Formale Sprachen und Automaten

Mathematische Grundlagen der Linguistik II, SS 03

Werner Saurer
Computerlinguistik
Uni Saarbrücken

Bemerkung/Würdigung:

Dieses Skript basiert zum grössten Teil auf dem Skript von Espen Vestre, SS 93.

Literatur

Lewis, H. / Papadimitriou, C.:

Elements of the Theory of Computation

Englewood Cliffs: Prentice Hall 1981

Ergänzend:

Partee/ter Meulen/Wall:
Mathematical Methods in Linguistics
Kluwer 1990

Hopcroft/Ullmann:
Introduction to Automata Theory, Languages and Computation
Addison-Wesley 1979

Alles im Handapparat zur Verfügung, H/U auch in deutscher Ausgabe.

Programm

❶ Reguläre Sprachen
Endliche Automaten L & P Kap. 1.8 - 2.

❷ Kontextfreie Sprachen
Kellerautomaten L & P Kap. 3

❸ Typ 0 - Sprachen, Chomsky-Hierarchie
Turingmaschinen L & P Kap. 4 + 5.2

Zeitplan (vorläufig)

Woche	Mo 14-16	Mi 13-14	Mi 16-18	Teil
1.	-	Vo 23. 4.	-	❶
2.	Vo 28.4.	Vo 30.4.	-	❶
3.	Vo 5.5.	Vo 7.5.	Üb1 7.5.	❶
4.	Vo 12.5.	Vo 14. 5.	Üb2 14.5.	❶
5.	Vo 19. 5	Vo 21. 5.	Üb3 21. 5.	❶
6.	Vo 26. 5.	Vo 28. 5.	Üb4 28. 5.	❷
7.	Vo 2.6.	Vo 4.6	Üb5 4.6.	❷
8.	Pfingstmontag	1. Prüfg 11. 6.	-	❷
9.	Vo 16. 6.	Vo 18..6.	Üb6 18. 6.	❷
10.	Vo 23. 6.	Vo 25. 6.	Üb7 25. 6.	❸
11.	Vo 30.6.	Vo 2.7.	Üb8 2.7.	❸
12.	Vo 7.7.	Vo 9.7.	Üb9 9.7.	❸
13.	Vo 14. 7.	2. Prüfung 16. 7.	-	
14.		Nachklausur 23.7.		

Wozu?

— Linguistische Grundlagen:
Z.B. Komplexität der natürlichen Sprachen

— Grundlage der Parsingtheorie und direkte Anwendungen in der Sprachverarbeitung (z.B. endliche Automaten für morphologische Analyse).

— Als mathematische Grundausrüstung um besser (kritisch) mit z.B. Grammatikformalismen umgehen zu können.

Alphabet Σ:

endliche Menge beliebiger Elemente (aber meistens sind es *Buchstaben*), *Symbole* oder *Zeichen* genannt.

Wort *w* über Σ:

Zeichenkette aus in Σ vorkommenden Zeichen.

Wortlänge:

$|w|$ = die Länge der Kette.

+ Wörter können auch als Funktionen w: {1, …, $|w|$} → Σ betrachtet werden.

$w(i)$ ist dann das i'te Zeichen im Wort w.

Das leere Wort

ε, $|\varepsilon| = 0$.

Konkatenation

$w = x \circ y$ gdw

i) $|w| = |x| + |y|$

ii) $w(j) = x(j)$, $j = 1, \ldots, |x|$

iii) $w(|x|+j) = y(j)$, $j = 1, \ldots, |y|$

Beispiele:

$wort \circ länge = wortlänge$

$x \circ \varepsilon = \varepsilon \circ x = x$, für beliebige x.

Definition

$w^0 = \varepsilon$

$w^{i+1} = w^i \circ w$

Beispiel:

$(da)^2 = dada$

Spiegelung

Beispiel: $(espen)^R = nepse$

Definition:

$|w| = 0$: $w^R = w = \varepsilon$

$|w| > 0$: dann $w = ua$, u Wort und a Zeichen, und: $w^R = au^R$.

Sprache

Eine Sprache L ist eine beliebige Menge von Wörtern über einem gegebenen Alphabet ∑.

Kleene'sche Hülle (L^*)

$L^+ = \{w_1 \circ \ldots \circ w_n : n \geq 1, w_i \in L \text{ für } 1 \leq i \leq n\}$

$L^* = L^+ \cup \{\varepsilon\}$

Sprache-Konkatenation

$L_1L_2 = L_1 \circ L_2 = \{x \circ y : x \in L_1, y \in L_2\}$

Mathematische Induktion

Durch mathematische Induktion lassen sich Eigenschaften natürlicher Zahlen beweisen.

Nehmen wir an, wir wollen die Aussage

für alle n: E(n)

beweisen:

Ein Beweis durch Mathematische Induktion besteht aus:

1. Induktionsbasis: Ein Beweis für $E(0)$.
2. Induktionsannahme: Die Annahme, daß für k ≤ n: $E(k)$.
3. Induktionsschritt: Ein Beweis für $E(n+1)$, mit Hilfe der Induktionsannahme.

…dies läßt sich am besten durch ein konkretes Beispiel erklären:

Induktion über Wortlänge (Beispiel):

Wir beweisen folgendes (aus L&P, Seite 30-31):

Für Wörter w und x: $(wx)^R = x^R w^R$

(z.B. $(mark)^R = (rk)^R(ma)^R = kram$)

+ Das wichtigste bei einem Induktionsbeweis ist, die richtige Induktionsannahme zu formulieren!

Hier ist in dem zu beweisenden Satz keine natürliche Zahl vorhanden. Aber der Satz läßt sich durch *Induktion über Wortlänge* beweisen, und zwar über die Wortlänge von x:

Induktionsannahme:

Für Wörter wx mit $|x| \leq n$ gilt: $(wx)^R = x^R w^R$

Induktionsbasis:

$|x| = 0$ heißt daß $x = \varepsilon$:

$(wx)^R = w^R = \varepsilon^R w^R = x^R w^R$

Der Induktionsschritt erfordert – wie immer – etwas mehr Aufwand:

Induktionsschritt:

Wenn $|x| = n+1$, ist $x=ua$ für ein Wort u, mit $|u| = n$, und ein Zeichen a.

$(wx)^R = (w(ua))^R$
$= ((wu)a))^R$ (Assoziativität, siehe Übung)
$= a(wu)^R$ (def. Spiegelung)
$= a(u^R w^R)$ (**NB:** Induktionsannahme!)
$= (ua)^R w^R$ (ass. + def. Spiegel. „rückwärts“)
$= x^R w^R$

+ Im Induktionsschritt haben wir gezeigt: Wenn die Induktionsannahme für $|x| = n$ gilt, *muß* sie auch für $|x| = n+1$ gelten.

Von der Induktionsbasis wissen wir, daß die Induktionsannahme für $|x| = 0$ gilt, also gilt sie auch für $|x| = 1, 2, 3, \ldots$
d.h., sie gilt für *Wörter x von beliebiger Länge*.

Gilt sie auch für Wörter *w* von beliebiger Länge? Nun, der Beweis wurde so konstruiert, daß er für *beliebige w* gilt. Eine *Doppelinduktion* ist also in diesem Falle nicht erforderlich! (Jedoch in manch anderen Fällen!)

Reguläre Ausdrücke

Mit regulären Ausdrücken lassen sich einige unendliche Sprachen endlich beschreiben:

Definition der Menge der regulären Ausdrücke

1) $\varnothing$ ist ein r.A.
2) Jedes Zeichen $\sigma \in \sum$ ist ein r.A.
3) Wenn α und β r.A. sind, so auch $(\alpha\beta)$, $(\alpha \cup \beta)$ und α^*.
4) Die Menge der regulären Ausdrücke ist durch die Formationsregeln 1) - 3) vollständig beschrieben.

Die in 3) vorkommenden Zeichen **(**, **)**, **U** und ***** haben bis jetzt *keine* Bedeutung (deshalb sind sie auch mit Absicht etwas anders als (,), ∪ und * geschrieben). Sie sind reine syntaktische Objekte.

Jetzt zu ihrer Bedeutung:

Reguläre Sprachen (reguläre Mengen)

Die von einem regulären Ausdruck α beschriebene Sprache $L(\alpha)$ ist gegeben durch:

1) $L(\varnothing) = \varnothing$
2) $L(\sigma) = \{\sigma\}$ für jedes $\sigma \in \sum$
3) $L((\alpha\beta)) = L(\alpha)L(\beta)$
   $L((\alpha \cup \beta)) = L(\alpha) \cup L(\beta)$
   $L(\alpha^*) = L(\alpha)^*$

Die Sprachen, die mit einem regulären Ausdruck beschreibbar sind, heißen *reguläre Sprachen* (oder reguläre Mengen).

Zur Verbesserung der Lesbarkeit werden unnötige Paranthesen weggelassen:

Statt $((a \cup b) \cup c)^*(de))$ schreiben wir $(a \cup b \cup c)^*de$.

Beispiele

1) $\sum = \{a, b\}$

   $L((a \cup b)^*aa(a \cup b)^*) =$
   $\{w \in \sum^*$: in w kommen zwei a's nacheinander vor$\}$.

2) $\sum = \{0, 1, \ldots, 9, ,\}$

   $(0 \cup ((1 \cup \ldots \cup 9)(0 \cup \ldots \cup 9)^*)),(0 \cup \ldots \cup 9)(0 \cup \ldots \cup 9)$

   beschreibt Zahlen mit zwei Dezimalstellen.

Vereinfachte Darstellung regulärer Mengen:

Da die beiden Schreibweisen

$L((a \cup b)^*aa(a \cup b)^*)$

und

$(\{a\} \cup \{b\})^*\{a\}\{a\}(\{a\} \cup \{b\})^*$

ziemlich umständlich sind, werden wir (in unserer Metatheorie) folgende abgekürzte Darstellungsweise benutzen:

$(a \cup b)^*aa(a \cup b)^*$

Endliche Automaten

Mit regulären Ausdrücken läßt sich jede reguläre Sprache auf endliche Weise beschreiben.

Aber wie beantworten wir die Frage

Ist $w \in L$?

für L regulär und w ein beliebiges Wort über $\sum$?

Ein

deterministischer endlicher Automat (DEA)

ist ein sehr einfaches Modell eines Computers:

Er liest eine endliche *Eingabe*, befindet sich während des Lesens zu jeder Zeit in einem einer endlichen Zahl *Zustände*, und gibt schließlich „akzeptiert“ oder „nicht akzeptiert“ als *Ausgabe*.

Die Eingabe ist ein *Wort* einer Sprache und wird Zeichen für Zeichen, von links nach rechts, gelesen. Der aktuelle Zustand und das Zeichen bestimmen zusammen den nächsten Zustand.

Komponenten eines deterministischen endlichen Automaten

K – eine endliche Menge von *Zuständen*

$\sum$ – ein *Alphabet*

s – ein *Startzustand* ($s \in K$)

F – eine Menge von *Endzuständen* ($F \subseteq K$)

Was noch fehlt sind die Regeln für die Zustandsänderungen. Diese werden durch eine *Übergangsfunktion* modelliert:

δ – eine Funktion von K und $\sum$ in K.

Das Verhalten des Automaten ist zu jeder Zeit von zwei Faktoren abhängig:

- der aktuelle Zustand
- die noch zu lesenden Zeichen

Der Zustand und die noch zu lesenden Zeichen bilden zusammen eine *Konfiguration*.

Formal ist also jedes Paar (q, w) mit $q \in K, w \in \sum^*$ eine mögliche Konfiguration, und diese Konfiguration bestimmt eindeutig den weiteren Verlauf der Berechnung.

Beispiel

Die Maschine $M_a = (K, \sum, \delta, s, F)$

wo: $K = \{p, q\}$
$\sum = \{a, b\}$
$s = p$
$F = \{p\}$

$\delta(p, a) = p \qquad \delta(q, a) = p$
$\delta(p, b) = q \qquad \delta(q, b) = q$

Die Zustände der Maschine M_a bilden einen *Speicher* für das zuletzt gelesene Zeichen:

In p wurde a zuletzt gelesen (außer am Anfang)

In q wurde b zuletzt gelesen.

Der einzige Endzustand ist der Startzustand, also der Zustand in dem entweder noch nichts gelesen wurde, oder zuletzt ein a gelesen wurde.

Eingabebeispiel:

Die Startkonfiguration ist immer (s, w). (w Eingabe)

Bei Eingabe *bba* ist die Startkonfiguration also bei unserem Beispielautomaten M_a

(p, bba)

$\delta(p, b) = q$, also ist die nächste Konfiguration (q, ba), und wir schreiben:

$(p, bba) \vdash_{M_a} (q, ba)$

und sagen (p, bba) *ergibt* (q, ba) *in einen Schritt*.

Es folgen zwei weitere Berechnungsschritte:

$(q, ba) \vdash_{M_a} (q, a) \vdash_{M_a} (p, \varepsilon)$

Jetzt hält der Automat, und die *Ausgabe* wird folgendermaßen abgelesen:

Weil $p \in F$, ist die Eingabe „akzeptiert".

Die Relation $\vdash_M^*$

Die „ergibt"-Relation $\vdash_M$ wurde im Beispiel eingeführt. Formal sieht die Definition so aus:

$(q, w) \vdash_M (q', w')$

gdw

$w = \sigma w'$ für ein $\sigma \in \Sigma$ und $\delta(q, \sigma) = q'$

$\vdash_M^*$ (lese: *ergibt*) ist die *reflexiv-transitive Hülle* von $\vdash_M$, oder zu Deutsch: Wenn $(q, w) \vdash_M^* (q', w')$, *ergibt* die erste Konfiguration die zweite in null oder mehrere Schritte.

Akzeptanz formal definiert

$L(M)$, *die von M akzeptierte Sprache,* definieren wir so:

$w \in L(M)$

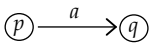
gdw

Es gibt ein $q \in F$ so daß $(s, w) \vdash_M^* (q, \varepsilon)$

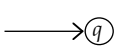
Zu Deutsch: $L(M)$ sind die Wörter, die *akzeptiert* werden, d.h., die als Eingabe den Automaten zum Halten in einem Endzustand bringen.

Zustandsdiagramme

Zur einfachen Darstellung endlicher Automaten werden *Zustandsdiagramme* benutzt. In den Zustandsdiagrammen werden Zustände durch kleine Kreise symbolisiert. δ, s und F werden folgendermaßen dargestellt:



für $\delta(p, a) = q$

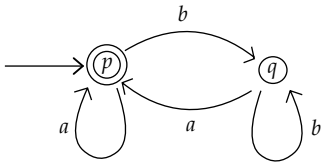


für $s = q$

für $q \in F$

Beispiel

Als Beispiel betrachten wir wieder M_a :



Oft werden die Zustandsnamen einfach weggelassen, weil sie bei einem mit einem Zustandsdiagramm dargestellten Automaten keine wichtige Rolle spielen.

Toter Zustand

Ein toter Zustand ist ein nicht-Endzustand aus dem sich der Automat nicht mehr bewegen kann. Der Automat ist „gefangen".

Formal:

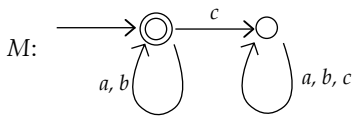
p ist tot

gdw

$p \notin F \wedge \forall \sigma \in \sum(\delta(p, \sigma) = p)$

Beispiel

$\sum = \{a, b, c\}$



$L(M) = \{w \in \Sigma^*: c \text{ kommt in } w \text{ nicht vor}\}$

Nicht-deterministische endliche Automaten (NEA)

Nicht-deterministische endliche Automaten können in jedem Berechnungsschritt in einen von *null oder mehreren* Zuständen übergehen. Sie sind nicht deterministisch, weil man von der aktuellen Konfiguration nicht (immer) voraussagen kann, in welcher Konfiguration sich der Automat nach dem nächsten Berechnungsschritt befindet.

Ein Wort wird von einem nicht-deterministischen endlichen Automaten *akzeptiert*, wenn von allen möglichen Berechnungen, mindestens eine zur Akzeptanz (im normalen Sinne) führt.

Wir können uns vorstellen, daß sich der Automat *selbst kopiert*, und daß jeder einzelne Automat die verschiedenen möglichen nächsten Zustände ausprobiert. Die Eingabe ist akzeptiert, wenn sich schließlich mindestens einer von diesen vielen Automaten in einem Endzustand mit leerer Eingabe befindet.

Bemerke, daß es bei NEAs auch "Sackgassen"-Konfiguration gibt, die keine neuen Konfiguratio-nen ergeben. Bei DEAs dagegen, ist die Funktion δ *total*.

Übergänge in denen mehrere Zeichen gelesen werden

Dies ist eine weitere Verallgemeinerung, die wir gleichzeitig mit dem Nichtdeterminismus einführen. Wir lassen zu, daß *mehrere* oder auch keine Zeichen in einem Schritt gelesen werden können.

Insgesamt haben wir:

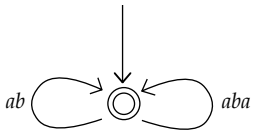
> Ein nicht-deterministischer endlicher Automat hat statt einer Übergangsfunktion eine *Übergangsrelation* Δ, die auf $K \times \Sigma^* \times K$ definiert ist.

Die sonstigen Komponenten sind wie bei einem DEA.

Bemerke: Als Sonderfall kann eine Relation auf $K \times \Sigma^* \times K$ auch eine *totale Funktion* $K \times \Sigma^* \to K$ sein. Also ist jeder DEA auch ein NEA.

Beispiel

Ein Automat, der die Sprache $(ab \cup aba)^*$ akzeptiert:



Die „ergibt"-Relation für NEA

Konfigurationen sind wie bei einem DEA.

> $(q, w) \vdash_M (q', w')$
>
> gdw
>
> $w = uw'$ für ein $u \in \Sigma^*$ und $\Delta(q, u, q')$
>
> $\vdash_M^*$ (lese: *ergibt*) ist wieder die *reflexiv-transitive Hülle* von $\vdash_M$

Bemerke: Die Definition von $\vdash_M$ *und* $\vdash_M^*$ für DEA ist ein Sonderfall der obigen Definition!

Die von einem NEA akzeptierte Sprache

Unsere Definition von $L(M)$ für einen DEA M ist auf einen NEA M direkt übertragbar:

> $w \in L(M)$
>
> gdw
>
> Es gibt ein $q \in F$ so daß $(s, w) \vdash_M^* (q, \varepsilon)$

Lemma

Sei $M = (K, \Sigma, \Delta, s, F)$ ein NEA, q, r Zustände, und x, y Wörter über Σ.

Dann gilt $(q, xy) \vdash_M^* (r, \varepsilon)$ wenn es einen Zustand p gibt, so daß $(q, x) \vdash_M^* (p, \varepsilon)$ und $(p, y) \vdash_M^* (r, \varepsilon)$

Beweis

$(q, x) \vdash_M^* (p, \varepsilon)$ bedeutet, daß es ein $n \geq 0$, Zustände $q_0, \ldots, q_n$ und Wörter $x_0, \ldots, x_n$ gibt, so daß:

$(q, x) = (q_0, x_0) \vdash_M (q_1, x_1) \vdash_M \ldots \vdash_M (q_n, x_n) = (p, \varepsilon)$

$(q_i, x_i) \vdash_M (q_{i+1}, x_{i+1})$ bedeutet (nach der Definition) daß es ein Wort u_i gibt, so daß $x_i = u_i x_{i+1}$ und $(q_i, u_i, q_{i+1}) \in \Delta$. Dann ist aber auch $x_i y = u_i x_{i+1} y$, und nach der Definition von $\vdash_M$ auch $(q_i, x_i y) \vdash_M (q_{i+1}, x_{i+1} y)$.

Deshalb gilt auch $(q, xy) \vdash_M^* (p, y)$.

Wenn dann auch $(p, y) \vdash_M^* (r, \varepsilon)$, gilt (weil $\vdash_M^*$ transitiv ist), wie gezeigt werden sollte, daß $(q, xy) \vdash_M^* (r, \varepsilon)$.❑

Äquivalenz von deterministischen und nicht-deterministischen endlichen Automaten.

Wie schon bemerkt, ist ein DEA auch immer ein NEA.

Wir werden gleich feststellen, daß es zu jedem NEA auch einen äquivalenten DEA gibt.

Aber was heißt „äquivalent“?

Definition

M und M' sind äquivalent gdw $L(M) = L(M')$

Theorem

Zu jedem nicht-deterministischen endlichen Automaten gibt es einen äquivalenten deterministischen endlichen Automaten.

Beweis des Theoremes

Wir fangen mit dem einfachsten Teil an:

Lemma 1: Sei $M = (K, \sum, \Delta, s, F)$ ein NEA. Dann gibt es einen äquivalenten NEA M', der nur solche Übergänge (p, u, q) hat, in denen $|u| \leq 1$.

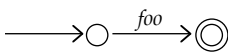
Beweis: Wir können von M einen erweiterten Automaten $M' = (K', \sum, \Delta', s, F)$ folgendermaßen konstruieren:

Jeder Übergang $(p) \xrightarrow{a_1 \ldots a_n} (q)$ wird durch Übergänge

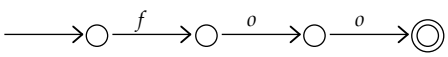
$(p) \xrightarrow{a_1} (p_1) \xrightarrow{a_2} \cdots \xrightarrow{a_{n-1}} (p_{n-1}) \xrightarrow{a_n} (q)$

ersetzt, indem wir n-1 neue Zustände einführen, und die Δ-Relation entsprechend erweitern.p

Beispiel: Von



bilden wir



Jetzt werden wir zu M' einen äquivalenten deterministischen Automaten

$$M'' = (K'', \sum, \delta'', s'', F'')$$

konstruieren.

Wir werden

$$K'' = 2^{K'}$$

(die Menge der Teilmengen von K') setzen. Die Idee ist, daß wenn M'' sich in einem Zustand $\{q_1, \ldots, q_n\}$ befindet, M' sich bei gleich viel gelesener Eingabe *in einem der Zustände* $q_1, \ldots, q_n$ *befinden könnte*.

Weiter werden wir

$$F'' = \{X \subseteq K' : X \cap F \neq \varnothing\}$$

setzen.

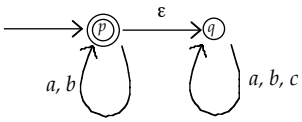
(Diejenigen, die Mengen als Zustände nicht „akzeptieren“ wollen, können einfach für jede Teilmenge von K' einen *neuen* Zustand einführen)

Bevor wir schließlich s'' und δ'' definieren, bereiten uns die ε-Übergänge die noch in M' vorhanden sind, einige Schwierigkeiten:

Definition Für beliebige q:

$$E(q) = \{p \in K: (q, \varepsilon) \vdash^*_{M'} (p, \varepsilon)\}$$

Beispiel: In diesem Automaten ist $E(p) = K$, $E(q) = \{q\}$:



Jetzt setzen wir

$$s'' = E(s')$$

(zu Deutsch: Der neue Startzustand ist die Menge der Zustände, die vom Ursprünglichen Startzustand ohne das Lesen der Eingabe erreichbar sind)

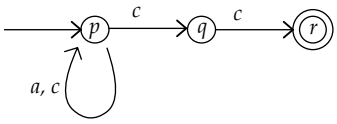
und für jede Teilmenge Q von K' (d.h.: Element von K''!) und Symbol σ:

$$\delta''(Q, \sigma) = \cup\{E(p) : p \in K' \text{ und für ein } q \in Q: \Delta'(q, \sigma, p)\}$$

(der ursprüngliche NEA M' kann, wenn in q das Zeichen σ gelesen wird, in jeden solchen Zustand p übergehen, und kann auch ohne noch ein Zeichen zu lesen in einen der Zustände $E(p)$ übergehen)

Beispiel

Wir betrachten den folgenden einfachen Automaten M über dem Alphabet $\{a, c\}$:

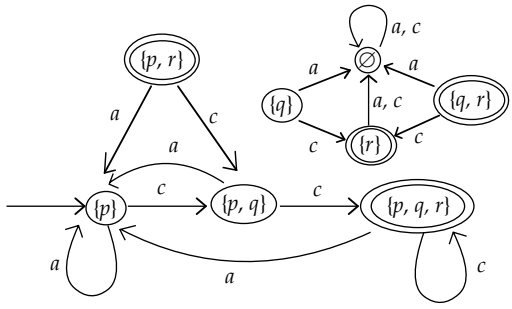


($L(M)$ sind diejenigen Wörter, die mit „cc“ enden)

Hier ist $M' = M$, und

$E(p) = \{p\}$, $E(q) = \{q\}$ und $E(r) = \{r\}$,
$s'' = E(s) = \{p\}$
und $K'' = \{\varnothing, \{p\}, \{q\}, \{r\}, \{p, q\}, \{p, r\}, \{q, r\}, \{p, q, r\}\}$.

M'' sieht dann folgendermaßen aus:



Die oberen fünf Zustände haben hier keine Funktion!

Lemma 2: M'' ist deterministisch.

Beweis: Folgt direkt, da δ'' eine wohl definierte totale Funktion ist.p

Jetzt sind wir bereit, das Theorem zu beweisen.

Wir werden folgendes durch Induktion über Wortlänge beweisen:

Lemma 3:

$$(q, w) \vdash^*_{M'} (p, \varepsilon)$$
gdw
Es gibt ein P, mit $p \in P$, so daß $(E(q), w) \vdash^*_{M''} (P, \varepsilon)$

Bevor wir Lemma 3 beweisen, stellen wir schon fest, das dies mehr als ausreichend ist, um das Theorem zu beweisen:

$w \in L(M')$

gdw $(s, w) \vdash^*_{M'} (f, \varepsilon)$ für ein $f \in F$ [Def. von $L(M')$]

gdw $(E(s), w) \vdash^*_{M''} (Q, \varepsilon)$ für ein Q mit $f \in Q$ [Lemma 3]

gdw $(s'', w) \vdash^*_{M''} (Q, \varepsilon)$ für ein $Q \in F''$ [Def. von M'']

gdw $w \in L(M'')$ [Def. von $L(M'')$]

Beweis von Lemma 3 durch Induktion über Wortlänge

Induktionsannahme

Die Induktionsannahme ist:

Lemma 3 gilt für $|w| \leq n$.

NB: Statt Äquivalenz von M' und M'' direkt zu beweisen, haben wir eine etwas stärkere Induktionsannahme formuliert!

Induktionsbasis

$|w| = 0$, d.h. $w = \varepsilon$:

Wir setzen ε für w in Lemma 3 ein:

$(q, \varepsilon) \vdash^*_{M'} (p, \varepsilon)$

gdw

Es gibt ein P, mit $p \in P$, so daß $(E(q), \varepsilon) \vdash^*_{M''} (P, \varepsilon)$.

Die erste Aussage gilt gdw $p \in E(q)$.

M'' ist deterministisch, d.h. die zweite Aussage gilt gdw $P = E(q) \wedge p \in P$, d.h. gdw $p \in E(q)$.

Induktionsschritt

$|w| = n+1$, d.h. $w = va$, wo a ein Symbol ist und v ein Wort mit $|v| = n$:

⇒: Wenn $(q, w) \vdash^*_{M'} (p, \varepsilon)$ gibt es Zustände r und t, wo

(*) $(q, va) \vdash^*_{M'} (r, a) \vdash_{M'} (t, \varepsilon) \vdash^*_{M'} (p, \varepsilon)$.

Mit anderen Worten: Der Automat liest erst v, liest dann in einem Schritt a, und macht schließlich (vielleicht) einige ε-Schritte.

Es gilt dann:

$(q, v) \vdash^*_{M'} (r, \varepsilon)$

Weil $|v| \leq n$, können wir hier die Induktionsannahme anwenden, und kriegen:

Es gibt einen R, mit $r \in R$, so daß $(E(q), v) \vdash^*_{M''} (R, \varepsilon)$.

(*) gibt uns auch, daß $\Delta'(r, a, t)$ gilt, und daß $p \in E(t)$. Von der Konstruktion von δ'' wissen wir, daß $p \in \delta''(R, a)$, und deshalb daß

$(R, a) \vdash_{M''} (P, \varepsilon)$ für einen P, mit $p \in P$, und dann auch $(E(q), va) \vdash^*_{M''} (P, \varepsilon)$, wie wir zeigen wollten.

⇐: Wenn $(E(q), va) \vdash^*_{M''} (P, \varepsilon)$, für ein P mit $p \in P$, gibt es einen Zustand R, mit $\delta''(R, a) = P$, so daß

(**) $(E(q), va) \vdash^*_{M''} (R, a)$.

Nach der Konstruktion von δ'' bedeutet $\delta''(R, a) = P$, daß es einen Zustand t von M' gibt, mit $p \in E(t)$, und wo für irgendeinen $r \in R$: $\Delta'(r, a, t)$.

Wenn (**) gilt, gilt auch

$(E(q), v) \vdash^*_{M''} (R, \varepsilon)$

und wenden wir hier die Induktionsannahme an ($|v| \leq n$!), bekommen wir:

$(q, v) \vdash^*_{M'} (r, \varepsilon)$ [$r \in R$!]

und deshalb auch

$(q, va) \vdash^*_{M'} (t, \varepsilon)$ [$\Delta'(r, a, t)$!]

und schließlich

$(q, va) \vdash^*_{M'} (p, \varepsilon)$ [$p \in E(t)$!]

wie wir zeigen wollten.

Damit ist Lemma 3, und dadurch das Theorem, bewiesen.❑

Endliche Automaten und reguläre Sprachen

Theorem

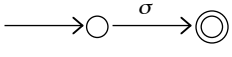
Für jede reguläre Sprache L gibt es einen endlichen Automaten M so daß $L = L(M)$.

Beweis

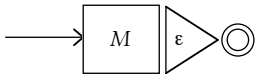
Wenn L regulär ist, gibt es einen regulären Ausdruck α mit $L(\alpha) = L$. Wir konstruieren einen NEA M durch Induktion über α, d.h. wir werden für jedes α einen Teilautomaten $M(\alpha)$ konstruieren, wo $L(M(\alpha)) = L(\alpha)$. Jeder Automat in der Konstruktion benutzt das Alphabet Σ der Sprache L (die übrigen formalen Details der Konstruktion können von den Zeichnungen leicht abgeleitet werden):

1) $\alpha = \varnothing$: Dann sieht $M(\alpha)$ so aus:

2) $\alpha = \sigma \in \sum$: Dann sieht $M(\alpha)$ so aus:

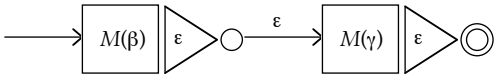


Wir werden jetzt Automaten schematisch zusammenbauen, indem

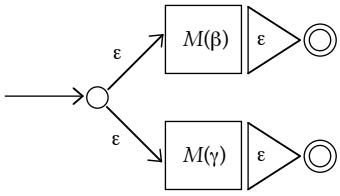


den Automaten darstellt, der wie M ist, außer daß die Endzustände durch ε-Übergänge mit einem einzelnen neuen Endzustand ersetzt sind.

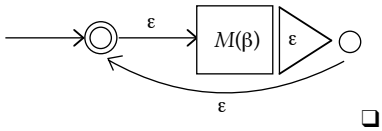
3) $\alpha = (\beta\gamma)$: Dann sieht $M(\alpha)$ so aus:



4) $\alpha = (\beta \cup \gamma)$: Dann sieht $M(\alpha)$ so aus:



5) $\alpha = \beta^*$: Dann sieht $M(\alpha)$ so aus:

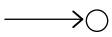


❑

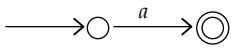
Beispiel

Ein Automat der $L((a^* \cup \varnothing))$ akzeptiert:

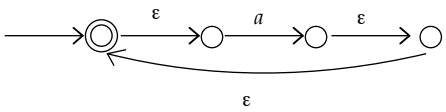
Der Automat für $\varnothing$:



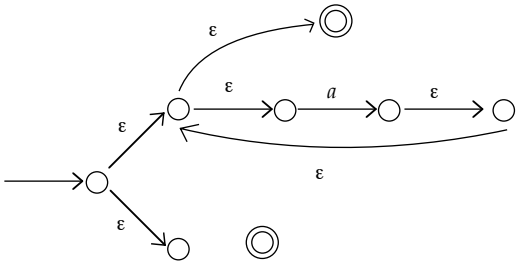
– für a:



– für a^*:



– und schließlich für $(a^* \cup \varnothing)$:



Theorem

Jede Sprache die von einem endlichen Automaten akzeptiert wird, ist regulär.

Beweis

Sei M ein endlicher Automat. Weil wir schon bewiesen haben, daß es zu jedem NEA einen äquivalenten DEA gibt, können wir annehmen, daß M deterministisch ist. Durch Konstruktion werden wir Zeigen, daß es eine reguläre Sprache R gibt, wo $R = L(M)$.

Erstens müssen die Zustände von M numeriert werden:

$K = \{q_1, \ldots, q_n\}$ (wo $q_1 = s$)

Wir werden R aus einer Menge von einfacheren Sprachen definieren: Wir definieren $R(i, j, k)$ als die Wörter, die gelesen werden, wenn M sich vom Zustand q_i bis Zustand q_j bewegt, ohne sich zwischen q_i und q_j durch einen Zustand q_m, wo $m \geq k$, zu bewegen (i und j dürfen aber $\geq k$ sein!).

Dies hat einen Sinn, weil

$L(M) = \cup\{R(1, j, n+1) : q_j \in F\}$

(Gute Frage: Wozu sind jetzt i und k in $R(i, j, k)$ da? Die Antwort lautet (Überraschung??): Um eine stärkere Induktionsannahme machen zu können!!)

Formal definieren wir:

Für $1 \leq i \leq n$, $1 \leq j \leq n$, $1 \leq k \leq n+1$:

$R(i, j, k) = \{w \in \sum^* :$ $(q_i, w) \vdash_M^* (q_j, \varepsilon)$ und
für jedes $x \in \sum^*$ und jedes m:
Wenn $(q_i, w) \vdash_M^* (q_m, x)$, dann ist
entweder $m<k$
oder $x=\varepsilon$ und $m=j$
oder $x=w$ und $m=i\}$

Wir werden jetzt beweisen, daß jedes $R(i, j, k)$ regulär ist. Es folgt dann, daß R – als Vereinigung endlich vieler $R(i, j, k)$ – auch regulär ist. Der Beweis wird durch Induktion über k durchgeführt:

Induktionsannahme

Für $m \leq k$ ist $R(i, j, m)$ regulär.

Induktionsbasis

Wir zeigen, daß $R(i, j, 1)$ regulär ist.

Wenn $i=j$ ist $R(i, j, 1) = \{\varepsilon\} \cup \{\sigma \in \sum : \delta(q_i, \sigma) = q_j\}$.

Wenn $i \neq j$ ist $R(i, j, 1) = \{\sigma \in \sum : \delta(q_i, \sigma) = q_j\}$

In beiden Fällen ist $R(i, j, 1)$ endlich, und damit regulär.

Induktionsschritt

Wir stellen fest, daß

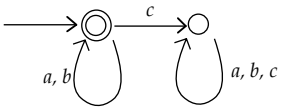
$R(i, j, k+1) = R(i, j, k) \cup R(i, k, k)R(k, k, k)^*R(k, j, k)$

Zu Deutsch: Wenn der Automat sich von q_i nach q_j bewegt, und sich dazwischen nie durch einen Zustand q_m, wo $m>k$, bewegt, bewegt er sich entweder auch nicht durch q_k, oder er bewegt sich von q_i nach q_k ohne durch q_k zu passieren, bewegt sich dann beliebig viel Mal von q_k nach q_k, und bewegt sich schließlich nach q_j, ohne wieder durch q_k zu kommen.

Die Induktionsannahme sagt, daß jede der Sprachen auf der rechten Seite regulär ist. Und die regulären Sprachen sind unter den Operationen von Union, Konkatenation und Kleene'scher Hüllenbildung abgeschlossen. ❑

Beispiel

Wir betrachten wieder den deterministischen Automaten



der als Alphabet {*a*, *b*, *c*} hat, und der die Sprache {*a*, *b*}* akzeptiert.

Numeriere die Zustände von links nach rechts. Es gilt dann:

$L(M) = R(1, 1, 3)$.

$R(1, 1, 3) = R(1, 1, 2) \cup R(1, 2, 2)R(2, 2, 2)^*R(2, 1, 2)$

$R(1, 1, 2) = R(1, 1, 1) \cup R(1, 1, 1)R(1, 1, 1)^*R(1, 1, 1)$
$R(2, 1, 2) = R(2, 1, 1) \cup R(2, 1, 1)R(1, 1, 1)^*R(1, 1, 1)$

$R(1, 1, 1) = a \cup b \cup \varepsilon$
$R(2, 1, 1) = \varnothing$

und dann:

$R(1, 1, 2) = a \cup b \cup \varepsilon \cup (a \cup b \cup \varepsilon)(a \cup b \cup \varepsilon)^*(a \cup b \cup \varepsilon) = (a \cup b \cup \varepsilon)(a \cup b \cup \varepsilon)^* = (a \cup b)^*$

$R(2, 1, 2) = \varnothing \cup \varnothing R(1, 1, 1)^*R(1, 1, 1) = \varnothing$

Mehr brauchen wir dann gar nicht zu berechnen:

$R(1, 1, 3) = (a \cup b)^* \cup R(1, 2, 2)R(2, 2, 2)^*\varnothing = (a \cup b)^*$

Abschlußeigenschaften der Klasse der regulären Sprachen

Theorem

Die regulären Sprachen sind abgeschlossen unter

a) Vereinigung
b) Konkatenation
c) Kleene'scher Hüllenbildung
d) Komplementbildung
e) Schnittbildung

Beweis

a) - c) wissen wir schon.

d) Sei L eine reguläre Sprache. Dann gibt es einen DEA $M = (K, \sum, \delta, s, F)$ der L akzeptiert. Die Komplementsprache $\sum^* - L$ wird dann von

$\overline{M} = (K, \sum, \delta, s, K–F)$

akzeptiert.

e) Seien L_1 und L_2 reguläre Sprachen. Es gilt:

$L_1 \cap L_2 = \sum^* - ((\sum^* - L_1) \cup (\sum^* - L_2))$

(L_1 und L_2 können eventuell verschiedene Alphabete $\sum_1$ und $\sum_2$ haben, bilde dann $\sum = \sum_1 \cup \sum_2$) ❑

Entscheidbare Probleme bei endlichen Automaten (regulären Sprachen)

Theorem

Es gibt Entscheidungsalgorithmen für die folgenden Probleme:

(M, M_1, M_2 sind endliche Automaten, $w \in \sum^*$)

a) Ist $w \in L(M)$?
b) Ist $L(M) = \varnothing$?
c) Ist $L(M) = \sum^*$?
d) Ist $L(M_1) \subseteq L(M_2)$?
e) Ist $L(M_1) = L(M_2)$?

Beweis

Wir dürfen annehmen, daß die Automaten deterministisch sind.

a) Einfach dem Automaten die Eingabe w geben. Nach $|w|$ Berechnungsschritten erfolgt die Antwort.
b) Äquivalent zu der Frage: Ist mindestens ein Endzustand vom Startzustand erreichbar? Diese Frage läßt sich mit Hilfe des (endlichen!) Zustandsdiagrammes beantworten.
c) Äquivalent zu der Frage: Ist $\sum^* - L(M) = \varnothing$?
d) Äquivalent zu: Ist $(\sum^* - L(M_2)) \cap L(M_1) = \varnothing$?
e) Äquivalent zu: Ist $L(M_1) \subseteq L(M_2) \wedge L(M_2) \subseteq L(M_1)$? ❑

Das Pumping-Lemma

Mit den Abschlußeigenschaften läßt sich positiv feststellen, daß eine Sprache regulär ist. Z.B. ist die Sprache {$w \in \{a, b\}^*$: die Länge von w ist nicht teilbar durch 3, die Anzahl der a's dagegen ist teilbar durch 3} regulär.

Mit dem *Pumping-Lemma* läßt sich auch zeigen, daß Sprachen *nicht* regulär sind:

Theorem (Pumping-Lemma)

Sei L eine unendliche reguläre Sprache.
Dann gibt es Wörter x, y und z, wo $y \neq \varepsilon$ und für jedes $n \geq 0$: $xy^nz \in L$.

Vor dem Beweis erst ein Beispiel der Anwendung:

Theorem

$\{a^nb^n : n \geq 0\}$ ist nicht regulär.

Beweis: Wenn L regulär wäre, gäbe es nach dem Pumping-Lemma x, y und z, wo $y \neq \varepsilon$ und für jedes $n \geq 0$: $xy^nz \in L$. Wir können 3 Fälle unterscheiden:

1) $y = a^m$ für $m > 0$. In xyz sind dann die ganzen b's innerhalb z, und xy^2z hat dann m mehr a's als b's. Kontradiktion!

2) $y = b^m$ für $m > 0$ ist dann auch unmöglich.

3) y hat a's und b's. Dann kommen in y^2 b's vor a's vor – Kontradiktion! p
Wir werden die vorgestellte Version des Pumping-Lemmas nicht direkt beweisen, sondern eine stärkere Version davon, die in Beweisen leichter einsetzbar ist:

Theorem

Sei $M = (K, \sum, \delta, s, F)$ ein DEA, und sei w ein *beliebiges* Wort in $L(M)$ mit $|w| \geq |K|$.

Dann gibt es Wörter x, y, z so daß $w = xyz$, $|xy| \leq |K|$, und für jedes $n \geq 0$: $xy^nz \in L(M)$.

Beweis

Sei $l = |w|$. Dann ist $w = \sigma_1 \ldots \sigma_l$, wo $\sigma_1, \ldots, \sigma_l \in \Sigma$. Weil M deterministisch ist, liest M w in l Schritten, d.h., es gibt Zustände $q_0, \ldots, q_l$, wo $q_0 = s$, $q_l \in F$, so daß:

$$(q_0, \sigma_1 \ldots \sigma_l) \vdash_M \ldots \vdash_M (q_{l-1}, \sigma_l) \vdash_M (q_l, \varepsilon)$$

Nun ist $l = |w| \geq |K| = k$. Also müssen mindestens zwei der k+1 Zustände $q_0, \ldots, q_k$ gleich sein! Es gibt also i, j, wo $0 \leq i < j \leq k$, so daß $q_i = q_j$. Dies heißt, daß der Automat während er das Teilwort $\sigma_{i+1} \ldots \sigma_j$ liest, sich von q_i und zurück nach q_i bewegt. Dies heißt wiederum, daß das Wort

$$\sigma_1 \ldots \sigma_i (\sigma_{i+1} \ldots \sigma_j)^n \sigma_{j+1} \ldots \sigma_l$$

für alle n akzeptiert wird.

Jetzt setzen wir $x = \sigma_1 \ldots \sigma_i$, $y = \sigma_{i+1} \ldots \sigma_j$ und $z = \sigma_{j+1} \ldots \sigma_l$. Dann ist auch $|xy| = j \leq k$. ❑

Beispiel

Ein *Palindrom* ist ein Wort (in diesem Zusammenhang nicht unbedingt ein echtes Wort!), das mit seiner eigenen Spiegelung identisch ist. Beispiele: *abba*, *renner*, *saippuakauppias* (finnisch für Seifenverkäufer!), *qcßcq*.

Formal: $PAL = \{w \in \{a, \ldots, z, ä, ö, ü, ß\}^* \mid w^R = w\}$.

PAL keine reguläre Sprache, und dies läßt sich mit der stärkeren Version des Pumping-Lemmas zeigen:

Angenommen, *PAL* ist regulär. Sei M ein *DEA*, der *PAL* akzeptiert, und k die Zahl der Zustände in M.

$w = a^kba^k \in PAL$. Das Theorem sagt uns, daß es Wörter x, y und z gibt, wo $w = xyz$, $y \neq \varepsilon$ und $|xy| \leq k$, und wo $xy^nz \in PAL$ für jede $n \geq 0$.

Dann ist aber $y = a^m$, wo $1 \leq m \leq k$, und wir stellen fest, daß xy^2z zu viele a's vor dem b hat – Kontradiktion!

Deterministischer endlicher Transducer (DET)

Ein *deterministischer endlicher Transducer* ist wie ein DEA, hat aber keine Endzustände[1], dafür hat er eine *Ausgabe*.

Formal: $M = (K, \Sigma, \delta, s)$,

wo δ eine Funktion von $K \times \Sigma$ in $K \times \Sigma^*$ ist. (Der DET schreibt also in jedem Schritt 0, 1 oder mehrere Zeichen.)

Konfigurationen sind bei einem DET geordnete*Tripel*: (p, u, v) bedeutet, daß der Transducer sich im Zustand p befindet, Eingabe u noch zu lesen hat, und Ausgabe v schon geschrieben hat.

Die *ergibt*-Relation

"ergibt in einem Schritt":

$(q, u, v) \vdash_M (q', u', v')$

gdw

$\exists \sigma \in \Sigma \; \exists w \in \Sigma^* (u = \sigma u' \wedge \delta(q, \sigma) = (q', w) \wedge v' = vw)$

$\vdash_M^*$ (lese: *ergibt*) ist wieder die *reflexiv-transitive Hülle* von $\vdash_M$

Die Ausgabe eines DET

Da ein DET (nach unserer Definition) keine Endzustände hat, gibt es keinen Akzeptanz-Begriff. Dafür haben wir folgende Definition:

M erzeugt Ausgabe u bei Eingabe w

gdw

Es gibt ein $q \in K$ so daß $(s, w, \varepsilon) \vdash_M^* (q, \varepsilon, u)$

Sei f eine Zeichenketten-Funktion, $f: \Sigma^* \rightarrow \Sigma^*$:

M berechnet f

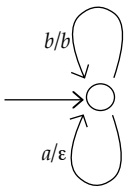
gdw

Für jedes $w \in \Sigma^*$ gibt es $q \in K$ so daß $(s, w, \varepsilon) \vdash_M^* (q, \varepsilon, f(w))$

Zustandsdiagramme

Nur die Übergänge werden in den Zustandsdiagrammen eines DET anders gezeichnet als die eines DEA. "σ/*u*" bedeutet, daß σ gelesen und *u* geschrieben wird.

Beispiele

1. 

– entfernt jedes *a* in der Eingabe.

2. Sei M ein DEA. Wir können M mit einem DET folgendermaßen simulieren:

   Wir erweitern das Alphabet um drei neue Symbole: \$, Y ("yes") und N ("no"). Einen M-Übergang in dem σ gelesen wird, ersetzen wir durch einen Übergang, in dem σ gelesen und ε geschrieben wird. Zusätzlich führen wir für jeden nicht-Endzustand einen Übergang \$/N ein, und für jeden Endzustand einen Übergang \$/Y.

   M mit Eingabe w wird vom DET mit Eingabe w\$ folgendermaßen simuliert: Würde M w akzeptieren, erzeugt der DET die Ausgabe Y, sonst erzeugt er die Ausgabe N.

[1] Oft werden auch DETs *mit* Endzuständen definiert

Kontextfreie Grammatiken

Kontextfreie Grammatiken sind eine Erfindung Chomskys, und wurden für die Beschreibung natürlicher Sprachen konzipiert.

Eine kontextfreie Grammatik hat eine Menge *Umschreibungsregeln* (rewriting rules) und ein *Startsymbol*. Die Zeichenketten der beschriebe-nen Sprache werden generiert, indem man mit dem Startsymbol anfängt, und beliebige Umschreibungsregeln auf dieses Symbol anwendet, bis eine Kette aus Zeichen, die nicht weiter umschreibbar sind, entsteht.

Beispiel:

$S \rightarrow NP\ VP$
$NP \rightarrow D\ N$
$D \rightarrow$ *'ein'*
$N \rightarrow$ *'Mann'*
$VP \rightarrow$ *'lacht'*

(Übrigens: Ist diese Sprache regulär?)

Mathematische Definition von kontextfreien Grammatiken

Eine kontextfreie Grammatik besteht aus

V	- dem Alphabet
$\Sigma \subseteq V$	- den *Terminalsymbolen*
$S \in V - \Sigma$	- dem *Startsymbol* (oder *Startkategorie*)

und schließlich R, *den Regeln* (oder *Produktionen*), die wir als eine Relation definieren:

R - eine endliche Teilmenge von $(V - \Sigma) \times V^*$

$V - \Sigma$ nennt man die Menge der *Nichtterminalen* (oder *Variablen*, *Kategorien*, *Metasymbole*).

Wenn $(A, u) \in R$, schreiben wir $A \underset{G}{\rightarrow} u$, oder einfach $A \rightarrow u$.

Notation

Große Buchstaben für die Nichtterminalen, kleine Buchstaben für die Terminalsymbole.

Ableitungen

Für $u, v \in V^*$ definieren wir:

v ist *direkt ableitbar* von u, mit Symbolen $u \underset{G}{\Rightarrow} v$,

gdw

Es gibt $x, y, v' \in V^*$ und $A \in V - \Sigma$, wo $u = xAy$, $v = xv'y$ und $A \underset{G}{\rightarrow} v'$.

Eine *Ableitung* ist eine Kette von direkten Ableitungen:

$w_0 \underset{G}{\Rightarrow} w_1 \underset{G}{\Rightarrow} \dots \underset{G}{\Rightarrow} w_n$

Wir schreiben dann $w_0 \overset{n}{\underset{G}{\Rightarrow}} w_n$ (w_n *ist ableitbar von* w_0 *in n Schritten*).

Wenn es ein $n \geq 0$ gibt, so daß $u \overset{n}{\underset{G}{\Rightarrow}} v$, heißt v *ableitbar* aus u, und wir schreiben $u \overset{*}{\underset{G}{\Rightarrow}} v$.

($\overset{*}{\underset{G}{\Rightarrow}}$ ist dann die reflexiv-transitive Hülle von $\underset{G}{\Rightarrow}$)

Die kontextfreien Sprachen

Die Sprache $L(G)$, die eine kontextfreie Grammatik G generiert, besteht aus den Zeichenketten über Σ, die vom Startsymbol ableitbar sind:

Formal: $L(G) = \{w \in \Sigma^* : S \overset{*}{\underset{G}{\Rightarrow}} w\}$

Wenn $L = L(G)$ für eine kontextfreie Grammatik G, heißt L *kontextfrei*.

Beispiel: Eine kontextfreie, nicht reguläre Sprache

$G = (V, \Sigma, R, S)$, wo

$V = \{S, a, b\}$,
$\Sigma = \{a, b\}$,
$R = \{S \rightarrow aSb, S \rightarrow \varepsilon\}$

$L(G) = \{a^n b^n : n \geq 0\}$

Reguläre Grammatiken

Sei G eine kontextfreie Grammatik.

G ist *regulär*

gdw

$R \subseteq (V - \Sigma) \times \Sigma^*((V - \Sigma) \cup \{\varepsilon\})$

Beispiel

Diese Grammatik ist regulär:

$G = (V, \Sigma, R, S)$, wo

$V = \{S, B, a, b\}$,
$\Sigma = \{a, b\}$,
$R = \{S \rightarrow aS, S \rightarrow B, B \rightarrow bB, B \rightarrow \varepsilon\}$

- und $L(G) = a^*b^*$

Theorem

Sei L eine kontextfreie Sprache. L ist regulär gdw $L=L(G)$ für eine reguläre Grammatik G.

Beweis

$\Rightarrow$: Angenommen, L ist regulär. Sei M ein DEA der L akzeptiert, $M = (K, \Sigma, \delta, s, F)$. Wir können annehmen, daß K und Σ keine gemeinsamen Elemente haben. Wir konstruieren jetzt die reguläre Grammatik $G = (V, \Sigma, R, S)$, wo

$V = \Sigma \cup K$,
$S = s$,
$R = \{q \rightarrow ap : \delta(q, a) = p\} \cup \{q \rightarrow \varepsilon : q \in F\}$

Weil $q \rightarrow ap$ gdw $\delta(q, a) = p$, gilt:

$$(q_0, \sigma_1\ldots\sigma_n) \vdash_M (q_1, \sigma_2\ldots\sigma_n) \vdash_M \ldots \vdash_M (q_{n-1}, \sigma_n) \vdash_M (q_n, \varepsilon)$$

gdw

$$q_0 \underset{G}{\Rightarrow} \sigma_1 q_1 \underset{G}{\Rightarrow} \ldots \underset{G}{\Rightarrow} \sigma_1\ldots\sigma_n q_n$$

Deshalb gilt:

$w \in L(M)$
gdw
Es gibt ein $p \in F$ so daß $(s, w) \vdash_M^* (p, \varepsilon)$ (Def.)
gdw
Es gibt ein $p \in F$ so daß $s \underset{G}{\overset{*}{\Rightarrow}} wp$
gdw
$w \in L(G)$

Zur letzten Äquivalenz: $\Downarrow$ gilt, weil es zu jedem $q \in F$ eine Regel $q \rightarrow \varepsilon$ gibt. $\Uparrow$ gilt, weil eine Regel vom Typ $q \rightarrow \varepsilon$ immer die zuletzt angewandte in einer Ableitung sein muß.

$\Leftarrow$: Angenommen, $L = L(G)$ für eine reguläre Grammatik $G = (V, \Sigma, R, S)$.

Wir konstruieren einen NEA $M = (K, \Sigma, \Delta, s, F)$, wo

$K = (V-\Sigma) \cup \{f\}$ (f neu)
$s = S$
$F = \{f\}$
$\Delta = \{(A, w, B) : A \underset{G}{\rightarrow} wB \text{ und } B \in V-\Sigma\}$
$\quad \cup \{(A, w, f) : A \underset{G}{\rightarrow} w \text{ und } w \in \Sigma^*\}$

Es gilt:

$w \in L(G)$

gdw

Es gibt $A_1, \ldots A_n \in V-\Sigma$, $w_0, \ldots w_n \in \Sigma^*$:
$w = w_0\ldots w_n$ und
$S \underset{G}{\Rightarrow} w_0A_1 \underset{G}{\Rightarrow} \ldots \underset{G}{\Rightarrow} w_0\ldots w_{n-1}A_n \underset{G}{\Rightarrow} w_0\ldots w_n$

gdw

Es gibt $A_1, \ldots A_n \in V-\Sigma$, $w_0, \ldots w_n \in \Sigma^*$:
$w = w_0\ldots w_n$ und
$(S, w_0\ldots w_n) \vdash_M (A_1, w_1\ldots w_n) \vdash_M \ldots$
$\quad \vdash_M (A_n, w_n) \vdash_M (f, \varepsilon)$

gdw

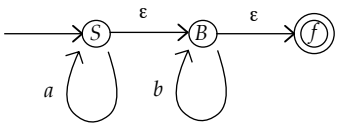
$w \in L(M)$ ❑

Beispiel

Betrachten wir wieder $G = (V, \Sigma, R, S)$, wo

$V = \{S, B, a, b\}$,
$\Sigma = \{a, b\}$,
$R = \{S \rightarrow aS, S \rightarrow B, B \rightarrow bB, B \rightarrow \varepsilon\}$

Die Konstruktion aus dem Beweis gibt uns den nichtdeterministischen Automaten



Korrolar

Die regulären Sprachen bilden eine echte Teilmenge der Menge der kontextfreien Sprachen.

Beweis

Jede reguläre Sprache ist auch kontextfrei: Sie wird von einer regulären (und deshalb auch kontextfreien) Grammatik generiert. Auf der anderen Seite gibt es kontextfreie Sprachen, die nicht regulär sind. ❑

Ableitungsbäume

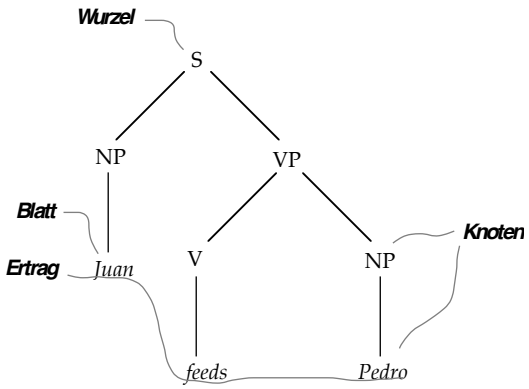
Ableitungsbäume stellen Ableitungen graphisch dar. Z.B. können wir die beiden in gewissem Sinne äquivalenten Ableitungen

S ⇒ NP VP ⇒ *Juan* VP ⇒ *Juan* V NP ⇒
Juan feeds NP ⇒ *Juan feeds Pedro*

und

S ⇒ NP VP ⇒ *Juan* VP ⇒ *Juan* V NP ⇒
Juan V *Pedro* ⇒ *Juan feeds Pedro*

mit diesem Ableitungsbaum darstellen:



Das Pumping-Lemma für kontextfreie Sprachen

Theorem

Sei G eine kontextfreie Grammatik. Dann gibt es eine Zahl K, so daß für jede Zeichenkette $w \in L(G)$ wo $|w| > K$ folgendes gilt:

Es gibt Zeichenketten $u, v, x, y, z \in \Sigma^*$ so daß:

1) $w = uvxyz$
2) mindestens eins von v und y ist nicht ε
3) Für jede $n \geq 0$: $uv^nxy^nz \in L(G)$

Um das Pumping-Lemma beweisen zu können, brauchen wir erst noch ein Paar Definitionen zu Ableitungsbäumen:

Ein *Pfad* in einem Ableitungsbaum ist eine Folge von Knoten, die mit der Wurzel anfängt und mit einem Blatt endet, und wo jedes Element in der Folge im Baum durch eine Kante mit dem vorigen Element verbunden ist.

Die *Länge* des Pfades ist gleich der Anzahl der Kanten, d.h. gleich der Anzahl der Knoten minus 1.

Die *Höhe* eines Baumes ist gleich der Länge des längsten Pfades.

Beweis des Pumping-Lemmas:

Sei $G = (V, \Sigma, R, S)$. Es genügt, folgendes zu zeigen:

(*) Es gibt ein K, so daß jede Zeichenkette in $L(G)$ länger als K eine Ableitung

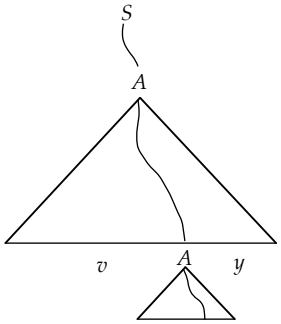
$$S \overset{*}{\underset{G}{\Rightarrow}} uAz \overset{*}{\underset{G}{\Rightarrow}} uvAyz \overset{*}{\underset{G}{\Rightarrow}} uvxyz$$

hat, wo $u, v, x, y, z \in \Sigma^*$, $A \in V-\Sigma$, und $v \neq \varepsilon \vee y \neq \varepsilon$,

weil die Ableitung $A \overset{*}{\underset{G}{\Rightarrow}} vAy$ dann beliebig viele Male (auch 0-mal) wiederholt werden kann.

Sei $p = \max\{|\alpha| : A \underset{G}{\rightarrow} \alpha\}$. Dann hat ein Ableitungsbaum der Höhe m höchstens p^m Blätter. Oder umgekehrt: Wenn ein Ableitungsbaum einen Ertrag mit Länge $> p^m$, dann hat der Baum einen Pfad der länger als m ist.

Sei $m = |V-\Sigma|$, p wie oben, $K = p^m$, w ein Wort mit Länge $> K$. Sei T ein Ableitungsbaum mit Wurzel S und Ertrag w. Dann hat T mindestens einen Pfad mit mehr als $|V-\Sigma|+1$ Knoten, und so ein Pfad muß mindestens zwei Knoten haben, die mit dem selben Nichtterminalen markiert sind :

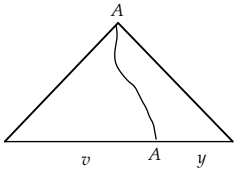


Wir haben also für jeden solchen Pfad eine Ableitung

$$S \overset{*}{\underset{G}{\Rightarrow}} uAz \overset{*}{\underset{G}{\Rightarrow}} uvAyz \overset{*}{\underset{G}{\Rightarrow}} uvxyz,$$

wie wir zeigen wollten. Nun muß nur noch gezeigt werden, daß es mindestens einen solchen Pfad gibt, wo mindestens eins von v und y nicht-leer ist.

Für jeden Fall, wo v und y beide leer sind, können wir den Teilbaum



einfach entfernen, ohne daß sich der Ertrag des Baumes ändert. Dies kann aber nicht für jeden Pfad mit Länge $> m$ möglich sein, weil es sonst einen Baum mit Höhe $\leq m$ gäbe, der w als Ertrag hat, und dies ist nicht möglich da $|w| > K = p^m$.❑

Beispiel

$L = \{a^n b^n c^n : n \geq 0\}$ ist *nicht* kontextfrei.

Beweis:

Nehmen wir an, L ist kontextfrei, d.h. $L = L(G)$ für kontextfreie G. Sei K wie im Pumping-Lemma, und sei $n > K/3$. Dann erfüllt $w = a^n b^n c^n$ die Voraussetzung des Pumping-Lemmas ($|w| > K$).

Also ist $w = uvxyz$, wo mindestens eins von v und y nicht-leer ist. Es gibt zwei Möglichkeiten:

1) Mindestens eins von v oder y enthält zwei verschiedene Zeichen (a und b oder b und c), und deshalb die Teilzeichenkette ab oder bc. Dann kommt aber in uv^2xy^2z mindestens ein b vor einem a vor, oder ein c vor einem b, also $w \notin L(G)$ – Kontradiktion!

2) v oder y oder beide bestehen nur aus gleichen Zeichen (z.B. $v = bb$, $y = cc$). Dann hat aber uv^2xy^2z nicht die selbe Anzahl von a's, b's und c's (sondern z.B. 2 a's zuwenig, wenn $v=bb$ und $y=cc$) – Kontradiktion! ❑

Abschlußeigenschaften der Klasse der kontextfreien Sprachen

Wir fangen mit einem negativen Ergebnis an, das direkt aus dem vorigen Beispiel folgt:

Theorem

Die Klasse der kontextfreien Sprachen ist unter Schnittbildung *nicht* abgeschlossen.

Beweis

Die Sprachen L_1 und L_2 sind kontextfrei (Grammatiken können leicht gegeben werden):

$L_1 = \{a^m b^n c^n : m, n \geq 0\}$

$L_2 = \{a^m b^m c^n : m, n \geq 0\}$

Aber $L_1 \cap L_2 = \{a^n b^n c^n : n \geq 0\}$, und diese Sprache ist, wie wir jetzt wissen, *nicht* kontextfrei. ❑

Theorem

Die Klasse der kontextfreien Sprachen ist unter Vereinigung, Konkatenation und Kleene'scher Hüllenbildung abgeschlossen.

Beweis

Seien $G_1 = (V_1, \Sigma_1, R_1, S_1)$ und $G_2 = (V_2, \Sigma_2, R_2, S_2)$. Wir können annehmen, daß $(V_1 - \Sigma_1) \cap (V_2 - \Sigma_2) = \varnothing$ (z.B. durch das Umbenennen der ganzen Nichtterminalen, indem man die Symbole mit 1, bzw. 2, indexiert)

Vereinigung:

Sei S ein neues Symbol. $G = (V_1 \cup V_2, \Sigma_1 \cup \Sigma_2, R, S)$ generiert $L(G_1) \cup L(G_2)$, wo

$R = R_1 \cup R_2 \cup \{S \rightarrow S_1, S \rightarrow S_2\}$.

Daß $L(G) \supseteq L(G_1) \cup L(G_2)$ ist klar. Weil G_1 und G_2 keine gemeinsamen Nichtterminale haben, gilt auch die umgekehrte Richtung.

Konkatenation: Ähnlich, mit neuer Regel $S \rightarrow S_1S_2$.

Kleene'sche Hülle:

Sei $G = (V_1, \Sigma_1, R_1 \cup \{S_1 \rightarrow \varepsilon, S_1 \rightarrow S_1S_1\}, S_1)$. Dann ist $L(G) = L(G_1)^*$.

❑

Korrolar

Die Klasse der kontextfreien Sprachen ist unter Komplementbildung *nicht* abgeschlossen.

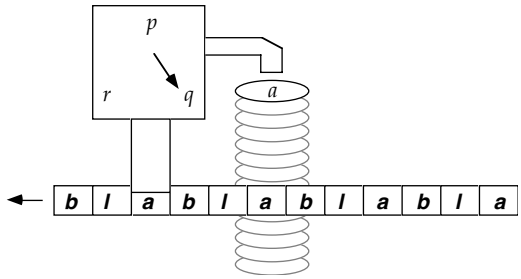
Beweis

Wenn sie das wäre, wäre sie auch unter Schnittbildung abgeschlossen, weil:

$L_1 \cap L_2 = \Sigma^* - ((\Sigma^* - L_1) \cup (\Sigma^* - L_2))$ ❑

Kellerautomaten

Kellerautomaten sind wie *nicht*-deterministische endliche Automaten, mit einem zusätzlichen *Keller*:



Der Keller ist wie ein "bodenloser" Stapel von Zeichen. Der Automat darf bei jedem Schritt beliebig viele Zeichen von diesem Stapel entfernen (aber nur von oben), und beliebig viele Zeichen auf den Stapel legen.

Kellerautomaten formal definiert:

Ein Kellerautomat M besteht aus

- K – einer endlichen Menge von *Zuständen*
- Σ – einem *Eingabealphabet*
- Γ – einem *Kelleralphabet*
- s – einem *Startzustand* ($s \in K$)
- F – einer Menge von *Endzuständen* ($F \subseteq K$)
- Δ – einer *Übergangsrelation*: Eine endliche Teilmenge von $(K \times \Sigma^* \times \Gamma^*) \times (K \times \Gamma^*)$

Intuitiv bedeutet $\Delta((p, u, \beta), (q, \gamma))$, daß M im Zustand p, mit β als oberstem Teil des Kellers, u lesen darf, und dabei in den Zustand q übergeht, β vom Keller entfernt, und γ auf den Keller legt.

Konfigurationen

Eine Konfiguration eines KA gibt den Zustand, die noch nicht gelesene Eingabe und den Inhalt des Kellers (von oben bis unten) an, z.B.

$(p, abba, 001110)$ (bei $\Sigma = \{a, b\}$, $\Gamma = \{0, 1\}$).

Die *ergibt*-Relation für KA

$(q, w, \alpha) \vdash_M (q', w', \alpha')$

gdw

$w = uw'$ für ein $u \in \Sigma^*$,
$\alpha = \beta\gamma$ und $\alpha' = \beta'\gamma$ für Zeichenketten $\beta, \beta', \gamma \in \Gamma^*$,
und $\Delta((q, u, \beta), (q', \beta'))$.

$\vdash_M^*$ (lese: *ergibt*) ist wieder die *reflexiv-transitive Hülle* von $\vdash_M$

Die von einem KA akzeptierte Sprache

Ein Kellerautomat akzeptiert die Eingabe, wenn es eine Berechnung gibt, bei deren Ende der Automat sich mit leerer Resteingabe *und* leerem Keller in einem Endzustand befindet:

$w \in L(M)$

gdw

Es gibt ein $q \in F$ so daß $(s, w, \varepsilon) \vdash_M^* (q, \varepsilon, \varepsilon)$

Beispiele

1. Jeder NEA kann als ein KA mit leerem Kelleralphabet betrachtet werden.
2. Ein sehr nicht-deterministischer Automat für die Sprache $\{ww^R: w \in \{a, b\}^*\}$:

   $M = (K, \Sigma, \Gamma, \Delta, s, F)$, wo

   $K = \{s, f\}$,
   $\Sigma = \Gamma = \{a, b\}$,
   $F = \{f\}$,
   $\Delta = \{ ((s, a, \varepsilon), (s, a)),$
   $((s, b, \varepsilon), (s, b)),$
   $((s, \varepsilon, \varepsilon), (f, \varepsilon)),$
   $((f, a, a), (f, \varepsilon)),$
   $((f, b, b), (f, \varepsilon)) \}$

   Im mittleren Übergang wird "geraten", daß die Mitte der Eingabe erreicht wurde, und der Automat fängt an, die im Keller gespeicherten Zeichen mit dem Rest der Eingabe zu vergleichen.

Linksableitungen

Eine *direkte Linksableitung* ist eine direkte Ableitung wo das am weitesten links stehende Nichtterminalsymbol ersetzt wird.

Eine *Linksableitung* ist eine Kette von direkten Linksableitungen.

Notation:

$w \overset{L}{\underset{G}{\Rightarrow}} w'$ für direkte Linksableitungen

$w \overset{L^n}{\underset{G}{\Rightarrow}} w'$ für Linksableitungen in n Schritten

$w \overset{L^*}{\underset{G}{\Rightarrow}} w'$ wie immer wenn: für ein n ≥ 0: $w \overset{L^n}{\underset{G}{\Rightarrow}} w$.

Theorem

Sei G eine kontextfreie Grammatik und $w \in \Sigma^*$.
Dann gilt: $w \in L(G)$ gdw $S \overset{L^*}{\underset{G}{\Rightarrow}} w$

Beweis: Jede Linksableitung ist eine Ableitung. Für die andere Richtung: $w \in L(G)$ gdw $S \overset{*}{\underset{G}{\Rightarrow}} w$. Zu einer Ableitung von w gehört ein Ableitungsbaum. Von diesem Ableitungsbaum läßt sich eine Linksableitung von w leicht erzeugen.❑

Kontextfreie Sprachen und Kellerautomaten

Theorem

Eine Sprache L ist kontextfrei gdw $L = L(M)$ für einen Kellerautomaten M.

Wir werden nur eine Richtung dieses Theoremes beweisen. Die folgende Hälfte werden wir *nicht* beweisen:

Theorem

Wenn $L = L(M)$ für einen Kellerautomaten M, ist L kontextfrei.

Der Beweis ist ziemlich umständlich, und bringt für computerlinguistische Zwecke nicht besonders viel.

Zu zeigen bleibt also:

Theorem

Zu jeder kontextfreien Sprache gibt es einen KA, der L akzeptiert.

Beweis

Sei G eine kontextfreie Grammatik, $G = (V, \Sigma, R, S)$.

Wir werden zeigen daß $L(G) = L(M)$, wo

$M = (K, \Sigma, \Gamma, \Delta, s, \{f\})$,

wo $K = \{s, f\}$,
$\Gamma = V$,
$\Delta = \{((s, \varepsilon, \varepsilon), (f, S))\}$
$\cup \{((f, \varepsilon, A), (f, x)) : A \underset{G}{\rightarrow} x\}$
$\cup \{((f, a, a), (f, \varepsilon)) : a \in \Sigma\}$

Der Automat simuliert eine Linksableitung: Jede Anwendung einer Regel wird durch das Ersetzen des obersten Kellersymbols mit dem entsprechenden Zeichen aus der Regel simuliert, und wenn danach das oberste Kellersymbol ein Terminalsymbol ist, wird es gelöscht – falls es mit der Eingabe übereinstimmt.

Beispiel: Wir betrachten wieder
$G = (\{S, a, b\}, \{a, b\}, \{S \rightarrow aSb, S \rightarrow \varepsilon\}, S)$

Dann ist $M = (\{s, f\}, \{a, b\}, \{S, a, b\}, \Delta, s, \{f\})$,
wo $\Delta = \{ ((s, \varepsilon, \varepsilon), (f, S)),$
$((f, \varepsilon, S), (f, aSb)),$
$((f, \varepsilon, S), (f, \varepsilon)),$
$((f, a, a), (f, \varepsilon)),$
$((f, b, b), (f, \varepsilon)) \}$

und z.B.: $(s, aabb, \varepsilon) \vdash_M (f, aabb, S) \vdash_M (f, aabb, aSb)$
$\vdash_M (f, abb, Sb) \vdash_M (f, abb, aSbb) \vdash_M (f, bb, Sbb)$
$\vdash_M (f, bb, bb) \vdash_M (f, b, b) \vdash_M (f, \varepsilon, \varepsilon)$

Wir beweisen die folgenden zwei Behauptungen:

(1) Wenn $S \overset{L^*}{\underset{G}{\Rightarrow}} \alpha_1\alpha_2$, wo $\alpha_1 \in \Sigma^*$ und $\alpha_2 \in (V-\Sigma)V^* \cup \{\varepsilon\}$, dann $(f, \alpha_1, S) \vdash_M^* (f, \varepsilon, \alpha_2)$.

(2) Wenn $(f, \alpha_1, S) \vdash_M^* (f, \varepsilon, \alpha_2)$, wo $\alpha_1 \in \Sigma^*$ und $\alpha_2 \in V^*$, dann $S \overset{L^*}{\underset{G}{\Rightarrow}} \alpha_1\alpha_2$.

(1) und (2) reichen zusammen aus, um das Theorem zu beweisen (setze $\alpha_2 = \varepsilon$ in beiden Fällen).

Beweis von (1):

Induktion über *Ableitungslänge*:

Induktionsannahme:

(1) gilt für Ableitungen mit Länge ≤ n.

Induktionsbasis:

$S \overset{L^*}{\underset{G}{\Rightarrow}} \alpha_1\alpha_2$ ist eine Ableitung in 0 Schritten.
D.h.: $S = \alpha_1\alpha_2$, also $\alpha_1 = \varepsilon$, $\alpha_2 = S$.
$(f, \alpha_1, S) \vdash_M^* (f, \varepsilon, \alpha_2)$ ist dann einfach
$(f, \varepsilon, S) \vdash_M^* (f, \varepsilon, S)$ – trivial!

Induktionsschritt:

Wir betrachten den letzten Schritt einer Linksableitung in n+1 Schritten:
Wenn $S \overset{L^{n+1}}{\underset{G}{\Rightarrow}} \alpha_1\alpha_2$, dann $S \overset{L^n}{\underset{G}{\Rightarrow}} \beta_1 A\beta_2 \Rightarrow \beta_1\gamma\beta_2 = \alpha_1\alpha_2$,
wo $\beta_1 \in \Sigma^*$, $\beta_2, \gamma \in V^*$ und $A \underset{G}{\rightarrow} \gamma$.

Für die ersten n Schritte gilt die I.A., also gilt

(*) $(f, \beta_1, S) \vdash_M^* (f, \varepsilon, A\beta_2)$.

Weil $\alpha_1 \in \Sigma^*$ ist $\alpha_1 = \beta_1\delta$, wo $\delta \in \Sigma^*$.
Und dann ist $\delta\alpha_2 = \gamma\beta_2$.

Es gilt:

$(f, \alpha_1, S) =$	$(f, \beta_1\delta, S)$	(weil $\alpha_1 = \beta_1\delta$)
	$\vdash_M^* (f, \delta, A\beta_2)$	(wegen (*) + kleiner Beweis (Aufgabe!))
	$\vdash_M (f, \delta, \gamma\beta_2)$	($A \underset{G}{\rightarrow} \gamma$)
	$\vdash_M^* (f, \varepsilon, \alpha_2)$	(weil $\delta\alpha_2 = \gamma\beta_2$)

und damit ist der Induktionsschritt vollständig.

Beweis für (2):

Induktion über die Länge einer M-Berechnung:

Induktionsannahme:

(2) gilt für Berechnungen in ≤ n Schritten.

Induktionsbasis:

$(f, \alpha_1, S) \vdash_M^* (f, \varepsilon, \alpha_2)$ in 0 Schritten,
d.h. $\alpha_1 = \varepsilon$, $\alpha_2 = S$, und dann gilt $S \overset{L^*}{\underset{G}{\Rightarrow}} \alpha_1\alpha_2$

Induktionsschritt:

Wenn $(f, \alpha_1, S) \vdash_M^{n+1} (f, \varepsilon, \alpha_2)$, dann

$(f, \alpha_1, S) \vdash_M^n (f, \beta, \gamma) \vdash_M (f, \varepsilon, \alpha_2)$, wo $\beta \in \Sigma^*$, $\gamma \in \Gamma^*$.

Der zuletzt benützte Übergang war entweder

$((f, \beta, \beta), (f, \varepsilon))$, wo $\beta \in \Sigma$ (Fall (1))
oder $((f, \varepsilon, A), (f, \delta))$, wo $A \underset{G}{\rightarrow} \delta$ (Fall (2))

Fall (1):

Dann ist $\alpha_1 = \delta\beta$ (β ist das letzte Zeichen der Eingabe!) für $\delta \in \Sigma^*$, $\gamma = \beta\alpha_2$, und:

$(f, \alpha_1, S) = (f, \delta\beta, S) \vdash_M^n (f, \beta, \beta\alpha_2) \vdash_M (f, \varepsilon, \alpha_2)$

…dann aber auch $(f, \delta, S) \vdash_M^n (f, \varepsilon, \beta\alpha_2)$, und für diese Berechnung gilt die I.A, also gilt:

$S \overset{L^*}{\underset{G}{\Rightarrow}} \delta\beta\alpha_2 = \alpha_1\alpha_2$.

Fall (2):

Dann ist $\beta = \varepsilon$ (im letzten Schritt wird keine Eingabe gelesen!), $\gamma = A\gamma_1$ für ein $\gamma_1 \in \Gamma^*$, und $\alpha_2 = \delta\gamma_1$.

Weil $\beta = \varepsilon$, kann die I.A. direkt auf die n ersten Berechnungsschritte angewendet werden, und es gilt:

$S \overset{L^*}{\underset{G}{\Rightarrow}} \alpha_1 A\gamma_1 \Rightarrow \alpha_1\delta\gamma_1 = \alpha_1\alpha_2$.❑

Theorem

Sei L kontextfrei und R regulär.
Dann ist $L \cap R$ kontextfrei.

Beweis

Zu L gibt es einen KA M_1, der L akzeptiert.
Zu R gibt es einen DEA M_2, der R akzeptiert.

Wir werden aus diesen Automaten einen KA M konstruieren, der $L \cap R$ akzeptiert. Weil jede von einem KA akzeptierte Sprache kontextfrei ist (nicht bewiesen…), ist dann auch $L \cap R$ kontextfrei.

M ist wie M_1, speichert aber gleichzeitig, in welchem Zustand M_2 sich bei gleich viel gelesener Eingabe befindet (es ist wichtig, daß M_2 deterministisch ist!):

Sei $M = (K, \Sigma, \Gamma_1, \Delta, s, F)$,

wo $K = K_1 \times K_2$,
$\Sigma = \Sigma_1 \cup \Sigma_2$
$s = (s_1, s_2)$,
$F = F_1 \times F_2$,

und
$\Delta(((q_1, q_2), u, \beta), ((p_1, p_2), \gamma))$

gdw

$\Delta_1((q_1, u, \beta), (p_1, \gamma))$ und $(q_2, u) \vdash_{M_2}^* (p_2, \varepsilon)$ ❑

Beispiel

$L = \{w \in \{a, b, c\}^* : \#a\text{'s in } w = \#b\text{'s in } w = \#c\text{'s in } w\}$

ist *nicht* kontextfrei.

Beweis: Wenn L kontextfrei wäre, dann auch $L \cap a^*b^*c^* = \{a^nb^nc^n : n \geq 0\}$! ❑

Entscheidbarkeit

Sei G eine kontextfreie Grammatik.
Können wir die Frage

Ist $w \in L(G)$?

beantworten?

Statt eine direkte Antwort zu geben, werden wir zeigen das es zu G einen *Parser* gibt. Die Antwort *ja* folgt dann unmittelbar.

Parsing

Was ist Parsing?

> **Analyse, Syntaktische.** (Parsing) Verfahren zur Strukturanalyse von Sätzen einer formalen oder natürlichen Sprache mit Hilfe der zugrundeliegenden Grammatik. (…)
>
> (Lexikon Informatik und Kommunikationstechnik, VDI Verlag)

Bei Parsing interessiert uns also nicht nur die Frage, ob eine Zeichenkette zu einer Sprache gehört, sondern auch welchen Ableitungsbaum (oder, wenn die Grammatik mit Absicht mehrdeutig ist, wie es bei natürlichen Sprachen oft der Fall ist, Ableitungs-*bäume*) die Zeichenkette in der gegebenen Grammatik hat.

Im Folgenden werden wir uns der Frage widmen, inwieweit Kellerautomaten, die von Grammatiken abgeleitet sind, als Parser benutzt werden können.

Wir haben schon gezeigt, daß wir zu jeder Grammatik einen KA erzeugen können, der die von der Grammatik generierte Sprache akzeptiert. Wir betrachten jetzt einen Automaten, der durch diese Methode entstanden ist.

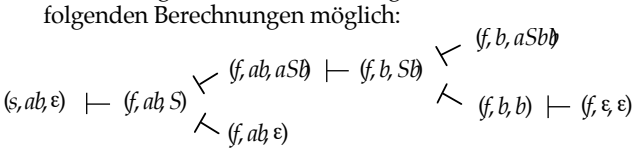
Beispiel

Von $G = (\{a, b, S\}, \{a, b\}, \{S \rightarrow aSb, S \rightarrow \varepsilon\}, S)$, die $L = \{a^nb^n : n \geq 0\}$ erzeugt, wird folgender Automat konstruiert:

$M = (\{s, f\}, \{a, b\}, \{S, a, b\}, \Delta, s, \{f\})$,
wo $\Delta = \{$ $((s, \varepsilon, \varepsilon), (f, S))$,
$((f, \varepsilon, S), (f, aSb))$,
$((f, \varepsilon, S), (f, \varepsilon))$,
$((f, a, a), (f, \varepsilon))$,
$((f, b, b), (f, \varepsilon))$ $\}$

Die beiden Transitionen, die den Regeln entsprechen, sind *kompatibel*, also ist M nicht deterministisch.

Trotzdem ist M ein anwendbarer Parser für L: Für jede Eingabe gibt es nur endlich viele, endlich lange Berechnungen. Z.B. sind bei Eingabe *ab* nur die folgenden Berechnungen möglich:



Streng genommen ist M kein Parser, weil er keine Ausgabe gibt. Eine Linksableitung (und damit ein Ableitungsbaum) kann aber von den Schritten der Berechnung, die Regelanwendungen entsprechen, direkt abgelesen werden ($S \Rightarrow aSb \Rightarrow ab$). Einige Lehrbücher führen *Transducer*-KA's ein, die Regelanwendungen auf ein Ausgabeband schreiben.

Top-down-Parser

Den Automaten von Beweis k.f.G. -> KA nennen wir den *Top-down-KA zu G*. Für die Beispiel-grammatik war dieser KA direkt als Top-down-Parser anwendbar. Im allgemeinen können wir aber nicht sicher sein, daß der konstruierte Automat bei jeder Eingabe hält.

Wir stellen jetzt fest, daß der Automat genau dann in unendliche Schleifen gerät, wenn die Grammatik *linksrekursiv* ist:

Definition

Eine Grammatik ist linksrekursiv gdw sie ein vom Startsymbol erreichbares linksrekursives Nichtterminalsymbol hat, d.h. gdw

Es gibt ein $A \in V - \Sigma$ und $x, y, u \in V^*$ so daß:
$S \overset{*}{\underset{G}{\Rightarrow}} xAy \overset{+}{\underset{G}{\Rightarrow}} xAu$

Beispiel

Wir werden folgende Beispielgrammatik

$G = (\{S, L, (,), a, \_\}, \{(,), a, \_\}, R, S)$

benutzen, wo R die folgenden Regeln enthält:

$R = \{\ S \to (),\quad S \to a \quad S \to (L)$
$\quad\ L \to S,\quad L \to L\_S\ \}$

G stellt die Syntax von "S-Expressions" dar, die der Hauptbestandteil der Syntax der Programmier-sprache LISP sind.

(Im Vergleich zu wirklichen CommonLisp-S-Expressions wird hier *a* als Stellvertreter der Symbole, und _ des "whitespace", benutzt)

Wir betrachten jetzt die Linksrekursion in der zweiten L-Regel ($L \to L\_S$). Linksrekursion heißt hier, daß das Zeichen der linken Seite einer Regel als am weitesten links stehendes Zeichen auf der rechten Seite vorkommt.

Der von unserer Beispielgrammatik direkt konstruierte Parser hat die folgende Transition:

$((f, \varepsilon, L), (f, L\_S))$

Diese Transition ist beliebig viele Male nacheinander anwendbar, deshalb ist dieser Automat kein Parser.

Allgemein können wir beweisen:

Theorem

Sei G eine Grammatik, und sei M der entsprechende Top-down-Kellerautomat. Dann gibt es zu jeder Eingabe nur endlich viele, endlich lange Berechnungen gdw G nicht linksrekursiv ist.

Beweis

Wenn G linksrekursiv ist, hat der Automat Teilberechnungen, die folgendermaßen aussehen:

$(q, x, A) \vdash_M^* (q, x, A\alpha)$

wo A vom S aus erreichbar ist. M hat dann die Möglichkeit, diese Teilberechnung beliebig viele Male durchzuführen, und kann deshalb in eine unendliche Schleife geraten.

Umgekehrt: M legt zuerst S auf den Keller, und tauscht nachher das oberste Kellersymbol um, bis ein Terminalsymbol als oberstes Kellersymbol vorkommt. Dann wird das entsprechende Zeichen von der Eingabe gelesen. Wenn G nicht linksrekursiv ist, ist jede solche Kette von Substitutionen höchstens $|V - \Sigma|$ lang, also hat keine Berechnung mehr als $|w|\,|V - \Sigma|$ Schritte. Deshalb gibt es auch nur endlich viele mögliche Berechnungen.❑

Zurück zum Beispiel:

Es ist klar, daß die folgenden Regeln genau die selben Ausdrücke generieren wie die in der Grammatik gegebenen L-Regeln:

$L \to SL'$
$L' \to \_SL'$
$L' \to \varepsilon$

Im allgemeinen läßt sich Linksrekursion innerhalb von Regeln folgendermaßen entfernen:

Methode zum Entfernen von Linksrekursion:

Sei G eine Grammatik mit den folgenden A-Regeln (wir nehmen an, die Regel $A \to A$ kommt nicht vor): $A \to A\alpha_1, \ldots, A \to A\alpha_n, A \to \beta_1, \ldots, A \to \beta_m$, wo $n>0$ und kein β fängt mit einem A an.

Ersetze dann diese Regeln durch die Regeln

$A \to \beta_1 A', \ldots, A \to \beta_m A'$
$A' \to \alpha_1 A', \ldots, A' \to \alpha_n A'$
$A' \to \varepsilon$

wo A' ein neues Nichtterminalsymbol ist.

Theorem

Sei G eine kontextfreie Grammatik, und G' die Grammatik, die von G durch Anwendung der obigen Methode entsteht. Dann ist $L(G') = L(G)$.

Beweis

<u>$L(G') \supseteq L(G)$:</u> Betrachte folgende Linksableitung von einem beliebigen Wort w in G:

$S \overset{L}{\underset{G}{\Rightarrow}} \gamma_0 \overset{L}{\underset{G}{\Rightarrow}} \gamma_1 \overset{L}{\underset{G}{\Rightarrow}} \ldots \overset{L}{\underset{G}{\Rightarrow}} \gamma_k = w$

Ein Ableitungsschritt, in dem eine von den "β-Regeln" angewandt wurde; d.h.: wo

$\gamma_j = xA\gamma'$ und $\gamma_{j+1} = x\beta_i\gamma'$, $i \le m$,

läßt sich durch die beiden Schritte

$xA\gamma' \overset{L}{\underset{G'}{\Rightarrow}} x\beta_iA'\gamma' \overset{L}{\underset{G'}{\Rightarrow}} x\beta_i\gamma'$ ersetzen.

Eine Reihe von l Ableitungsschritten, in denen linksrekursive Regeln angewandt wurden, sieht im allgemeinen folgendermaßen aus:

(1) $xA\gamma' \overset{L}{\underset{G}{\Rightarrow}} xA\alpha_{j_1}\gamma' \overset{L}{\underset{G}{\Rightarrow}} \dots \overset{L}{\underset{G}{\Rightarrow}} xA\alpha_{j_l}\dots\alpha_{j_1}\gamma' \overset{L}{\underset{G}{\Rightarrow}} x\beta_p\alpha_{j_l}\dots\alpha_{j_1}\gamma'$

und kann mit

(2) $xA\gamma' \overset{L}{\underset{G'}{\Rightarrow}} x\beta_pA'\gamma' \overset{L}{\underset{G'}{\Rightarrow}} x\beta_p\alpha_{j_l}A'\gamma' \overset{L}{\underset{G'}{\Rightarrow}} \dots \overset{L}{\underset{G'}{\Rightarrow}} x\beta_p\alpha_{j_l}\dots\alpha_{j_1}\gamma'$

ersetzt werden.

$L(G') \subseteq L(G)$: Wenn w aus S G'-ableitbar ist, müssen wir Teilableitungen, die A' enthalten, entfernen. Solche sehen aber wie (2) (möglicherweise mit l=0) aus, und wir können sie mit Ableitungen wie (1) ersetzen.❑

Indirekte Linksrekursivität

Linksrekursivität kommt nicht nur in einzelnen Regeln vor. Z.B. ist A in den folgenden Regeln linksrekursiv:

$A \to Bc$, $B \to Aa$

Es gibt aber eine Erweiterung des obigen Algorithmus, die *jede* Art von Linksrekursion beseitigt (wird nicht vorgestellt). Also gilt:

Theorem

Zu jeder kontextfreien Sprache gibt es eine Grammatik ohne Linksrekursion.

Bottom-up-Verfahren: Shift-reduce-Parser

Statt zur Eingabe w von S aus nach Linksableitungen zu suchen, können wir die umgekehrte Strategie wählen: Von w aus *Rechtsableitungen* rekonstruieren. Dies nennt man *Bottom-up*-Parsing, weil man den Ableitungsbaum von unten, vom Ertrag aus, konstruiert.

Wir können statt den schon vertrauten Top-down-KA einen Bottom-up-KA konstruieren:

Von der Grammatik $G = (V, \Sigma, R, S)$ konstruieren wir den Automaten $M = (\{s, f\}, \Sigma, V, \Delta, s, \{f\})$, wo Δ die folgenden Transitionen hat:

Für jede $\sigma \in \Sigma$:	$((s, \sigma, \varepsilon), (s, \sigma))$	("*shift*")
Für jede $A \to \alpha \in R$:	$((s, \varepsilon, \alpha^R), (s, A))$	("*reduce*")
und schließlich:	$((s, \varepsilon, S), (f, \varepsilon))$	

M benutzt ein sogenanntes *shift-reduce*-Verfahren: Er hat zwei Typen von Bewegungen zur Wahl: Eine Möglichkeit ist, daß M ein Zeichen von der Eingabe liest und auf den Keller legt (wenn nur dies gemacht wird, enthält der Keller schließlich die ganze Eingabe in umgekehrter Reihenfolge). Die andere Möglichkeit ist, daß die obersten Zeichen des Kellers der rechten Seite einer Regel (gespiegelt) entsprechen. Dann kann M diese Zeichen mit der entsprechenden linken Seite austauschen.

Beispiel

Von der LISP-Grammatik (in der ursprünglichen Version) erhalten wir folgenden Automaten:

$M = (\{s, f\}, \{\mathbf{(}, \mathbf{)}, a, \_\}, \{S, L, \mathbf{(}, \mathbf{)}, a, \_\}, \Delta, s, \{f\})$, wo

$\Delta = \{$	$((s, \mathbf{(}, \varepsilon), (s, \mathbf{(}))$,	(shift **(**)
	$((s, \mathbf{)}, \varepsilon), (s, \mathbf{)}))$,	(shift **)**)
	$((s, a, \varepsilon), (s, a))$	(shift *a*)
	$((s, \_, \varepsilon), (s, \_))$	(shift _)
(5)	$((s, \varepsilon, \mathbf{)}\mathbf{(}), (s, S))$	(reduce: **()** ← *S*)
(6)	$((s, \varepsilon, a), (s, S))$	(reduce: *a* ← *S*)
(7)	$((s, \varepsilon, \mathbf{)}L\mathbf{(}), (s, S))$	(reduce: **(***L***)** ← *S*)
(8)	$((s, \varepsilon, S), (s, L))$	(reduce: *S* ← *L*)
(9)	$((s, \varepsilon, S\_L), (s, L))$	(reduce: *L*_*S* ← *L*)
	$((s, \varepsilon, S), (f, \varepsilon))$ }	

Z.B. macht M u.a. die folgende Berechnung bei Eingabe **(()**_*a***)**, d.h. mit Startkonfig. (*s*, **(()**_*a***)**, ε):

Erst dreimal *Shift*:
$\vdash_M (s, \mathbf{()}\_a\mathbf{)}, \mathbf{()} \vdash_M (s, \mathbf{)}\_a\mathbf{)}, \mathbf{(()} \vdash_M (s, \_a\mathbf{)}, \mathbf{)(()}$

Dann zweimal *Reduce*, mit den Transitionen (5), (8):
$\vdash_M (s, \_a\mathbf{)}, S\mathbf{()} \vdash_M (s, \_a\mathbf{)}, L\mathbf{()}$

Zweimal *Shift*:
$\vdash_M (s, a\mathbf{)}, \_L\mathbf{()} \vdash_M (s, \mathbf{)}, a\_L\mathbf{()}$

Und zweimal *Reduce*, mit den Transitionen (6), (9):
$\vdash_M (s, \mathbf{)}, S\_L\mathbf{()} \vdash_M (s, \mathbf{)}, L\mathbf{()}$

Schließlich ein *Shift*, ein *Reduce* und fertig:
$\vdash_M (s, \varepsilon, \mathbf{)}L\mathbf{()} \vdash_M (s, \varepsilon, S) \vdash_M (f, \varepsilon, \varepsilon)$

Hält der Automat bei jeder Eingabe?

Man sieht, daß M bei jeder Eingabe nur endlich viele, endlich lange Berechnungen macht: Es besteht zwar die Möglichkeit zwischen einem Shift und einem Reduce zu wählen. Aber man kann nur so viel Shifts machen wie es Zeichen in der Eingabe gibt.

Ein Problem gäbe es, wenn es möglich wäre, beliebig viele Male zu reduzieren, ohne daß die Zahl der Zeichen auf dem Keller geringer würde. Bei dieser Grammatik ist das kein Problem: Die einzigen Transitionen, in denen der Kellerinhalt nicht kürzer wird, sind (6) und (8), aber ein L kann danach nicht weiter reduziert werden.

Im allgemeinen müssen wir verlangen, daß G nicht *zirkulär* ist, d.h. $A \overset{+}{\underset{G}{\Rightarrow}} A$ soll für kein $A \in V - \Sigma$ gelten.

Ein anderes Problem besteht, wenn G ε-Regeln enthält. Die kann man ja *immer* anwenden, und deshalb den Keller beliebig vergrößern. Deshalb werden wir die hier nicht zulassen. (Es *ist* aber möglich, ε-Regeln bis zu einem gewissen Grad zuzulassen!)

Wir können jetzt folgendes feststellen:

Theorem

Wenn eine Grammatik ε-frei und nicht-zirkulär ist, macht der entsprechende Bottom-up-KA bei jeder Eingabe nur endlich viele, endlich lange Berechnungen.❑

Theorem

Sei G eine kontextfreie Grammatik, und M der entsprechende Bottom-up-Kellerautomat. Dann gilt $L(M) = L(G)$.

Beweis

Wie fast immer bei Induktionsbeweisen, beweisen wir eine etwas stärkere Aussage:

Für $x \in \Sigma^*$ und $\gamma \in \Gamma^*$ gilt:
$(s, x, \gamma) \vdash_M^* (s, \varepsilon, S)$ gdw $S \overset{*R}{\underset{G}{\Rightarrow}} \gamma^R x$.

⇒: Induktion über Berechnungslänge:

Induktionsannahme:

Die ⇒-Richtung der Behauptung gilt für Berechnungen mit bis zu k Schritten.

Induktionsbasis:

k=0 bedeutet, daß x=ε und $\gamma = S$, und $S \overset{*R}{\underset{G}{\Rightarrow}} \gamma^R x$ gilt.

Induktionsschritt:

Eine Berechnung in k+1 Schritten sieht so aus:

$(s, x, \gamma) \vdash_M (s, w, \beta) \vdash_M^k (s, \varepsilon, S)$

Nach der Induktionsannahme gilt $S \overset{*R}{\underset{G}{\Rightarrow}} \beta^R w$.

Für den ersten Schritt müssen wir 2 Fälle unterscheiden:

1) Er ist ein *shift*-Schritt: Dann ist $x = \sigma w$ und $\beta = \sigma\gamma$ für ein Terminalsymbol σ, und $\gamma^R x = \gamma^R \sigma w = \beta^R w$.
2) Er ist ein *reduce*-Schritt: Dann ist $x = w$ und es gibt eine Regel $A \rightarrow \alpha$ und ein $\delta \in \Gamma^*$ so daß $\beta = A\delta$ und $\gamma = \alpha^R\delta$. Es gilt dann $S \overset{*R}{\underset{G}{\Rightarrow}} \beta^R w = \delta^R A w = \delta^R A x \overset{R}{\underset{G}{\Rightarrow}} \delta^R \alpha x = \gamma^R x$.

⇐: Induktion über die Länge einer Rechtsableitung von $\gamma^R x$:

Induktionsannahme:

Die ⇐-Richtung der Behauptung gilt für Rechtsableitungen mit bis zu k Schritten.

Induktionsbasis:

k=0: Dann x=ε und $\gamma = S$, und $(s, x, \gamma) \vdash_M^* (s, \varepsilon, S)$.

Induktionsschritt:

Eine Ableitung in k+1 Schritten sieht so aus:

$S \overset{kR}{\underset{G}{\Rightarrow}} \beta^R A w \overset{R}{\underset{G}{\Rightarrow}} \beta^R \alpha w = \gamma^R x$

Die Induktionsannahme gibt uns, daß

$(s, w, A\beta) \vdash_M^* (s, \varepsilon, S)$

Weil $|x| \geq |w|$ (Rechtsableitung!) gibt es ein u so daß $\beta^R\alpha = \gamma^R u$ und $uw = x$. Ein *reduce*-Schritt gibt uns $(s, w, \alpha^R\beta) \vdash_M (s, w, A\beta)$, und $|u|$ *shift*-Schritte gibt uns:

$(s, x, \gamma) = (s, uw, \gamma) \vdash_M^* (s, w, u^R\gamma) = (s, w, \alpha^R\beta)$

Also gilt $(s, x, \gamma) \vdash_M^* (s, \varepsilon, S)$. q

Jetzt wissen wir, daß der Bottom-up Parser *korrekt* ist. Wir stellen nun fest, daß jede kontextfreie Sprache eine Grammatik hat, die sich für Bottom-up Parsing eignet:

Lemma 1

Es gibt einen Algorithmus, der jede k.f.G. G in eine k.f.G. G' transformiert, wo $L(G') = L(G)$, und wo G' die folgenden Eigenschaften hat:

1) für jedes Wort außer ε gibt es Ableitungen, in denen Regeln mit leeren rechten Seiten nicht angewendet werden.
2) Falls $\varepsilon \in L(G)$ hat G' die Regel $S \rightarrow \varepsilon$.

Beweis

Der Algorithmus läuft folgendermaßen:

Initialisierung: $G' = G$.

1. Finde 2 Regeln der Typen $A \rightarrow \varepsilon$ bzw. $B \rightarrow uAv$ aus R', wo $B \rightarrow uv$ *nicht* in R' ist.

2. Wenn kein Regelpaar in 1) gefunden wurde, sind wir fertig. Sonst, $R' := R' \cup \{B \rightarrow uv\}$, und Punkt 1) wird wiederholt.

Der Algorithmus terminiert, weil jede neue Regel auf der rechten Seite eine Zeichenkette hat, die durch Entfernung von einem Zeichen aus einer schon vorhandenen Regel gebildet wurde – der Algorithmus terminiert spätestens, wenn jede mögliche solche Zeichenkette gebildet worden ist (und das sind endlich viele).

Der Algorithmus ist korrekt, weil jede G-Ableitung, in der eine ε-Regel verwendet wird, folgendermaßen aussieht (für $w \neq \varepsilon$):

$S \overset{*}{\underset{G}{\Rightarrow}} xBy \underset{G}{\Rightarrow} xuAvy \underset{G}{\Rightarrow} xuvy \overset{*}{\underset{G}{\Rightarrow}} w$

In G' wird es die Regel $B \rightarrow uv$ geben, also können wir auf die ε-Regel verzichten. Für den Fall $w = \varepsilon$ ist es klar, daß G' die Regel $S \rightarrow \varepsilon$ haben wird.

Umgekehrt ist auch jede G'-Ableitung durch eine G-Ableitung ersetzbar – nichts neues wird generiert.❑

Lemma 2

Es gibt einen Algorithmus, der jede k.f.G. G in eine k.f.G. G' transformiert, wo $L(G') = L(G)$, und wo G' die folgende Eigenschaft hat:

Für jedes Wort gibt es Ableitungen, in denen Regeln vom Typ $A \rightarrow B$, wo $A, B \in V$–Σ, nicht angewandt werden.

Beweis

Erstens, es ist für $A, B \in V$–Σ entscheidbar ob $A \overset{+}{\underset{G}{\Rightarrow}} B$, weil solche Ableitungen immer in $\leq |V$–Σ| Schritten durchführbar sind.

Wir wenden dann den folgenden einfachen Algorithmus an:

Für jede Regel $A \underset{G'}{\rightarrow} u$:
Für jedes B so daß $B \overset{*}{\underset{G'}{\Rightarrow}} A$:
Führe $B \underset{G'}{\rightarrow} u$ als neue Regel ein. ❑

Es ist klar, daß wir die oben erzeugten Grammatiken wieder "abspecken" können, indem wir die $A \rightarrow \varepsilon$ -Regeln (außer $S \rightarrow \varepsilon$) und die $A \rightarrow B$ -Regeln entfernen.

Bottom-up Parsing vs. Top-down Parsing

Jetzt wissen wir daß der Bottom-up-Kellerautomat *korrekt* ist. Außerdem wissen wir, daß es zu jeder kontextfreien Grammatik möglich ist, eine äquivalente nicht-zirkuläre, ε-freie (außer eventuell $S \rightarrow \varepsilon$, man kann dies als Sonderfall behandeln) Grammatik zu finden. Also ist das Bottom-up Verfahren für jede Grammatik anwendbar.

Tatsächlich hat das Bottom-up Verfahren einen kleinen Vorteil: Wenn ein G' aus G durch das Entfernen von ε-Regeln und Zirkularität entstanden ist, kann man von einem G'-Ableitungsbaum immer einen G-Ableitungsbaum rekonstruieren. Für Grammatiken, die linksrekursionsfrei gemacht worden sind, ist dies nicht immer möglich (wird hier nicht gezeigt).

Bessere Parsingalgorithmen

Die generellen Top-down- und Bottom-up-Verfahren haben beide exponentielle Komplexität (das heißt, daß die Parsingzeit c^n beträgt, wo n die Eingabelänge ist, und c irgendeine Konstante).

In natürlich-sprachlichen kontextfreien Parsern werden oft allgemein einsetzbare Algorithmen mit Komplexität cn^3 (oder noch besser), wie z.B. *Earleys* Algorithmus, eingesetzt (Aber Prologs DCG ist ein reines Top-down-Verfahren!). Für Programmiersprachen werden meistens deterministische Bottom-up-Parser mit linearer Komplexität benutzt.

Kontextfreie Grammatiken für natürliche Sprachen

Bei Grammatiken für natürliche Sprachen teilt man die Nichtterminalsymbole oft in *nichtlexikalische Kategorien* und *Präterminale* oder *lexikalische Kategorien* auf. D.h., man begrenzt sich auf Regeln, die entweder nur Nichtterminalsymbole enthalten (oft nur die *Regeln* genannt) oder ein einziges Terminalsymbol auf der rechten Seite (diese Teilmenge der Regeln nennt man das *Lexikon*).

Beispiel: $G = (V, \Sigma, R, S)$, wo:

$V = \{S, NP, VP, PP, N, Nprop, V, P, Det\} \cup \Sigma$.

$\Sigma = \{John, the, man, robot, hill, telescope, sees, with, \ldots\}$

und R enthält die folgenden Regeln

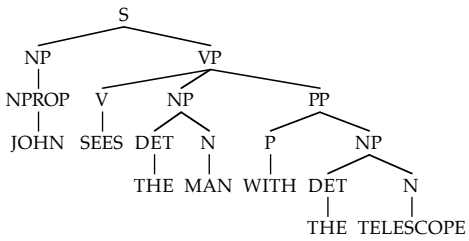
$S \rightarrow NP\ VP$
$VP \rightarrow V$
$VP \rightarrow V\ PP$
$VP \rightarrow V\ NP$
$VP \rightarrow V\ NP\ PP$
} kürzer: $VP \rightarrow V\ (NP)\ (PP)$

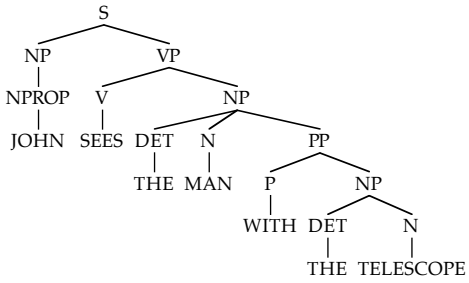
$PP \rightarrow P\ NP$

$NP \rightarrow Nprop$
$NP \rightarrow Det\ N$
$NP \rightarrow Det\ N\ PP$
} kürzer: $NP \rightarrow \{Nprop \mid Det\ N\ (PP)\}$

$Nprop \rightarrow John$,
$N \rightarrow man$, $N \rightarrow robot$, $N \rightarrow hill$, $N \rightarrow telescope$,
$V \rightarrow sees$, $P \rightarrow with$, $Det \rightarrow the$.

Beispiel: Zwei Ableitungsbäume für *John sees the man with the telescope*:





Hier hat Mehrdeutigkeit in der Grammatik einen Sinn!

Ein weiteres entscheidbares Problem

Wir wissen jetzt, daß die Frage "Ist w in $L(G)$" für kontextfreie G entscheidbar ist. Hier kommt noch ein entscheidbares Problem:

Theorem

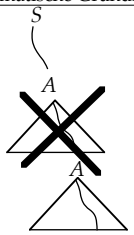
Es gibt einen Algorithmus, der für jede G die Frage

Ist $L(G) = \varnothing$?

beantwortet.

Beweis

Wenn $L(G) \neq \varnothing$ gibt es einen Ableitungsbaum mit Höhe $\leq |V - \Sigma|$, und ein Wort in der Sprache als Ertrag, weil höhere Bäume einen Pfad haben, in dem das selbe Nichtterminalsymbol mehr als einmal vorkommt. Das Teilbäumchen das zwischen diesen beiden Vorkommen erzeugt wird, läßt sich entfernen, und der Ertrag bleibt ein Wort der Sprache:



Der Algorithmus ist dann einfach:

Generiere jeden Ableitungsbaum der Grammatik, der nicht höher als $|V-\Sigma|$ ist, und überprüfe, ob dessen Ertrag ein Wort über Σ ist. ❑

Unentscheidbare Probleme der kontextfreien Sprachen

Es gibt zu den kontextfreien Sprachen erstaunlich viele Probleme, für die es keinen allgemeinen Entscheidungsalgorithmus gibt:

Theorem

Die folgenden Probleme sind unentscheidbar (für L kontextfreie Sprache, G, G_1, G_2 Grammatiken, R reguläre Sprache):

– Ist $L(G) = \Sigma^*$?
– Ist $L(G_1) = L(G_2)$?
– Ist $L(G_1) \subseteq L(G_2)$?
– Ist $L(G) = R$?
– Ist $R \subseteq L(G)$?

– Ist $L(G)$ regulär?
– Ist L mehrdeutig?

– Ist $\Sigma^* - L(G)$ kontextfrei?
– Ist $L(G_1) \cap L(G_2)$ kontextfrei?

Beweis

Nicht durchgeführt, da er viel Turingmaschinen-Theorie voraussetzt.

Deterministische Kellerautomaten

Ein *KA* heißt *deterministisch* (DKA) wenn er zu jeder Konfiguration *höchstens* (nicht *genau*, wie bei DEA) eine mögliche Transition hat.

Für die formale Definition der DKA brauchen wir einige Hilfsdefinitionen:

Zwei Zeichenketten sind *konsistent* gdw die eine ein Präfix der anderen ist.

(Also ist w konsistent mit wu, und ε ist mit jeder Zeichenkette konsistent)

Zwei Transitionen

$((p, w_1, \gamma_1), (q_1, \delta_1))$ und $((p, w_2, \gamma_2), (q_2, \delta_2))$

sind *kompatibel* wenn w_1 und w_2 konsistent sind und γ_1 und γ_2 konsistent sind.

Definition

Ein KA M ist *deterministisch* gdw M keine zwei unterschiedlichen Transitionen hat, die kompatibel sind.

Deterministisch kontextfreie Sprachen

Eine kontextfreie Sprache L heißt *deterministisch*

gdw

$L\$ = L(M)$ für einen DKA M (wo \$ ein neues Zeichen ist)

Die Definition stammt von Lewis & Papadimitriou. Eine andere mögliche Definition ist:

Eine kontextfreie Sprache L heißt deterministisch gdw $L = L_f(M)$ für einen DKA M.

Hier ist $L_f(M)$ die Sprache, die M im Endzustand (aber nicht notwendigerweise mit leerem Keller) akzeptiert (siehe Übungsaufgaben!).

(Diese Def. wird von Hopcroft & Ullman benutzt; ihr $L(M)$ (für KA im Allgemeinen) ist unser $L_f(M)$.)

Der Grund für die kleine Komplikation bei der Definition ist, daß man gerne Sprachen wie

$a^* \cup \{a^n b^n : n \geq 0\}$

als deterministisch bezeichnen möchte, ein DKA nach unserer Definition kann aber nicht bei Eingabe *aaa* den Keller leeren, bevor die ganze Eingabe gesehen wurde, weil ja die Eingabe auch *aaabbb* hätte sein können, und dann ist es "zu spät". Deshalb führen wir die Eingabe-Ende-Markierung \$ ein.

Theorem

Die regulären Sprachen bilden eine echte Teilmenge der deterministisch kontextfreien Sprachen.

Beweis

Jeder DEA kann als ein DKA betrachtet werden, der seinen Keller nicht benutzt. Auf der anderen Seite gibt es Sprachen wie $\{a^n b^n : n \geq 0\}$, die deterministisch, aber nicht regulär sind.❑

Theorem

Die deterministisch kontextfreien Sprachen sind unter *Komplementbildung* abgeschlossen.

Beweis-Skizze

Intuitiv kann man wie bei den DEAs einfach einen Automaten konstruieren, in dem Endzustände und Nicht-Endzustände umgetauscht wurden. Folgende mögliche Eigenschaften des DKAs macht dies nicht ohne weiteres möglich: 1: Zustände, von denen aus es keine gültigen Übergänge mehr gibt. 2: Schleifen, wo keine Eingabe gelesen wird. 3: ε-Bewegungen zwischen End- und nicht-Endzuständen.
Wir werden hier nicht beweisen, daß diese Probleme sich überwinden lassen.

Theorem

Die deterministisch kontextfreien Sprachen bilden eine echte Teilmenge der kontextfreien Sprachen.

Beweis

Jede deterministisch kontextfreie Sprache ist auch kontextfrei. Aber nicht jede kontextfreie Sprache ist deterministisch kontextfrei, es gibt ja kontextfreie Sprachen mit einem Komplement das nicht kontextfrei – und dadurch auch nicht deterministisch kontextfrei – ist.❑

Die letzte Aussage läßt sich auch konstruktiv bestätigen, wir werden folgendes zeigen:

Lemma

Das Komplement der Sprache $L_{abc} = \{a^n b^n c^n : n \geq 0\}$ ist kontextfrei.

Beweis

$\Sigma^* - L_{abc} = (\Sigma^* - a^*b^*c^*) \cup (a^*b^*c^* - L_{abc})$

$\Sigma^* - a^*b^*c^*$ ist regulär.

$a^*b^*c^* - L_{abc} =$
$\{a^m b^n c^p : m>n\} \cup \{a^m b^n c^p : n>m\} \cup$
$\{a^m b^n c^p : n>p\} \cup \{a^m b^n c^p : p>n\} \cup$
$\{a^m b^n c^p : m>p\} \cup \{a^m b^n c^p : p>m\} =$

$aa^*\{a^n b^n : n \geq 0\}c^* \cup \{a^n b^n : n \geq 0\}bb^*c^* \cup$
$a^*bb^*\{b^n c^n : n \geq 0\} \cup a^*\{b^n c^n : n \geq 0\}cc^* \cup$
$aa^*\{a^n b^m c^n : n,m \geq 0\} \cup \{a^n b^m c^n : n,m \geq 0\}cc^*.$

(die letzten zwei lassen sich einfach von kontextfreien Grammatiken erzeugen). ❑

Theorem

Die deterministisch kontextfreien Sprachen sind unter Vereinigung nicht abgeschlossen.

Beweis

Im vorigen Beweis haben wir die nicht-deterministische Sprache $\Sigma^* - L_{abc}$ als eine endliche Vereinigung von deterministischen Sprachen dargestellt: $\Sigma^* - a^*b^*c^*$ ist regulär, und damit det.k.f.; für die anderen Sprachen lassen sich leicht deterministische Automaten konstruieren.❑

Theorem

Die deterministisch kontextfreien Sprachen sind unter Schnittbildung nicht abgeschlossen.

Beweis

$L_1 \cup L_2 = \Sigma^* - ((\Sigma^* - L_1) \cap (\Sigma^* - L_2))$,

und die det.k.f. Sprachen sind abgeschlossen unter Komplementbildung, d.h.: Wären sie unter Schnittbildung abgeschlossen, wären sie auch unter Vereinigung abgeschlossen.❑

Theorem

Die deterministisch kontextfreien Sprachen sind unter Schnittbildung und Vereinigung mit regulären Sprachen abgeschlossen.

Beweis

Der Beweis, daß die kontextfreien Sprachen unter Schnittbildung mit regulären Sprachen abgeschlossen sind, läßt sich auch ohne sonstige Änderung mit einem DKA statt einem KA durchführen.

Weil $L \cup R = \Sigma^* - ((\Sigma^* - L) \cap (\Sigma^* - R))$ sind die deterministisch kontextfreien Sprachen auch unter Vereinigung mit regulären Mengen abgeschlossen.❑

Theorem

Die deterministisch kontextfreien Sprachen sind unter Konkatenation und Kleene'scher Hüllenbildung nicht abgeschlossen.

Beweis

$L_1 = \{a^m b^n c^p : m=n\}$ und $L_2 = \{a^m b^n c^p : n=p\}$ sind beide deterministisch kontextfrei, aber $L = L_1 \cup L_2$ ist es nicht, weil das Komplement von L nicht kontextfrei ist. Dies läßt sich folgendermaßen zeigen: Nehmen wir an, daß $\Sigma^* - L$ kontextfrei ist. Dann muß aber $(\Sigma^* - L) \cap a^*b^*c^*$ auch kontextfrei sein. Die letzte Sprache ist aber $\{a^m b^n c^p : m \neq n \text{ und } n \neq p\}$. Mit einer starken Version des Pumping-Lemmas (das Ogden-Lemma) läßt sich zeigen, daß diese Sprache nicht kontextfrei ist.

Jetzt betrachten wir $L_3 = dL_1 \cup L_2$. Diese Sprache ist deterministisch: Ein Automat kann nach dem Lesen vom ersten Zeichen schon entscheiden, ob die Eingabe zu L_1 oder zu L_2 gehört.

Jetzt bilden wir die Konkatenation von den beiden deterministisch kontextfreien Sprachen d^* und L_3. Die dadurch entstandene Sprache ist *nicht* deterministisch kontextfrei, weil sonst auch

$$L_4 = d^*L_3 \cap da^*b^*c^* = dL_1 \cup dL_2 = dL$$

deterministisch kontextfrei wäre, was offensichtlich nicht der Fall sein kann.

Für Kleene'sche Hüllenbildung betrachten wir die Sprache $L_5 = d \cup L_3$. L_5 ist eine deterministisch kontextfreie Sprache, weil L_3 es ist.

Aber L_5^* kann nicht deterministisch sein, weil

$L_5^* \cap da^*b^*c^* = dL_1 \cup dL_2 = dL$.❑

Prädiktiver Parser mit Lookahead

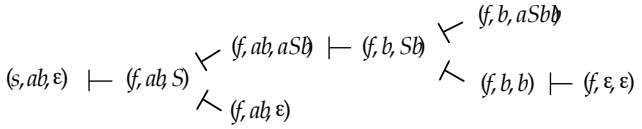
Wir betrachten wieder die Grammatik
$G = (\{a, b, S\}, \{a, b\}, \{S \rightarrow aSb, S \rightarrow \varepsilon\}, S)$,
die $L = \{a^n b^n : n \geq 0\}$ erzeugt.

Hier der Top-down-KA zu G wiederholt:

$M = (\{s, f\}, \{a, b\}, \{S, a, b\}, \Delta, s, \{f\})$,
wo $\Delta = \{ ((s, \varepsilon, \varepsilon), (f, S)),$
$((f, \varepsilon, S), (f, aSb)),$
$((f, \varepsilon, S), (f, \varepsilon)),$
$((f, a, a), (f, \varepsilon)),$
$((f, b, b), (f, \varepsilon)) \}$

M ist nicht deterministisch, L ist aber eine deterministisch kontextfreie Sprache.

Wir betrachten wieder die möglichen Berechnungen bei Eingabe *ab*:



Wir sehen, daß der Automat immer die richtige Regel hätte wählen können, wenn er Information über das nächste Zeichen der Eingabe hätte.

Lookahead

Wir konstruieren aus M einen DKA M' mit *Lookahead*: M' liest immer das nächste Zeichen, um zwischen zwei anwendbaren Regeln unterscheiden zu können. Das gelesene Zeichen wird als Zustand gespeichert:

$M' = (\{s, q, q_a, q_b, q_\$\}, \{a, b, \$\}, \{S, a, b\}, \Delta', s, \{q_\$\})$, wo

$\Delta = \{$ $((s, \varepsilon, \varepsilon), (q, S))$, (Start)
$((q, a, \varepsilon), (q_a, \varepsilon))$, (Lese: *a*)
$((q_a, \varepsilon, a), (q, \varepsilon))$, (*a* generiert, weiterlesen)
$((q, b, \varepsilon), (q_b, \varepsilon))$, (Lese: *b*)
$((q_b, \varepsilon, b), (q, \varepsilon))$, (*b* generiert, weiterlesen)
$((q, \$, \varepsilon), (q_\$, \varepsilon))$, (Lese: $ – halt!)
$((q_a, \varepsilon, S), (q_a, aSb))$, (Nur $S \to aSb$ möglich!)
$((q_b, \varepsilon, S), (q_b, \varepsilon))$ $\}$ (Nur $S \to \varepsilon$ möglich!)

In den Zuständen q_a und q_b wird überprüft, ob das entsprechende Zeichen schon mit einer Regel erzeugt wurde. Wenn dies zutrifft, wird das Zeichen vom Keller weggenommen und ein neues Zeichen wird gelesen. Wenn ein Terminalzeichen noch nicht erzeugt worden ist, werden die anwendbaren Regeln ausprobiert (in diesem Fall nur eine Regel je Zustand).

Wir schauen uns jetzt an, wie der Automat die Eingabe *aabb*$ akzeptiert.

Zustand	Unverbrauchte Eingabe	Keller	Regel, die angewandt wurde bzw. Kommentar
s	*aabb$*	ε	*Start*
q	*aabb$*	S	
q_a	*abb$*	S	*a gelesen (Lookahead)*
q_a	*abb$*	*aSb*	$S \to aSb$ (1)
q	*abb$*	*Sb*	*a korrekt*
q_a	*bb$*	*Sb*	*a gelesen (Lookahead)*
q_a	*bb$*	*aSbb*	$S \to aSb$ (2)
q	*bb$*	*Sbb*	*a korrekt*
q_b	*b$*	*Sbb*	*b gelesen (Lookahead)*
q_b	*b$*	*bb*	$S \to \varepsilon$ (3)
q	*b$*	*b*	*b korrekt*
q_b	*$*	*b*	*b gelesen (Lookahead)*
q	*$*	ε	*b korrekt*
$q_\$$	ε	ε	*$ gelesen, akzeptiert!*

Bei Eingabe *aab$* hält der Automat mit *b* auf dem Keller.

Bei Eingabe *aabbb$* bleibt der Automat in q_b hängen.

Linksfaktorisieren

Wir betrachten jetzt wieder die *LISP*-Grammatik $G = (\{S, L, (,), a, \_\}, \{(,), a, \_\}, R, S)$ wo

$R = \{ S \to (), \quad S \to a \quad S \to (L)$
$\quad L \to S, \quad L \to L\_S \}$

Es gibt zwei S-Regeln, die mit demselben Symbol anfangen, $S \to ()$ und $S \to (L)$.

Dies heißt, daß ein Automat mit einem Lookahead von nur einem Zeichen nicht zwischen diesen beiden Regeln entscheiden kann. Wir können aber die beiden Regeln mit den folgenden drei ersetzen:

$S \to (S',\ S' \to L)$ und $S' \to)$

Wir wenden folgende heuristische Methode an:

Linksfaktorisieren:

Wenn $A \to \alpha\beta$ und $A \to \alpha\gamma$ Regeln sind, wo $\alpha \neq \varepsilon$, dann ersetze sie mit

$A \to \alpha A'$, $A' \to \beta$, und $A' \to \gamma$.

Theorem

Linksfaktorisieren ändert nicht die generierte Sprache.

Beweis: Einleuchtend!❑

LL(k)-Grammatiken

Eine Grammatik heißt *LL*(*k*) wenn ein *Lookahead* von *k* Symbolen genügt, um, für jedes *A*, immer nur eine von den ganzen *A*-Regeln wählen zu können.

Durch entfernen von Linksrekursion und Links-Faktorisieren lassen sich viele Grammatiken in *LL*(1)-Grammatiken umwandeln. Aus der *LISP*-Grammatik können wir folgende *LL*(1)-Grammatik ableiten:

$S \to (S' \quad S' \to L) \quad L \to SL'$
$S \to a \quad S' \to) \quad L' \to \_SL'$
$\quad L' \to \varepsilon$

Der entsprechende Automat ist deterministisch und hat die folgenden 22 (!) Übergänge:

$((s, \varepsilon, \varepsilon), (q, S))$
$((q, \sigma, \varepsilon), (q_\sigma, \varepsilon))$ – für $\sigma =$ **(**, **)**, _, *a*, $ (lookahead)
$((q_\sigma, \varepsilon, \sigma), (q, \varepsilon))$ – für $\sigma =$ **(**, **)**, _, *a*

$((q_($, ε, *S*), $(q_($, **(***S'*)**)$ $(S \to$ **(***S'*)
$((q_a, \varepsilon, S), (q_a, a))$ $(S \to a)$
$((q_\sigma, \varepsilon, S'), (q_\sigma, L))$ – für $\sigma =$ **(**, *a* $(S' \to L)$
$((q_)$, ε, *S'*), $(q_)$, **)**)) $(S' \to$ **)**)
$((q_\sigma, \varepsilon, L), (q_\sigma, SL'))$ – für $\sigma =$ **(**, *a* $(L \to SL')$
$((q_\_, \varepsilon, L'), (q_\_, \_SL'))$ $(L' \to \_SL')$
$((q_\sigma, \varepsilon, L'), (q_\sigma, \varepsilon))$ – für $\sigma =$ **(**, **)**, *a*, $ $(L' \to \varepsilon)$

Z.B. gibt es zu der Regel $L \to SL'$ eine Transition aus q_a und eine aus $q_($, weil *a* und **(** die möglichen Anfangssymbole von *S* sind.

Parsen des S-Expressions **(()_*a*)**:

Zustand	Rest-eingabe	Keller	Regel, die angewandt wurde/Kommentar
s	**(()**_a)$	ε	*Start*
q	**(()**_a)$	S	
$q_($	**()**_a)$	S	**(** *gelesen (Lookahead)*
$q_($	**()**_a)$	**(**S'	$S \rightarrow$ **(**S'
q	**()**_a)$	S'	**(** *korrekt*
$q_($	**)**_a)$	S'	**(** *gelesen (Lookahead)*
$q_($	**)**_a)$	L**)**	$S' \rightarrow L$**)**
$q_($	**)**_a)$	SL'**)**	$L \rightarrow SL'$
$q_($	**)**_a)$	**(**$S'L'$**)**	$S \rightarrow$ **(**S'
q	**)**_a)$	$S'L'$**)**	**(** *korrekt*
$q_)$	_a)$	$S'L'$**)**	**)** *gelesen (Lookahead)*
$q_)$	_a)$	**)**L'**)**	$S' \rightarrow$ **)**
q	_a)$	L'**)**	**)** *korrekt*
$q_\_$	a)$	L'**)**	_ *gelesen (Lookahead)*
$q_\_$	a)$	_SL'**)**	$L' \rightarrow$ _SL'
q	a)$	SL'**)**	_ *korrekt*
q_a	)$	SL'**)**	*a gelesen (Lookahead)*
q_a	)$	aL'**)**	$S \rightarrow a$
q	)$	L'**)**	*a korrekt*
$q_)$	$	L'**)**	**)** *gelesen (Lookahead)*
$q_)$	$	**)**	$L' \rightarrow \varepsilon$
q	$	ε	**)** *korrekt*
$q_\$$	ε	ε	$ *gelesen, akzeptiert!*

Die Hierarchie der *LL*-Sprachen

Es fragt sich jetzt: Gibt es zu jeder deterministisch kontextfreien Sprache eine $LL(k)$-Grammatik (für irgendein k)? Die Antwort ist *nein*, es kann sogar gezeigt werden, daß die Klasse der Sprachen, die von $LL(k+1)$-Grammatiken generiert werden, eine echte Obermenge der Sprachen ist, die von $LL(k)$-Grammatiken generiert werden.

Die Chomsky-Hierarchie

Durch Verallgemeinerungen der Regeln unserer kontextfreien Grammatiken können wir viel größere Klassen von Sprachen beschreiben. Wir werden also Grammatiken immer noch als $G = (V, \Sigma, R, S)$ darstellen, und die generierte Sprache ist wie gewohnt $L(G) = \{w \in \Sigma^*: S \overset{*}{\underset{G}{\Rightarrow}} w\}$, nur verlangen wir nicht mehr, daß die linke Seite der Regeln aus einem einzigen Nichtterminalsymbol besteht:

Typ-0-Grammatiken

Typ-0-Grammatiken haben Regeln

$\alpha \rightarrow \beta$, wo $\alpha \in V^+$, und $\beta \in V^*$,

Die von Typ-0-Grammatiken generierten Sprachen nennen wir Typ-0-Sprachen oder *rekursiv aufzählbare Sprachen*.

Typ-1- oder *kontextsensitive* Grammatiken

sind wie Typ-0-Grammatiken, aber mit der Einschränkung daß $|\alpha| \le |\beta|$. Es gibt für Typ-1-Grammatiken folgende Normalform:

$\alpha A \gamma \rightarrow \alpha\beta\gamma$ (wo A ein Nichtterminalsymbol ist)

Solche Regeln sind fast wie die kontextfreien Regeln, erlauben aber nur die Substitution von A in einem *Kontext*, daher die Bezeichnung *kontextsensitiv*.

Typ-2- und Typ-3-Grammatiken

Diese kennen wir schon: Es sind die kontextfreien bzw. regulären Grammatiken.

Typ-0-Grammatik: Beispiel

Die folgende Grammatik $G = (V, \Sigma, R, S)$ generiert die Sprache $L(G) = \{a^{2^n}: n \ge 0\}$:

$V = \{[,], D, S, a\}$

$\Sigma = \{a\}$

$R = \{ \; S \rightarrow [a],$
$\quad [\rightarrow [D,$
$\quad Da \rightarrow aaD,$ (diese Regel ist auch Typ-1)
$\quad D] \rightarrow],$ (eine echte Typ-0-Regel)
$\quad [\rightarrow \varepsilon,$ (diese Regeln sind auch kontextfrei,
$\quad] \rightarrow \varepsilon \; \}$ aber nicht Typ-1)

Beiliebig viele D's (D für duplizieren!) können links eingefügt werden. Sie können nach rechts durch die schon erzeugten a's "wandern", und auf dem Weg werden aus jedem a zwei gemacht. Ein D kann erst dann wieder gelöscht werden, wenn es das]-Zeichen rechts erreicht hat.

Beispiel: Ableitung von *aaaa*:

$S \Rightarrow [a] \Rightarrow [Da] \Rightarrow [aaD] \Rightarrow [aa] \Rightarrow [Daa] \Rightarrow [aaDa] \Rightarrow [aaaaD] \Rightarrow [aaaa] \Rightarrow aaaa] \Rightarrow aaaa$

Theorem

Die Typ-0-Sprachen bilden eine echte Obermenge der kontextfreien Sprachen.

Beweis

Die gerade vorgestellte Sprache ist nicht kontextfrei:

Annahme: $\{a^{2^n}: n \ge 0\}$ sei kontextfrei. Dann gibt es eine k.f. Grammatik G mit $L(G) = \{a^{2^n}: n \ge 0\}$. Das Pumping-Lemma (in der schwächeren Version) sagt uns, daß es ein K gibt, so daß für $|w| > K$ Wörter u, v, x, y, z gibt, wo $w = uvxyz$, $vy \ne e$ und für jedes $n \ge 0$: $uv^nxy^nz \in L(G)$.

Z.B. ist dann $uv^2xy^2z \in L(G)$. Es gilt: $|w| < |uv^2xy^2z| \le 2|w|$ und $w = a^{2^k}$ für ein k. Die nächste größere Zeichenkette in $L(G)$ nach w ist $a^{2^{k+1}}$, und $|a^{2^{k+1}}| = 2|w|$. Also muß $vy = w$. Aber dann ist $|uv^3xy^3z| = 3|w|$. Dies ergibt eine Kontradiktion, weil die nächste größere Zeichenkette in $L(G)$ $a^{2^{k+2}}$ ist, und $|a^{2^{k+2}}| = 4|w|$. q

Die Sprache ist aber auch mit einer kontextsensitiven Grammatik generierbar (wird nicht gezeigt). Tatsächlich ist es nicht so leicht Sprachen zu finden, die von keiner kontextsensitiven Grammatik generiert werden, es sei denn man nimmt dann eine kontextfreie Sprache, die ε enthält. Regeln wie $A \rightarrow \varepsilon$ sind nach der Definition einer kontextsensitiven Grammatik nicht zugelassen.

Theorem

Die kontextsensitiven Sprachen bilden eine echte Obermenge der ε-freien kontextfreien Sprachen

Beweis

Wir betrachten die Sprache $\{a^n b^n c^n : n \geq 1\}$, die ja nicht kontextfrei ist.

Die Sprache wird von einer kontextsensitiven Grammatik mit den folgenden Regeln generiert:

$S \rightarrow abc$ $S \rightarrow aSBc$

$cB \rightarrow Bc$ $bB \rightarrow bb$.

Z.B. ist *aaabbbccc* in $L(G)$:

$S \Rightarrow aSBc \Rightarrow aaSBcBc \Rightarrow aaabcBcBc \Rightarrow aaabBccBc \Rightarrow aaabbccBc \Rightarrow aaabbcBcc \Rightarrow aaabbBccc \Rightarrow aaabbbccc$. ❑

Die rekursiven Sprachen

Definition

Eine Sprache L heißt *rekursiv* wenn es einen Algorithmus gibt, der für eine gegebene Zeichenkette $w \in \Sigma^*$ entscheiden kann, ob $w \in L$.

Wir wissen schon, daß die kontextfreien Sprachen rekursiv sind. Es gilt auch:

Theorem

Jede kontextsensitive Sprache ist rekursiv.

Beweis

Sei $G = (V, \Sigma, R, S)$ eine kontextsensitive Grammatik, und sei $w \in \Sigma^*$, $|w| = n$. Wir bilden jetzt die Menge $V(n)$ der Zeichenketten über V (mit oder ohne Nichtterminale), die höchstens n Zeichen haben (diese Menge hat $(|V|^{n+1} - 1)/(|V| - 1)$ Elemente…). Wir bilden jetzt einen Graphen mit den Elementen aus $V(n)$ als Knoten, und wo es von u nach v eine Kante gibt wenn v von u in G direkt ableitbar ist.

Die Pfade im Graphen entsprechen Ableitungen in G, und zwar allen möglichen, die eine Zeichenkette der Länge n erzeugen (weil $|\alpha| \leq |\beta|$ wenn $\alpha \Rightarrow \beta$). Also ist w genau dann in $L(G)$ wenn es im Graphen einen Pfad von S nach w gibt (wir werden hier aber keinen Pfad-Suche-Algorithmus vorstellen). ❑

Korrolar

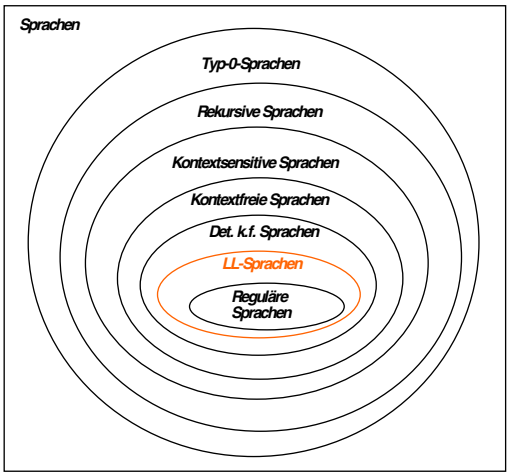
Es gibt Parsingalgorithmen für die kontextsenstiven Grammatiken.

Beweis

Der im vorigen Beweis vorgestellte Algorithmus erzeugt jede mögliche Ableitung eines Wortes in einer kontextsensitiven Grammatik. ❑

Übersicht der Sprachklassen

Wir können uns jetzt das folgende Bild der Sprachklassen, die wir kennengelernt haben, bilden:



Wir wissen schon, daß einige dieser Inklusionen *echte* Inklusionen sind. Es gibt nicht-reguläre deterministisch kontextfreie Sprachen (wie $\{a^n b^n : n \geq 0\}$), es gibt nicht-deterministische kontextfreie Sprachen (wie $\Sigma^* - \{a^n b^n c^n : n \geq 0\}$), und es gibt nicht-kontextfreie kontextsensitive Sprachen (wie $\{a^n b^n c^n : n \geq 0\}$).

Tatsächlich ist *jede* der Inklusionen echt – wir lassen aber den Beweis, daß es nicht-kontextsensitive rekursive Sprachen gibt, aus.

Um zu zeigen, daß es nicht-rekursive Typ-0-Sprachen gibt, brauchen wir *einen Automaten*, der Typ-0-Sprachen akzeptiert, und ein formales Modell eines Algorithmus. Tatsächlich können wir für beide Zwecke denselben Typ von Automaten – *die Turing-Maschinen* – benutzen. Und bis jetzt hat niemand Algorithmen gefunden, die außerhalb des Turing-Maschinen-Modells fallen. Dies gibt eine gute Unterstützung für:

Die Church'sche Hypothese

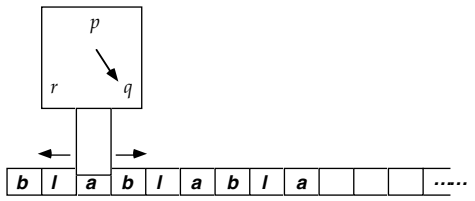
Alles, was berechenbar ist, läßt sich von einer Turing-Maschine berechnen.

Dies ist und bleibt eine *Hypothese*, solange *berechenbar* nicht formalisiert werden kann.

Was aber bewiesen werden kann ist, daß andere Algorithmenmodelle mit Turingmaschinen äquivalent sind. Z.B. sind die Typ-0-Grammatiken mit Turingmaschinen äquivalent, und auch die *rekursiven Funktionen*, eine mathematische Formalisierung der Berechenbarkeit.

Turing-Maschinen

Kurz gesagt hat eine Turing-Maschine wie ein endlicher Automat eine endliche Kontrolle, hat aber ein unendlich langes Eingabeband (ein Teil davon ist die *endliche* Eingabe), auf dem er auch schreiben darf. Außerdem darf sich der Automat auf der Eingabe auch nach *links* bewegen:



Turing-Maschinen brauchen keine Endzustände; wie bei den endlichen Transducern schreibt die Maschine das Ergebnis ihrer Berechnung auf das Band.

Es gibt aber einen besonderen *Haltezustand*, der signalisiert, daß die Berechnung beendet ist. Dieser Zustand ist derselbe für jede Maschine, und wird mit *h* bezeichnet.

Das Band ist unendlich nach *rechts*. Wenn die Maschine sich links vom am weitesten links stehenden Zeichen bewegt, hört die Berechnung auf, aber wir sagen dann nicht, daß *M* hält. Die Eingabe ist auf einem endlichen Teil des Bandes geschrieben. Sonst ist das Band mit dem besonderen Zeichen 'Blank' (#) gefüllt.

Turing-Maschinen formal definiert

Eine Turingmaschine M (= (K, Σ, δ, s)) besteht aus:

- K – Eine Menge von *Zuständen*, wo $h \notin K$
- Σ – Ein Alphabet, wo $\# \in \Sigma$ aber $L, R \notin \Sigma$.
- s – Ein Startzustand ($s \in K$)
- δ – Eine Funktion von $K \times \Sigma$ in $(K \cup \{h\}) \times (\Sigma \cup \{L, R\})$

$\delta(q, a) = (p, b)$ bedeutet, daß die Maschine wenn sie im Zustand q ist und ein a sieht, in den Zustand p übergehen soll und außerdem:

- wenn b L oder R ist, sich in die entsprechende Richtung auf der Band bewegt
- wenn $b \in \Sigma$ ist, auf der Eingabe ein b schreibt (das a wird überschrieben).

M ist deterministisch, weil δ eine totale Funktion ist. Die Berechnung läuft bis M in h ist (*M hält*), oder M sich links von dem am weitesten links stehenden Zeichen bewegt (*M hängt*)

Beispiel

Sei M die Maschine (K, Σ, δ, q), wo
$K = \{q\}$,
$\Sigma = \{a, \#\}$

und: $\delta(q, a) = (q, L)$, $\delta(q, \#) = (h, \#)$.

Diese Maschine bewegt sich nach links bis sie ein # sieht. Wenn links von der (vorläufig beliebigen) Startkonfiguration *kein* # ist, dann *hängt* die Maschine.

Konfigurationen

Eine Konfiguration einer Turingmaschine M,

(p, u, σ, v), besteht aus:

- p – dem aktuellen Zustand
- u – dem Teil des Bandes *links* von der aktuellen Position
- σ – dem in der aktuellen Position gelesenen Zeichen
- v – dem Teil des Bandes *rechts* von der aktuellen Position, aber nur bis einschließlich dem letzten nicht-blanken Zeichen.

Wenn Z.B. eine Turingmaschine M in einem Zustand p ist, das Eingabeband *##aabaa##…* ist, und M sich in der Eingabe am b befindet, dann ist die Konfiguration $(p, \#\#aa, b, aa)$.

Wenn $p=h$, ist die Konfiguration eine *Haltekonfiguration*. Wenn M links vom Band ist, ist sie in eine *Hängekonfiguration*.

Vereinfachte Notation für Konfigurationen

Statt (p, u, σ, v) schreiben wir $(p, u\underline{\sigma}v)$; z.B. schreiben wir $(p, \#\#aa, b, aa)$ einfacher als $(p, \#\#aa\underline{b}aa)$.

Die *ergibt*-Relation für Turing-Maschinen

Wir sparen uns die volle Formalität der Definition, und formulieren es lieber so:

Sei $M = (\ K, \Sigma, \delta, q)$ eine Turing-Maschine, sei $(q, u\underline{a}v)$ eine Konfiguration von M und sei $\delta(q, a) = (p, b)$.

Dann gilt:

Wenn $b \in \Sigma$: $(q, u\underline{a}v) \vdash_M (p, u\underline{b}v)$

Wenn $b = L$:
  Wenn u als $u'c$ geschrieben werden kann gilt:
    Wenn $av \neq \#$: $(q, u\underline{a}v) \vdash_M (p, u'\underline{c}av)$
    Sonst: $(q, u'c\underline{\#}) \vdash_M (p, u'\underline{c})$
  Sonst hängt die Maschine.

Wenn $b = R$:
  Wenn $v = cv'$ für ein $c \in \Sigma$:
    $(q, u\underline{a}v) \vdash_M (p, uac\underline{v}')$
  Sonst ($v = \varepsilon$):
    $(q, u\underline{a}) \vdash_M (p, ua\underline{\#})$

Die ergibt-in-einem-Schritt-Relation ist durch dies vollständig beschrieben.

$\vdash_M^*$ ist wie immer die reflexiv-transitive Hülle von $\vdash_M$.

Eingabe für Turing-Maschinen

Wir werden jetzt die Eingabe einer Turing-Maschine standardisieren: Sei M eine Turingmaschine, und w eine Zeichenkette über dem Alphabet der Maschine:

Definition

Eine Turingmaschine *M hält bei Eingabe w*

gdw

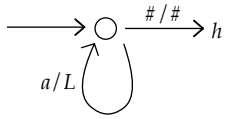
$(s, \#w\underline{\#})$ ergibt eine Haltekonfiguration.

M hängt bei Eingabe w gdw $(s, \#w\underline{\#})$ eine Hängekonfiguration ergibt.

Zustandsdiagramme

Wir können DEA-ähnliche Zustandsdiagramme für Turing-Maschinen einführen. Nehmen wir an, $\delta(q, a) = (p, b)$. Der Pfeil zwischen q und p wird dann mit a/b markiert ($<a, b>$ wird auch manchmal benutzt).

Hier z.B. ein Zustandsdiagramm für die Maschine aus dem ersten Beispiel:



Berechnungen mit Turing-Maschinen

Turing-Berechenbarkeit

Seien Σ_1 und Σ_2 Alphabete, wo $\# \notin \Sigma_1$ und $\# \notin \Sigma_2$, und sei f eine Funktion von Σ_1^* in Σ_2^*.

Eine Turing-Maschine $M = (K, \Sigma, \delta, s)$ *berechnet* f

gdw

Für jedes w in Σ_1^*: $(s, \#w\underline{\#}) \vdash_M^* (h, \#f(w)\underline{\#})$

Wenn es zu f ein M gibt, das f berechnet, heißt f *Turing-berechenbar*.

Erweiterung auf mehrstellige Funktionen:

Eine mehrstellige Funktion f von $(\Sigma_1^*)^n$ (wo $n \geq 0$) in Σ_2^* kann ähnlich berechnet werden; eine solche Funktion wird von M berechnet gdw

Für jedes w in $(\Sigma_1^*)^n$:
$(s, \#w_1\#w_2 \ldots \#w_n\underline{\#}) \vdash_M^* (h, \#f(w_1, w_2, \ldots, w_n)\underline{\#})$

Zahlenfunktionen:

Zahlenfunktionen lassen sich als ein Sonderfall der Zeichenkettenfunktionen behandeln. Wir brauchen nur ein Zahlensystem festzulegen, z.B. können wir *unäre* Repräsentation benutzen (0 als ε, 1 als 1, 2 als 11, 3 als 111 usw.).

Turing-akzeptierbar

Sei Σ ein Alphabet ohne #. M *akzeptiert* $w \in \Sigma^*$ gdw M hält bei Eingabe w.

Sei L eine Sprache über einem Alphabet Σ. M *akzeptiert* L gdw $L = \{w \in \Sigma^*: M \text{ akzeptiert } w\}$.

Sei L eine Sprache. L ist *Turing-akzeptabel* gdw es eine Turing-Maschine gibt, die L akzeptiert.

Turing-entscheidbar

Eine Sprache L über Σ ist Turing-entscheidbar, wenn es eine Turingmaschine gibt, die für jede Zeichenkette w über Σ entscheiden kann, ob $w \in L$ oder $w \notin L$. D.h., die Turingmaschine kann die charakteristische Funktion χ_L der Menge L berechnen. Wir definieren Turing-Entscheidbarkeit deshalb via χ_L: Sei Σ ein Alphabet ohne #, und seien $\mathfrak{m}$ und $\mathfrak{n}$ Symbole, die in Σ nicht vorkommen. Definiere χ_L von Σ^* in $\{\mathfrak{m}, \mathfrak{n}\}$ folgendermaßen:

$$\chi_L(w) = \begin{cases} \mathfrak{m} & \text{wenn } w \in L \\ \mathfrak{n} & \text{wenn } w \notin L \end{cases}$$

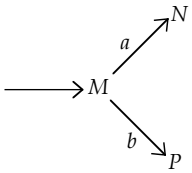
Definition:
L ist Turing-entscheidbar gdw χ_L Turing-berechenbar ist.

M entscheidet L gdw M χ_L berechnet.

Notation für zusammengesetzte Turingmaschinen

Wenn M und N zwei Turingmaschinen sind, ist $M \rightarrow N$ (oder einfach MN) die Turingmaschine, die erst die Berechnung von M durchführt, und falls M halten würde, in den Startzustand von N übergeht.

Wenn M und N Turingmaschinen sind, ist $M \xrightarrow{a} N$ die Maschine, die erst die Berechnung von M durchführt. Falls M mit einem a unter dem Lesekopf halten würde, geht $M \xrightarrow{a} N$ dann in den Startzustand von N über, sonst hält sie.
Entsprechend können wir auch Maschinen bilden, die je nach gelesenem Zeichen in einer von mehreren Maschinen ihre Berechnung fortsetzt, wie:



Basismaschinen

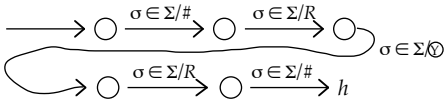
Für die drei möglichen Transitionstypen führen wir Basismaschinen ein:

- L ist für jedes Alphabet Σ die Maschine, die für jedes Zeichen a in Σ die Transition $\delta(s, a) = (h, L)$ hat (und die keine anderen Transitionen hat).
- R ist für jedes Alphabet Σ die Maschine, die für jedes Zeichen a in Σ die Transition $\delta(s, a) = (h, R)$ hat (und keine anderen Transitionen hat).
- σ ist für jedes Alphabet Σ die Maschine, die für jedes Zeichen a in Σ die Transition $\delta(s, a) = (h, \sigma)$ hat (und die keine anderen Transitionen hat).

Beispiel

Die Maschine $\#R\mathfrak{m}R\#$ kommt als Teilmaschine in vielen Turingmaschinen vor.

Wir hätten auch die Maschine folgendermaßen darstellen können:



Die Notation $\#R\mathfrak{m}R\#$ ist etwas gewöhnungsbedürftig, aber sehr platzsparend!

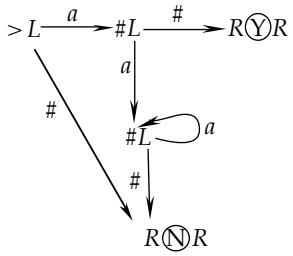
Einige nützliche Teilmaschinen

Eine nützliche Erweiterung der Notation ist folgende: Wenn über einem Pfeil ein Zeichen $\bar{a}$ vorkommt, bedeutet dies, daß dem Pfeil bei jedem anderen Zeichen als a gefolgt werden soll.

Die Maschine L_σ ist für jedes σ die Maschine, die sich nach links bewegt, bis sie ein σ findet. Besonders viel Anwendung findet $L_\#$, die folgendermaßen aussieht:

Beispiel

Diese Maschine entscheidet die Sprache {a} (und arbeitet mit dem Alphabet {a, #, m, m̊}):



Nicht-deterministische Turing-Maschinen

sind identisch mit den normalen Turing-Maschinen, außer daß δ nicht mehr eine Funktion sein muß, sondern eine Relation sein kann (und wird dann normalerweise mit Δ bezeichnet).

Zu jeder Sprache L die von einer nicht-deterministischen Turing-Maschine akzeptiert wird, gibt es eine deterministische Turing-Maschine die L akzeptiert (wird nicht bewiesen).

Universalmaschinen

Problem: Eine Turing-Maschine U zu finden, die Berechnungen jeder beliebigen Turing-Maschine simulieren kann.

Erst müssen wir die Annahme machen, daß es (aufzählbare, unendliche) Mengen $V_K = \{q_0, q_1, q_2, \ldots\}$ (wo $q_0 = h$) und $V_\Sigma = \{\sigma_0, \sigma_1, \sigma_2, \ldots\}$ (wo $\sigma_0 = \#$) gibt, so daß jede Turing-Maschine M ein Alphabet $\Sigma \subseteq V_\Sigma$ und eine Zustandsmenge $K \subseteq V_K$ hat. Diese Annahme ist harmlos: Jede Turingmaschine hat *endliche* Σ und K, die immer durch endliche Teilmengen von V_Σ bzw. V_K repräsentiert werden können.

Die Idee ist jetzt (hier weichen wir von L&P etwas ab), wie in richtigen Computern die Zeichen und Zustände als *Bytes*, d.h. als eine Zeichenkette über {0, 1} zu repräsentieren.

Wir müssen dann erst die für eine Maschine M erforderliche *Bytelänge* finden. Sei i die größte Zahl, so daß $\sigma_i \in \Sigma$ oder $q_i \in K$. Die Bytelänge ist dann die *kleinste* Zahl l so daß $2^l > i$.

Wir kodieren jetzt q_j und σ_j beide als j in binärer Darstellung, rechtsgerückt und links mit Nullen auf Länge l verlängert. Diese binäre Darstellung von j bezeichen wir mit $\beta(j, l)$.

Kodieren vom Band und Lesekopf

Den nicht-blanken Teil des Bandes, $w = \sigma_{i_1}\ldots\sigma_{i_n}$, werden wir als

$$\rho(w) = B\beta(i_1, l)B\beta(i_2, l)B\ldots B\beta(i_n, l)B$$

kodieren.

Die Position des Lesekopfes wird folgendermaßen kodiert: *Rechts* der Repräsentation des Zeichens das M gerade sieht, steht statt B ein b.

Kodieren von Transitionen

Eine Transition $\delta(q_i, \sigma_j) = (q_n, \sigma_m)$ wird als

$$D\beta(i, l)D\beta(j, l)D\beta(n, l)D\beta(m, l)D$$

kodiert. Eine Transition $\delta(q_i, \sigma_j) = (q_n, R)$ wird als

$$D\beta(i, l)D\beta(j, l)D\beta(n, l)DRD$$

kodiert, entsprechend auch für L.

Kodieren von δ

Die ganze Funktion δ, mit $n = |K| \cdot |\Sigma|$ Transitionen, wird als

$$\rho(\delta) = T_1\ldots T_n$$

kodiert, wo $T_1, \ldots, T_n$ die Kodierungen der Transitionen sind. Damit sind die Kodierungen der Transitionen durch *zwei D*'s getrennt.

Kodieren vom Startzustand und Maschine:

Der Startzustand $s{=}q_j$ wird einfach durch

$$\rho(s) = Z\beta(j, l)Z$$

kodiert.

Mehr brauchen wir nicht, um eine Maschine vollständig zu repräsentieren. Was die Mengen K und Σ sind, ergibt sich implizit. Also setzen wir

$$\rho(M) = \rho(\delta)\rho(s).$$

Die vollständige Eingabe für die Universalmaschine

Um eine Berechnung von M bei Eingabe w zu simulieren, wird U in der Konfiguration

$(s, \#\rho(M)\rho(w)\underline{\#})$ gestartet.

Das Feld mit dem Startzustand wird später benutzt, um dort den aktuellen Zustand von M zu speichern. Wenn dort $\beta(0, l)$ erscheint, ist M im Haltezustand, und U hält auch, und zwar in der Konfiguration

$(h, \#\rho(\delta)Z\beta(0, l)Z\rho(w')\underline{\#})$

wo w' das Band von M beim Halten ist.

Beispiel

Nehmen wir als Beispiel eine Maschine, die nach rechts nach dem ersten Vorkommen von zwei *a*'s sucht, und nehmen wir an, daß $\sigma_1 = a$ und $\sigma_2 = b$.

Die Maschine ist $M = (\ \{q_1, q_2\}, \{a, b, \#\}, \delta, q_1)$

wo $\delta(q_1, a) = (q_2, R)$
$\delta(q_1, b) = (q_1, R)$
$\delta(q_1, \#) = (q_1, R)$
$\delta(q_2, a) = (h, a)$
$\delta(q_2, b) = (q_1, R)$
$\delta(q_2, \#) = (q_1, R)$

Die erforderliche Bytelänge ist dann 2, und wir kriegen:

$\rho(\delta) =$ *D01D01D10DRDD01D10D01DRDD01D00D01DRD*
D10D01D00D01DD10D10D01DRDD10D00D01DRD

Wenn die Startkonfiguration von M (q_1, *aab*) ist, wird U in folgender Konfiguration gestartet:

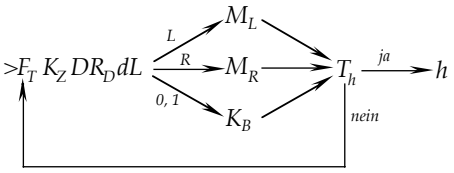
(*s*, *#D01D01D10DRDD01D10D01DRDD01D00D01DRD*
D10D01D00D01DD10D10D01DRDD10D00D01DRD
Z01ZB01b01B10B#)

Zusätzliche Hilfszeichen

Bis jetzt haben wir die Zeichen #, 0, 1, *L*, *R*, *D*, *Z*, *B*, *b* benutzt. Als zusätzliche Hilfszeichen führen wir *d*, *o* und *i* ein. *d* wird als Markierung beim Auffinden von der richtigen Transition benutzt. *o* und *i* werden bei Kopierverfahren als "markierte" Ausgaben von 0 und 1 benutzt.

Die Universalmaschine *U*

sieht in den Grundzügen folgendermaßen aus:



Die Teilmaschine F_T ("finde Transition") findet die Transition, die mit der Kopfposition und dem Zustand übereinstimmt. Die Transition wird markiert, in dem das *D* nach der Kodierung des neuen Zustandes mit einem *d* ersetzt wird.

Beispiel: Wenn das Band vor der Berechnung von F_T so aussieht:

#D101D001D100DLDD…DZ101ZB011B001b#,

sieht es nach der Berechnung folgendermaßen aus:

#D101D001D100dLDD…DZ101ZB011B001b#

Die Teilmaschine K_Z ("kopiere Zustand") kopiert den Zustand links vom *d* zum "Zustandsspeicher" zwischen den beiden *Z*'s, und hält auf dem *d*.

Beispiel: Das Band sieht jetzt so aus:

#D101D001D100dLDD…DZ100ZB011B001b#

Nach diesen Teilmaschinen kommt die Sequenz DR_DdL. Hier wird das *d* in ein *D* umgewandelt. Dann wird das nächste *D* rechts gesucht und in ein *d* umgewandelt. Die Maschine steht jetzt direkt rechts von der Kodierung der Aktion, die *M* ausführen soll (Zeichen schreiben, links oder rechts gehen). Es wird getestet, ob links ein *L*, ein *R* oder die Kodierung eines Zeichens steht. Dann wird entweder eine Bewegungs-Teilmaschine (M_R oder M_L) oder eine Zeichenkopierungs-Teilmaschine (K_B, "kopiere zum Band") benutzt.

Jetzt sieht unser Beispielband folgendermaßen aus:

#D101D001D100DLDD…DZ100ZB011b001B#

Schließlich wird getestet ob der aktuelle Zustand der Haltezustand ist (steht zwischen *Z* und *Z* nur Nullen?), wenn nein, wird eine neue Schleife angefangen, wenn ja, hält die Maschine.

Das Halteproblem für Turing-Maschinen

Das Halteproblem ist das folgende: Gibt es eine Turing-Maschine, die für jede Turing-Maschine *M* und jede Zeichenkette *w* entscheiden kann, ob *M* bei Eingabe *w* hält, das heißt, ob *M w* akzeptiert?

Das Halteproblem ist nicht lösbar. Um dies zu Zeigen führen wir die Sprache K_0 ein:

$K_0 = \{\rho(M)\rho(w)$: *M* akzeptiert $w\}$

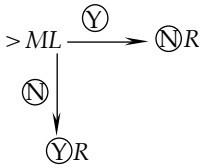
Aber erst zwei Lemmas:

Lemma 1

Wenn *L* Turing-entscheidbar ist, ist $\Sigma^* - L$ es auch.

Beweis

Wenn *L* von *M* entschieden wird, wird $\Sigma^* - L$ von



entschieden. ❑

Lemma 2

Wenn *L* Turing-entscheidbar ist, ist *L* Turing-akzeptabel.

Beweis

Wenn *M L* entscheidet, wird *L* von dieser Maschine akzeptiert:

❑

Theorem

Keine Turingmaschine kann die Sprache K_0 entscheiden.

Beweis

Nehmen wir an, K_0 ist Turing-entscheidbar. Dann ist auch

$K_1 = \{\rho(M)\colon M \text{ akzeptiert } \rho(M)\}$

Turing-entscheidbar (Wenn K_0 von M_0 entschieden wird, können wir eine Maschine M' konstruieren, die aus w erst $w\rho(w)$ berechnet, und dann dies als Eingabe für M_0 weitergibt).

Dann ist aber auch (nach Lemma 1) $\overline{K_1} = \Sigma^* - K_1$ Turing-entscheidbar, nach Lemma 2 ist $\overline{K_1}$ dann auch Turing-akzeptabel. Wir zeigen, daß dies nicht der Fall sein kann (!):

Sei M^* eine Turing-Maschine die $\overline{K_1}$ akzeptiert. Dann gilt:

$\rho(M^*) \in \overline{K_1}$
gdw
M^* akzeptiert nicht $\rho(M^*)$ (Def. von $\overline{K_1}$!)
gdw
$\rho(M^*) \notin \overline{K_1}$ ($\overline{K_1}$ ist die akz. Sprache von M^*!)

Kontradiktion! Dies heißt, daß $\overline{K_1}$ nicht Turing-akzeptabel sein kann. Deshalb ist K_1 nicht Turing-*entscheidbar*, und dann ist auch K_0 nicht Turing-entscheidbar. ❑

Korrolar

Es gibt Sprachen (z.B. $\overline{K_1}$), die nicht Turing-akzeptierbar sind. ❑

Theorem

Es gibt Turing-akzeptierbare Sprachen, die nicht Turing-entscheidbar sind.

Beweis

Wir zeigen, daß K_0 Turing-akzeptabel ist:

Die Universalmaschine U ist *fast* eine Maschine, die K_0 akzeptiert.

Wenn U bei Eingabe w hält, ist es entweder, weil die simulierte Maschine hält (dann ist w eine Kodierung einer Turingmaschine mit Eingabe und $w \in K_0$) oder (möglicherweise) weil die Eingabe nicht die Kodierung einer Turing-Maschine mit Eingabe ist.

Den letzten Fall müssen wir verhindern. Also bauen wir eine Maschine S_U, die nur Eingaben akzeptiert, die korrekte Kodierungen sind (wir lassen die Konstruktion von S_U hier aus).

K_0 wird dann von der zusammengesetzten Maschine $S_U U$ akzeptiert. ❑

Korrolar

Die Turing-akzeptierbaren Sprachen sind nicht unter Komplementbildung abgeschlossen.

Beweis

K_1 ist Turing-akzeptabel, weil K_0 es ist. $\overline{K_1}$ ist aber nicht Turing-akzeptabel. ❑

Die rekursiv aufzählbaren Sprachen und die rekursiven Sprachen

Wenn die Church'sche These wahr ist, bedeutet dies, daß die Begriffe *berechenbar* und *Turing-berechenbar* identisch sind. Weil bis jetzt nichts gegen die Church'sche These spricht, werden deshalb viele Begriffe von Berechenbarkeit *via* Turingmaschinen definiert:

Definition

Eine Sprache heißt *rekursiv aufzählbar*,[2] wenn sie von einer Turing-Maschine akzeptiert wird.

Wir können jetzt auch unsere Definition der *rekursiven* Sprachen präziser machen:

Definition (Präzisierung)

Eine Sprache heißt *rekursiv*, wenn sie von einer Turing-Maschine entschieden wird.

Typ-0-Grammatiken und Turing-Maschinen

Theorem

Die Klasse der rekursiv aufzählbaren Sprachen und die Klasse der Typ-0-Sprachen (von Typ-0-Grammatiken generierten Sprachen) sind identisch.

Beweis

Erst die eine Richtung:

Zu jeder Typ-0-Grammatik gibt es eine Turing-Maschine die $L(G)$ akzeptiert.

Sei G eine Typ-0-Grammatik. Wir können eine Turing-Maschine M_G folgendermaßen konstruieren:

M_G behält während der ganzen Berechnung das Eingabewort w auf dem Band.

M_G erzeugt jede mögliche Zeichenkette über $(V \cup \{\Rightarrow\})^*$, geordnet nach Länge, und für jede Länge in lexikographischer Reihenfolge (z.B. a, S, $\Rightarrow$, aa, aS, $a\Rightarrow$, Sa, SS, $S\Rightarrow$, $\Rightarrow a$, $\Rightarrow S$, $\Rightarrow\Rightarrow$, aaa, … usw.).

Für jede erzeugte Zeichenkette überprüft M_G , ob die Zeichenkette eine G-Ableitung ist, und ob w das letzte Wort der Ableitung ist. Wenn ja, hält M_G, wenn nein setzt sie die Generierung fort.

Dann die andere Richtung (etwas skizzenhaft):

Zu jeder Turing-Maschine gibt es eine Grammatik, die die von M akzeptierte Sprache generiert.

2Eine Sprache wird von einer Turing-Maschine *aufgezählt* gdw sie von einer Turing-Maschine akzeptiert wird (wird nicht gezeigt).

Die Idee ist, eine Konfiguration (q, u, a, v) mit der Zeichenkette [$uqav$] zu repräsentieren. Wir nehmen an, daß die Zustandssymbole nicht Alphabetsymbole sind. Also ist in der Zeichenkette das Zustandssymbol als solches erkennbar, und es markiert gleichzeitig die Position des Lesekopfes (rechts von dem Zustand).

Erst führen wir Regeln ein, mit denen wir jede Zeichenkette

$w[\#ws\#]$

erzeugen können, wo w eine beliebige Zeichenkette über dem Alphabet der Turing-Maschine ist. Diesen Teil der Grammatik können wir sehr ähnlich einer Grammatik für COPY machen (wie in der Übung!), wir müssen nur aufpassen, daß die Regeln nicht auch später angewandt werden können, daß die linke Kopie von w in $w[\#ws\#]$ bei weiteren Ableitungen unberührt bleibt.

Dann führen wir für jede Transition $\delta(q, a) = (p, b)$ in M die Regel $qa \rightarrow pb$ ein (schreibt z.B. $aa[\#aqa]$ in $aa[\#apb]$ um), und für jede Transition $\delta(q, a) = (p, R)$ die Regeln $qab \rightarrow apb$ (für jedes b im Alphabet der Turing-Maschine) und $qa] \rightarrow ap\#]$, ähnliche Regeln werden auch für L-Bewegungen eingeführt.

Schließlich führen wir Regeln ein, die, falls ein h auftaucht, den Teil zwischen [und] löscht, z.B.

$ha \rightarrow h$ und $ah \rightarrow h$

für beliebige Zeichen *a außer* [und], und schließlich

$[h] \rightarrow \varepsilon$.

Wenn nun die Grammatik genau das Alphabet der Turing-Maschine als Nichtterminale hat, sind die einzigen Zeichenketten aus Terminalzeichen genau diejenigen, wo zuletzt die Regel $[h] \rightarrow \varepsilon$ angewandt wurde, also diejenigen Zeichenketten die die Turing-Maschine akzeptiert. ❑

Schlussbemerkung

Wir haben die folgenden Sprachklassen mit zugehörenden Automaten und Grammatiken studiert: Die Sprachklassen sind (mit der Ausnahme von kontextfreien Sprachen mit ε) von oben nach unten durch echte Inklusion geordnet. Ein "echtes Beispiel" ist ein Beispiel einer Sprache der aktuellen Klasse, die nicht zu der engeren Klasse gehört:

Sprachklasse	*"echtes" Beispiel*	*Grammatik-Typ*	*Charakterisierung durch Automaten*
Reguläre Sprachen	a^*	Typ 3	D(N)EA akzeptiert
Det. kontextfreie Spr.	$\{a^n b^n : n \geq 0\}$	—	DKA akzeptiert
Kontextfreie Spr.	*Kompl. von* $\{a^n b^n c^n : n \geq 0\}$	Typ 2	KA akzeptiert
Kontextsensitive Spr.	$\{a^n b^n c^n : n \geq 0\}$	Typ 1	*LBA (nicht vorgestellt)*
Rekursive Sprachen	*(nicht vorgestellt)*	—	Turing-M. entscheidet
Rekurs. aufzählb. Spr.	K_0	Typ 0	Turing-M. akzeptiert
Nicht r.k.a.S.	*Kompl. von* K_1	—	Keine TM akz.