Informatik III

(Automatentheorie und formale Sprachen)

Prof. Dr. Jürgen Dix

Computational Intelligence Group

TU Clausthal

Clausthal, WS 2020/21

Zeit und Ort:

Vorlesung: Montag und Dienstag, jeweils 10–12 im Audimax (wg. Corona)

Übung: Di, vierzehntäglich (**Tobias Ahlbrecht**, Hiwis: P. Böhm/E. Fittschen). **Anmeldung unter StudIP**.

Vorlesungsunterlagen

`https://studip.tu-clausthal.de/...`

Regelmäßig besuchen! Dort finden Sie alles Wichtige über die Vorlesung, Dokumente, Übungen et cetera.

Prüfungsklausur: 29. März 2021, 9 Uhr

Zulassung zur Klausur: >50% der Übungspunkte, und bei maximal einem Hausübungsblatt weniger als 20%.

Es gibt **Bonuspunkte** für die Klausur, wenn deutlich mehr als die notwendigen Punkte erzielt wurden.

Outline

Danksagung

Historie

Diese Vorlesung orientiert sich stark an "Theoretische Informatik" von Katrin Erk und Lutz Priese.

Vielen Dank an Katrin Erk, die mir vor vielen Jahren ihr gesamtes Material zur Verfügung gestellt hat. Dank auch an Michaela Huhn, die weitere Folien beigesteuert hat.

Der Teil über P/NP stammt im Wesentlichen aus Hopcroft/Ullmann und wurde von mir mehrmals in Manchester gelesen.

Organisatorisches

- Zulassung zur Klausur: (1) 50 % der Übungspunkte (jedes Übungsblatt muss mit mindestens 20 % bestanden werden), und (2) jeder Student muss mindestens zweimal an der Tafel eine Aufgabe erfolgreich vorgerechnet haben.
- Übungen können zu Gruppen von maximal zwei Studenten abgegeben werden. Bei Plagiaten werden alle identischen Blätter mit 0 Punkten bewertet.
- Zur Klausur sind keine Bücher, Folien oder Mitschriften zugelassen. **Lediglich ein handgeschriebenes Din A4 Blatt ist erlaubt.**
- Die Übungszettel sind jeweils bis Montags, 13 Uhr in den Zettelkasten zu legen.
- Für diejenigen die deutlich mehr als die 50 % schaffen, gibt es Bonuspunkte. Diese werden bei der Klausur angerechnet: falls man nah an der Grenze zur besseren Note steht, bekommt man sie (man muss allerdings bestanden haben).

Worum geht es eigentlich? (1)

Um Grundbegriffe der theoretischen Informatik:

Sprachen: das sind Mengen von **Zeichenketten** (eine Zeichenkette heißt auch **Wort**),

Grammatiken: das sind **Kalküle**, um Sprachen zu **erzeugen** (jedes Wort wird erzeugt),

Automaten: das sind **Maschinen**, die Sprachen **definieren oder transformieren** (indem sie testen, ob ein Wort zu einer Sprache gehört, Sprachen erzeugen, oder Eingabe-Sprachen in Ausgabe-Sprachen übersetzen).

Warum? (1)

Sprachen und ihre Verarbeitung sind ein zentrales Thema der Informatik.

- Programmiersprachen und ihre Compiler;
- Skriptsprachen und Interpreter, Protokolle;
- Eingabesprachen für Webdienste, mobile Dienste, . . . ;
- Werkzeuge für die Textverarbeitung, Engineering-Aufgaben oder domänenspezifische Sprachen;
- . . .

Sprachen und Automaten gehören in den Werkzeugkoffer jedes Informatikers!

Worum geht es eigentlich? (2)

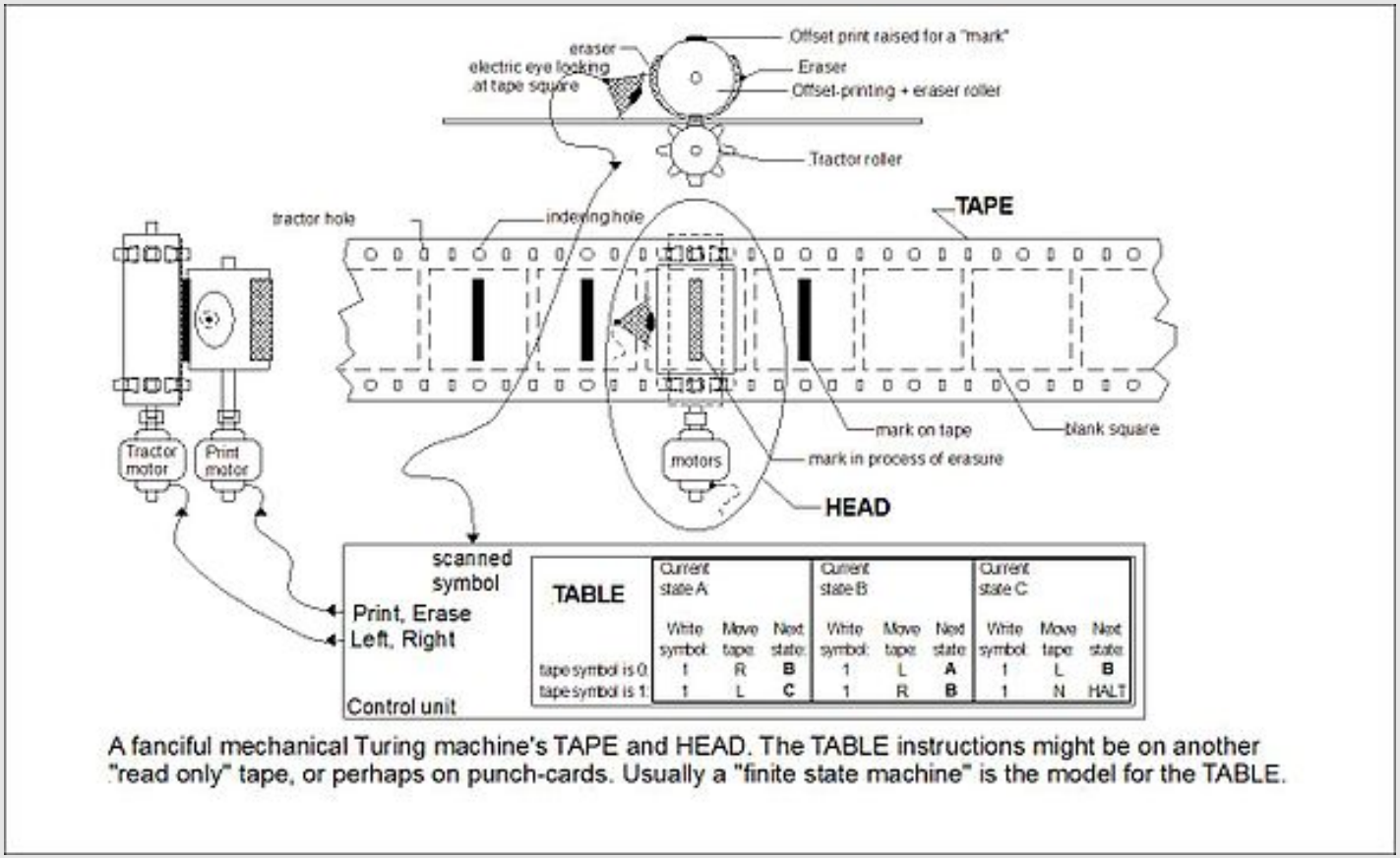
Präzise Definition des Algorithmus-Begriffs: ein Algorithmus ist der Programmcode einer **Turingmaschine**.

Der Algorithmusbegriff ist **universell**, d.h. auch mit anderen Berechnungsmodellen wie While- oder GoTo- Programmen, μ-rekursiven Funktionen oder Termersetzungssystemen funktioniert es.

Wir zeigen, daß das Halte- und viele weitere Probleme **unentscheidbar** sind, d.h. sie können grundsätzlich nicht berechnet werden.

Warum? (2)

Berechenbarkeitstheorie hat die Mathematik in der ersten Hälfte des 20. Jahrhunderts erschüttert!



A fanciful mechanical Turing machine's TAPE and HEAD. The TABLE instructions might be on another "read only" tape, or perhaps on punch-cards. Usually a "finite state machine" is the model for the TABLE.

Abbildung 1: Turing-Maschine ©Wikipedia

Worum geht es eigentlich? (3)

Auf Basis einer präzisen Definition des Algorithmus-Begriffs, des Programmcodes einer Turingmaschine, kann die Komplexität von Problemen untersucht werden:

Wir führen Komplexitätsklassen ein. Die wichtigsten sind die Klassen $\mathbf{P}$ und $\mathbf{NP}$.

Wir zeigen von vielen Problemen, daß sie dieselbe Komplexität haben, ohne diese Komplexität genau zu kennen (sie sind $\mathbf{NP}$ vollständig).

Warum? (3)

Wie entwickeln sich Speicherplatz und Rechenzeit bei großen Probleminstanzen?

Abbildung 2: Traveling Salesperson Instanz in Google Maps

Lernstrategie

JedeR[a] kann 10 km Laufen[b]!

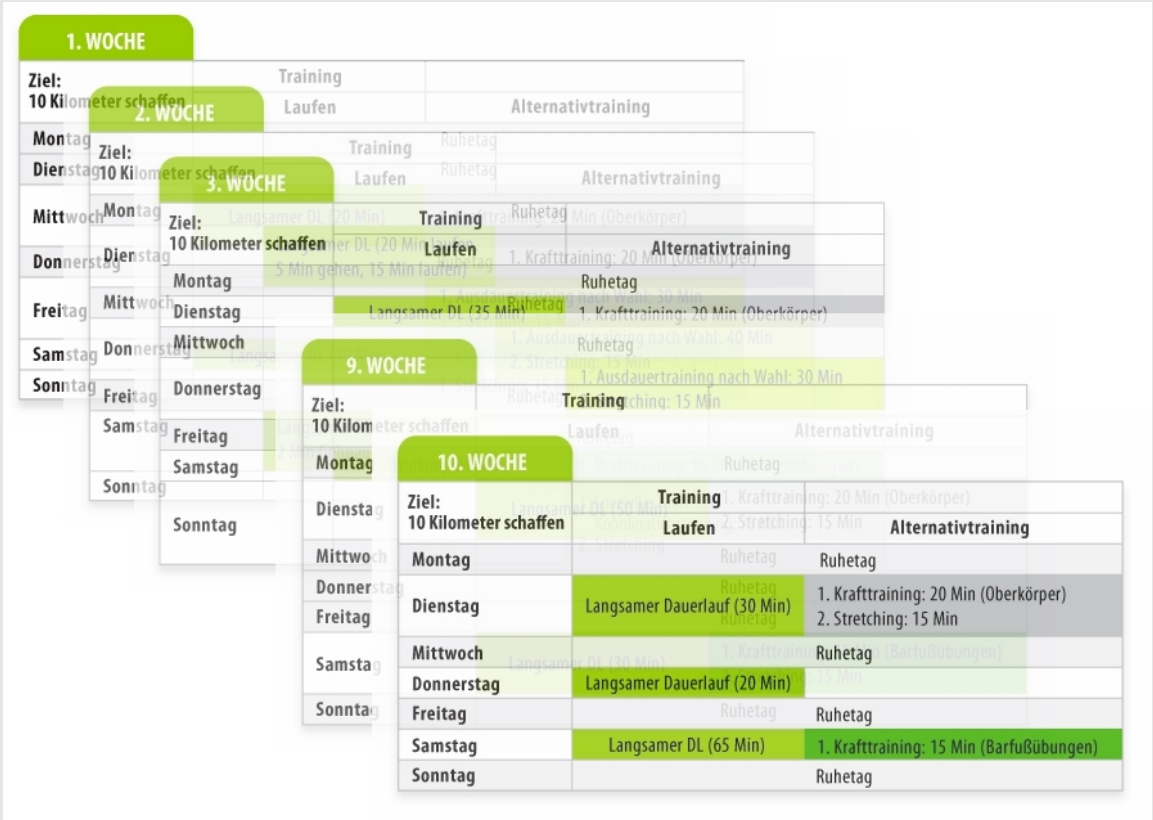
Abbildung 3: ©www.lauf-infos.de

wenn...

[a] gesund, Alter zwischen 15 und 50,...
[b] Ziel: 10 km schaffen, nicht U60', U50' oder gar U35'

Zur Lernstrategie

Wenn er/sie 10 - 12 Wochen konsequent trainiert!



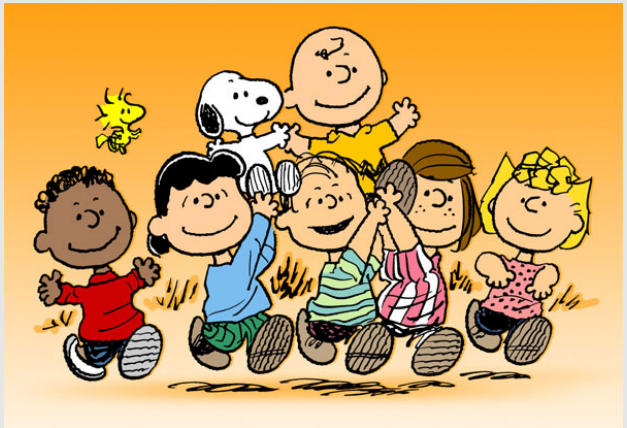
10. WOCHE

Ziel: 10 Kilometer schaffen	Training	
	Laufen	Alternativtraining
Montag		Ruhetag
Dienstag	Langsamer Dauerlauf (30 Min)	1. Krafttraining: 20 Min (Oberkörper) 2. Stretching: 15 Min
Mittwoch		Ruhetag
Donnerstag	Langsamer Dauerlauf (20 Min)	
Freitag		Ruhetag
Samstag	Langsamer DL (65 Min)	1. Krafttraining: 15 Min (Barfußübungen)
Sonntag		Ruhetag

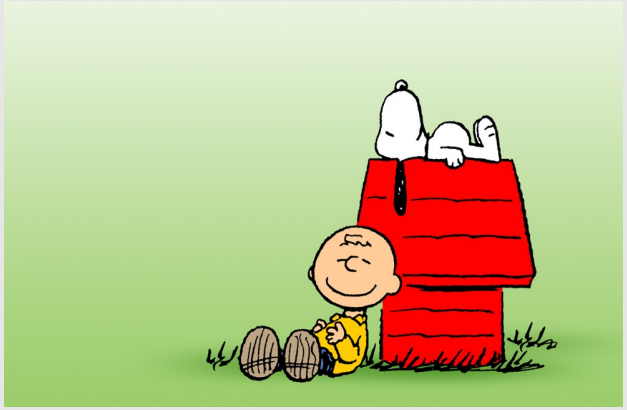
Abbildung 4: ©www.joggen-online.de

Gründe für Erfolg oder Mißerfolg:

Konsequente Teilnahme am Training $\Rightarrow$ 10 km Lauf $\surd$



Einstieg verpasst, Trainingslücken,. . . $\Rightarrow$ ✘



Für einen 10 km Lauf und für Info III gilt:

Trainieren **Sie** - jetzt!

- **Das erste Übungsblatt ist schon in zwei Wochen fällig.**
- **Ebenso die erste Übung.**

Literatur I

Erk, Katrin and Priese, Lutz (2001).
Theoretische Informatik: Eine umfassende Einführung.
Springer.

Hoffmann, Dirk (2011).
Theoretische Informatik.
Carl Hanser.

Hopcroft, J. and J. Ullman (1979).
Introduction to Automata Theory, Languages, and Computation.
Reading, MA: Addison Wesely.

Lin, S. and T. Rado (1965).
Computer studies of turing machine problems.
Journal of the ACM 12(2), 196–212.

Literatur II


Papadimitriou, C. (1994).
Computational Complexity.
Addison-Wesley.


Rado, T. (1962).
On non-computable functions.
The Bell System Technical Journal XLI(3), 877–884.


Turing, A. M. (1936).
On Computable Numbers with an Application to the Entscheidungsproblem.
Proc. London Mathematical Soc., series 2 42, 230–265.
corrections ibid., 43:544–546.

1. Terminologie

Motivation

In diesem Kapitel führen wir die Begriffe

- Sprache und
- Grammatik

ein.

In der gesamten Vorlesung geht es darum zu untersuchen, welche Sprachen man durch welche Grammatiken ausdrücken kann (und welche nicht).

Inhalt

Wir untersuchen insbesondere:

1. Wie man Probleme aus der Mathematik (Graphentheorie, Logik) als **Probleme über Sprachen** formulieren kann.
2. Wie man Klassen von Grammatiken von steigendem Schwierigkeitsgrad definiert: **Chomsky-Hierarchie**.
3. **Wieviele** Grammatiken und Sprachen **gibt es überhaupt** (soviele wie natürliche Zahlen, reelle Zahlen oder komplexe Zahlen)?

1.1 Sprache, Grammatik

Grundlage einer Sprache ist das **Alphabet**. Es legt die zur Verfügung stehenden Zeichen (auch **Buchstaben** genannt) fest. Ein Alphabet ist oft **endlich**.
Eine Sprache besteht aus **Wörtern**: endliche Sequenzen von Buchstaben (Zeichenreihen). Es gibt mehrere Operationen auf Wörtern und auf Sprachen.

Alphabet: Σ, endlich;

Wort: w, auch **String** oder **Zeichenreihe** genannt, ist immer über einem Alphabet Σ definiert: es ist eine **endliche Folge** von Symbolen aus Σ. Die Länge eines Wortes (bezeichnet mit $|w|$) ist die Anzahl seiner Buchstaben. ε bezeichnet das leere Wort: es hat als einziges die Länge 0.
Achtung: ε ist ein **Metasymbol** für ein Wort, kein Symbol aus dem Alphabet Σ.

Sprache: **Eine Sprache L ist eine Menge von Wörtern** über einem Alphabet Σ. Auf der Menge aller Sprachen $\mathsf{FS}(\Sigma)$ definieren wir folgende Operationen: (sei $L, M \in \mathsf{FS}(\Sigma)$)

1. **Konkatenation ($\circ$):**
   $L \circ M := \{w \circ w' : w \in L,\ w' \in M\}$,
   oft läßt man "$\circ$" weg: LM, ww'.
2. **i-te Potenz von L (^{i}):** $L^0 := \{\varepsilon\}$, $L^{i+1} := LL^i$,
3. **Kleene-Hülle von L (*):** $L^* := \bigcup_{i=0,1,2,\ldots} L^i$,
4. **Variante der Kleene-Hülle von L (L^+):**
   $L^+ := LL^*$. Man zeigt leicht, daß
   $L^+ = \bigcup_{i=1,2,\ldots} L^i$.
5. **Reverse (^{R}):** w^R bezeichnet das Wort w rückwärts gelesen: $(abbcd)^R := dcbba$. Analog ist $L^R := \{w^R : w \in L\}$.
   Dabei ist $\varepsilon^R := \varepsilon$.

Beispiel 1.1 (Sprache der geraden Zahlen)

Σ	$\{0, 1, 2, 3, \ldots, 9\}$
L	Menge der geraden Zahlen
z.B.	$2, 4, 6, 8, 10, 12, 14, \ldots$
aber nicht	$1, 3, 5, \ldots, 111, 113, \ldots$

Tabelle 1: Gerade Zahlen

Beispiel 1.2 (Aussagenlogische Ausdrücke)

Σ	$\{x, y, z, \wedge, \vee, \neg, (,)\}$
L	Menge der aussagenlogischen Ausdrücke über den Variablen x, y und z
z.B.	$x \wedge y$, $\neg y$, $(x \vee y) \vee (z \wedge x), \ldots$
aber nicht	$xx, \neg yz, \wedge\vee, \neg, \ldots$

Tabelle 2: Sprache: Aussagenlogische Ausdrücke

Σ^* ist also die **Menge aller Wörter über Σ**.
$\Sigma^+ := \Sigma^* \setminus \{\varepsilon\}$ ist die Menge aller Wörter über Σ der Länge echt größer als 0.

Mithilfe der obigen Operationen bildet man **reguläre Ausdrücke**, z.B.:

$$(aba)^*bb^* \ + \ (aba)^*a \ + \ (ba).$$

Definition 1.3 (Reguläre Ausdrücke $\mathfrak{Reg}_\Sigma$)

Reguläre Ausdrücke sind wie folgt definiert:

1. 0 ist ein **regulärer Ausdruck**.
2. Für jedes $a \in \Sigma$ ist a ein **regulärer Ausdruck**.
3. Sind r und s **reguläre Ausdrücke**, so auch
   - $(r + s)$ (Vereinigung),
   - (rs) (Konkatenation),
   - (r^*) (Kleene Stern).

Wir lassen die Klammern weg und vereinbaren, daß * Vorrang vor der Konkatenation hat, die wiederum Vorrang vor $+$ hat.

Definition 1.4 (Semantik regulärer Ausdrücke)

Ein **regulärer Ausdruck** r stellt eine Sprache $\mathfrak{I}(r)$ über Σ wie folgt dar:

- $\mathfrak{I}(0) := \emptyset$,
- $\mathfrak{I}(a) := \{a\}$, für $a \in \Sigma$,
- $\mathfrak{I}(r+s) := \mathfrak{I}(r) \cup \mathfrak{I}(s)$,
- $\mathfrak{I}(rs) := \mathfrak{I}(r)\mathfrak{I}(s)$,
- $\mathfrak{I}(r^*) := \mathfrak{I}(r)^*$.

Oft benutzen wir auch a^+ in einem regulären Ausdruck (beachte, daß dies ein **Makro** ist). Wir benutzen "1", definiert durch

$$1 := 0^*$$

Man rechnet leicht nach, daß $\mathfrak{I}(1) = \{\varepsilon\}$. **Vorsicht:** Falls $0, 1 \in \Sigma$, kann es zu Verwechslungen kommen.

Übung:

Welche Sprachen stellen die folgenden regulären Ausdrücke dar?

- aa,
- $(a+b)^*$,
- $aa^* + bb^*$.

Beachten Sie, daß gilt:

- $\{abb\}c^*\{b\} = abbc^*b$,
- $\{aba\}^*\{a\} \cup \{ba\} = (aba)^*a + ba$.

Nur die Ausdrücke auf den rechten Seiten sind reguläre Ausdrücke. Wir benutzen im folgenden beide Schreibweisen, um Sprachen bzw. Mengen auszudrücken.

Beispiel 1.5 (Sprache der geraden Zahlen)

Σ	$\{0, 1, 2, 3, \ldots, 9\}$
L	Menge der geraden Zahlen
z.B.	$2, 4, 6, 8, 10, 12, 14, \ldots$
aber nicht	$1, 3, 5, \ldots, 111, 113, \ldots$
reg. Ausdruck	$(1+2+3+4+5+6+7+8+9)(0+1+2+3+4+5+6+7+8+9)^*(0+2+4+6+8)+(0+2+4+6+8)$

Tabelle 3: Gerade Zahlen als regulärer Ausdruck

Eine **Grammatik** ist ein **Kalkül** um eine Sprache zu definieren. Also eine **Menge von Regeln**, mit deren Hilfe man Wörter ableiten kann.
Die zu einer Grammatik gehörende **Sprache** besteht aus den **ableitbaren, terminalen Wörtern**.

Definition 1.6 (Σ-Grammatik)

Eine **Grammatik** G über einem Alphabet Σ (kurz **Σ-Grammatik**) ist ein Tupel $G = (V, T, R, S)$. Dabei ist

- V eine endliche Menge von **Variablen**;
- $T \subseteq \Sigma$ eine endliche Menge von **Terminalen** mit $V \cap T = \emptyset$;
- R eine endliche Menge von **Regeln**. Eine Regel ist ein Element (P, Q) aus $\left((V \cup T)^* \, V \, (V \cup T)^*\right) \times (V \cup T)^*$. Das heißt,
  - P (die Prämisse) ist ein Wort über $(V \cup T)$, das mindestens eine Variable aus V enthält,
  - Q (die Conclusio) ist ein beliebiges Wort über $(V \cup T)$.

  Für eine Regel $(P, Q) \in R$ schreibt man üblicherweise auch $P \to_G Q$ oder nur $P \to Q$.
- S das Startsymbol, $S \in V$.

Oft sagen wir einfach “Grammatik G”, da Σ vorher festgelegt wurde und sich nicht ändert.

Beispiel 1.7

$S \to B \qquad B \to do\ begin\ B\ end \qquad B \to A$
$A \to nop\ A \qquad A \to \varepsilon$

- Wir schreiben normalerweise Variablen als Großbuchstaben,
- Terminale als Kleinbuchstaben.
- Wenn es mehrere Regeln mit derselben Prämisse gibt, also z.B. $P \to Q_1$, $P \to Q_2$, $P \to Q_3$, so schreibt man dafür auch kurz

$$P \to Q_1 \mid Q_2 \mid Q_3.$$

Rechnung einer Grammatik:

- Beginn beim Startsymbol, einer Variablen.
  Dies ist ein Wort.
- Eine Grammatikregel $P \to Q$ sagt, daß ein Vorkommen von P durch Q ersetzt werden darf:
  Wenn das aktuelle Wort die Prämisse P enthält, dann kann man P durch die dazugehörige Conclusio Q ersetzen. Das Ergebnis ist ein neues aktuelles Wort.
  Eine Prämisse muss eine Variable enthalten.
  Umgekehrt ist ein Wort, das noch eine Variable enthält, nicht “fertig”.
- Am Ende steht ein terminales Wort: also eines, in dem keine Variablen vorkommen. Es besteht nur aus Terminalen.

Beispiel 1.8 (Einfache Grammatiken)

- Sei $G_a = (\{S\}, \{a\}, \{R_1, R_2\}, S)$ eine Grammatik mit $R_1 = S \to aS$, $R_2 = S \to \epsilon$.
- Sei $G_{ab} = (\{S\}, \{a, b\}, \{R_1, R_2\}, S)$ eine Grammatik mit $R_1 = S \to aSb$, $R_2 = S \to \epsilon$.
- Sei $G_{gerade} = (\{S, S_0\}, \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}, \{R_1, R_2\}, S)$ eine Grammatik mit $R_1 = S \to 1S \,|\, 2S_0 \,|\, 3S \,|\, 4S_0 \,|\, 5S \,|\, 6S_0 \,|\, 7S \,|\, 8S_0 \,|\, 9S$, $R_2 = S_0 \to S \,|\, \epsilon$.

Welche Wörter kann man ableiten?

Definition 1.9 (Ableitung, Rechnung)

- Ist $G = (V, T, R, S)$ eine Grammatik, w, w' Wörter aus $(V \cup T)^*$,

so gilt "$\mathbf{w \Longrightarrow w'}$", **w geht über in w'**, falls gilt: es gibt $u, v \in (V \cup T)^*$, und $P \to Q \in R$ mit

$$\big(w = uPv \text{ und } w' = uQv\big).$$

Für "w geht über in w' ...

- ... mit der Grammatik G" schreibt man auch: $w \Longrightarrow_G w'$
- ... durch die Regel $P \to Q$" schreibt man auch: $w \Longrightarrow_{P \to Q} w'$.

Falls es Wörter $w_0, \ldots, w_n \in (V \cup T)^*$ gibt mit $w = w_0$, $w_n = w'$ und $w_i \Longrightarrow_G w_{i+1}$ für $0 \leq i < n$, so schreiben wir dafür auch $w \Longrightarrow_G^* w'$.

$n = 0$ ist erlaubt, das heißt, $w \Longrightarrow_G^* w$ gilt stets.

Die Folge $w_0, \ldots, w_n$ heißt **Ableitung** oder **Rechnung** (von w_0 nach w_n in G) der Länge n.

Vorsicht Indeterminismus:

Es kann mehrere Wege geben, um zum gleichen Wort zu kommen.

Beispiel 1.10 (Indeterminismus)

Wir betrachten die Grammatik

$$G = (\{S, B\}, \{a, b, c\}, \{R_0, R_1, R_2, R_3\}, S)$$

$$\begin{aligned} R_0 &= S \to aBBc \\ R_1 &= B \to b \\ R_2 &= B \to ba \\ R_3 &= BB \to bBa \end{aligned}$$

Drei Möglichkeiten, das Wort $abbac$ zu erzeugen:

$$
\begin{array}{llllllll}
S & \Longrightarrow_{R_0} & aBBc & \Longrightarrow_{R_1} & abBc & \Longrightarrow_{R_2} & abbac \\
S & \Longrightarrow_{R_0} & aBBc & \Longrightarrow_{R_2} & aBbac & \Longrightarrow_{R_1} & abbac \\
S & \Longrightarrow_{R_0} & aBBc & \Longrightarrow_{R_3} & abBac & \Longrightarrow_{R_1} & abbac
\end{array}
$$

Wenn ein Wort w mehrere Regel-Prämissen enthält, gibt es mehrere mögliche weitere Rechnungen.

Warum ist das ein *feature* und kein *bug*?

- Erlaubt oft einfachere Definitionen von Grammatiken.
- Manchmal gibt es keine eindeutige Grammatiken.
- Eine Grammatik beschreibt die Struktur der Wörter.
  Diese Struktur kann etwas bedeuten.
  Wenn dann die Grammatik nicht eindeutig ist, heißt das, daß manche Wörter mehrere mögliche Strukturen haben.
- Für **natürliche Sprachen** braucht man das unbedingt.
  Manche Sätze sind mehrdeutig, also müssen auch die Grammatiken mehrdeutig sein!

Time flies like an arrow.
Fruit flies like a banana.

Definition 1.11 (Erzeugte Sprache $L(G)$, Äquivalenz)

Die von einer Grammatik G **erzeugte Sprache** $\boldsymbol{L(G)}$ ist die Menge aller **terminalen** Wörter, die durch G vom Startsymbol S aus erzeugt werden können:

$$L(G) := \{w \in T^* \mid S \Longrightarrow_G^* w\} \tag{1}$$

Zwei Grammatiken G_1, G_2 heißen **äquivalent**, falls $L(G_1) = L(G_2)$.

Übung

Wir betrachten die Grammatik

$$G_2 = (\{S\}, \{a, b\}, \{R_1, R_2\}, S)$$

mit $R_1 = S \to aSb$, $R_2 = S \to \varepsilon$ von Beispiel 1.8. Um ein Wort zu erzeugen, starten wir mit dem Startsymbol S:

$$S \Longrightarrow_{R_1} aSb \Longrightarrow_{R_1} aaSbb \Longrightarrow_{R_1} aaaSbbb \Longrightarrow_{R_2} aaabbb$$

Damit haben wir mit G_2 das Wort a^3b^3 erzeugt. Es lassen sich durch mehr oder weniger Anwendungen von R_1 auch andere Wörter hervorbringen.

Lemma 1.12

G_{ab}, die Grammatik aus Beispiel 1.8, erzeugt die Sprache $L(G_{ab}) = \{a^n b^n \mid n \in \mathbb{N}_0\}$.

Beweis

Daß G_{ab} tatsächlich genau diese Sprache erzeugt, zeigen wir allgemein, indem wir alle möglichen Ableitungen von G_{ab} betrachten.

$\subseteq$: zu zeigen: **Jedes terminale Wort, das G_{ab} erzeugt, hat die Form $a^n b^n$.**
Wir zeigen für alle $w \in (V \cup T)^*$: Falls $S \Longrightarrow^*_{G_{ab}} w$, dann gilt entweder $w = a^n S b^n$ oder $w = a^n b^n$ für ein $n \in \mathbb{N}_0$.

Dazu verwenden wir eine **Induktion über die Länge einer Ableitung** von S nach w.

Induktionsanfang: $w = S = a^0 S b^0$

Induktionsschritt: Es gelte $S \Longrightarrow^*_{G_{ab}} w \Longrightarrow_{G_{ab}} w'$, und für w gelte nach der Induktionsvoraussetzung bereits $w = a^n b^n$ oder $w = a^n S b^n$. Außerdem sei $w \Longrightarrow_{G_{ab}} w'$ eine Ableitung in einem Schritt. Nun ist zu zeigen: $w' = a^m b^m$ oder $w' = a^m S b^m$ für irgendein m.

Fall 1: $w = a^n b^n$. Dann konnte keine Regel angewandt werden, da w schon terminal ist, also tritt dieser Fall nie auf.

Fall 2: $w = a^n S b^n$. Dann wurde von w nach w' entweder Regel R_1 oder R_2 angewandt.
Falls R_1 angewandt wurde, dann gilt $w = a^n S b^n \Longrightarrow_{R_1} a^n a S b b^n = a^{n+1} S b^{n+1} = w'$.
Falls R_2 angewandt wurde, dann gilt $w = a^n S b^n \Longrightarrow_{R_2} a^n \varepsilon b^n = w'$. Dies Wort ist terminal und hat die geforderte Form $a^n b^n$.

$\supseteq$: zu zeigen: Für alle n kann $a^n b^n$ von G_{ab} erzeugt werden:
$S \Longrightarrow^*_{G_{ab}} a^n b^n \; \forall n \in \mathbb{N}_0$.
Um $a^n b^n$ zu erzeugen, wende man auf S n-mal die Regel R_1 und dann einmal die Regel R_2 an. $\square$

Definition 1.13 (Dycksprache D_k)

Für $k \in \mathbb{N}$ sei $V_k := \{x_1, \overline{x_1}, x_2 \ldots, x_k, \overline{x_k}\}$ ein Alphabet über den $2k$ Symbolen $x_1, \overline{x_1}, \ldots, x_k, \overline{x_k}$. Die Dycksprache D_k ist die **kleinste Menge** die folgende Bedingungen erfüllt:

1. $\epsilon \in D_k$,
2. Falls $w \in D_k$, so auch $x_i w \overline{x_i}$.
3. Falls $u, v \in D_k$, so auch uv.

Interpretiert man die x_i als öffnende, die $\overline{x_i}$ als zugehörige schließende Klammern, so kann man die Dycksprache als die **Menge aller korrekten Klammerausdrücke** sehen.

1.2 Warum Sprachen?

Einige interessante Mengen wurden schon als Sprachen beschrieben, z.B. gerade Zahlen, Klammersprachen.
In der Einleitung wurde versprochen:

Relevante Probleme der Mathematik und Informatik!

Diesem Anspruch versuchen wir jetzt gerecht zu werden:

Fakt 1.14

Viele Fragestellungen der Informatik und der Mathematik können als Probleme über Sprachen formalisiert werden.

Die folgenden Beispiele zeigen, wie man Probleme als Sprachen formalisieren kann.

Primzahlen

Kann man die Menge der Primzahlen als formale Sprache über einem Alphabet sehen?

Beispiel 1.15 (Primzahlen)

Σ	$\{\,\vert\,\}$
L_{prim}	Menge der Primzahlen
z.B.	$\vert\vert, \vert\vert\vert, \vert\vert\vert\vert\vert, \vert\vert\vert\vert\vert\vert\vert, \vert\vert\vert\vert\vert\vert\vert\vert\vert\vert, \ldots$
aber nicht	$\vert, \vert\vert\vert\vert, \vert\vert\vert\vert\vert\vert, \vert\vert\vert\vert\vert\vert\vert, \vert\vert\vert\vert\vert\vert\vert\varepsilon, \vert, \ldots$

Tabelle 4: Primzahlen in Unärdarstellung

Statt | können wir natürlich auch 0, 1, l oder ★ benutzen.

Wichtig:

Gebraucht wird ein Eingabealphabet $\Sigma := \{0, 1, \ldots, n-1\}$, um eine Ganzzahl zur Basis n darzustellen ($n > 1$). Zum Beispiel wird die Zahl 5 binär durch 101 dargestellt. Für die unäre Darstellung benutzt man $|||||$ oder auch 11111 (man könnte auch 00000 benutzen um es mit der Darstellung oben konsistent zu machen).

Anmerkung:

Die n-äre Darstellung ($n > 1$) einer Zahl m führt zu einer Speicherersparnis: Statt m Zellen eines Speicherbandes zu benutzen (in Unärdarstellung) werden nur $\log_n m$ benötigt.

Trotzdem ist nur der Schritt von der unären zur binären Darstellung – also von m nach $\lg m$ – eine wirkliche Verbesserung.

Der Umstieg auf $\log_n m$ lässt nur die Einsparung eines (linearen) Faktors zu.

- Wie kann man die Menge aller **booleschen Ausdrücke** der Form $x_3 \vee x_1 \wedge x_2 \vee \neg x_1$ als Sprache darstellen?
- Wie kann man die **Länge der Formel** $x_3 \vee x_1 \wedge x_2 \vee \neg x_1$ definieren?

Boolesche Ausdrücke

Idee: Eine Variable x_i wird durch das Symbol x gefolgt vom Index i in Binärnotation kodiert. x_{11} ist damit $\rightarrow x11$.

Beispiel 1.16 (Boolesche Audrücke)

Σ_{sat}	$\{\wedge, \vee, \neg, (,), x, 0, 1\}$
L_{bool}	Menge der Booleschen Ausdrücke
z.B.	$x1 \vee x101, (x11 \vee x1) \wedge (x10 \vee \neg x1), \ldots$
aber nicht	$\wedge, \neg\vee, x(), x101x10x11, \ldots$

Tabelle 5: Boolesche Ausdrücke mit Variablenindizes in Binärdarstellung

Boolesche Ausdrücke

Welches Alphabet benötigen wir um boolesche Formeln (wie etwa $x_1 \vee (x_2 \wedge x_4)$) zu kodieren?

$\Sigma_{\text{sat}} := \{\wedge, \vee, \neg, (,), x, 0, 1\}$. Dies genügt, um einen booleschen Ausdruck w mit den Variablen $\{x_1, x_2, \ldots\}$ zu definieren. Eine Variable x_i wird durch das Zeichen x gefolgt von der Zahl i in binärer Notation kodiert.
Wie üblich wird ein ganzer Ausdruck abhängig von der Belegung der Variablen als **t** (true) or **f** (false) ausgewertet.

Ein wichtiges Problem ist es, von einem booleschen Ausdruck zu entscheiden, ob er erfüllbar ist: satisfiability-problem (SAT).

Frage

Existiert eine Belegung der Variablen im Ausdruck w derart, daß w als **t** ausgewertet wird?

Wie kann man dies als Sprache (oder Grammatik) darstellen?

Frage

Angenommen, es läge ein boolescher Ausdruck (in der üblichen Form) mit n Symbolen vor. **Wie lang ist die kodierte Version in Σ_{sat}?**

Es können nur $\lceil \frac{n}{2} \rceil$ Variablen vorkommen, jede benötigt nicht mehr als $1 + \lceil \lg n \rceil$ Symbole: also höchstens

$$\begin{aligned}
&\leq \lceil \tfrac{n}{2} \rceil (1 + \lceil \lg n \rceil) + \lceil \tfrac{n}{2} \rceil \\
&\leq \lceil \tfrac{n}{2} \rceil (2 + \lceil \lg n \rceil) \\
&\leq \lceil \tfrac{n}{2} \rceil (2 \lceil \lg n \rceil) \\
&\leq n \lceil \lg n \rceil
\end{aligned}$$

Fakt 1.17

Alle späteren Komplexitäts-Betrachtungen werden unabhängig davon sein, **ob nun n oder $n \lg n$ als Länge der Eingabe** *betrachtet wird.*

Wir definieren:

Definition 1.18 (Satisfiability)

$$L_{\text{sat}} := \{w \in \Sigma^*_{sat} : \text{es gibt Belegungen für } x_i \text{ so, daß die Formel } w \text{ wahr ist}\}$$

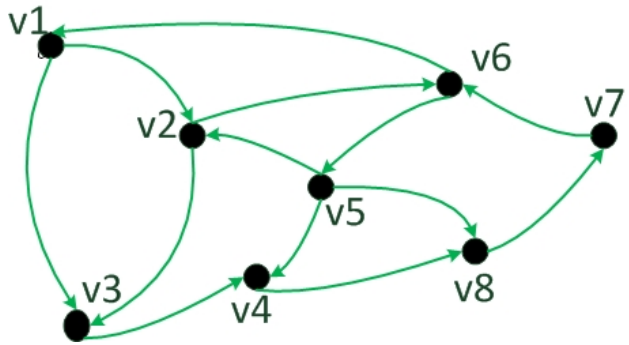
Graphen

Definition 1.19 (Graph)

Ein **Graph** $\mathcal{G} = \langle V, E \rangle$ besteht aus

- einer Menge von Knoten V, und
- $E \subseteq V \times V$, der Kantenmenge.

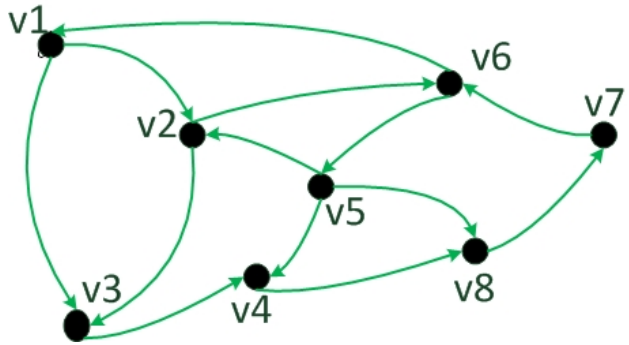
Wenn für alle $u, v \in V$ gilt: $(u, v) \in E \Leftrightarrow (v, u) \in E$, dann heißt $\mathcal{G}$ ein ungerichteter Graph, sonst heißt $\mathcal{G}$ gerichtet.



Frage: Erreichbarkeitsproblem

Gibt es einen Weg in $\mathcal{G}$ von v_i zu v_j?

Graphen



Beispiel 1.20 (Graphsprache)

Alphabet: $\Sigma_{\text{graph}} := \{v, e, 0, 1, (,), \#\}$.

v_i wird repräsentiert durch die Zeichenkette v gefolgt vom Index binär kodiert.
Eine gerichtete Kante $e_{i,j}$ wird kodiert als Zeichenkette $(\text{string1}\#\text{string2})$, wobei $string1$ die binäre Darstellung von i und $string2$ die binäre Darstellung von j ist.

Prüfen Sie Ihr Verständnis: Ergänzen Sie fehlende Details zur Darstellung von Graphen, und schreiben Sie den obigen Graphen als Wort über Σ_{graph} auf!

Definition 1.21 (Erreichbarkeitsproblem L_{reach})

Das Erreichbarkeitsproblem ist wie folgt als formale Sprache definiert:

$$L_{\text{reach}} := \{w \in \Sigma^*_{\text{graph}} : \begin{array}{l}\text{es gibt einen Weg in } w \\ \text{von der ersten Ecke } v_1 \\ \text{zur letzten Ecke } v_n\}\end{array}$$

Ganzzahlige lineare Programmierung

Gegeben sei eine $m \times n$ Matrix $\mathbf{A}$ über $\mathbb{Z}$ und ein Spaltenvektor $\mathbf{b}$.

Frage:

Gibt es einen Spaltenvektor $\mathbf{x}$, der $\mathbf{Ax} \geq \mathbf{b}$ erfüllt?

Repräsentation: Wörter der Sprache sind die Einträge von $\mathbf{A}$ und $\mathbf{b}$: alle werden binär dargestellt.

Definition 1.22

$L_{\text{ILP}} := \{w \in \Sigma^*_{\text{ILP}} :$ w repräsentiert ein ILP-Problem $\langle \mathbf{A}, \mathbf{b} \rangle$ so, daß es ein $\mathbf{x}$ mit $\mathbf{Ax} \geq \mathbf{b}$ gibt.$\}$

Die Beispiele dienen als Beleg für Fakt 1.14.[1]

Fakt 1.14

Viele Fragestellungen der Informatik und der Mathematik können als Probleme über Sprachen formalisiert werden.

Die Darstellung als formale Sprache bringt uns aber nur weiter, wenn wir die aufgeworfenen Probleme mit Info-III-Mitteln auch lösen können!

[1] Zweifler formalisieren noch einige weitere Probleme, wie "n ist die Summe zweier Quadratzahlen".

Was haben wir bis jetzt erreicht?

- Definition von **formalen Sprachen**.
- Reguläre Ausdrücke sind ein endliches Beschreibungsmittel für manche Sprachen.
- Eine Grammatik ist ein Kalkül zur Erzeugung formaler Sprachen: ein weiteres endliches Beschreibungsmittel.
- Ein Wort wird durch Ableitung von einer Grammatik erzeugt. Damit kann man — oft ganz einfach — zeigen, daß ein Wort zu einer Sprache gehört.
- Beweise über Ableitungsfolgen (strukturelle Induktion). Damit kann man — oft mit etwas Aufwand — zeigen, daß bestimmte Wörter nicht von einer Grammatik erzeugt werden.

Beispiel 1.23 (Grammatik-Ausschnitt: Dangling Else)

$S \rightarrow$ `if` E `then` S `else` S
$S \rightarrow$ `if` E `then` S
$S \rightarrow$ `other_stat` `//sonstige Anweisungen`
$E \rightarrow$ `b` `//boolescher Ausdruck`
`//Terminale: if, then, else, b, other_stat`

Leiten Sie das folgende Wort ab

`ifbthenifbthenother_statelseother_stat`

- Stellen Sie Ihre Ableitung als Strukturbaum dar.
- Wieviele Ableitungen gibt es?
- Warum ist das ein Problem?

Grammatiken sind die Grundlage für das Parsen von (Programmier-) Sprachen.

1.3 Die Chomsky-Hierarchie

Unsere Grammatik-Definition ist sehr allgemein!

Einzige Einschränkungen:

- nur endlich viele Regeln und
- jede Regelprämisse muss mindestens eine Variable enthalten.

Das lässt viel Flexibilität, führt aber leicht zu Wildwuchs![2]

Daher wollen wir jetzt die Regeln, die in einer Grammatik zulässig sind, beschränken.

Dann erhält man Grammatiktypen und damit auch Sprachklassen von verschiedenen Schwierigkeitsgraden.

[2] z.B.: Ein Wort kann im Lauf der Ableitung beliebig in der Länge wachsen und wieder schrumpfen.

Definition 1.24 (Sprachklassen)

Eine Grammatik $G = (V, T, R, S)$ heißt

rechtslinear, falls für alle $P \to Q \in R$:

$$\big(P \in V \text{ und } Q \in T^* \cup T^+V\big).$$

Das heißt:

- Es wird eine einzelne Variable ersetzt.
- Mit einer Regelanwendung wird höchstens eine Variable erzeugt.
- Wenn eine Regelanwendung eine Variable erzeugt, steht sie ganz rechts im Wort.

kontextfrei (cf) falls für alle $P \to Q \in R$:
$\big(P \in V$ und $Q \in (V \cup T)^*\big)$.

Das heißt:

- Es wird eine einzelne Variable ersetzt.
- Was links und rechts von der Variablen steht, kann man in der Prämisse nicht kontrollieren.
- Das Wort in der Conclusio kann Variablen und Terminale in beliebiger Mischung enthalten.

kontextsensitiv (cs), falls für alle $P \to Q \in R$ gilt

- **entweder** $\Big(\exists u, v, \alpha \in (V \cup T)^* \; \exists A \in V$ $\big(P = uAv$ und $Q = u\alpha v$ mit $|\alpha| \geq 1\big)\Big)$**, oder** die Regel hat die Form $S \to \varepsilon$
- S kommt in keiner Regelconclusio vor.

Das heißt:

- Eine Variable A wird in einen String α mit einer Länge von mindestens 1 überführt. **Das Wort wird nicht kürzer, außer $\varepsilon \in L$.**
- Diese Ersetzung von A durch α findet aber nur statt, wenn der in der Regel geforderte **Kontext**, links u und rechts v, im Wort **vorhanden** ist.

beschränkt, falls für alle $P \to Q \in R$ gilt:

- **entweder** $\big(|P| \leq |Q|\big)$ **oder** die Regel hat die Form $S \to \varepsilon$
- S kommt in keiner Regelconclusio vor.

Das heißt:

- Die Conclusio jeder Regel ist mindestens so lang wie die Prämisse (außer falls $\varepsilon \in L$).
- Das heißt, das Wort kann im Lauf der Ableitung nur wachsen, nie kürzer werden.

Aufbauend auf diesen Grammatikarten kann man nun Sprachklassen definieren:

Definition 1.25 (Sprachklassen)

Klasse	definiert als	Sprache heißt
$\mathbf{L_3}$, RAT	$\{L(G) \mid G$ ist rechtslinear$\}$	Typ 3, regulär
$\mathbf{L_2}$, CFL	$\{L(G) \mid G$ ist cf$\}$	Typ 2, kontextfrei
$\mathbf{L_1}$, CSL	$\{L(G) \mid G$ ist cs$\}$	Typ 1, kontextsensitiv
	$\{L(G) \mid G$ ist beschränkt$\}$	beschränkt
$\mathbf{L_0}$, r.e.	$\{L(G) \mid G$ beliebig$\}$	Typ 0, aufzählbar
$\mathbf{L}$	$\{L \mid L \subseteq \Sigma^* \}$	beliebige Sprache

Grammatiken können kompliziert sein!

Beispiel 1.26 ($a^n b^n c^n$)

Sei $G_{abc} = (\{S, X_1, X_2\}, \{a, b, c\}, \{R_1, \ldots, R_5\}, S)$ eine Grammatik mit

$$\begin{array}{lllll} R_1 = & S & \to & abc \mid aX_1bc \\ R_2 = & X_1b & \to & bX_1 \\ R_3 = & X_1c & \to & X_2bcc \\ R_4 = & bX_2 & \to & X_2b \\ R_5 = & aX_2 & \to & aa \mid aaX_1 \end{array}$$

- Ist diese Grammatik **kontextsensitiv**?
- Ist sie **beschränkt**?

1.4 Probleme über Sprachen

Oft interessiert uns die Frage, **ob ein gegebenes Wort in einer Sprache** (definiert durch eine Grammatik) **enthalten ist**. Oder ob zwei Grammatiken dieselbe Sprache erzeugen. Natürlich interessiert dann auch, ob es einen Algorithmus gibt, um solch ein Problem zu lösen. Das führt uns dazu, eine vorläufige, intuitive Definition für die beiden Begriffe **Problem** und **Algorithmus** zu geben.

Definition 1.27 (Problem, Algorithmus)

Ein **Problem P** ist eine Frage, ob eine bestimmte Eigenschaft auf gegebene Objekte zutrifft.
Die Objekte sind aus einer bekannten, abzählbaren Grundmenge. Ferner gilt für jedes Objekt o: die Eigenschaft trifft auf o zu oder nicht.
Ein **Algorithmus** für ein Problem P ist eine Vorschrift (ein Programm), die zu beliebigem Objekt o berechnet, ob die Eigenschaft für o zutrifft oder nicht.

Beispiel 1.28 (Einige Probleme)

- Ob ein Wort aus einer Grammatik ableitbar ist (Wortproblem), oder ob die zu einer Grammatik gehörende Sprache leer (endlich, unendlich) ist.
- Ob eine Zahl prim ist. Ob für gegebenes n die Gleichung $a^n + b^n = c^n$ eine Lösung in den natürlichen Zahlen hat.
- Ob ein Java-Programm auf eine Eingabe terminiert (Halteproblem).

1.5 Endlich, unendlich und dann?

Wie charakterisiert man die **Mächtigkeit einer Menge**?

- **Abzählen der Elemente** funktioniert **gut bei endlichen Mengen.**
- **Nur was machen wir mit $\mathbb{N}^2$ oder $\mathbb{R}$?**

Definition 1.29 (Mächtigkeit von Mengen, Kardinalität)

Zwei nichtleere Mengen M sind N sind **gleichmächtig** (haben die gleiche Kardinalität), geschrieben $|N| = |M|$, wenn es eine **bijektive Abbildung** $f : M \to N$ gibt.

Eine Menge M ist **mindestens so mächtig** wie eine Menge N, geschrieben $|N| \leq |M|$, wenn es eine **surjektive** Abbildung $f : M \to N$ gibt.

- Offensichtlich gilt auch mit dieser Definition: $|\{1, 2, 3\}| < |\{a, b, c, d\}|$ und $|\{a, aa, aaa, aaaa\}| < |\mathbb{N}|$

Mächtigkeit von Mengen

Was nützt uns die Funktion f?

- f systematisiert das Aufzählen der Elemente (s. Satz: 1.30)
- f hilft, wenn Aufzählen nicht funktioniert.
- f hilft in Fällen, wo $|N| \neq |M|$ gezeigt werden muss.

Betrachten Sie die beiden Intervalle von reellen Zahlen $[1, 4]$ und $[1, 100]$ graphisch in der Ebene. **Können Sie eine explizite Bijektion zwischen beiden Mengen angeben?**

Obwohl eine Menge eine echte Teilmenge der anderen ist, haben beide gleich viele Elemente!!!

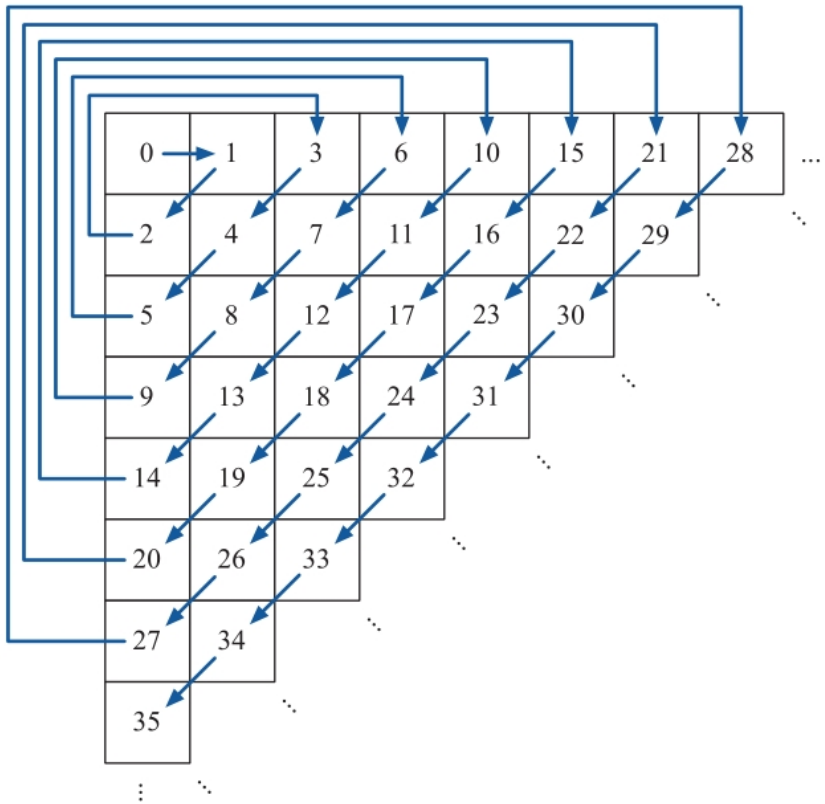


Abbildung 5: Cantorsche Paarung für $\mathbb{N}^2$, ©[Hoffmann2011]

Satz 1.30

$|\mathbb{N} \times \mathbb{N}| = |\mathbb{N}|$

Beweisidee: $\pi(x, y) = y + \frac{(x+y)(x+y+1)}{2}$

Wie charakterisiert man eine **endliche** Menge?

Definition 1.31 (Endlich, unendlich)

Dedekind: Eine Menge heißt **endlich**, wenn sie keiner ihrer echten Teilmengen gleichmächtig ist.

$\mathbb{N}_0$: Eine Menge heißt **endlich**, wenn sie zu einer natürlichen Zahl n gleichmächtig ist.

Hierbei ist $\mathbb{N}_0$ wie folgt definiert:

$$0 := \emptyset,\ 1 := \{0\},\ \ldots n := \{0, 1, \ldots, n-1\}, \ldots$$

Eine Menge die nicht endlich ist, heißt **unendlich**.

- Man betrachte Funktionen

$$f : \mathbb{N}_0 \to \mathbb{N}_0$$

  Welche sind, jedenfalls prinzipiell, **berechenbar**?
- Welche sind **schnell und effizient** zu berechnen?
- Was genau ist ein **Algorithmus**?

Wieviele Grammatiken (Algorithmen, Sprachen) gibt es überhaupt?

Mögliche Antworten:

1. Endlich viele.
2. Unendlich viele.
3. Soviele wie es natürliche Zahlen gibt.
4. Soviele wie es reelle Zahlen gibt.
5. Nicht klar für Algorithmen, da dieser Begriff nicht genau definiert wurde.

Σ^*: Sei Σ endlich oder abzählbar unendlich. Dann ist Σ^* abzählbar unendlich. Jede endliche Zeichenreihe kann als natürliche Zahl repräsentiert werden (Primfaktorzerlegung).

Algorithmen: Es gibt soviele Algorithmen wie natürliche Zahlen.

$f : \mathbb{N}_0 \to \{0, 1\}$: **Es gibt überabzählbar (mehr als abzählbar) viele solche Funktionen.**

Zu gegebenen $f_1, \ldots, f_n, \ldots$ sei

$$C : \mathbb{N}_0 \to \mathbb{N}_0; n \mapsto \begin{cases} 1, & \text{falls } f_n(n) = 0; \\ 0, & \text{sonst.} \end{cases}$$

C ist von allen f_i verschieden!

Teilmengen von $\mathbb{N}_0$: Es gibt überabzählbar viele, denn man kann sie identifizieren mit Funktionen $f : \mathbb{N}_0 \to \{0, 1\}$.

Also kann nicht jede Sprache durch eine Grammatik dargestellt werden.

Hilbert's Hotel: Hilbert's Hotel hat abzählbar unendlich viele Zimmer. Alle sind belegt.

$+1$: Können Sie noch einen weiteren Gast unterbringen?

$+\mathbb{N}$: Können Sie auch abzählbar viele zusätzliche Gäste unterbringen?

$+\mathbb{N} \times \mathbb{N}$: Nun kommen abzählbar viele Busse mit jeweils abzählbar vielen Fahrgästen. Wie bringen Sie diese unter?

https://www.youtube.com/watch?v=Uj3_KqkI9Zo

Lemma 1.32

Für abzählbare Alphabete Σ ist Σ^* abzählbar.

Beweis

Sei $\Sigma = \{a_1, a_2, a_3, \dots\}$ abzählbar unendlich.

Dann ordnen wir jedem (endlichen!!) Wort $a_{i_1} a_{i_2} a_{i_3} a_{i_4} \cdots \in \Sigma^*$ die folgende Zahl zu

$$2^{i_1} \cdot 3^{i_2} \cdot 5^{i_3} \cdot 7^{i_4} \dots \ .$$

Offenbar ist diese Abbildung injektiv (eindeutige Darstellung einer Zahl in ihre Primfaktoren), d.h. $|\Sigma^*| \leq |\mathbb{N}|$. $\square$

Lemma 1.33

1. $\mathbb{N}$ *ist* ***abzählbar***.
2. *Die Menge aller Teilmengen von* $\mathbb{N}$*, bezeichnet mit* $2^{\mathbb{N}}$*, ist* ***überabzählbar*** *(und lässt sich nicht auf* $\mathbb{N}$ *bijektiv abbilden).*
3. *Die Menge aller reellen Zahlen ist überabzählbar.*
4. *Die Menge aller Sprachen ist überabzählbar.*
5. *Folgende Mengen sind* ***abzählbar****:*
   - *Menge aller Grammatiken,*
   - *Menge aller Algorithmen,*
   - *Menge aller aus einem abzählbaren Alphabet gebildeten endlichen Zeichenreihen.*

Satz 1.34 (Cantor, 1874: $|\mathbb{R}| > |\mathbb{N}|$)

$[0,1] = \{r : 0 \leq r \leq 1, r \in \mathbb{R}\}$ *ist überabzählbar.*[a]

[a] Schon Euklid (ca. 300 B.C.) wußte, daß nicht-rationale Zahlen existieren. Lindemann zeigte dass π nicht algebraisch ist, aber erst Cantor zeigte, daß es echt mehr reelle als natürliche Zahlen gibt.

Beweis: (Cantor, 1877)

Diagonalisierung

Widerspruchsbeweis: Annahme: $[0,1]$ ist abzählbar.

Dann gibt es eine bijektive Funktion $f : \mathbb{N} \to [0,1]$. f bildet jede natürliche Zahl n eineindeutig auf eine reelle Zahl r in $[0,1]$ ab.

Wir zählen die Funktionswerte von f auf, beschränken uns dabei aber auf die Nachkommastellen:

Beweis: (Cantor, 1877)

f(0) = 0,	0	1	3	4	5	4	6	1	0	2	9	...
f(1) = 0,	2	1	2	5	8	7	9	9	2	4	0	...
f(2) = 0,	3	4	1	2	4	3	8	3	3	2	8	...
f(3) = 0,	2	1	4	4	5	4	5	2	0	3	5	...
f(4) = 0,	0	1	3	6	5	7	6	1	0	0	0	...
f(5) = 0,	2	1	5	5	8	6	5	9	2	4	0	...
f(6) = 0,	7	4	8	4	3	6	8	5	3	3	5	...
...												

Beweis: (Cantor, 1877)

Diagnonalisieren: Betrachte $r_{diag} = 0, d_0 d_1 d_2 d_3 d_4 d_5 \ldots$, wobei die Ziffer $d_i \in \{0, \ldots, 8\}$ jeweils so gewählt wird, daß sie **nicht** mit der $i+1$-ten Ziffer von $f(i)$ übereinstimmt [a]:

f(0) = 0,	**0**	1	3	4	5	4	6	1	0	2	...	$\mathbf{d_0 = 2}$
f(1) = 0,	2	**1**	2	5	8	7	9	9	2	4	...	$\mathbf{d_1 = 2}$
f(2) = 0,	3	4	**1**	2	4	3	8	3	3	2	...	$\mathbf{d_2 = 2}$
f(3) = 0,	2	1	4	**4**	5	4	5	2	0	3	...	$\mathbf{d_3 = 3}$
f(4) = 0,	0	1	3	6	**5**	7	6	1	0	0	...	$\mathbf{d_4 = 3}$
f(5) = 0,	2	1	5	5	8	**6**	5	9	2	4	...	$\mathbf{d_5 = 3}$
f(6) = 0,	7	4	8	4	3	6	**8**	5	3	3	...	$\mathbf{d_6 = 4}$
...												

[a] Ein(!) mögliches Beispiel in rot neben der Tabelle

Beweis: (Cantor, 1877)

f(0) = 0,	**0**	1	3	4	5	4	6	1	0	2	...	$\mathbf{d_0 = 2}$
f(1) = 0,	2	**1**	2	5	8	7	9	9	2	4	...	$\mathbf{d_1 = 2}$
f(2) = 0,	3	4	**1**	2	4	3	8	3	3	2	...	$\mathbf{d_2 = 2}$
f(3) = 0,	2	1	4	**4**	5	4	5	2	0	3	...	$\mathbf{d_3 = 3}$
f(4) = 0,	0	1	3	6	**5**	7	6	1	0	0	...	$\mathbf{d_4 = 3}$
f(5) = 0,	2	1	5	5	8	**6**	5	9	2	4	...	$\mathbf{d_5 = 3}$
f(6) = 0,	7	4	8	4	3	6	**8**	5	3	3	...	$\mathbf{d_6 = 4}$

...

$\mathbf{r_{diag} = 0, d_0 d_1 d_2 d_3 d_4 d_5 \ldots}$ kommt nicht in dieser Aufzählung nicht vor, formal: $r_{diag} \neq f(n)$ für alle $n \in \mathbb{N}$, **aber es gilt $\mathbf{r_{diag} \in [0,1]}$!**

Dann ist f nicht surjektiv! Also $|\mathbb{N}| < |[0,1]|$. $\square$

Anmerkung für genaue Beobachter:

Genau genommen, müssen wir noch gewährleisten, daß die nicht eindeutige Zahlendarstellung uns den Beweis nicht kaputt macht:

Problem: Reelle Zahlen mit abbrechender Dezimaldarstellung können alternativ als nichtabbrechende Dezimalzahl, die mit einer unendlichen Folge von Neunen enden, geschrieben werden. z.B. $0.5 = 0.4999999\ldots$.
In dem Beweis argumentieren wir über die Darstellung von r_{diag} und schließen, daß - weil die Darstellung von r_{diag} in der Auflistung fehlt - auch der Wert r_{diag} fehlt.

Also jetzt ganz genau: Wir machen unsere Liste der Funktionswerte von f eindeutig: Dort wird jede reelle Zahl auf die "kürzestmögliche" Art aufgeschrieben und für die Konstruktion der Zahl r_{diag} gegebenenfalls mit Nullen verlängert. Das ist - ohne Beschränkung der Allgemeinheit - möglich.

Bei der Wahl der Ziffern für r_{diag} verbieten wir die 9 als Ziffer zu wählen. Das ist - dank des Dezimalsystems mit 10 verschiedenen Ziffern - immer möglich, und garantiert, daß keine unendliche Neunerfolge entstehen kann, d.h. r_{diag} ist keine dieser "alternativen" Darstellungen (mit Neunerfolge) einer abbrechenden Dezimalzahl.

Damit gilt endgültig: Wenn die Darstellung von r_{diag} in der Liste fehlt, dann kommt der Wert r_{diag} nicht als Funktionswert von f vor.

Gleich nochmal:

Satz 1.35 (Cantor: $|2^M| > |M|$)

Für jede nichtleere Menge M ist die Potenzmenge[a] , genannt 2^M, mächtiger als M.

[a] Die Potenzmenge von M ist die Menge aller Teilmengen von M. Sie kann auch als die Menge aller charakteristischen Funktionen $f : M \to \{0, 1\}$ verstanden werden.

Beweis: (Cantor)

Annahme: Nehmen wir an, daß $|2^M| = |M|$ für eine Menge M gilt.

Dann gibt es eine bijektive Funktion $f : M \to 2^M$. f bildet jedes Element $m \in M$ eineindeutig auf eine Teilmenge $f(m) \subseteq M$ ab.

Beweis: (Cantor)

Diagnonalisieren:

Nun gibt es sicher m, für die $m \in f(m)$ gilt und solche, für die $m \notin f(m)$ gilt. Sei

$$Diag\_Set = \{m \in M \mid m \notin f(m)\} \subseteq M.$$

f war bijektiv, d.h. auch für $Diag\_Set$ gibt es ein m_{diag}, sodaß $f(m_{diag}) = Diag\_Set$ gilt.
Es stellt sich die Frage: $m_{diag} \in Diag\_Set$?

$$m_{diag} \in Diag\_Set \Rightarrow m_{diag} \notin f(m_{diag}) \Rightarrow m_{diag} \notin Diag\_Set$$
$$m_{diag} \notin Diag\_Set \Rightarrow m_{diag} \in f(m_{diag}) \Rightarrow m_{diag} \in Diag\_Set$$

Es gibt also kein bijektives $f : M \to 2^M$. Also $|2^M| > |M|$. □

Konsequenz: Es gibt keine "maximale" Unendlichkeit.

Diagonalisieren

(engl.: dovetailing - verzahnen)

Beschreibung eines Elements, das

- auf die Definition der Funktion f Bezug nimmt
- aber bei jedem Vergleichselement "auf der Diagonale" abweicht.

Verzahnung von

- **Referenz auf die Definition**
- **und Anwendung der Definition**

= Selbstreferenz.

Konsequenz:

Satz 1.36 (Existenz nichtberechenbarer Funktionen)

Es muss also Sprachen geben, die nicht mit einer Grammatik erzeugt werden können.

Es muss auch Probleme und Funktionen geben, für die es keinen Algorithmus gibt, die also nicht berechenbar sind.

Die Beweise beruhen (nur) auf Kardinalitätsargumenten.
D.h. noch haben wir keine nichtberechenbare Funktion "gesehen", dürfen also noch hoffen, daß uns die nichtberechenbaren Probleme auch nicht interessieren.

2. Endliche Automaten (reg)

2 Endliche Automaten (reg)

- DEAen (e.a.)
- NEAs (nd e.a.)
- Automaten mit ε-Kanten (ε nd. e.a.)
- rational = Typ 3
- Pumping Lemma
- Wortprobleme
- Rational = Regulär

Inhalt (1)

In diesem Kapitel führen wir den endlichen Automaten ein: ein vereinfachtes Modell eines Computers. Wir zeigen,

1. daß die von endlichen Automaten erkannten Sprachen genau die vom Grammatik-Typ 3 (rechtslinear) sind;
2. daß deterministische und indeterminierte endliche Automaten äquivalent sind;

Inhalt (2)

3. ein Kriterium, das **Pumping Lemma**, das es erlaubt, eine Sprache als nicht rational nachzuweisen;
4. daß es Algorithmen gibt, um **Probleme über endlichen Automaten**, bzw. Typ 3 Sprachen zu lösen;
5. daß Typ 3 Sprachen genau die sind, die durch **reguläre Ausdrücke** beschrieben werden können.

2.1 DEAen (e.a.)

- Die Sprache $L = \{aa\}\{ab\}^*\{c\}$ ist regulär und wird erzeugt von der Grammatik

$$G = (\{S, A\}, \{a, b, c\}, R, S),$$

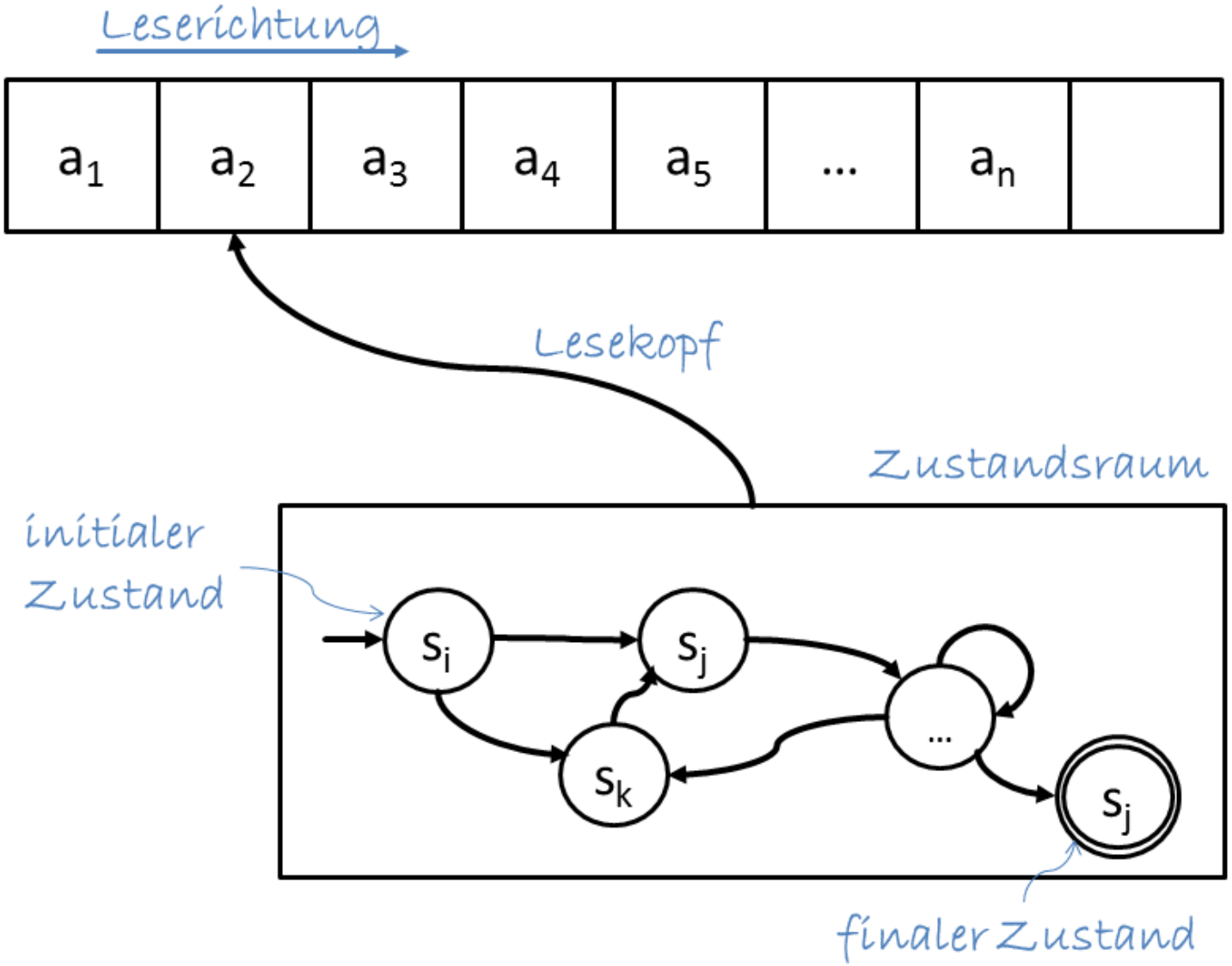
wobei R aus folgenden Regeln besteht:

$$S \to aaA$$
$$A \to abA \mid c$$

- Auch die Sprache aller durch 3 teilbaren Dezimalzahlen ist regulär. Eine erzeugende Grammatik ist $G = (\{S, S_0, S_1, S_2\}, \{0, \ldots, 9\}, R, S)$ mit der folgenden Regelmenge R:

$$
\begin{aligned}
S &\to 3S_0 \mid 6S_0 \mid 9S_0 \mid 1S_1 \mid 4S_1 \mid 7S_1 \mid 2S_2 \mid 5S_2 \mid 8S_2 \mid 0\\
S_0 &\to 0S_0 \mid 3S_0 \mid 6S_0 \mid 9S_0 \mid 1S_1 \mid 4S_1 \mid 7S_1 \mid 2S_2 \mid 5S_2 \mid 8S_2 \mid \varepsilon\\
S_1 &\to 0S_1 \mid 3S_1 \mid 6S_1 \mid 9S_1 \mid 1S_2 \mid 4S_2 \mid 7S_2 \mid 2S_0 \mid 5S_0 \mid 8S_0\\
S_2 &\to 0S_2 \mid 3S_2 \mid 6S_2 \mid 9S_2 \mid 1S_0 \mid 4S_0 \mid 7S_0 \mid 2S_1 \mid 5S_1 \mid 8S_1
\end{aligned}
$$

Ohne das ε in der zweiten Regel wäre nur die "0" als Terminalwort herleitbar. In der ersten Zeile könnte man auch $0S_0$ schreiben (an der letzten Stelle).


Leserichtung
a1
a2
a3
a4
a5
...
an
Lesekopf
Zustandsraum
initialer Zustand
Si
Sj
Sk
...
Sj
finaler Zustand

Ein endlicher Automat:

- Gegeben $w \in \Sigma^*$, ein endlicher Automat testet, ob $w \in L$ gilt.
- Ein endlicher Automat hat:
  - einen **Lesekopf**, um w zu lesen. Den kann er nur von links nach rechts bewegen.
  - einen **internen Zustand**, mit endlich vielen Werten
- Er fängt an in einem **initialen Zustand**.
- Bei jedem Buchstaben, den er liest, ändert er seinen Zustand.
- Wenn er am Ende von w in einem **finalen Zustand** ist, sagt er "ja". Das heißt, er **akzeptiert** w: w ist in L. Sonst sagt er "nein".
- **Er stoppt auf jeden Fall** nach $|w|$ Schritten.

Darstellung als Graph:

- ein **Knoten** für jeden möglichen **Zustand**,
- **Kanten** sind mit Buchstaben beschriftet: Sie beschreiben **Zustandsänderungen**.
- **Initiale** Zustände werden mit einem **Pfeil** gekennzeichnet,
- **finale Zustände** mit einem **doppelten Kreis**.

Bemerkung zur Definition des endlichen Automaten

In einem ea ist δ eine totale Funktion von $K \times \Sigma \to K$, also ist für jedes Paar aus Zustand und Symbol, ein Nachfolgezustand definiert.

Wenn δ eine beliebige, d.h. möglicherweise partielle Funktion von $K \times \Sigma \to K$ ist, dann kann es Paare (q, a) geben, für die kein Nachfolger definiert ist, also $\delta(q, a) = \perp$ (undefiniert). Wir setzen $\delta^*(\perp, a) = \perp$ für alle $a \in \Sigma$.

Wenn man beim Lesen eines Wortes w in einen Zustand kommt, in dem kein Nachfolgezustand für den aktuellen Buchstaben a definiert ist, also der Fall $\delta(q, a) = \perp$, dann kann das Wort nicht zu Ende gelesen werden, und die Verarbeitung im Endzustand enden, also $w \notin L(\mathrm{A})$. **Solche Automaten behandeln wir später: indeterminierte ea'en.**

Definition 2.1 (Endlicher Automat)

Ein endlicher Automat (e.a. oder auch DEA) (finite automaton) ist ein Tupel $A = (K, \Sigma, \delta, s_0, F)$. Dabei ist

- K eine endliche Menge von **Zuständen**,
- Σ ein **endliches Alphabet** (aus dessen Buchstaben die Eingabewörter bestehen können),
- $\delta : K \times \Sigma \to K$ die (totale) Übergangsfunktion,
- $s_0 \in K$ der **Startzustand**, und
- $F \subseteq K$ die Menge der **finalen Zustände**.

$\delta(q, a) = q'$ bedeutet:

- Der Automat ist im Zustand q,
- liest ein a und
- geht in den Zustand q' über.

Wir erweitern δ zu δ^*:
$\delta^* : K \times \Sigma^* \to K$ ist strukturell rekursiv über Σ^* definiert:

$$\begin{aligned} \delta^*(q, \varepsilon) &:= q \\ \delta^*(q, wa) &:= \delta(\delta^*(q, w), a) \end{aligned}$$

Wenn klar ist, was gemeint ist, wird δ^* auch einfach als δ geschrieben.

Beispiel 2.2

Die Sprache $\{a^{2n} \mid n \in \mathbb{N}_0\}$ über dem Alphabet $\{a, b\}$ wird akzeptiert von dem endlichen Automaten $A = (\{s_0, s_1, s_2\}, \{a, b\}, \delta, s_0, \{s_0\})$ mit

$$\delta(s_0, a) = s_1 \qquad \delta(s_1, a) = s_0 \qquad \delta(s_2, a) = s_2$$
$$\delta(s_0, b) = s_2 \qquad \delta(s_1, b) = s_2 \qquad \delta(s_2, b) = s_2$$

Ein Beispiel für δ^*:

$$
\begin{aligned}
\delta^*(s_0, aab) &= \delta\big(\delta^*(s_0, aa), b\big) \\
&= \delta\big(\delta\big(\delta^*(s_0, a), a\big), b\big) \\
&= \delta\big(\delta\big(\delta(\delta^*(s_0, \varepsilon), a\big), a\big), b\big) \\
&= \delta\big(\delta\big(\delta(s_0, a), a\big), b\big) \\
&= \delta\big(\delta(s_1, a), b\big) \\
&= \delta(s_0, b) \\
&= s_2
\end{aligned}
$$

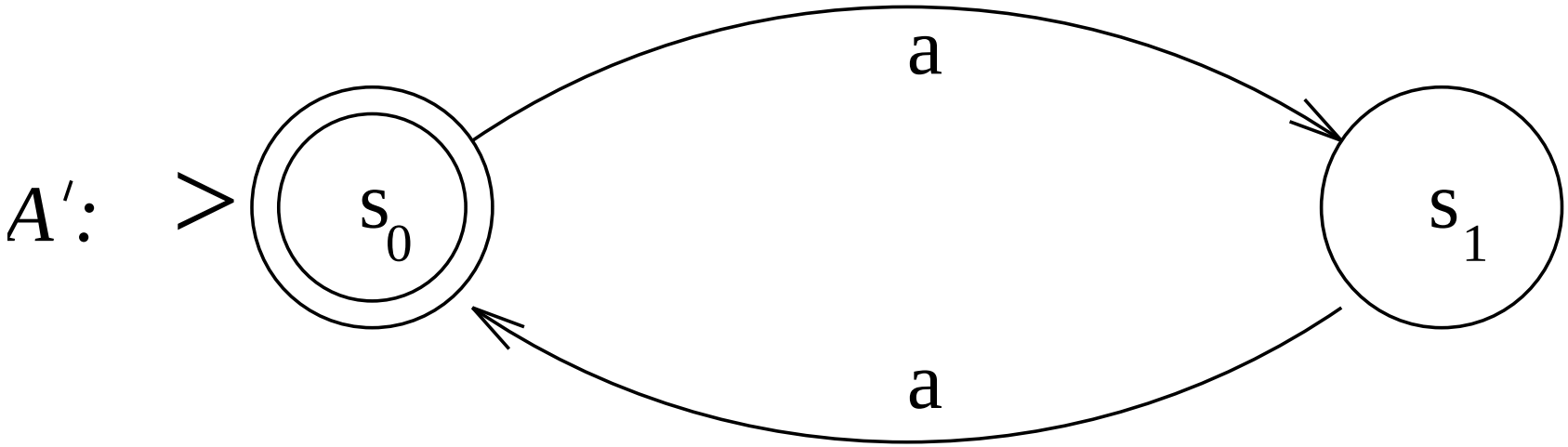


Abbildung 6: Eine Variante des Automaten aus Beispiel 2.2 mit gleicher Sprache, aber kleinerem Alphabet

Definition 2.3 (Von einem e.a. akzeptierte Sprache)

$L(A)$, die von einem Automaten A akzeptierte Sprache, ist definiert als

$$L(A) := \{w \in \Sigma^* \mid \delta^*(s_0, w) \in F\}.$$

Die Menge der von endlichen Automaten akzeptierten Sprachen ist

RAT $:= \{L \mid$ es gibt endlichen Automaten A, mit $L = L(A)\}$

und heißt die Menge der rationalen Sprachen.

Wir zeigen bald, daß dies genau die Menge der durch rechtslineare Grammatiken dargestellten Sprachen ist. Am Ende dieses Kapitels zeigen wir:

- **RAT** besteht genau aus den regulären Sprachen, also aus den durch reguläre Ausdrücke dargestellten Sprachen.

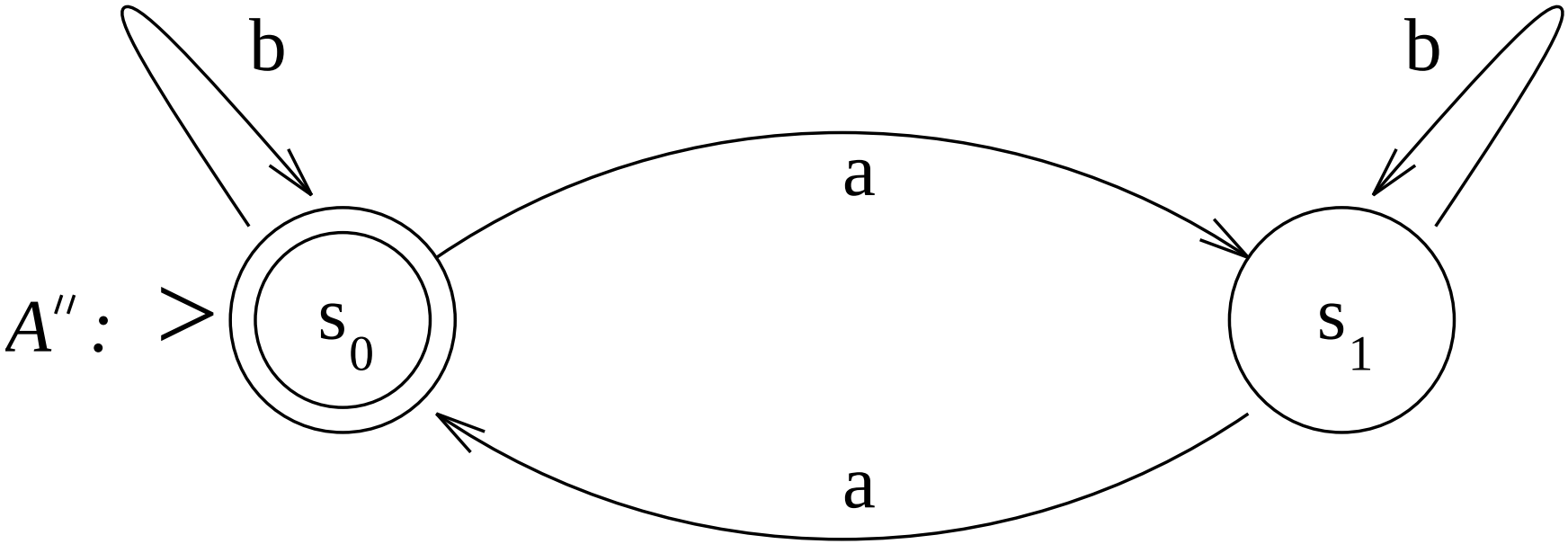


Abbildung 7: A'' akzeptiert Wörter, die eine gerade Anzahl von "a"s haben

Beispiel 2.4

Der Automat A'' in Abb. 7 akzeptiert $L(A'') = \{w \in \{a,b\}^* \mid \exists n : \ \#_a(w) = 2n\}$. Dabei ist $\#_a(w)$ die Anzahl der "a"s in w.

Zum Beispiel würde A'' das Wort $bbaabaab$ wie folgt akzeptieren:

$$\underset{\in F}{s_0} \xrightarrow{b} \underset{\in F}{s_0} \xrightarrow{b} \underset{\in F}{s_0} \xrightarrow{a} s_1 \xrightarrow{a} \underset{\in F}{s_0} \xrightarrow{b} \underset{\in F}{s_0} \xrightarrow{a} s_1 \xrightarrow{a} \underset{\in F}{s_0} \xrightarrow{b} \underset{\in F}{s_0}$$

Beispiel 2.5

$L = \{w \in \{0,1\}^* \mid w$ enthält genau zwei Einsen $\}$ wird akzeptiert von dem endlichen Automaten in Abb. 8.

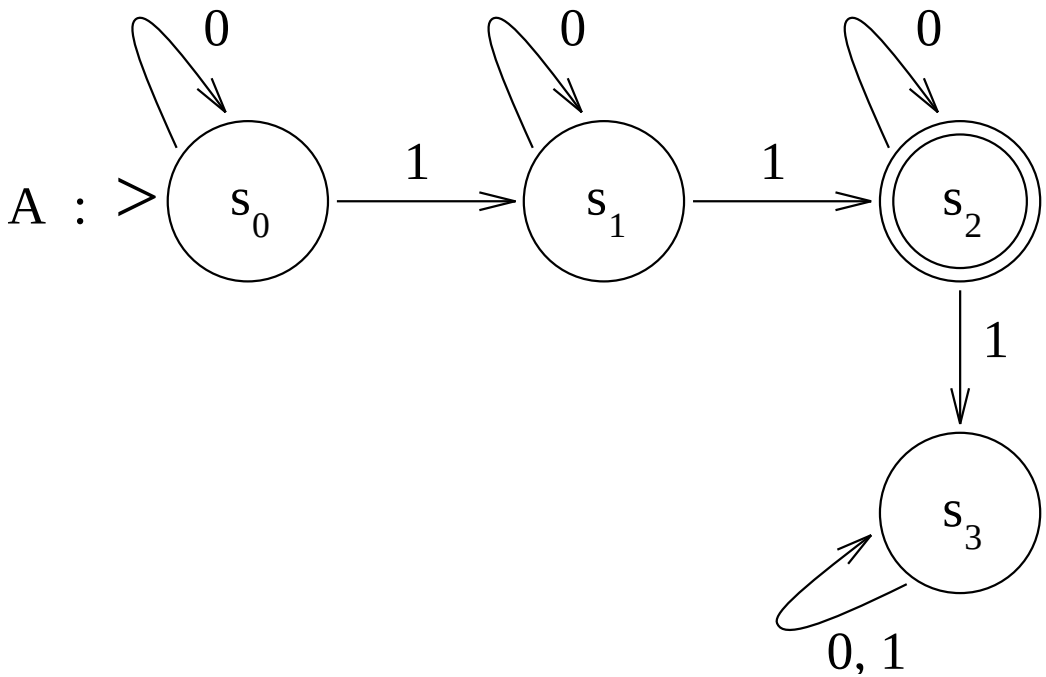


Abbildung 8: Dieser Automat akzeptiert Wörter, mit genau 2 Einsen

Beispiel 2.6

Der endliche Automat in Abb. 9 akzeptiert die Sprache aller durch 3 teilbaren Dezimalzahlen.

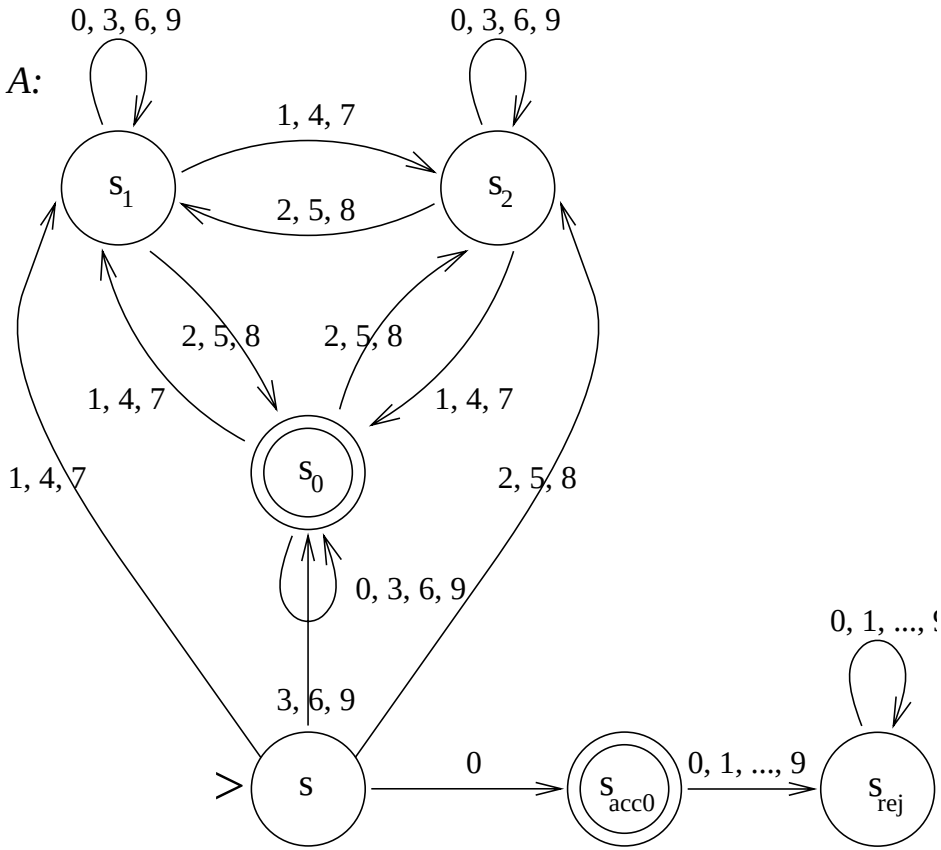


Abbildung 9: Ein endlicher Automat für die durch 3 teilbaren Dezimalzahlen

2.2 NEAs (nd e.a.)

Determinierte endliche Automaten:

- in Zustand q bei Eingabe a **ein einziger** Nachfolgezustand
- festgelegt durch Übergangs**funktion** δ.

Indeterminierter endlicher Automat:

- in Zustand q bei Eingabe a evtl. mehrere Nachfolgezustände – oder gar keiner.
- Übergangsrelation Δ.

Definition 2.7 (Indeterminierter endlicher Automat)

Ein **indeterminierter** endlicher Automat (nd e.a.) A ist ein Tupel $A = (K, \Sigma, \Delta, I, F)$. Dabei ist

- K eine endliche Menge von Zuständen,
- Σ ein endliches Alphabet,
- $\Delta \subseteq (K \times \Sigma) \times K$ eine Übergangs**relation**,
- $I \subseteq K$ eine Menge von Startzuständen und
- $F \subseteq K$ eine Menge von finalen Zuständen.

$\Delta^* \subseteq (K \times \Sigma^*) \times K$ ist definiert wie folgt:

$$(q, \varepsilon) \ \Delta^* \ q' \quad \underline{\text{gdw}} \quad q' = q$$

$$(q, wa) \ \Delta^* \ q' \quad \underline{\text{gdw}} \quad \exists q'' \in K \ \big((q, w) \ \Delta^* \ q'' \text{ und } (q'', a) \ \Delta \ q'\big)$$

Schreibweisen:

- statt $(q, w)\ \Delta^*\ q'$ auch $\big((q, w), q'\big) \in\ \Delta^*$ oder $q' \in\ \Delta^*(q, w)$.
- Δ^* wird auch als Δ abgekürzt, wenn es im Kontext nicht zu Verwechslungen führt.

Wann akzeptiert ein indeterminierter Automat ein Wort?

Ein nd e.a. A akzeptiert ein Wort w, wenn

- es **mindestens einen** Weg mit der **Beschriftung** w durch A gibt,
- der in einem **finalen** Zustand endet.

Definition 2.8 (Von nd e.a. akzeptierte Sprache)

Die von einem indeterminierten endlichen Automaten A akzeptierte Sprache ist

$$L(A) := \{w \in \Sigma^* \mid \exists s_0 \in I \ \exists q \in F \ (s_0, w)\Delta^* q\}$$

Ein nd e.a. kann lügen: er kann ein Wort nicht akzeptieren (also "nein" sagen), obwohl es einen anderen Weg gibt, der es akzeptiert. Wenn er allerdings "ja" sagt, dann lügt er nicht.

Wie kann man sich Indeterminismus vorstellen?

- Determinierte endliche Automaten:
  Algorithmus für das Wortproblem ablesbar.
- Indeterminierte endliche Automaten:
  Alg.= Automat plus **Suchstrategie.**
- Um zu prüfen, ob ein Wort w akzeptiert wird, müssen alle Wege mit **Beschriftung** w durch den Automaten getestet werden.
- Der ganze Suchbaum muss durchlaufen werden.

- Zwei Sichtweisen auf indet. Automaten:
  - Der Automat durchläuft alle Wege **parallel**.
  - Der Automat **rät**, welchen von mehreren möglichen Folgezuständen er wählt.

Automat $A = (\{s_0, s_1, s_2\}, \{a, b\}, \Delta, s_0, \{s_0\})$ mit

$$\begin{aligned} \Delta(s_0, a) &= \{s_1\} \\ \Delta(s_1, b) &= \{s_0, s_2\} \\ \Delta(s_2, a) &= \{s_0\} \end{aligned}$$

Darstellung als Graph:

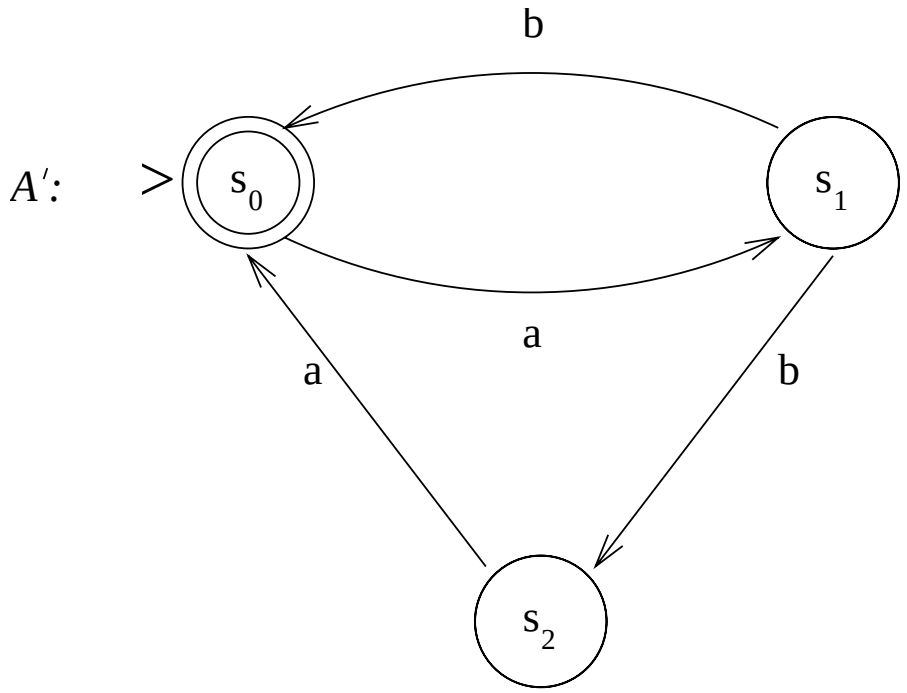


Abbildung 10: Ein indeterminierter Automat

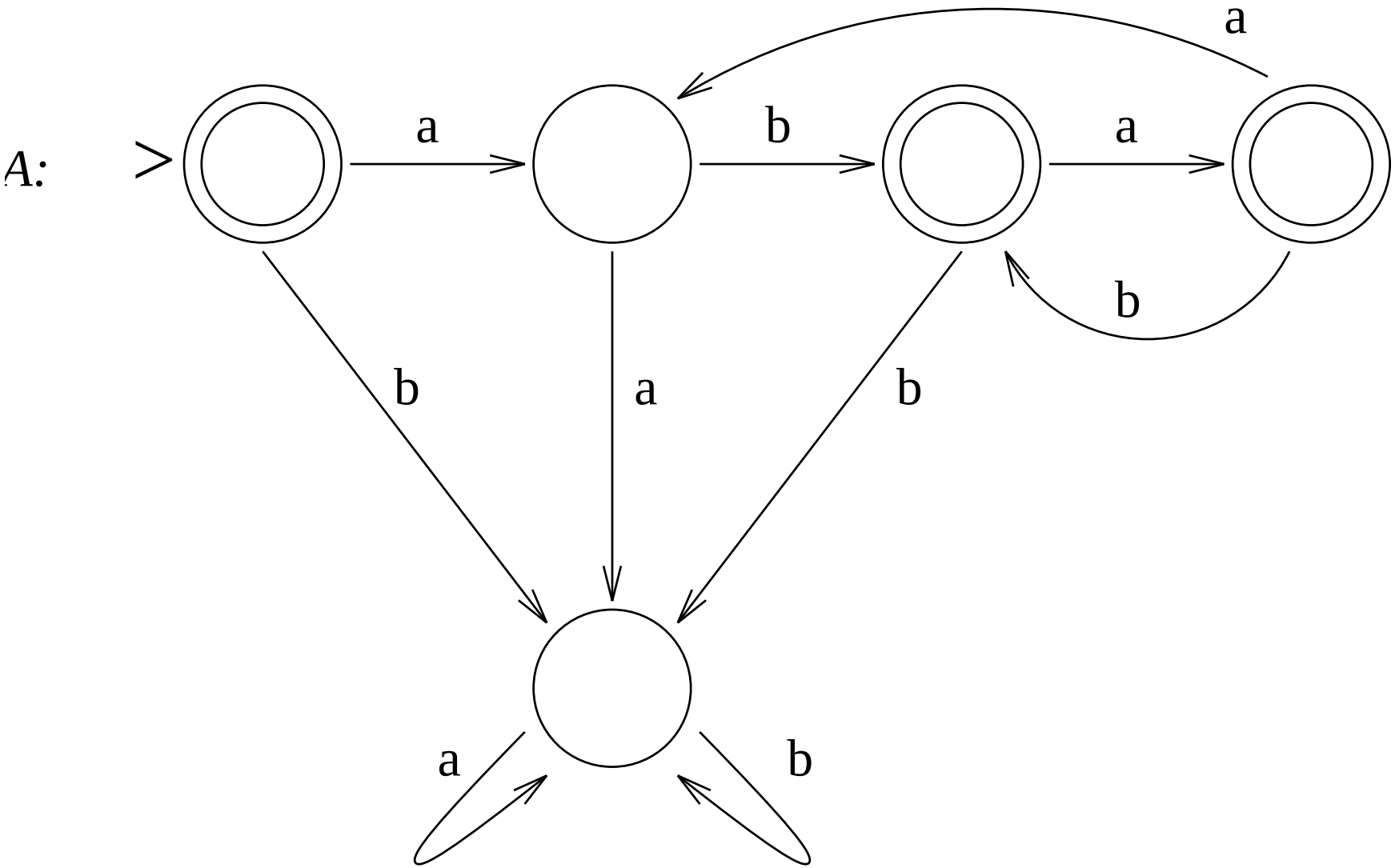


Abbildung 11: Ein deterministischer Automat für $L = \{ab, aba\}^*$

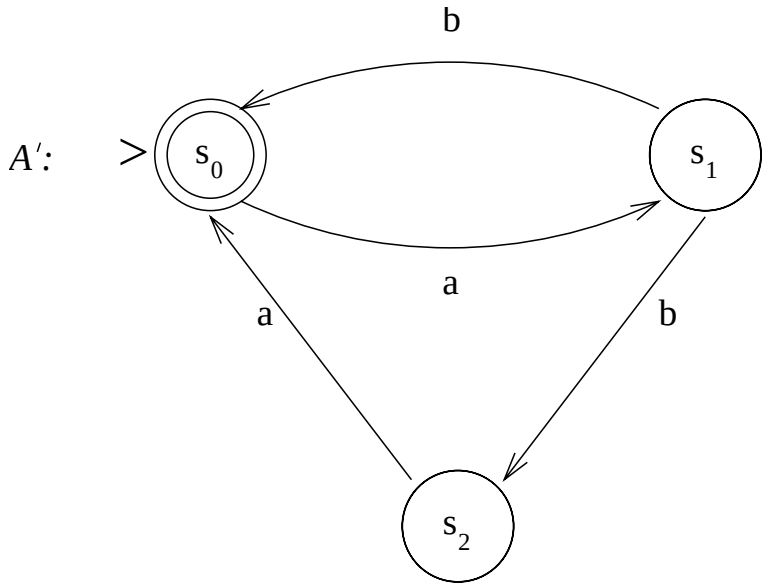


Abbildung 12: Ein indeterminierter Automat für $L = \{ab, aba\}^*$

Die Sprache $L = \{ab, aba\}^*$ wird

- von dem deterministischen Automaten A aus Abb. 11 und
- von dem indeterminierten Automaten A' aus Abb. 12

akzeptiert.

$$L = \{a, b\}^* \{a\} \{a, b\}$$

ist die Sprache aller Wörter über $\{a, b\}$, deren zweitletzter Buchstabe ein a ist.

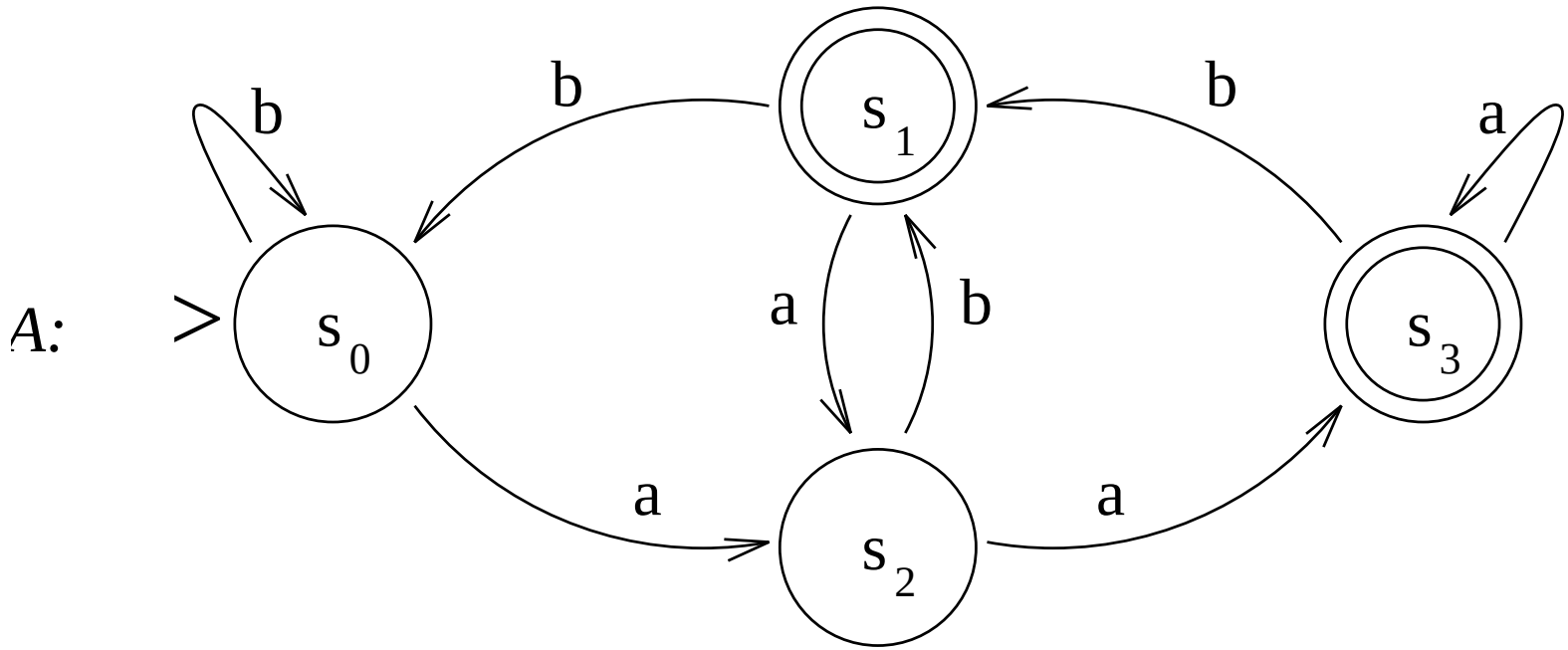


Abbildung 13: Ein deterministischer Automat für $L = \{a, b\}^* \{a\} \{a, b\}$

Idee:

- im Zustand jeweils die letzten zwei Buchstaben merken
- wichtig: **war der vorletzte Buchstabe ein “a”?**
- endlich viele Zustände:
  Die Anzahl der Buchstaben “im Speicher” steht fest—es sind immer höchstens 2.
  Deshalb kann ein endlicher Automat die Sprache L akzeptieren.

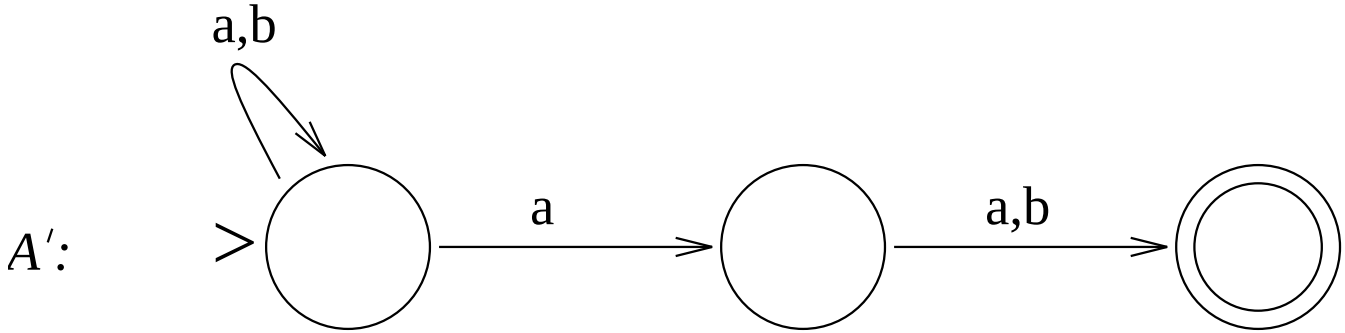


Abbildung 14: Ein indeterminierter Automat für $L = \{a, b\}^*\{a\}\{a, b\}$

Größenvergleich: Sprache über $\{a, b\}$ der Wörter, deren n-t-letzter Buchstabe ein "a" ist:

- **deterministischer** Automat: $\mathbf{2^n}$ Zustände, einen für jede Buchstabenkombination der Länge n
- **indeterminierter** Automat: $\mathbf{n + 1}$ Zustände

Ausführliche Darstellung an der Tafel.

Ein indeterminierter endlicher Automat heißt...

deterministisch gdw $\forall a \in \Sigma \, \forall q \in K \; |\Delta(q,a)| \leq 1$

vollständig gdw $\forall a \in \Sigma \, \forall q \in K \; |\Delta(q,a)| \geq 1$

Lemma 2.9

*Ein endlicher Automat (**e.a.**) ist ein vollständiger und deterministischer indeterminierter endlicher Automat (**nd. e.a.**).*

Oft sagt man auch **nichtdeterministischer** Automat. Damit ist aber **nicht** gemeint, daß der Automat auf jeden Fall nicht deterministisch ist: es wird nur nicht gefordert, daß er deterministisch sein muss.

Theorem 2.10 (ea gleich mächtig wie nd ea)

*Eine Sprache ist **rational** (wird von einem ea akzeptiert) gdw sie von einem **indeterminierten ea akzeptiert** wird.*

Beweis.

"$\Rightarrow$"

- Sei L eine rationale Sprache.
- Dann gibt es laut Definition einen endlichen Automaten A mit $L = L(A)$.
- Jeder endliche Automat ist aber schon ein (vollständiger und deterministischer) indeterminierter endlicher Automat.

□

"$\Leftarrow$"

- Sei $A = (K, \Sigma, \Delta, I, F)$ ein indeterminierter endlicher Automat, der die Sprache $L(A)$ akzeptiert.
- Dann kann man aus A einen deterministischen Automaten A' konstruieren mit $L(A) = L(A')$ mit Hilfe einer **Potenzmengenkonstruktion**:
  In A kann es zu einem Zustand q und einem gelesenen Zeichen a mehrere mögliche Folgezustände geben, die alle quasi parallel beschritten werden.
- Wir konstruieren einen **Zustand von A'** als eine **Menge von Zuständen von A**:
  - Gelangt man mit einem Eingabewort w in A indeterminiert in einen der Zustände q_1 bis q_n,
  - so gelangt man mit w in A' in einen Zustand $q' = \{q_1, \ldots, q_n\}$.

Zunächst ein konkretes Beispiel:

$L_{aab/aba} = \{w \in \{a,b\}^* \mid w$ hat aab oder aba als Teilwort $\}$
wird akzeptiert von folgendem nd. e.a.:

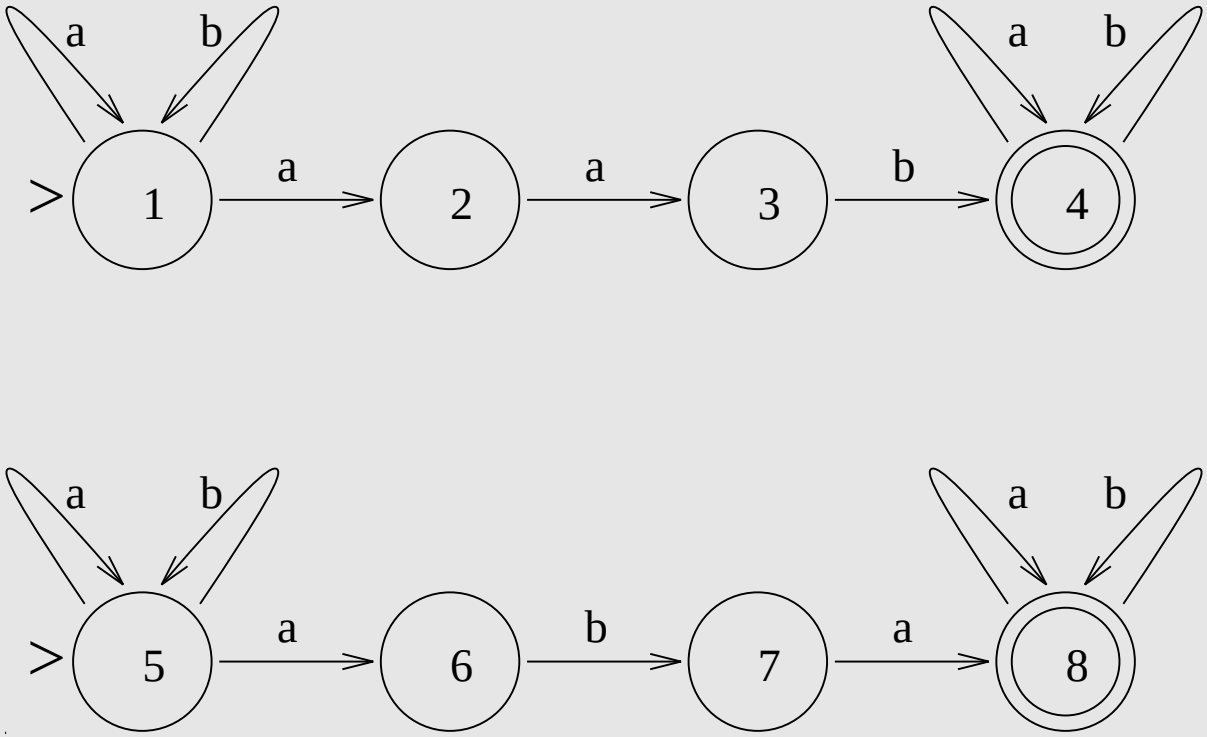


Abbildung 15: Indeterminierter endlicher Automat für $L_{aab/aba}$

Neuer Startzustand: Menge der alten Startzustände, also $\{1, 5\}$.
Nächster Schritt: Übergang von $\{1, 5\}$ mit a.

$$\begin{aligned}\Delta(1, a) &= \{1, 2\}\\ \Delta(5, a) &= \{5, 6\}\end{aligned}$$

Also, neuer Zustand $\{1, 2, 5, 6\}$ mit

$$\delta_{A'}(\{1, 5\}, a) = \{1, 2, 5, 6\}$$

Nächster Schritt: Übergang von $\{1, 5\}$ mit b.

$$\begin{aligned}\Delta(1, b) &= \{1\}\\ \Delta(5, b) &= \{5\}\end{aligned}$$

Für den Eingabebuchstaben b bleibt A' also im Startzustand.
Nächster Schritt: Übergang von $\{1,2,5,6\}$ mit a.

$$\begin{aligned}\Delta(1,a) &= \{1,2\}\\ \Delta(2,a) &= \{3\}\\ \Delta(5,a) &= \{5,6\}\\ \Delta(6,a) &= \emptyset\end{aligned}$$

Also, neuer Zustand $\{1,2,3,5,6\}$ mit

$$\delta_{A'}(\{1,2,5,6\},a) = \{1,2,3,5,6\}$$

Finale Zustände von A' sind die, die mindestens einen finalen Zustand von A enthalten.

Damit gilt:

- Wenn man mit dem Eingabewort w in einen Zustand von A' kommt, der einen finalen Zustand von A enthält,
- dann gibt es in A eine Rechnung, so daß mit w ein finaler Zustand erreicht wird.
- Damit ist $w \in L(A)$ nach der Definition der Sprache eines indeterminierten endlichen Automaten.

Für den nd e.a. aus dem Beispiel ergibt sich im Endeffekt:

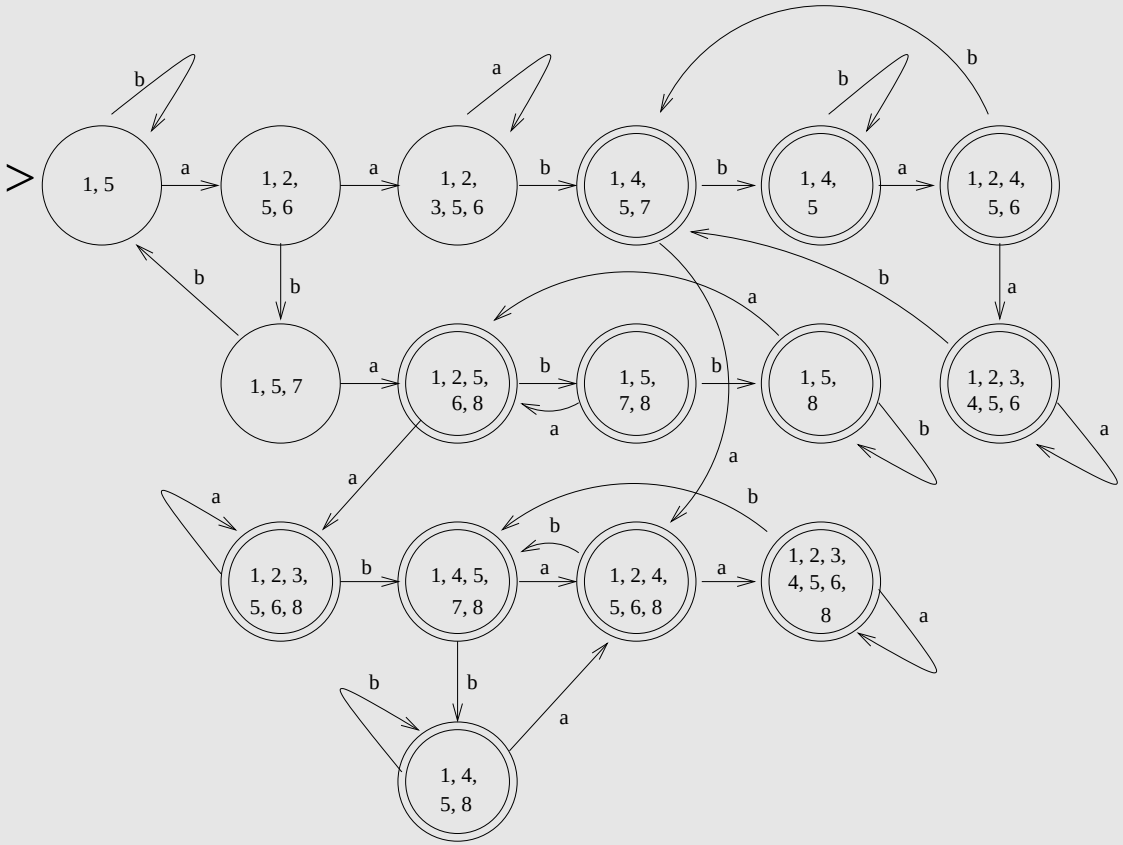


Abbildung 16: Determinierter endlicher Automat für $L_{aab/aba}$

Diesen Automaten kann man noch vereinfachen.

Wenn $\mathcal{A}'$ einen Zustand $\{q_1, \ldots, q_n\}$ hat, so daß es für a keinen Übergang in $\mathcal{A}$ gibt, für keinen der Zustände q_1 - q_n:

- Das Eingabewort wird von hier aus auf keinen Fall mehr akzeptiert.
- $\mathcal{A}'$ enthält dann einen Zustand $\emptyset$, einen **Abseitszustand**.
- Alle Übergänge von dort führen wieder in den Zustand $\emptyset$ zurück.

Konstruktion des deterministischen endlichen Automaten A' formal:

- Sei $A = (K, \Sigma, \Delta, I, F)$ ein indeterminierter endlicher Automat, der $L(A)$ akzeptiert.
- Dann gibt es einen deterministischen endlichen Automaten A' mit $L(A) = L(A')$ ist.
- Die Übergangsfunktion von A' ist Abbildung $\hat{\Delta} : 2^K \times \Sigma \to 2^K$ mit $\hat{\Delta}(M, a) := \bigcup_{q \in M} \Delta(q, a)$.

  $\hat{\Delta}^*$ sei als Erweiterung von $\hat{\Delta}$ auf mehrere Schritte definiert gemäß der allgemeinen Definition von δ^*.

Hilfsüberlegung: Es ist $\hat{\Delta}^*(M,w) = \bigcup_{q\in M} \Delta^*(q,w)$. Das beweisen wir durch Induktion über die Länge von w:

Induktionsanfang: $\hat{\Delta}^*(M,\varepsilon) = M = \bigcup\limits_{q\in M} \{q\} = \bigcup\limits_{q\in M} \Delta^*(q,\varepsilon)$

Induktionsschritt:

$$
\begin{aligned}
& \hat{\Delta}^*(M,wa) \\
= \;& \hat{\Delta}(\hat{\Delta}^*(M,w),a) && \text{allg. Def. von } \hat{\Delta}^* \text{ aus 2.1} \\
= \;& \textstyle\bigcup_{p\in\hat{\Delta}^*(M,w)} \Delta(p,a) && \text{Definition von } \hat{\Delta} \\
= \;& \textstyle\bigcup_{p\in \bigcup\limits_{q\in M} \Delta^*(q,w)} \Delta(p,a) && \text{Ind.-Vor. für } \hat{\Delta}(M,w) \\
= \;& \{q' \mid \exists q\in M\ \exists p\in\Delta^*(q,w)\ q'\in\Delta(p,a)\} \\
= \;& \{r \mid \exists q\in M\ r\in\Delta^*(q,wa)\} && \text{allg. Def. von } \Delta^* \text{ aus 2.1} \\
= \;& \textstyle\bigcup_{q\in M} \Delta^*(q,wa)
\end{aligned}
$$

Sei nun $A' = (K', \Sigma, \delta', s_0', F')$ mit

- $K' := 2^K$, $\delta' := \hat{\Delta}$,
- $s_0' := I$, und $F' := \{M \subseteq K \mid M \cap F \neq \emptyset\}$.

Dann gilt: $w \in L(A')$

gdw $(\delta')^*(s_0', w) \in F'$ (Definition der Sprache eines Automaten)

gdw $\hat{\Delta}^*(I, w) \in F'$ (Definition von δ' und da $s_0' = I$)

gdw $\hat{\Delta}^*(I, w) \cap F \neq \emptyset$ (Definition von F')

gdw $\bigcup_{q \in I} \Delta^*(q, w) \cap F \neq \emptyset$ (nach Hilfsüberlegung)

gdw $\exists q \in I \, \exists q' \in F \, \left(q' \in \Delta^*(q, w)\right)$

gdw $w \in L(A)$

2.3 Automaten mit ε-Kanten (ε nd. e.a.)

- bisher: Kanten eines Automaten waren jeweils mit einem *Buchstaben* beschriftet
- jetzt: **Kanten, die mit einem Wort beschriftet sind.**
- Es darf auch das leere Wort ε sein!
- **Ein Automat mit ε-Kanten:**
  - kann in einem Schritt ein ganzes Wort verarbeiten,
  - kann einen Zustandsübergang machen, ohne dabei einen Eingabebuchstaben zu lesen.

Definition 2.11 (Automat mit ε-Kanten)

Ein **Automat mit ε-Kanten (ε-nd e.a., oder kurz NEA)** A ist ein Tupel $A = (K, \Sigma, \Delta, I, F)$. Dabei ist

- K eine endliche Menge von Zuständen,
- Σ ein endliches Alphabet,
- Δ eine endliche Teilmenge von $(K \times \Sigma^*) \times K$,
- $I \subseteq K$ die Menge von Startzuständen, und
- $F \subseteq K$ die Menge der finalen Zustände

Wir erweitern Δ zu $\Delta^* \subseteq (K \times \Sigma^*) \times K)$ wie folgt:

$$(q, \varepsilon)\ \Delta^*\ q' \quad :\Longleftrightarrow_{def} \quad q' = q \text{ oder } ((q, \varepsilon), q') \in \Delta$$

$$(q, w_1 w_2)\ \Delta^* q' \quad :\Longleftrightarrow_{def} \quad \exists q'' \in K \left(\left((q, w_1), q''\right) \in \left(\Delta \cup \Delta^*\right) \right.$$
$$\left. \text{und } \left((q'', w_2), q'\right) \in \left(\Delta \cup \Delta^*\right) \right)$$

Wie verarbeitet ein ε-nd e.a. ein Wort $w \in \Sigma^*$?

- In einem Schritt von Δ oder
- in mehreren Schritten von Δ^*.

Statt Δ^* schreibt man auch kurz Δ.

Definition 2.12 (ε-nd e.a. akzeptierte Sprache)

Die von einem ε-nd e.a. $A = (K, \Sigma, \Delta, I, F)$ **akzeptierte Sprache** ist

$$L(A) := \{w \in \Sigma^* \mid \exists s_0 \in I \; \exists q \in F \; \big((s_0, w) \; \Delta^* \; q\big)\}$$

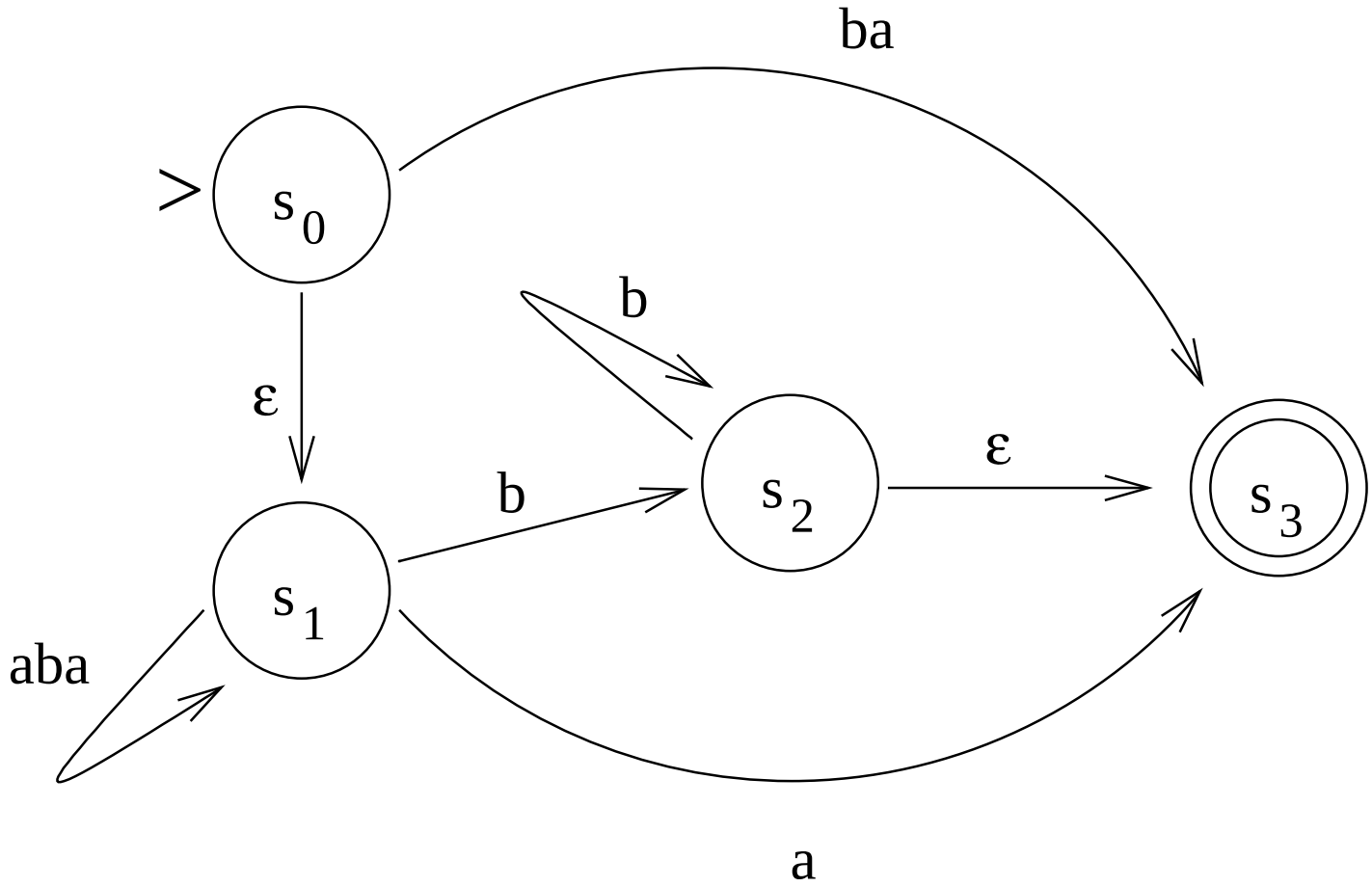


Abbildung 17: Ein Automat mit ε–Kanten für $\{aba\}^*\{b\}\{b\}^* + \{aba\}^*\{a\} + \{ba\}$

Noch ein Beispiel:

$L = \{abc\}^*\{b\}^* + \{abc\}^*\{bab\}\{b\}^*$

Theorem 2.13 (ε-nd e.a. gleich mächtig wie nd e.a.)

Zu jedem ε-nd e.a. A existiert ein nd e.a. A' mit $L(\mathrm{A}) = L(\mathrm{A}')$.

Beweis.

Idee: Ersetze Übergänge aus A:

- nur mit einem Buchstaben markiert: beibehalten
- mit n Buchstaben markiert: ersetzen durch n Übergänge mit einem Buchstaben
- ε-Übergänge: Wenn in A gilt: $(q, a)\ \Delta\ q'$ und $(q', \varepsilon)\ \Delta\ q''$, dann ersetze $(q', \varepsilon)\ \Delta\ q''$ durch $(q, a)\ \Delta\ q''$.

□

A' enthalte also die Zustände von A und

- für jeden Übergang $(q_1, a)\ \Delta_A\ q_2$ mit $a \in \Sigma$ den Übergang $(q_1, a)\ \Delta_{A'}\ q_2$,
- für jeden Übergang $(q_1, w)\ \Delta_A\ q_2$ für Wörter $w = a_1 \ldots a_n$ einer Länge $n \geq 2$ (wobei $a_i \in \Sigma$ gilt für $1 \leq i \leq n$) neue Zustände $p_{(w,1)}, \ldots, p_{(w,n-1)}$ und die Übergänge

$$\begin{array}{rcl} (q_1, a_1) & \Delta_{A'} & p_{(w,1)} \\ (p_{(w,i)}, a_{i+1}) & \Delta_{A'} & p_{(w,i+1)} \text{ für alle } i < n-1 \\ (p_{(w,n-1)}, a_n) & \Delta_{A'} & q_2 \end{array}$$

- für jeden Übergang $(q_1, \varepsilon)\ \Delta_A^*\ q_2$ im alten und jeden Übergang $(q_0, a)\ \Delta_{A'}\ q_1$ im neuen Automaten auch den Übergang $(q_0, a)\ \Delta_{A'}\ q_2$.

Es sei $F_{A'} := F_A$ und
$I_{A'} := I_A \cup \{q \in K_A \mid \exists s \in I_A \left((s, \varepsilon)\ \Delta_A^*\ q\right)\}$.

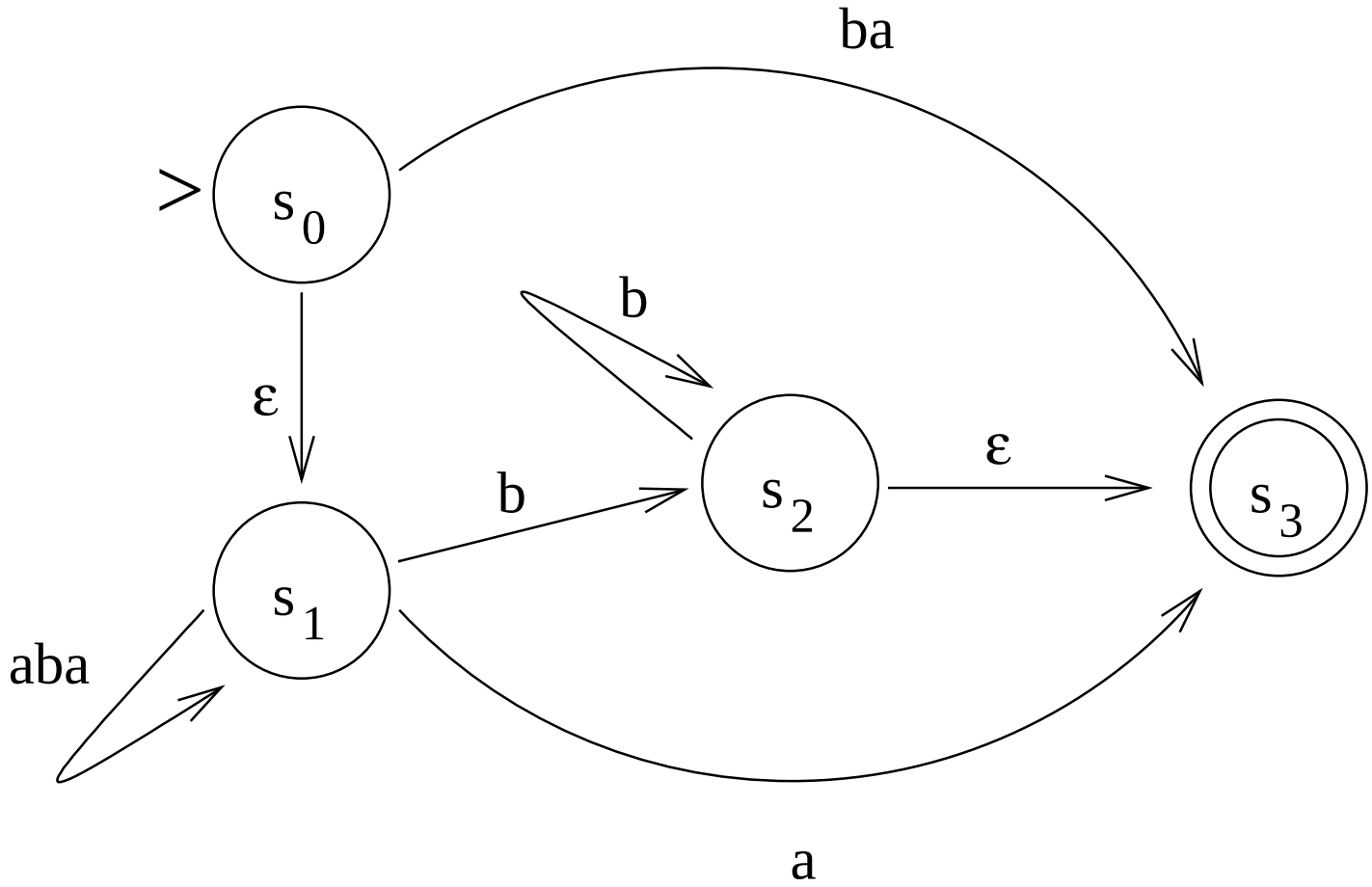


Abbildung 18: Ein Automat mit ε–Kanten für $\{aba\}^*\{b\}\{b\}^* + \{aba\}^*\{a\} + \{ba\}$

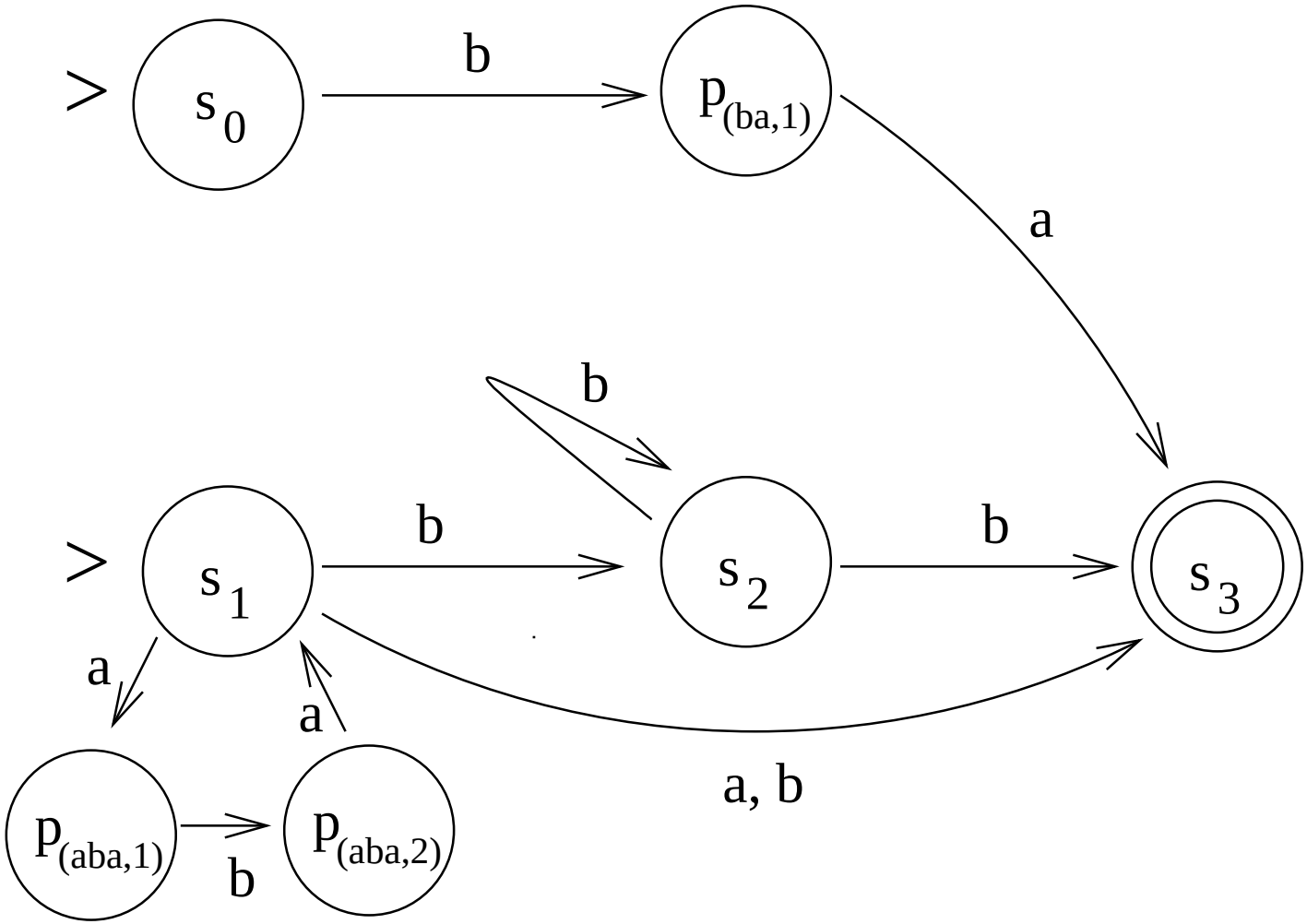


Abbildung 19: Ein indeterminierter Automat, der die gleiche Sprache akzeptiert wie der aus Abb. 18

2.4 rational = Typ 3

Theorem 2.14 (Satz von Kleene: RAT = $\mathbf{L_3}$)

Eine Sprache L ist rational gdw $L \in \mathbf{L}_3$.

Beweis

"$\Rightarrow$" Zu zeigen: **Wenn eine Sprache L von einem endlichen Automaten A akzeptiert wird, kann sie durch eine rechtslineare Grammatik dargestellt werden.**

- Sei also $L = L(A)$,
- A sei ein endlicher Automat mit $A = (K, \Sigma, \delta, s_0, F)$.
- Dazu konstruieren wir eine Grammatik $G = (V, T, R, S)$:

Automat A:	in **Zustand q**,	**liest a**,	geht in **Zustand q'**
Grammatik:	**Endvariable q**,	**erzeugt a**	neue **Endvariable q'**

$V := K$,
$T := \Sigma$,
$S := s_0$, und
$q \to aq' \in R$ falls $\delta(q, a) = q'$,
$q \to \varepsilon \in R$ falls $q \in F$

Mit Induktion über die Länge eines Wortes w kann man nun zeigen, daß gilt: $S \Longrightarrow_G^* wq$ gdw $\delta^*(s_0, w) = q$.

Daraus wiederum folgt

$$\begin{aligned} S \Longrightarrow_G^* w \quad & \textbf{gdw} \quad \exists q \in F \left(S \Longrightarrow_G^* wq \Longrightarrow w \right) \\ & \textbf{gdw} \quad \exists q \in F \left(\delta(s_0, w) = q \right) \\ & \textbf{gdw} \quad w \in L(\mathrm{A}) \end{aligned}$$

"$\Leftarrow$" Zu zeigen: **Wenn eine Sprache L durch eine rechtslineare Grammatik dargestellt werden kann, dann gibt es einen endlichen Automaten, der sie akzeptiert.**

- Sei $G = (V, T, R, S)$ eine rechtslineare Grammatik mit $L = L(G)$.
- Wir konstruieren zu G einen ε-nd e.a. A mit $L = L(A)$.
- Sei $A = (K, \Sigma, \Delta, I, F)$ mit

$$K := V \cup \{q_{stop}\}$$
$$I := \{S\}$$
$$\Sigma := T$$
$$F := \{q_{stop}\}$$

Dabei sei q_{stop} neu, d.h. $q_{stop} \notin V$.

Für Δ definieren wir

$$(X, u)\ \Delta\ X' \quad :\Longleftrightarrow_{def} \quad X \to uX' \in R$$
$$(X, u)\ \Delta\ q_{stop} \quad :\Longleftrightarrow_{def} \quad X \to u \in R$$

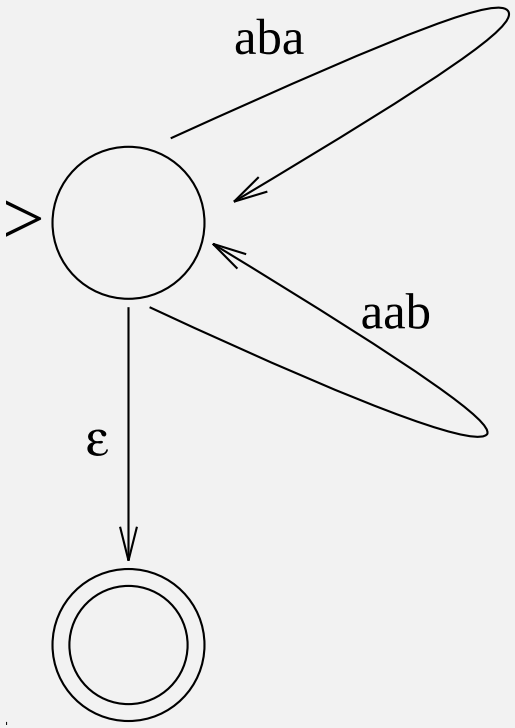
mit $X, X' \in K$ und $u \in \Sigma^*$.

Damit gilt:
$\big(S \Longrightarrow_G^* w\big)$ gdw $\big((S, w)\ \Delta^*\ q_{stop}\big)$ gdw $\big(w \in L(\mathrm{A})\big)$.

Daß dies so ist, kann man durch Induktion über die Länge einer Ableitung in G zeigen. □

Beispiel 2.15

Die Grammatik mit den Regeln $S \to abaS \mid aabS \mid \varepsilon$ wird zu diesem Automaten mit ε-Kanten:



2.5 Pumping Lemma

Wie kann man feststellen, ob eine Sprache regulär ist?

- Rationale Sprachen sind die, die man mit regulären Ausdrücken beschreiben kann (Beweis später).
- **Wie kann dann eine rationale Sprache mit** unendlich vielen Wörtern **aufgebaut sein?** Es gibt nur einen Operator, um mit einem regulären Ausdruck unendlich viele Wörter zu beschreiben: den Kleene-Stern. Also müssen sich in den Wörtern einer unendlichen regulären Sprache bestimmte Buchstabenketten beliebig oft wiederholen.

- Genauso: Wie kann ein endlicher Automat eine unendliche Sprache akzeptieren?
- Er hat nur **endlich** viele Zustände. Also muss er Schleifen enthalten.

Ziel: Zeigen, daß manche Sprachen nicht regulär sind.

- Wenn eine Sprache nicht das einfache Schema von regulären Zeichen-Wiederholungen hat,
- dann kann kein endlicher Automat sie akzeptieren.
- Also ist die Sprache nicht regulär.

Theorem 2.16 (Pumping-Lemma für L_3-Sprachen)

*Sei $L \in$ **RAT**. Dann **existiert ein** $n \in \mathbb{N}$, so daß gilt:*
Für alle $x \in L$ *mit* $|x| \geq n$ **existieren** $u, v, w \in \Sigma^*$ *mit*

- $x = uvw$,
- $1 \leq |v| < n$, *und*
- $uv^m w \in L$ für alle $m \in \mathbb{N}_0$.

- In einer regulären Sprache L
- gibt es zu jedem Wort x in L ab einer bestimmten Länge n neue Wörter uv^iw die auch wieder in der Sprache liegen,
- dabei ist v ein Mittelteil des Wortes x.

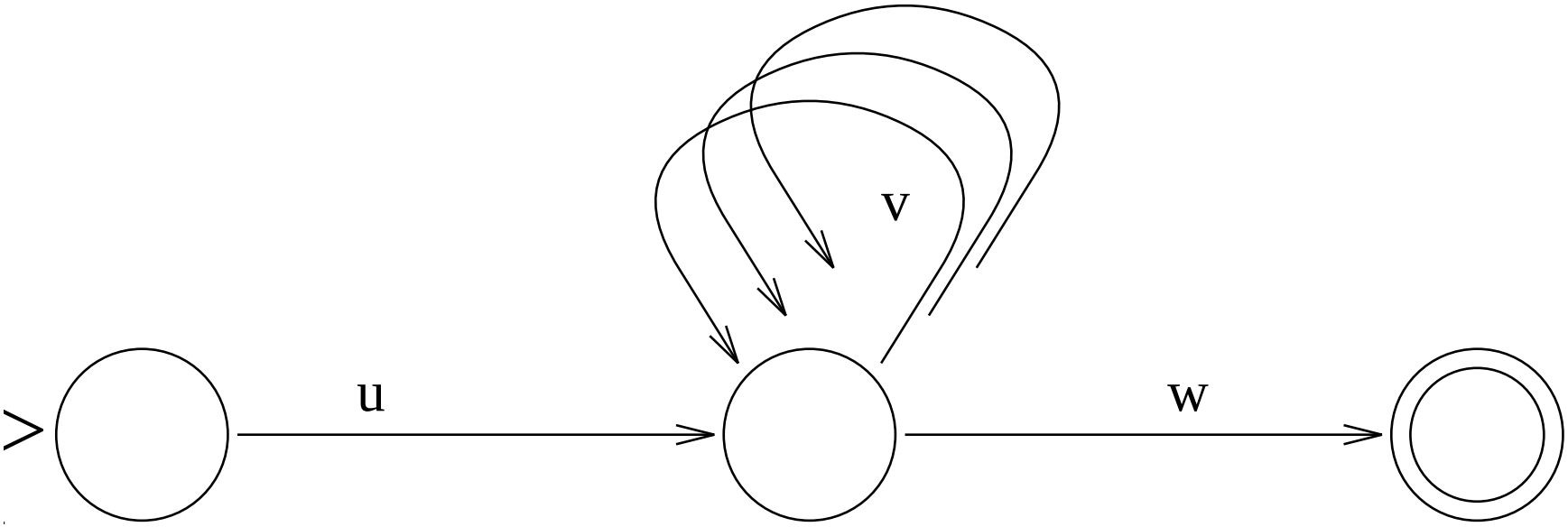


Abbildung 20: Ein Automat, der erst u liest, dann beliebig oft v und dann w

Beweis

Sei L eine reguläre Sprache.

1. Fall: L ist endlich. Sei w_{max} das längste Wort in L. Dann setzen wir n, die Konstante aus dem Satz, auf $n = |w_{max}| + 1$. Dann gibt es keine Wörter $x \in L$, für die $|x| \geq n$ gilt und für die die Bedingungen des Pumping-Lemmas erfüllt sein müßten.

2. Fall: L ist unendlich. Dann gilt:

- in L: beliebig lange Wörter
- erkennender Automat: nur **endlich** viele Zustände.
- Wort länger als Anzahl der Zustände: Dann muss der Automat mindestens einen Zustand q mehrmals durchlaufen.
- Die Schleife, die bei q beginnt und endet, kann der Automat beliebig oft durchlaufen.

Wir wählen folgende Werte:

- Sei $L = L(\mathrm{A})$ und $\mathrm{A} = (K, \Sigma, \delta, s_0, F)$ ein endlicher Automat.
- Für die Konstante n: Wir wählen $n := |K| + 1$.
- Wir betrachten ein beliebiges Wort $x \in L$ mit

$$|x| = t \geq n,$$

$$x = x_1 x_2 \ldots x_t$$

für $x_i \in \Sigma$.

- Seien $q_0, q_1, \ldots, q_t \in K$ die Zustände, die beim Akzeptieren von x durchlaufen werden, mit

$$q_0 = s_0, q_t \in F, \text{ und}$$

$$\delta(q_i, x_{i+1}) = q_{i+1} \quad \forall 0 \leq i \leq t-1$$

Damit gilt: Da $t \geq |K| + 1$ ist, gibt es zwei Werte i und $j \in \{0, \ldots, t\}$, so daß $i \neq j$ und $q_i = q_j$.

Falls $|j - i| \geq |K| + 1$ ist, ist das gleiche Argument wiederum anwendbar, und es gibt zwei weitere, näher beieinander liegende Zustände $q_{i'}, q_{j'}$ im Intervall zwischen q_i und q_j mit $q_{i'} = q_{j'}$.

Also finden wir Werte $i, j \in \{0, \ldots, t\}$ mit $0 < j - i < |K| + 1$ und $q_i = q_j$.

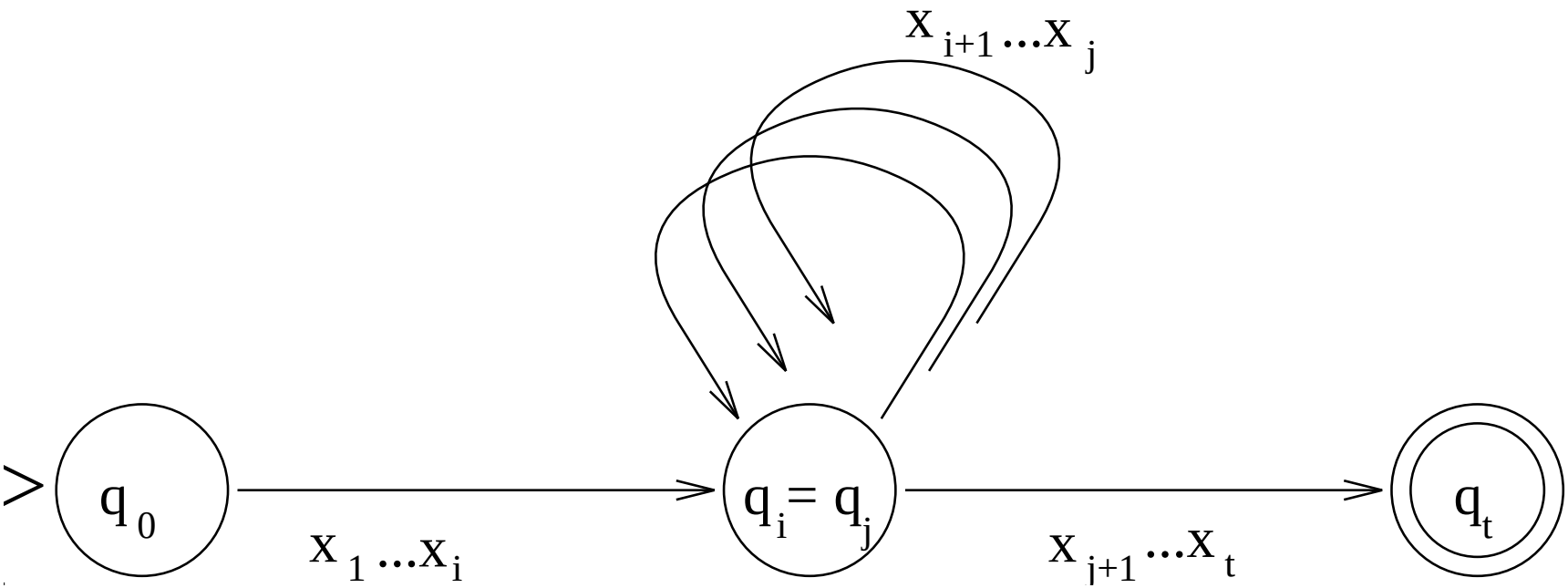


Abbildung 21: Akzeptanz eines **pumpbaren** Wortes x durch einen Automaten

Wir wählen nun

$$\left.\begin{aligned} u &:= x_1 \dots x_i \\ v &:= x_{i+1} \dots x_j \\ w &:= x_{j+1} \dots x_t \end{aligned}\right\} x = uvw \text{ mit } 1 \leq |v| < n.$$

Damit haben wir das gewünschte Ergebnis:

- **Für alle $m \geq 0$ gibt es Wege von q_0 zu q_i mit Beschriftung uv^m, und somit Wege von q_0 nach q_t mit Beschriftung $uv^m w$.**
- **Also gilt $\forall m \in \mathbb{N}_0$, daß $uv^m w \in L$.**
- **Wegen $j - i < |K| + 1 = n$ ist außerdem $|v| < n$.** □

Aus $A \Rightarrow B$ folgt nicht $B \Rightarrow A$!

Das Pumping-Lemma ist **nur** eine Implikation:

Vorsicht: Es gibt Sprachen, die

- nicht rational sind,
- aber für die die Aussage des $\mathbf{L}_3$-Pumping-Lemmas gilt.

Also kann man **nicht** schliessen: Wenn für eine Sprache die Aussage des $\mathbf{L}_3$-Pumping-Lemmas gilt, ist sie rational.

Aber es gilt: Aus $A \Rightarrow B$ folgt $\neg B \Rightarrow \neg A$!

Wenn für eine Sprache die **Aussage des L_3-Pumping-Lemmas nicht gilt**, *dann ist die Sprache* **nicht regulär**.

Wie zeigt man mit dem Pumping-Lemma, daß eine Sprache nicht regulär ist?

*Man zeigt, daß es für jedes $n \in \mathbb{N}$ ein Wort der Sprache gibt, das wenn man es **entsprechend** dem Pumping-Lemma **aufpumpt**, Wörter hervorbringt, die **nicht zu der Sprache gehören**.*

Beispiel 2.17 (Anwendung Pumping-Lemma)

Folgende Sprachen sind nicht rational

1. $L_1 := \{a^i b a^i \mid i \in \mathbb{N}_0\}$
2. $L_2 := \{a^p \mid p \text{ ist Primzahl}\}$

Beweis

(1) $L_1 := \{a^i b a^i \mid i \in \mathbb{N}\}$.

Es gibt beliebig lange Wörter in L_1. Angenommen, L_1 ist rational. Dann existiert eine Zahl n mit den Eigenschaften des Pumping-Lemmas.

Betrachte das Wort $a^n b a^n \in L_1$. Nach dem Pumping-Lemma existieren Teilwörter u, v, w mit $a^n b a^n = uvw$, so daß die restlichen Aussagen des Pumping-Lemmas gelten.

Die Frage ist nun, wie "liegt" das Teilwort v in $a^n b a^n$?

Es gibt 3 Möglichkeiten, wie sich $a^n b a^n$ auf die Teilwörter u, v, w verteilen kann:

1. $u = a^k, v = a^j, w = a^i b a^n$ mit $i, k \geq 0$, $j > 0$ und $k + j + i = n$.
   Wenn wir nun v einmal aufpumpen, erhalten wir $uv^2w = a^k a^{2j} a^i b a^n = a^{k+2j+i} b a^n = a^{n+j} b a^n \notin L_1$, ein Widerspruch.
2. $u = a^n b a^i, v = a^j, w = a^k$ führt analog zu 1. zum Widerspruch.
3. $u = a^k, v = a^j b a^i, w = a^l$ mit $k + j = i + l = n$ und $i, j, k, l \geq 0$
   Pumpen wir nun wieder v einmal auf.
   $uv^2w = a^k a^j b a^i a^j b a^i a^l = a^{k+j} b a^{i+j} b a^{i+l} \notin L_1$, schon wegen der zwei "$b$" ein Widerspruch.

Also ist L_1 nicht rational.

Beweisschema zur Anwendung des PL's

Behauptung: L ist nicht rational.

- Beweis durch Widerspruch: **Annahme: L ist rational.**
- Für rationale Sprachen, also auch L, gilt das PL.
- Es gibt also ein $n \in \mathbb{N}$, sodaß ...
  // ... die Details des Pumping Lemmas gelten.
- Nun suchen wir ein "kritisches" Wort $x \in L$, d.h.
  - $|x| > n$, *d.h. x ist "lang genug" zum Pumpen*
  - *egal (**=Fallunterscheidung**) wie die Zerlegung $x = uvw$ auch zu liegen kommt, wir geben für jeden Fall ein **böses m an, sodaß** $uv^m w \notin L$.*
- Also gilt für L das $\mathbf{L_3}$-Pumping-Lemma nicht: ein Widerspruch[3]

[3] Der kreative Teil des Beweises ist das kritische Wort in Abhängigkeit von n anzugeben und die Fallunterscheidung sauber durchzuführen., der Rest ist immer gleich!

Theorem 2.18 (Erweitertes Pumping-Lemma für L_3)

*Sei $L \in$ **RAT**. Dann existiert ein $n \in \mathbb{N}$, so daß für alle Wörter $x = abc \in L$ mit $|x| \geq n$ und $|b| = n$ gilt*

- $b = uvw$,
- $|v| \geq 1$,
- für alle $i \in \mathbb{N}_0 : auv^iwc \in L$.

Anschaulich bedeutet dies, daß der zu pumpende Teil v des Wortes x beliebig gewählt werden kann (solange er in einem Teilwort einer gewissen Länge vorkommt).

2.6 Wortprobleme

Kann man bestimmen,

- ob eine gegebene **rechtslineare Grammatik** leer ist?
- ob **zwei rechtslineare Grammatiken** äquivalent sind?
- ob ein **Wort** w in $L(G)$ enthalten ist?

Definition 2.19 (erreichbar, co-erreichbar, trim)

Sei $A = (K, \Sigma, \Delta, I, F)$ ein indeterminierter endlicher Automat. Ein Zustand $q \in K$ heißt

- erreichbar $:\Longleftrightarrow_{def} \exists s \in I \, \exists w \in \Sigma^* \, (s, w)\Delta^* q.$
  (Es gibt ein Wort, mit dem q vom Startzustand aus erreicht wird.)
- co-erreichbar $:\Longleftrightarrow_{def} \exists w \in \Sigma^* \, \exists f \in F \, (q, w) \, \Delta^* f.$
  (Es gibt ein Wort, mit dem von q aus ein finaler Zustand erreicht wird.)
- trim $:\Longleftrightarrow_{def} q$ ist erreichbar und co-erreichbar.

Analog heißt der Automat A erreichbar, falls alle $q \in K$ erreichbar sind. A heißt co-erreichbar, falls alle $q \in K$ co-erreichbar sind, und A heißt trim, falls alle $q \in K$ trim sind.

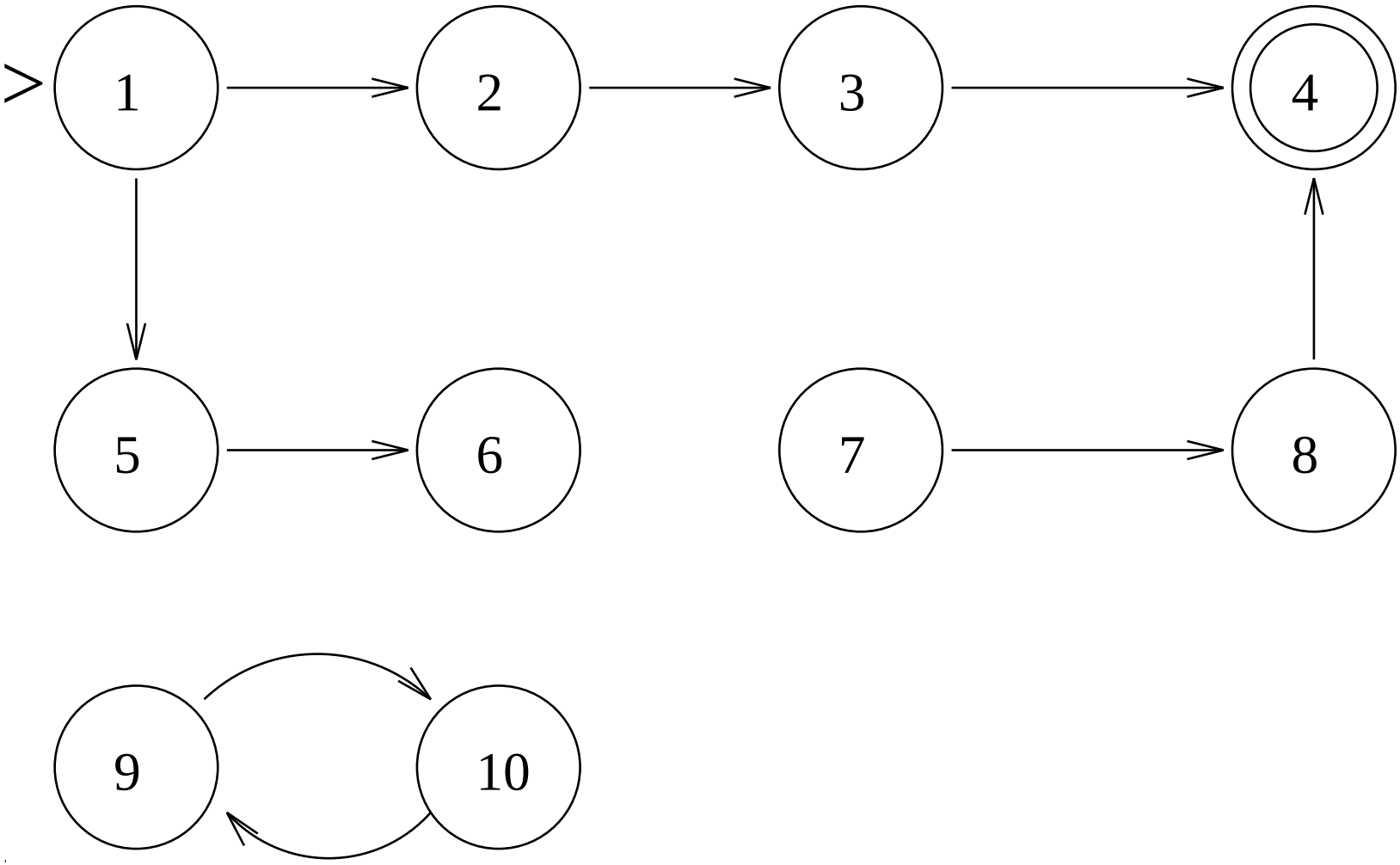


Abbildung 22: In diesem Automaten sind die Zustände 1 – 6 erreichbar, 1 – 4, 7 und 8 co-erreichbar, und die Zustände 1 – 4 sind trim

Kann man bei gegebenem endlichen Automaten feststellen, welche Zustände **erreichbar**, **co-erreichbar** oder **trim** sind?

Definition 2.20 (Teilautomaten)

Seien $A = (K_A, \Sigma_A, \Delta_A, I_A, F_A)$, $A' = (K_{A'}, \Sigma_{A'}, \Delta_{A'}, I_{A'}, F_{A'})$ Automaten. A' heißt Teilautomat von A gdw

$$K_{A'} \subseteq K_A, \quad \Sigma_{A'} \subseteq \Sigma_A, \quad \Delta_{A'} \subseteq \Delta_A, \quad I_{A'} \subseteq I_A, \quad F_{A'} \subseteq F_A$$

A' heißt der von K' erzeugte Teilautomat von A gdw

- $K' \subseteq K_A$,
- $A' = (K', \Sigma_A, \Delta_A \cap ((K' \times \Sigma_A) \times K'), I_A \cap K', F_A \cap K')$.

Diese Teilautomaten kann man leicht konstruieren.

Definition 2.21 ($A^{\mathbf{err}}$, $A^{\mathbf{co-e}}$, $A^{\mathbf{trim}}$)

$A^{\mathbf{err}}$ ist der von den erreichbaren Zuständen von A erzeugte Teilautomat von A.

$A^{\mathbf{co-e}}$ ist der von den co-erreichbaren Zuständen von A erzeugte Teilautomat von A.

$A^{\mathbf{trim}}$ ist der von den trimmen Zuständen von A erzeugte Teilautomat von A.

Lemma 2.22

1. $A^{trim} = (A^{err})^{co-e} = (A^{co-e})^{err}$.
2. $L(A) = L(A^{err}) = L(A^{co-e}) = L(A^{trim})$.

Lemma 2.23

*Sei A ein endlicher Automat. Es ist **entscheidbar**, ob die Menge*

1. *$L(A)$ leer ist.*
2. *$L(A)$ unendlich ist.*

Beweis

Zu (i): Um festzustellen, ob $L(A) = \emptyset$ ist, berechnet man zu dem e.a. A den erreichbaren nd. e.a. A^{err} (man eliminiert also alle die Zustände von A, die nicht vom Startzustand aus erreichbar sind). $L(A) = L(A^{err})$ ist genau dann nicht leer, wenn A^{err} noch finale Zustände hat.

Zu (ii): Um festzustellen, ob $L(A)$ unendlich ist, eliminiert man aus A^{err} alle nicht co-erreichbaren Zustände und erhält den nd. e.a. A^{trim}. $L(A) = L(A^{trim})$ ist offensichtlich genau dann unendlich, wenn in der graphischen Darstellung von A^{trim} ein Kreis enthalten ist. □

Dies funktioniert auch für nd e.a.'en.

Lemma 2.24

*Seien A_1, A_2, A_3 endliche Automaten. Es ist **entscheidbar**, ob folgendes gilt:*

(i) $L(A_1) \cap L(A_2) = \emptyset$.

(ii) $L(A_1) \cup L(A_2) = L(A_3)$.

(iii) $L(A_1) = L(A_2)$.

Beweis

Seien A_1, A_2 endliche Automaten.

Zu (i): Es gibt einen endlichen Automaten $A_\cap$ mit $L(A_\cap) = L(A_1) \cap L(A_2)$ (**warum?**).
Nach dem obigen Lemma ist die Frage, ob $L(A_\cap) = \emptyset$ ist, entscheidbar. Also ist auch die Frage, ob $L(A_1) \cap L(A_2) = \emptyset$ ist, entscheidbar.

Zu (ii): Man kann zu A_1 und A_2 einen ea A_3' effektiv konstruieren, mit: $L(A_3') = L(A_1) \cup L(A_2)$ (wie in (i)). Von diesem Automaten konstruiert man effektiv den Komplement-Automaten A_3''. Nun wendet man (1) auf A_3'' und A_3: dies gilt genau dann wenn $L(A_1) \cup L(A_2) = L(A_3)$.

Zu (iii): Man kann zu A_1 und A_2 einen endlichen Automaten $A_=$ konstruieren mit $L(A_=) = (L(A_1) \cap \overline{L(A_2)}) \cup (\overline{L(A_1)} \cap L(A_2))$. (**Warum?**).
Wenn man nun nachrechnet, stellt man fest, daß $\big(L(A_1) = L(A_2)$ gdw $L(A_=) = \emptyset\big)$ ist. Die Frage, ob $L(A_=) = \emptyset$ ist, ist nach dem obigen Lemma entscheidbar, also gilt Gleiches für die Frage, ob $L(A_1) = L(A_2)$ ist. □

Korollar

An diesen Beweisen kann man sehen, daß $A_\cap$, $A_\cup$ und $A_=$ effektiv aus A_1 und A_2 konstruierbar sind.

Angenommen, für zwei e.a.'en gilt $L(A_1) = L(A_2)$. Muss dann auch ${A_1}^{\mathbf{trim}} = {A_2}^{\mathbf{trim}}$ gelten?

2.7 Rational = Regulär

Bisher haben wir oft von **regulären Sprachen** gesprochen, wenn wir **Sprachen vom Typ 3**, also Sprachen die rechtslinearen Grammatiken entsprechen, meinten. Richtiger hätten wir von rationalen Sprachen sprechen müssen.

Dieser Sprachgebrauch hat sich eingebürgert. Allerdings kann man unter **regulären Sprachen** auch solche Sprachen verstehen, die durch **reguläre Ausdrücke** definiert werden können.

Wir zeigen nun, daß Sprachen die durch reguläre Ausdrücke definiert werden können, genau die **rationalen Sprachen** sind, also die, die durch einen **endlichen Automaten** charakterisiert werden können.

Theorem 2.25 (Hauptsatz von Kleene)

*Die durch endliche Automaten akzeptierten Sprachen (also die **rationalen**) sind genau die, die man durch reguläre Ausdrücke beschreiben kann.*

Beweis

"$\Rightarrow$" Dies zeigen wir durch Induktion über den Weg während des Akzeptierens eines Wortes im Automaten A. OBdA numerieren wir die Zustände durch und setzen: $s_0 = q_1$.

$$R^k_{i,j} := \{w \in \Sigma^* : \delta^*(q_i, w) = q_j \text{ und für alle Präfixe } u \text{ von } w \text{ mit } \varepsilon \neq u \neq w \text{ gilt } \delta^*(q_i, w) \in \{q_1, q_2, \ldots, q_k\}\}$$

Beweis: "$\Rightarrow$"

$R^k_{i,j}$ beschreibt alle Wörter, die zwischen q_i und q_j in A durchlaufen werden können, mit der Einschränkung, daß nur $\{q_1, q_2, \ldots, q_k\}$ als Zwischenzustände aufgesucht werden dürfen.

Offensichtlich ist

$$L(\mathrm{A}) = \bigcup_{q_f \in F} R^n_{1,f}$$

Es genügt daher zu zeigen, daß alle Mengen $R^n_{1,f}$ durch reguläre Ausdrücke beschrieben werden können.
Dazu zeigen wir, durch Induktion über k:

$$R^k_{i,j} \in \mathfrak{Reg}_\Sigma \text{ für alle } i, j \leq n$$

Beweis: "$\Rightarrow$"

Induktionsanfang: $k = 0$

$$R^0_{i,j} = \begin{cases} \{a \in \Sigma \mid \delta(q_i, a) = q_j\} & \text{falls } i \neq j \\ \{\varepsilon\} \cup \{a \in \Sigma \mid \delta(q_i, a) = q_j\} & \text{falls } i = j \end{cases}$$

$R^0_{i,j}$ kann man offensichtlich als reguläre Ausdrücke darstellen.

Induktionsschritt: $k \to k+1$: Induktionsvoraussetzung: $R^k_{i,j} \in \mathfrak{Reg}_\Sigma$ für alle $i, j \leq n$. Solange wir also nur $q_1, \ldots q_k$ als Zwischenzustände zulassen, können wir einen regulären Ausdruck angeben.
Nun erlauben wir zusätzlich q_{k+1} als Zwischenzustand:

Beweis: "$\Rightarrow$"

Welche Wege gibt es von q_i nach q_j, falls $\{q_1, \ldots, q_k, q_{k+1}\}$ als Zwischenzustände erlaubt sind?

- q_{k+1} wird gar nicht angelaufen, d.h. der Weg ist schon in $R^k_{i,j}$ beschrieben.
- Wir laufen von q_i zu q_{k+1} (erster Besuch!), laufen dann beliebig oft eine Schleife, d.h. von q_{k+1} nach q_{k+1} (letzter Besuch!), und von dort zu q_j. Formal: $R^k_{i,k+1}(R^k_{k+1,k+1})^* R^k_{k+1,j}$

macht insgesamt:

$$R^{k+1}_{i,j} = R^k_{i,j} \cup R^k_{i,k+1}(R^k_{k+1,k+1})^* R^k_{k+1,j}$$

Beweis: "$\Rightarrow$"

$$R_{i,j}^{k+1} = R_{i,j}^{k} \cup R_{i,k+1}^{k}(R_{k+1,k+1}^{k})^* R_{k+1,j}^{k}$$

Für $R_{i,j}^{k+1}$ verwenden wir nur Vereinigung, Konkatenation und den Kleene-Stern.

Wir erinnern uns an Definition 1.3 (Reguläre Ausdrücke) und schließen, da - gemäß der Induktionsvoraussetzung - alle Mengen auf der rechten Seite als reguläre Ausdrücke beschreibbar sind, gilt das auch für den Gesamtausdruck $R_{i,j}^{k+1}$. □

Beweis: "$\Leftarrow$"

Wir zeigen nun, daß jeder reguläre Ausdruck auch durch einen endl. Automaten akzeptiert wird. Dies geht, wie üblich, durch Induktion über den Aufbau regulärer Ausdrücke. Es genügt zu zeigen, daß dies für ε-NEAs gilt[a].

Induktionsanfang: Offensichtlich gibt es NEAs, die den regulären Ausdrücken "0" und "a" entsprechen: Startzustand und (davon verschiedener) Finalzustand. Im Fall "0" gibt es keine Übergänge, im Fall "a" gibt es einen Übergang für a.

[a] denn aus diesen lassen sich ja äquivalente DEAs konstruieren

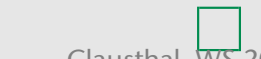
Beweis: "$\Leftarrow$"

Induktionsschritt: Wir müssen zeigen, daß es ε-NEAs zu $R + S$, RS und zu R^* gibt, unter der Voraussetzung, daß es ε-NEAs A_R und A_S gibt.

Für A_{R+S}: siehe Beweis von Theorem 2.24.

Für A_{RS} schalten wir die Automaten A_R und A_S hintereinander (und ergänzen ε-Übergänge von den Finalzuständen von A_R in die Startzustände von A_S).

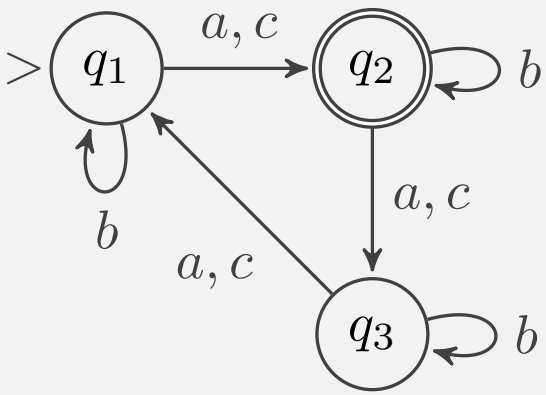
A_{R^*} konstruiert man, indem man zunächst einen neuen akzeptierenden Startzustand einführt und von diesem zu jedem ehemaligen Startzustand einen ε-Übergang. Zusätzlich wird von jedem alten Finalzustand ein ε-Übergang zu allen ehemaligen Startzuständen hinzugefügt.



Aus dem Beweis ergeben sich sofort folgende Konstruktionsverfahren:

- regulärer Ausdruck $\rightarrow$ e.a;
- e.a. $\rightarrow$ regulärer Ausdruck.

Beispiel 2.26 (e.a. $\rightarrow$ reg. Ausdruck)



$$R^0_{1,1} = R^0_{2,2} = R^0_{3,3} = \{b\} \cup \{\varepsilon\},\ r^0_{1,1} = r^0_{2,2} = r^0_{3,3} = b + 1$$
$$R^0_{1,2} = R^0_{2,3} = R^0_{3,1} = \{a, c\},\ r^0_{1,2} = r^0_{2,3} = r^0_{3,1} = a + c$$
$$R^1_{1,1} = R^0_{1,1} \cup R^0_{1,1}(R^0_{1,1})^* R^0_{1,1},\ r^1_{1,1} = \cdots = b^*$$
$$R^1_{2,2} = R^0_{2,2} \cup R^0_{2,1}(R^0_{1,1})^* R^0_{1,2},\ r^1_{2,2} = r^0_{2,2} = a + c$$
$$R^1_{3,3} = R^0_{3,3} \cup R^0_{3,1}(R^0_{1,1})^* R^0_{1,3},\ r^1_{3,3} = r^0_{3,3} = a + c$$
$$R^1_{1,2} = R^0_{1,2} \cup R^0_{1,1}(R^0_{1,1})^* R^0_{1,2},$$
$$r^1_{1,2} = a + c + (b + 1)b^*(a + c) = b^*(a + c)$$
$$R^2_{1,2} = \ldots,\ r^2_{1,2} = \cdots = b^*(a + c)b^*$$
$$R^3_{1,2} = \ldots,\ r^3_{1,2} = \cdots = ((b^*(a + c))^3)^* b^*(a + c)b^*$$

Entscheidungsprobleme für reguläre Sprachen

Sei $L, L_1, L_2 \in \mathbf{L_3}$ gegeben, d.h. wir haben Automaten A, A_1, A_2 mit $L = L(A)$, $L_1 = L(A_1)$ und $L_2 = L(A_2)$

Problem	Input	Frage	Entscheidbar?
Wortproblem	$w \in \Sigma^*$	$w \in L$?	**Ja**
Leerheitsproblem		$L = \emptyset$?	**Ja**
Endlichkeitsproblem		$\|L\|$ endlich?	**Ja**
Äquivalenzproblem		$L_1 = L_2$?	**Ja**

Abschlusseigenschaften für reguläre Sprachen

Sei $L, L_1, L_2 \in \mathbf{L_3}$ gegeben, d.h. wir haben Automaten A, A_1, A_2 mit $L = L(A)$, $L_1 = L(A_1)$ und $L_2 = L(A_2)$

Operation	Frage	Konstruktion	Antwort
Vereinigung	$L_1 \cup L_2 \in \mathbf{L_3}$?	Produkt-Automat	**Ja**
Schnitt	$L_1 \cap L_2 \in \mathbf{L_3}$?	Produkt-Automat	**Ja**
Konkatenation	$L_1 \circ L_2 \in \mathbf{L_3}$?	ε-Kanten zw. A_1, A_2	**Ja**
Stern	$L^* \in \mathbf{L_3}$?	ε-Kanten von F zu I	**Ja**
Negation	$\overline{L} \in \mathbf{L_3}$?	e.a. negieren	**Ja**