verschiedener Sprachklassen:
  gesteuerte Ersetzungsverfahren
  parallele Ersetzungsverfahren

- Algebraische Ansätze:
  abstrakte Sprachfamilien
  formale Potenzreihen

Organisatorisches

Vorlesung: Mittwochs 12-14 Uhr, HZ 203,
Vorschlag: 12.25-13.55 (schafft bequemere Mittagspause)
Ausnahme: 25.4., 12.00-13.00.

Übungen (Stefan Gulan): Montags 16-18 Uhr, HZ 204
Beginn 2. Semesterwoche

Meine Sprechstunde: MO, 14-15 Uhr

Kontakt: fernau@informatik.uni-trier.de

Hausaufgaben / Schein ?! n.V.

Kleiner Werbespot

Hinweis: Im Seminar: Exakte und approximative Algorithmen für Logik-Probleme sind noch Plätze für Kurzentschlossene frei.

Bitte umgehend bei Interesse anmelden !

Veranstaltungsform: Blockform nach Pfingsten

Motivation

Theoretische Informatik (ThI) versucht ganz allgemein mathematische Grundlagen der Informatik bereitzustellen.

Die konkreten Ausprägungen von ThI müssen in diesem Kontext gesehen werden.

Ursprüngliche (Informatik-)Motivation
für die Entwicklung der Formalen Sprachen war der *Compilerbau*.

Einordnung

Formale Sprachen und Komplexitätstheorie sind die klassischen Gebiete der Theoretischen Informatik, und zwar unabhängig von nationalen Moden.

Während sonst in Deutschland beispielsweise die Algorithmik stark die Theoretische Informatik dominiert, herrschen in Frankreich eher Logik und Semantik vor.

Stets gehören aber FS und KT zum Kanon der Theorie.

Wir können und werden daher hier auch auf Ihre Kenntnisse aus den Grundvorlesungen aufbauen.

Erinnerung: Chomsky-Hierarchie...

war zunächst linguistisch, nicht informatisch motiviert.

Entsprechungen bei den Compilerbau-Phasen:

lexikalische Analyse (Scanner): Reguläre Sprachen, Typ 3
syntaktische Anslyse (Parsing): Kontextfreie Sprachen, Typ 2
semantische Analyse: muss immer noch entscheidbar sein...

Weitere Eigenschaften, motiviert aus Compilerbau:

Determinismus: Jedes Programm sollte eine eindeutig bestimmte, “leicht” zu ermittelnde Bedeutung besitzen.

Nichtdeterminismen gestatten aber manchmal einfachere, evtl. “menschlichere” Spezifikationen (z.B. durch reguläre Ausdrücke.

Transduktionen: ein vornehmer Ausdruck für übersetzende Maschinen: es genügt nicht, wenn der Compiler antwortet: “Ja, dies ist ein syntaktisch korrektes C-Programm !” Stattdessen sollte (wenigstens dann) auch eine gültige Übersetzung in Maschinencode erfolgen.

Weitere Motivationen

Linguistik: eigentlich die “ursprüngliche”, aber Anwendungen von Computern auf natürliche Sprachen sind nicht nur ein spannendes Gebiet (Computerlinguistik), sondern werfen immer wieder auch neue theoretische Fragestellungen auf.

Sondersprachen: Es gibt ja nicht nur C, JAVA, u.ä. “Hochsprachen”, die alle mehr oder minder erfolgreich als universelle Programmiersprachen dienen. In ihrer Masse evtl. noch wichtiger sind Spezialsprachen für Datenbanken, semistrukturierte Dokumente (XML), Graphik, Internet etc. Die dort auftretenden Effekte lassen sich zumeist sauber mit formalen Sprachen beschreiben.

Anwendungen in der Computergraphik

Zwei Vorgehensweisen:

1. Betrachte genuine Beschreibungsverfahren für Bilder anstelle von Wörtern
2. Deute Wörter als Beschreibungen von Bildern und benutze die “üblichen” Wortbeschreibungsverfahren.

Besonders bekannt: Kettencodes und Turtle-Graphik (Codierung absoluter bzw. relativer Bewegungen)

In der letzten halben Stunde der ersten Vorlesung werden wir viele Beispiele hierzu sehen.

Ersetzungssysteme

1. Ein *Ersetzungssystem* ist ein Paar $ES = (V, F)$, wobei
   – V ein Alphabet
   – F eine endliche Teilmenge von $V^* \times V^*$ ist.
   Die Elemente $(x, y) \in F$ heißen *(Ersetzungs-) Regeln* oder *Produktionen*.
   Schreibweise: $x \to y$.

2. Ein Wort $v \in V^*$ leitet (direkt bzw. in einem Schritt) ein Wort $w \in V^*$ ab genau dann, wenn

$$\exists z, z' \in V^* : \exists\, x \to y \in F : (v = zxz' \wedge w = zyz')$$

   Schreibweise: $x \Rightarrow^1_{ES} y$ bzw. $x \Rightarrow y$, wenn das Ersetzungssystem aus dem Kontext hervorgeht.

$\Rightarrow_{ES}^*$ bezeichnet den *reflexiv-transitiven Abschluss* von $\Rightarrow_{ES}$:
$v \Rightarrow_{ES}^* w \iff v = w \vee \exists u \in V^* : (v \Rightarrow_{ES}^* u \wedge u \Rightarrow_{ES}^1 w)$, d.h., gilt $v \Rightarrow_{ES}^* w$, so existieren $v = u_0, u_1, \ldots, u_{k-1}, u_k = w \in V^*, k \in \mathbb{N}$ und $u_{i-1} \Rightarrow_{ES}^1 u_i$ für $i = 1, \ldots, k$. Die Folge $(u_0, \ldots, u_k)$ heißt dann eine *Ableitung* von v nach w (bzgl. ES der Länge k).
$v \Rightarrow_{ES}^k w \iff$ es gibt in ES eine Ableitung von v nach w der Länge k.

3. Sei $AX \subseteq V^*$ eine Axiomenmenge.
   Die von ES aus AX erzeugte Sprache:
   $L_g(ES, AX) = \{w \mid \exists v \in AX : v \Rightarrow_{ES}^* w\}$.
   Die von ES mit AX akzeptierte Sprache:
   $L_a(ES, AX) = \{w \mid \exists v \in AX : w \Rightarrow_{ES}^* v\}$.

Bemerkungen:

- Die Ableitung von Wörtern ist nicht eindeutig, d.h., weder die Reihenfolge noch die Anwendungsstelle von Produktionen muß deterministisch sein.
  Beispiele:
  (1) $\mathrm{F} = \{ \mathrm{A} \to \mathrm{B}, \mathrm{B} \to \mathrm{D}, \mathrm{A} \to \mathrm{C}, \mathrm{C} \to \mathrm{D} \}$ hat Ableitungen $(\mathrm{A}, \mathrm{B}, \mathrm{D})$ und $(\mathrm{A}, \mathrm{C}, \mathrm{D})$.
  (2) $\mathrm{F}' = \{ \mathrm{A} \to \mathrm{bAA} \}$ hat Ableitungen $(\mathrm{AA}, \mathrm{AbAA}, \mathrm{bAAbAA})$ und $(\mathrm{AA}, \mathrm{bAAA}, \mathrm{bAAbAA})$.
  (3) $\mathrm{F}'' = \{ \mathrm{A} \to \mathrm{AbA} \}$ zeigt bei der Ableitung $(\mathrm{AbA}, \mathrm{AbAbA})$ eine Art "interner Mehrdeutigkeit", da selbst bei Vorgabe der Ableitung nicht klar ist, wo ersetzt wurde.

- Ersetzungssysteme entsprechen *Semi-Thue-Systemen*. Hingegen werden bei *Thue-Systemen* ungerichtete Regeln betrachtet, also der reflexiv-transitiv-symmetrische Abschluss bezüglich der Ableitungen. In Vorlesungen über Termersetzungssysteme werden Sie es mit Thue-Systemen zu tun bekommen.

  Übungsvorschlag: Versuchen Sie, die Arbeitsweise einer (Ein-Band-)Turing-Maschine als Ersetzungssystem zu formalisieren.

Beispiel: $F = \{a \to aba, aa \to b\}, V = \{a, b\}, ES = \{V, F\}$

$L_g(ES, \{a\}) = a(ba)^* = (ab)^*a$

$L_g(ES, \{aa\}) = (ab)^*aa(ba)^* \cup (ab)^*b(ba)^*$

$L_a(ES, b^*) \cap a^* = \{a^n \mid n \geq 2\}$

$L_g(ES, a^n) \cap b^* = \{b^m \mid m = 2n-3, 2n-6, 2n-9, \ldots, 2n-3j \geq 3\}$ $(n \neq 2, 4)$:

$a^2 \Rightarrow b$

$a^3 \Rightarrow a^2ba^2 \Rightarrow b^2a \Rightarrow b^3$

$a^4 \Rightarrow a^2b \Rightarrow b^2$

$a^5 \Rightarrow ba^3 \Rightarrow^* b^4$

$a^5 \Rightarrow a^3ba^3 \Rightarrow^* b^4a^3 \Rightarrow b^7$

(Chomsky-) Grammatiken Eine *Grammatik* ist ein Quadrupel $\mathrm{G} = (V_\mathrm{N}, V_\mathrm{T}, S, \mathrm{P})$, wobei

- V_N ein Alphabet von *Nonterminalen* oder *Variablen*
- V_T ein zu V_N disjunktes Alphabet von *Terminalen*
- $S \in V_\mathrm{N}$ das *Axiom* oder *Startsymbol*
- $\mathrm{P} \subseteq (V^* \circ V_\mathrm{N} \circ V^*) \times V^*$ eine endliche Menge von *Regeln* oder *Produktionen*; hierbei ist $V := V_\mathrm{N} \cup V_\mathrm{T}$.

(V, P) heißt dann das von G induzierte *Ersetzungssystem*. Die zugehörige *Ableitungsrelation* wird mit $\Rightarrow_G^1$ und $\Rightarrow_G^*$ bezeichnet. Die von G erzeugte Sprache $L(G) := \{w \in V_T^* \mid S \Rightarrow_G^* w\} = L_g((V, P), \{S\}) \cap V_T^*$.

Beobachtung:
Man kann Grammatiken auch als Sprachanalysierer begreifen, indem man die von G akzeptierte Sprache als $\{w \in V_T^* \mid w \Rightarrow_G^* S\} = L_a((V, P), \{S\}) \cap V_T^*$ definiert. Dann ist $P \subseteq V^* \times (V^* \circ V_N \circ V^*)$.

Beispiele:

- Wir geben eine Grammatik für $L_1 := \{ a^n b^n c^n \mid n \geq 1 \}$ an.
  $G = (V_N, V_T, S, P)$ mit $V_N = \{ R, S, T \}$, $V_T = \{ a, b, c \}$ und

  $$\begin{aligned} P = \{ S &\to abc \mid aRbc, \\ Rb &\to bR, \\ Rc &\to Tbcc, \\ bT &\to Tb, \\ aT &\to aaR \mid aa \} \end{aligned}$$

  Wie kann man $L_1 = L(G)$ zeigen? Z.B. kann man induktiv für alle $i \geq 1$ (bzw. $n \geq 0$) beweisen, daß $L(i) := \{ w \in (V_N \cup V_T)^* \mid S \Rightarrow_G^{(i+1)i-1} w \} = \{ a^i b^i c^i, a^i R b^i c^i \}$ sowie $L[n] := \{ w \in V_T^* \mid S \Rightarrow_G^n w \} \neq \emptyset \iff \exists i \geq 1 : n = (i+1)i - 1$ gilt. Daraus folgt dann unmittelbar die Behauptung.

Der Induktionsanfang ist trivial. Gilt die Behauptung für ein gewisses $i \geq 1$, so finden wir als einzig mögliche Ableitungen:

$$a^i R b^i c^i \Rightarrow_G^i a^i b^i R c^i \Rightarrow_G^1 a^i b^i T b c^{i+1} \Rightarrow_G^i a^i T b^{i+1} c^{i+1} \Rightarrow_G^1 \begin{cases} a^{i+1} b^{i+1} c^{i+1} \\ a^{i+1} R b^{i+1} c^{i+1} \end{cases}$$

Wegen $(i+2)(i+1) - 1 - [(i+1)i - 1] = 2(i+1)$ folgt die Behauptung für $L(i)$. Aus der Eindeutigkeit der Ableitungsfolge ergibt sich auch sofort die Behauptung betreffs $L[n]$.

- Wir geben eine Grammatik für $L_2 := \{ ww \mid w \in \{ a, b \}^* \}$.
  $G = (V_N, V_T, S, P)$ mit
  $V_N = \{ S, A, B, X, Y, Z \}$, $V_T = \{ a, b \}$ und

  $$\begin{aligned} P = \{ & S \to XYZ, \\ & XY \to aXA \mid bXB \mid \lambda, \\ & Aa \to aA, \\ & Ab \to bA, \\ & Ba \to aB, \\ & Bb \to bB, \\ & AZ \to YaZ, \\ & BZ \to YbZ, \\ & aY \to Ya, \\ & bY \to Yb, \\ & Z \to \lambda \} \end{aligned}$$

  zum Induktionsschritt:

  $$wXYwZ \Rightarrow_G^1 waXAwZ \Rightarrow_G^{|w|} waXwAZ \Rightarrow_G^1 waXwYaZ \Rightarrow_G^{|w|} waXYwaZ$$

Eine Grammatik $G = (V_N, V_T, S, P)$ ist

- vom *Typ 0* $\iff$ true
  (d.h., jede Grammatik ist vom Typ 0).

- vom *Typ 1* (*kontextsensitiv*) $\iff$
  $\forall x \to y \in P : (\exists \alpha, \beta \in V^*, A \in V_N, v \in V^+ : x = \alpha A \beta \wedge y = \alpha v \beta) \vee$
  $(x = S \wedge y = \lambda \wedge P \subseteq V^* V_N V^* \times (V \setminus \{S\})^*)$ $[*]$
  (d.h. die Regeln haben die Form $\alpha A \beta \to \alpha v \beta$ bzw. $S \to \lambda$ und S kommt in keiner rechten Regelseite vor).

- vom *Typ 2* (*kontextfrei, algebraisch*) $\iff$ $\forall x \to y \in P : x \in V_N$
  (d.h. die Produktionen haben die Form $A \to \alpha$).

- vom *Typ 3* (*rechts-/linkslinear, rational, regulär*) $\iff$
  $P \subseteq V_N \times ((V_N \cup \{\lambda\}) V_T^*)$ (linkslinear) oder
  $P \subseteq V_N \times (V_T^* (V_N \cup \{\lambda\}))$ (rechtslinear)
  (d.h. die Produktionen haben die Form $A \to w$ und $A \to Bw$ bzw. $A \to wB$ für gewisse $A, B \in V_N$, $w \in V_T^*$).

Bemerkungen

Kontextsensitive Regeln sehen nur deshalb so kompliziert aus, weil wir noch die Ableitung des leeren Wortes ermöglichen wollen; nur dafür ist die “Ausnahmeregel” $[*]$ gedacht.

Übungsvorschlag: Geben Sie eine kontextfreie Grammatik für $L := \{ a, b \}^* \setminus \{ ww \mid w \in \{ a, b \}^* \}$.

Die Begriffe algebraisch und rational entstammen einem weiteren Zugang zu den formalen Sprachen, nämlich dem sehr algebraisch-analytischen über formale Potenzreihen; dort finden sich tatsächlich erstaunliche Bezüge zu den entsprechenden Begriffen aus der Funktionentheorie.

Lemma: [λ-Freiheit] Zu jeder Typ-2-Grammatik $G = (V_N, V_T, S, P)$ gibt es eine Typ-2-Grammatik $G' = (V'_N, V_T, S', P')$ mit $L(G) = L(G')$, wobei gilt $\forall x \to y \in P' : (x \in V_N \wedge y \in V^+) \vee (x = S' \wedge y = \lambda \wedge P' \subseteq V_N \times (V \setminus \{S'\})^*)$, d.h. die Produktionen sind von der Form $A \to \alpha$ bzw $S' \to \lambda$, wobei dann S' in keiner Produktion auf der rechten Regelseite auftritt.

Beweis: $G = (V_N, V_T, S, P)$ sei eine Typ-2-Grammatik.
[Schritt 1] Bestimme die Menge $V_\lambda := \{ A \in V_N \mid A \Rightarrow_G^* \lambda \}$, also die Menge der Nonterminale, von denen aus λ abgeleitet werden kann:
$V_\lambda^0 := \{ A \in V_N \mid A \to \lambda \in P \}$
$V_\lambda^{i+1} := \{ A \in V_N \mid \exists A \to \nu \in P : \nu \in V_\lambda^{i*} \} \cup V_\lambda^i$
Es gilt: $V_\lambda^i \subseteq V_\lambda \subseteq V_N$ und $V_\lambda^i \subseteq V_\lambda^{i+1}, i = 0, 1, 2, \ldots$
$V_\lambda^i = V_\lambda^{i+1} \Rightarrow V_\lambda^j = V_\lambda^{j+1}, j \geq i$
Da V_N endlich existiert ein i_0 mit $V_\lambda^{i_0} = V_\lambda^{i_0+1}$ und es gilt $V_\lambda = V_\lambda^{i_0}$.
Es müssen also solange neue V_λ^i berechnet werden, bis keine Veränderungen mehr auftreten.

[Schritt 2] Konstruiere P' mit $P' := \{ A \to B'_1 \dots B'_m \mid A \to B_1 \dots B_m \in P \}$ mit $B'_i = \begin{cases} B_i & ; B_i \notin V_\lambda \\ B_i \text{ oder } \lambda & ; \text{sonst} \end{cases}$, aber $B'_1 \dots B'_m \neq \lambda$.
Mit anderen Worten: lösche alle λ-Produktionen und füge für alle Variablen, die nach λ abgeleitet werden können und auf rechten Regelseiten vorkommen, entsprechende Regeln hinzu, die diese Variablen rechts nicht mehr beinhalten. Falls $\lambda \notin L(G)$ (also falls $S \notin V_\lambda$), so setze $V'_N = V_N$ und $S' = S$. Falls $\lambda \in L(G)$ (also falls $S \in V_\lambda$), dann sei $V'_N = V_N \cup \{S'\}$, wobei S' ein neues Startsymbol ist, und die Regelmenge P' sei gleich $P \cup \{S' \to S, S' \to \lambda\}$.

Typ	Eigenschaft	Automatenmodell
0	Grammatiken ohne Einschränkungen	Turing-Maschine
1	kontextsensitive Grammatiken, monotone Grammatiken für alle Regeln $v \to w$ gilt $\|v\| \leq \|w\|$	linear beschränkter Automat
2	kontextfreie Grammatiken für alle Regeln $v \to w$ gilt: v ist Variable (die Produktionen haben die Form $A \to w$)	Kellerautomat
3	reguläre Grammatiken für alle Regeln $v \to w$ gilt: v ist Variable und w ist entweder ein Terminal oder ein Terminal gefolgt von einer Variable	endlicher Automat

Fragestellungen: Hierarchiefragen

Formale Sprachen (FS) untersuchen Sprachbeschreibungsmechanismen (SBM) (Grammatiken, Automaten, Ausdrücke) gewisser, vorgegebener Bauart und damit Sprachklassen.

Man kann zwei oder mehrere SBM vergleichen, indem man die zugehörigen Sprachklassen (aufgefasst als Mengen von Sprachen) mengentheoretisch miteinander vergleicht. Dies führt insbesondere auf *Hierarchiefragen*.

Beispiel: Chomsky-Hierarchie:

$$\textbf{REG} \subsetneq \textbf{KF} \subsetneq \textbf{KS} (\subsetneq \textbf{ENT}) \subsetneq \textbf{RA}$$

oder kürzer:

$$\mathcal{L}_3 \subsetneq \mathcal{L}_2 \subsetneq \mathcal{L}_1 \subsetneq \mathcal{L}_0$$

Fragestellungen: Hierarchiefragen

Was sind natürliche Fragen in diesem Zusammenhang, eine “neue” Sprachklasse $\mathcal{L}$ betreffend ?

1. Wie ordnet sich $\mathcal{L}$ im Vergleich mit bekannten Sprachklassen, insbesondere aus der Chomsky-Hierarchie, ein ?

2. Oft werden Hierarchien von Sprachbeschreibungsmechanismen eingeführt; dann erhebt sich die Frage, ob diese Hierarchie auch strikt auf Sprachebene ist, oder welche möglichst genauen Trennergebnisse bewiesen werden können. Manchmal *kollabieren* scheinbare Hierarchien auch.

3. Spezieller gibt es häufig triviale (schwache) Inklusionsbeziehungen, und die Aufgabe besteht darin, die Striktheit der Inklusion nachzuweisen.

4. Ebenso interessant sind *Unvergleichbarkeitsergebnisse* zwischen einander fernstehenden Familien; $\mathcal{L}_a$ und $\mathcal{L}_b$ heißen *unvergleichbar*, wenn

$$(\mathcal{L}_a \cap \mathcal{L}_b) \cap (\mathcal{L}_a \setminus \mathcal{L}_b) \cap (\mathcal{L}_b \setminus \mathcal{L}_a) \neq \emptyset.$$

Fragestellungen: Universalität (Spezialfall der Hierarchiefragen)
Bekanntermaßen dienen sowohl Turing-Maschinen als auch Typ-0-Grammatiken zur Kennzeichnung von $\mathcal{L}_0$, sind also — gemäß der Church-Turing-These — *(berechnungs-)universell*.

Wir eine neue Methodik zur Durchführung von Berechnungen vorgeschlagen, so muss man sich fragen, ob hiermit ebenfalls “alles” berechnet werden kann.

Typische “neue” Gebiete in den letzten 15 Jahren waren: Quantencomputing und Biocomputing.

Fragestellungen: Entscheidbarkeitsfragen

Gegeben eine Sprachbeschreibung und / oder Teile der Sprache, soll algorithmisch eine bestimmte Eigenschaft nachgewiesen werden.

Beispiel: (Nicht-)Leerheit, (Un-)Endlichkeit: Also: Gegeben z.B. eine Grammatik (aus einem bestimmten fixierten Formalismus), beschreibt diese Grammatik (nicht) eine leere bzw. endliche Sprache ?

Erinnerung an Grundvorlesungen:

Satz: Das Endlichkeitsproblem für endliche Automaten ist entscheidbar.

Satz: Das Leerheitsproblem für endliche Automaten ist entscheidbar.

Erinnern Sie die Konstruktionen ? Wie “schwierig” sind die Algorithmen ?

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 $\mathrm{L} \in$ **ENT** gilt.
(2) lässt sich (oft) durch Beantwortung der Inklusionsfrage $\mathcal{L} \subseteq$ **ENT** beantworten, jedenfals im positiven Fall und wenn $\mathrm{L} \in \mathcal{L}$ bekannt ist.
Sind gewisse Abschlusseigenschaften (s.u.), insbesondere Schnitt mit einelementigen Sprachen, konstruktiv bekannt, 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 ?
Ein unentscheibares Beispiel liefert das Halteproblem für Turingmaschinen.

Weitere Entscheidbarkeitsfragen

Insbesondere bei der Betrachtung von Sprachklassen, die “schräg” zur Chomkyhierarchie liegen, sind Fragen der folgenden Bauart von Interesse:

Beschreibt die Grammatikfamilie $\mathcal{G}$ die “schräge” Sprachklasse $\mathcal{L}$, so mag man fragen:
— Gegeben $\mathrm{G} \in \mathcal{G}$, ist $\mathrm{L}(\mathrm{G})$ vom Typ i, $\mathrm{i} = 1, 2, 3$?
— Gegeben Typ-i-Grammatik G, ist $\mathrm{L}(\mathrm{G}) \in \mathcal{L}$?
Wir werden solche Fragen *Typisierungsprobleme* nennen.

Analog zum Wortproblem kann man auch (allgemeine / spezielle) Präfixprobleme, Teilwortprobleme etc. betrachten.

Abschlusseigenschaften

Diese Fragen zielen auf algebraische Eigenschaften von Sprachfamilien ab. Eine Sprachklasse $\mathcal{L}$ ist (u.a.) ein Mengensystem, und daher ist es naürlich zu fragen, ob $\mathcal{L}$ gegen gewisse Mengenoporationen abgeschlossen ist. 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.

Abschlusseigenschaften: Beispiel (Homo-)Morphismen

Erinnerung: In den Grundvorlesungen hatten wir gelernt, dass wir eine Abbildung $\mathfrak{h} : \Sigma \to \Gamma^*$ zu einer Monoidabbildung $\mathfrak{h} : \Sigma^* \to \Gamma^*$ erweitern können. so dass $\mathfrak{h}$ in dieser Deutung zu einem Monoidmorphismus wird.

Jede Abbildung von Wortmengen auf Wortmengen kann man zu einer Abbildung erweitern, die Sprachen auf Sprachen abbildet: $\mathfrak{h}(\mathrm{L}) = \{\mathfrak{h}(w) \mid w \in \mathrm{L}\}$.

Also betrachten wir als Operatorenmenge H die Menge der Wortmorphismen.

(Um mengentheoretischen Problemen aus dem Weg zu gehen, kann man sich auf ein abzählbares Grundalphabet einigen, aus dem alle konkreten endlichen Wortsprachenalphabete gewonnen werden.)

Offenkundig ist $\mathfrak{h} \in \mathrm{H}$ nicht für alle Sprachen definiert, sondern nur für solche, deren Wörter sämtlich in dem Definitionsbereich von $\mathfrak{h}$ liegen.

Welche der Typ-$\mathfrak{i}$-Sprachfamilien sind gegen Morphismen abgeschlossen ?

Wie sieht die Antwort aus, wenn wir nur *nicht-löschende Morphismen* zulassen, also verbieten, dass ein Zeichen auf das leere Wort λ abgebildet wird ?

Abschlusseigenschaften: Beispiel *inverse (Homo-)Morphismen*

Jede Abbildung von Wortmengen auf Wortmengen kann man noch auf eine andere Weise zu einer Abbildung erweitern, die Sprachen auf Sprachen abbildet: $\mathrm{h}^{-}(\mathrm{L}) = \{w \mid \mathrm{h}(w) \in \mathrm{L}\}$.

Welche der Typ-i-Sprachfamilien sind gegen Morphismen abgeschlossen ?
Wie kann man das beweisen ?

Spezielle **Abschlusseigenschaften**:

Die regulären Sprachen gelten als besonders grundlegende Sprachfamilie.

Daher untersucht man oft auch, ob eine vorgelegte Sprachfamilie $\mathcal{L}$ abgeschlossen ist gegen Durchschnitt mit regulären Sprachen,

Wie sieht dies für die Typ-i-Sprachfamilien aus und wie beweist man dies ?

Noch ein paar Begriffe

Eine Sprachfamilie heiß *Trio*, wenn sie gegen nicht-löschende Morphismen, inverse Morphismen und Durchschnitt mit regulären Sprachen abgeschlossen ist. Ein Trio heißt *voll*, wenn es auch noch gegen beliebige Homomorphismen abgeschlossen ist.

Ein Trio heißt *abstrakte Familie von Sprachen*, kurz (aus dem Englischen) AFL, wenn es gegen Vereinigung, Konkatenation und Kleene Stern abgeschlossen ist.

Wie sieht das mit den Typ-$\mathtt{i}$-Sprachen aus ?

Kombinatorische Eigenschaften

Zumindest für “einfache” Sprachfamilien lassen sich oft Eigenschaften finden, die Sprachen aus solch einer Sprachfamilie auszeichnen (aber im Allgemeinen nicht kennzeichnen).
Beispiel: Die Pumping-Lemmata für reguläre und kontextfreie Sprachen sind kombinatorische Eigenschaften.

Kombinatorische Eigenschaften dienen oft dazu, von gewissen Sprachen zu zeigen, dass sie nicht zu der gerade interessierenden Sprachfamilie gehören. Im Beweisgang nimmt man an, dass (im Widerspruch zur Behauptung) die betrachtete Sprache zur nämlichen Sprachfamilie gehörte, weshalb ihr die bekannte kombinatorische Eigenschaft zukommt. Nun wird gezeigt, dass die betrachtete Sprache eben diese Eigenschaft nicht besitzen kann.
$\leadsto$ Wichtiges Hilfsmittel für Beweis der Striktheit von Inklusionen und von Unvergleichbarkeitsergebnissen.

Kombinatorische Eigenschaften: Normalformen

Normalformen kann man auch als kombinatorische Eigenschaften im weiteren Sinne auffassen.

Beispiel: Chomsky-Normalform für kontextfreie Grammatiken; der Beweis der Chomsky NF geht meist über andere (schwächere Normalformergebnisse

Jedenfalls gestatten solche Normalformen oft vereinfachte Beweisführungen, indem nämlich nur noch Grammatiken in Normalform zu betrachten sind.

Kennen Sie die Greibach-NF für Typ-2 oder die Kuroda-NF für Typ-0 ?

Lemma: [Normalform]Zu jeder Typ-i-Grammatik $G = (V_N, V_T, S, P), i = 0, 1, 2$ existiert eine äquivalente Typ-i-Grammatik $G' = (V'_N, V_T, S, P')$ mit $P' \subseteq {V'_N}^+ \times {V'_N}^* \cup V'_N \times V_T$.

Beweisidee: Führe für jedes Terminal a in G eine neue Variable $\hat{a}$ in G' ein, die allein a ableitet. Dann ersetze alle Vorkommen von Terminalen auf rechten Regelseiten durch diese neuen Variablen. Setze $V'_N := V_n \cup \hat{V}_T$, wobei $\hat{V}_T$ ein „neues“ Alphabet sei mit $\hat{V}_T := \{\, \hat{a} \mid a \in V_T \,\}$ und $\hat{V}_T \cap V = \emptyset$.
Definiere den Homomorphismus $\varphi : V^* \to {V'_N}^*$ durch $\varphi(x) := \begin{cases} A & ; x = A \in V_N \\ \hat{a} & ; x = a \in V_T \end{cases}$.
Setze weiter $P' := \{\, \varphi(x) \to \varphi(y) \mid x \to y \in P \,\} \cup \{\, \hat{a} \to a \mid a \in V_T \}$.
Offenbar gilt: $L(G) = L(G')$. (Die eine Richtung ist klar: $v \in L(G) \iff S \Rightarrow_G^* v \iff S \Rightarrow_{G'}^*, \varphi(v) \Rightarrow_{G'}^*, v$ woraus folgt: $v \in L(G')$. Die andere Richtung ist aufwendiger, tut aber nichts zum Verständnis.)

Lemma: [links-/rechtslineare Grammatiken] Zu jeder linkslinearen Grammatik G existiert eine äquivalente rechtslineare Grammatik G' mit $L(G) = L(G')$. Ebenso gibt es zu jeder rechtslinearen Grammatik eine äquivalente linkslineare.

Beweis: Es sei $G = (V_N, V_T, S, P)$ eine linkslineare Grammatik mit $P \subseteq V_N \times V_N V_T^* \cup V_N \times V_T^*$. Setze
$V'_N := V_N \cup \{S'\}$, S' neues Symbol, und
$P' := \{S' \to wA \mid A \to w \in P, w \in V_T^*\} \cup \{B \to wA \mid A \to Bw \in P\} \cup \{S \to \lambda\}$.
Dann ist $G' := (V'_N, V_T, S', P')$ äquivalent zu G, denn gilt in G
$S \Rightarrow_G^1 A_1 w_1 \Rightarrow_G^1 A_2 w_2 w_1 \Rightarrow_G^* A_{n-1} w_{n-1} \dots w_2 w_1 \Rightarrow_G^1 w_n \dots w_1$, so gilt in G'
$S' \Rightarrow_{G'}^1 w_n A_{n-1} \Rightarrow_{G'}^1 w_n w_{n-1} A_{n-2} \Rightarrow_{G'}^* w_n w_{n-1} \dots w_1 S \Rightarrow_{G'}^1 w_n w_{n-1} \dots w_1$.
Entsprechendes gilt auch für die Umkehrung.

Quintessenz

Die Lemmata stellen sicher, daß wir o.B.d.A. davon ausgehen können, dass im folgenden betrachtete Grammatiken in Normalform vorliegen, d.h., die Produktionen von haben die Form
$A_1 \dots A_n \to B_1 \dots B_m$, A_i, B_i Nonterminale, $n > 0$, $m \geq 0$ bzw.
$A \to a$, A Nonterminal, a Terminal, (oder sie sind linkslinear).

Was haben wir davon ?

Satz: [Abgeschlossenheit] Jedes $\mathcal{L}_i$, $i = 0, 1, 2, 3$, ist abgeschlossen unter den *regulären Operationen* $\vee$ (Vereinigung),$\circ$ (Konkatenation), * (Kleene-Stern) und enthält die Familie FIN der endlichen Sprachen.

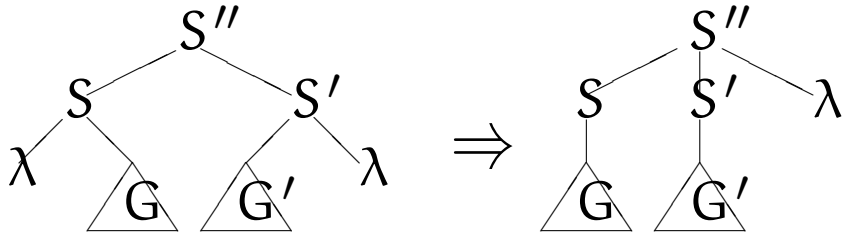
Beweis: $G := (V_N, V_T, S, P)$ und $G' := (V'_N, V'_T, S', P')$ seien Typ-i-Grammatiken in Normalform. O.B.d.A. gelte $V_N \cap V'_N = \emptyset$.*

(1) Vereinigung

- $i \neq 1$: $L(G) \cup L(G')$ wird von der Grammatik
  $G'' := (V_N \cup V'_N \cup \{S''\}, V_T \cup V'_T, S'', P \cup P' \cup \{S'' \to S, S'' \to S'\}$ erzeugt, denn
  $x \in L(G) \cup L(G') \iff S \Rightarrow^*_G x \vee S' \Rightarrow^*_{G'} x \iff S'' \Rightarrow^*_{G''} x.$

*Dies kann z.B. durch „Markieren“ der Nonterminale geschehen, also durch das Bilden von Kreuzprodukten $(a, 1), a \in V_N$ bzw. $(b, 2), b \in V'_N$.

- $i = 1$: Problem: falls P oder P' die Regel $S \to \lambda$ bzw. $S' \to \lambda$ enthält (also wenn gilt $\lambda \in L(G) \cup L(G')$), so ist das so definierte G'' keine Typ-1-Grammatik.
  Dann setze $P'' := (P \setminus \{S \to \lambda\}) \cup (P' \setminus \{S' \to \lambda\}) \cup \{S'' \to S, S'' \to S', S'' \to \lambda\}$.



(2) Konkatenation

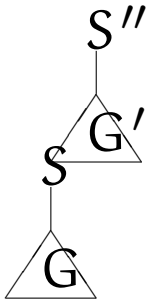
Wir werden im folgenden Kapitel sehen, daß der Konkatenationsabschluß hier nicht explizit gezeigt werden müßte aufgrund des Abschlusses der Typ-i-Sprachen unter Vereinigung, Stern und den Trio-Operationen.

- $i = 3$: $G'' := \{ V_N \cup V'_N, V_T \cup V'_T, s', P \cup \{ A' \to Sw \mid A' \to w \in P', w \in V_T^* \} \cup \{ A' \to B'w \mid A' \to B'w \in P' \})$ ist Typ-3-Grammatik und erzeugt $L(G) \circ L(G')$:

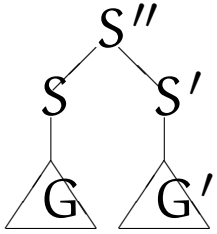
  $$\left.\begin{array}{l} S' \Rightarrow^*_{G'} w_2 \iff S' \Rightarrow^*_{G''} Sw_2 \\ S \Rightarrow^*_G w_1, \text{ also } S' \Rightarrow^*_{G''} w_1 w_2 \end{array}\right\} \Rightarrow L(G) \circ L(G') \subseteq L(G'')$$

  $$\left.\begin{array}{l} S' \Rightarrow^*_{G''} v, \text{ also } S' \Rightarrow^*_{G''} Sv_2 \Rightarrow^*_{G''} v_1 v_2 = v \\ \Rightarrow v_1 \in L(G), v_2 \in L(G') \end{array}\right\} \Rightarrow L(G'') \subseteq L(G) \circ L(G')$$

  Zunächst wird also das Teilwort aus G' abgeleitet, dann das Teilwort aus G („sequentiell“).



- $i = 0, 2$: $G'' := (V_n \cup V'_N \cup \{S''\}, V_T \cup V'_T, S'', P \cup P' \cup \{S'' \to SS'\})$ ist Typ-i-Grammatik und erzeugt $L(G) \circ L(G')$.
  Es wird gleichzeitig das Teilwort aus G und das Teilwort aus G' erzeugt („parallel“). Im Fall $i = 0$ können keine Interferenzen zwischen linker und rechter Teilableitung entstehen, da $V_N \cap V'_N = \emptyset$ und G, G' in Normalform. Im Fall $i = 2$ gibt es diese Probleme offensichtlich nicht, da die linken Regelseiten nur ein Nichtterminal enthalten (und die sind in G und G' disjunkt).



- $i = 1$:

Fall 1: $\lambda \notin L(G) \cup L(G')$

Also gilt weder $S \to \lambda \in P$ noch $S' \to \lambda \in P'$. Dann funktioniert die Konkatenation wie im Fall $i = 0$.

Fall 2: $\lambda \in L(G), \lambda \notin L(G')$

$L(G) \circ L(G') = (L(G) \setminus \{\lambda\}) \circ L(G') \cup L(G')$.

$P'' := (P \setminus \{S \to \lambda\}) \cup P' \cup \{S'' \to SS', S'' \to S'\}$

Fall 3: $\lambda \notin L(G), \lambda \in L(G')$

$L(G) \circ L(G') = L(G) \circ (L(G') \setminus \{\lambda\}) \cup L(G)$.

$P'' := P \cup (P' \setminus \{S' \to \lambda\}) \cup \{S'' \to SS', S'' \to S\}$

Fall 4: $\lambda \in L(G), \lambda \in L(G')$

$L(G) \circ L(G') = (L(G) \setminus \{\lambda\}) \circ (L(G') \setminus \{\lambda\}) \cup (L(G) \setminus \{\lambda\}) \cup (L(G') \setminus$

$\{\lambda\}) \cup \{\lambda\}\, P'' := (P \setminus \{S \to \lambda\}) \cup (P' \setminus \{S' \to \lambda\}) \cup \{S'' \to SS', S'' \to S, S'' \to S', S'' \to \lambda\}$

(3) Kleene'sche Hülle†

- $i = 3$: $G'' := (V_N \cup \{S''\}, V_T, S'', P'' := P \cup \{S'' \to \lambda, S'' \to S\} \cup \{A \to Sw \mid A \to w \in P, w \in V_T^*\})$ ist eine Typ-3-Grammatik und erzeugt $L(G)^*$.

- $i = 2$: $G'' := (V_N \cup \{S''\}, V_T, S'', P'' := P \cup \{S'' \to \lambda, S'' \to S''S\})$ ist eine Typ-2-Grammatik und erzeugt $L(G)^*$.

†Insbesondere hier ist es lehrreich sich klarzumachen, warum für die einzelnen Klassen der Chomsky-Hierarchie unterschiedliche Konstruktionen angegeben werden. Konkrete Frage: Warum könnte die Typ-2-Konstruktion bei Typ-1 fehlschlagen?

- $i = 0, 1$: $G'' := (V_N \cup \{ S'', \hat{S} \}, V_T, S'', P'' := \tilde{P} \cup \{ S'' \to \lambda, S'' \to S, S'' \to \hat{S}S \} \cup \{ \hat{S}a \to Sa, \hat{S}a \to \hat{S}Sa \mid a \in V_T \})$, wobei $\tilde{P} = P$ im Falle $i = 0$ und $\tilde{P} = P \setminus \{S \to \lambda\}$ im Falle $i = 1$;

  (a) $L(G)^* \subseteq L(G'')$

  $$S'' \Rightarrow^1_{G''} \begin{cases} \lambda \text{ entspricht } (L(G))^0 \\ S \text{ erzeugt } (L(G))^1 \\ \hat{S}S \Rightarrow^*_{G''} \hat{S}\,L(G) \Rightarrow^1_{G''} \begin{cases} S\,L(G) \text{ erzeugt } (L(G))^2 \\ \hat{S}S\,L(G) \Rightarrow^1_{G''} \ldots \end{cases} \end{cases}$$

  (b) $L(G'') \subseteq L(G)^*$

  Der entsprechende formale Beweis würde weitere 2-3 Folien beanspruchen und sei hier “gespart”.

Es bleibt noch zu zeigen: $\mathrm{FIN} \subseteq \mathcal{L}_i$.
Wegen der bekannten Chomsky-Hierarchie und der Abgeschlossenheit gegenüber regulären Operationen genügt es zu zeigen, daß $\{\lambda\} \in \mathcal{L}_3, \emptyset \in \mathcal{L}_3$ und $\{a\} \in \mathcal{L}_3$ für alle a aus einem terminalen Grundvorrat, denn wenn $L \in \mathrm{FIN}$, dann $L = \emptyset$ oder $L = \{\lambda\} \cup F$ oder $L = F$, wobei F eine nichtleere endliche Menge ist mit $\lambda \notin F$.
Setze $F := \{\nu_1, \nu_2, \ldots, \nu_k\}$, $\nu_i := a_{i_1} a_{i_2} \ldots a_{i_{m(i)}}$, $m(i) := |\nu_i|$, $a_{i_j} \in V_T$, dann ist
$F = \{a_{1_1}\} \circ \{a_{1_2}\} \circ \ldots \circ \{a_{1_{m(1)}}\} \cup \{a_{2_1}\} \circ \{a_{2_2}\} \circ \ldots \circ \{a_{2_{m(2)}}\} \cup \ldots \cup \{a_{k_1}\} \circ \{a_{k_2}\} \circ \ldots \circ \{a_{k_{m(k)}}\}$.
Es gilt: $\emptyset = L((\{S\}, \{a\}, S, \emptyset))$,
$\{\lambda\} = L((\{S\}, \{a\}, S, \{S \to \lambda\}))$ und
$\{a\} = L((\{S\}, \{a\}, S, \{S \to a\}))$.

Kombinatorische Eigenschaften: Beschreibungskomplexität

Bei diesem Teilgebiet der Formalen Sprachen geht es darum herauszubekommen, wie weit man die “Ressourcen” beschränken kann, ohne die Beschreibungsmächtigkeit insgesamt zu verringern.
Man kann natürlich auch Sprachklassen betrachten, die sich durch solche eingeschränkten Ressourcen definieren.

Typische Ressourcen sind: # Nichtterminale, # Regeln, . . .

Mit dem Komplexitätsmaß “maximale Länge einer rechten Seite einer Regel” lässt sich die Chomsky-NF für Typ-2 auch als Beschreibungskomplexitätsergebnis lesen.
Wie lautet so ein Ergebnis im Detail ?

Es gibt auch nette Bezüge zur Berechnungs-Komplexität.

Algorithmische Aspekte

(Fast) alle bislang besprochenen Fragestellungen an Formale Sprachen (Sprachfamilien, Grammatikfamilien) lassen sich algorithmisch beantworten, d.h., es gibt einen Lösungsalgorithmus für das genannte Problem.
$\leadsto$ Frage nach der Komplexität der Fragestellung

Beispiel: Das Nichtleerheitsproblem für endliche Automaten entspricht offenbar der Frage nach der Existenz eines Pfades von dem Startzustand im Automatengraphen zu einem der Endzustände. Welche Algorithmen kennen Sie hierfür (aus anderen Vorlesungen) ? Mit welchem Platz kommt man aus ?

Man kann aber auch gänzlich andere algorithmische Fragen stellen, z.B. die nach der automatischen Lernbarkeit von Sprachfamilien (unter unterschiedlichen Modellannahmen).

Grundprobleme bei Sprach- und Grammatikfamilien & ihre Komplexitäten

	Typ 3	Typ 2	Typ 1	Typ 0
$\in$	NC^1	SAC^1	PSPACE	unentsch.
$\emptyset$	NL	P	unentsch.	unentsch.
$\equiv$	PSPACE	unentsch.	unentsch.	unentsch.

ACHTUNG (nochmals):
Lesen Sie diese Tabelle (u.ä. aus der Literatur) “geeignet”, d.h., unter Zugrundelegung “vernünftiger” Sprachbeschreibungsmechanismen.