

Roter Faden zum Kurs Theoretische Informatik, WS 05/06

Literatur zu den Teilen 1 bis 3:

Ulrich Hedtstück: Einführung in die Theoretische Informatik, Oldenbourg, seit 2000 wiederholt neu aufgelegt

Asteroth/Baier: Theoretische Informatik, Pearson Studium, 2002, inzwischen zweite Auflage

Güting/Erwig: Übersetzerbau, Springer, 1999

Uwe Schöning: Theoretische Informatik - kurzgefaßt, seit 1999 wiederholt neu aufgelegt, Spektrum Akademischer Verlag

Thomas W. Parsons: Introduction to Compiler Construction, Freeman, Second Printing 2003

Der "Hedtstück" ist ein gut geschriebenes und lesbares Lehrbuch, das allerdings wenig über Parser enthält.

Die Skriptteile über Top-Down-Analyse orientieren sich am "Güting/Erwig", die Teile über Bottom-Up-Analyse sind ähnlich dargestellt wie im "Asteroth/Baier".

Der "Schöning" ist kurz, inhaltsreich und sehr komprimiert geschrieben.

Der "Parsons" ist ein außerordentlich gut geschriebenes, weiterführendes Buch über die Grundlagen der Automatentheorie und die klassischen Inhalte des Compilerbaus.

Teil 1 Reguläre Sprachen

Beispiele zur Illustration der Begriffe *Problem*, *Algorithmus*, *Komplexität*:

PARTITION (algorithmisch schwer zugängliches Problem)

gegeben: Eine endliche Menge von Päckchen mit bekannten Gewichten

Frage: Läßt sich die Päckchenmenge in zwei Teilmengen mit jeweils gleichem Gesamtgewicht zerlegen?

Datensatz_1: 1 , 2 , 3 , 4 , 9 , 13

Datensatz_2: 1 , 5 , 6 , 7 , 11

ZEHNTES HILBERTSCHES PROBLEM (1900, algorithmisch unzugänglich)

Gegeben: Eine Polynomgleichung mit ganzzahligen Koeffizienten.

Frage: Hat die Gleichung eine ganzzahlige Lösung?

Datensatz_1: $x^2 + y^2 = z^2$

Datensatz_2: $x^2 + y^2 = 3$

Beispiel für einen endlichen Automaten in Tabellenform:

	0	1	2	3	4	5	6	7	8	9	
Startzustand	*	↓	↓↓	*	↓	↓↓	*	↓	↓↓	*	Ja
	*	↓	↑↑	*	↓	↑↑	*	↓	↑↑	*	Nein_1
	*	↑↑	↑	*	↑↑	↑	*	↑↑	↑	*	Nein_2

Datensatz_1: 1 2 4 2

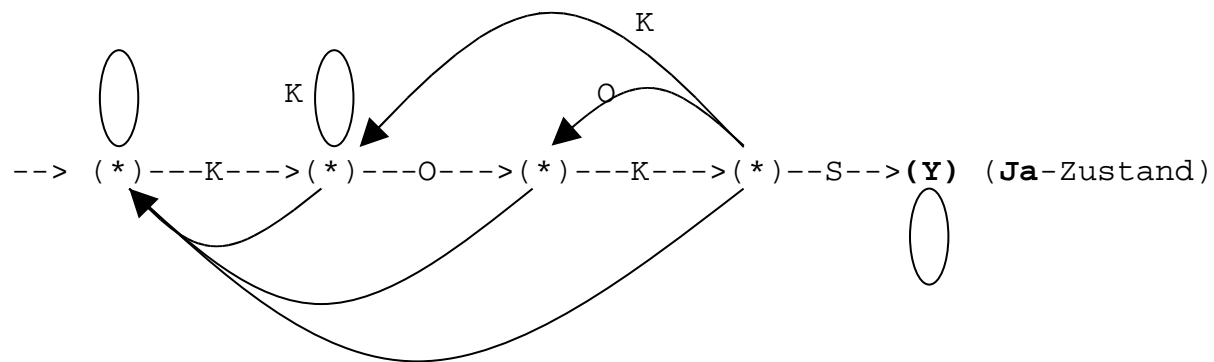
Datensatz_2: 3 4 4 6

Datensatz_3: 1 5

Datensatz_4: 2 5

Man überlege sich, welche Frage dieser Automat entscheidet.

Endlicher Automaten als Graph (Suche in Strings nach KOKS)



[einige Texte ausprobieren]

[Folie: Hedtstück S.6 (Scanner, Parser)]

Definition (Kleenesche Hülle, Formale Sprache)

Für eine endliche Menge von Symbolen (ein **Alphabet**) Σ bezeichnet Σ^* (**Kleenesche Hülle**) die Menge aller endlichen Strings von Sym-

bolen aus Σ (einschließlich des **leeren Strings** ϵ).

Eine (**formale**) **Sprache** über Σ ist eine Teilmenge L von Σ^* .

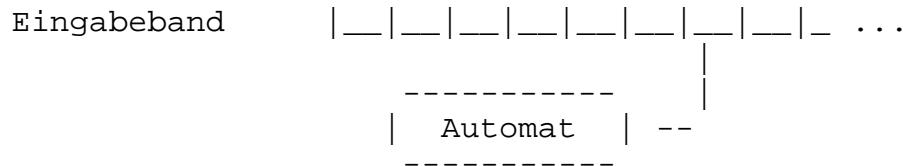
Definition (Deterministischer endlicher Automat, DFA)

Ein deterministischer Automat $A = (Q, \Sigma, q_0, \rho, F)$

- befindet sich stets in einem Zustand einer endlichen Zustandsmenge.
Diese Zustandsmenge Q enthält einen speziellen Startzustand q_0 und eine Teilmenge F (**Final**) aus Q von akzeptierenden Endzuständen (**Ja-Zuständen**).
- liest aus einem Eingabeband Zeichen eines endlichen Alphabets Σ (oft ist $\Sigma = \{0,1\}$).
- wird in Gang gehalten von einer Zustandsüberföhrungsfunktion (von einem **Programm**, konkret von einer Matrix)

$$\rho: Q \times \Sigma \rightarrow Q$$

die also einen Zustand abhängig von aktuell gelesenen Zeichen in einen neuen Zustand überföhrt.



Der Automat funktioniert in natürlicher Weise: Ein String $x \in \Sigma^*$ wird auf das Eingabeband geschrieben, der Lesekopf wird auf den Anfang des Strings gesetzt und der Startzustand q_0 wird angenommen.

Der Lesekopf liest das aktuelle Zeichen, nimmt gemäß Programm ρ einen neuen Zustand an und bewegt den Lesekopf um eine Position nach rechts. Erreicht der Lesekopf das erste Leerzeichen nach dem Ende des Strings, so stoppt der Automat.

Befindet sich der Automat bei Stop in einem Zustand $q \in F$ (in einem Ja-Zustand), so hat er den Eingabestring **akzeptiert**.

Die vom Automaten A **akzeptierte Sprache** $L(A)$ ist die Menge aller akzeptierten Strings.

Definition (Reguläre Ausdrücke)

- (1) Der Ausdruck (0) produziert den String 0
Der Ausdruck (1) produziert den String 1
- (2) Sind (a) und (b) Ausdrücke, so produziert der Ausdruck (a+b) die Strings, die von (a) oder die von (b) produziert werden.
- (3) Sind (a) und (b) Ausdrücke, so produziert der Ausdruck (ab) alle Strings der Form uv, für die u von (a) und

- v von (b) produziert wird.
- (4) Ist (a) ein Ausdruck, so produziert $(a)^*$ alle Strings, die endliche Ketten von Strings aus (a) sind (inklusive der leeren Kette).

Die Menge der von einem Ausdruck produzierten Strings nennen wir auch die vom Ausdruck **produzierte Sprache**.

Definition Eine **Grammatik** enthält

- Terminalsymbole (für uns oft 0 und 1)
- Hilfssymbole (wir benutzen Großbuchstaben)
- ein Startsymbol S (eines der Hilfssymbole)
- Ersetzungsregeln der Form $str1 \rightarrow str2$ ($str1$ und $str2$ sind Strings)

Eine Ersetzungsregel ist so zu verstehen, daß man überall, wo man im aktuellen String den Teilstring $str1$ vorfindet, stattdessen auch $str2$ schreiben darf.

Die ausgehend vom Startsymbol S nur mit den Regeln produzierten und nur aus Terminalsymbolen bestehenden Strings bilden die von der Grammatik **abgeleitete Sprache**.

Definition (Reguläre Grammatiken)

Eine Grammatik heißt **linkslinear**, wenn ihre Ersetzungsregeln alle von der Form

$$U \rightarrow \epsilon \quad \text{oder} \quad U \rightarrow Vstr$$

sind, wobei U und V Hilfssymbole sind und str ein String aus Terminalsymbolen ist. Entsprechend heißt eine Grammatik

rechtslinear, wenn ihre Ersetzungsregeln die Form

$$U \rightarrow \epsilon \quad \text{oder} \quad U \rightarrow strV$$

haben. Eine Grammatik ist **regulär**, wenn sie entweder linkslinear oder rechtslinear ist.

Man kann zeigen: Sprachen, die von linkslinearen Grammatiken erzeugt werden können, lassen sich auch von rechtslinearen Grammatiken erzeugen und umgekehrt.

Beispiel Die Sprache aller binären Strings, die eine gerade Anzahl von Einsen haben, wird akzeptiert von dem unten angegebenen Automaten.

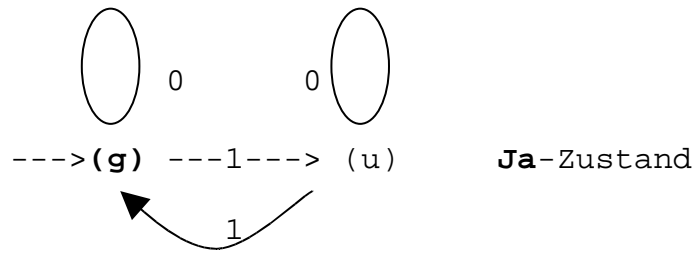
Den beiden Zuständen des Automaten kann man die folgende Bedeutung geben:

g: bisher geradzahlig viele Einsen gelesen

u: bisher ungeradzahlig viele Einsen gelesen

1 - 5

p	0	1
g	g	u
u	u	g



Ein zu dieser Sprache passender regulärer Ausdruck ist

$0^*[10^*10^*]^*$

Eine passende Grammatik ist

$S \rightarrow \epsilon$, $S \rightarrow 0S$, $S \rightarrow 1T$, $T \rightarrow 0T$, $T \rightarrow 1U$, $U \rightarrow S$

Es geht auch kürzer (man vergleiche den obigen Graphen):

$S \rightarrow \epsilon$, $S \rightarrow 0S$, $S \rightarrow 1U$, $U \rightarrow 0U$, $U \rightarrow 1S$

Letzteres in Kompaktnotation:

$S \rightarrow \epsilon | 0S | 1U$, $U \rightarrow 0U | 1S$

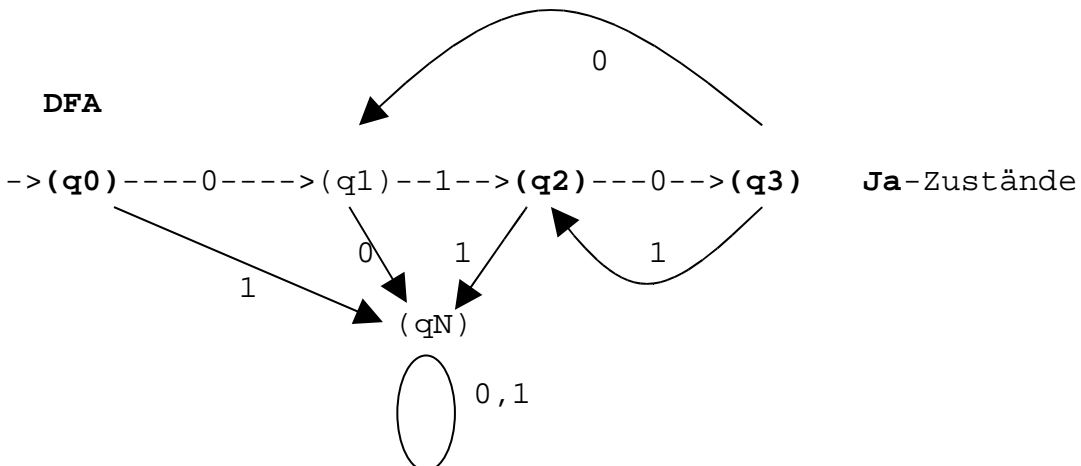
[einige Ableitungen ausprobieren]

Beispiel

Sei L die formale Sprache aller endlichen Ketten aus 01 und 010 (einschließlich der leeren Kette).

Zu L gehören etwa 0101001001, 0101010

Zu L gehört *nicht* 0100110 (warum nicht?)



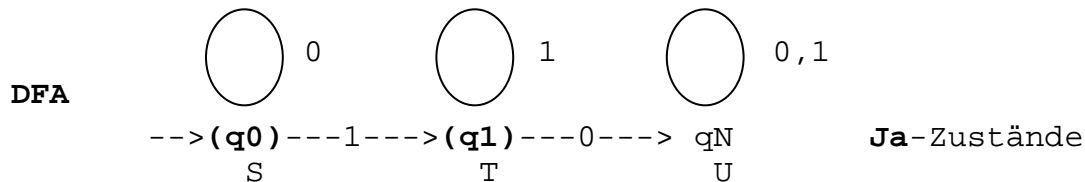
Die naheliegende Frage ist, wie man systematisch zu diesem Automaten kommt (siehe unten).

Die aus 01 und 010 bestehenden Ketten durch einen regulären Ausdruck zu beschreiben ist offenbar trivial: $(01+010)^*$

Die folgenden drei Transformationen illustrieren den weiter unten notierten zentralen Satz der Automatentheorie (daß nämlich alle bisherigen drei Automatentypen äquivalent sind).

DFA => Reguläre Grammatik

Beispiel L : binäre Strings der Form $0^n 1^m$; $m, n \geq 0$
also die Strings, in denen die Nullen vor den Einsen stehen.



Dieser Automat transformiert sich zu der Grammatik:

$$S \rightarrow \epsilon \mid 0S \mid 1T \quad , \quad T \rightarrow \epsilon \mid 1T \mid 0U \quad , \quad U \rightarrow 0U \mid 1U$$

oder kürzer:

$$S \rightarrow \epsilon \mid 0S \mid 1T \quad , \quad T \rightarrow 1T \mid \epsilon$$

oder noch kürzer:

$$S \rightarrow 0S \mid T \quad , \quad T \rightarrow 1T \mid \epsilon$$

Reguläre Grammatik => Regulärer Ausdruck

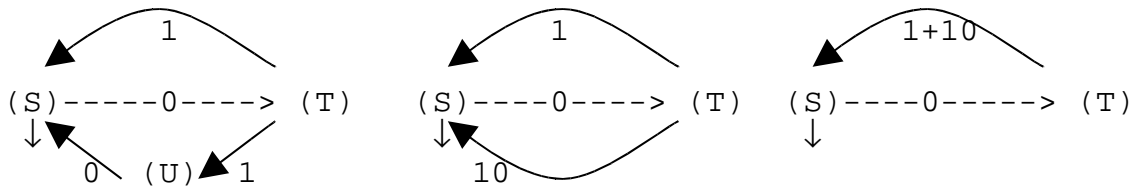
Beispiel (siehe oben)

L: beliebige Ketten aus 01 und 010

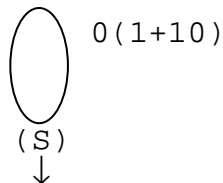
Eine mögliche reguläre Grammatik ist:

$$S \rightarrow \epsilon \mid 0T \quad , \quad T \rightarrow 1S \mid 1U \quad , \quad U \rightarrow 0S$$

Den zugehörigen regulären Ausdruck bekommt man durch einige einleuchtende *elementare Transformationen*:



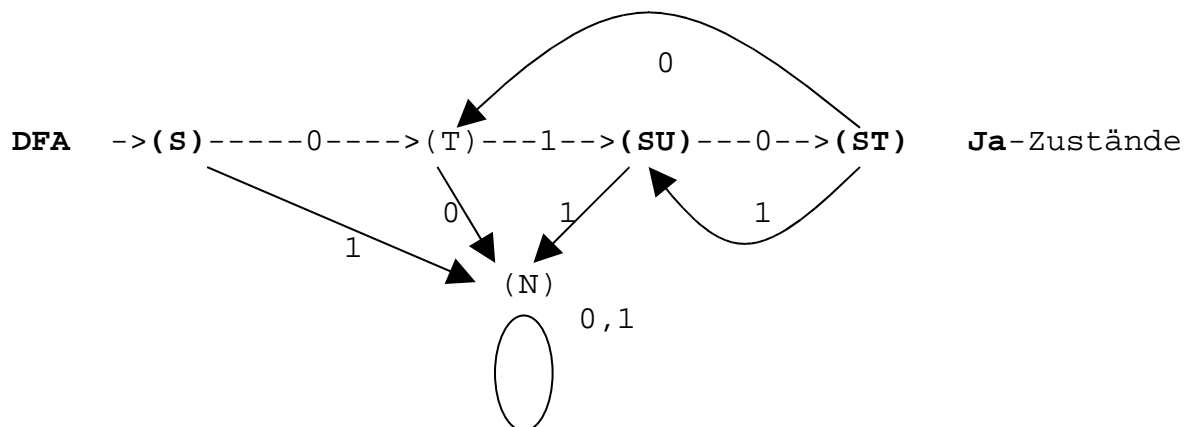
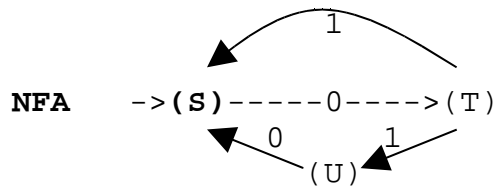
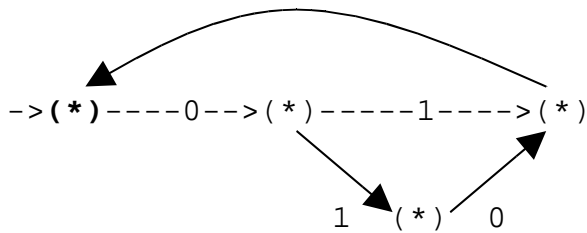
und schließlich



Also $[0(1+10)]^* \approx (01+010)^*$

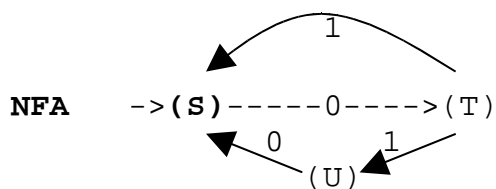
Regulärer Ausdruck (\Rightarrow NFA) \Rightarrow DFA

Beispiel (L von oben): $[0(1+10)]^*$



Letzteres ist (abgesehen von den Bezeichnungen) genau der DFA, der oben schon notiert war!

Der hier unversehens aufgetauchte "nicht-deterministische" Automat NFA ist eigentlich gar kein richtiger Automat, denn er verstößt gegen die Konstruktionsvorschriften, die für "richtige" deterministische Automaten DFA festgelegt wurden. Dennoch ist der NFA äußerst praktisch zum Prüfen von Strings:



0 1 0 0 1:

(S)--0-->(T)--1-->(SU)--0-->(ST)--0-->(T)--1-->(SU)

String akzeptiert, da S erreicht wurde

0 1 0 0:

(S)--0-->(T)--1-->(SU)--0-->(ST)--0-->(T)

String abgelehnt, da S nicht erreicht wurde

0 1 0 1:

(S)--0-->(T)--1-->(SU)--0-->(ST)--1-->(SU) akzeptiert

0 1 1 0:

(S)--0-->(T)--1-->(SU)--1-->()

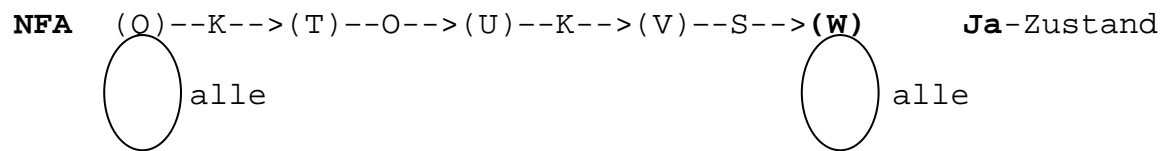
String abgelehnt (der Automat bleibt stecken).

Diese "Debug-Beispiele" zeigen im übrigen, wie man den "richtigen" deterministischen Automaten DFA durch Teilmengenbildung aus dem NFA konstruieren kann.

Nichtdeterministische Automaten werden später auch theoretisch wichtig. An unserem Beispiel notieren wir einige Beobachtungen:

- Ein String (z.B. 010) gehört zur Sprache L
- Beim Abarbeiten des Strings auf NFA *kann* der Ja-Zustand S erreicht werden (bei richtiger Wahl der Übergänge)
- Beim Abarbeiten des Strings auf DFA wird ein Zustand erreicht, der S enthält

Beispiel (nicht-deterministischer Automat für Suche in Strings nach KOKS)



Suchstring: O K O K O K S S

(Q) --O--> (Q) --K--> (QT) --O--> (QU) --K--> (QTV) --O--> (QU)

--K--> (QTV) --S--> (QW) --S--> (QW) String akzeptiert

Definition (Nichtdeterministischer endlicher Automat, NFA)

Zu einem nichtdeterministischen Automaten $A = (Q, \Sigma, q_0, \rho, F)$ gehört

- eine endliche Menge Q von Zuständen mit Startzustand q_0 und Menge F von Ja-Zuständen.
- das Eingabealphabet Σ
- die Zustandsüberführungs-Relation

$\rho: Q \times \Sigma \rightarrow P(Q)$, $P(Q)$: Potenzmenge von Q
 die dem Automaten für jeden Zustand in Abhängigkeit vom gegebenen Zeichen eine Menge von möglichen Nachfolgerzuständen zur Auswahl anbietet.

Aus den Zuständen, die als Nachfolgerzustände angeboten werden, sucht sich der Automat zufällig (also nicht-deterministisch) einen aus.

Die von einem nichtdeterministischen Automaten **akzeptierte Sprache** ist die Menge der Strings, die vom Automaten akzeptiert werden *können*, die also akzeptiert werden, falls man bei den Zustandsübergängen stets eine geeignete Wahl trifft.

Satz Die folgenden Aussagen über eine formale Sprache L sind äquivalent

- L wird von einem endlichen deterministischen Automaten DFA akzeptiert
- L wird von einem endlichen nicht-deterministischen Automaten NFA akzeptiert
- L wird von einer regulären Grammatik abgeleitet
- L wird von einem regulären Ausdruck produziert

Ein Scanner für Strings der Form $(a^*b+ac)d$

Wortmuster (**Patterns**) werden üblicherweise durch reguläre Ausdrücke dargestellt:

$(a^*b + ac)d$

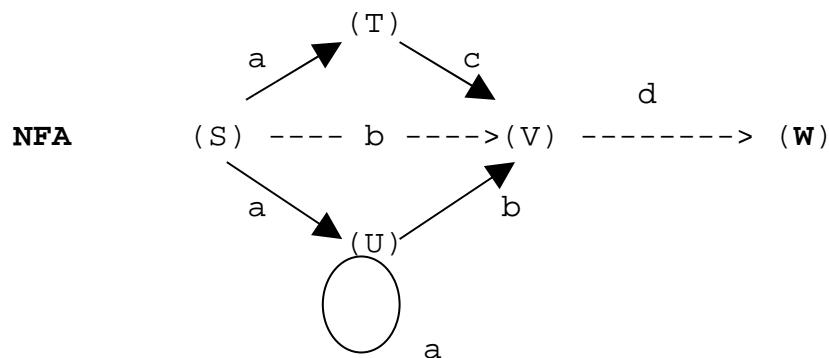
Von beliebig eingegebenen Strings wüßten wir nun gern, ob sie zu diesem Muster passen oder nicht:

aaabd -> paßt

aaacd -> paßt nicht

Wir werden sehen, daß für die Lösung dieser Aufgabe endliche Automaten nützlich sind und daß bei der Implementierung eine spezielle Datenstruktur (DEQUE: double ended queue) sinnvoll eingesetzt werden kann.

Zunächst können wir den regulären Ausdruck in einen nicht-deterministischen Automaten umwandeln:



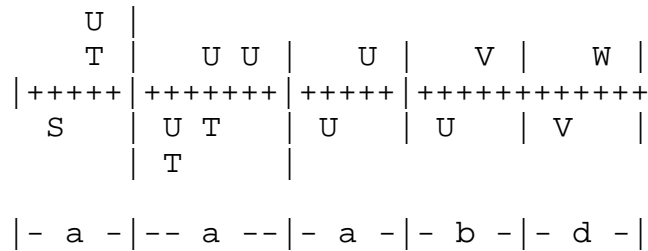
[kompletten DFA für dieses Beispiel bestimmen]

a a a b d:

(S) --a--> (TU) --a--> (U) --a--> (U) --b--> (V) --d--> (W) akzeptiert

Wir wissen, daß wir einen nicht-deterministischen Automaten stets in einen deterministischen Automaten DFA transformieren können (der aber oft sehr groß ist).

Für unsere aktuellen Zwecke ist das aber gar nicht nötig. Was wir bei der Abarbeitung unseres Strings vom DFA wirklich brauchen, enthält die folgende Figur, die wir an Hand des Strings aaabd direkt aus dem nicht-deterministischen Automaten NFA gewinnen können:



In dieser Skizze sind unterhalb der Grenze die Zustände notiert, die zur Abarbeitung anstehen, die Zustände oberhalb der Grenze sind schon für "die nächste Runde" vorgemerkt.

Zur Verwaltung der Zustände während der Abarbeitung des Strings benutzt man sinnvollerweise eine eigene Datenstruktur. Was man braucht, ist eine **DEQUE** (double ended queue), also eine Schlange, bei der man an beiden Enden Zugriff hat:

```

class DEQUE [G]
  top: G          -- vorderstes Element
  pop             -- entferne vorderstes Element
  push (g: G)     -- füge g vorn hinzu
  put (g: G)      -- füge g hinten an
end

```

Literatur: [Sedgewick: Algorithmen, Addison-Wesley, 1992]

Unsere Automatentheorie ist damit fertig, könnte man meinen. Wir haben (sogar vier unterschiedliche) Automatenmodelle und damit reichlich Möglichkeiten, formale Probleme zu beschreiben. Tatsache ist jedoch, dass unsere bisher entwickelte Maschinerie nur recht geringe Reichweite hat. Schon das folgende simple Problem ist für alle unsere Automatenmodelle zu komplex:

Satz: Die formale Sprache $L = \{0^n 1^n, n \geq 0\}$ ist nicht regulär.

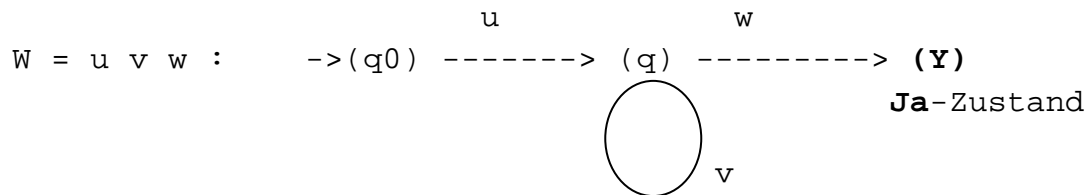
Man beachte: Bei dieser Sprache handelt es sich (in 0 und 1 ausgedrückt) um die Menge der einfach geschachtelten korrekten Klammern.

Die negative Aussage des Satzes ist intuitiv ohne weiteres einleuchtend: Endliche Automaten haben als "Gedächtnis" nichts anderes als ihre endlich vielen Zustände. In denen kann man sich nicht unbegrenzt viele Nullen merken.

Beweis (indirekt): Angenommen L würde von einem endlichen deterministischen Automaten mit k Zuständen akzeptiert. Nehmen wir nun das zu L gehörende Wort

$$W = 0^k 1^k$$

Da dieses Wort doppelt so lang ist wie Zustände vorhanden sind, muß es einen Zustand q geben, der mehr als einmal vorkommt, der also nach Lesen eines Teilwortes v zum zweitenmal angenommen wird:



Damit ist klar, daß alle Worte der Form $u v^i w$ (für beliebiges $i \geq 0$) auch zur Sprache L gehören. Für die Positionierung von v in W kann es nur drei Möglichkeiten geben:

$W = +--u--+v--+-----w-----+$
 $000000000000011111111111111$

$W = +----u-----+--v--+----w-----+$
 $000000000000011111111111111$

$W = +-----u-----+--v--+--w--+$
 $000000000000011111111111111$

Nun ist es offensichtlich in keinem der drei Fälle möglich, das Teilwort v mehrfach zu wiederholen und mit dem Resultat dennoch in L zu bleiben.

Also kann die obige Annahme nur falsch sein. Die Sprache L ist demzufolge nicht regulär. //

Die Methodik dieses Beweises wollen wir noch als allgemeinen Satz festhalten:

Satz (Pumping-Lemma)

Sei L eine reguläre Sprache. Dann gibt es eine Zahl n , so daß sich alle Wörter W aus L mit Mindestlänge n zerlegen lassen in drei Teile

$$W = u v w$$

mit den Eigenschaften: v ist nicht leer, uv hat höchstens die Länge n und

$$u v^i w$$

gehört zu L für beliebige $i \geq 0$.

Übungen zu TI, Teil 1

(1) Für binäre Strings mit denen im Folgenden angegebenen Eigenschaften bestimme man endlichen deterministischen Automaten, einen regulären Ausdruck und eine reguläre Grammatik:

- (a) Strings, in denen 11 vorkommt
- (b) Strings mit genau einer 1
- (c) Strings mit mindestens zwei Bits
- (d) Strings mit genau zwei Bits
- (e) nichtleere Ketten von 01
- (f) Strings, in denen nach jeder 0 (mindestens) eine 1 kommt
- (g) Strings, in denen 0 und 1 sich abwechseln

(2) Man bestimme Grammatiken für die im Folgenden angegebenen formalen Sprachen:

- (a) die binären **Palindrome**, also die Strings, die sich von vorn genauso lesen wie von hinten
- (b) die korrekten Klammerausdrücke, also etwa $(())$ oder $((((()))))$
- (c) die binären Strings mit ebensovielen Nullen wie Einsen.

(3) [Hedtstück] Man konstruiere einen deterministischen Automaten DFA und einen nicht-deterministischen Automaten NFA für die Dateinamen der Form

*a.txt

wobei vor a.txt kein weiterer Punkt vorkommen darf.

(4) Welche Strings in $0,1,2$ liefert die folgende Grammatik?

```
S -> 0SBC|0BC
CB -> BC
0B -> 01 , 1B -> 11
1C -> 12 , 2C -> 22
```

(5) Aus dem Buch von Schöning entnommen ist dieses Beispiel einer nicht-regulären Sprache, für die aber dennoch die Bedingung des Pumping-Lemmas gilt:

$$L = \{c^m a^n b^n \mid m,n \geq 0\} \cup \{a,b\}^*$$