

Automatentheorie und formale Sprachen für Studierende der Technischen Informatik der HAW

Ralph Hollatz

30. September 2015

Kapitel 1 - Sprachen

Version vom 26.11.2014

Sprachen

Vorlesung 1: Sprachen

Vorlesung 2: Endlicher Automat

Vorlesung 3: Endlicher Automat II

Vorlesung 4: Nicht deterministischer endlicher Automat

Vorlesung 5: Epsilon-Endlicher Automat

Vorlesung 6: Reguläre Ausdrücke

Vorlesung 7: Reguläre Ausdrücke II

Vorlesung 8: Pumping-Lemma

Vorlesung 9: Kontextfreie Grammatiken

Vorlesung 10: Exoten

Vorlesung 11: Formale Grammatiken

Motivation 0

Für den Einstieg mit Hilfe des Abenteuers, z.B.: Adventure

Motivation I

Das Vorgehen ist hier über die erweiterte Backus-Naur-Form (EBNF), diese wird gebraucht um Grammatiken zu entwickeln, zu einfachen endlichen Automaten und dann zu den Spezialformen Mealy- und Moore-Automat und dann zu Umwandlungen unter diesen und zu weiteren Spezialisierungen.

Die Backus-Naur-Form (Abk.: BNF) ist eine Notation zur Beschreibung kontextfreier Grammatiken und damit Grundlage der Syntax vieler Programmiersprachen.

Sie verwendet Ersetzungsregeln (**BNF-Regeln oder Produktions**), die eine linke und eine rechte Seite besitzen und in denen Terminal- und Nichtterminalzeichen vorkommen.

Motivation II

Die BNF ist benannt nach J.W. Backus^a und P. Naur^b, die diese Form zur Definition der Syntax der Programmiersprache Algol 60 verwendeten.

```
'COMMENT' HALLO, WELT PROGRAMM IN ALGOL 60;  
'BEGIN'  
    OUTSTRING(2, ('HALLO, WELT'));  
'END'
```

^aJohn Warner Backus (* 1924, † 2007) war Amerikaner und der Sohn eines deutschen Emigranten. Er war Leiter des IBM-Entwicklungsteams von Fortran, der ersten Programmierhochsprache. Er gewann den Turing-Preis 1977.

^bPeter Naur (* 1928) ist Däne und gewann den Turing-Preis 2005.

Regeln der BNF

Die Nichtterminalsymbole der Grammatik werden in der BNF durch **spitze Klammern** $\langle \dots \rangle$ gekennzeichnet. Dadurch können auch ganze Wörter statt nur einzelne Symbole zur Bezeichnung von Nichtterminalsymbolen verwendet werden, wodurch die Bedeutung der Nichtterminalzeichen viel klarer ausgedrückt werden kann.

Die linke und rechte Seite einer Regel werden durch das Symbol $::=$ getrennt.

Gibt es mehrere rechte Seiten für ein Nichtterminalsymbol, so werden diese durch **senkrechte Striche** | getrennt hintereinander geschrieben, ohne dass die linke Seite noch einmal hingeschrieben werden muss.

Ein Beispiel für BNF

Beispiel (Bezeichner / Variable – Anfang)

Ein Bezeichner wird meist durch folgende BNF-Regeln definiert:

<Ziffer> ::= 0|1|2|3|4|5|6|7|8|9

<Buchstabe> ::= A|B|C|D| ... |Z

<Bezeichner> ::= <Buchstabe>|<Buchstabe><Zeichenkette>

<Zeichenkette> ::= <Buchstabe>|<Ziffer>
 |<Buchstabe><Zeichenkette>
 |<Ziffer><Zeichenkette>

Ein Beispiel für BNF

Beispiel (Bezeichner / Variable – Bedeutung)

Eine Ziffer ist eines der Symbole 0, ..., 9 (Regel 1).

Eine Zeichenkette ist entweder ein Buchstabe oder eine Ziffer oder ein Buchstabe, gefolgt von einer Zeichenkette, oder eine Ziffer, gefolgt von einer Zeichenkette (Regel 4).

Zulässige Bezeichner sind also z.B. D, A5, D3F2, nicht aber 7B.

Erweiterte Backus-Naur-Form (EBNF) 0...1

Im Laufe der Entwicklung der BNF wurden weitere Abkürzungsmöglichkeiten eingeführt, wodurch die Erweiterte Backus-Naur-Form (Abk. EBNF) entstand:

1. Symbole oder Symbolfolgen, die auch fehlen können, werden in **eckige Klammern [...]** eingeschlossen.

Beispiel

Die dritte Regel im ersten Beispiel kann man folgendermaßen abkürzen:

$\langle \text{Bezeichner} \rangle ::= \langle \text{Buchstabe} \rangle [\langle \text{Zeichenkette} \rangle]$

Erweiterte Backus-Naur-Form (EBNF) 0...n

2. Symbole oder Symbolfolgen, die beliebig oft wiederholt oder aber auch ganz weggelassen werden können, werden in **geschweifte Klammern** { ... } eingeschlossen.

Innerhalb der geschweiften (wie auch der eckigen) Klammern können wieder mehrere Alternativen angegeben werden. Bei den Alternativen ist die Reihenfolge nicht von Bedeutung.

Beispiel

Statt der dritten und vierten Regel im ersten Beispiel können wir dann schreiben:

<Bezeichner> ::= <Buchstabe> {<Buchstabe>|<Ziffer>}

Aufgabe: BNF für Internetadressen

Internetadressen werden entweder durch vier Zahlen oder durch eine Zeichenkette, den sog. Namen, dargestellt.

Beispiele für symbolische Bezeichnungen und IP-Adressen sind:

robot@vs.informatik.gym-hamburg.de

joe.smith@computer.subdomain.domain

info@192.168.115.113

Versuchen Sie, für diese Arten von Adressen eine EBNF aufzustellen.

Lösung: EBNF für Internetadressen I

Die obigen Adressangaben zerfallen in zwei Teile: den Benutzernamen und die Adresse des Rechners, auf dem der Benutzer zu erreichen ist.

Diese beiden Teile werden durch das @-Zeichen getrennt.

Ein Benutzernamen enthält in unserem Beispiel entweder einen oder zwei Namen, die ggf. durch einen Punkt getrennt sind.

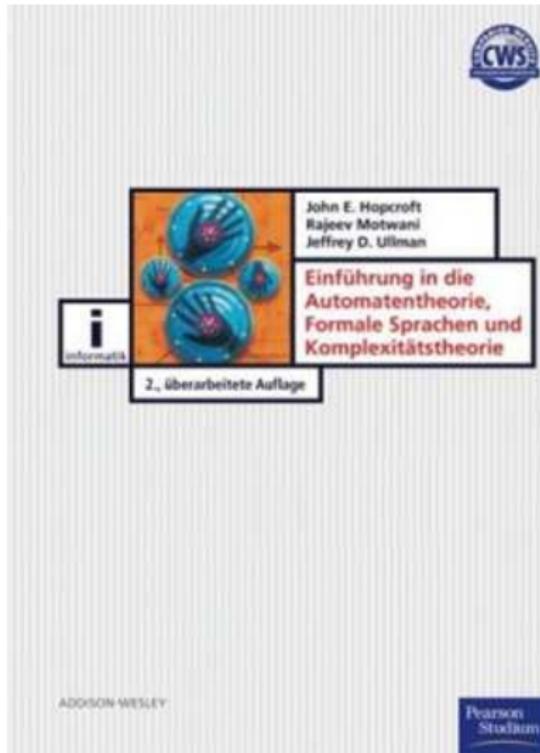
Die Rechneradresse besteht aus einer Folge von mindestens einem Namen, wobei aufeinanderfolgende Namen durch Punkte getrennt sind, oder aus vier Zahlen, die ebenfalls durch Punkte getrennt sind.

Lösung: EBNF für Internetadressen II

```
<Adresse> ::= <User>"@"<Host>
<User>   ::= <Name> ["."]<Name>
<Host>   ::= <NumAdres>|<SymAdres>
<NumAdres> ::= <Zahl>".",<Zahl>".",<Zahl>".",<Zahl>
<SymAdres> ::= <Name> {"."<Name>}
<Name>   ::= <Buchstabe> {<Ziffer>|<Buchstabe>|"-"}
<Zahl>   ::= <Ziffer> {<Ziffer>}
<Ziffer> ::= 0|1|2|3|4|5|6|7|8|9
<Buchstabe> ::= a|b|c|d|...|z
```

Selbststudium

- Das Standardwerk
 - ausführlich
 - verständlich
- Übungen
 - Wichtigste Grundlage dieser Veranstaltung
 - Vorlesung orientiert sich eng am Buch
 - Referenzen auf die Kapitel



ISBN:

978-3-8273-7020-4

ADDITION-WESLEY



Zeit und Raum

- Vorlesung: MI 12:30 - 14:00
Raum: 04.05
- Übung: MI 14:30 - 16:00 oder 16:15 - 17:45
Raum: 04.05

eMail-Adresse:

drhchh@aol.com
oder
ralph.hollatz@haw-hamburg.de

Kapitel 2 - Endlicher Automat

Version vom 16.07.2015

Vorlesung 1: Sprachen

[Vorlesung 2: Endlicher Automat](#)

Vorlesung 3: Endlicher Automat II

Vorlesung 4: Nicht deterministischer endlicher Automat

Vorlesung 5: Epsilon-Endlicher Automat

Vorlesung 6: Reguläre Ausdrücke

Vorlesung 7: Reguläre Ausdrücke II

Vorlesung 8: Pumping-Lemma

Vorlesung 9: Kontextfreie Grammatiken

Vorlesung 10: Exoten

Vorlesung 11: Formale Grammatiken

Automat

Endlicher Automat

Diskrete Ein-/Ausgabe

- ❶ Das System reagiert nur auf diskrete Eingabe-Werte und gibt nur diskrete Werte aus.

Endliche Anzahl von Zuständen

- ❷ Das System kann nur in einer endlichen Zahl von internen Konfigurationen sein, die wir Zustände nennen.

Gedächtnislosigkeit

- ❸ Der Zustand, in dem das System gerade ist, umfasst die gesamte Information über die Vergangenheit des Systems.
- ❹ Nur dieser Zustand entscheidet bei der nächsten Eingabe, was der nächste Zustand ist.

Grundbegriffe I

Definition (Alphabet)

Alphabet ein Zeichenvorrat, genauer eine endliche, nichtleere Menge von Zeichen

Römisches Alphabet = $\{a, b, c, \dots, z\}$

Griechisches Alphabet = $\{\alpha, \beta, \gamma, \dots, \omega\}$

Binäres Alphabet = $\{0, 1\}$

Definition (Wort)

Wort eine endliche Zeichenkette aus Zeichen des Alphabets sprich: Wort über einem Alphabet
Das **leere Wort** ϵ enthält kein Symbol.
Ein Wort wird von dem EA akzeptiert, wenn dieser sich nach Einlesen aller Zeichen von dem Wort in einem seiner Endzustände befindet.

Grundbegriffe II

Definition (Sprache)

Sprache eine Menge von Wörtern.

Die Menge aller akzeptierten Wörter bezeichnet man als die von EA akzeptierte Sprache.

Endliche Sprachen können aufgelistet werden.

Unendliche werden durch Abstraktion dargestellt.

Endlicher Automat

Definition (Endlicher Automat)

Ein **endlicher Automat** ist ein 5-Tupel

$\mathbb{A}_{EA} = (Z, \Sigma, \delta, z_0, E)$, wobei gilt:

Z ist eine **nichtleere endliche** Menge von Zuständen,

Σ ist ein **nichtleerer endlicher** Zeichenvorrat (also ein Alphabet),

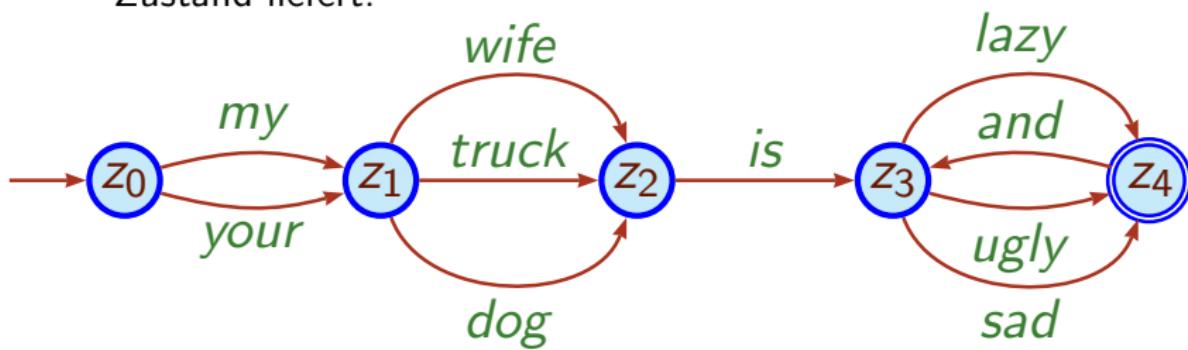
$\delta : Z \times \Sigma \Rightarrow Z$ ist eine Übergangsfunktion (engl.: transition function), wobei gilt: $\delta(z, s)$ ist ein definierter Zustand für jeden Zustand z und Eingabe $s \in \Sigma$,

$z_0 \in Z$ ist ein Anfangszustand,

E ist eine **endliche** Menge von Endzuständen.

Endlicher Automat

- ① Menge von Zuständen Z
- ② Eingabealphabet Σ
- ③ Übergangsfunktion δ , die unter Beachtung der Eingabe und des aktuellen Zustandes einen Übergang in einen ggf. neuen Zustand liefert:



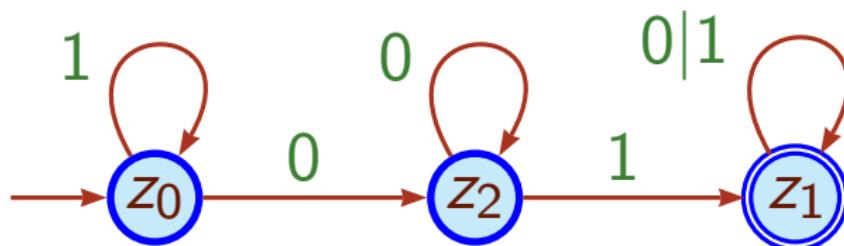
- ④ einen Startzustand z_0
- ⑤ Menge von Endzuständen $E = \{z_4\}$

Zustandsgraphen endlicher Automaten

$\mathbb{A}_{EA} = (Z, \Sigma, \delta, z_0, E)$ mit

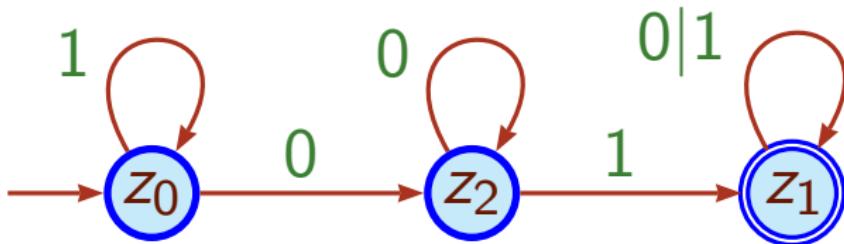
- ❶ Menge von Zuständen
 $Z = \{z_0, z_1, z_2\}$
- ❷ Eingabealphabet $\Sigma = \{0, 1\}$
- ❸ Übergangsfunktion
 $\delta : Z \times \Sigma \Rightarrow Z$ mit
Übergangstabelle:
- ❹ einen Startzustand z_0
- ❺ Endzustände $E = \{z_1\}$

Zustand	Eingabe	
	0	1
$\rightarrow z_0$	z_2	z_0
z_2	z_2	z_1
$* z_1$	z_1	z_1



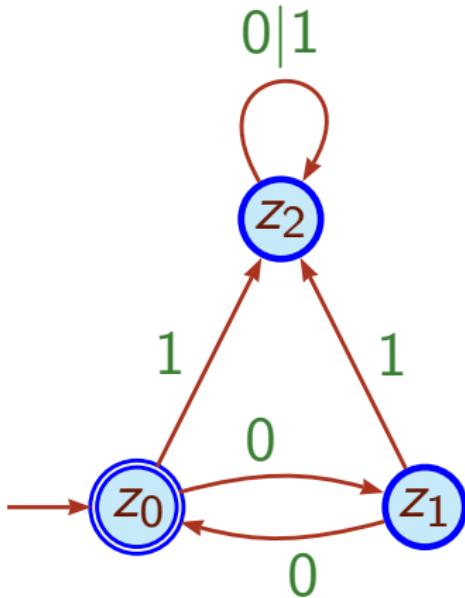
Zustandsgraphen endlicher Automaten

Was leistet dieser DEA (deterministischer endlicher Automat)?
Also welche Worte akzeptiert $\mathbb{A}_{EA} = (Z, \Sigma, \delta, z_0, E)$ mit



Aufgabe

- ① Welche Wörter erkennt \mathbb{A} ? Welche nicht?
Geben Sie bitte Beispiele.
- ② Welche Sprache $L(\mathbb{A})$, also die Menge der Wörter, erkennt \mathbb{A} ?
- ③ Notieren Sie \mathbb{A} formal mathematisch inkl. Übergangstabelle.

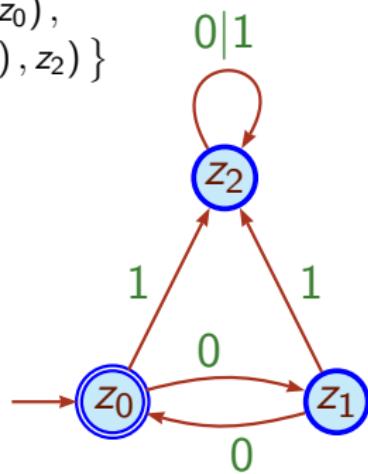


Lösung der Aufgabe

- ① $00, 0000, 000000 \in L(\mathbb{A})$, aber $1001, 1001100 \notin L(\mathbb{A})$
- ② $L(\mathbb{A})$ enthält alle Wörter mit einer geraden Anzahl von Nullen:

$$L(\mathbb{A}) = \{ (00)^n \mid n \geq 0 \}$$
- ③ $\mathbb{A} = (Z, \Sigma, \delta, z_0, E)$ mit
 - ① $Z = \{z_0, z_1, z_2\}$
 - ② $\Sigma = \{0, 1\}$
 - ③ $\delta = \{ ((z_0, 0), z_1), ((z_0, 1), z_2), ((z_1, 0), z_0), ((z_1, 1), z_2), ((z_2, 0), z_2), ((z_2, 1), z_2) \}$
 - ④ einen Startzustand z_0
 - ⑤ Endzustände $E = \{z_0\}$

Zustand	Eingabe	
	0	1
$* \rightarrow z_0$	z_1	z_2
z_1	z_0	z_2
z_2	z_2	z_2



Endlicher Automat

Definition (Moore-Automat)

Ein **Moore-Automat**^a ist ein 6-Tupel

$\mathbb{A}_{Moore} = (Z, \Sigma, T, \delta, \lambda, z_0)$, wobei gilt:

Z, Σ, δ, z_0 sind wie beim endlichen Automaten definiert,

T ist das Ausgabealphabet,

$\lambda : Z \Rightarrow T$ ist eine Abbildung von Z nach T . In **einem Zustand** wird also **ein Zeichen** $t \in T$ ausgegeben.

^aEdward F. Moore, * 1925 † 2003.

Endlicher Automat

Definition (Mealy-Automat)

Ein **Mealy-Automat**^a ist ein 6-Tupel

$$\mathbb{A}_{\text{Mealy}} = (Z, \Sigma, T, \delta, \lambda, z_0), \text{ wobei gilt:}$$

Z, Σ, δ, z_0 sind wie beim endlichen Automaten definiert,

T ist das Ausgabealphabet,

$\lambda : Z \times \Sigma \Rightarrow T$ ist eine Abbildung von $Z \times \Sigma$ nach T . Beim
Übergang von einem Zustand zu einem anderen
Zustand wird also bei **Eingabe** $s \in \Sigma$ **ein Zeichen**
 $t \in T$ ausgegeben.

^aGeorge H. Mealy, * 1927 † 2010.

Der Mealy-Automat I

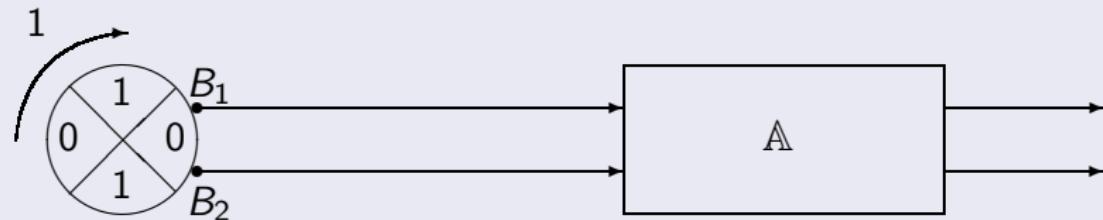


Abbildung:

Aufbau Drehsinn-Automaten \mathbb{A}

In einem Kraftübertragungssystem soll der Umdrehungssinn einer zylindrischen Transmissionswelle durch ein selbsttätig arbeitendes Gerät ständig überwacht werden.

Das Gerät soll entsprechende Signale, die weiter verarbeitet werden können, in regelmäßigen Zeitabständen aussenden. Es soll ein Automat gemäß G.H. Mealy moduliert werden.

Der Mealy-Automat II

Als Messvorrichtung sei an einem Ende der Welle eine Scheibe befestigt, die in vier Sektoren eingeteilt sei, von denen ein Paar gegenüberliegender Sektoren aus leitendem, das andere aus nichtleitendem Material bestehen möge. Die Scheibe sei gegen die Welle isoliert. Über eine die freie Seite der Scheibe überstreichende Bürste stehe die Scheibe stets unter einer konstanten Spannung. Zwei weitere Bürsten B_1 und B_2 seien so angebracht, dass sie den Rand der Scheibe berühren und die verschiedenen Sektoren sich nacheinander an ihnen vorbei bewegen. Die beiden Bürsten müssen so dicht stehen, dass sie auch den kleinsten Sektor gleichzeitig berühren können. Die jeweils an B_1 und B_2 liegenden Spannungen seien die Eingaben für den zu konstruierenden Automaten \mathbb{A} . Bei entsprechender Normierung treten dann nur die Werte 0 und 1 an den Eingängen von \mathbb{A} auf.

Der Mealy-Automat III

Am Ausgang des Automaten möge die Spannung 1 liegen, wenn sich die Scheibe im Uhrzeigersinn dreht, und die Spannung 0, wenn sie sich im entgegengesetzten Sinne dreht.

Was die technische Realisierung anbetrifft, auf deren Details jedoch hier nicht weiter eingegangen werden soll, muss noch die Existenz eines Taktgebers vorausgesetzt werden, der die Zeitpunkte festlegt, zu denen der Automat \mathbb{A} die an den beiden Bürsten liegenden Spannungen feststellen und zum entsprechenden Ausgabesignal verarbeiten soll.

Man beachte, dass die Zeitspannen, innerhalb derer jeweils die Spannungen an den Bürsten festgestellt werden, sehr kurz im Vergleich zur Umdrehungszeit der Scheibe sein müssen.

Der Mealy-Automat IV

Wir haben also hier vier verschiedene Eingabekombinationen (kurz Eingaben) für \mathbb{A} :

$$a = (0, 0), \quad b = (1, 0), \quad c = (1, 1), \quad d = (0, 1),$$

wobei ein Paar (i, j) bedeute, dass an B_1 die Spannung i und an B_2 die Spannung j liege.

Da offensichtlich aus einer einzelnen Eingabe der Automat den Drehsinn nicht bestimmen kann, muss er die vorausgegangenen Eingaben speichern oder sich den jeweiligen Zustand des Systems im vorangegangenen Zeitpunkt merken. Als Zustände des Systems kommen die 8 Paare aus der letzten Eingabe und dem Drehsinn (0 oder 1) zum vorangegangenen Zeitpunkt in Betracht:

$$\begin{aligned} z_1 &= (a, 1), \quad z_2 = (b, 1), \quad z_3 = (c, 1), \quad z_4 = (d, 1), \\ z_5 &= (d, 0), \quad z_6 = (c, 0), \quad z_7 = (b, 0), \quad z_8 = (a, 0). \end{aligned}$$

Der Mealy-Automat V

Aus Zustand und (neuer) Eingabe (etwa z_1 und a oder z_1 und b oder z_1 und d) ergibt sich der Drehsinn (Ausgabe 1 bzw. 0) unmittelbar.

Einige Kombinationen von Zustand und Eingabe sind jedoch nicht erlaubt:

z_1 oder z_8 mit c	z_2 oder z_7 mit d
z_3 oder z_6 mit a	z_4 oder z_5 mit b

Der Mealy-Automat VI

In diesen Fällen muss ein Messfehler vorliegen. Der Automat \mathbb{A} soll dann ein Fehlersignal (1) ausgeben. Es wird angenommen, dass der Fehler behoben ist, wenn \mathbb{A} eine Eingabe erhält, die von der fehlerhaften verschieden ist. \mathbb{A} hat also zwei Ausgänge: einen für die Angabe des Drehsinns und einen für die Fehlermeldung (0, falls kein Fehler vorliegt).

Das ergibt die 4 Ausgabekombinationen (kurz Ausgaben):

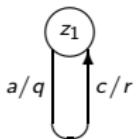
$$p = (0, 0), \quad q = (1, 0), \quad r = (1, 1), \quad s = (0, 1),$$

wobei die erste Komponente in jedem Paar den Drehsinn angibt.

Der Mealy-Automat VII

Wir können nun die Arbeitsweise von \mathbb{A} durch eine Tabelle beschreiben, in der der neue Zustand und die jeweilige Ausgabe in Abhängigkeit vom alten Zustand und der betreffenden Eingabe angegeben sind:

Zustand	Eingabe			
	a	b	c	d
z_1	z_1/q	z_2/q	z_1/r	z_5/p
z_2				
z_3				
z_4				
z_5				
z_6				
z_7				
z_8				



Der Mealy-Automat VIII

Man beachte noch, dass wir keine Voraussetzung gemacht haben, in welchem Zustand sich der Automat zu Beginn seiner Arbeit befinden soll, so dass die ersten Ausgaben falsch sein können. Ebenso kann nach einem Fehler die Ausgabe falsch sein, nach spätestens einer vollen Umdrehung der Welle ist aber (falls kein neuer Fehler eintritt) die Ausgabe korrekt.

Der Automat \mathbb{A} lässt sich sehr schön durch einen Graphen beschreiben. Man vergleiche die Richtungen der Pfeile von z_1 nach z_2 , bzw. z_5 nach z_6 usw. mit dem Drehsinn der Scheibe.

Der Mealy-Automat IX

Das gesuchte Anzeigegerät lässt sich also beschreiben als ein Automat mit

der Eingabemenge

$$\Sigma = \{a, b, c, d\},$$

der Zustandsmenge

$$Z = \{z_1, z_2, z_3, z_4, z_5, z_6, z_7, z_8\}$$

und der Ausgabemenge

$$T = \{p, q, r, s\},$$

dessen Arbeitsweise (Änderung des jeweiligen Zustandes und Ausgabe, beides bewirkt durch eine Eingabe **im** betreffenden Zustand) dargestellt wird durch obige Tabelle, d.h.

durch die zwei Funktionen

$$\delta : Z \times \Sigma \rightarrow Z$$

und

$$\lambda : Z \times \Sigma \rightarrow T.$$

Einige erste Aufgaben I

Aufgaben I

Gegeben sei $\Sigma = \{0, 1\}$

Entwickeln Sie einen DEA \mathbb{A} , der die Sprache

$$L(\mathbb{A}) = \{w \mid w \text{ hat eine gerade Anzahl von } 0\text{'en}\}$$

erkennt, z.B. sind $1110000, 0110, 0010110 \in L(\mathbb{A})$

Einige erste Aufgaben II

Aufgaben II

Gegeben sei $\Sigma = \{0, 1\}$

Entwickeln Sie einen DEA \mathbb{A} , der die Sprache

$$L(\mathbb{A}) = \{w \mid w \text{ hat eine ungerade Anzahl von } 0\text{'en}\}$$

erkennt, z.B. sind $111000, 011, 010110 \in L(\mathbb{A})$

Kapitel 3 - Endlicher Automat II

Version vom 16.07.2015

Automat

Vorlesung 1: Sprachen

Vorlesung 2: Endlicher Automat

Vorlesung 3: Endlicher Automat II

Vorlesung 4: Nicht deterministischer endlicher Automat

Vorlesung 5: Epsilon-Endlicher Automat

Vorlesung 6: Reguläre Ausdrücke

Vorlesung 7: Reguläre Ausdrücke II

Vorlesung 8: Pumping-Lemma

Vorlesung 9: Kontextfreie Grammatiken

Vorlesung 10: Exoten

Vorlesung 11: Formale Grammatiken

Der Moore-Automat I

Zur Einführung betrachten wir ein vereinfachtes Beispiel aus dem Gebiet der Programmiersprachen. Zum Verständnis ist es nötig, dass hierfür der Gebrauch der metalinguistischen^a „Formeln“ der Backus-Naur-Form vertraut ist. Alles weitere ist nach E.F. Moore^b.

Sei $\Sigma_D = \{0, 1, .\}$. Mit folgenden 8 metalinguistischen Formeln, die der Reihe nach mit den Buchstaben m, n, \dots, t bezeichnet werden sollen, wird diejenige Teilmenge von $L(\Sigma_D)$ definiert, deren Elemente als vorzeichenlose Dualzahlen gelesen werden können.

^aMetalinguistisch bedeutet über Sprache mittels Sprache zu reden

^bEdward Forrest Moore, * 1925 † 2003

Der Moore-Automat II

<i>m</i> :	<vorzeichenlose Dualzahl>	::=	<ganze Dualzahl>
<i>n</i> :	<vorzeichenlose Dualzahl>	::=	<Dualbruch>
<i>o</i> :	<vorzeichenlose Dualzahl>	::=	<ganze Dualzahl> <Dualbruch>
<i>p</i> :	<Dualbruch>	::=	<Dualpunkt> <ganze Dualzahl>
<i>q</i> :	<ganze Dualzahl>	::=	<ganze Dualzahl> <Dualziffer>
<i>r</i> :	<ganze Dualzahl>	::=	<Dualziffer>
<i>s</i> :	<Dualpunkt>	::=	.
<i>t</i> :	<Dualziffer>	::=	0 1

Der Moore-Automat III

Wir wollen eine Auswertung einer metalinguistischen Formel eine Rechtsauswertung nennen, wenn wir jeweils die am weitesten rechts stehende metalinguistische Variable gemäß einer Formel ersetzen. Die vorzeichenlose Dualzahl $.1$ erhält man z.B. durch Rechtsauswertung der Formel n wie folgt:

<vorzeichenlose Dualzahl>	
::=	<Dualbruch> <i>n</i>
::=	<Dualpunkt> <ganze Dualzahl> <i>p</i>
::=	<Dualpunkt> <Dualziffer> <i>r</i>
::=	<Dualpunkt> 1 <i>t</i>
::=	.1 <i>s</i>

Der Moore-Automat IV

Zur Beschreibung dieses Auswertungsprozesses genügt es, die benutzten Formeln in der Reihenfolge ihrer Anwendung, d.h. die sog. Rechtsauswertungsfolge, anzugeben, also:

n p r t s.

Man erhält z.B. 10.011 durch folgende Rechtsauswertungsfolge:

Der Moore-Automat V

Unser Ziel soll es jetzt sein, einen Analysator für vorzeichenlose Dualzahlen zu entwerfen, d.h. einen Automaten, der wie folgt arbeitet:

Sei $\Sigma = \Sigma_D \cup \{_\}$, wobei $_$ ein von den Elementen von Σ_D verschiedenes Zeichen sei, das als Begrenzungszeichen dient. Legt man dem Automaten ein Wort w aus $L(\Sigma)$ vor, so soll er es von links nach rechts Zeichenweise lesen und jedes Mal, wenn er ein Zeichen gelesen hat ein Zeichen aus:

$$T = \{m, s, tr, tq, po, pn, _, ?\}$$

ausgeben derart, dass folgendes gilt:

Wenn $w = u__$ und u eine vorzeichenlose Dualzahl im Sinne obiger Definition ist, soll die Folge der Ausgaben (abgesehen von der letzten) gerade die Rechtsauswertungsfolge für u in umgekehrter Reihenfolge sein.

Der Moore-Automat VI

Hat der Automat das ganze Wort w gelesen, so macht er solange „ \downarrow “, bis er eine neue vorzeichenlose Dualzahl als Lesestoff erhält. Ferner wollen wir, der Vollständigkeit halber vorsehen, dass der Automat eine Fehlermeldung (das Zeichen „?“) ausgeben kann - sie muss mindestens einmal erscheinen, wenn das vorgelegte Wort nicht die Form $u\perp\perp$ hat.

Im Übrigen soll uns jedoch nicht weiter interessieren, wie der Automat auf Worte, die nicht die gewünschte Form haben, reagiert.

Wir gehen jetzt von einer etwas anderen Vorstellung von der Arbeitsweise eines Automaten aus als beim Mealy-Automaten. Und zwar soll in jedem Zustand des Automaten eine bestimmte Ausgabe erfolgen – unabhängig von der zuvor erhaltenen Eingabe. Als Zustände des Automaten bieten sich dann die folgenden Situationen an - die jeweilige Ausgabe wird gleich mit angegeben:

z_0	Start (Vorlage einer Dualzahl erwartet)	
z_1	Nur eine Dualziffer gelesen	<i>tr</i>
z_2	Nur Dualziffern gelesen, aber mindestens zwei	<i>tq</i>
z_3	Nur Dualziffern und einmal „ „ gelesen	<i>m</i>
z_4	Nach mindestens einer Dualziffer einen Dualpunkt gelesen	<i>s</i>
z_5	Im Zustand z_4 eine Dualziffer gelesen	<i>tr</i>
z_6	Nach Erreichen von z_4 mindestens zwei Dualziffern gelesen	<i>tq</i>
z_7	Nach Erreichen von z_4 mindestens eine Dualziffer und einmal „ „ gelesen	<i>po</i>
z_8	Erstes gelesenes Zeichen war der Dualpunkt	<i>s</i>
z_9	In z_8 mindestens eine Dualziffer gelesen	<i>tr</i>
z_{10}	Nach Erreichen von z_8 mindestens zwei Dualziffern gelesen	<i>tq</i>
z_{11}	In z_9 oder z_{10} ein „ „ gelesen	<i>pn</i>
z_{12}	Zustand, der in allen anderen Fällen eingenommen wird – verlassbar nur durch ein „ „	?

Der Moore-Automat VIII

Jetzt ist klar, wie der Automat arbeitet. Wir beschreiben ihn durch einen gerichteten Graphen, dessen Ecken den Zuständen entsprechen – außer der Bezeichnung des Zustandes wird in die Ecken auch die Ausgabe, die dieser Zustand erzeugt, geschrieben.

Die Kanten des Graphen geben die durch die Eingaben bestimmten Zustandsübergänge an (dabei schreiben wir der Kürze halber „sonst“ an eine Kante, wenn an diese Kante alle sonst noch nicht an einer von der gleichen Ecke wegführenden Kante stehenden Eingabezeichen geschrieben werden müssten – besser ist aber alle möglichen Eingaben durch „|“ getrennt aufzulisten).

Umwandlung Mealy- in einen Moore-Automaten I

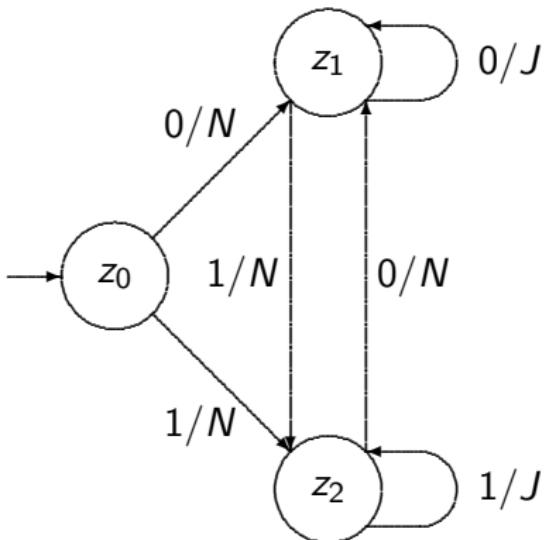
Satz

Jeder Mealy-Automat kann durch Hinzufügen von Zuständen zu einem äquivalenten Moore-Automaten gemacht werden. Die maximale Anzahl der Zustände im Moore-Automat beträgt dann:

$$\text{Anzahl(Mealy-Zustände)} \times \text{Anzahl(Mealy-Ausgaben)}.$$

Umwandlung Mealy- in einen Moore-Automaten II

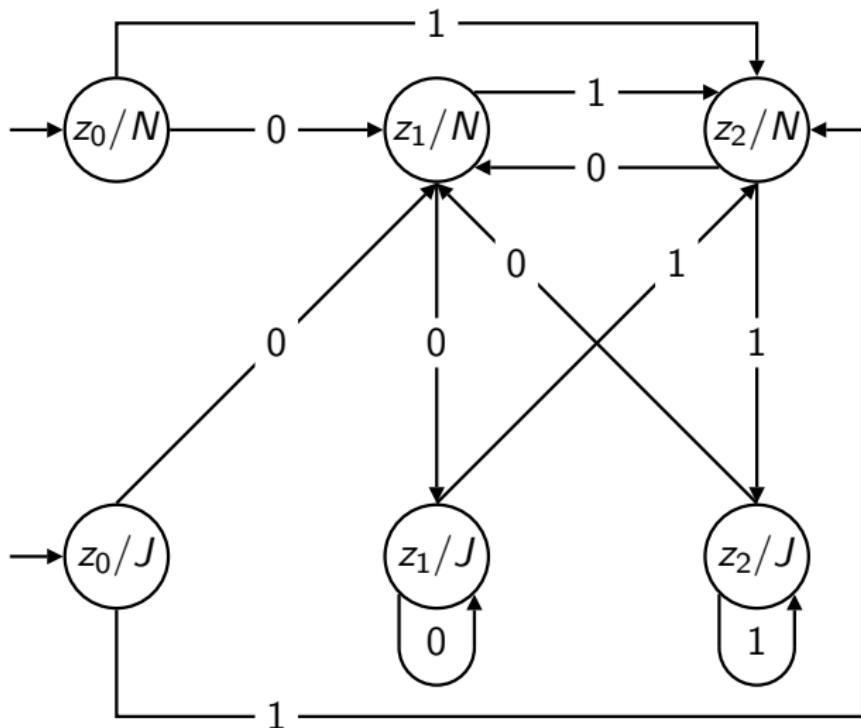
Hier sehen wir einen typischen Mealy-Automaten. Seine (für dieses Thema eigentlich unwichtige) Aufgabe: Prüfe, ob die zwei zuletzt eingegebenen Werte (0 oder 1) gleich waren, je nach Ergebnis soll er ein J oder ein N ausgeben (Ja/Nein).



Umwandlung Mealy- in einen Moore-Automaten III

Nun nimmt man sich den Mealy-Automaten und schreibt für jeden der vorhandenen Zustände so viele auf ein Blatt Papier, wie es mögliche Ausgaben gibt. Zur besseren Übersicht hängt man die Ausgabe, die jetzt im Zustand erfolgt, mit einem Schrägstrich / an den entsprechenden Zustand an. Die Übergangspfeile ergeben sich, indem man für jeden Zustand die möglichen Eingaben ermittelt und prüft, welche Ausgaben beim Mealy-Automaten auftreten würden. Mittels dieser Ausgabe und dem Folgezustand im Mealy-Automaten bekommt man dann den Folgezustand im Moore-Automaten.

Umwandlung Mealy- in einen Moore-Automaten III



Äquivalenter Moore-Automat zum vorherigen Mealy-Automaten.

Minimalisieren von Automaten I

Zum Minimalisieren eines Automaten gibt es folgende einfache Vorgehensweise: (Auch als Table-filling-Algorithmus bekannt.)

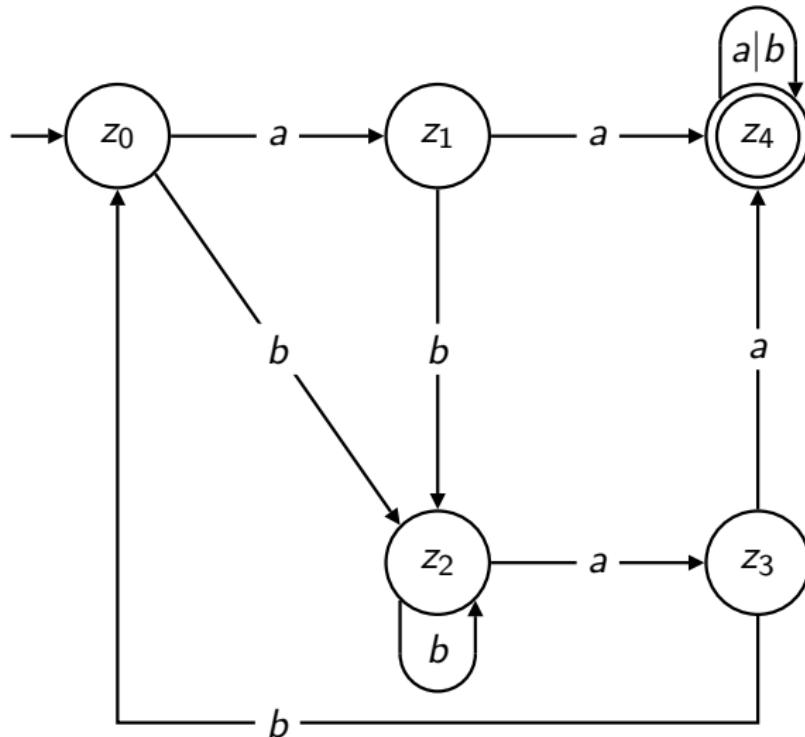
- 0 Entfernen Sie alle nicht erreichbaren Zustände.

Erstellung der Überdeckungsmatrix mit folgenden Regeln:

- 1 Alle End- und Nicht-End-Zustände sind nicht äquivalent ($-$).
- 2 Für jedes noch unmarkierte Paar $\{z_i, z_j\}$ und jedes $s \in \Sigma$ testen Sie, ob $\{\delta(z_i, s), \delta(z_j, s)\}$ nicht bereits markiert ist.
Wenn ja, markieren Sie auch $\{z_i, z_j\}$ (\ominus).
- 3 Fortsetzen bis keine neuen Nicht-Äquivalenzen mehr hinzugefügt werden können.
- 4 Alle jetzt noch unmarkierten Paare können jeweils zu einem Zustand verschmolzen werden.

Minimalisieren von Automaten II

Folgendes Zustandsübergangsdiagramm sei gegeben:



Minimalisieren von Automaten III

Erstellen der Überdeckungsmatrix mit den Regeln eins bis drei:

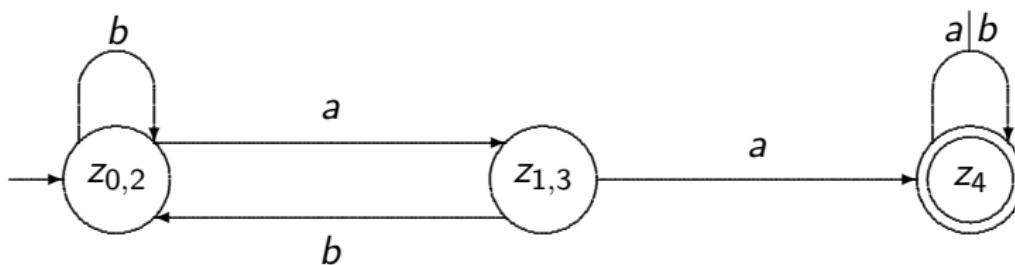
z_1	\ominus			
z_2		\ominus		
z_3	\ominus		\ominus	
z_4	—	—	—	—
	z_0	z_1	z_2	z_3

- | | |
|---|---|
| $\{\delta(z_0, a), \delta(z_1, a)\} \rightarrow \{z_1, z_4\}$ markieren | $\{\delta(z_0, b), \delta(z_1, b)\} \rightarrow \{z_2\}$ |
| $\{\delta(z_0, a), \delta(z_2, a)\} \rightarrow \{z_1, z_3\}$ | $\{\delta(z_0, b), \delta(z_2, b)\} \rightarrow \{z_2\}$ |
| $\{\delta(z_0, a), \delta(z_3, a)\} \rightarrow \{z_1, z_4\}$ markieren | $\{\delta(z_0, b), \delta(z_3, b)\} \rightarrow \{z_2, z_0\}$ |
| $\{\delta(z_1, a), \delta(z_2, a)\} \rightarrow \{z_4, z_3\}$ markieren | $\{\delta(z_1, b), \delta(z_2, b)\} \rightarrow \{z_2\}$ |
| $\{\delta(z_1, a), \delta(z_3, a)\} \rightarrow \{z_4\}$ | $\{\delta(z_1, b), \delta(z_3, b)\} \rightarrow \{z_2, z_0\}$ |
| $\{\delta(z_2, a), \delta(z_3, a)\} \rightarrow \{z_3, z_4\}$ markieren | $\{\delta(z_2, b), \delta(z_3, b)\} \rightarrow \{z_2, z_0\}$ |

Minimalisieren von Automaten IV

Mit Regel vier gilt offensichtlich, dass z_0 äquivalent zu z_2 ist und z_1 äquivalent zu z_3 ist. Hiermit lässt sich nun abschließend der reduzierte Automat aufstellen.

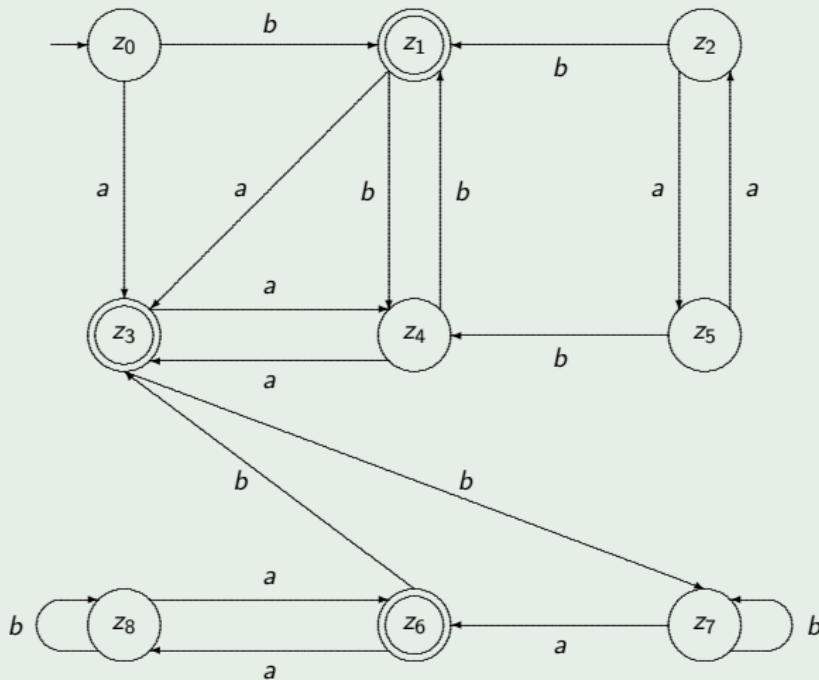
Man erhält:



Einige erste Aufgaben III

Aufgaben III

Folgendes Zustandsübergangsdiagramm sei gegeben:



Einige erste Aufgaben III Lösungen – Anfang

Einige erste Aufgaben IV

Aufgaben IV

Konstruieren Sie einen DEA, der alle Ketten aus $\{0, 1\}^*$ akzeptiert, die mit einer 00 beginnen und bei denen auf jede 1 unmittelbar mindestens eine 0 folgt.

Einige erste Aufgaben V

Aufgaben V

Konstruieren Sie einen DEA, der alle Ketten aus $\{0, 1, 2\}^+$ akzeptiert, so dass jeder 2 direkt die Kette 10 folgt.

Einige erste Aufgaben VI

Aufgaben VI

$L = \{ w \mid w \text{ beginnt mit der Prefix } „bab“ \text{ oder}$
 $w \text{ beinhaltet mindestens vier } b\text{'s} \}$ wobei $\Sigma = \{a, b\}$

Einige erste Aufgaben VII

Aufgaben VII

Konstruieren Sie eine Mealy-Maschine, die Ketten aus $\{0, 1\}^+$ liest und die Eingabe um zwei Zeichen versetzt reproduziert. Die Ausgabe beginnt mit „xx“ (vorausgesetzt, dass mindestens zwei Zeichen eingegeben werden), die letzten beiden Zeichen der Eingabe werden nicht ausgegeben.

Beispiel

Eingabe 011101

Ausgabe xx0111

Einige erste Aufgaben VII

Kapitel 4 - Nicht deterministischer endlicher Automat

Version vom 16.07.2015

Automat

Vorlesung 1: Sprachen

Vorlesung 2: Endlicher Automat

Vorlesung 3: Endlicher Automat II

Vorlesung 4: Nicht deterministischer endlicher Automat

Vorlesung 5: Epsilon-Endlicher Automat

Vorlesung 6: Reguläre Ausdrücke

Vorlesung 7: Reguläre Ausdrücke II

Vorlesung 8: Pumping-Lemma

Vorlesung 9: Kontextfreie Grammatiken

Vorlesung 10: Exoten

Vorlesung 11: Formale Grammatiken

Minimalisierung von Automaten V

Bisher hatten wir die Äquivalenz von endlichen Automaten und regulären Sprachen betrachtet. Die Automaten, die wir auf diese Weise erhalten, besitzen eine Vielzahl von Zuständen, so dass die Frage entsteht, ob man sie nicht auch vereinfachen kann. Wir haben nun einen Algorithmus, der zu einem endlichen Automaten einen Automaten konstruiert, der dieselben Eingaben akzeptiert, aber die minimale dafür nötige Anzahl von Zuständen besitzt. Wir haben dabei zunächst nur deterministische endliche Automaten ohne Ausgabe betrachtet.

Da man zu jedem NEA einen äquivalenten DEA konstruieren kann, ist dies keine Einschränkung. Außerdem soll $\delta(z, s)$ für alle Paare (z, s) vollständig definiert sein. Notfalls ist ein Fehlerzustand nachzurüsten. Wir werden auch sehen, dass dieser Automat (bis auf Isomorphie) eindeutig bestimmt ist. Aber um dieses Ergebnis zu beweisen, benötigen wir noch einige Tatsachen, die wir uns jetzt ansehen werden.

Äquivalenz, Minimaler Automat

Definition (Äquivalenz zweier Automaten)

Zwei Automaten heißen **äquivalent**, wenn sie dieselbe Sprache akzeptieren.

Definition (Minimale Automaten)

Ein deterministischer Automat heißt **minimal**, wenn sie keinen äquivalenten deterministischen Automaten mit weniger Zuständen gibt.

Eindeutigkeit minimaler DEAs

Satz

Zu einem minimalen DEA \mathbb{M} gibt es keinen anderen DEA \mathbb{N} mit weniger Zuständen, der dieselbe Sprache akzeptiert.

Erläuterung

Jeder zu \mathbb{M} äquivalente DEA kann nur anders benannte Zustände haben.

Beweis der Eindeutigkeit minimaler DEAs

Beweis durch Widerspruch

- ▶ Annahme: Sei \mathbb{M} minimal und \mathbb{N} habe weniger Zustände.
- ▶ Startzustände sind nicht unterscheidbar wg. $L(\mathbb{M}) = L(\mathbb{N})$.
- ▶ Nachfolger der Startzustände nicht unterscheidbar für jedes Eingabesymbol s , weil Startzustände nicht unterscheidbar.
- ▶ Durch Zeichenreihen $s_1 s_2 \dots s_k$ lässt sich schließen:
Jeder Zustand von \mathbb{M} ist von mindestens einem Zustand von \mathbb{N} nicht unterscheidbar^a.
- ▶ Da \mathbb{N} weniger Zustände als \mathbb{M} besitzt, gibt es **mindestens zwei** Zustände in \mathbb{M} , die von demselben Zustand in \mathbb{N} nicht unterscheidbar sind.
Daher auch untereinander nicht unterscheidbar.
- ▶ Also \mathbb{M} nicht minimal. **Widerspruch** □

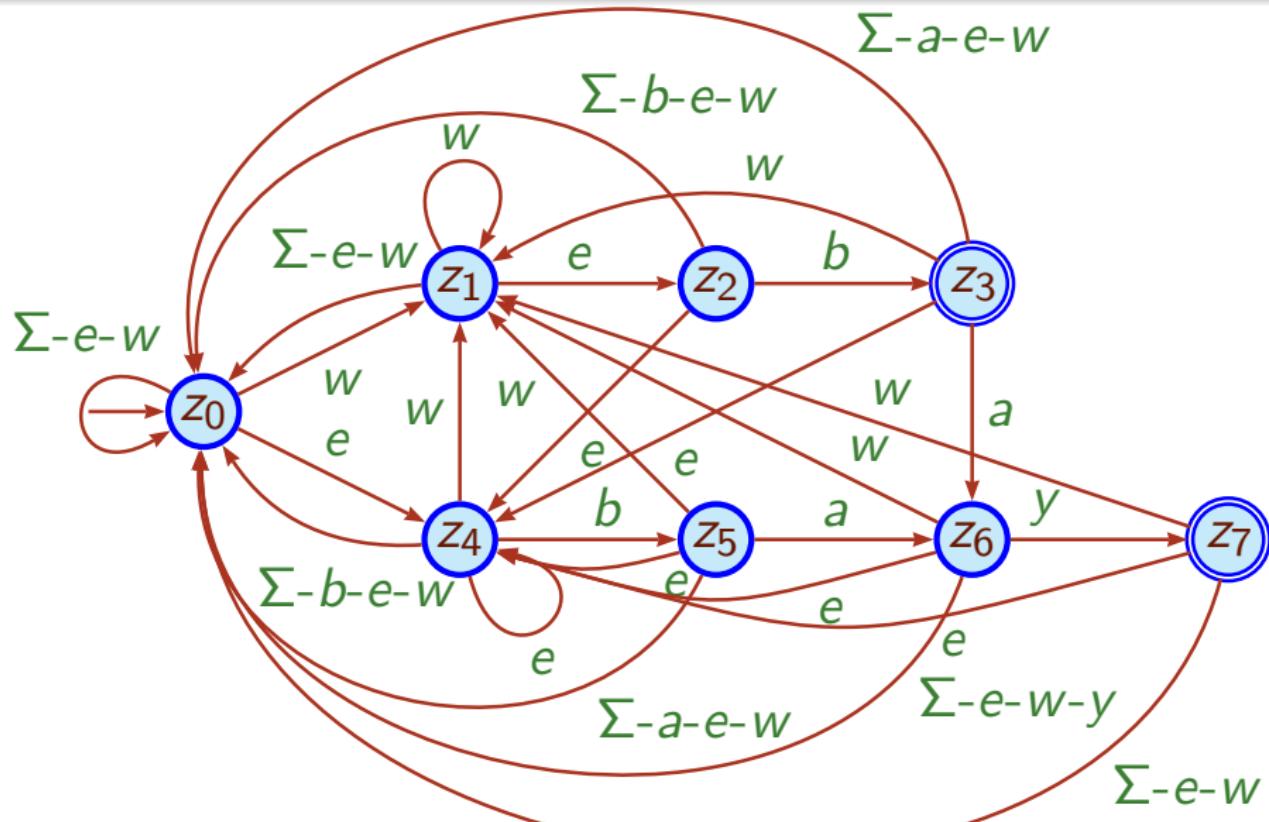
^aWeder \mathbb{M} noch \mathbb{N} haben nicht-erreichbare Zustände

Nichtdeterministischer endlicher Automat (NEA) I

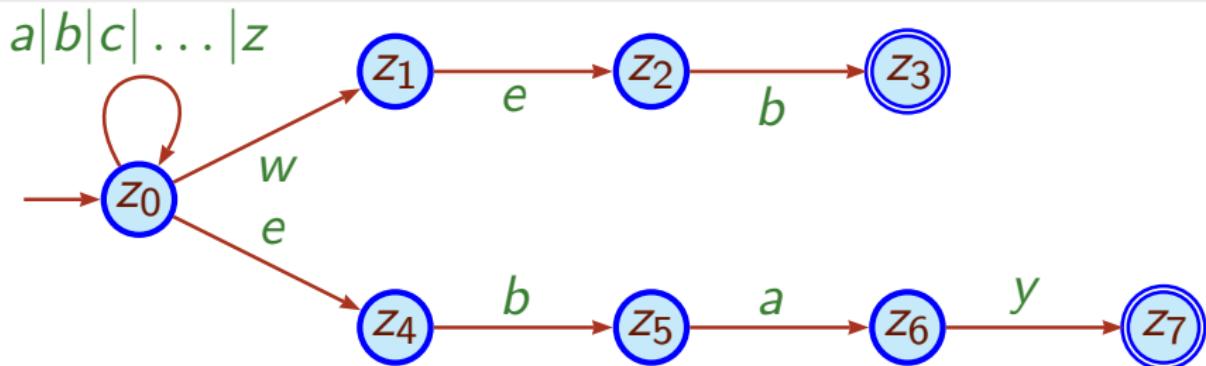
Warum überhaupt ein NEA?

- ① Für eine gegebene Aufgabenstellung ist es meist wesentlich einfacher, einen NEA zu konstruieren als einen DEA.
- ② Jeder NEA lässt sich **automatisch** in einen äquivalenten DEA überführen. Das werden wir aber erst später sehen.

Beispiel: DEA



Beispiel: NEA



Was hat sich verändert?

- ① Im Zustand z_0 kann der Automat mit „e“ nach z_0 oder z_4 übergehen.
- ② Im Zustand z_4 hat der Automat mit z.B. „a“ keinen Übergang.
- ③ Im Zustand z_3 hat der Automat mit „a“ keinen Übergang zu z_6 .

Nichtdeterministischer endlicher Automat (NEA) II

Definition (**nichtdeterministischer endlicher Automat**)

Ein **nichtdeterministischer endlicher Automat** (NEA)^a ist ein endlicher Automat, bei dem es Zustände gibt, welche zu einem Eingabezeichen verschiedene Möglichkeiten des Zustandsübergangs erlauben.

Ein NEA und ein DEA unterscheiden sich also **nur** durch den Typ des Rückgabewerts von δ :

NEA es ist eine **Teilmenge** der Zustände.

DEA es ist ein einziger Zustand.

^aNEA engl. NFA und DEA engl. DFA

Nichtdeterministischer endlicher Automat (NEA) III

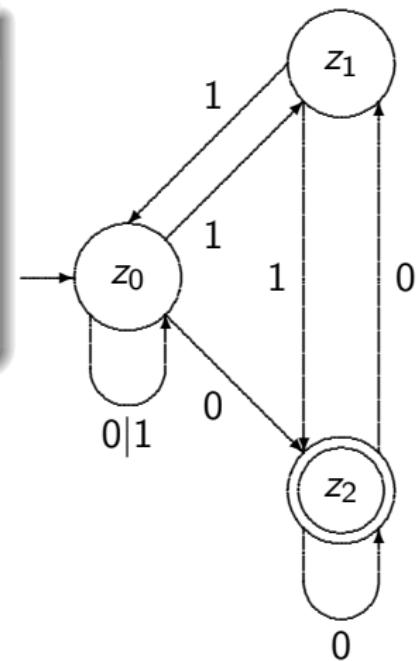
Beispiel

Der rechts abgebildete Automat \mathbb{A} mit Startzustand z_0 akzeptiert die Eingabe 111000 auf unterschiedliche Möglichkeiten. Welche?

Die Eingabe 101 wird nicht akzeptiert.
Warum?

Da bei einem NEA schwer zu erkennen ist, welche Eingaben er akzeptiert, wäre es schön, wenn man einen äquivalenten deterministischen endlichen Automaten (DEA) entwickeln könnte (DEA sind die Automaten, die wir bis jetzt immer hatten).

Genau das ist **immer** möglich!



Nichtdeterministischer endlicher Automat (NEA) IV

Satz

Zu jedem nichtdeterministischen endlichen Automaten (NEA) \mathbb{A} (mit n Zuständen) gibt es einen deterministischen endlichen Automaten (DEA) \mathbb{A}' mit maximal 2^n Zuständen.

Der Beweis ist äußerst umfangreich, wenn er allgemein geführt werden soll. Wir benutzen den Satz deshalb einfach und wenden ihn auf obiges Beispiel an, das Vorgehen ist dabei auf jeden anderen Automaten übertragbar. Der Startzustand ist z_0 .

- 1 Der Automat \mathbb{A}' bekommt den Startzustand z_0 .
 - + Schau nach, welche Zustände von z_0 mit der Eingabe „0“ aus erreichbar sind. Dies sind z_0 und z_2 . Füge für den neuen Automaten einen neuen Zustand $z_{0,2}$ ein.
 - + Schau nach, welche Zustände von z_0 mit der Eingabe „1“ aus erreichbar sind. Dies sind z_0 und z_1 . Füge für den neuen Automaten einen neuen Zustand $z_{0,1}$ ein.

Nichtdeterministischer endlicher Automat (NEA) V

- + ggf. weiter mit weiteren Eingaben.
 - + protokolliere deine Zustände in Form einer Tabelle.
- 2** Der Automat \mathbb{A}' wurde um die Zustände $z_{0,2}$ und $z_{0,1}$ erweitert. Arbeitet nun diese nacheinander, wie bei z_0 , ab:
- Zuerst $z_{0,2}$.
 - + Schau nach, welche Zustände von $z_{0,2}$, also z_0 oder z_2 , mit der Eingabe „0“ aus erreichbar sind. Dies sind z_0 , z_1 und z_2 . Füge für den neuen Automaten einen neuen Zustand $z_{0,1,2}$ ein.
 - + Schau nach, welche Zustände von $z_{0,2}$, also z_0 oder z_2 , mit der Eingabe „1“ aus erreichbar sind. Dies sind z_0 und z_1 . Diesen Zustand gibt es aber schon.
 - + ggf. weiter mit weiteren Eingaben.
 - + Erweitere die Tabelle um die eben genannten Zustandsübergänge.

Nichtdeterministischer endlicher Automat (NEA) VI

- Nun $z_{0,1}$.
- + Schau nach, welche Zustände von $z_{0,1}$, also z_0 oder z_1 , mit der Eingabe „0“ aus erreichbar sind. Diesen Zustand gibt es aber schon.
- + Schau nach, welche Zustände von $z_{0,1}$, also z_0 oder z_1 , mit der Eingabe „1“ aus erreichbar sind. Dies sind z_0 , z_1 und z_2 . Diesen Zustand gibt es allerdings schon.
- + ggf. weiter mit weiteren Eingaben.
- + protokolliere deine Zustände in Form einer Tabelle.

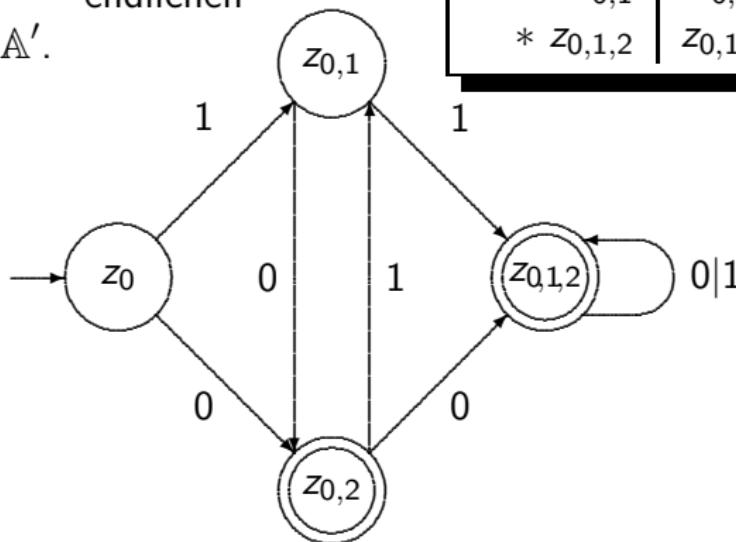
Nichtdeterministischer endlicher Automat (NEA) VII

- 3** Der Automat \mathbb{A}' wurde um den Zustand $z_{0,1,2}$ erweitert.
Arbeite nun diesen, wie bei z_0 , ab:
- + Schau nach, welche Zustände von $z_{0,1,2}$, also z_0 oder z_1 oder z_2 , mit der Eingabe „0“ aus erreichbar sind. Dies sind alle Zustände, also z_0 , z_1 und z_2 . Diesen Zustand gibt es allerdings schon.
 - + Schau nach, welche Zustände von $z_{0,1,2}$, also z_0 oder z_1 oder z_2 , mit der Eingabe „1“ aus erreichbar sind. Dies sind alle Zustände, also z_0 , z_1 und z_2 . Diesen Zustand gibt es allerdings schon.
 - + ggf. weiter mit weiteren Eingaben.
- 4** Da z_2 Endzustand war werden $z_{0,2}$ und $z_{0,1,2}$ auch Endzustand, d. h. alle Zustände, die mit Zustand z_2 kombiniert wurden.

Nichtdeterministischer endlicher Automat (NEA) VIII

Ist das Verfahren abgeschlossen, so erhält man die folgende Zustandsübergitungstabelle bzw. den darunter dargestellten endlichen Automaten \mathbb{A}' .

Zustand	Eingabe	
	0	1
$\rightarrow z_0$	$z_{0,2}$	$z_{0,1}$
$* z_{0,2}$	$z_{0,1,2}$	$z_{0,1}$
$z_{0,1}$	$z_{0,2}$	$z_{0,1,2}$
$* z_{0,1,2}$	$z_{0,1,2}$	$z_{0,1,2}$



Besondere Merkmale eines NEA

- ▶ Mehrere Zustände erreichbar
- ▶ Alternativen werden „parallel“ betrachtet
- ▶ Übergangsfunktion liefert Menge von Zuständen
- ▶ Zustandsüberführungen:
 - ▶ nicht wie in Programmen
 - ▶ eher wie in Regelsystemen
- ▶ Berechnungen sind nicht eindeutig bestimmt:
Das Ergebnis hängt von der geschickten Anwendung der Regeln ab.

historisch: [Turing]

- ▶ „Wahl-“ maschinen (c-machines, c wie choice)
 - (nichtdeterministisch)
- ▶ automatische Maschinen
 - (deterministisch)

Einige erste Aufgaben VIII

Aufgaben VIII

Es soll ein endlicher Automat \mathbb{A} gebildet werden.

Gegeben sei $\Sigma = \{a, b, c\}$ und $Z = \{z_0, z_1, z_2, z_3\}$, dessen Übergangsfunktion durch die nebenstehende Tabelle definiert ist.

Es gilt $z_0 = \text{Anfangszustand}$ und $z_3 = \text{Endzustand}$.

Zustand	Eingabe		
	a	b	c
$\rightarrow z_0$	z_1	z_3	z_2
z_1	z_0	z_1	z_0
z_2	z_1	z_1	z_3
$* z_3$	z_3	z_0	z_2

- ➊ Zeichnen Sie den Übergangsgraphen für diesen Automaten.
- ➋ Welche der folgenden Wörter gehören zum akzeptierten Sprachschatz dieses Automaten: abc , a^3bc^3 , $a^2b^2c^2$, $a^3b^2c^2$

Einige erste Aufgaben VIII und Lösungen

Einige erste Aufgaben IX

Aufgaben IX

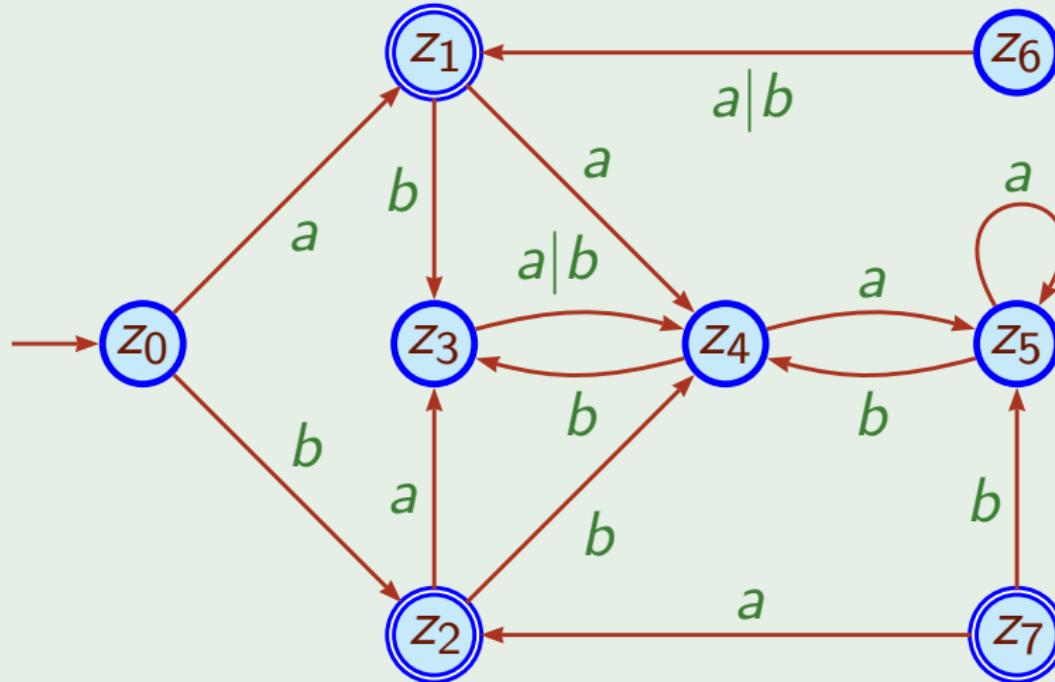
- 1 Geben Sie den Übergangsgraphen und die Übergangstabelle eines endlichen deterministischen Automaten mit $\Sigma = \{a, b\}$ an, dessen akzeptierter Sprachschatz $L(\mathbb{A})$ aus der Menge aller Worte aus Σ^* besteht, die mit a beginnen und bb nicht als Teilsting enthalten.
- 2 Formulieren Sie außerdem $L(\mathbb{A})$ in der üblichen Mengenschreibweise.

Einige erste Aufgaben IX und Lösungen

Einige erste Aufgaben X

Aufgaben X

Berechnen Sie bitte den minimierten Automaten von:

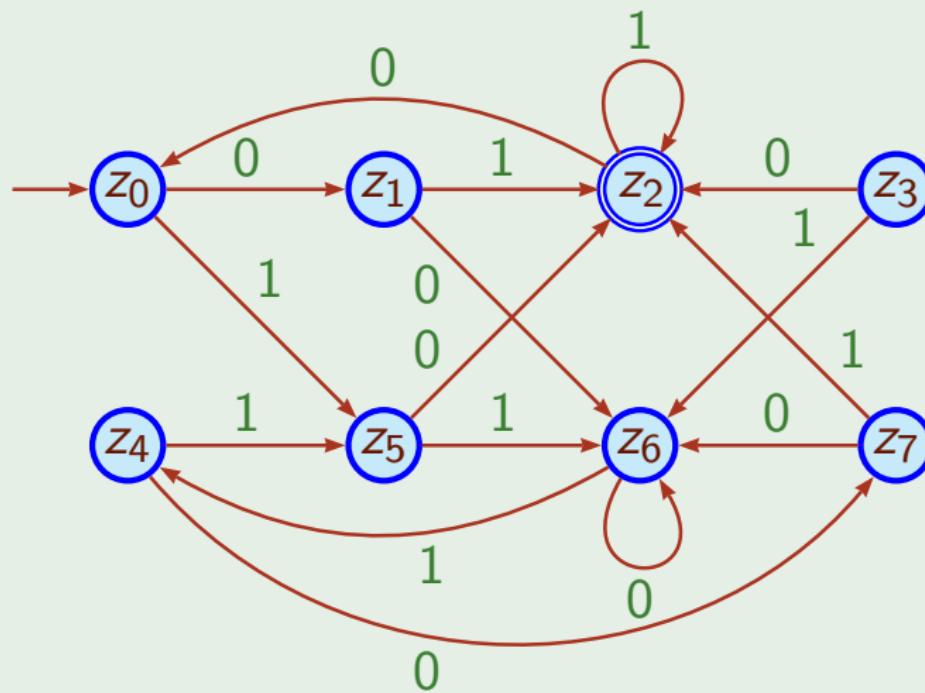


Einige erste Aufgaben X und Lösungen

Einige erste Aufgaben XI

Aufgaben XI

Berechnen Sie bitte den minimierten Automaten von:



Einige erste Aufgaben XI und Lösungen

Einige erste Aufgaben XII

Aufgaben XII

$L = \{ w \mid \text{zwischen zwei } a\text{'s in } w \text{ sind mindestens drei } b\text{'s und } w \text{ hat Länge zwei modulo drei (also die Länge } 2, 5, 8, 11, 14, \dots \text{)} \}$
wobei $\Sigma = \{a, b\}$ und $|w| \bmod 3 = 2$.

Konstruieren Sie bitte einen NEA, der L erkennt.

Kapitel 5 - Epsilon-Endlicher Automat

Version vom 09.09.2015

Automat

Vorlesung 1: Sprachen

Vorlesung 2: Endlicher Automat

Vorlesung 3: Endlicher Automat II

Vorlesung 4: Nicht deterministischer endlicher Automat

Vorlesung 5: Epsilon-Endlicher Automat

Vorlesung 6: Reguläre Ausdrücke

Vorlesung 7: Reguläre Ausdrücke II

Vorlesung 8: Pumping-Lemma

Vorlesung 9: Kontextfreie Grammatiken

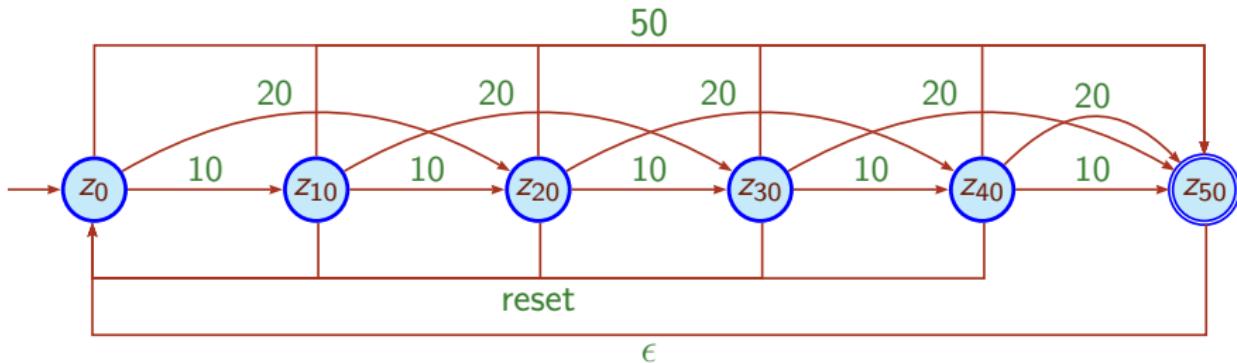
Vorlesung 10: Exoten

Vorlesung 11: Formale Grammatiken

ϵ -Übergänge für automatische Zustandsänderungen

50ct Kaffeeautomat

- ▶ Akzeptiert 10ct, 20ct und 50ct Münzen
- ▶ Gibt kein Geld zurück
- ▶ Mit einer Reset Option
- ▶ Automatische Rücksetzung möglich



Endlicher Automat mit Epsilon-Übergängen I

Eine Erweiterung des endlichen Automaten

Neues Leistungsmerkmal:

Übergänge für die leere Zeichenreihe ϵ sind zugelassen.

O.B.d.A. betrachten wir zuerst

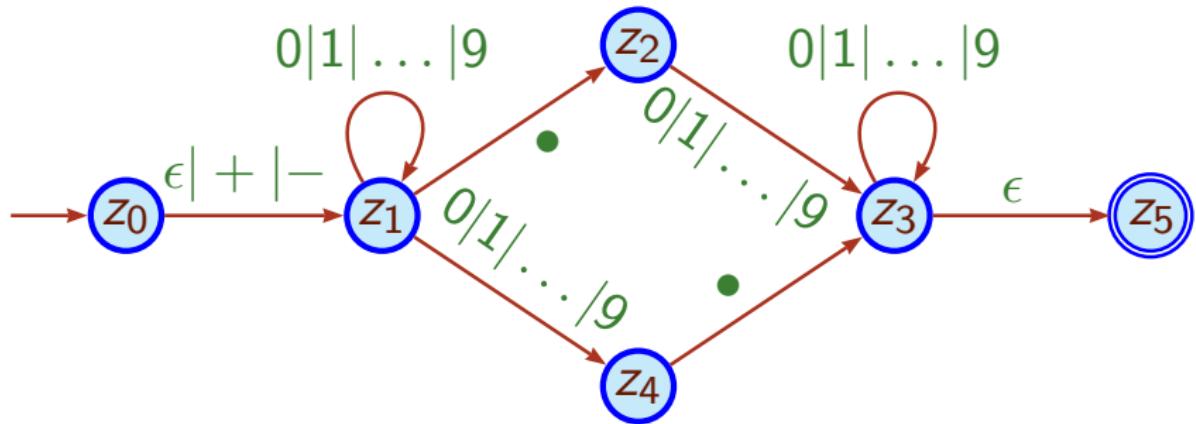
NEA's mit ϵ -Übergängen, kurz ϵ -NEAs.

Auf der nächsten Seite ist ein ϵ -NEA, der Dezimalzahlen akzeptiert, die sich aus folgenden Komponenten zusammensetzen:

- ① ein optionales Plus- (+) oder Minuszeichen (-),
- ② einer Zeichenreihe von Ziffern,
- ③ einen Dezimalpunkt • und
- ④ einer weiteren Zeichenreihe von Ziffern.

Sowohl die letzte Zeichenreihe von Ziffern als auch die Zeichenreihe unter 2 können leer sein, aber **nie beide zusammen**.

Endlicher Automat mit Epsilon-Übergängen II



In z_3 können weitere Ziffern gelesen werden, sofern noch welche folgen, aber der ϵ -NEA kann von z_3 auch **spontan** in den Zustand z_5 übergehen, gewissermaßen in der Vermutung, dass die abschließende Ziffernfolge vollständig gelesen wurde.

ϵ -NEAs

Definition (ϵ -NEA)

Ein **ϵ -NEA** (nichtdeterministischer endlicher Automat mit ϵ -Übergängen) ist ein 5-Tupel $\mathbb{A}_\epsilon = (Z, \Sigma, \delta, z_0, E)$, wobei gilt:

Z ist eine nichtleere endliche Zustandsmenge,

Σ ist ein nichtleeres endliches Eingabealphabet mit $\epsilon \notin \Sigma$,

$\delta : Z \times (\Sigma \cup \{\epsilon\}) \Rightarrow \mathcal{P}(Z)$ ist eine Übergangsfunktion,

$z_0 \in Z$ ist ein Anfangszustand,

E ist eine endliche Menge von Endzuständen, mit $E \subseteq Z$.

Mengen - Potenzmenge, kartesisches Produkt

Definition (Potenzmenge)

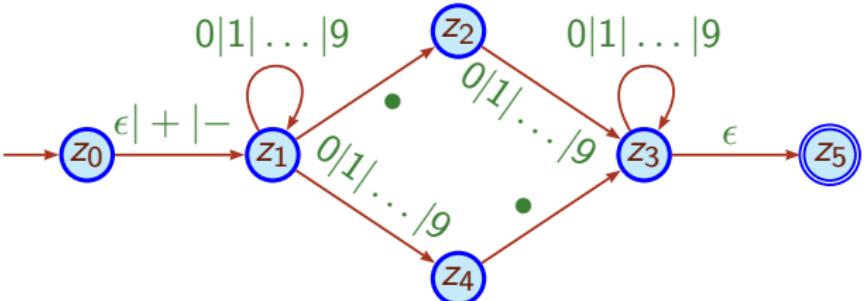
Sei M eine Menge. Die Menge aller Teilmengen von M heißt **Potenzmenge** $\mathcal{P}(M)$ von M .

Definition (kartesisches Produkt)

Seien M_1, \dots, M_n Mengen. Das **kartesische Produkt** $M_1 \times \dots \times M_n$ ist definiert als die Menge aller n -Tupel (m_1, \dots, m_n) , bei denen $m_1 \in M_1, \dots, m_n \in M_n$ gilt

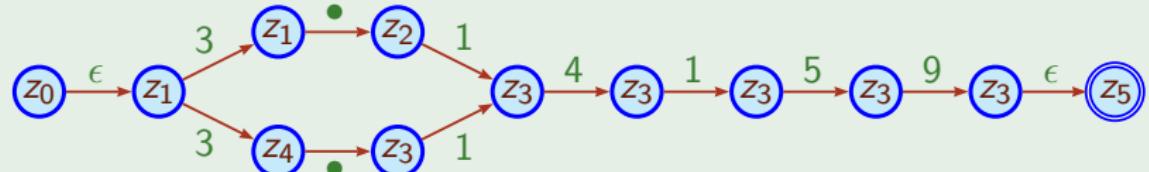
$$\begin{aligned} M_1 \times \dots \times M_n = \\ \{(m_1, \dots, m_n) \mid \text{für jedes } i = 1, \dots, n \text{ ist } m_i \in M_i\} \end{aligned}$$

Arbeitsweise von ϵ -NEAs



- ▶ Die Teilworte $+$, $-$ und ϵ führen nach z_1 ,
- ▶ Teilworte gemäß $\{+, -, \epsilon\} \{0 \dots 9\}^+$ führen nach z_1 oder z_4 ,
- ▶ Teilworte gemäß $\{+, -, \epsilon\} \{0 \dots 9\}^+.$ führen nach z_2 oder z_3 ,
- ▶ Teilworte gemäß $\{+, -, \epsilon\} \{0 \dots 9\}^* \cdot \{0 \dots 9\}^+$ führen nach z_3 ,
- ▶ Wörter die nach z_3 führen, führen auch zum Endzustand z_5 .

Beispiel (Abarbeitung von 3.14159)



Hülle von ϵ -NEAs

Definition (ϵ -Hülle eines Zustands z)

Ein **ϵ -Hülle** ist die von einem Zustand z nur mit ϵ -Übergängen (also ohne Eingaben) erreichbare Menge von Zuständen.

Iterative Definition: Kleinste Menge mit der Eigenschaft
 $z \in \epsilon\text{-Hülle}(z)$ und $p \in \epsilon\text{-Hülle}(z) \wedge r \in \delta(p, \epsilon) \Rightarrow r \in \epsilon\text{-Hülle}(z)$

Erweiterte Überführungsfunktion von ϵ -NEAs

Definition (Erweiterte Überführungsfunktion $\hat{\delta} : Z \times \Sigma^* \Rightarrow \mathcal{P}(Z)$)

Die **Erweiterte Überführungsfunktion** dient dem Aufsammeln aller bei der Abarbeitung erreichbaren Zustände einschließlich derjenigen, die ohne Eingabe erreicht werden.

Induktive Definition (kaskidisches Aufsammeln von Zuständen):

$$\hat{\delta}(z, w) = \begin{cases} \epsilon\text{-Hülle}(z) & \text{falls } w = \epsilon \\ \bigcup_{z' \in \hat{\delta}(z, v)} \bigcup_{z'' \in \delta(z', a)} \epsilon\text{-Hülle}(z'') & \text{falls } w = va \end{cases}$$

mit $a \in \Sigma$

Akzeptierte Sprache von ϵ -NEAs

Definition (Akzeptierte Sprache)

Die **akzeptierte Sprache** ist die Menge der Eingaben w , für die $\hat{\delta}(z_0, w)$ einen Endzustand enthält

$$L(\mathbb{A}_\epsilon) = \left\{ w \in \Sigma^* \mid \hat{\delta}(z_0, w) \cap E \neq \emptyset \right\}$$

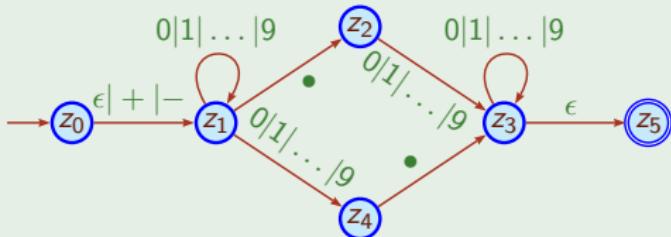
bzw. (für deterministische Automaten)

$$L(\mathbb{A}_D) = \left\{ w \in \Sigma^* \mid \hat{\delta}(z_0, w) \in E \right\}$$

ϵ -Hülle am Beispiel

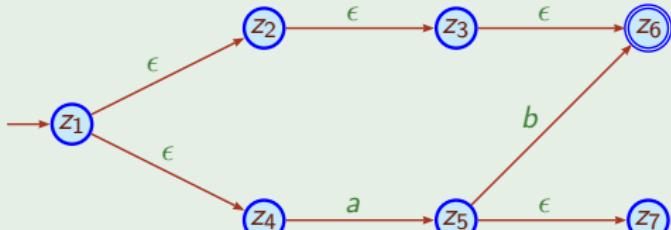
Dezimalautomat

- ▶ Nur 2 ϵ -Übergänge
- ▶ ϵ -Hülle(z_0) = { z_0, z_1 }
- ▶ ϵ -Hülle(z_3) = { z_3, z_5 }
- ▶ ϵ -Hülle(z_i) = { z_i } sonst

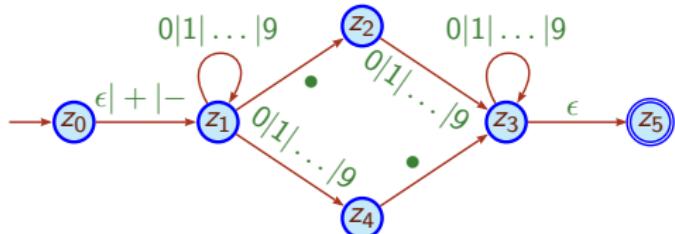


Viele ϵ -Übergänge

- ▶ ϵ -Hülle(z_1) = { z_1, z_2, z_3, z_4, z_6 }
- ▶ ϵ -Hülle(z_2) = { z_2, z_3, z_6 }
- ▶ ϵ -Hülle(z_3) = { z_3, z_6 }
- ▶ ϵ -Hülle(z_4) = { z_4 }
- ▶ ϵ -Hülle(z_5) = { z_5, z_7 }
- ▶ ϵ -Hülle(z_6) = { z_6 }
- ▶ ϵ -Hülle(z_7) = { z_7 }



Überführungsfunktion $\hat{\delta}$ (mit ϵ -Übergängen)



$$\hat{\delta}(z_0, \epsilon) = \text{ε-Hülle}(z_0) = \underline{\underline{\{z_0, z_1\}}}$$

$$\hat{\delta}(z_0, 3) : \delta(z_0, 3) \cup \delta(z_1, 3) = \emptyset \cup \{z_1, z_4\} = \{z_1, z_4\}$$

$$\hat{\delta}(z_0, 3) = \text{ε-Hülle}(z_1) \cup \text{ε-Hülle}(z_4) = \{z_1\} \cup \{z_4\} = \underline{\underline{\{z_1, z_4\}}}$$

$$\hat{\delta}(z_0, 3.) : \delta(z_1, .) \cup \delta(z_4, .) = \{z_2\} \cup \{z_3\} = \{z_2, z_3\}$$

$$\hat{\delta}(z_0, 3.) = \text{ε-Hülle}(z_2) \cup \text{ε-Hülle}(z_3) = \{z_2\} \cup \{z_3, z_5\} = \underline{\underline{\{z_2, z_3, z_5\}}}$$

$$\hat{\delta}(z_0, 3.1) : \delta(z_2, 1) \cup \delta(z_3, 1) \cup \delta(z_5, 1) = \{z_3\} \cup \{z_3\} \cup \emptyset = \{z_3\}$$

$$\hat{\delta}(z_0, 3.1) = \text{ε-Hülle}(z_3) = \underline{\underline{\{z_3, z_5\}}}$$

$$\hat{\delta}(z_0, 3.14) = \text{ε-Hülle}(z_3) = \underline{\underline{\{z_3, z_5\}}}$$

⋮

$$\hat{\delta}(z_0, 3.14159) = \text{ε-Hülle}(z_3) = \underline{\underline{\{z_3, z_5\}}}$$

Umwandlung eines ϵ -NEA in einen DEA

Gut für Charakterisierung optionaler Teilstexte

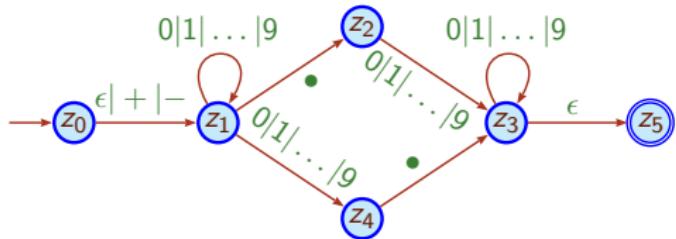
- ▶ Man kann im Zweifel einfach weiterspringen

Sehr ähnliche Teilmengenkonstruktion

- ▶ Sei $\mathbb{A}_\epsilon = (Z_\epsilon, \Sigma, \delta_\epsilon, z_0, E_\epsilon)$ ein ϵ -NEA
- ▶ Konstruiere äquivalenten DEA $\mathbb{A}_D = (Z_D, \Sigma, \delta_D, z_D, E_D)$ mit
 - $Z_D = \mathcal{P}(Z_\epsilon)$
 - $z_D = \epsilon\text{-Hülle}(z_0)$ (statt $\{z_0\}$)
 - $E_D = \{S \mid S \in Z_D \text{ und } S \cap E_\epsilon \neq \emptyset\}$
 - $\delta_D(S, a) = \bigcup_{z \in S} \hat{\delta}_\epsilon(z, a)$ (schließt ϵ -Hülle mit ein)

Optimierung: $Z_D \stackrel{\Delta}{=} \text{erreichbare Zustände}$

- ▶ Iterative Konstruktion gleichzeitig mit δ_D
- ▶ Start: $Z_0 := \{z_D\}$
- ▶ Schritt: $Z_{i+1} := Z_i \cup \{\delta_D(S, a) \mid S \in Z_i, a \in \Sigma\}$ (konstruiere dabei die $\delta_D(S, a)$)
- ▶ Abschluss: Wenn $Z_{i+1} = Z_i$, dann halte an und setze $Z_D := Z_i$

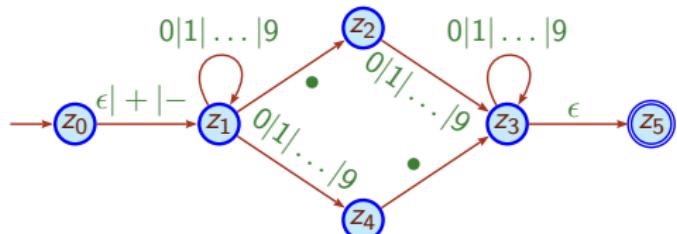
Konstruktion von Z_D und δ_D |

$$Z_D = \{z_D\} = \{\epsilon\text{-H\"ulle}(z_0)\} = \{\{z_0, z_1\}\}$$

- ▶ $\delta_D(\{z_0, z_1\}, +) = \{z_1\}, \quad \delta_D(\{z_0, z_1\}, -) = \{z_1\}$
- ▶ $\delta_D(\{z_0, z_1\}, 0) = \{z_1, z_4\}, \dots, \delta_D(\{z_0, z_1\}, 9) = \{z_1, z_4\}$
- ▶ $\delta_D(\{z_0, z_1\}, .) = \{z_2\}$

$$Z_1 = \{\{z_0, z_1\}, \{z_1\}, \{z_1, z_4\}, \{z_2\}\}$$

- ▶ $\delta_D(\{z_1\}, +) = \delta_D(\{z_2\}, +) = \delta_D(\{z_1, z_4\}, +) = \emptyset \dots$
- ▶ $\delta_D(\{z_1\}, 0) = \delta_D(\{z_1, z_4\}, 0) = \{z_1, z_4\},$
 $\delta_D(\{z_2\}, 0) = \{z_3, z_5\}, \dots$
- ▶ $\delta_D(\{z_1\}, .) = \{z_2\}, \delta_D(\{z_2\}, .) = \emptyset, \delta_D(\{z_1, z_4\}, .) = \{z_2, z_3, z_5\}$

Konstruktion von Z_D und δ_D II

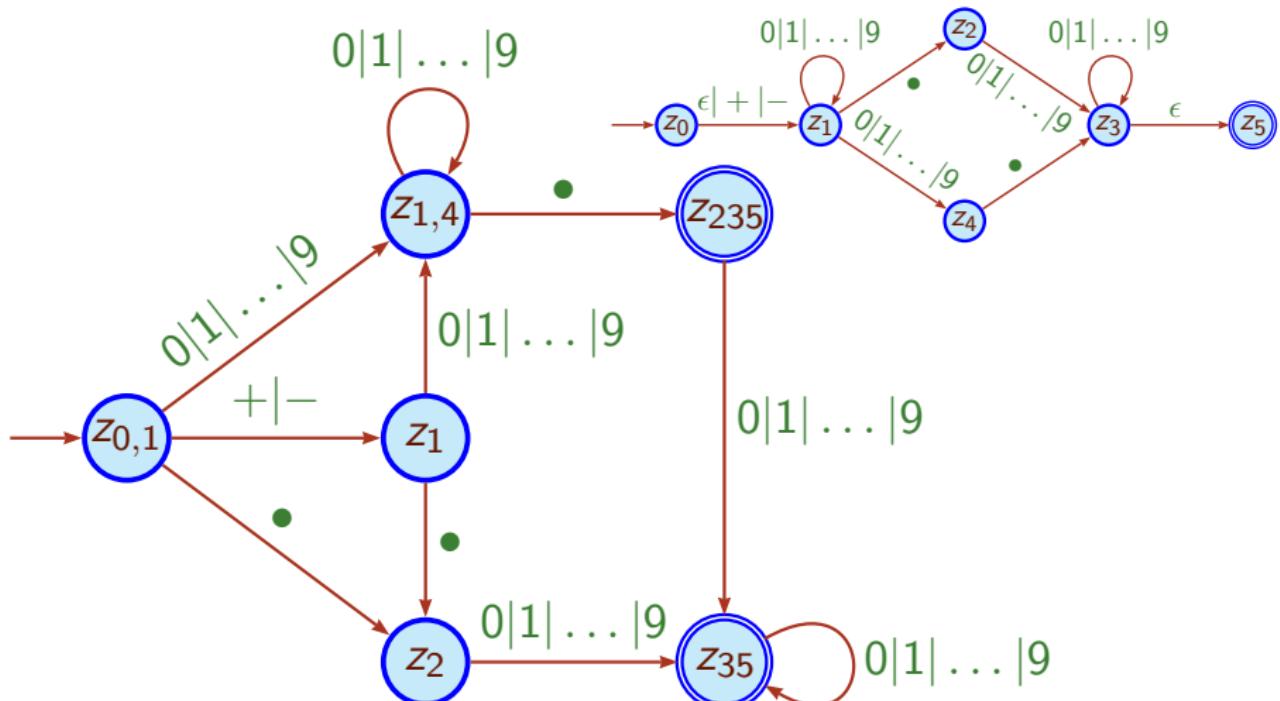
$$Z_2 = \{\{z_0, z_1\}, \{z_1\}, \{z_1, z_4\}, \{z_2\}, \emptyset, \{z_3, z_5\}, \{z_2, z_3, z_5\}\}$$

- $\delta_D(\{\emptyset\}, +) = \delta_D(\{z_2, z_3, z_5\}, +) = \delta_D(\{z_3, z_5\}, +) = \emptyset, \dots$
- $\delta_D(\{\emptyset\}, 0) = \emptyset,$
 $\delta_D(\{z_2, z_3, z_5\}, 0) = \delta_D(\{z_3, z_5\}, 0) = \{z_3, z_5\}, \dots$
- $\delta_D(\{\emptyset\}, .) = \delta_D(\{z_2, z_3, z_5\}, .) = \delta_D(\{z_3, z_5\}, .) = \emptyset$

$$Z_3 = \{\{z_0, z_1\}, \{z_1\}, \{z_1, z_4\}, \{z_2\}, \emptyset, \{z_3, z_5\}, \{z_2, z_3, z_5\}\} = Z_2$$

Also folgt:

$$Z_D = \{\{z_0, z_1\}, \{z_1\}, \{z_1, z_4\}, \{z_2\}, \emptyset, \{z_3, z_5\}, \{z_2, z_3, z_5\}\}$$

Konstruktion von Z_D und δ_D III

Fertiger DEA Übergänge zum Zustand \emptyset ($= z_F$) werden **nicht** gezeigt.

Für den konstruierten DEA gilt $L(\mathbb{A}_D) = L(\mathbb{A}_\epsilon)$

- ▶ Zu zeigen: $\hat{\delta}_D(z_D, w) = \hat{\delta}_\epsilon(z_0, w) \quad \forall w \in \Sigma^*$
- ▶ Beweis schreit nach vollständiger Induktion, da die Sprache des Automaten auch so aufgebaut ist:
es gibt ggf. Wörter mit Länge=0, Länge=1, Länge=2 usw..
- ▶ Bei einem ϵ -NEA geht es **immer** bei 0 los, also für ein Wort welches gleich ϵ ist.
- ▶ Wir benutzen, dass für jedes weitere Wort **immer** gilt:
 $w = va$, wobei v ein Wort ist, dessen Länge 1 kürzer ist als w .

Für den konstruierten DEA gilt $L(\mathbb{A}_D) = L(\mathbb{A}_\epsilon)$

$$\hat{\delta}_D(z_D, w) = \hat{\delta}_\epsilon(z_0, w) \quad \forall w \in \Sigma^*$$

Beweis:

Induktionsanfang Sei $w = \epsilon$:

$$\hat{\delta}_D(z_D, \epsilon) = z_D = \epsilon\text{-H\"ulle}(z_0) = \hat{\delta}_\epsilon(z_0, \epsilon)$$

Induktionsannahme Es sei $\hat{\delta}_D(z_D, v) = \hat{\delta}_\epsilon(z_0, v)$ mit $|v| = k$

Induktionsschluss Es gilt für $n = k + 1$ mit $w = va$ und $a \in \Sigma$

$$\begin{aligned}\hat{\delta}_D(z_D, w) &= \delta_D(\hat{\delta}_D(z_D, v), a) \\ &= \delta_D(\hat{\delta}_\epsilon(z_0, v), a) \\ &= \bigcup_{z' \in \hat{\delta}_\epsilon(z_0, v)} \hat{\delta}_\epsilon(z', a) \\ &= \bigcup_{z' \in \hat{\delta}_\epsilon(z_0, v)} \bigcup_{z'' \in \delta_\epsilon(z', a)} \epsilon\text{-H\"ulle}(z'') \\ &= \hat{\delta}_\epsilon(z_0, w)\end{aligned}$$

Satz von Rabin und Scott

Tatsächlich haben wir hier sogar etwas bewiesen, was als bekannter Satz formuliert wurde:

Satz: Rabin und Scott

Jede von einem NEA akzeptierte Sprache ist auch von einem DEA erkennbar. (Oder: für jeden NEA gibt es einen DEA, der die gleiche Sprache erkennt.)

ϵ -NEAs und DEAs akzeptieren dieselbe Sprachen

Einige erste Aufgaben XIII

Aufgaben XIII

Wandeln Sie bitte diesen NEA in einen DEA um:

Zustand	Eingabe	
	0	1
$\rightarrow p$	$\{p, q\}$	$\{p\}$
q	$\{r\}$	$\{r\}$
r	$\{s\}$	\emptyset
$* s$	$\{s\}$	$\{s\}$

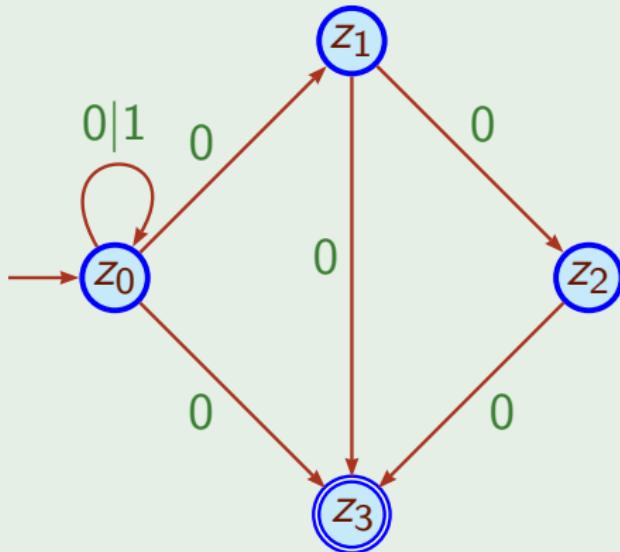
- 1 Benutzen Sie für die neuen Zustände die Abkürzungen z_0 (Anfangszustand) bis z_n (Tipp: $n < 10$).
- 2 Überprüfen Sie ob dieser Automat minimal ist und bestimmen ggf. den minimalen Automaten.

Einige erste Aufgaben XIII und Lösungen

Einige erste Aufgaben XIV

Aufgaben XIV

Gegeben seien die Ein-gabemenge $\Sigma = \{0, 1\}$ und der folgende Zu-standsübergangsgraph für den NEA:



- ➊ Konstruieren Sie bitte den DEA, der dieselbe Sprache erkennt.
- ➋ Überprüfen Sie ob dieser Automat minimal ist und bestimmen ggf. den minimalen Automaten.

Einige erste Aufgaben XIV und Lösungen

Einige erste Aufgaben XV

Aufgaben XV

Wandeln Sie bitte diesen ϵ -NEA in einen DEA um:

Zustand	Eingabe		
	0	1	ϵ
$\rightarrow p$	$\{t\}$	\emptyset	\emptyset
q	$\{s\}$	$\{s\}$	$\{r\}$
r	$\{q, r, s\}$	$\{r\}$	\emptyset
$* s$	\emptyset	\emptyset	\emptyset
$* t$	\emptyset	\emptyset	$\{r\}$

- 1 Benutzen Sie für die neuen Zustände die Abkürzungen z_0 (Anfangszustand) bis z_n (Tipp: $n < 10$) oder auch rt statt $\{r, t\}$.
- 2 Überprüfen Sie ob dieser Automat minimal ist und bestimmen ggf. den minimalen Automaten.

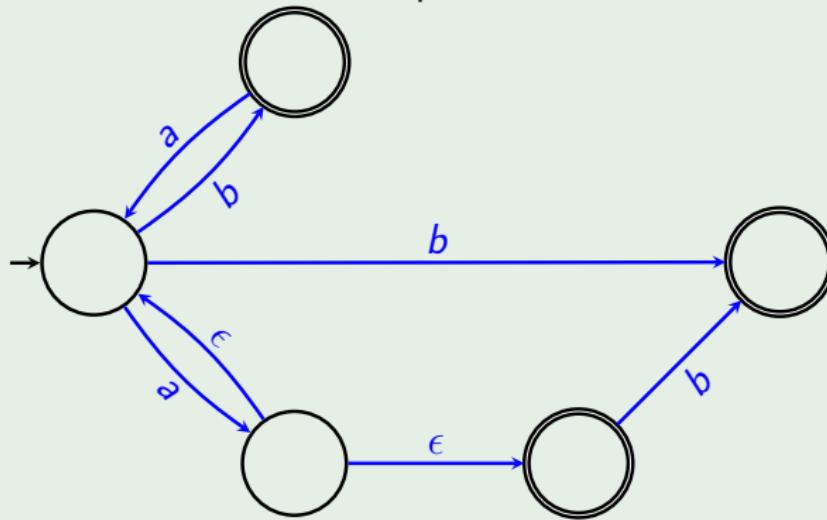
Einige erste Aufgaben XV und Lösungen

Einige erste Aufgaben XV und Lösungen

Einige erste Aufgaben XVI

Aufgaben XVI

Konstruieren Sie einen äquivalenten DEA ohne Epsilon-Übergänge



WHENEVER I LEARN A
NEW SKILL I CONCOCT
ELABORATE FANTASY
SCENARIOS WHERE IT
LETS ME SAVE THE DAY.

OH NO! THE KILLER
MUST HAVE FOLLOWED
HER ON VACATION!

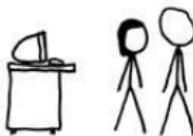


BUT TO FIND THEM WE'D HAVE TO SEARCH
THROUGH 200 MB OF EMAILS LOOKING FOR
SOMETHING FORMATTED LIKE AN ADDRESS!



IT'S HOPELESS!

EVERYBODY STAND BACK.



I KNOW REGULAR
EXPRESSIONS.



Kapitel 6 - Reguläre Ausdrücke

Version vom 20.11.2014

Sprachen

Vorlesung 1: Sprachen

Vorlesung 2: Endlicher Automat

Vorlesung 3: Endlicher Automat II

Vorlesung 4: Nicht deterministischer endlicher Automat

Vorlesung 5: Epsilon-Endlicher Automat

Vorlesung 6: Reguläre Ausdrücke

Vorlesung 7: Reguläre Ausdrücke II

Vorlesung 8: Pumping-Lemma

Vorlesung 9: Kontextfreie Grammatiken

Vorlesung 10: Exoten

Vorlesung 11: Formale Grammatiken

Nachtrag: Was kann man mit Wörtern alles machen?

- ▶ verketten und immer wieder verketten
(Ge)Bettuch vs. Bettuch
- ▶ die Länge berechnen
 $w = \text{Bettuch} \Rightarrow |w| = 8$
- ▶ umdrehen

Sarg - Gras¹²

¹Ein Palindrom (von griechisch Παλινδρομός palindromos „rückwärts laufend“) ist eine Zeichenkette, die von vorn und von hinten gelesen gleich bleibt.

²Ein Palindrom in der Informatik ist ein Wort w über dem Alphabet Σ mit der Eigenschaft w^R , wobei w^R bedeutet, dass der Operator R der Spiegelung (bzw. Umkehrung der Reihenfolge der Zeichen) auf das Wort w angewandt wird. Zu beachten ist, dass ein Palindrom hier nicht unbedingt einen Sinn ergeben muss; das entsprechende Wort muss lediglich symmetrisch um seine Mitte aufgebaut sein.

Nachtrag: Verkettung von Wörtern I

- ▶ $v \circ w = vw$ ist die Verkettung der Wörter v und w
z.B. $1100 \circ 0101 = 11000101$
- ▶ $\epsilon \circ w = w \circ \epsilon$
das leere Wort ist neutral bzgl. der Verkettung
- ▶ v^n ist die n-malige Verkettung von v :
 $v^0 = \epsilon$ und $v^{n+1} = v^n v$ für $n \geq 0$ z.B. $(10)^3 = 101010$

Nachtrag: Verkettung von Wörtern II

Σ^i die Menge aller Wörter über Σ mit der Länge i
 (auch die i te Potenz)

Beispiel:

① $\Sigma^0 = \{\epsilon\}$, $\Sigma^1 = \Sigma$ oder $\Sigma^2 = \{w \mid |w| = 2\}$

② für z.B. $\Sigma = \{0, 1\}$

dann $\Sigma^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$

$\Sigma^{\leq n} = \bigcup_{i=0}^n \Sigma^i$ die Menge aller Wörter über Σ

mit höchstens n Symbolen

$\Sigma^* = \bigcup_{i \in \mathbb{N}_0} \Sigma^i$ die Menge aller Wörter über Σ

$\Sigma^+ = \bigcup_{i \in \mathbb{N}} \Sigma^i$ die Menge aller nicht-leeren Wörter über Σ

Reguläre Ausdrücke I

„Logik ist der Anfang aller Weisheit, Lieutenant Valeris, nicht das Ende.“

Spock in Star Trek VI: Das unentdeckte Land

Definition (Reguläre Ausdrücke)

Sei Σ ein Alphabet, dann gilt:

- ① \emptyset ist ein regulärer Ausdruck über Σ .
- ② ϵ ist ein regulärer Ausdruck über Σ .
- ③ Für jedes $a \in \Sigma$ ist a ein regulärer Ausdruck über Σ .
- ④ Wenn E und F reguläre Ausdrücke über Σ sind, so sind auch
 - $E \circ F = EF$ (Verkettung),
 - $(E|F)$ (Alternative/Vereinigung) und
 - $(E)^*$ (Kleenesche Hülle)reguläre Ausdrücke über Σ .

Reguläre Ausdrücke II

Definition (Sprache eines regulären Ausdrucks)

Sei Σ ein Alphabet und R ein regulärer Ausdruck über Σ , dann wird die von R beschriebene Sprache $L(R) \subseteq \Sigma^*$ wie folgt definiert

- ① Für $R = \emptyset$ gilt $L(R) = \emptyset$.
- ② Für $R = \epsilon$ gilt $L(R) = \{\epsilon\}$.
- ③ Für $R = a$ mit $a \in \Sigma$ gilt $L(R) = \{a\}$.
- ④ Für $R = EF$ gilt $L(R) = L(E) \circ L(F)$.
- ⑤ Für $R = (E|F)$ gilt $L(R) = L(E) \cup L(F)$.
- ⑥ Für $R = (E)^*$ gilt $L(R) = (L(E))^*$.

Äquivalenz regulärer Ausdrücke

Definition (Äquivalenz regulärer Ausdrücke)

Zwei reguläre Ausdrücke E und F heißen **äquivalent**, in Zeichen $E \equiv F$, wenn $L(E) = L(F)$ gilt.

Beispiele: Äquivalenz

$$((a|b))^* \equiv ((a|b)(a|b))^* ((a|b)|\epsilon)$$

$$E(FE)^* \equiv (EF)^* E$$

Reguläre Ausdrücke - Rechengesetze I

Satz

Seien A , B , C reguläre Ausdrücke. Für die Operationen $*$, \circ , $|$ gelten folgende Rechengesetze

Kommutativgesetz:

$$A \circ B \equiv B \circ A \quad (A|B) \equiv (B|A)$$

Assoziativgesetz:

$$(A \circ (B \circ C)) \equiv ((A \circ B) \circ C) \quad (A|(B|C)) \equiv ((A|B)|C)$$

Distributivgesetz:

$$A \circ (B|C) \equiv (A \circ B|A \circ C) \quad A|(B \circ C) \equiv (A|B \circ A|C)$$

$$(B|C) \circ A \equiv (B \circ A|C \circ A) \quad (B \circ C)|A \equiv (B|A \circ C|A)$$

Reguläre Ausdrücke - Rechengesetze II

Satz

neutrales Element:

$$A \circ \{\epsilon\} \equiv \{\epsilon\} \circ A \equiv A \quad (A|\emptyset) \equiv A$$

$$(A|A) \equiv A \quad \text{Idempotenzgesetz}$$

$$A \circ \emptyset \equiv \emptyset \circ A \equiv \emptyset \quad \text{Nulloperator der Verkettung}$$

$$((A)^*)^* \equiv (A)^*$$

$$(\emptyset)^* \equiv \{\epsilon\} \quad (\{\epsilon\})^* \equiv \{\epsilon\}$$

$$((A| \{\epsilon\}))^* \equiv (A)^*$$

$$(A)^* \equiv (\{\epsilon\} | A) \circ (A)^* \equiv ((\{\epsilon\} | A))^* \circ (A)$$

Reguläre Ausdrücke - Vereinbarungen

- ▶ Wir vereinbaren, dass wir Klammern, die nicht notwendigerweise gebraucht werden, weglassen können. Zum Beispiel können wir statt $(A|(B|C))$ auch $(A|B|C)$ schreiben. Wir schreiben auch $L(A|B)$ statt $L((A|B))$ sowie a^* statt $(a)^*$ für alle $a \in \Sigma$.
- ▶ In der Literatur findet man oft auch abweichende Schreibweisen der regulären Ausdrücke. Zum Beispiel findet man für $(A|B)$ auch $(A + B)$ oder auch $(A \cup B)$ (siehe unser Buch). Auch wird oft statt $A \circ B$ auch $A \cdot B$ bzw. AB zugelassen.
- ▶ Wir benutzen die abkürzende Schreibweise R^n für $\underbrace{RR\dots R}_{n-\text{mal}}$.
- ▶ Wir benutzen die abkürzende Schreibweise R^+ für R^*R .
- ▶ Oft wird nicht zwischen regulärem Ausdruck und beschriebener Sprache unterschieden, das heißt, man identifiziert einen regulären Ausdruck mit der beschriebenen Sprache.

Reguläre Ausdrücke - Beispiele

$(a|b)^*$ beschreibt die Menge **aller** Wörter über dem Alphabet $\{a, b\}$.

$(a|b)^+$ beschreibt die Menge **aller** Wörter über dem Alphabet $\{a, b\}$, die **nicht dem leeren Wort entsprechen**.

$(a|b)^*abba(a|b)^*$ beschreibt die Menge **aller** Wörter über dem Alphabet $\{a, b\}$, die das Teilwort **abba** enthalten.

$(a|b)^*a(a|b)^2$ beschreibt die Menge **aller** Wörter über dem Alphabet $\{a, b\}$, deren **drittletztes Symbol ein a ist**.

$((a|b)(a|b))^*$ beschreibt die Menge **aller** Wörter über dem Alphabet $\{a, b\}$, deren **Länge gerade** ist.

$(b|\epsilon)(ab)^*(a|\epsilon)$ beschreibt die Menge **aller** Wörter über dem Alphabet $\{a, b\}$, die **nicht das Teilwort aa und nicht das Teilwort bb enthalten**.

Reguläre Ausdrücke - Rechengesetze Ia

Satz

Seien A , B , C reguläre Ausdrücke. Für die Operationen $*$, \cdot , $+$ gelten folgende Rechengesetze

Kommutativgesetz:

$$AB = BA \quad A + B = B + A$$

Assoziativgesetz:

$$A(BC) = (AB)C \quad A + (B + C) = (A + B) + C$$

Distributivgesetz:

$$A(B + C) = AB + AC \quad A + (BC) = (A + B)(A + C)$$

$$(B + C)A = (BA) + (CA) \quad (BC) + A = (B + A)(C + A)$$

Reguläre Ausdrücke - Rechengesetze IIa

Satz

neutrales Element:

$$A\epsilon = \epsilon A = A \quad A + \emptyset = A$$

$$A + A = A \text{ *Idempotenzgesetz*} \quad A^* + A = A^*$$

$A\emptyset = \emptyset A = \emptyset$ Nulloperator der Verkettung

$$(A^*)^* = A^* \quad AA^* = A^*A$$

$$\emptyset^* = \epsilon \quad \epsilon^* = \epsilon$$

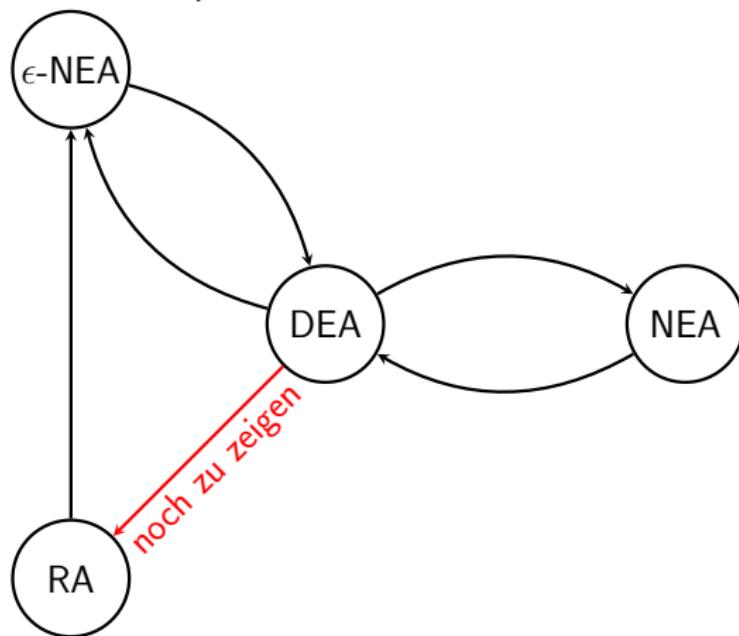
$$(A + \epsilon)^* = A^* \quad \epsilon + AA^* = A^*$$

$$A^* = (\epsilon + A)A^* = (\epsilon + A)^*A$$

Reguläre Ausdrücke und endliche Automaten

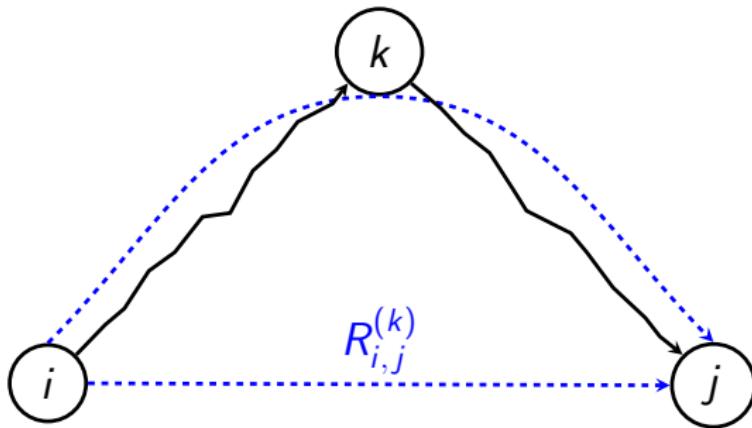
- ▶ EAs und RAs beschreiben dieselben Sprachen:
Reguläre Sprachen

- ▶ Beweisplan für die Äquivalenz:

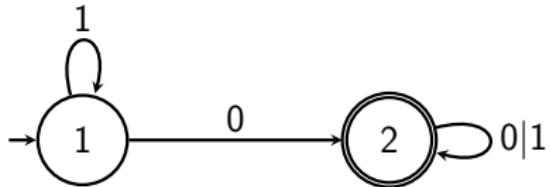


DEA \Rightarrow RA: k -Pfad

- ▶ **Idee:** Pfade in einem DEA auf Basis der Kantenbeschriftung in RA überführen
- ▶ Dazu Zustände des DEA in $1, 2, \dots, n$ umbenennen
- ▶ Der k -Pfad $R_{i,j}^{(k)}$ bezeichnet den RA, dessen Sprache genau die Wörter umfasst, die den Beschriftungen der Pfade von Zustand i nach Zustand j entsprechen, wobei kein Zustand mit einem Namen $> k$ durchlaufen werden darf.



k -Pfad



$R_{i,j}^{(0)}$ für direkten Weg von i zu j
(also ohne Knoten)

k -Pfad	reg. Ausdruck
$R_{1,1}^{(0)}$	$\epsilon + 1$
$R_{1,2}^{(0)}$	0
$R_{2,1}^{(0)}$	\emptyset
$R_{2,2}^{(0)}$	$\epsilon + 0 + 1$

$R_{i,j}^{(1)}$ für den Weg über maximal 1

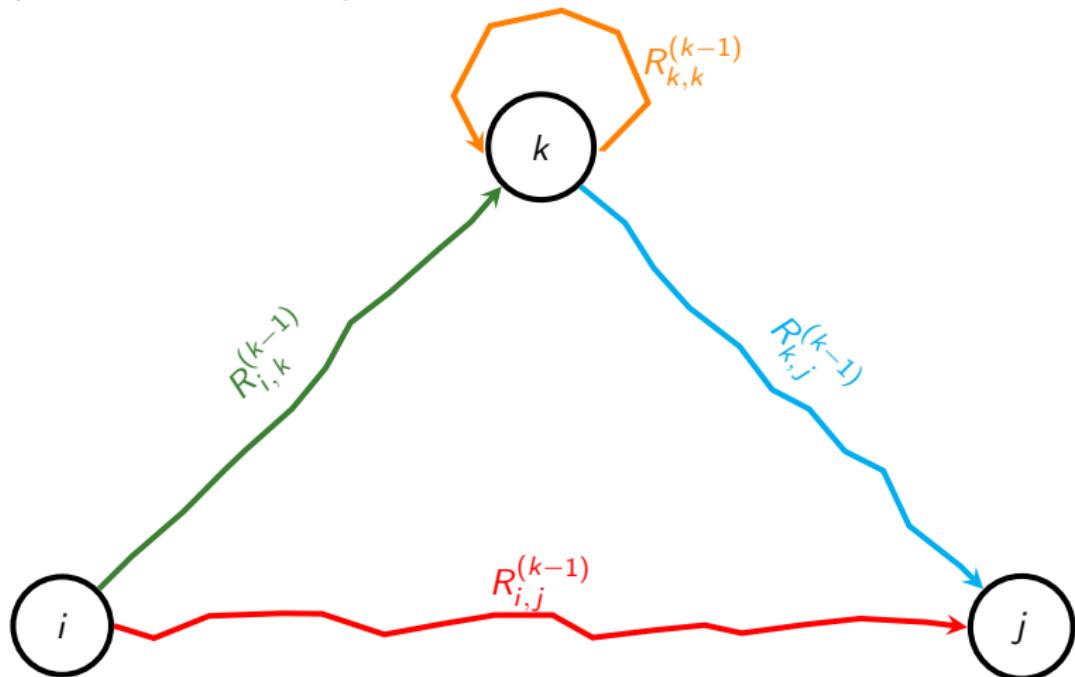
k -Pfad	reg. Ausdruck
$R_{1,1}^{(1)}$	1^*
$R_{1,2}^{(1)}$	1^*0
$R_{2,1}^{(1)}$	\emptyset
$R_{2,2}^{(1)}$	$\epsilon + 0 + 1$

$R_{i,j}^{(2)}$ für den Weg über 2

k -Pfad	reg. Ausdruck
$R_{1,1}^{(2)}$	1^*
$R_{1,2}^{(2)}$	$1^*0(0 + 1)^*$
$R_{2,1}^{(2)}$	\emptyset
$R_{2,2}^{(2)}$	$(0 + 1)^*$

Induktive Berechnung der k -Pfade

$R_{i,j}^{(k)}$ lässt sich aus $R_{i,j}^{(k-1)}$ berechnen



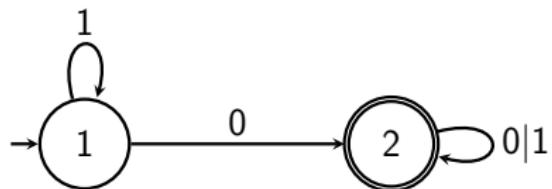
$$R_{i,j}^{(k)} = R_{i,j}^{(k-1)} + R_{i,k}^{(k-1)} \left(R_{k,k}^{(k-1)} \right)^* R_{k,j}^{(k-1)}$$

DEA \Rightarrow RA: Induktiv

- ▶ Zustände des DEA werden in $1, 2, \dots, n$ umbenannt
- ▶ $R_{i,j}^{(k)}$ bezeichnet den RA, dessen Sprache genau die Zeichenreihen umfasst, die den Beschriftungen der Pfade von Zustand i nach Zustand j entsprechen, wobei kein Zustand mit einem Namen $> k$ durchlaufen werden darf.
- ▶ $R_{i,j}^{(k)}$ lässt sich aus $R_{i,j}^{(k-1)}$ berechnen

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

- ▶ RA ist die Summe aller $R_{1,e}^{(k)}$ mit 1 dem Startzustand und $e \in E$ einem Endzustand

k-Pfad

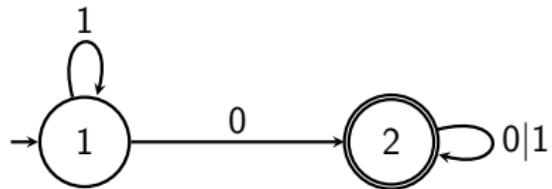
<i>k</i> -Pfad	reg. Ausdruck
$R_{1,1}^{(0)}$	$\epsilon + 1$
$R_{1,2}^{(0)}$	0
$R_{2,1}^{(0)}$	\emptyset
$R_{2,2}^{(0)}$	$\epsilon + 0 + 1$

Reguläre Ausdrücke $R_{i,j}^{(1)}$ für den Weg über höchstens 1 von i zu j :

$$R_{i,j}^{(1)} = R_{i,j}^{(0)} + R_{i,1}^{(0)} \left(R_{1,1}^{(0)} \right)^* R_{1,j}^{(0)}$$

<i>k</i> -Pfad	ind. Pfadausdruck	reg. Ausdruck
$R_{1,1}^{(1)}$	$\epsilon + 1 + (\epsilon + 1)(\epsilon + 1)^*(\epsilon + 1)$	1^*
$R_{1,2}^{(1)}$	$0 + (\epsilon + 1)(\epsilon + 1)^*0$	1^*0
$R_{2,1}^{(1)}$	$\emptyset + \emptyset(\epsilon + 1)^*(\epsilon + 1)$	\emptyset
$R_{2,2}^{(1)}$	$\epsilon + 0 + 1 + \emptyset(\epsilon + 1)^*0$	$\epsilon + 0 + 1$

k -Pfad



k -Pfad	reg. Ausdruck
$R_{1,1}^{(1)}$	1^*
$R_{1,2}^{(1)}$	1^*0
$R_{2,1}^{(1)}$	\emptyset
$R_{2,2}^{(1)}$	$\epsilon + 0 + 1$

Reguläre Ausdrücke $R_{i,j}^{(2)}$ für den Weg über höchstens 2 von i zu j :

$$R_{i,j}^{(2)} = R_{i,j}^{(1)} + R_{i,2}^{(1)} \left(R_{2,2}^{(1)} \right)^* R_{2,j}^{(1)}$$

k -Pfad	ind. Pfadausdruck	reg. Ausdruck
$R_{1,1}^{(2)}$	$1^* + 1^*0(\epsilon + 0 + 1)^*\emptyset$	1^*
$R_{1,2}^{(2)}$	$1^*0 + 1^*0(\epsilon + 0 + 1)^*(\epsilon + 0 + 1)$	$1^*0(0 + 1)^*$
$R_{2,1}^{(2)}$	$\emptyset + (\epsilon + 0 + 1)(\epsilon + 0 + 1)^*\emptyset$	\emptyset
$R_{2,2}^{(2)}$	$(\epsilon + 0 + 1) + (\epsilon + 0 + 1)(\epsilon + 0 + 1)^*(\epsilon + 0 + 1)$	$(0 + 1)^*$

Bewertung der Konstruktion von RA

- ▶ Ist immer anwendbar
- ▶ Funktioniert auch für NEA und ϵ -NEA
- ▶ Aufwändig
Nach n Induktionsschritten können Ausdrücke eine Länge von 4^n Symbolen erreichen
- ▶ Andere Methode: Eliminierung von Zuständen

Idee:

Zustände sukzessive eliminieren
und im Gegenzug Beschriftungen durch entsprechend
komplexere RAs ersetzen

Einige erste Aufgaben XVII

Aufgaben XVII

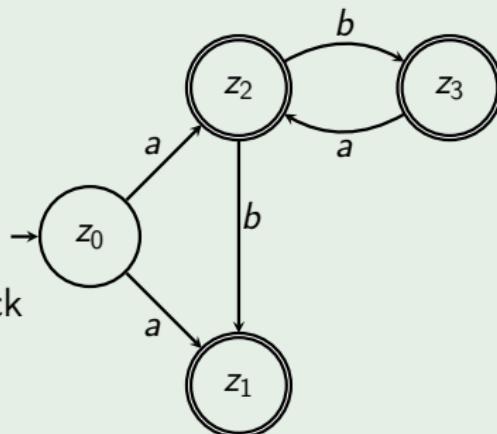
Gegeben sei folgender nichtdeterministischer endlicher Automat:

- 1 Konstruieren Sie einen äquivalenten deterministischen endlichen Automaten (DEA).

- 2 Bilden Sie den minimalen deterministischen endlichen Automaten, den minDEA.

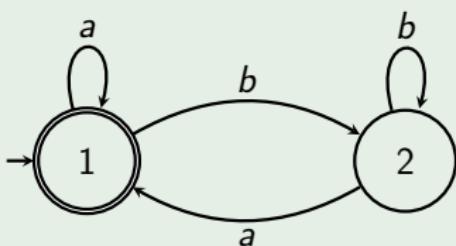
- 3 Geben Sie den regulären Ausdruck an, der die von dem Automaten akzeptierte Sprache beschreibt.

- 4 Geben Sie eine rechtslineare Grammatik an, die dieselbe Sprache generiert, die der Automat akzeptiert.



Aufgaben XVIII

Gegeben sei folgender deterministischer endlicher Automat:



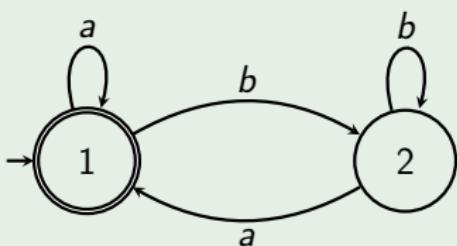
k-Pfad	reg. Ausdruck
$R_{1,1}^{(0)}$	$(\epsilon +)a$
$R_{1,2}^{(0)}$	b
$R_{2,1}^{(0)}$	a
$R_{2,2}^{(0)}$	$(\epsilon +)b$

$$R_{i,j}^{(1)} = R_{i,j}^{(0)} + R_{i,1}^{(0)} \left(R_{1,1}^{(0)} \right)^* R_{1,j}^{(0)}$$

k-Pfad	ind. Pfadausdruck	reg. Ausdruck
$R_{1,1}^{(1)}$	$a + a(a)^*a$	aa^*
$R_{1,2}^{(1)}$	$b + a(a)^*b$	a^*b
$R_{2,1}^{(1)}$	$a + a(a)^*a$	aa^*
$R_{2,2}^{(1)}$	$b + a(a)^*b$	a^*b

Aufgaben XVIII

Gegeben sei folgender deterministischer endlicher Automat:



k -Pfad	reg. Ausdruck
$R_{1,1}^{(1)}$	aa^*
$R_{1,2}^{(1)}$	a^*b
$R_{2,1}^{(1)}$	aa^*
$R_{2,2}^{(1)}$	a^*b

$$R_{i,j}^{(2)} = R_{i,j}^{(1)} + R_{i,2}^{(1)} \left(R_{2,2}^{(1)} \right)^* R_{2,j}^{(1)}$$

k -Pfad	ind. Pfadausdruck	reg. Ausdruck
$R_{1,1}^{(2)}$	$aa^* + a^*b(a^*b)^*aa^*$???
$R_{1,2}^{(2)}$	$aa^* + a^*b(a^*b)^*aa^*$????
$R_{2,1}^{(2)}$	$aa^* + a^*b(a^*b)^*aa^*$???
$R_{2,2}^{(2)}$	$aa^* + a^*b(a^*b)^*aa^*$????

Kapitel 7 - Reguläre Ausdrücke II

Version vom 16.07.2015

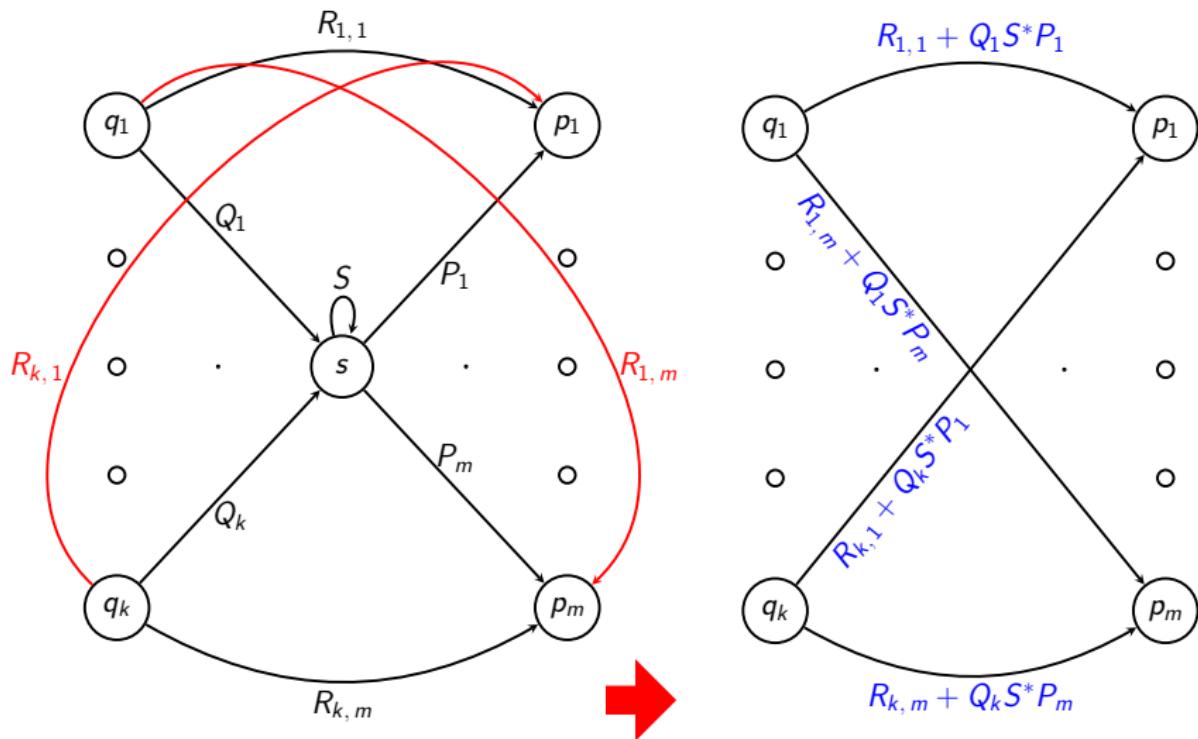
Sprachen

- Vorlesung 1: Sprachen
- Vorlesung 2: Endlicher Automat
- Vorlesung 3: Endlicher Automat II
- Vorlesung 4: Nicht deterministischer endlicher Automat
- Vorlesung 5: Epsilon-Endlicher Automat
- Vorlesung 6: Reguläre Ausdrücke
- Vorlesung 7: Reguläre Ausdrücke II**
- Vorlesung 8: Pumping-Lemma
- Vorlesung 9: Kontextfreie Grammatiken
- Vorlesung 10: Exoten
- Vorlesung 11: Formale Grammatiken

Bewertung der Konstruktion von RA

- ▶ Ist immer anwendbar.
- ▶ Funktioniert auch für NEA und ϵ -NEA.
- ▶ Aufwändig
Nach n Induktionsschritten können Ausdrücke eine Länge von 4^n Symbolen erreichen.
- ▶ Andere Methode: Eliminierung von Zuständen
Idee:
Zustände sukzessive eliminieren
und im Gegenzug Beschriftungen durch entsprechend komplexe RAs ersetzen.

Eliminierung von Zuständen: Generischer Ansatz



Vorgehensweise bei der Eliminierung

- ▶ **Ziel:** Automaten, die nur noch aus zwei Zuständen bestehen:
Startzustand und Zielzustand
- ▶ **Problem:** Und wenn es im Ursprungsautomaten mehrere
Zielzustände gibt?
- ▶ **Lösung:** Für jeden Zielzustand einen „Zweizustandsautomaten“
erzeugen, der nur diesen einen Zielzustand berücksichtigt, und
die resultierenden RAs alternativ zulassen!
- ▶ **Neues Problem:** Wenn Startzustand akzeptierend ist?
- ▶ **Lösung:** Reduktion bis auf einen Zustand!

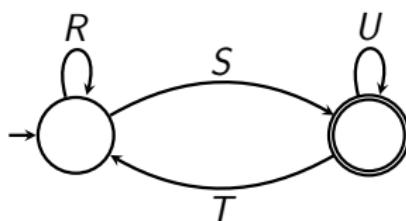
Generische Resultate der Zustandseliminierung

- ▶ **Ein Zustand:** falls
Startzustand zugleich
akzeptierend
Resultierender regulärer
Ausdruck: R^*



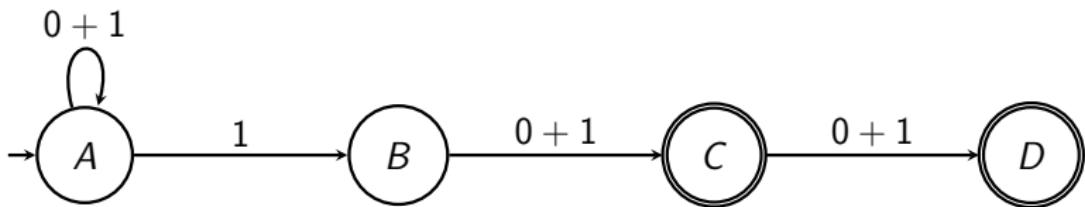
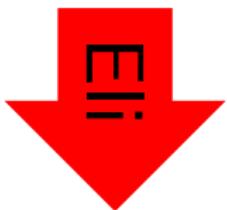
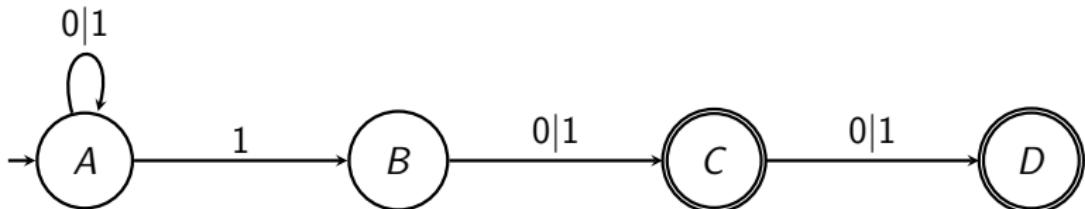
ODER

- ▶ **Zwei Zustände:** sonst
Resultierender regulärer
Ausdruck: $(R + SU^* T)^* S U^*$



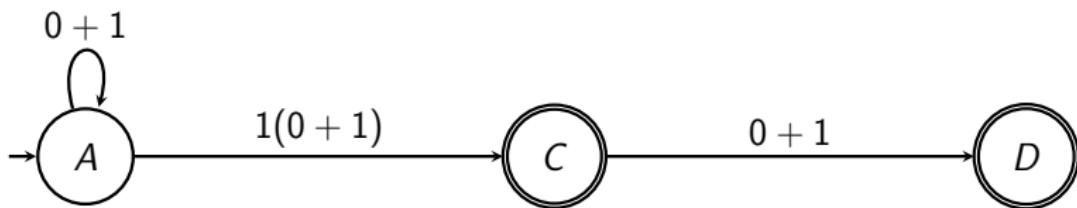
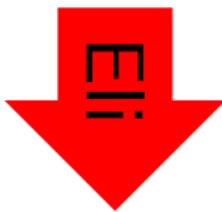
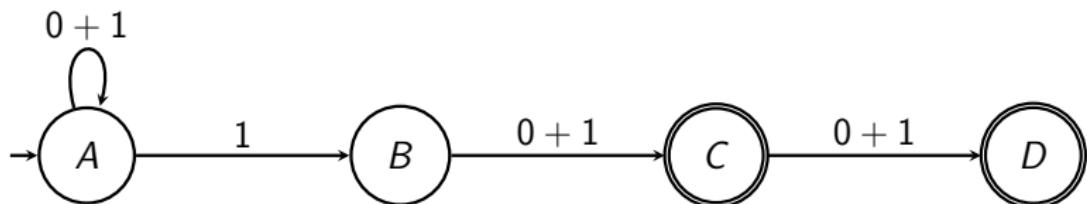
Zustandseliminierung RA I

Beschriftung als reguläre Ausdrücke ($| \Rightarrow +$):



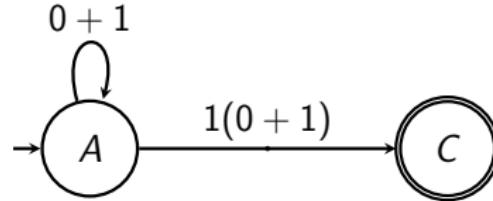
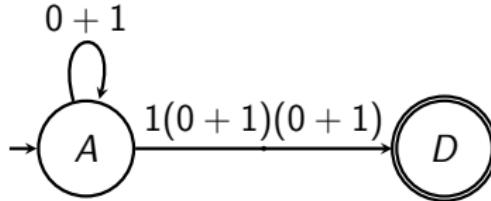
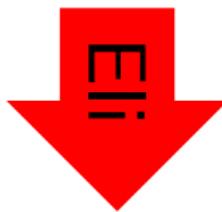
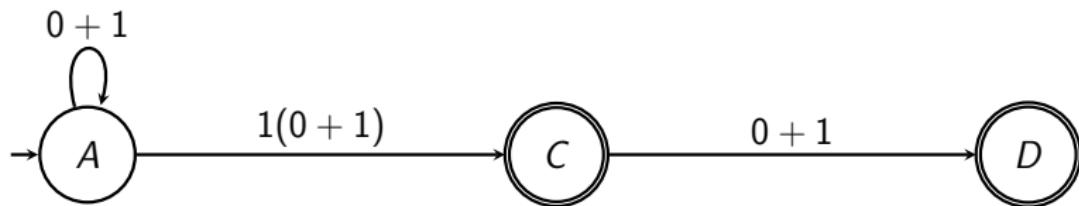
Zustandseliminierung RA II

Eliminierung des Zustands B :



Zustandseliminierung RA III

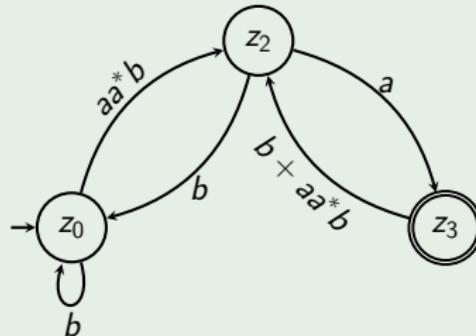
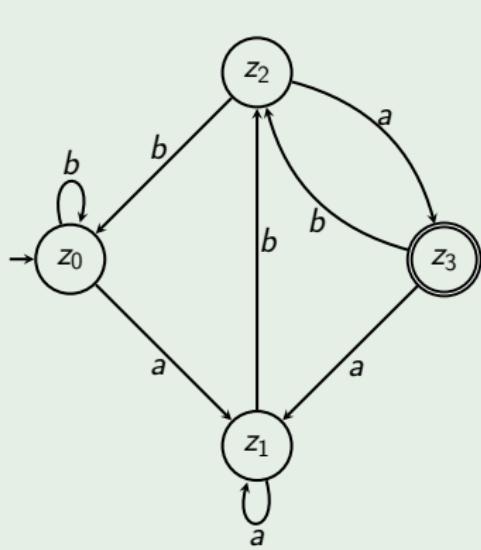
Zwei Zielzustände, also zwei alternative Eliminierungen:



Resultierender RA: $(0 + 1)^*1(0 + 1)(0 + 1) + (0 + 1)^*1(0 + 1)$

Zustandseliminierung RA IV

Beispiel (Bestimmen Sie den RA)



$$\begin{aligned}
 R &= b + aa^*bb & S &= aa^*ba & U &= (b + aa^*b)a \\
 T &= (b + aa^*b)b
 \end{aligned}$$

Resultierender regulärer Ausdruck:

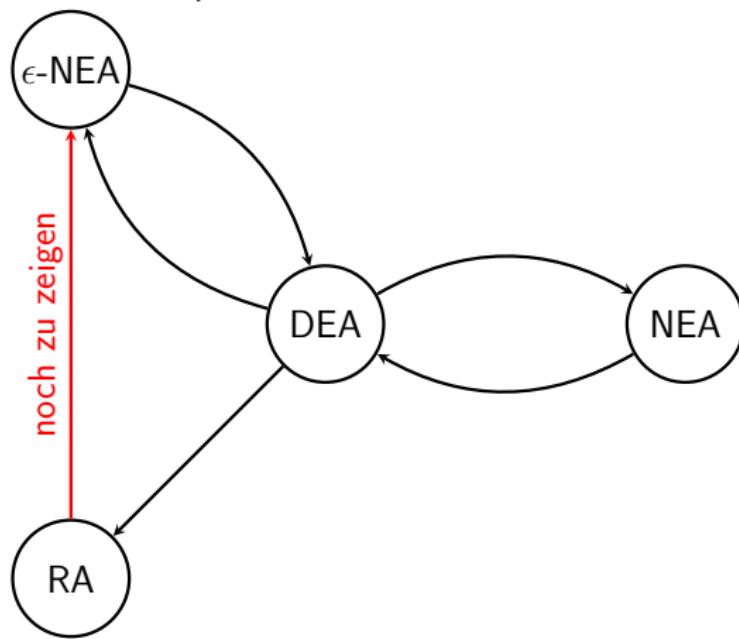
$$(R + SU^*T)^*SU^*$$

$$(b + aa^*bb + (aa^*ba)((b + aa^*b)a)^*(b + aa^*b)b)^*(aa^*ba)((b + aa^*b)a)^*$$

Reguläre Ausdrücke und endliche Automaten

- ▶ EAs und RAs beschreiben dieselben Sprachen:
Reguläre Sprachen

- ▶ Beweisplan für die Äquivalenz:



RA \implies ϵ -NEA

Satz

Für jede Sprache $L(RA)$ existiert ϵ -NEA E , so dass $L(RA) = L(E)$.

Beweisidee:

- ▶ Konstruktion von E
- ▶ genau einen Endzustand
- ▶ keine Transition in den Startzustand
- ▶ keine Transition aus dem Endzustand

Beweisidee: Induktion über Anzahl der Operationen in RA

RA $\implies \epsilon\text{-NEA}$

Satz

Für jede Sprache $L(RA)$ existiert ϵ -NEA E , so dass $L(RA) = L(E)$.

Beweis:

Induktionsanfang

Es gilt für

$$r = \epsilon \quad r = \emptyset \quad r = a \quad \forall a \in \Sigma$$

Induktionsannahme

r, s sind RA mit n Operationen und auch Sprache von ϵ -NEA E

Induktionsschluss

Es ist zu zeigen, das dann auch:

$$r + s \quad r \cdot s \quad r^* \quad (r)$$

Induktionsanfang: ϵ -NEAs für elementare RAs

1 $r = \epsilon$

$$L(\epsilon) = L(E) = \{\epsilon\}$$



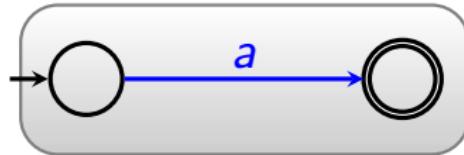
2 $r = \emptyset$

$$L(\emptyset) = L(E) = \emptyset$$



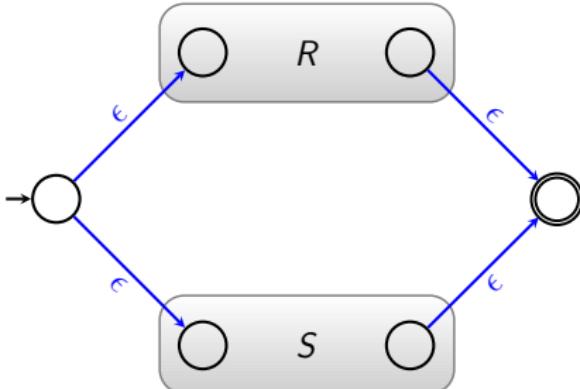
3 $r = a \quad \forall a \in \Sigma$

$$L(a) = \{a\} = L(E)$$

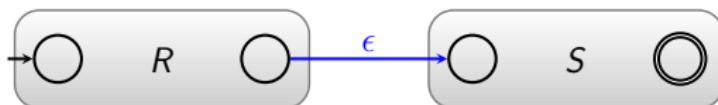


Induktionsschluss: ϵ -NEAs für zusammengesetzte RAs

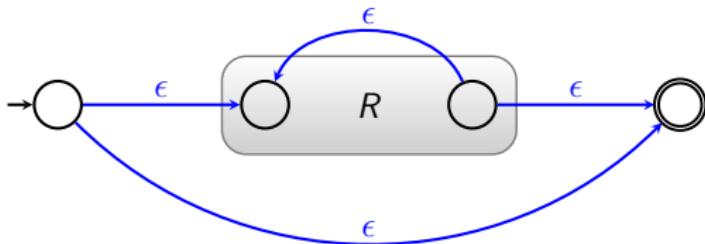
- 1 Vereinigung: $r + s$
 $L(E) = L(r) \cup L(s)$



- 2 Verkettung: $r \cdot s$
 $L(E) = L(r) \circ L(s)$



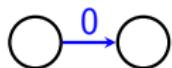
- 3 Hülle: r^*
 $L(E) = \bigcup_{i \geq 0} L(r)^i = L(r^*)$



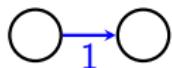
- 4 Klammern: (r)
 $L(E) = L(r) = L((r))$

Beispiel: Übertragung des RA $(0 + 1)^*1(0 + 1)$ – Anfang

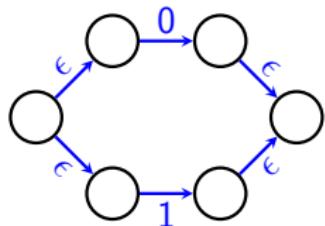
1. ϵ -NEA für 0

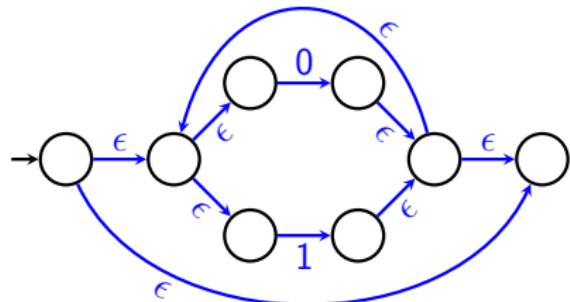
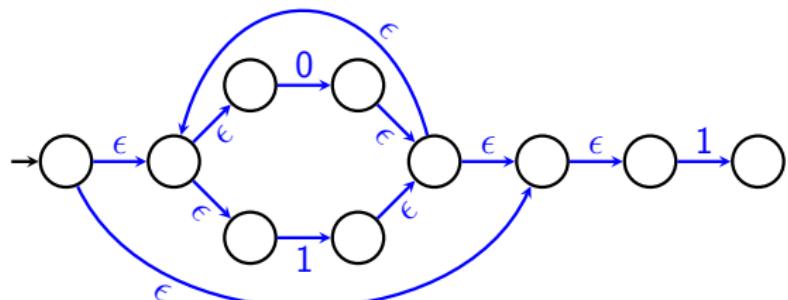


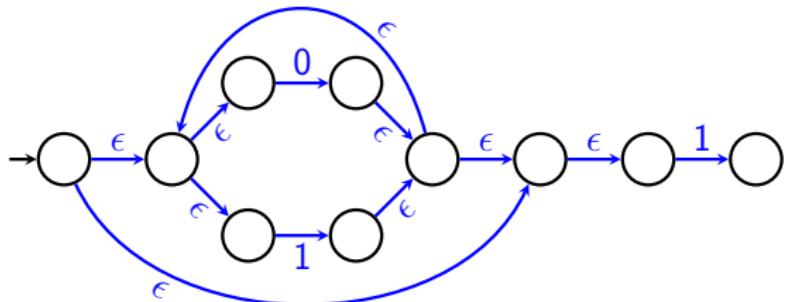
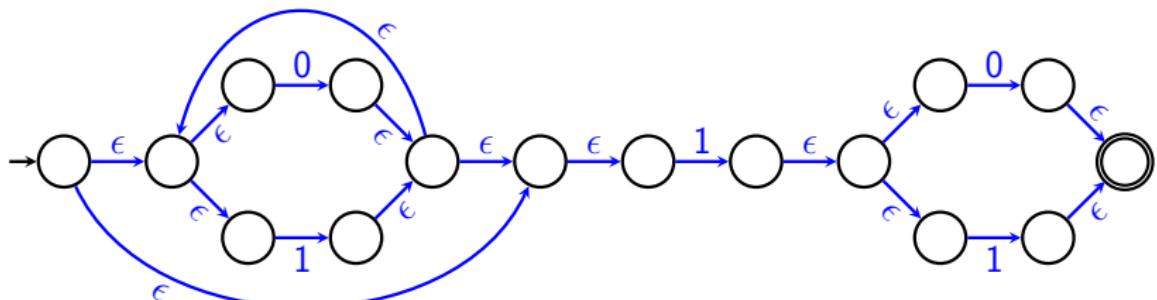
2. ϵ -NEA für 1



3. ϵ -NEA für $(0 + 1)$



Beispiel: Übertragung des RA $(0 + 1)^*1(0 + 1)$ – Mitte4. ϵ -NEA für $(0 + 1)^*$ 5. ϵ -NEA für $(0 + 1)^*1$ 

Beispiel: Übertragung des RA $(0 + 1)^*1(0 + 1) - \text{Ende}$ 5. ϵ -NEA für $(0 + 1)^*1$ 6. ϵ -NEA für $(0 + 1)^*1(0 + 1)$ 

Einige erste Aufgaben XIX

Aufgaben XIX

Kreuzen Sie an was stimmt, korrekte Antworten ergeben je 0,5 Punkte und falsche je –0,5 Punkte.
Keine Antwort gibt natürlich 0 Punkte.

Die Sprachen L_1 und L_2 sind regulär, dann ist auch

- | | | |
|--|-------------------------------|---------------------------------|
| a) die Vereinigung $L_1 \cup L_2$ regulär | <input type="checkbox"/> wahr | <input type="checkbox"/> falsch |
| b) die Verkettung $L_1 \circ L_2$ regulär | <input type="checkbox"/> wahr | <input type="checkbox"/> falsch |
| c) die Potenz L_1^n regulär | <input type="checkbox"/> wahr | <input type="checkbox"/> falsch |
| d) der Stern L_1^* regulär | <input type="checkbox"/> wahr | <input type="checkbox"/> falsch |
| e) das Komplement $\overline{L_1}$ regulär | <input type="checkbox"/> wahr | <input type="checkbox"/> falsch |
| f) der Durchschnitt $L_1 \cap L_2$ regulär | <input type="checkbox"/> wahr | <input type="checkbox"/> falsch |

Abgeschlossenheit regulärer Sprachen I

Wenn zwei Sprachen L_1 und L_2 regulär sind, dann sind folgende Sprachen auch regulär:

- ▶ die Vereinigung $L_1 \cup L_2$:

Gegeben r_i mit $L(r_i) = L_i$ für $i \in \{1, 2\}$,

dann ist $L(r_1 + r_2) = L_1 \cup L_2$ also regulär.

- ▶ die Verkettung $L_1 \circ L_2$:

Gegeben r_i mit $L(r_i) = L_i$ für $i \in \{1, 2\}$,

dann ist $L(r_1 \circ r_2) = L_1 \circ L_2$ also regulär.

- ▶ die Potenz L_1^n :

Gegeben r_1 mit $L(r_1) = L_1$,

dann ist $L(\underbrace{r_1 \circ r_1 \circ \dots \circ r_1}_{n-\text{mal}}) = L_1^n$ also regulär.

- ▶ der Stern L_1^* :

Gegeben r_1 mit $L(r_1) = L_1$,

dann ist $L(r_1^*) = L_1^*$ also regulär.

Abgeschlossenheit regulärer Sprachen II

Satz

Wenn zwei Sprachen L_1 und L_2 regulär sind, dann ist auch

- ① das Komplement $\overline{L_1}$ und
- ② der Durchschnitt $L_1 \cap L_2$ regulär.

Beweisidee:

- ① Gegeben ein DEA \mathbb{A} mit $L(\mathbb{A}) = L_1$. Wie kann dann daraus der Automat für $\overline{L_1}$ konstruiert werden?
- ② Das müsste sich dann „einfach“ durch Mengenoperationen nachweisen lassen.

Abgeschlossenheit regulärer Sprachen III

Beweis.

- ① Gegeben ein DEA $\mathbb{A} = (Z, \Sigma, \delta, z_0, E)$ mit $L(\mathbb{A}) = L$.
Dann kann daraus der Automat für \overline{L} konstruiert werden,
indem die Endzustände und die „Nicht“-Endzustände
vertauscht werden, also durch

$$\overline{\mathbb{A}} = (Z, \Sigma, \delta, z_0, Z \setminus E)$$

$$\begin{aligned} \text{Es gilt } L(\overline{\mathbb{A}}) &= \{ w \mid \delta(z_0, w) \in Z \setminus E \} \\ &= \Sigma^* \setminus \{ w \mid \delta(z_0, w) \in E \} \\ &= \Sigma^* \setminus L(\mathbb{A}) = \Sigma^* \setminus L = \overline{L} \end{aligned}$$

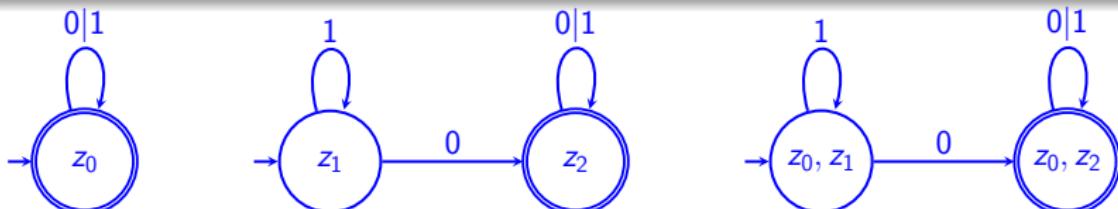
- ② Da $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ gilt, und reguläre Sprachen unter
Komplement und Vereinigung abgeschlossen sind, ist auch
 $L_1 \cap L_2$ regulär.



Definition (Schnitt)

Um einen Automaten zu konstruieren, der genau die Wörter akzeptiert, die zwei Automaten akzeptieren bildet man ihren Produktautomaten. Hierzu sind folgende Schritte notwendig:

- 1 Kreuzprodukt der beiden Zustandsmengen bilden.
⇒ neue Zustände sind Paare
- 2 Transitionen entsprechend übertragen.
- 3 Endzustände sind die Zustände die nur Endzustände enthalten
⇒ Automat akzeptiert genau dann wenn die beiden Ausgangsautomaten akzeptieren.
- 4 Startzustand ist der Zustand, der das Paar aus den beiden ursprünglichen Startzuständen enthält.



Zusammenfassung

„Wir neigen dazu, Dinge für unmöglich zu halten,
nur weil wir sie noch nicht entdeckt haben.“

Spock in M.W. Bonnano: Der Fremde vom Himmel

- ▶ Endliche Automaten erkennen reguläre Sprachen.
- ▶ Reguläre Sprachen sind abgeschlossen unter Komplement, Durchschnitt, Vereinigung, Verkettung, Potenz und Kleene-Stern.
(Tatsächlich auch noch für die Differenz zweier regulärer Sprachen, der Spiegelung einer regulären Sprache, für den Homomorphismus und für den inversen Homomorphismus einer regulären Sprache.)
- ▶ Reguläre Ausdrücke beschreiben reguläre Sprachen.
- ▶ offen: Gibt es Sprachen, die nicht regulär sind?

Was, wenn es keinen Automaten gibt?

Konstruieren Sie bitte einen DEA oder einen regulären Ausdruck über $\{a, b\}$ für

$$L = \{a^n b^n \mid n \geq 0\}$$

Dafür gibt es keinen, aber

- ▶ warum
- ▶ und wie beweisen?

L ist regulär, wenn sich ein endlicher Automat \mathbb{A} oder ein regulärer Ausdruck E finden lässt, der L beschreibt, also:

$$L(\mathbb{A}) = L = L(E)$$

Aber was nun, wenn der sich nicht finden lässt?

Selber schuld

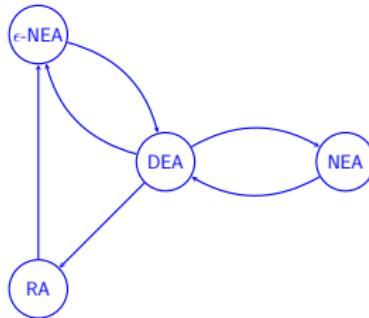
oder

doch nicht?

Nicht-reguläre Sprachen

- ▶ Reguläre Sprachen können beschrieben werden durch:

- ▶ DEAs
- ▶ NEAs
- ▶ ϵ -NEAs
- ▶ Reguläre Ausdrücke



- ▶ Offenbar gibt es aber Sprachen, die nicht regulär sind.

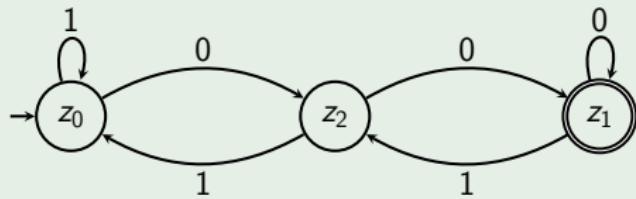
z.B. $L = \{a^n b^n \mid n \geq 0\}$

Dies kann mit Hilfe des **Pumping-Lemmas** bewiesen werden.

Einige erste Aufgaben XX

Aufgaben XX

Berechnen Sie bitte durch Eliminierung den äquivalenten regulären Ausdruck.



Einige erste Aufgaben XXI

Aufgaben XXI

Konstruieren Sie bitte zu den folgenden regulären Ausdrücken, die ϵ -NEAs, die die gleiche Sprache beschreiben:

- ❶ 01^*
- ❷ $(0 + 1)01$
- ❸ $00(0 + 1)^*$

Einige erste Aufgaben und Lösungen XXI

Einige erste Aufgaben XXII

Aufgaben XXII

Konstruieren Sie den DEA, der die Schnittmenge der beiden Sprachen erkennt, die von diesen beiden gegebenen DEA's erkannt werden:

$$\mathbb{A}_1 = \left(Z_1 = \{z_0, z_1\}, \Sigma_1 = \{a\}, \delta_1 = \{(z_0, a) \rightarrow z_1, (z_1, a) \rightarrow z_0\}, z_0, E_1 = \{z_0\} \right) \text{ und}$$

$$\mathbb{A}_2 = \left(Z_2 = \{z_2, z_3, z_4\}, \Sigma_2 = \{a\}, \delta_2 = \{(z_2, a) \rightarrow z_3, (z_3, a) \rightarrow z_4, (z_4, a) \rightarrow z_2\}, z_2, E_2 = \{z_2\} \right).$$

Einige erste Aufgaben und Lösungen XXII

Kapitel 8 - Pumping-Lemma

Version vom 16.07.2015

Sprachen

Vorlesung 1: Sprachen

Vorlesung 2: Endlicher Automat

Vorlesung 3: Endlicher Automat II

Vorlesung 4: Nicht deterministischer endlicher Automat

Vorlesung 5: Epsilon-Endlicher Automat

Vorlesung 6: Reguläre Ausdrücke

Vorlesung 7: Reguläre Ausdrücke II

Vorlesung 8: Pumping-Lemma

Vorlesung 9: Kontextfreie Grammatiken

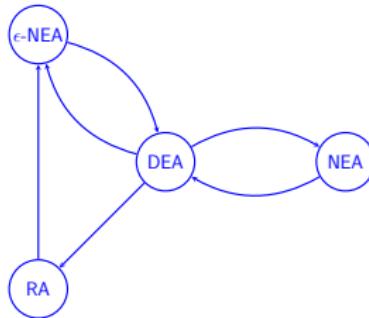
Vorlesung 10: Exoten

Vorlesung 11: Formale Grammatiken

Nicht-reguläre Sprachen

- ▶ Reguläre Sprachen können beschrieben werden durch:

- ▶ DEAs
- ▶ NEAs
- ▶ ϵ -NEAs
- ▶ Reguläre Ausdrücke



- ▶ Offenbar gibt es aber Sprachen, die nicht regulär sind.

z.B. $L = \{a^n b^n \mid n \geq 0\}$

Dies kann mit Hilfe des **Pumping-Lemmas** bewiesen werden.

Informelle Argumentation für nicht-reguläre Sprachen I

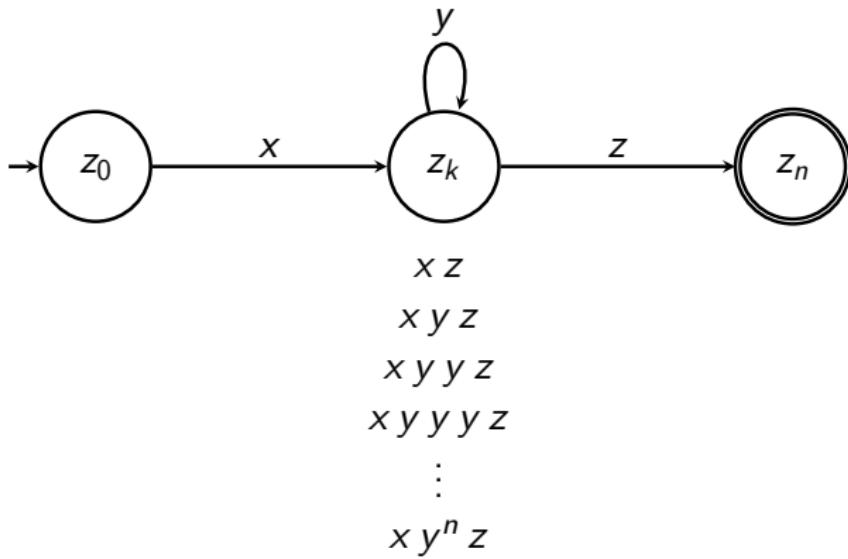
Annahme: $L = \{0^n 1^n \mid n \geq 1\}$ ist regulär

- ⇒ Es gibt einen DEA, der L akzeptiert.
 - o.B.d.A. habe dieser DEA k -Zustände.
 - Nach $k + 1$ Nullen muss mindestens ein Zustand z zweimal erreicht worden sein.
- ⇒ 0^i und 0^j führen beide zu z .

DEA kann nicht „erinnern“, ob er dafür i oder j Nullen gelesen hat.

- ⇒ DEA „weiß“ nicht, ob er 1^i oder 1^j noch lesen muss und kann folglich L nicht erkennen.
- ⇒ Es gibt keinen DEA für L , also doch nicht regulär.

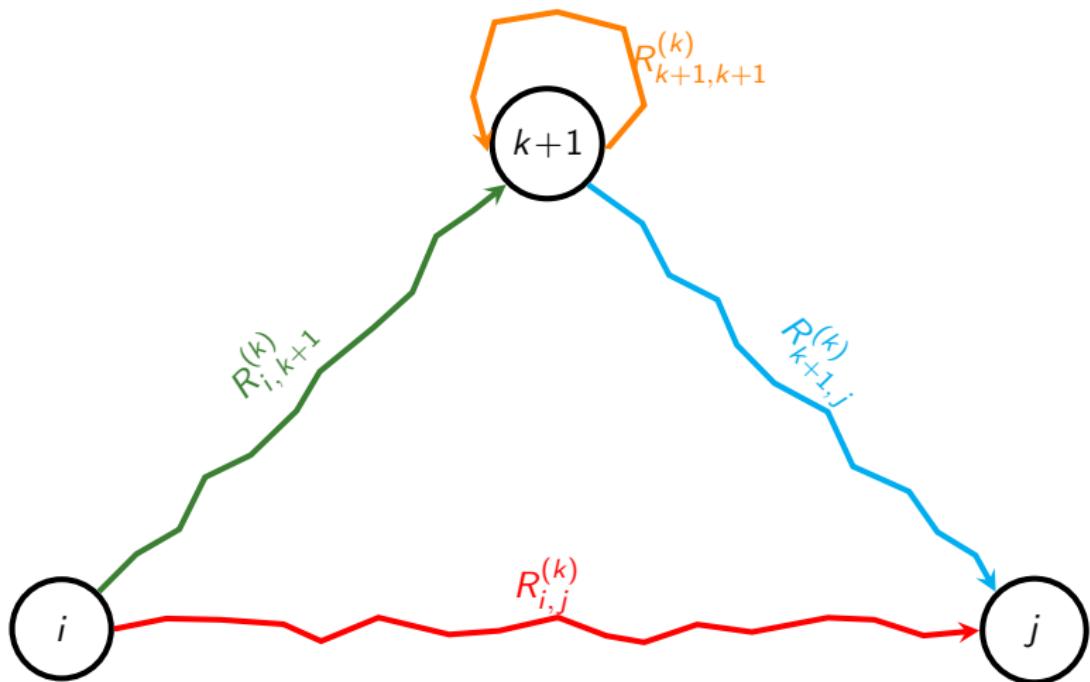
Informelle Argumentation für nicht-reguläre Sprachen II



Noch informeller

Pumping Lemma benutzt die Endlichkeit des Gedächtnisses

- ▶ Wörter dürfen beliebig lang sein.
 - ▶ Gedächtnis (z.B. Anzahl Zustände) ist aber endlich.
 - ▶ Nur möglich, wenn beim Einlesen eines Wortes mindestens ein Zustand mehrfach aufgesucht wird.
 - ▶ Dann hat man eine Schleife.
 - ▶ Die Schleife kann beliebig oft durchlaufen werden.
 - ▶ Entsprechende Wörter müssen dann natürlich auch in der Sprache sein, weil ja kein Gedächtnis (keine obere Schranke) über die Anzahl der Durchläufe existiert.
-
- ▶ **Also:**
Jedes genügend lange Wort liefert die Grundlage für Zyklen.



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

Pumping-Lemma

Pumping Lemma

Sei L eine reguläre Sprache.

Dann gibt es eine natürliche Zahl $n \in \mathbb{N}$,

(die so genannte **PL-Konstante**)

so dass jedes Wort $w \in L$ mit $|w| \geq n$ zerlegt werden kann in

$$w = xyz$$

mit

- 1 $|xy| \leq n$
- 2 $y \neq \epsilon$
- 3 $xy^kz \in L \quad \forall k \in \mathbb{N}_0$

Pumping-Lemma – Beweis Anfang

Beweis Pumping Lemma

L ist regulär.

Also: Es gibt einen DEA \mathbb{A} mit $L = L(\mathbb{A})$.

Annahme: \mathbb{A} habe n Zustände:

Dann gibt es $w = s_1s_2 \dots s_m$, so dass $m \geq n$

Zustände $z_i = \hat{\delta}(z_0, s_1s_2 \dots s_i)$ für $0 \leq i \leq n$

(d.h. $n + 1$ Zustände z_i)

Also: Es existieren i und j , so dass $i < j$, mit $z_i = z_j$.

Zerlegung von w : $w = xyz$ mit

$x = s_1s_2 \dots s_i$,

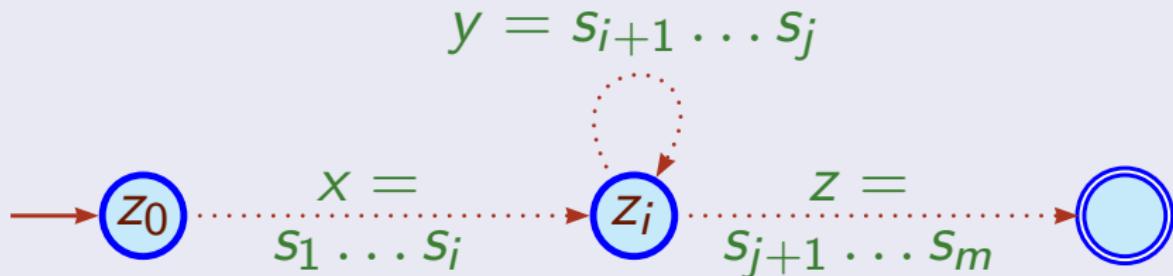
$y = s_{i+1}s_{i+2} \dots s_j$ und

$z = s_{j+1}s_{j+2} \dots s_m$.

Pumping-Lemma – Beweis Ende

Beweis Pumping Lemma

Also: y führt von z_i nach z_i



x kann leer sein, falls $i = 0$ und

z kann leer sein, falls $j = n = m$,

aber **nicht** y , weil $i < j$

Also: xy^kz für $k \geq 0$ ist in L enthalten.



Einige erste Aufgaben XXIII

Aufgaben XXIII

Für die folgenden regulären Sprachen:

- ❶ $L_1 = L(a(bb)^*cbc)$
- ❷ $L_2 = \{w \in \{0, 1\} \mid |w| \geq 13\}$
- ❸ $L_3 = \{a^n \mid n \text{ prim und } n < 10000\}$

bestimmen Sie bitte jeweils eine PL-Konstante p_i , so dass jedes Wort $w \in L_i$ mit $|w| \geq p_i$ zerlegt werden kann in

$$w = xyz$$

mit

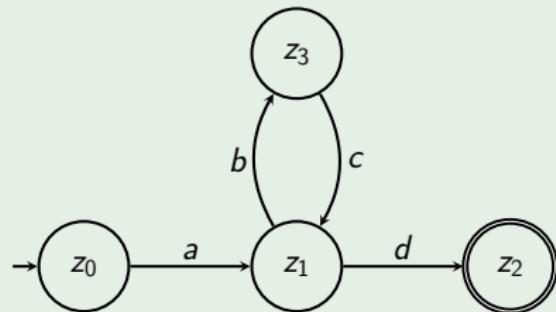
- ❶ $|xy| \leq p_i$
- ❷ $y \neq \epsilon$
- ❸ $xy^kz \in L_i \quad \forall k \in \mathbb{N}_0$

Gesucht ist die PL-Konstante Lösungen XXIII

Einige erste Aufgaben XXIV

Aufgaben XXIV

Sei $L = \{a(bc)^n d \mid n \in \mathbb{N}\}$ mit dem rechts abgebildeten EA für L :



Pumping-Lemma – Anwendungen

Was hilft uns nun dieses Wissen?

Wäre $L = \{a^n b^n \mid n \geq 0\}$ regulär, könnte man L aufpumpen, gemäß dem Lemma.

Ok, but where's the beef?

Indirekter Beweis

Repetitorium – Indirekter Beweis

Indirekter Beweis

$$((A \Rightarrow B) \wedge (A \Rightarrow \overline{B})) \implies \overline{A}$$

Dies ist das Prinzip des *indirekten Beweises*: Wenn aus einer Aussage A etwas Widersprüchliches (nämlich sowohl eine Aussage B als auch ihr Gegenteil \overline{B}) gefolgert werden kann, dann ist damit das Gegenteil von A , also \overline{A} , bewiesen.

Repetitorium – Indirekter Beweis II

Modus Tollens

Der lateinische Name Modus tollendo tollens, „durch Aufheben aufhebende Schlussweise“ erklärt sich daraus, dass es sich um eine Schlussfigur (modus) handelt, die bei gegebener erster Prämisse, $A \Rightarrow B$, durch das „Aufheben“ (tollendo) des Satzes B , also durch das Setzen seiner Verneinung, \overline{B} , einen anderen Satz, nämlich A , ebenfalls „aufhebt“ (tollens), also zu \overline{A} führt.

$$((A \Rightarrow B) \wedge \overline{B}) \Rightarrow \overline{A}$$

Repetitorium – Indirekter Beweis III

ES GELTE

$A \Rightarrow B$

Wenn es regnet, dann ist die Straße nass.

ABER NICHT

$B \Rightarrow A$

Wenn die Straße nass ist, dann regnet es.

Die Straßenreinigung war gerade da...

Es schneit und es ist über 0° C...

ABER DOCH

$\overline{B} \Rightarrow \overline{A}$

Wenn die Straße NICHT nass ist, dann regnet es NICHT.

PL – Indirekter Beweis

Pumping Lemma

Wenn L regulär, dann L aufpumpbar.

ABER NICHT

Wenn L aufpumpbar, dann L regulär.

$L = \{a^n b^n c^m \mid n, m > 0\}$ ist aufpumpbar, aber nicht regulär.
usw.

ABER DOCH

Wenn L NICHT aufpumpbar, dann L NICHT regulär.

Beweise mit Hilfe des Pumping Lemmas

- ▶ Wozu kann das Pumping Lemma benutzt werden?
- ▶ Für Beweise, dass bestimmte Sprachen nicht regulär sind.
- ▶ Wie funktionieren diese Beweise dann? Unter der Annahme, dass die Sprache regulär ist, muss man zeigen, dass es keine langen Wörter geben kann, die in der Sprache enthalten sind und (!) eben die Bedingungen des Pumping Lemmas erfüllen.
- ▶ Der Beweis erfolgt dann durch Widerspruch:
 - ① Man nimmt an, dass die Sprache regulär ist.
 - ② Man betrachtet lange Wörter.
 - ③ Man zeigt, dass bestimmte Zusammenhänge dann aufgrund des Pumping Lemmas gelten müssen.
 - ④ Man zeigt, dass unter Berücksichtigung dieser Zusammenhänge das Wort nicht in der Sprache enthalten sein kann $\Rightarrow \leftarrow$

Beweise mit Hilfe des Pumping Lemmas II

Pumping Lemma

WENN L regulär,

DANN L aufpumpbar:

Dann gibt es eine PL-Konstante $n \in \mathbb{N}$,
so dass jedes Wort $w \in L$ mit $|w| \geq n$ zerlegt werden
kann in

$$w = xyz$$

mit

- 1** $|xy| \leq n$
- 2** $y \neq \epsilon$
- 3** $xy^kz \in L \quad \forall k \in \mathbb{N}_0$

Beweise mit Hilfe des Pumping Lemmas III

Pumping Lemma

WENN L NICHT aufpumpbar:

Dann gibt es **keine** PL-Konstante $n \in \mathbb{N}$,
so dass jedes Wort $w \in L$ mit $|w| \geq n$ zerlegt werden
kann in

$$w = xyz$$

mit

- 1 $|xy| \leq n$
- 2 $y \neq \epsilon$
- 3 $xy^kz \in L \quad \forall k \in \mathbb{N}_0$

DANN L NICHT regulär.

Beweise mit Hilfe des Pumping Lemmas IIIa

Pumping Lemma

WENN L NICHT aufpumpbar:

Dann gibt es für jede beliebige PL-Konstante $n \in \mathbb{N}$, ein Wort $w \in L$ mit $|w| \geq n$, für das es **keine** Zerlegung gibt in

$$w = xyz$$

mit

- 1 $|xy| \leq n$
- 2 $y \neq \epsilon$
- 3 $xy^kz \in L \quad \forall k \in \mathbb{N}_0$

DANN L NICHT regulär.

Beweise mit Hilfe des Pumping Lemmas IIIb

Pumping Lemma

WENN L NICHT aufpumpbar:

Dann gibt es für jede beliebige PL-Konstante $n \in \mathbb{N}$, ein Wort $w \in L$ mit $|w| \geq n$, für das für jede Zerlegung in

$$w = xyz$$

gilt

- 1** $|xy| \leq n$
- 2** $y \neq \epsilon$
- 3** ABER $xy^kz \notin L \quad \exists k \in \mathbb{N}_0$

DANN L NICHT regulär.

$L_{balance} = \{a^m b^m \mid m \geq 0\}$ ist nicht regulär

Beispiel

$L_{balance} = \{a^m b^m \mid m \geq 0\}$ ist nicht regulär

Annahme: $L_{balance}$ ist regulär.

Sei $n \in \mathbb{N}$ eine beliebige PL-Konstante.

Dann gibt es das Wort $w = a^n b^n \in L_{balance}$ mit $|w| = 2n \geq n$, so dass für jede Zerlegung $w = xyz$, nämlich mit

$$x = a^i \text{ mit } 0 \leq i < n$$

$$y = a^j \text{ mit } j > 0 \text{ und } i + j \leq n$$

$$z = a^{(n-j-i)} b^n$$

gilt

1 $|xy| = i + j \leq n$

2 $y \neq \epsilon$, da $j > 0$

3 Aber $xy^0 z = xz = a^i a^{(n-j-i)} b^n = a^{(n-j)} b^n \notin L_{balance}$,

weil $n - j \neq n$ für $j > 0$ ↗

Also ist $L_{balance}$ nicht regulär. □

$L =$

ist nicht regulär

Beispiel

 $L =$

ist nicht regulär

Annahme: L ist regulär.Sei $n \in \mathbb{N}$ eine beliebige PL-Konstante.Dann gibt es das Wort $w = \dots \in L$ mit $|w| = \dots \geq n$, so dass für jede Zerlegung $w = xyz$, nämlich mit

$x = \dots$

$y = \dots$

$z = \dots$

gilt

- 1 $|xy| = \dots \leq n$, gemäß ...
- 2 $y \neq \epsilon$, gemäß ...
- 3 Aber $xy^kz = \dots \notin L$,

weil ... ↴

Also ist L nicht regulär.

□

$L = \{1^m 0 1^m \mid m > 0\}$ ist nicht regulär

Beispiel

$L = \{1^m 0 1^m \mid m > 0\}$ ist nicht regulär

Annahme: L ist regulär.

Sei $n \in \mathbb{N}$ eine beliebige PL-Konstante.

Dann gibt es das Wort $w = 1^n 0 1^n \in L$ mit $|w| = 2n + 1 \geq n$, so dass für jede Zerlegung $w = xyz$, nämlich mit

$$x = 1^i \text{ mit } 0 \leq i < n$$

$$y = 1^j \text{ mit } 0 < j \text{ und } i + j \leq n$$

$$z = 1^{n-i-j} 0 1^n$$

gilt

1 $|xy| = |1^i 1^j| = i + j < n$, gemäß Wahl von i und j

2 $y \neq \epsilon$, gemäß Wahl von y

3 Aber $xy^2z = 1^i 1^j 1^j 1^{n-i-j} 0 1^n = 1^{n+j} 0 1^n \notin L$,

weil $n + j \neq n$ für $j > 0$



Also ist L nicht regulär.



Einige erste Aufgaben

Aufgaben XXV

Behauptung: die Sprache $L = \{0^r 1^s \mid \text{mit } r < s\}$ ist nicht regulär.

Aufgaben XXVI

Behauptung: die Sprache $L = \{a^3 b^m c^{m-3} \mid \text{mit } m > 3\}$ ist nicht regulär.

Aufgaben XXVII

Behauptung: die Sprache $L_{\text{quad}} = \{0^{m^2} \mid \text{mit } m \geq 0\}$ ist nicht regulär.

Einige erste Aufgaben XXVIII

Aufgaben XXVIII

Behauptung: die Sprache $L = \{uu \mid u \in \{0, 1\}^*\}$ ist nicht regulär.

$L = \{uu \mid u \in \{0, 1\}^*\}$ Lösungen XXVIII

Kapitel 9 - Kontextfreie Grammatiken

Version vom 16.07.2015

Sprachen

Vorlesung 1: Sprachen

Vorlesung 2: Endlicher Automat

Vorlesung 3: Endlicher Automat II

Vorlesung 4: Nicht deterministischer endlicher Automat

Vorlesung 5: Epsilon-Endlicher Automat

Vorlesung 6: Reguläre Ausdrücke

Vorlesung 7: Reguläre Ausdrücke II

Vorlesung 8: Pumping-Lemma

Vorlesung 9: Kontextfreie Grammatiken

Vorlesung 10: Exoten

Vorlesung 11: Formale Grammatiken

Kontextfreie Grammatiken

Definition (Kontextfreie Grammatik)

Eine **kontextfreie Grammatik** ist ein 4-Tupel

$G_{kfg} = (N, \Sigma, P, S)$, wobei gilt:

- N nennt man die endliche Menge der **Nichtterminale**,
- Σ nennt man die endliche Menge der **Terminale**, mit $N \cap \Sigma = \emptyset$,
- P ist die endliche Menge der **Produktionen** oder **Regeln**, mit $P \subseteq N \times (N \cup \Sigma)^*$,
- S ist das Startsymbol, mit $S \in N$.

Für die Ersetzungsregeln $\alpha \rightarrow \beta$ gilt, dass auf der linken Seite genau ein Nichtterminal $\alpha \in N$ steht. Diese Regeln gelten egal welche Symbole α in einer Zeichenfolge umgeben, somit ist die Auswahl der Regeln unabhängig vom Kontext von α .

$G_{pal} = (\{S\}, \{a, b\}, P, S)$ ist kontextfrei

Beispiel

$G_{pal} = (\{S\}, \{a, b\}, P, S)$ ist kontextfrei

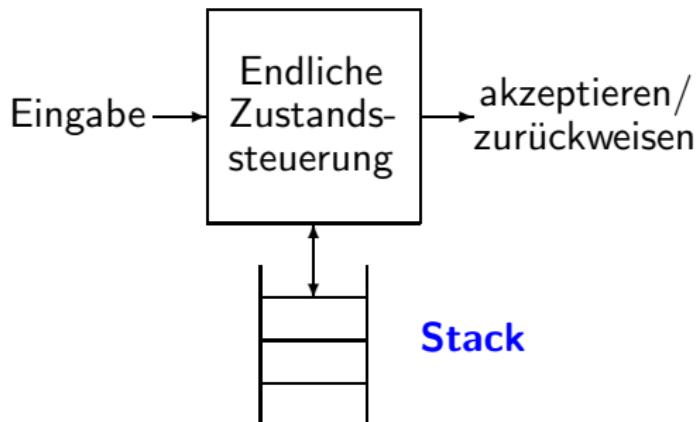
Die Produktion der Grammatik der Palindrome über dem Alphabet $\{a, b\}$ lässt sich mit folgender Tabelle:

Regel	Formel
1.	$S \rightarrow \epsilon$
2.	$S \rightarrow a$
3.	$S \rightarrow b$
4.	$S \rightarrow aSa$
5.	$S \rightarrow bSb$

oder kurz: $S \rightarrow \epsilon|a|b|aSa|bSb$ darstellen.

Kellerautomaten

- ▶ Automaten für kontextfreie Sprachen:
englische Bezeichnung: Pushdown-Automat
deutsche Bezeichnung: Kellerautomat
- ▶ ϵ -NEA mit **Stack**:



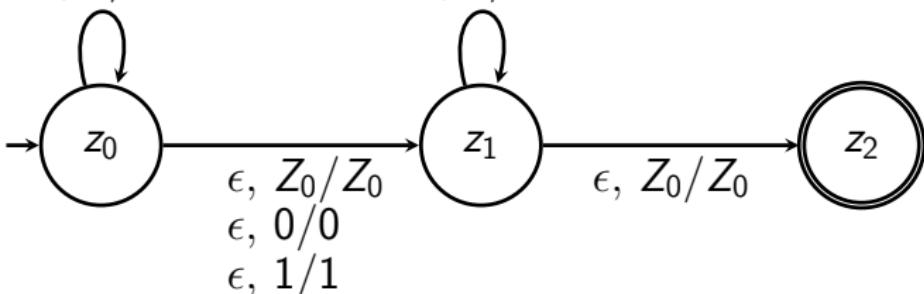
Kellerautomaten II

- ▶ Zusätzliches Stack-Alphabet
- ▶ Übergangsfunktion δ hängt ab von:
 - ▶ Zustand
 - ▶ Eingabesymbol
 - ▶ Oberstem Symbol des Stacks
- ▶ Übergangsfunktion δ legt fest:
 - ▶ Nachfolge-Zustand
 - ▶ Zeichenreihe aus Stacksymbolen, die oberstes Symbol ersetzen
- ▶ Akzeptanz:
 - ▶ durch Endzustand oder
 - ▶ durch leeren Stack

Grafische Notation eines Kellerautomaten

0, $Z_0/0 Z_0$
 1, $Z_0/1 Z_0$
 0, 0/0
 0, 1/01
 1, 0/10
 1, 1/11

0, 0/ ϵ
 1, 1/ ϵ

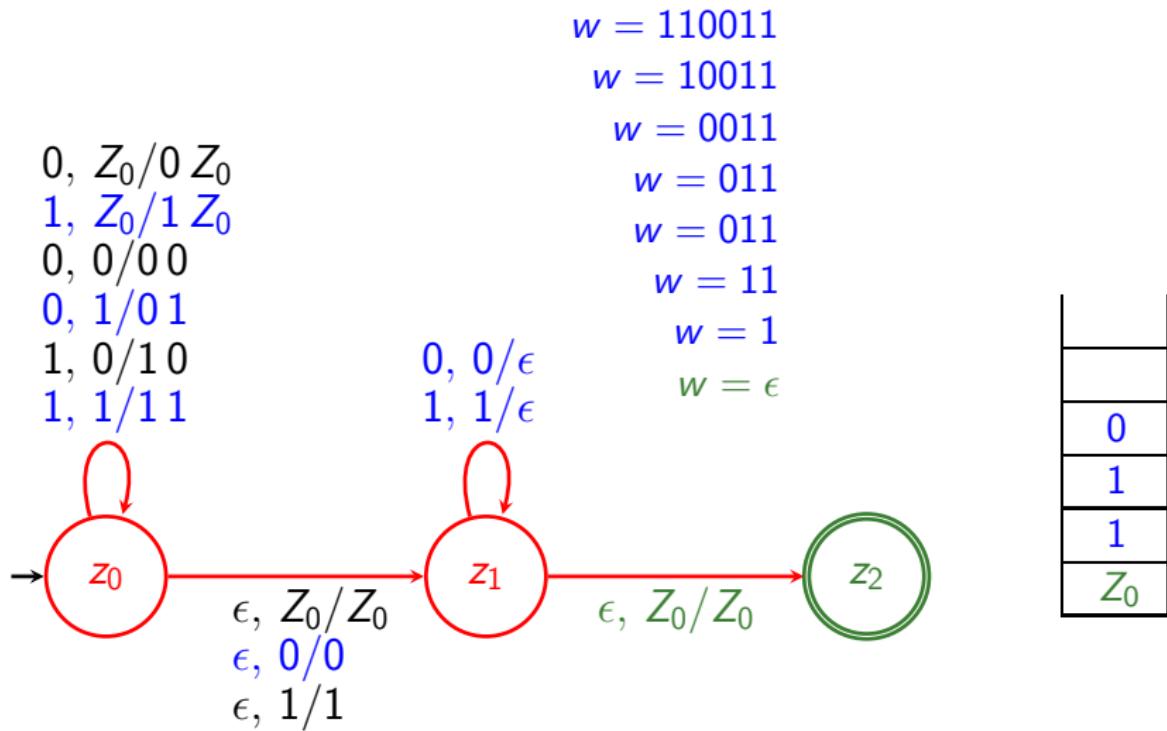


Beispiel

$$L_{ww^R} = \{ ww^R \mid w \in (0+1)^* \}$$

- ❶ ww^R : Zeichenkette w gefolgt von Umkehrung w
- ❷ Alphabet $\Sigma = \{0, 1\}$
- ❸ Stack-Symbole $\Gamma = \{0, 1, Z_0\}$
- ❹ Stack-Startsymbol: Z_0

Arbeitsweise $w = 110011 \in L_{ww^R}$



Arbeitsweise $w = 100 \notin L_{ww^R}$

1. Versuch

$0, Z_0/0 Z_0$

$1, Z_0/1 Z_0$

$0, 0/00$

$0, 1/01$

$1, 0/10$

$1, 1/11$

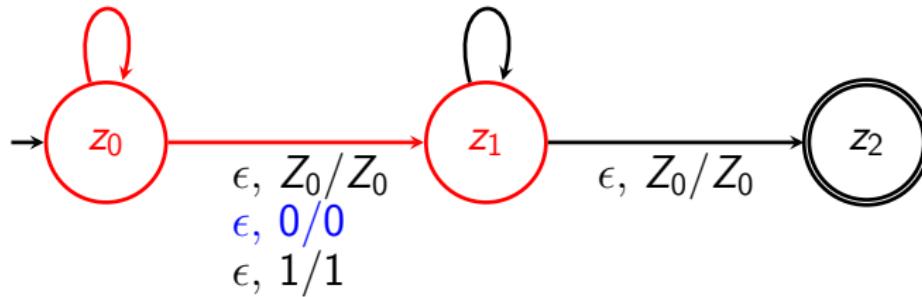
$w = 100$

$w = 00$

$w = 0$

$w = \epsilon$

$w = \epsilon$



0
0
1
Z0

Arbeitsweise $w = 100 \notin L_{ww^R}$

2. Versuch

$0, Z_0/0 Z_0$

$1, Z_0/1 Z_0$

$0, 0/00$

$0, 1/01$

$1, 0/10$

$1, 1/11$

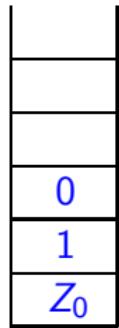
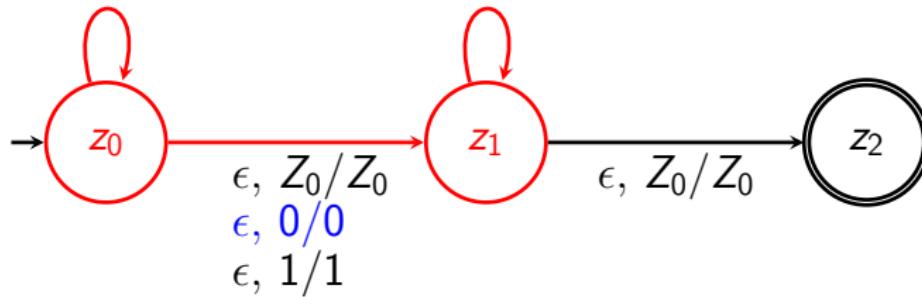
$w = 100$

$w = 00$

$w = 0$

$w = 0$

$w = \epsilon$



Arbeitsweise $w = 100 \notin L_{ww^R}$

$w = 100$

$w = 00$

$w = 00$

$0, Z_0/0 Z_0$

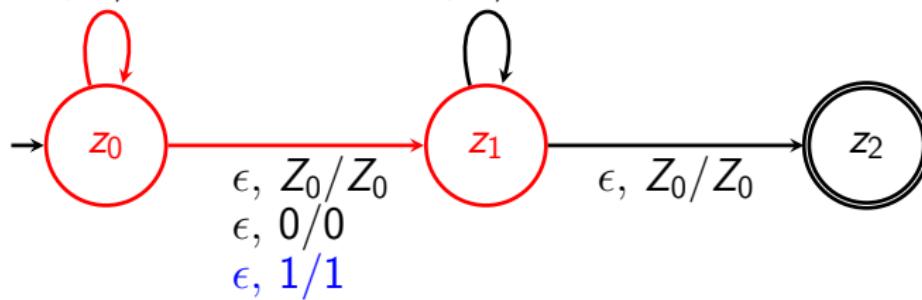
$1, Z_0/1 Z_0$

$0, 0/00$

$0, 1/01$

$1, 0/10$

$1, 1/11$



Arbeitsweise $w = 100 \notin L_{ww^R}$

4. Versuch

$w = 100$

$w = 100$

$w = 100$

0, $Z_0/0 Z_0$

1, $Z_0/1 Z_0$

0, 0/00

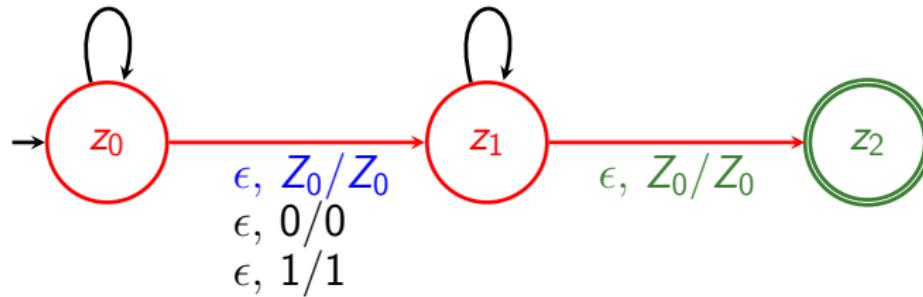
0, 1/01

1, 0/10

1, 1/11

0, 0/ ϵ

1, 1/ ϵ



Kellerautomaten

Definition (NKA)

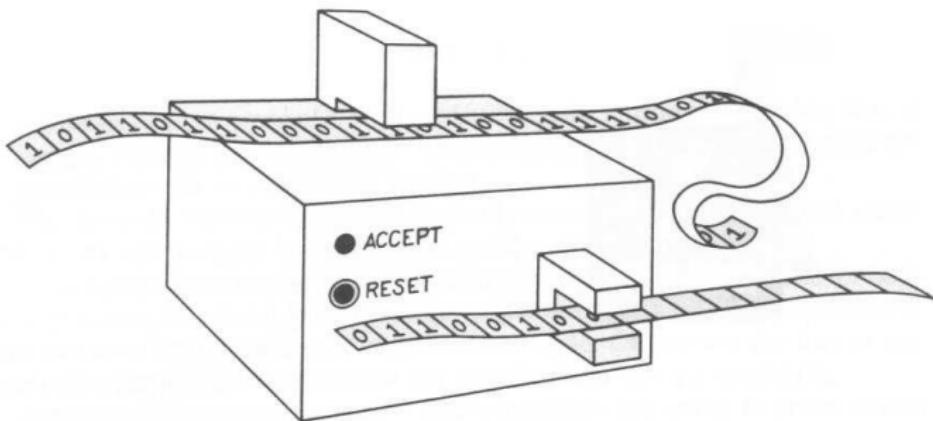
Ein **NKA** (nichtdeterministischer Kellerautomat) ist ein 7-Tupel
 $\mathbb{A}_{NKA} = (Z, \Sigma, \Gamma, \delta, z_0, Z_0, E)$, bestehend aus:

- Z der endlichen **Zustandsmenge**,
- Σ dem endlichen **Eingabealphabet**,
- Γ dem endlichen **Kellaralphabet**,
- δ der **Übergangsfunktion** $\delta : Z \times (\Sigma \cup \{\epsilon\}) \times \Gamma \Rightarrow \mathcal{P}(Z \times \Gamma^*)$,
- $z_0 \in Z$ dem **Anfangszustand**,
- $Z_0 \in \Gamma$ dem **Kellerstartsymbol** (unterstes Kellerzeichen) und
- $E \subseteq Z$ der endliche Menge von **Endzuständen**.

Kellerautomaten I

Notationskonvention

- ▶ $p, q \in Z$
- ▶ $a, b, c, \dots \in \Sigma$
- ▶ $X, Y, Z \in \Gamma$
- ▶ $u, v, w \in \Sigma^*$
- ▶ $\alpha, \beta, \gamma \in \Gamma^*$



Kellerautomaten – Konfiguration

Konfiguration

Konfiguration $K = (q, w, \gamma)$

wobei q der aktuelle Zustand,
 w das noch zu bearbeitende Wort und
 $\gamma \in \Gamma^*$ der Keller ist.

Folgekonfiguration $(q, aw, X\beta) \mapsto (p, w, \alpha \circ \beta)$

falls $(p, \alpha) \in \delta(q, a, X)$

(akzeptierende) Endkonfiguration (q, ϵ, γ)

Akzeptanz durch Endzustand mit $q \in E$ oder

Akzeptanz durch leeren Keller mit $\gamma = \epsilon$

Kellerautomaten – Akzeptanz

Akzeptanz

1 Akzeptanz durch Endzustände:

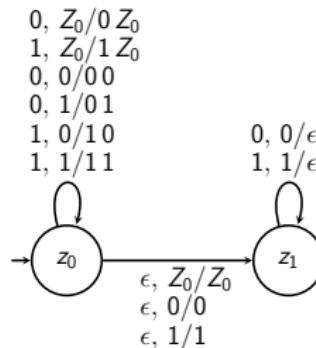
- ▶ Lesen des kompletten Eingabewortes
- ▶ akzeptiert wird, wenn Endzustand
Stack ist egal
- ▶ die durch die Endzustände akzeptierte Sprache ist

$$L(K) = \{ w \in \Sigma^* \mid (z_0, w, Z_0) \xrightarrow{*} (z_e, \epsilon, \gamma) \text{ für bel. } \gamma \text{ und } z_e \in E \}$$

2 Akzeptanz durch leeren Keller:

- ▶ Lesen des kompletten Eingabewortes
- ▶ akzeptiert wird, wenn der Keller leer ist
es gibt keine Endzustände
- ▶ die akzeptierte Sprache ist

$$L(K) = \{ w \in \Sigma^* \mid (z_0, w, Z_0) \xrightarrow{*} (z, \epsilon, \epsilon) \text{ für beliebiges } z \in Z \}$$

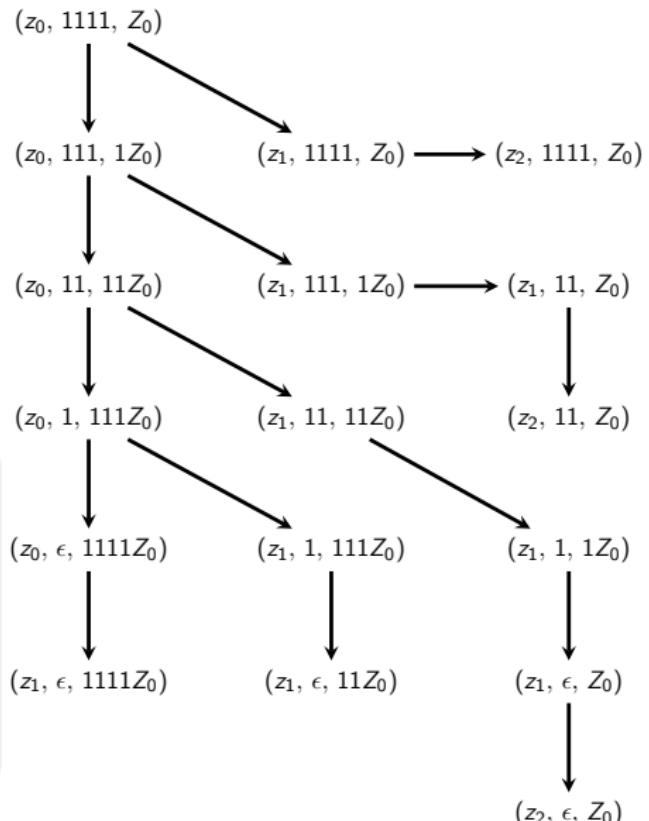
Kellerautomaten – $w = 1111 \in L_{WW^R}$ 

$K = (q, w, \gamma)$ Tripel mit

q : Zustand

w : restliche Eingabe

γ : Inhalt des Stack



Deterministische Kellerautomaten

Deterministischer Kellerautomat

Gegeben sei der Kellerautomat $\mathbb{A}_{DKA} = (Z, \Sigma, \Gamma, \delta, z_0, Z_0, E)$,
dieser ist **deterministisch** sofern für alle $z \in Z$ und alle $X \in \Gamma$ gilt:

- ▶ $|\delta(q, a, X)| \leq 1$, also
maximal ein Folgezustand pro Eingabe- und Stacksymbol
- ▶ $\delta(q, a, X) \neq \emptyset$, dann $\delta(q, \epsilon, X) = \emptyset$, also
entweder spontane oder normale Übergänge

Dadurch ist der Folgezustand, sofern es ihn gibt, eindeutig bestimmt.

Gibt es zu **jedem** NKA einen DKA?

Deterministische Kellerautomaten – II

Gibt es zu jedem NKA einen DKA?

Es gibt keinen deterministischen KA für $L_{ww^R} = \{ww^R \mid w \in \Sigma^*\}$

Warum?

Es ist nicht klar, wo w aufhört und w^R beginnt.

Aber: $L_{w\$w^R} = \{w\$w^R \mid w \in \Sigma^* \wedge \$ \notin \Sigma\}$ kann erkannt werden
(spezielles Symbol als Mittelmarkierung)

Einige erste Aufgaben XXIX

Aufgaben XXIX

Geben Sie bitte einen DKA mit dem Alphabet $\Sigma = \{0, 1\}$ für die Sprache $L_{w\$w^R} = \{w\$w^R \mid w \in \Sigma^* \wedge \$ \notin \Sigma\}$ an.

Deterministische Kellerautomaten – Akzeptanz

- ① Bei NKAs sind die Akzeptanzbedingungen durch leeren Keller oder Endzustände äquivalent.
- ② Bei DKAs ist dies nicht so!
Leerer Keller ist schwächer als Endzustand.

Aufgabe XXIX: Warum?

Weil die Berechnung abbricht, wenn der Keller leer ist.

Satz

Gibt es ein Wort $vw \in L$, dessen echtes Präfix auch ein Wort der Sprache $v \in L$ ist, so akzeptiert der DKA das Präfix v . Da der Keller leer ist, da ja das Präfix akzeptiert wird, kann der DKA vw nicht abarbeiten.

Mit der Endzustandsakzeptanz kann der DKA das Wort bis zum Ende abarbeiten.

Einige erste Aufgaben XXX

Aufgaben XXX

Geben Sie bitte einen DKA mit dem Alphabet $\Sigma = \{a, b\}$ für die Sprache $L_1 = \{a^j b^k \mid j = k\}$ an.

Einige erste Aufgaben XXXI

Aufgaben XXXI

Geben Sie bitte einen DKA mit dem Alphabet $\Sigma = \{a, b\}$ für die Sprache $L_3 = \{a^j b^k \mid j \geq k\}$ an.

Einige erste Aufgaben XXXII

Aufgaben XXXII

Geben Sie bitte einen DKA mit dem Alphabet $\Sigma = \{a, b\}$ für die Sprache $L_2 = \{a^j b^k \mid j < k\}$ an.

Kapitel 10 - Exoten

Version vom 16.07.2015

Vorlesung 1: Sprachen

Vorlesung 2: Endlicher Automat

Vorlesung 3: Endlicher Automat II

Vorlesung 4: Nicht deterministischer endlicher Automat

Vorlesung 5: Epsilon-Endlicher Automat

Vorlesung 6: Reguläre Ausdrücke

Vorlesung 7: Reguläre Ausdrücke II

Vorlesung 8: Pumping-Lemma

Vorlesung 9: Kontextfreie Grammatiken

Vorlesung 10: Exoten

Vorlesung 11: Formale Grammatiken

Automaten

Büchi-Automaten

- ▶ nach dem Schweizer Mathematiker Julius Richard Büchi
- ▶ zum Modelchecking von temporal-logischen Formeln
- ▶ unendliche Wörter:
 $\pi = a_1 a_2 a_3 \dots$ mit $a_i \in \Sigma$ für $i \in \mathbb{N}$
- ▶ Sprachen aus unendlichen Wörtern:
 $L^\omega \subseteq \Sigma^\omega = \{a_1 a_2 a_3 \dots \mid a_i \in \Sigma \text{ für } i \in \mathbb{N}\}$
- ▶ normale Automaten (NEA) akzeptieren normale Wörter:
wenn bei Eingabe des Wortes der Automat in einen Endzustand übergehen kann
- ▶ neues vernünftiges Akzeptierungskriterium:
unendliche Wörter haben kein Ende;
keinen letzter Zustand akzeptiert ein unendliches Wort π ,
falls er unendlich häufig Zuständen aus E besucht.

Büchi-Automat

Definition (Büchi-Automat)

Ein **Büchi-Automat** ist ein 5-Tupel

$\mathbb{A}_B = (Z, \Sigma, \delta, Z_0, E)$, wobei gilt:

- Z eine nichtleere, endliche Zustandsmenge,
- Σ einem endlichen Eingabealphabet,
- δ einer Übergangsfunktion $\delta : Z \times \Sigma \rightarrow \mathcal{P}(Z)$,
- Z_0 einer Menge von Startzuständen mit $Z_0 \subseteq Z$ und
- E einer Menge von (akzeptierenden) Endzuständen mit $E \subseteq Z$

Also im Prinzip ein NEA.

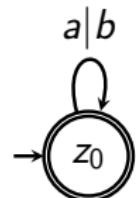
Büchi-Automat – Akzeptanz

Akzeptanz

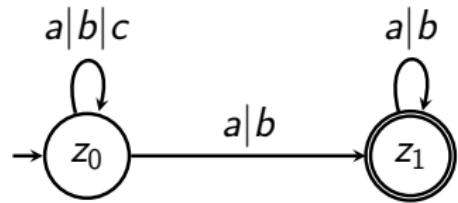
- ▶ Ablauf des Büchi-Automaten für ein unendliches Wort
 $\pi = a_0a_1a_2a_3 \dots$ mit $a_i \in \Sigma$ für $i \in \mathbb{N}_0$ ist eine unendliche Folge $z_0z_1 \dots z_i \dots$ von Zuständen, für die $z_0 \in Z_0$ und $z_{i+1} \in \delta(z_i, a_i)$ für alle $i \in \mathbb{N}_0$.
- ▶ Büchi-Automat $\mathbb{A}_B = (Z, \Sigma, \delta, Z_0, E)$ akzeptiert ein unendliches Wort π , falls es einen Ablauf für π gibt, der unendlich viele Elemente von E enthält.
- ▶ Die von \mathbb{A}_B akzeptierte Sprache L_B^ω ist die Menge der von B akzeptierten unendlichen Wörter.
- ▶ Eine Sprache L^ω heißt ω -regulär, falls es einen endlichen Büchi-Automaten \mathbb{A}_B mit $L^\omega = L_B^\omega$ gibt.

BSP. Büchi-Automaten I

- $L_B^\omega = \{w \in \{a, b, c\}^\omega \mid w \text{ enthält kein } c\}$

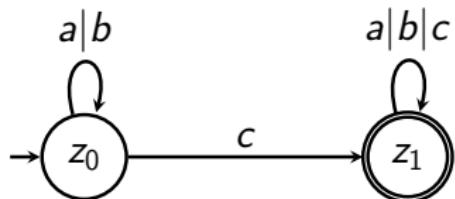


- $L_B^\omega = \{w \in \{a, b, c\}^\omega \mid w \text{ enthält endlich viele } c's\}$

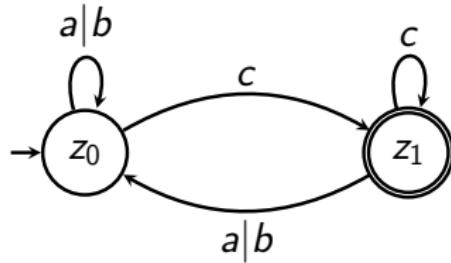


BSP. Büchi-Automaten II

- $L_B^\omega = \{w \in \{a, b, c\}^\omega \mid w \text{ enthält mindestens ein } c\}$



- $L_B^\omega = \{w \in \{a, b, c\}^\omega \mid w \text{ enthält unendlich viele } c's\}$

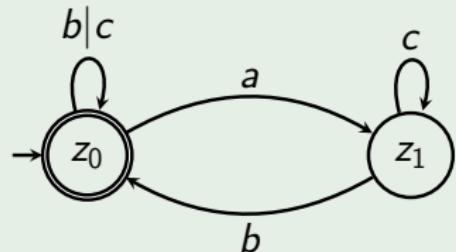


Einige erste Aufgaben XXXIII

Aufgaben XXXIII

Geben Sie bitte einen Büchi-Automaten an für:

$$L_B^\omega = \{ w \in \{a, b, c\}^\omega \mid \text{in } w \text{ folgt auf jedes } a \text{ irgendwann ein } b \}$$



Kapitel 11 - Formale Grammatiken

Version vom 09.09.2015

Sprachen

Vorlesung 1: Sprachen

Vorlesung 2: Endlicher Automat

Vorlesung 3: Endlicher Automat II

Vorlesung 4: Nicht deterministischer endlicher Automat

Vorlesung 5: Epsilon-Endlicher Automat

Vorlesung 6: Reguläre Ausdrücke

Vorlesung 7: Reguläre Ausdrücke II

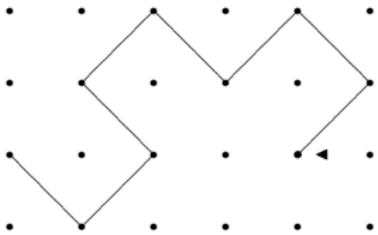
Vorlesung 8: Pumping-Lemma

Vorlesung 9: Kontextfreie Grammatiken

Vorlesung 10: Exoten

Vorlesung 11: Formale Grammatiken

Formale Grammatiken



en.wikipedia.org/wiki/File:Fractal_weeds.jpg

Drachenkurve

Formale Grammatiken und Sprachen

für natürliche Sprachen:

- ▶ neue Äußerungen können produziert und verstanden werden
- ▶ Grammatiken sind Beschreibungen von sprachlichen Strukturen (Schablonen, die über Äußerungen gelegt werden)

für formale Sprachen:

- ▶ Eine Grammatik produziert alle Wörter einer formalen Sprache.
- ▶ Eine Grammatik ist ein deduktives System, das Axiome und Herleitungsregeln enthält.
- ▶ Jedes Wort wird aus den Axiomen erzeugt.

Formale Sprachen und Chomsky-Grammatiken

- ▶ Eine formale Sprache ist eine Menge von Wörtern über einem Alphabet.
- ▶ Eine Chomsky-Grammatik besteht aus
 - ▶ einem terminalen Alphabet
 - ▶ einem nicht-terminalen Alphabet
 - ▶ einem Startsymbol und
 - ▶ einer Regelmenge.
- ▶ Eine Chomsky-Grammatik erzeugt eine formale Sprache.

Wer ist Noam Chomsky?

- ▶ * 1928 - , Linguist und Professor am MIT, USA
- ▶ entwickelte die Theorie der generativen Grammatik
- ▶ revolutionierte den Begriff von Sprache



Noam Chomsky auf dem Weltsozialforum 2003 [wikipedia]

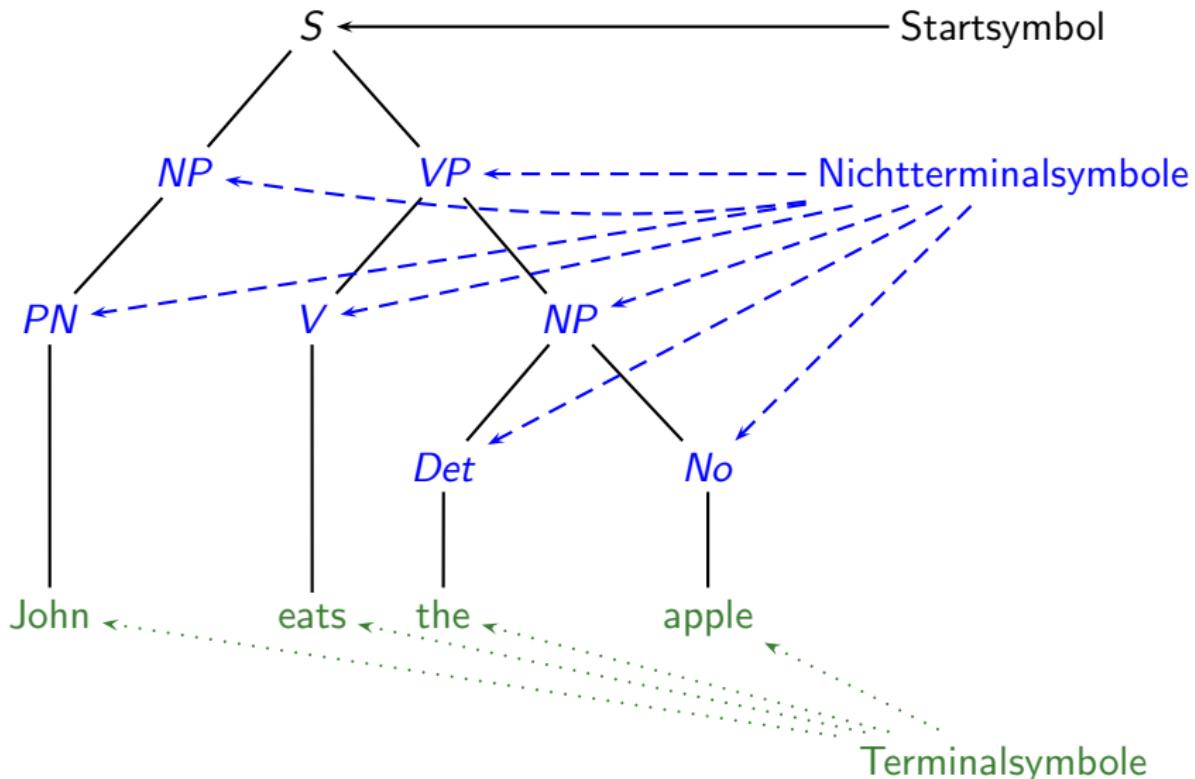
Heute:

- ▶ massiver Kritiker der US Regierung („a leading terrorist state“)
- ▶ Autor des kontroversen Bestsellers 9-11, eine Analyse des Angriffs auf das World Trade Center
- ▶ Globalisierungsgegner: aktiv bei Attac
- ▶ Medienkritiker: „Massenmedien decken nicht auf, sondern verhüllen“.

Chomskys Ideen

- ▶ Allen Sprachen unterliegt dieselbe tiefe Struktur, die von Menschen angewendet wird, ohne sie vorher erlernt zu haben.
- ▶ Sprache wird durch Regeln beschrieben.
- ▶ Unzählige syntaktische Kombinationen können aus der Anwendung von Regeln erzeugt werden.
„John sieht Mary.“ \Rightarrow „Mary wird von John gesehen.“
- ▶ Seine Sprachklassifikation (Chomsky-Hierarchie) hatte großen Einfluss auf die Informatik.
- ▶ Linguisten haben seine Arbeit zunehmend kritisiert. Sie bezweifeln die Regelhaftigkeit menschlicher Wahrnehmung.

BSP: John eats the apple



Chomsky-Grammatik

- $r1: S \rightarrow NP\ VP$ (Sentence, NominalPhrase, VerbalPhrase)
- $r2: NP \rightarrow PN$ (PersonalName)
- $r3: NP \rightarrow Det\ No$ (Determiner, Noun)
- $r4: VP \rightarrow V$
- $r5: VP \rightarrow V\ NP$
- $r6: PN \rightarrow John \mid Mary \dots$
- $r7: No \rightarrow man \mid woman \mid dog \mid bird \mid apple \dots$
- $r8: V \rightarrow sings \mid eats \mid bites \mid loves \dots$
- $r9: Det \rightarrow the$

Chomsky-Grammatik I

Definition (Chomsky-Grammatik)

Eine **Chomsky-Grammatik** ist ein 4-Tupel

$G = (N, T, S, R)$, wobei gilt:

N ist eine endliche Menge von Nichtterminalen,

T ist eine endliche Menge von Terminalen. Dabei gilt $N \cap T = \emptyset$,

S ist das Startsymbol mit $S \in N$ und

R ist eine Regelmenge bestehend aus Regeln $u \rightarrow v$.

Chomsky-Grammatik – Beispiele

$$G_1 = (N_1, T_1, S_1, R_1) \quad G_2 = (N_2, T_2, S_2, R_2) \quad G_3 = (N_3, T_3, S_3, R_3)$$

mit $T_1 = \{0, 1\}$ mit $T_2 = \{0, 1\}$ mit $T_3 = \{0, 1, 2\}$

$N_1 = \{S_1\}$ $N_2 = \{S_2\}$ $N_3 = \{S_3, E, K\}$

$R_1 : S_1 \rightarrow 01S_1|\epsilon$ $R_2 : S_2 \rightarrow 0S_21|\epsilon$ $R_3 : S_3 \rightarrow 0S_3E2|0K2|\epsilon$

$2E \rightarrow E2$

$KE \rightarrow 1K|11$

$$L(G_1) = (01)^*$$

regulär

$$L(G_2) = (0^n1^n)$$

nicht regulär
aber kontextfrei

$$L(G_3) = (0^n1^n2^n)$$

nicht regulär
nicht kontextfrei
aber kontextsensitiv

$$S_1 \Rightarrow 01S_1$$

$$\Rightarrow 0101S_1$$

$$\Rightarrow 010101S_1$$

$$\Rightarrow 010101$$

$$S_2 \Rightarrow 0S_21$$

$$\Rightarrow 00S_211$$

$$\Rightarrow 000S_2111$$

$$\Rightarrow 000111$$

$$S_3 \Rightarrow 0S_3E2$$

$$\Rightarrow 00K2E2$$

$$\Rightarrow 00KE22$$

$$\Rightarrow 001122$$

Reguläre-Grammatik I

Definition (Reguläre-Grammatik)

Eine **reguläre-Grammatik** ist ein 4-Tupel

$G = (N, T, S, R)$, wobei gilt:

N ist eine endliche Menge von Nichtterminalen,

T ist eine endliche Menge von Terminalen. Dabei gilt $N \cap T = \emptyset$,

S ist das Startsymbol mit $S \in N$ und

R ist eine Regelmenge bestehend aus Regeln $u \rightarrow v$ mit

- ▶ einer linken Seite $u = X$ mit $X \in N$ und

- ▶ einer rechten Seite v mit $v \in T^* \circ (N \cup \{\epsilon\})$.

Beispiel

Wie sieht die Grammatik zur Sprache

$$L = \{ (00)^n 1 \mid n \geq 0 \}$$

aus?

- ▶ $G = (N, T, S, R)$ mit
- ▶ $T = \{0, 1\}$
- ▶ $N = \{S, E\}$
- ▶ $R = \{S \rightarrow 1, S \rightarrow 0E, E \rightarrow 0S\}$

Ableiten von Wörtern

Anwenden einer Regel $u \rightarrow v$

aus einer regulären Grammatik G auf ein Wort wu : $wu \xrightarrow{G} wv$

BSP: 00001

Grammatik G durch Regelmenge:

$$R : S \rightarrow 1|0E$$

$$E \rightarrow 0S$$

Ableitung beginnend mit Startsymbol S :

$$S \xrightarrow{} 0E \xrightarrow{} 00S \xrightarrow{} 000E \xrightarrow{} 0000S \xrightarrow{} 00001$$

$w \xrightarrow{+} G w'$ Ableiten in mehreren aufeinanderfolgenden,

$w \xrightarrow{(n)} G w'$ Ableiten in n aufeinanderfolgenden und

$w \xrightarrow{*} G w'$ Ableiten in beliebig vielen Schritten.

Erzeugte Sprache

Gegeben sei eine reguläre Grammatik $G = (N, T, S, R)$

- **Satzformen:** $SF(G) = \{ w \in (T \cup N)^* \mid S \xrightarrow{*} G w \}$

Beispiele: 1010A und 00 sind Satzformen.

- **Erzeugte Sprache:** $L(G) = \{ w \in T^* \mid S \xrightarrow{*} G w \}$

Die Sprache enthält alle durch die Grammatik G erzeugbaren Wörter

- **Äquivalenzen von Grammatiken**

Zwei Grammatiken G_1 und G_2 heißen äquivalent, wenn sie dieselbe Sprache erzeugen:

$$L(G_1) = L(G_2)$$

Wortproblem

Definition (Wortproblem)

- ▶ Das **Wortproblem** ist das Entscheidungsproblem, zu einem gegebenen Wort w festzustellen, ob dieses zur Sprache L gehört oder nicht.
- ▶ Die Sprache L hat also ein **entscheidbares Wortproblem**, wenn es einen Algorithmus gibt, der in endlicher Zeit herausfindet, ob $w \in L$ oder nicht.
- ▶ **Jedes** Entscheidungsproblem lässt sich als Wortproblem einer formalen Sprache kodieren.

Wortproblem I

Gegeben ist eine Sprache $L \subseteq \Sigma^*$.

- ▶ Das Wortproblem ist die Frage, ob $w \in L$.
- ▶ Das Wortproblem einer Sprache L ist **entscheidbar**, wenn ihre **totale charakteristische Funktion**

$$\begin{aligned}\chi_L : \Sigma^* &\longrightarrow \{0, 1\} \\ w &\longmapsto \begin{cases} 1 & \text{falls } w \in L \text{ ist,} \\ 0 & \text{sonst.} \end{cases}\end{aligned}$$

berechenbar ist.

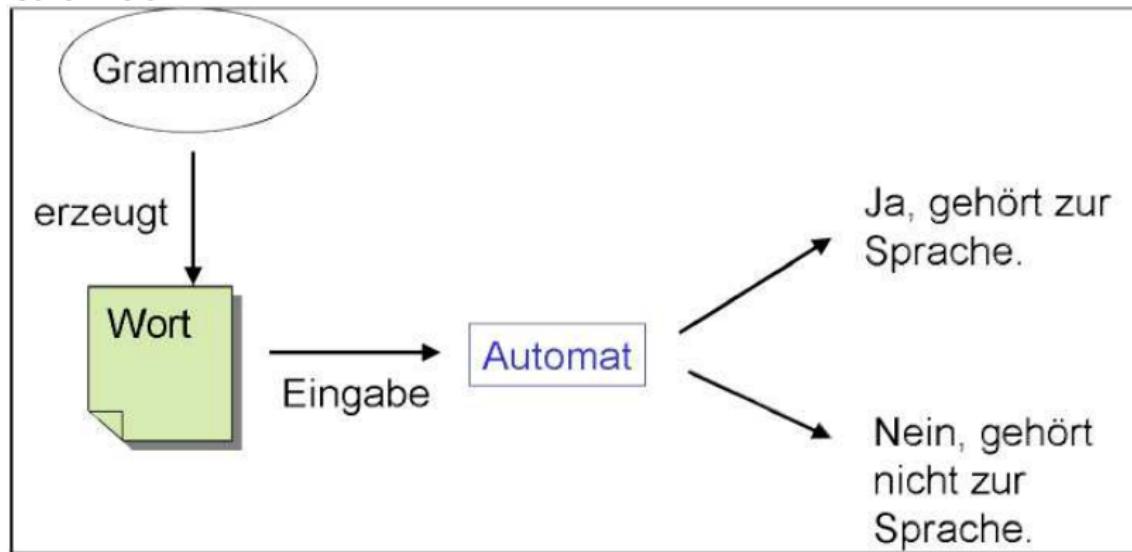
- ▶ Das Wortproblem einer Sprache L ist **semi-entscheidbar**, wenn ihre **partielle charakteristische Funktion**

$$\begin{aligned}\chi_L : \Sigma^* &\longrightarrow \{0, 1\} \\ w &\longmapsto \begin{cases} 1 & \text{falls } w \in L \text{ ist,} \\ \perp & \text{sonst.} \end{cases}\end{aligned}$$

berechenbar ist.

Wortproblem für reguläre Sprachen

ist einfach:



Entscheidbar

Entscheidbar

Beispiel: Aufzählbare Sprache, die nicht entscheidbar ist

$L \subseteq ASCII^*$

$L = \{xy \mid x \text{ ist ein Programm, } y \text{ ist eine Eingabe und}$
 $x \text{ stoppt bei der Eingabe von } y \text{ nach endlich vielen Schritten}\}$

Halteproblem !!

Definition (Kontextsensitive Grammatiken und Sprachen)

- ▶ Eine Grammatik $G = (N, T, S, R)$ heißt **kontextsensitiv** wenn alle Regeln die Form

$$u \rightarrow v \text{ mit } |u| \leq |v|$$

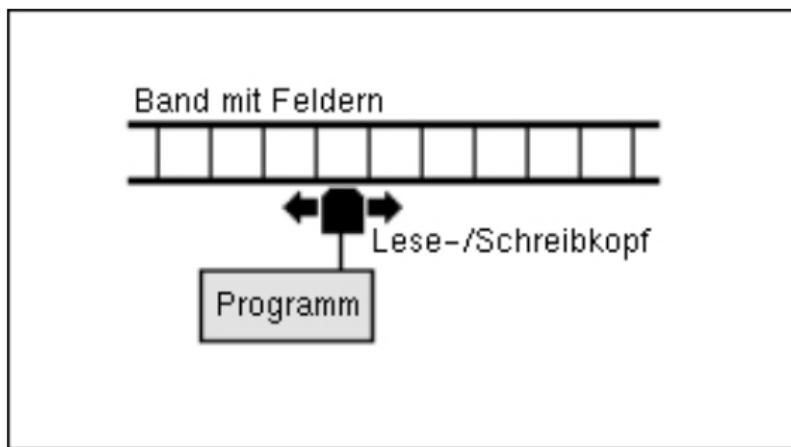
haben, d.h. wenn die linke Seite einer Regel ist nie länger als die rechte Seite.

- ▶ Als einzige Ausnahme ist die Regel $S \rightarrow \epsilon$ zugelassen, um das leere Wort ϵ zu erzeugen.
Dann aber darf das Startsymbol S nicht rekursiv sein,
d.h. nicht auf der rechten Seite einer Regel auftreten.
- ▶ Eine Regel, deren linke Seite nicht länger ist als die rechte Seite, heißt **monoton**.
Daher heißt eine **kontextsensitive Grammatik** auch
monotone Grammatik.

Turingmaschine

- ▶ 1936 Alan Turing
- ▶ Lösung des Entscheidungsproblems
- ▶ Maschine besteht aus
 - ▶ endliche Steuerung/Programm,
 - ▶ Lese/Schreibkopf und
 - ▶ unendlich langes Band.
- ▶ Und viele exotische Abwandlungen.

(Bild: Wikipedia)



Turingmaschine

Definition (TM)

Eine **Turingmaschine** (TM) ist ein 7-Tupel
 $\mathbb{A}_{TM} = (Z, \Sigma, \Gamma, \delta, z_0, \sqcup, E)$, bestehend aus:

- Z der endlichen **Zustandsmenge**,
- Σ dem endlichen **Eingabealphabet**,
- Γ dem endlichen **Bandalphabet**,
- δ der **Übergangsfunktion** $\delta : Z \times \Sigma \times \Gamma \Rightarrow Z \times \Gamma \times D$, mit Bewegungsrichtung der Steuerung $D = \{L, R\}$,
- $z_0 \in Z$ dem **Anfangszustand**,
- $\sqcup \in \Gamma \setminus \Sigma$ dem **leeren Feld (Blank)** (Vorbesetzung Band) und
- $E \subseteq Z$ der endliche Menge von **Endzuständen**.

Beispiel $L(\mathbb{A}_{TM}) = \{0^n 1^n \mid n \geq 1\}$

$$L(\mathbb{A}_{TM}) = \{0^n 1^n \mid n \geq 1\}$$

$$\mathbb{A}_{TM} = (Z, \Sigma, \Gamma, \delta, z_0, \sqcup, E)$$

- Z der endlichen Menge $Z = \{z_0, z_1, z_2, z_3, z_4\}$,
- Σ dem Eingabealphabet $\Sigma = \{0, 1\}$,
- Γ dem Bandalphabet $\Gamma = \{0, 1, X, Y, \sqcup\}$,
- δ der Übergangsfunktion $\delta : Z \times \Sigma \times \Gamma \Rightarrow Z \times \Gamma \times D$,
- z_0 dem Anfangszustand,
- \sqcup dem leeren Feld (Blank) und
- E der Menge $E = \{z_4\}$.

Beispiel $L(\mathbb{A}_{TM}) = \{0^n 1^n \mid n \geq 1\}$

$$L(\mathbb{A}_{TM}) = \{0^n 1^n \mid n \geq 1\}$$

Übergangsfunktion $\delta : Z \times \Sigma \times \Gamma \Rightarrow Z \times \Gamma \times D$ mit

Zustand	Eingabe				
	0	1	X	Y	\sqcup
$\rightarrow z_0$	(z_1, X, R)	—	—	(z_3, Y, R)	—
z_1	$(z_2, 0, R)$	(z_2, Y, L)	—	(z_1, Y, R)	—
z_2	$(z_2, 0, L)$	—	(z_0, X, R)	(z_2, Y, L)	—
z_3	—	—	—	(z_3, Y, R)	(z_4, \sqcup, R)
$* z_4$	—	—	—	—	—

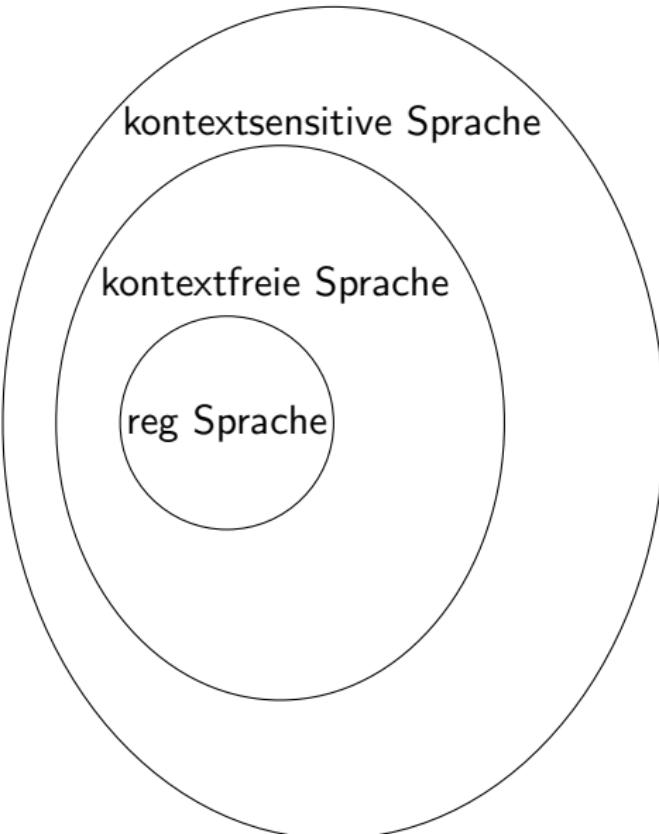
Einige erste Aufgaben XXXIV

Aufgaben XXXIV

Geben Sie bitte das Zustandsübergangsdiagramm für
 $L(\mathbb{A}_{TM}) = \{0^n 1^n \mid n \geq 1\}$ an.

Chomsky-Hierarchie

- ▶ Reguläre Sprachen
 - ▶ Beispiel:
 $L = \{a^n b^m \mid n, m \in \mathbb{N}\}$
 - ▶ Wie werden sie erkannt?
Endliche Automaten
- ▶ Kontextfreie Sprachen
 - ▶ Beispiel: $L = \{a^n b^n \mid n \in \mathbb{N}\}$
 - ▶ Wie werden sie erkannt?
Kellerautomaten
- ▶ Kontextsensitive Sprachen
 - ▶ Beispiel:
 $L = \{a^n b^n c^n \mid n \in \mathbb{N}\}$
 - ▶ Wie werden sie erkannt?
nichtdeterministische TMs



Vergleich von Grammatiken

Falsche Annahme:

Je mächtiger, desto mehr Wörter.

Umfangreichste Sprache Σ^* für gegebenes Alphabet – kann durch reguläre Grammatik beschrieben werden: $S \rightarrow aS|\epsilon$ für alle $a \in \Sigma$.

Statt dessen:

Grenze zwischen Wörtern in und außerhalb der Sprache hat einen komplexeren Verlauf.

Mengendiagramm für Sprachen

abzählbare Sprachen

aufzählbare Sprachen (Typ 0)

entscheidbare Sprachen

kontextsensitive Sprachen (Typ 1)

kontextfreie Sprachen (Typ 2)

**reguläre Sprachen
(Typ 3)**

Chomsky-Hierarchie mit Ergänzungen

Grammatik	Regeln	Sprachen	Automat	Abgeschlossenheit
Typ 0	$\alpha \rightarrow \beta$	aufzählbar	Turing Maschine	KSV*
Typ 1	$\alpha \rightarrow \beta$ mit $ \alpha \leq \beta $	kontext-sensitiv	Linear platzbeschränkte, nichtdeterministische Turing Maschine	CKSV*
Typ 2	$A \rightarrow \gamma$	kontextfrei	Nichtdeterministischer Keller-Automat (NKA)	KV*
		LR(k)	Deterministischer Keller-Automat	C
Typ 3	$A \rightarrow aB$ und $A \rightarrow a$	regulär	Endlicher Automat	CKSV*

A, B: Nichtterminale *a, b:* Zeichenketten aus Terminalen *α, β, γ:* beliebig

C: Komplement **K:** Verkettung **S:** Schnitt **V:** Vereinigung $*$: Kleene

Analogie: Bild einer Rose

► **Reguläre Sprache:**

Vertikale und horizontale
Liniensegmente für Silhouette.

► **Kontextfreie Sprache:**

Beliebige Winkel für
Liniensegmente plus
Kreissegmente für Silhouette.

► **Kontextsensitive Sprache:**

Beliebiger Kurvenverlauf für die
Silhouette.

► **Typ 0 Sprache:** Ein perfektes
Bild.

► Die „Gestalt“ der Rose selbst lässt
sich nicht durch eine endliche
Beschreibung vollständig erfassen.



Zusammenfassung

- ▶ Kontextfreie Grammatiken sind geeignet, Blockstrukturen und richtig geklammerte Ausdrücke zu erzeugen.
- ▶ Reguläre Sprachen sind kontextfrei. Es gibt kontextfreie Sprachen, die nicht regulär sind.
- ▶ Kontextfreie Grammatiken werden zur Semantikdefinition von Programmiersprachen benutzt. Sie werden üblicherweise in der Backus-Naur-Form notiert.
- ▶ NKAs (nicht-deterministische PDAs) akzeptieren kontextfreie Sprachen.

Beispiel $L(\mathbb{A}_{TM}) = \{a^n b^n c^n \mid n \in \mathbb{N}\}$

$$L(\mathbb{A}_{TM}) = \{a^n b^n c^n \mid n \in \mathbb{N}\}$$

$$\mathbb{A}_{TM} = (Z, \Sigma, \Gamma, \delta, z_0, \sqcup, E)$$

- Z der endlichen Menge $Z = \{z_0, z_1, z_2, z_3, z_4, z_5\}$,
- Σ dem Eingabealphabet $\Sigma = \{a, b, c\}$,
- Γ dem Bandalphabet $\Gamma = \{a, b, X, Y, Z, \sqcup\}$,
- δ der Übergangsfunktion $\delta : Z \times \Sigma \times \Gamma \Rightarrow Z \times \Gamma \times D$,
- z_0 dem Anfangszustand,
- \sqcup dem leeren Feld (Blank) und
- E der Menge $E = \{z_5\}$.

Beispiel $L(\mathbb{A}_{TM}) = \{a^n b^n c^n \mid n \in \mathbb{N}\}$

Übergangsfunktion $\delta : Z \times \Sigma \times \Gamma \Rightarrow Z \times \Gamma \times D$ mit

Zustand	Eingabe						
	a	b	c	X	Y	Z	„“
$\rightarrow z_0$	(z_1, X, R)	—	—	—	(z_4, Y, R)	—	—
z_1	(z_1, a, R)	(z_2, Y, R)	—	—	(z_1, Y, R)	—	—
z_2	—	(z_2, b, R)	(z_3, Z, L)	—	—	(z_2, Z, R)	—
z_3	(z_3, a, L)	(z_3, b, L)	—	(z_0, X, R)	(z_3, Y, L)	(z_3, Z, L)	—
z_4	—	—	—	—	(z_4, Y, R)	(z_4, Z, R)	$(z_5, „“, L)$
$* z_5$	—	—	—	(z_5, X, L)	(z_5, Y, L)	(z_5, Z, L)	—

- z_0 markiert ein a mit X und geht zu z_1 , oder liest ein Y und geht zu z_4
- z_1 sucht nach rechts um ein b zu finden und es durch ein Y zu markieren, und geht dann zu z_2
- z_2 überliest bs und markiert dann das erste c mit Z und geht zu z_3
- z_3 sucht das erste X von rechts, bleibt rechts davon stehen und geht zu z_0
- z_4 prüft, ob nur noch markierte Zeichen auf dem Band stehen
- z_5 fährt den Kopf an den Anfang

Einige erste Aufgaben XXXV

Aufgaben XXXV

Geben Sie bitte das Zustandsübergangsdiagramm für
 $L(\mathbb{A}_{TM}) = \{a^n b^n c^n \mid n \in \mathbb{N}\}$ an.

Einige erste Aufgaben XXXVI

Aufgaben XXXVI

Konstruieren Sie bitte den zu G äquivalenten Automaten (NEA):
 $G = (\{S, A, B, C\}, \{a, b, c\}, S, R)$ mit

$$S \rightarrow abA|bbB|ccS$$

$$R : A \rightarrow a$$

$$B \rightarrow C|abbB$$

Einige erste Aufgaben XXXVI

Einige erste Aufgaben XXXVII

Aufgaben XXXVII

Betrachten Sie für $\Sigma = \{0, 1\}$ die folgenden regulären Ausdrücke:

$$r_1 = (0^+1^+)^+ \qquad r_2 = (0^*11)^*0^*$$

$$r_3 = (0 + \epsilon)(1^+0)^*1^* \qquad r_4 = (0^*1)^+(01^*)^+$$

$$r_5 = (0 + 1)^*01^+ + (0 + 1)^*10^+ \quad r_6 = 1^*(01^*(0 + \epsilon)1^+)^*$$

Bearbeiten Sie folgende Teilaufgaben für jedes $i = 1, \dots, 6$:

- a)** Nennen Sie jeweils drei Worte, mit $w \in L(r_i)$ bzw. $w \notin L(r_i)$.
- b)** Beschreiben Sie die Sprache $L(r_i)$ in der Form

$$L(r_i) = \{w \in \Sigma^* \mid w \text{ besitzt Eigenschaft } E_i\}$$

Wählen Sie die Eigenschaften E_i dabei möglichst einfach.

- c)** Finden Sie einen regulären Ausdruck R_i^j , der zu R_i äquivalent ist und sich wesentlich von R_i unterscheidet.
- d)** Geben Sie weiterhin einen regulären Ausdruck $\overline{R_i}$ für die Sprache $\Sigma^* \setminus L(r_i)$ an.