Automatentheorie
SS 2001 - Prof. Indermark

Vorlesungsnotizen und Erläuterungen zum Skript von Matthias Hensler
URL: http://s-inf.de/

Stand: 13.08.2001 15:47

WARNUNG: Das hier sind meine eigenen Notizen zu der Vorlesung, diese sind nicht fehlerfrei oder vollständig! Ich empfehle dazu das Skript von Matthias Hensler (MH) unter http://s-inf.de .

INHALT

Kapitel 1: Alphabete, Wörter, Sprachen

1.1 Einführung

Σ = Alphabet
Σ* = alle Wörter
a = Zeichen
ε = leeres Wort
L = Sprache (Menge von Wörtern)

1.2 Operation auf Menge von Worten

- **Verkettung** von Wörtern (direkt aneinanderhängen)
  - klar: (neutrales Element ε) (man kann Klammern setzen)
- **Länge** eines Wortes
- **Potenzen** eines Wortes = w^3 = www
- **Spgelebild** eines Wortes
  - Algorithmus: letzten Buchstaben nach vorne, und mit dem Rest das gleiche tun

1.3 Grundbegriffe

p(Σ*) = Potenzmenge von Σ = Menge der formalen Sprachen über Σ . (z.B. URL's, ...)

1.4 Operationen auf p(S*)

- **Boolsche Operationen** von Sprachen L: Vereinigung, Schnitt, o Komplement
- **Komplexprodukt „x"** : Ein Wort aus der ersten Sprache (L_1), eins aus der zweiten.
- **Potenzen** einer Sprache: $\mathbf{L^m}$ = m-mal ein Wort aus der Sprache L. (?)
- **Stern** einer Sprache: $\mathbf{L^*}$**:** beliebig viele Wörter aus der Sprache L hintereinander.
- **Spiegelbild** einer Sprache: $\mathbf{L^R}$: alle Wörter rückwärts sind erlaubt. (?)

Kapitel 2: Reguläre Ausdrücke und endliche Automaten

2.1 Reguläre Ausdrücke

mit RegEx eine undendliche Sprache endlich beschreiben (z.B. „alle A-Folgen" oder Suchmaschine: „Mike or John", bzw. „(Mike | John)": alle Zeichenfolgen, in denen mike oder john vorkommt).

2.1.1 Definition (Syntax) der RegEx

Also. Eine RegEx umfasst

1. Das **leere** Wort
2. **alle** Buchstaben
3. **Oder** („Alternation")
4. **Verkettung** („concatenation")
5. **Wiederholung** („Repeteion")

2.1.2 Vereinfachte Schreibweise

Klammern weglassen, wo man Präzedenrgeln hat („Punkt vor Strich", „Mal weglassen"
Hinweis: „RegEx" ist auch eine formale Sprache!

2.1.3 Semantik von RegEx.

α = RegEx. = „**Muster, Pattern**“
L(α) = **[[** a **]]** = alle Wörter, die auf den RegEx passen. = „**Match**“

Def. für reguläre Sprachen. (wie RegEx).

2.1.4 Standardprobleme für RegEx(S)

1. Matching Problem: Liegt ein Wort in der Sprache?
2. Äquivalenzproblem: Leisten 2 RegEx genau das gleiche?
3. Leerheitsproblem: Erkennt diese RegEx überhaupt irgendein Wort?

2.2 Deterministische endliche Automaten (DFA)

deterministisch = wenn man vor einer Abzweigung steht, ist es eindeutig, wohin man gehen kann.

endlicher, deterministischer Automat: **gohtisch_A = < Q,** S**,** d**, q_0, F >**

Q = Zustandsmenge (**q_1**, **q_2**, …)
S = Eingabealphabet (a, b, c, …) – das, was an den Zustandsübergängen steht.
d = Transitionsfunktion (Aus Zustand **q_1** unter Eingabe von **a** in Zustand **q_2**)
q_0 = Anfangszustand (q_0) – Es gibt nur genau einen Anfangszustand !!
F = End-Zustandsmenge. (q_8, q_9)

Erweiterte Transitionsfunktion:

1.) Mit dem leeren Wort als Eingabe kann man immer im gleichen Zustand bleiben
2.) (q,wa) bedeutet: man geht von q aus zuerst den Weg „w“ und dann den Weg „a“. Den Zwischenzustand braucht man nicht zu benennen.

geschwungen_L = Sprachen, die durch DFA **erkannt** werden.

2.3 Nicht-deterministische endliche Automaten (NFA)

man hat von einem Zustand aus die Möglichkeit, viele Wege zu gehen.
Außerdem hat man eine **Epsilon-Hülle**: dies sind Zustandsübergänge, die man einfach so – ohne eine Eingabe machen kann, wenn man Lust dazu hat. (Will sagen: man kann sie machen, einen oder den anderen, oder auch gar nicht – dies ergibt eben die vielen Möglichkeiten.)

2.3.1 Der Potenzmengenautomat

Ein Automat, der vorher die Wörter „1“, „2“ und „3“ akzeptiert, akzeptiert jetzt auch noch die Wörter „12“, „23“, „13“, „21“, „31“, „32“, „123“, Epsilon (leeres Wort).

2.4 Synthese und Analyse endlicher Automaten

2.4.1 Synthese & Der Algorithmus von Thompson

Aus einer RegEx einen endlichen Automaten konstruieren.

Der Algorithmus von Thompson:

Jeder endliche Automat ist beschreibbar durch eine Turingmaschine, die sich nur nach rechts bewegen kann.

Achtung:
Anfangszustand muss Quelle sein (man darf nie mehr rein),
Endzustand muss Senke sein (man darf nie mehr raus).

Vereinigung von 2 Automaten (beide akzeptieren):
Einfach einen neuen Startzustand, und ε - Übergänge in beide Automaten. Und von den beiden Endzuständen aus auch wieder per ε - Transition in einen einzigen Endzustand.

Aneinanderreihung von 2 Automaten:
Anfangszustand des zweiten := Endzustand des ersten.
Seiteneffekte dürfen nicht sein (also man darf nicht mehr vom 2. Automaten in den 1. zurück.) Nach Def. Darf das en nicht sein, da man nie mehr in den Ausgangszustand zurückdarf. Wenn man ganz sicher gehen will, macht man ε - Transitionen dazwischen.

Stern: (beliebig viele Wörter oder keins aus L)
Wir haben einen Automaten, der Alpha erkennt. Nun wollen wir Alpha beliebig oft erkennen.
Lösung: Wir packen einen neuen Anfangszustand davor, von dem man per ε in den alten Anfang kommt, und einen neuen Endzustand, in den man per ε vom alten Endzustand kommt. Nun darf man von Alt-Ende per ε zu Alt-Anfang (Wiederholung), und von Neu-Anfang zu Neu-Ende per ε (gar kein Wort)

! Enlicher Automat = Turingmaschine, bei der das Band nur nach rechts gehen kann !.

2.4.2 Analyseaufgabe („W_{ij}^{k}“)

Man möchte von einem Automaten zu einem regulären Ausdruck kommen.
Sprache regulär → Automat existiert.

Wir nehmen uns alle Wörter, die der Automat versteht, und verknüfpen sie mit „oder“. Fertig.

W_{ij}^{k} bedeutet: Alle möglichen Wörter (bzw. Wege), vom Zustand i zum Zustand j zu kommen, und dabei maximal die Zustände 0 bis k als Zwischenzustände zu benutzen.

W_{ij}^{k} sind also die Wörter, die durch die reguläre Expression gematched werden.

BEISPIEL: W_{22}^{1} = Alle Wege direkt von 2 nach 2 (also „Rundpfeile“ und Epsilon-Transitionen), sowie alle Umwege über Zustand 1, wobei man im Zustand 1 beliebig oft bleiben kann.
Dies kann man sich rekusiv definieren, indem man erst alle W_{ij}^{0} definiert, und danach alle weiteren induktiv aufbaut (siehe Übung 2 Blatt 3).

2.5 Das Pumping-Lemma

hiermit kann man nicht-reguläre Sprachen nachweisen:

Zu jeder regulären Sprache existiert eine Schranke k, so dass zu jedem Wort, das mindestens k Buchstaben hat, eine Zerlegung „uvw“ existiert, wobei

- **v** nicht leer ist
- „**uv**“ maximal k Buchstaben hat
- **uw** und $\mathbf{uv^i w}$ und auch in der Sprache liegt

Zur Erklärung:
Wir betrachten einen Automaten mit k Zuständen. Jedes Wort, das mehr Buchstaben hat, als wir Zustände haben, muss also im Automaten mindestens eine Schleife durchlaufen.
u = vor der Schleife, v = die Schleife, w = nach der Schleife.

- v ist nun nicht leer, (ist ja unsere Schleife),
- uv hat maximal k Buchstaben (wir haben ja nur k Zustände und bisher keine Wiederholung gemacht), und
- uv^iw liegt auch in der Sprache (wir können ja die Schleife v beliebig oft durchlaufen).

k, also die Anzahl der Zustände im Automaten, heisst „Pumping-Index“.

Auf Deutsch: wir können immer eine Schleife finden, die wir beliebig oft durchlaufen.

Gegenbeispiel: $\mathbf{a^n b^n}$:
$a^k b^k$ = uvw.
Hierfür muss nun eine Zerlegung existieren in Vorderteil – Pumping-Teil – Schlussteil.
Dies geht hier nicht: da |uv| kleiner k sein muss, liegt uv komplett im Bereich „a^k“. Jetzt nehmen wir das nicht-leere v raus, und „a^k“ wird kürzer als „b^k“ → Widerspruch.

2.6 Zustandsreduktion endlicher Automaten

Ziel: Konstruktion von Automaten mit minimaler Zustandszahl.

- Weglassen nicht erreichbarer Zustände
- Verschmelzen äquivalenter Zustände

2.6.1 Ableitung einer Sprache

Ableitung einer Sprache nach dem Wort w = Die Teile hinter dem x bei den Wörtern, die mit x beginnen.

Bei einer Sprache bekommt man ja immer wieder die Sprache selbst.

LEMMA: Die Anzahl der Ableitungen ist durch die Zahl der Zustände beschränkt.

2.6.2 Der Ableitungsautomat

Wir haben eine Sprache, die nach w abgeleitet ist. Dann leiten wir sie nach wa ab.
(beachte: die Zustände sind hier Sprachen!)

Beachte: wenn nach Abschneiden von w dasselbe rauskommt wie nach Abschneiden von v, so kommt auch nach Abschneiden von wa dasselbe raus wie nach Abschneiden von va.

Lemma: die Sprache, die der Automat A_L erkennt, ist genau L.

Korollar: in unserem Automat haben wir die Zustände ja gerade so gewählt, dass sie die Ableitungen sind. Der ist zustandsminimal, schließlich haben wir vorher schon festgestellt, dass die Anzahl der Ableitungen kleinergleich der Anzahl der Zustände ist. Q.e.d.

Zustandsreduktion: Die Konstruktion eines zustandsminimalen Automaten aus einem gegebenen Automaten durch 2 Schritte:

1.) Weglassen nicht erreichbarer Zustände
2.) (Hauptpunkt): Verschmelzen äquivalenter Zustände

2 Zustände sind äquivalent, wenn die Sprachen, die von diesen Zuständen erkannt werden, gleich sind. (Diese stellen dann praktisch die gleichen Ableitungen dar.)
(Jeder Zustand ist hier also eine Ableitung der Sprache!)

2.6.3 Der Faktorautomat

2.6.4 K-Äquivalenz und K-Äquivalenzmatrizen

2 Zustände heißen **k-äquivalent**, wenn genau alle maximal k-buchstabigen die Wörter, die in der Sprache liegen, von beiden Zuständen akzeptiert werden.

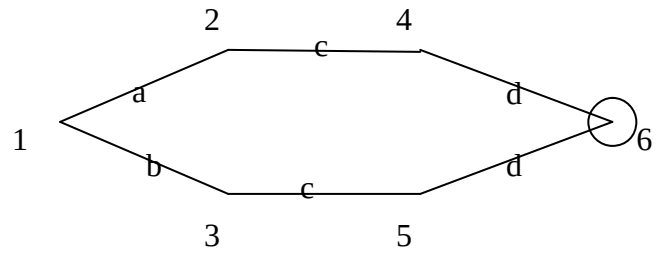
X-Koordinate: alle Zustände
Y-Kooridnate: alle Zustände.

1 = Zustandsübergang existiert, 0 = Zustandsübergang exisitiert nicht.

- Wenn q1 und q2 0-äquivalent sind, d.h. beide erkennen alle Wörter der Länge 0 aus der Sprache, dann sind beides Endzustände, weil man von diesen Zuständen mit einer Epsilon-Transition zum Ende kommt. (Das ein Wort akzeptiert wird, bedeutet ja, dass das Ende des Wortes an einem Endzustand liegt).
- Wenn 2 Zustände k+1-Äquivalent sind (d.h., die Wörter der Länge maximal k+1 werden von beiden Zuständen aus erkannt), dann sind die beiden Zustände 0-äquivalent, und alle Ableitungen nach beliebigen Buchstaben sind gleich. → klar.

2.6.5 Markierungsalgorithmus

Nehmen wir an, wir haben folgenden Graphen:



6 ist unser Endzustand.

1.) Markiere alle Zustandspaare „Endzustand – nicht Endzustand“

1						
2						
3						
4						
5						
6	X	X	X	X	X	
	1	**2**	**3**	**4**	**5**	**6**

2.) Betrachte alle noch nicht markierten Zustandspaare A B, und schaue, welche Zustände man von A und B (zusammen) erreichen kann. (B C). Wenn B C markiert ist, markiere auch A B. So oft durch alle Stellen durchlaufen, bis sich nichts mehr ändert. Die jetzt nicht markierten Zustandspaare sind äquivalent und können zusammengefasst werden.

1						
2	X					
3	X	-				
4	X	X	X			
5	X	X	X	-		
6	X	X	X	X	X	
	1	**2**	**3**	**4**	**5**	**6**

2.7 Entscheidbare Eigenschaften

2.7.1 Deterministische endliche Automaten

Wortproblem: passt das Wort auf diesen regulären Ausdruck?
(entscheidbar in w+1 Schritten).
Leerheitsproblem: passt gar kein Wort auf den regulären Ausdruck?
(alle Worte (alle kleiner q) durchschicken)
Äquivalenzproblem: sind 2 reguläre Ausdrücke äquivalent?
(alle Worte durchschicken)

b) Reguläre Ausdrücke, nicht-deterministische Automaten

2.7.2 Reguläre Ausdrücke, nicht-deterministische Automaten#

Wenn man diese 3 Probleme in DFA’s überführt, sind alle 3 Probleme entscheidbar.
Aber aufwendig.

2.8 Anwendungen

2.8.1 Endliche Automaten mit Ausgabe („Mealy-Automat“):

Zu jeder Eingabe gibt der Automat beim Zustandsübergang noch eine Ausgabe.
(Beispiel: Fahrkartenautomat)

2.8.2 Wortsuche in Texten (siehe Folie):

1.) NFA-Methode: u = aebwbwebba: Lauf durch den Automaten.
Wir betrachten immer ALLE Zustände, die wir theoretisch erreichen können:

{1}a{1}e{1,5}b{1,6}w{1,2}e{1,3}w{1,2}e{1,3,5}w{1,2}e{1,3,5}b{1,4,6}…

bedeutet:
Nach dem a kann ich in Zustand 1 sein.
Nach dem e kann ich in Zustand 1 oder 5 sein.
Nach dem b kann ich in Zustand 1 oder 6 sein.
Nach dem w kann ich in Zustand 1 oder 2 sein.
…

2.8.3 Verteilte Systeme / Parallele Prozesse

Zum Beispiel: Wenn Anna die Farbe nimmt und Bernd den Pinsel, haben wir einen „Deadlock“, da keiner der beiden malen kann.

2.8.4 Erweiterte Reguläre Ausdrücke: Schnitt und Komplement

Wie komplementiere ich einen nichtdeterministischen Automaten? (NFA)
(wichtig für Suchmaschinen!)

2.8.5 Zusammenhang reguläre Ausdrücke – While-Programme

(Zusammenhang Automaten – endliche Automaten – iterative Programme (Flußdiagramme))

While-Programme kann man als reguläre Ausdrücke schreiben!

Kapitel 3: Kontextfreie Grammatiken und Kellerautomaten

3.1 Kontextfreie Grammatiken (CFG)

kontextfrei = auf der linken Seite steht genau nur ein Nichtterminalsymbol, und sonst nichts.
Anwendung: XML, CSS, Compilerbau-Syntaxanalyse

Wir haben 4 Elemente: 2 Mengen (N und ?), ein Startsymbol aus N („S“) und ein P, das ein Element aus N hat und beliebig viele Elemente aus N oder ? .

Zeichenkonventionen: s. Skript.

Ableitung: Ersetzen eines Nichtterminalsymbols durch die rechte Seite.
Der Ableitungsschritt ist dann ein Tripel aus:

a) was vor dem Zeichen stand (Gamma 1)
b) was hinter dem Zeichen steht (Gamma 2)
c) Der Übergang Pi: Nichtterminalsymbol A → „rechte Seite“ Alpha.

G = Ableitungsrelation = alle Ableitungen, die vom Startsymbol abgeleitet werden können. („reflexive, transitive Hülle“)

Also: Ein Wort w liegt in der Sprache, wenn es abgeleitet werden kann. Ich wende ganz oft Pi an.

Beispiel: BNF (Backus-Naur-Form) (siehe Folie)
Beispiel: WHILE-Programme (siehe Folie)

3.1.1 Ableitungsbäume, Rechts- und Linksableitungen, Ein- und Mehrdeutigkeit

Normale, bekannte Ableitungsbäume.

18.05.2001

Ein Compiler baut aus dem Code (Zeichenkette) einen logischen Baum auf.
(HTML, XML, …)
Ableitung. Zwischen den Texten a und b steht das Nichtterminalsymbol Pi, das nun ersetzt wird durch ein Terminalsymbol:
Rechtsableitung: Ersetze das am weitesten rechts stehende Nichtterminalsymbol.
Linksableitung: Ersetze das am weitesten links stehende Nichtterminalsymbol.

Folgerung: Ein Ableitungsbaum hat genau eine Rechts- bzw. Linksableitung.

Def.: Eine Grammatik ist **eindeutig**, wenn es genau eine Rechtsableitung gibt. (oder Linksableitung). Sonst **mehrdeutig**. (also es gibt ein Wort, zu dem es verschiedene Ableitungsbäume gibt).
Bem.: Syntaktische Mehrdeutigkeit wird durch Präzenenzregeln beseitigt. (z. B. „Punkt vor Strich“)

3.2 Einseitig Lineare Grammatiken

Def.: Eine konextfreie Grammatik heißt **linkslinear**, wenn wir nur Regeln haben
A→Bw oder A→w. (entsprechend rechtslinear).

→ Einseitig lineare Grammatiken und nichtdeterministische Automaten sind das gleiche!
Korollar: diese sind Teil der Kontextfreien Grammatiken.

Es gibt jedoch eine kontextfreie Grammatik, die sich nicht durch linkslineare Grammatik ausdrücken lässt: (s. Skript, nachlesen!)

Lemma: Wenn L eine Sprache regulär ist, ist auch ihr **Spiegelbild** regulär.
Beweis: Zu einem deterministischen Automaten A über Sigma, der ja zu einer solchen Sprache gewählt werden kann, konstruieren wird einen weiteren Automaten A^R, der dann nicht mehr determinstistisch ist, aber das ist ja kein Problem, mit der Eigenschaft, dass dieser Automat genau die Sprache erkennt, die die Spiegelung von A ist.
Idee: wir ersetzen in dem Automaten einen solchen Übergang von Q nach Q' durch den gespiegelten Übergang („wir drehen den Pfeil um").
Und ergänze diesen NFA durch die folgende Konstruktion:
Wir nehmen einen neuen q0-Anfangszustand, und den lassen wir autonom in die Endzustände übergehen.

3.3 Normalformen

Vorlesung vom 22.05.2001

Kontextfreie Grammatiken werden sehr gerne für Programmiersprachen verwendet.
Gerne verwendet wird auch die „EBNF": extended Bakus-Naur-Form:
auf der rechten Seite der Regel steht nicht nur ein Wort, sondern ein regulärer Ausdruck.

Heute wollen wir versuchen, eine kontextfreie Grammatik auf eine einfache Struktur zurückzuformen.
Chompsky-Normalform: A→Aa: auf der rechten Seite stehen nur 2 Regeln.

3.3.1 Technisches Hilfsmittel: Vorgänger

X: Alle Symbole (Terminal und Nichtterminal)
wenn man aus Alpha Beta ableiten kann, nennt man Alpha „**direkten Vorgänger**" von Beta.
Wenn man mehrere Schritte gehen muss, ist Alpha ein **Vorgänger („Vorgängerabschluss")** von Beta. (können auch Null Schritte sein).

Sinn der ganzen Sache: LEMMA: Sei G eine kontektfreie Grammatik, und L eine Menge von Satzformen, dann gilt fogender Zusammenhang zwischen L und dem Vorgängerabschluss:
Wenn L regulär ist, dann ist auch der Vorgängerabschluss regulär! (also eine reguläre Sprache).

Beweis: Wenn L eine reguläre Menge ist, können wir einen nichtdeterministischen Automaten nehmen, der diese Menge repräsentiert.
Jetzt konstruiere ich einen, der nicht nur L erkennt, sondern auch den Vorgängerabschluss:
Konstruiere A': gleiche Zustandsmenge, gleiches Eingabealphabet, gleiche Zustdandsmengen: wir erweitern nur die Anzahl der Kanten.

1.) Die Kanten, die schon drin sind, lassen wir drin. (die Menge der Folgezusände.)
2.) Wenn wir eine Regel haben: A geht nach Alpha, in der Menge der Produktion P, und wir haben einen Übergang mit Alpha:
"Der Automat kann von q nach q' durch lesen von Alpha. Wenn ich den finde, ersetze ich ihn.". „Wir fügen genau das ein, was dem Vorgänger entspricht".

BEISPIEL: siehe Folie:

Ein Wort ist eine triviale reguläre Menge.

Def.:
Sei G eine kontextfreie Grammatik, und A ein Nichtterminlalsymbol. A heißt „**produktiv**" genau dann, wenn es ein Wort W aus Sigma* gibt, so dass wir eine Ableitung von A nach W haben.
A heisst „erreichbar", wenn es vom Startsymbol aus erreichbar ist. (wenn es also irgendwo in der Satzform ist).

Wir würden nie eine Grammatik konstruieren, die solche Eigenschaften hat.

Folgerung 1: A ist **produktiv** genau dann wenn A in dem Vorgängerabscnhluss von Simga* liegt.

Folgerung 2: A ist **erreichbar**, wenn das Startsymbol in dem Vorgängerabschluss einer Satzform isder das A auftritt:
Ich nehme einfach diese Menge der Satzformen, in denen ein A auftritt.
Hierzu kann ich einen endlichen Automaten bauen.

Def.: reduzierte Grammatik: jedes Nichtterminalsymbol A ist produktiv und erreichbar.
Oder (2. Fall): P ist leer.

Man kann eine Grammatik sehr einfach in eine solche reduzierte transformieren: Man schmeißt die nicht erreichbaren und nicht produktiven Symbole raus.
Fall A: S ist nicht produktiv. Wähle P' = leer.
Fall B: S ist produktiv. Ich lasse alle Nictterminalsymbole weg, die eine der beiden Eigenschaften nicht haben. (produktiv, erreichbar).

Zusätzlich lassen wir noch alle Regeln weg, in denen ein solches Nichtterminalsymbol auftreten. (da die ja jetzt gar keinen Sinn mehr machen).

Korrolar: Idee: „**Epsilon-frei**"
Es dürfen keine Epsilon-Regeln mehr vorkommen! (nur noch die eine Regel: „S → Epsilon", und S ist nie auf der rechten Seite!

Wie kann man eine Grammatik Epsilon-frei machen?
Lemma: X ----*----> epsilon: X ist die Vorgängermenge von epsilon.

3.3.2 Chompsky-Normalform (CNF)

…

3.3.3 Die Greibach-NF (GNF), Linksrekursion

Nur folgende Formen:

- $A \rightarrow aB_1 \ldots B_n$
- $A \rightarrow a$
- $S \rightarrow$ epsilon

Jede CFG kann man in die Greibach-Normalform (GNF) transformieren.

Hierfür müssen Regeln der Form A→A….. rausgeworfen werden:
(produktiv = ein Terminalbaum wird abgeleitet)

Vorher haben wir Regeln $A \rightarrow Ax \mid Ay$,: Wir können also beliebige xyyxxyx – Folgen bilden. Desweiteren haben wir Terminalregeln zum Abbruch: A→z.

Das schreiben wir jetzt andersrum: A→zT, T→xT | yT | x | y

3.4 Abschlusseigenschaften von CFL, Pumping-Lemma

Hilfsmittel: Substitutionssatz, Pumping-Lemma.

3.4.1 Substitutionssatz

Für jeden Buchstaben im abgeleiten Wort setze ich das Startsymbol einer anderen Grammatik. Ich hänge also verschiedene Grammatiken aneinander:

aus $A \rightarrow abc$ wird $A \rightarrow S_a S_b S_c$.
wenn nun abc in der Ausgangssprache liegt, liegt die Aneinanderreihumg von Wörtern aus $S_a S_b S_c$ in der sogenannten „abgeleiteten Sprache“.

Substitutionssatz: Ausgangssprache kontextfrei → abgeleitete Sprache kontextfrei.

3.4.2 Pumping-Lemma für CFL (uvwxy-Theorem)

Wir haben eine kontextfreie Sprache L.

Es gibt jetzt eine Grenze k, so dass für alle Wörter aus L …. … ???

3.5 Entscheidbare Eigenschaften von CFG

Satz: das Wortproblem und das Leerheitsproblem sind für kontextfreie Grammatiken entscheidbar.
(z.B.: Ist das Programm auch wirklich ein C – Programm?)

Beweis: w ist von den Grammatik g erzeugt genau dann, wenn das Startsymbol der Grammatik im Vorgängerabschluss der Sprache liegt, die grade aus dem Wort w besteht.

Bemerkung:

(i) Das Wortproblem ist in $O(n^3)$ (n=Länge des Wortes entscheidbar)
(ii) Das Äquivalenzproblem ist nicht entscheidbar: Man kann nicht feststellen, ob 2 Grammatiken die gleiche Sprache entscheiden.

Buchempfehlung: Simon Singh: „Fermats letzter Satz“

3.6 Kellerautomaten

3.6.1 Syntax

s. Skript

3.6.2 Semantik

Zur formalen Beschreibung benötige ich 3 Elemente:

1.) Q = Zustandsmenge
2.) Sigma* = Eingabealphabet
3.) Gamma* = Kellerspitze links

Dieses Tripel beschreibt den Gesamtzustand des Kellerautomaten.

„ Ich befinde mich im Zustand q, auf dem Band steht „w“, und im Keller steht „Alpha“. Jetzt kann ich in einen neuen Zustand q’ gehen, w’ auf das Band schreiben und Alpha’ auf den Keller schreiben.

Sigma-Schritt: arbeitet abhängig von der Eingabe auf dem Band
aw = aktuell auf dem Band das Zeichen a, dahinter der Rest w.
z-Alpha : z = oberstes Symbol auf dem Stapel, alles darunter = Alpha.

Epsilon-Schritt: Ändert nichts am Eingabeband.

Die von Q erkannte Sprache sind die worte, die von der Ausgangskonfiguration übergehen können in eine Konfiguration, …
Ich verlange, dass die Eingabe komplett gelöscht ist, d.h. dass ich die ganze Eingabe durchlaufen habe.

12.06.2001

Korollar: Die Menge P der Pascal-Programme ist nicht kontextfrei.
Beweis: L:={program T(output); var w:integer; begin w:=1 end. | w Î {a,b}* }

Also ist L Zeilmenge von P.
R sei bestimmt durch den regulären Ausdruck

1. Es gibt nur eine Möglichkeit, mit einem Kellersymbol weiterzugehen.
2. Wenn der Automat einen autonomen Schritt macht, dann darf kein Schritt in Abhängigkeit irgendeines Kellersymbols erfolgen.

Folgerung: zu jeder Konfiguration gibt es höchstens eine Folgekonfiguration. (also entweder Sigma-Schritt, oder Epsilon-Schritt, je nach dem, welches Kellersymbol wir gerade haben).

Die Klasse der Sigma-Sprachen, die von deterministischen Push-Down-Automaten erkannt wird, definieren wir als diejenige Klasse, die von solchen Automaten mit Endzustönden erkannt wird.

Diese Klasse ist echt größer als diejenigen SPrachenm, die mit einer der anderen Erkennungsmöglichkeiten erkannt werden (leerer Keller, sowie leerer Keller und Endzustände).

SATZ: Diese Klasse der von deterministischen Klassen erkennbaren Automaten ist unter Komplement abgeschlossen. - WIE SCHADE!

Beweisideen:
(deterministische Kellerautomaten haben es in sich).
Zu jedem A aus DPDA(Sigma) ist ein äquivalenter (also die gleiche Sprache erkennender) A-Schlange deterministisher Kellerautomat konstruierbar, so dass gilt:
Für jedes Eingabewort kann ich in folgendem Sinne eine Entscheidung herbeiführen:
Für jedes w aus Sigma-Stern gibt es einen Zustand q aus Q-Schlange und eine Kellerinschrift, mit folgender Einegschaft:
Wenn wir ein Wort w haben, lassen wir den Automaten auf dem Wort w arbeiten:
w auf dem Eingabeband, Startsymbol auf dem Keller. Irgendwann landet der in einer End-Konfiguration, d.h. eine Konfiguration ohne Folgekonfigutation.

Eine Konfiguration heisst **Schleifenkonfiguration**, wenn:
Für jedes I aus N ein entsprechender Zustand Q_i existiert, und ein Apha_i (eine Kellerinschrift), und die Alpha_i in der Menge niemals kürzer werden, (Keller bleibt gleich gross oder wächst).
Diese Eigenschaft ist entscheidbar.
Daraus folgt dann die Möglichkeit der Elemination. Möglichkeit.

3.7 Der Algorithmus von Cocke, Yonger, Kasami

Man verwendet dymanische Programmierung (man speichert Zwischenergebnisse):
n^3-Zeit, n^2-Platz.
oBdA wählen wir G in Chompsky-Normalform (ist so einfacher). (auf beliebige kontextfreie Grammatiken übertragbar, da man ja alle in CNF darstellen kann).

Sei G eine kontextfreie Grammatik in Chompsky-Normalform, (d.h. $A \to BC$, $A \to a$, $S \to \hat{I}$).
und w sei ein Wort $a_1 a_2 \ldots a_n$.

Für ein solches Wort definieren wir mal Teilwort: wir wollen ja wissen, ob das vom Startsymbol ableitbar ist.
Wir schauen erstmal, ob alle Buchstaben abgeleitet werden können. Dann schauen wir bei allen 2-Teilworten, dann 3-Teiloworten, etc:
Wir definieren ein Wort w_{ij} = das Wort, das bei Stelle i beginnt und bei Stelle j endet.
N_{ij} ist die Menge derjenigen Nichtterminalsymbole, aus denen dieses Teilwort abgeleitet werden kann.

Man kann diese Menge folgendermaßen berechnen:
Wir bestimmen zunächst die Mengen $N_{11}, N_{22}, \ldots, N_{nn}$. Dann können wir aus Kenntnis dieser Menge sehr leicht feststellen, welches die Nichtterminlasymbole sind, die Teilworte der Menge 2 ableiten: $N_{12}, N_{23}, \ldots, N_{n-1,n}$.
Daraus kann man dann $N_{13}, N_{24}, \ldots, N_{n-2,n}$ herleiten.
Letztendlich erreicht man dann $N_{1,n}$ – und das ist genau das gesuchte.

Die Regeln, nach denen diese Berechnung läuft sind die folgeden:

1.) Initialisierung:

Ein Nichtterminalsysmbol liegt genau in der Menge, wenn es eine Regel A → a gibt.
genau dann liegt das a in der Menge, und zwar genau dann: wir haben nämlich keine andere Möglichkeit, dieses Symbol abzuleiten

2.) Hochhangeln:

A ist in N_{ij} mit i kleiner j, (also ij ist Teilwort der Länge größer gleich 2), wenn es ein k gibt (praktisch die Trennung in die beiden Teilworte) so dass i kleiner k kleiner j. Man muss nun alle Aufteilungs-Möglichkeiten berücksichtigen.

ik j
i k j
i k j
i kj

CYK-Algorithmus:
das ganze mit Memoration machen. Es ginge auch rekursiv, aber das wäre wesentlich aufwendiger.
SIEHE BEISPIEL AUF DEM BLATT:
man muss jeweils 1z.B. 1-4 unterteilen in 1-1/2-4, 1-2/3-4, 1-3/4-4.

3.8 EBNF: Erweiterte kontextfreie Grammatiken

Hier dürfen wir auf der rechten Seite auch RegEx und FedEx verwenden.
→ höherer Beschreibungskomfort.

Zu jeder EBNF kann man aber auch eine äquivalente BNF konstruieren.

Def.: G ist eine erweiterte kontextfreie Grammatik (ECFG), wenn der Teil ohne das P eine kontextfreie Grammatik ist (also N, Sigma, LeereMenge, S), und P: N → RegEx(N vereinigt Sigma).
Wir schreiben wie üblich den Pfeil für diese Regeln:
A geht nach Alpha, falls Alpha eben P (A) ist.
Trick: ich ordne einer Regel dieser Grammatik eine unendliche Menge von Regeln zu (einfach alle Wörter nehmen, die auf die rechte Seite passen, wo wir jetzt den STERN haben.)

BEACHTE: P repräsentiert die Regelmenge P-Schlange mit folgender Eigenschaft:
A geht nach Alpha-Schlange (instanziierte Regel), genau dann, wenn das Alpha-Schlange ein Element der Semantik dieses regulären Ausdruckes P(A) ist.

SATZ: Die Klasse der kontextfreien Klassen über Sigma bekomme ich, wenn ich diesem erweiterten Mechanismus arbeite.
BEWEISIDEE:
1.) Inklusion: klar, nach Definition: BNF ist Teilmenge von EBNF.
2.) ahdersherum: durch Elemination regulärer Ausdrücke. (spielt bei XML eine Rolle!)

* Disjunktion: klar, „a oder b“ ist dasselbe wie „a | b“.
* Stern: neues Nichtterminalsymbol mit A → Alpha A |Epsilon.

3.9 Rekursive endliche Automaten (Syntaxdiagramme)

Syntaxdiagramme entsprechen endlichen Automaten, die sich rekursiv aufrufen können.

Def: Ein Automat heißt rekursiver endliche Automat über Sigma, falls das Delta eine Abbilung ist Q x Autonomes Wort (leeres Wort) oder ein Zustand aus Q, das ganze abgebildet in eine Teilmenge von q.
In den Transitionen dürfen also auch Zustandsnamen auftreten.
Sonst wie ein endlicher Automat.
Neue Bezeichnung: RFA (recursive finate automate)

Semantik:
Konfiguration: bestehen als aktuellem Zustand und noch zu verbrauchender Eingabe.
Transitionsrelation: Teilmenge von Q x Simga-Stern-Quadrat
Transitionsmenge ist definiert als kleinste Relation für die gilt:

1.) Wenn p Element Delta von q, a ist, d.h. normale Transition mit einem Terminalsymbol, dann bedeutet das, die Kondiguration qaw kann übergehen in pw.
2.) …

Das kannten wir schon. Jetzt die Erweiterung:

3.) Was passiert, wenn ich ein Delta hab, bei dem ein Zustand vorhanden ist:

SATZ: Alle Sprachen, die wir mit rekursiven Automaten erkennen können, können wir auch mit PDA's erkennen.
Beweis: Simulation eines RFA durch einen PDA:
Idee: Wir benutzen den Keller, um die Rücksprungadressen zu speichern.

Wir müssen ja nur einen endlichen Automaten ohne Stack simulieren. Den unbenutzen Stack benutzen wir einfach für die rekursiven Rücksprungadressen.

Hinweis:
nichtdeterministische Automat = man weiss nicht, in welchen Zustand man gehen muss.