Reguläre Ausdrücke, Automaten, reguläre Grammatiken

Katja Markert

Institut für Computerlinguistik
Uni Heidelberg
markert@cl.uni-heidelberg.de
(Zum Teil mit Folien von Anette Frank oder Yannick Versley)

October 31, 2022

Heute

1. Spezifikation von Zeichenketten
2. Reguläre Ausdrücke
3. zum Matching
4. zur Spezifikation
5. Darstellung dieser regulären Ausdrücke mittels Automaten
6. und regulärer Grammatiken

Motivation

- Suche nach Wörtern: verschiedene Wortformen

  Student
  Studentin
  Studenten
  Studentinnen
  ...

  Mind. 4 verschiedene Wortformen (+ gendering) $\longrightarrow$ 4 Suchoperationen?
  Besser: Suche nach *Student*($|(i|e|inne)n$)

- Anwendungen: Suche, Rechtschreibkorrektur, Ersetzungen, Eigennamenerkennung ...

Terminologie

- **Alphabet**: Eine endliche (nicht-leere) Menge von Zeichen/Symbolen. Beispiel:

$$\Sigma_1 = \{a, b\}$$

$$\Sigma_2 = \{a, 1, 4, \%, \widetilde{O}, \lambda, \ddot{\smile}\}$$

- **Zeichenkette (String/Wort)**: Konkatenation von Zeichen endlicher Länge

$$\Sigma_2^* \supseteq \{\varepsilon, aa, 1a4a, 14\%, \lambda\lambda, \lambda, \ldots\}$$

- ε empty string
- **Formale Sprache L über Alphabet Σ**: Menge von Zeichenketten, beliebige Teilmenge von Σ^*

$$L_2 = \{\lambda, \lambda\lambda, \lambda\lambda\lambda, \lambda\lambda\lambda\lambda, \ldots\} \subseteq \Sigma_2^*$$

Terminologie

- Konkatenation von Zeichenketten $\circ$:

$$\lambda\lambda \circ 14\% = \lambda\lambda 14\%$$

$$\lambda\lambda \circ \varepsilon = \lambda\lambda$$

- Operationen auf Sprachen (Beispiele mit L_2 von vorheriger Folie):

$$L_2 \circ \{14\%\} = \{\lambda 14\%, \lambda\lambda 14\%, \lambda\lambda\lambda 14\%, \ldots\}$$

$$L_2 \circ \emptyset = \emptyset$$

$$L_2 \circ \{\varepsilon\} = L_2$$

$$L_2 \cup \{14\%\} = \{14\%, \lambda, \lambda\lambda, \lambda\lambda\lambda, \ldots\}$$

$$L_2 \cup \emptyset = L_2$$

Reguläre Ausdrücke

- Ein regulärer Ausdruck spezifiziert/ist eine Menge von Zeichenketten
- Zur Definition einer Sprache (= Menge von Zeichenketten)
- Zur Suche nach Mustern in einem Text (Korpus)

Definition

Definition regulärer Ausdrücke (induktiv)

Menge Reg_Σ der regulären Ausdrücke über einem gegebenen endlichen Alphabet Σ:

1. Konstanten: $\emptyset \in Reg_\Sigma$; $\varepsilon := \{\varepsilon\} \in Reg_\Sigma$;
   $a(:= \{a\}) \in Reg_\Sigma \ \ \forall a \in \Sigma$
2. Operationen: Falls $R, S \in Reg_\Sigma$, dann sind auch in Reg_Σ:
   - Konkatenation aus einem String aus R und einem in S: RS
   - Veroderung= Vereinigung von R und S: $R|S$
   - Kleene Star: R^* bzw $R*$ kleinste Übermenge von R, die ε enthält sowie unter Konkatenation abgeschlossen ist.

Schreibweise: $R|S$ oft als $R + S$ oder $R \cup S$ geschrieben.

Beispiele

Oft zur Abgrenzung in Slashes.

Über dem Ascii Alphabet:

- /*Medici*/
- /*Opfer Noahs*/
- /*Opfer noahs*/
- /*i*/
- /(*j*|*J*)*unge Frau*/ oder /[*jJ*]*unge Frau*/
- /1|2|3|4|5|6|7|8|9|0/ oder /[1234567890]/
- /a^*/
- /$(ab)^*$/
- /$ab(ab)^*$/
- Äquivalent: $(a|b)^*$ und $(a^*b^*)^*$

Extended Regular Expressions

Definition der regulären Ausdrücke ist minimal. Zur Abkürzung werden oft noch andere Operationen und Zeichenklassen eingeführt, die aber die Expressivität nicht verändern.

Beispiele auf den folgenden Folien.

Optionalität und Wiederholung

- Kleene $*$: keine oder beliebig viele Instanzen. Beispiel: a^*
- Kleene $+$: eine oder beliebig viele Instanzen. Beispiel: $a+$ ($= aa^*$). Was matched $1+0$ über dem Alphabet $\{1,0\}$?
- Optionalität ?: eine oder keine Instanz. Beispiele:
  - $a?$ ($= a|\varepsilon$)
  - $/colou?r/$
- $\{m,n\}$: zwischen m und n Instanzen. Beispiel: $/Ba\{2,4\}!/$
- $\{n\}$: genau n Instanzen. Beispiel: $/Ba\{3\}!/$
- $\{m,\}$: mindestens m Instanzen
- $\{,n\}$: höchstens n Instanzen

Ranges, Zeichenklassen, Wildcard

Hängt oft von spezifischen Standards und Programmiersprachen ab:

Klasse/Muster	Matches	Beispiel
$[A-Z]$	Großer Buchstabe	**D**renched Blossoms
$[a-z]$	Kleiner Buchstabe	**m**y beans were impatient
$[A-Za-z]$	Buchstabe	4 **D**renched Blossoms
$[0-9]$	Einzelne Ziffer	Chapter **1**: Down the Rabbit hole
$\backslash d$	Einzelne Ziffer	Chapter **1**: Down the Rabbit hole
$\backslash D$	keine Ziffer	**C**hapter 1
$\backslash s$	jedes Leerzeichen	
$\backslash S$	kein Leerzeichen	
$\backslash w$	$[a-zA-Z0-9\_]$	!**D**o
$\backslash W$	kein alphanumerisches	**!**Do
. (Punkt)	Jedes Zeichen aus Σ	

Negation

Muster	Matches	Beispiel
[^A – Z]	Kein großer Buchstabe	D**r**enched Blossoms
[^S s]	Weder *s* noch *S*	S**o**phie was impatient
[e^]	*e* oder ^	Chapt**e**r 1
a^b	Das Muster a^b	Look up a^b now

Negation ^ direkt nach eckiger Klammer

Anker, Sonderzeichen und geschützte Zeichen

- Begrenzer (Anker): Beginn/Ende einer Zeile.
  - ^: Kennzeichnet den Beginn einer Zeile
  - $: Kennzeichnet Ende einer Zeile
  - /^*The*.**dog*\.$/ spezifiziert alle Zeilen, die mit *The* beginnen und mit *dog.* enden
- Zeilenumbruch: \n
- Leerzeichen: \s \t
- geschützte Zeichen: \. * \[\] \\ \+ \? \| \(\) \$ usw.

Noch ein paar Beispiele

Muster	Beispiele
colou?r	**color colour**
*oo*h!*	**oh! ooh! oooh!** . . .
o+h!	**oh! ooh! oooh!** . . .
*beg.*n*	**begin begun beg345tn**

Präzedenz der Operatoren

1. Klammern ()
2. Zähler *, +, ?, {n}, {n,m}
3. Sequenzen und Anker *abc* ^ $
4. Disjunktion |

Beispiele:

- /*fein*|*einer*|*einsten*/ → {*fein,einer,einsten*}
- /*f*(*ein*|*einer*|*einsten*)/ → {*fein,feiner,feinsten*}
- /(*EUR* [0 − 9]+,)+/ → EUR 13, EUR 22, EUR 99, ...
- /*the**/ → {th, the, the, thee ...}
- /*a*.+*a*/: greedy matching für plus, Kleene-Stern und andere Wiederholungen. Was matched /*a*.+*a*/ auf *anna in havanna*?
- /*a*.+?*a*/: lazy matching
- /*a*[^*a*]+*a*/ besser als lazy matching

Regexpal

Zum Üben: `https://www.regexpal.com/`

Welche Zeichenketten werden erkannt?

- $/a^*(b|cd)+d?a\{3\}/$
  1. cdbaaa
  2. aadaaa
  3. abcdcdaaa
- $/[\hat{}\ a-z][a-z]^*[1-3][0-9]\{3\}/$
  1. Inf325
  2. Dkfz2034
  3. 63865

Welche Zeichenketten werden erkannt?

- $/a^*(b|cd)+d?a\{3\}/$
  1. cdbaaa wird erkannt
  2. aadaaa wird nicht erkannt
  3. abcdcdaaa wird erkannt
- $/[\hat{}\ a-z][a-z]^*[1-3][0-9]\{3\}/$
  1. Inf325 wird nicht erkannt
  2. Dkfz2034 wird erkannt
  3. 63865 wird erkannt

Schreiben Sie reguläre Ausdrücke

Gegeben sei jeweils das Alphabet $\Sigma = \{a, b\}$. Schreiben Sie jeweils einen regulären Ausdruck für die folgenden Mengen von Zeichenketten.

- Alle Zeichenketten, die nicht mit *ab* enden. Ohne Verwendung des Negationscharakters.
- Alle Zeichenketten gerader Länge.
- Alle Zeichenketten, die genauso viele *a*s wie *b*s enthalten

Beweise, dass eine formale Sprache nicht regulär ist, führt man meist mit dem Pumping Lemma (nicht Bestandteil dieser Vorlesung).

Schreiben Sie reguläre Ausdrücke

Gegeben sei jeweils das Alphabet $\Sigma = \{a, b\}$. Schreiben Sie jeweils einen regulären Ausdruck für die folgenden Mengen von Zeichenketten.

- Alle Zeichenketten, die nicht mit *ab* enden. Ohne Verwendung des Negationscharakters.
- Alle Zeichenketten gerader Länge.
- Alle Zeichenketten, die genauso viele *a*s wie *b*s enthalten

Beweise, dass eine formale Sprache nicht regulär ist, führt man meist mit dem Pumping Lemma (nicht Bestandteil dieser Vorlesung).

Anwendung: Suche mit regulären Ausdrücken

Wie einfach ist dies? Suchen wir nach Vorkommen des englischen definiten Artikels *the*.

- /*the*/
- /[Tt]he/
- / [Tt]he/
- /(|,|\.)[Th]he(|,|\.)/
- /[^a-zA-Z][Tt]he[^a-zA-Z]/
- /(^|[^a-zA-Z])[Tt]he([^a-zA-Z]|$)/

Anwendung: Suche mit regulären Ausdrücken

Wie einfach ist dies? Suchen wir nach Vorkommen des englischen definiten Artikels *the*.

- /*the*/
- /[*Tt*]*he*/
- / [*Tt*]*he*/
- /(|,|\.)[*Th*]*he*(|,|\.)/
- /[^*a*−*zA*−*Z*][*Tt*]*he*[^*a*−*zA*−*Z*]/
- /(^|[^*a*−*zA*−*Z*])[*Tt*]*he*([^*a*−*zA*−*Z*]|$)/

Anwendung: Suche mit regulären Ausdrücken

Wie einfach ist dies? Suchen wir nach Vorkommen des englischen definiten Artikels *the*.

- /*the*/
- /[*Tt*]*he*/
- / [*Tt*]*he*/
- /(^|[^a-zA-Z])[*Tt*]*he*([^a-zA-Z]|$)/

Anwendung: Suche mit regulären Ausdrücken

Wie einfach ist dies? Suchen wir nach Vorkommen des englischen definiten Artikels *the*.

- /*the*/
- /[*Tt*]*he*/
- / [*Tt*]*he*/
- /(|,|\.)[*Th*]*he*(|,|\.)/
- /[^*a*−*zA*−*Z*][*Tt*]*he*[^*a*−*zA*−*Z*]/
- /(^|[^*a*−*zA*−*Z*])[*Tt*]*he*([^*a*−*zA*−*Z*]|$)/

Anwendung: Suche mit regulären Ausdrücken

Wie einfach ist dies? Suchen wir nach Vorkommen des englischen definiten Artikels *the*.

- /*the*/
- /[*Tt*]*he*/
- / [*Tt*]*he*/
- /(|,|\.)[*Th*]*he*(|,|\.)/
- /[^*a*−*zA*−*Z*][*Tt*]*he*[^*a*−*zA*−*Z*]/
- /(^|[^*a*−*zA*−*Z*])[*Tt*]*he*([^*a*−*zA*−*Z*]|$)/

Anwendung: Suche mit regulären Ausdrücken

Wie einfach ist dies? Suchen wir nach Vorkommen des englischen definiten Artikels *the*.

- /*the*/
- /[*Tt*]*he*/
- / [*Tt*]*he*/
- /(|,|\.)[*Th*]*he*(|,|\.)/
- /[^*a*−*zA*−*Z*][*Tt*]*he*[^*a*−*zA*−*Z*]/
- /(^|[^*a*−*zA*−*Z*])[*Tt*]*he*([^*a*−*zA*−*Z*]|$)/

Anwendung: Ersetzen mit regulären Ausdrücken

- Suche und Substitution in Commandline oder vielen Programmiersprachen

$$s/colour/color/$$

- Markierung von Teilsequenzen oft möglich. Nicht Teil normaler regulärer Sprachen (da Gedächtnis notwendig)
  - EUR 100 $\rightarrow$ 100 EUR
  - $s/EUR\ +([0-9]+)/\backslash 1\ EUR/$

Zwischenfazit

- Reguläre Ausdrücke spezifizieren Mengen von Zeichenketten (= formale Sprache) über einem Alphabet
- Sie können zur Suche nach Mustern oder Wortformen in Korpora eingesetzt werden
- Vielzahl von Operatoren und Metacharakteren
- Operatorenpräzedenz
- Es gibt formale Sprachen, die nicht durch reguläre Ausdrücke erzeugt werden können.

Literatur und Materialien

- Übungsblatt 1/2
- Jurafsky und Martin (3rd edition) Kapitel 2
- Knapp, Stefan (2018): Übungsbuch Automaten und Formale Sprachen. 117 Aufgaben und Lösungen. Springer. E-book verlinkt von der Kurswebseite oder siehe Heidi.

Reguläre Sprachen

Definition regulärer Sprachen

Gegeben Alphabet Σ:

1. Die leere Menge $\emptyset$ ist eine reguläre Sprache
2. $\{\varepsilon\}$ ist eine reguläre Sprache, wobei ε das leere Wort ist.
3. $\{a\}$ ist eine reguläre Sprache $\forall a \in \Sigma$
4. Sind L, L_1, L_2 eine reguläre Sprache, so sind auch:
   - die Konkatenation/Verkettung $L_1 \circ L_2 := \{xy | x \in L_1, y \in L2\}$
   - die Vereinigung/Disjunktion $L_1 \cup L_2 = \{x | x \in L_1 \ \textit{oder} \ x \in L_2\}$
   - der Kleene-Abschluss $L^* = \varepsilon \cup L \cup L \circ L \cup L \circ L \circ L \circ L \cup \ldots$

   reguläre Sprachen

Reguläre Sprachen sind genau die Sprachen, die durch reguläre Ausdrücke beschrieben/generiert werden.

Endliche Automaten: Motivation

Effiziente Struktur und Algorithmus für automatische Verarbeitung regulärer Sprachen. Suche mit regulären Ausdrücken:

- Konstruktion eines endlichen Automaten A, der dem regulären Suchausdruck entspricht
- Abgleich von Eingabezeichenketten *s* mit Automat *A*: Erkennungsproblem. Ist *s* ein Element der durch *A* definierten Sprache?

endlicher Automat = EA = Finite State Automaton = FSA = finite state machine

Endliche Automaten: Beispiel

- Gerichteter Graph mit Menge von Knoten (*Zustände*)
- Menge von etikettierten und gerichteten Kanten (*Übergänge*) zwischen Zuständen
- Ein Startzustand, mindestens ein Endzustand

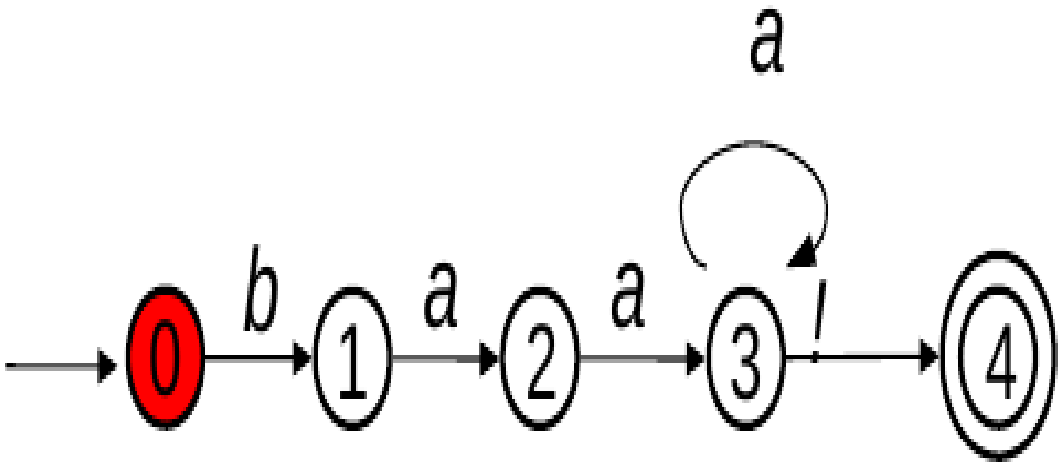


Bild aus Jurafsky und Martin, Edition 2

Erkennungsproblem

Akzeptanz

Eine Zeichenkette/Wort w wird von einem Automaten akzeptiert, falls sich der Automat nach dem (vollständigem) Lesen von w in einem Endzustand befindet.

Erkannte Sprache

Alle Worte, die der Automat ausgehend vom Startzustand lesen kann, so dass nach dem Lesen ein Endzustand erreicht wird.

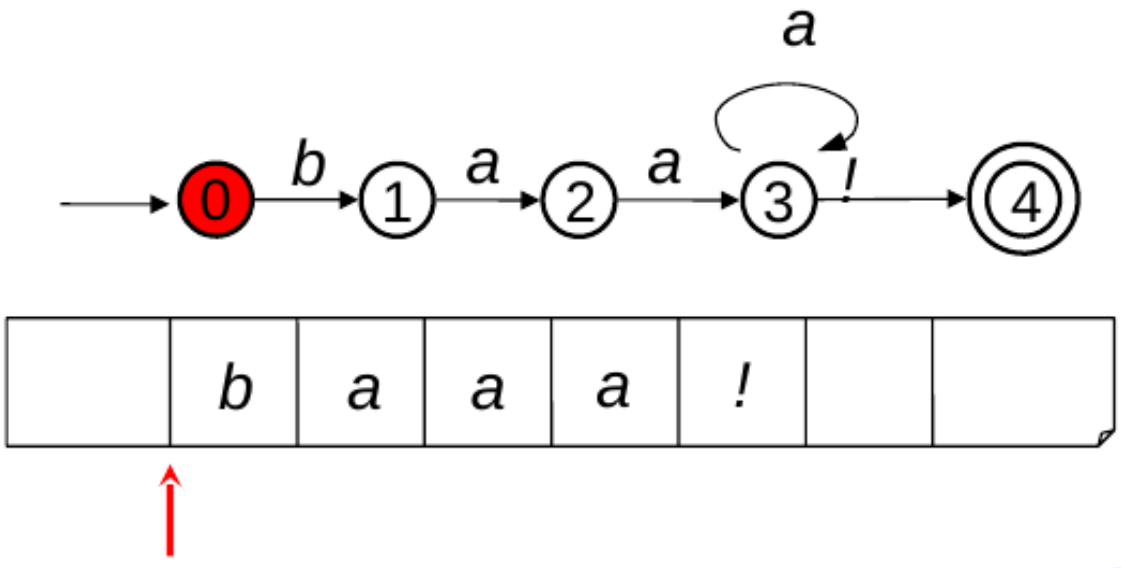
Erkennungsproblem

Akzeptanz

Eine Zeichenkette/Wort *w* wird von einem Automaten akzeptiert, falls sich der Automat nach dem (vollständigem) Lesen von *w* in einem Endzustand befindet.

Erkannte Sprache

Alle Worte, die der Automat ausgehend vom Startzustand lesen kann, so dass nach dem Lesen ein Endzustand erreicht wird.



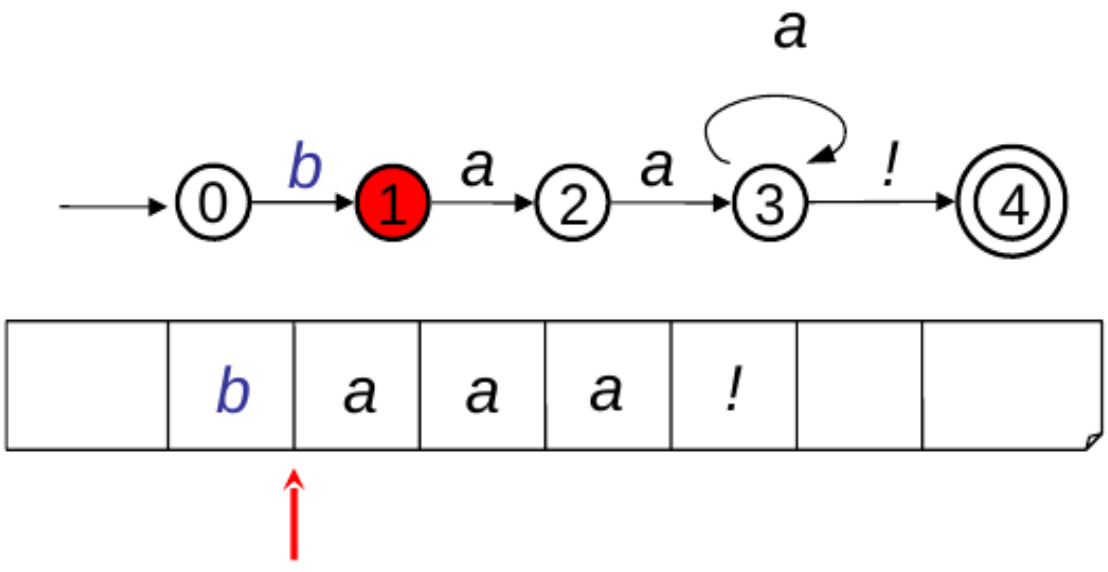
Erkennungsproblem

Akzeptanz

Eine Zeichenkette/Wort *w* wird von einem Automaten akzeptiert, falls sich der Automat nach dem (vollständigem) Lesen von *w* in einem Endzustand befindet.

Erkannte Sprache

Alle Worte, die der Automat ausgehend vom Startzustand lesen kann, so dass nach dem Lesen ein Endzustand erreicht wird.



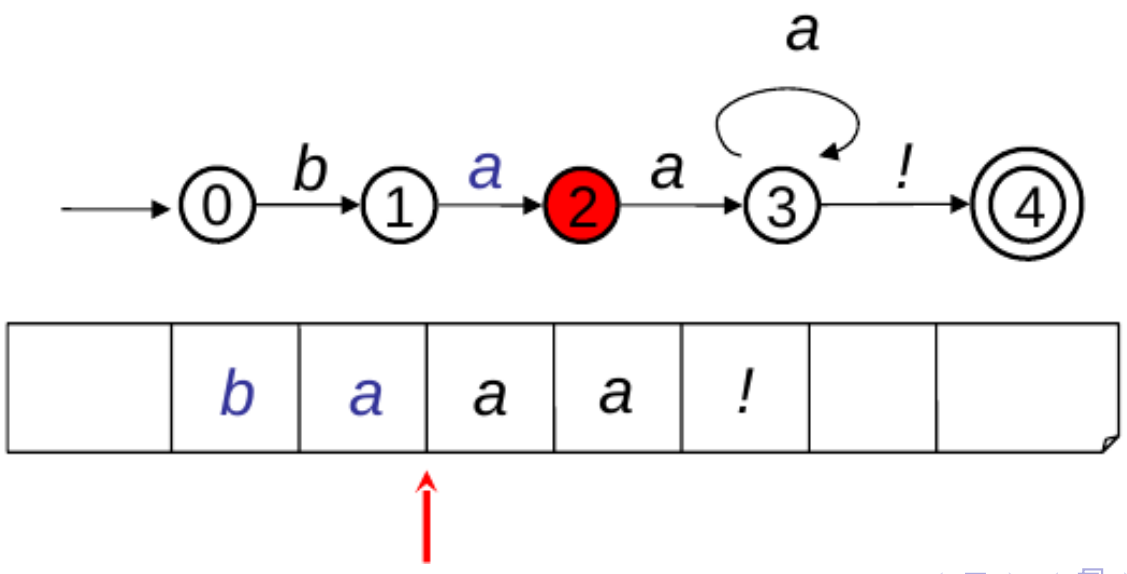
Erkennungsproblem

Akzeptanz

Eine Zeichenkette/Wort *w* wird von einem Automaten akzeptiert, falls sich der Automat nach dem (vollständigem) Lesen von *w* in einem Endzustand befindet.

Erkannte Sprache

Alle Worte, die der Automat ausgehend vom Startzustand lesen kann, so dass nach dem Lesen ein Endzustand erreicht wird.



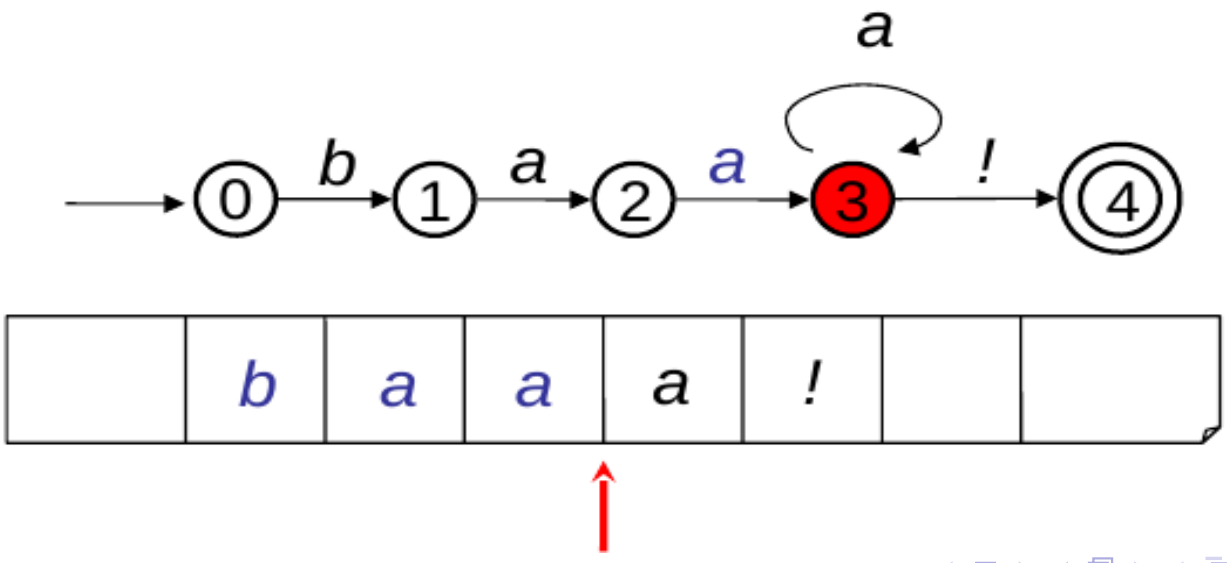
Erkennungsproblem

Akzeptanz

Eine Zeichenkette/Wort *w* wird von einem Automaten akzeptiert, falls sich der Automat nach dem (vollständigem) Lesen von *w* in einem Endzustand befindet.

Erkannte Sprache

Alle Worte, die der Automat ausgehend vom Startzustand lesen kann, so dass nach dem Lesen ein Endzustand erreicht wird.



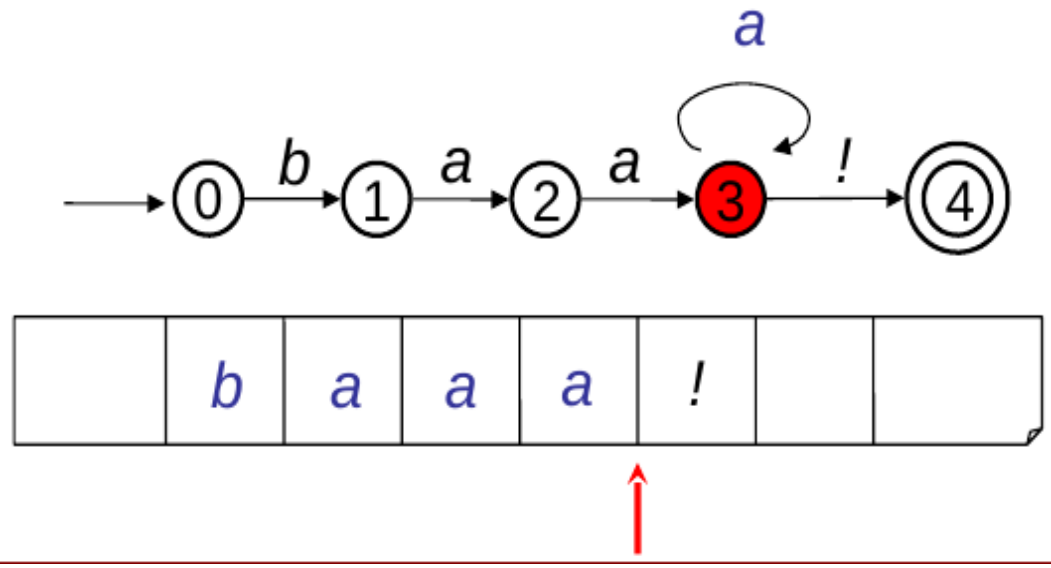
Erkennungsproblem

Akzeptanz

Eine Zeichenkette/Wort *w* wird von einem Automaten akzeptiert, falls sich der Automat nach dem (vollständigem) Lesen von *w* in einem Endzustand befindet.

Erkannte Sprache

Alle Worte, die der Automat ausgehend vom Startzustand lesen kann, so dass nach dem Lesen ein Endzustand erreicht wird.



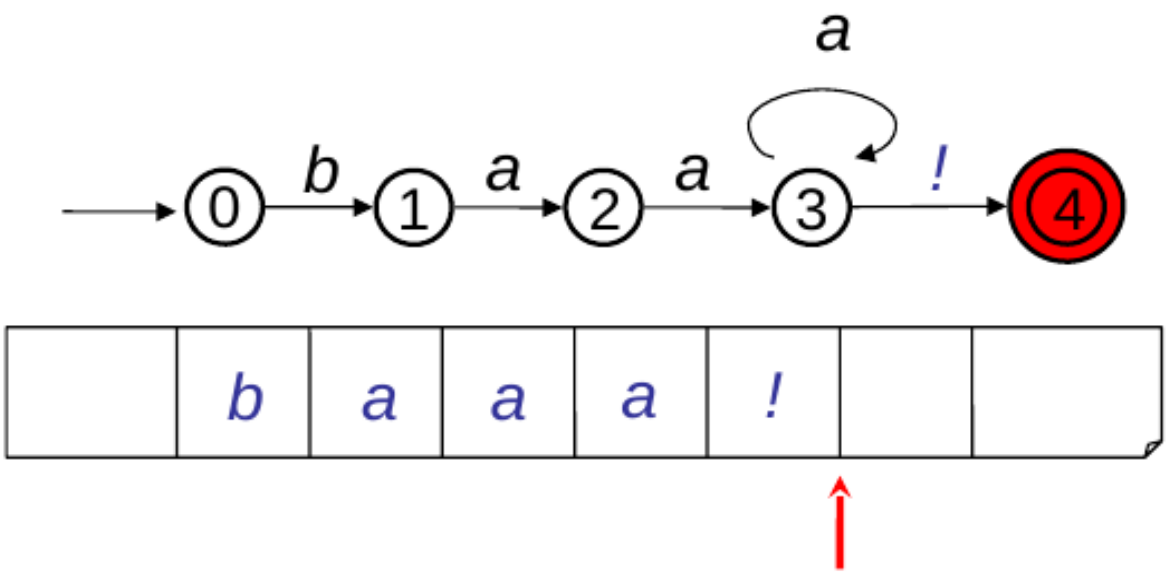
Erkennungsproblem

Akzeptanz

Eine Zeichenkette/Wort *w* wird von einem Automaten akzeptiert, falls sich der Automat nach dem (vollständigem) Lesen von *w* in einem Endzustand befindet.

Erkannte Sprache

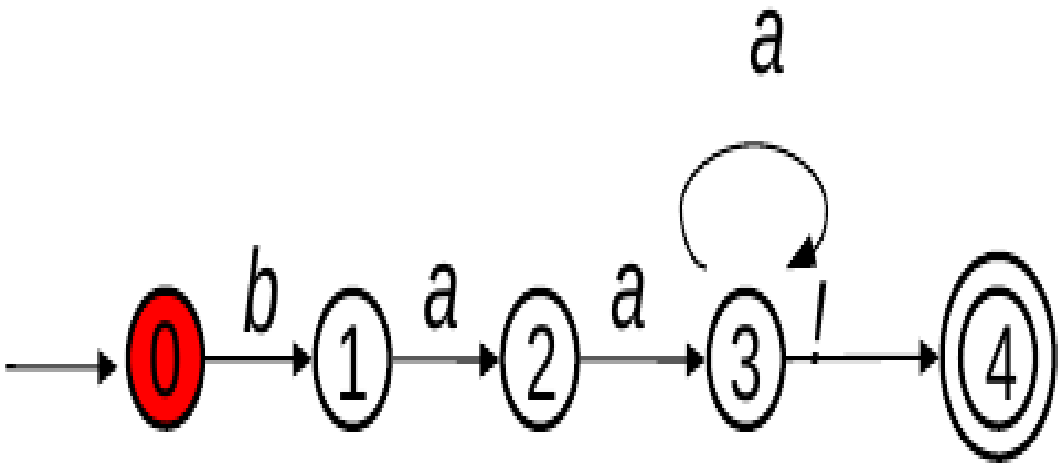
Alle Worte, die der Automat ausgehend vom Startzustand lesen kann, so dass nach dem Lesen ein Endzustand erreicht wird.



Endliche (deterministische) Automaten: Definition

Ein deterministischer endlicher Automat (DEA) ist ein 5-Tupel $A = \langle Q, \Sigma, q_0, F, \delta \rangle$, wobei

- Q: endliche, nichtleere Menge von Zuständen $q_0, \ldots q_n$
- Σ: (endliches, nichtleeres) Alphabet von Eingabesymbolen
- $q_0 \in Q$: Anfangszustand
- $F \subseteq Q$: Menge von Endzuständen
- δ: Übergangsfunktion $\delta : Q \times \Sigma \rightarrow Q$



Endliche (deterministische) Automaten: Definition

Ein deterministischer endlicher Automat (DEA) ist ein 5-Tupel $A = \langle Q, \Sigma, q_0, F, \delta \rangle$, wobei

- Q: endliche, nichtleere Menge von Zuständen $q_0, \ldots q_n$
- Σ: (endliches, nichtleeres) Alphabet von Eingabesymbolen
- $q_0 \in Q$: Anfangszustand
- $F \subseteq Q$: Menge von Endzuständen
- δ: Übergangsfunktion $\delta : Q \times \Sigma \to Q$

- δ kann als (partielle) Übergangstabelle dargestellt werden
- Deterministisch: In jedem Zustand gibt es für ein bestimmtes Eingabesymbol einen eindeutigen (oder keinen) Nachzustand $\to$ effiziente Verarbeitung

Nicht-deterministische endliche Automaten (ε-NEA)

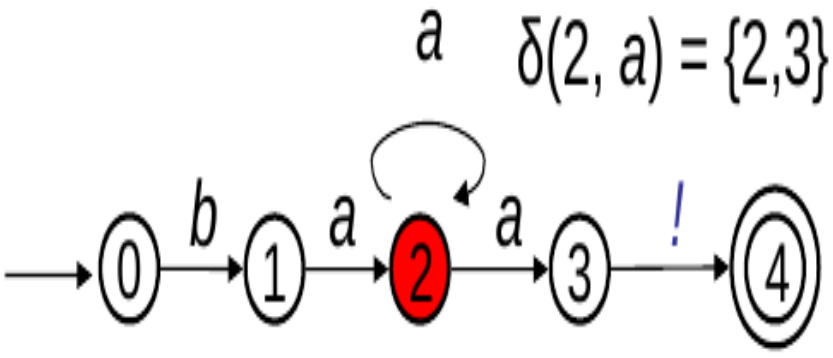
Ein nicht-deterministischer endlicher Automat (ε-NEA) ist ein 5-Tupel $A = \langle Q, \Sigma, q_0, F, \delta \rangle$, wobei

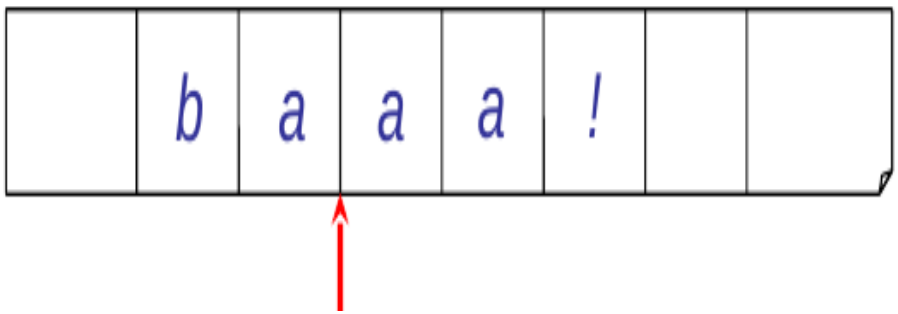
- Q: endliche, nichtleere Menge von Zuständen $q_0, \ldots q_n$
- Σ: (endliches, nichtleeres) Alphabet von Eingabesymbolen
- $q_0 \in Q$: Anfangszustand
- $F \subseteq Q$: Menge von Endzuständen
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to P(Q)$ (Potenzmenge von Q)

- Mehrere Möglichkeiten
- Springen ohne, dass man etwas gesehen hat (epsilon-Übergänge)

Hier kann es für einen Zustand bei einem bestimmtem Eingabesymbol unterschiedliche Nachzustände geben. Baum Durchlaufen des Automaten muss also eine Wahl getroffen werden.

Nichtdeterministische Automaten: Beispiele

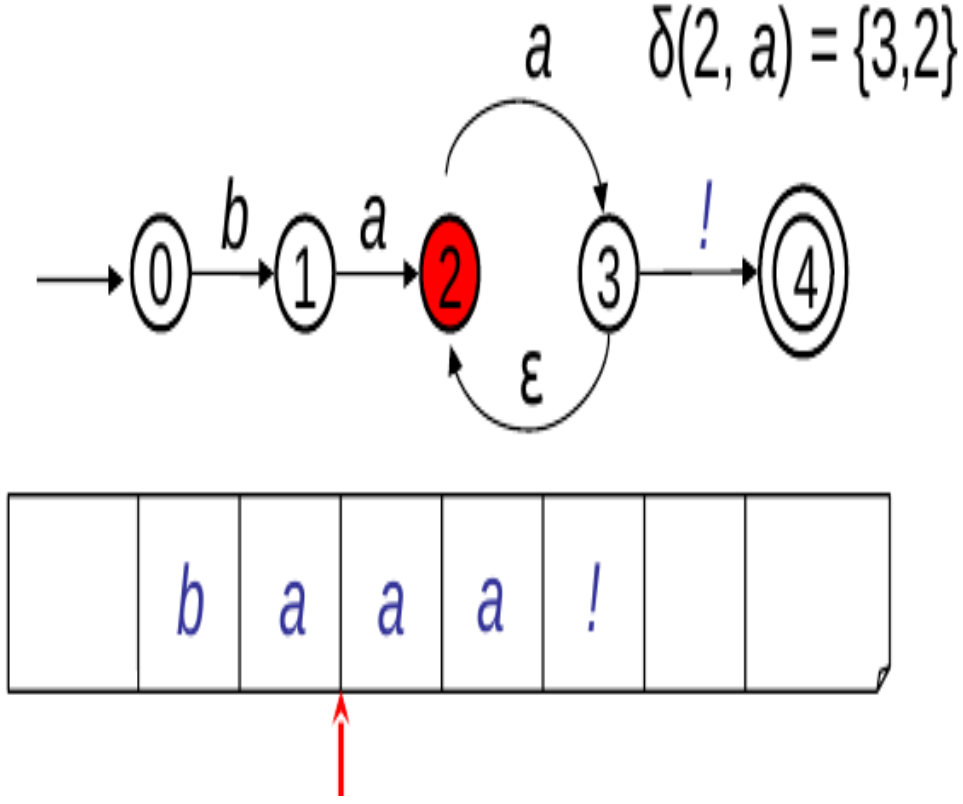




Nichtdeterminismus:
a) Alternative Übergänge für ein Eingabesymbol

Bild aus Jurafsky und Martin, 2nd edition

Nichtdeterministische Automaten: Beispiele



Nichtdeterminismus:
b) durch ε-Übergänge

Bild aus Jurafsky und Martin, 2nd edition

Nichtdeterministische Automaten: Beispiele

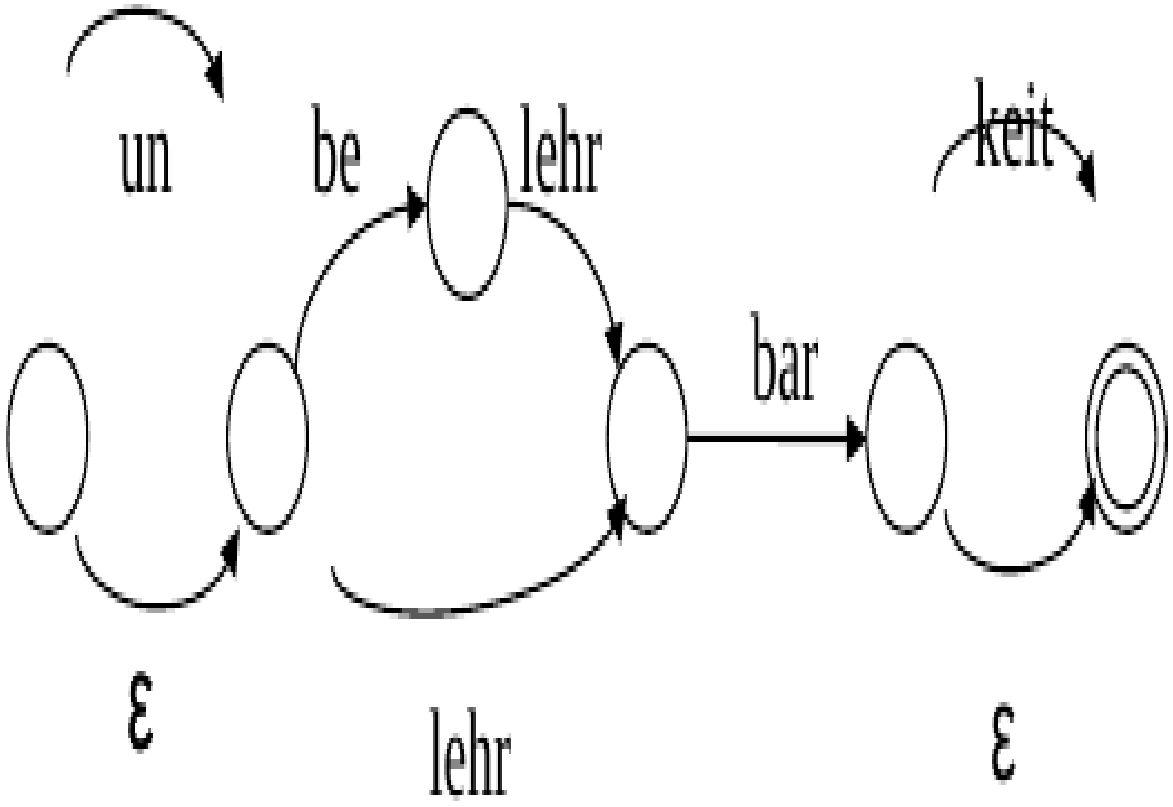
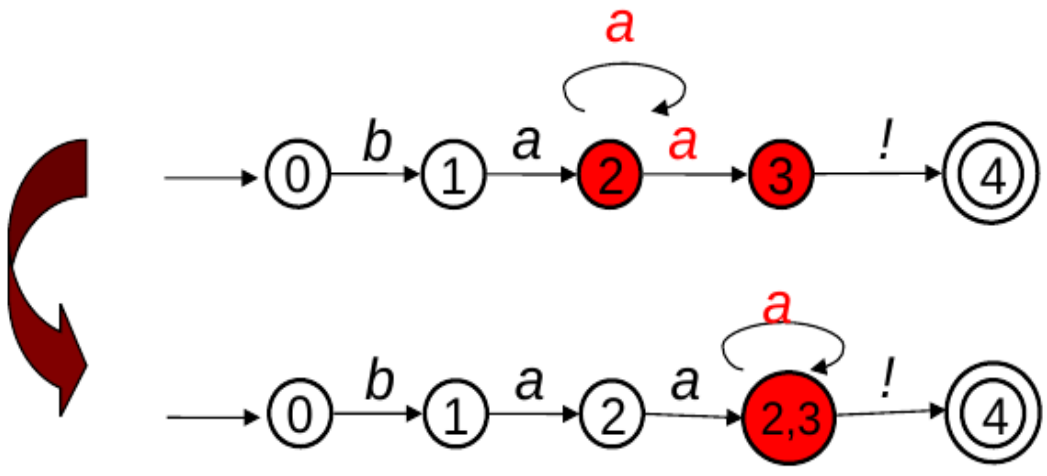


Bild aus Jurafsky und Martin, 2nd edition

Äquivalente Automaten

Äquivalenz

Zwei Automaten heissen äquivalent, wenn sie dieselbe Sprache akzeptieren



Insbesondere gilt: Zu jedem ε-NEA gibt es einen äquivalenten DEA, der dieselbe Sprache akzeptiert. (Powerset Construction mit Hüllenbildung; siehe optionale Zusatzfolien auf Moodle)

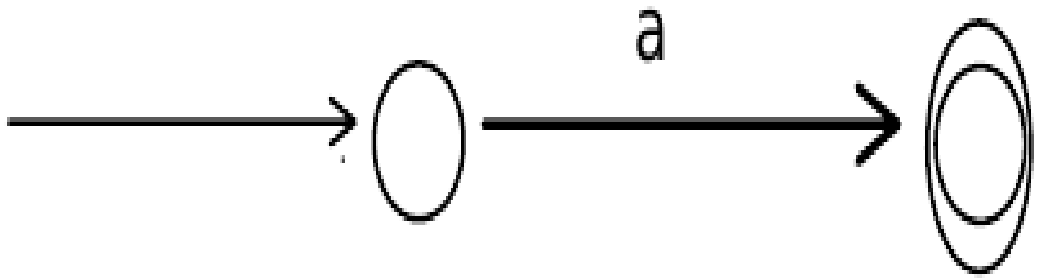
Für jeden regulären Ausdruck gibt es einen endlichen (nichtdeterministischen) Automaten: Thompson Algorithmus

Der Thompson-Algorithmus übersetzt einen regulären Ausdruck in einen (nicht-deterministischen) endlichen Automaten.

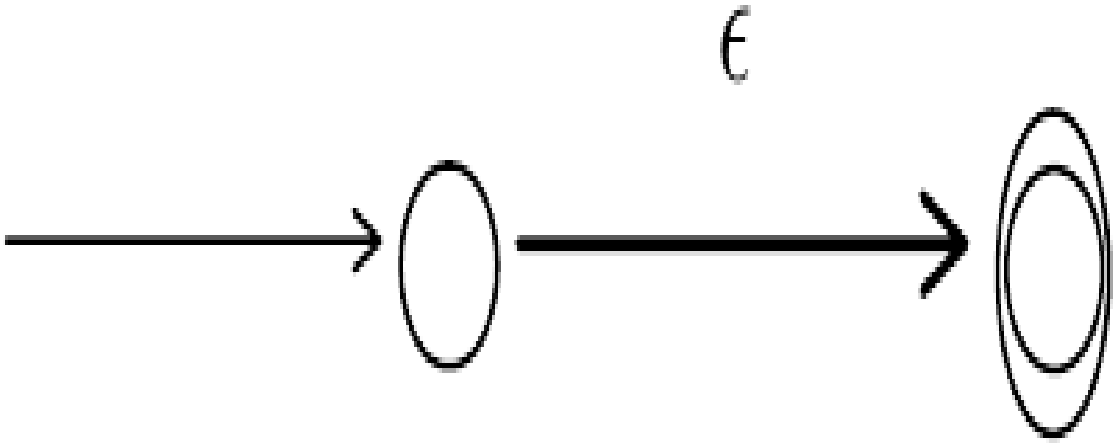
Beispiele. Dann Algorithmus. Dann komplexere Übung $(ab|c)*$.

Thompson I: Literale

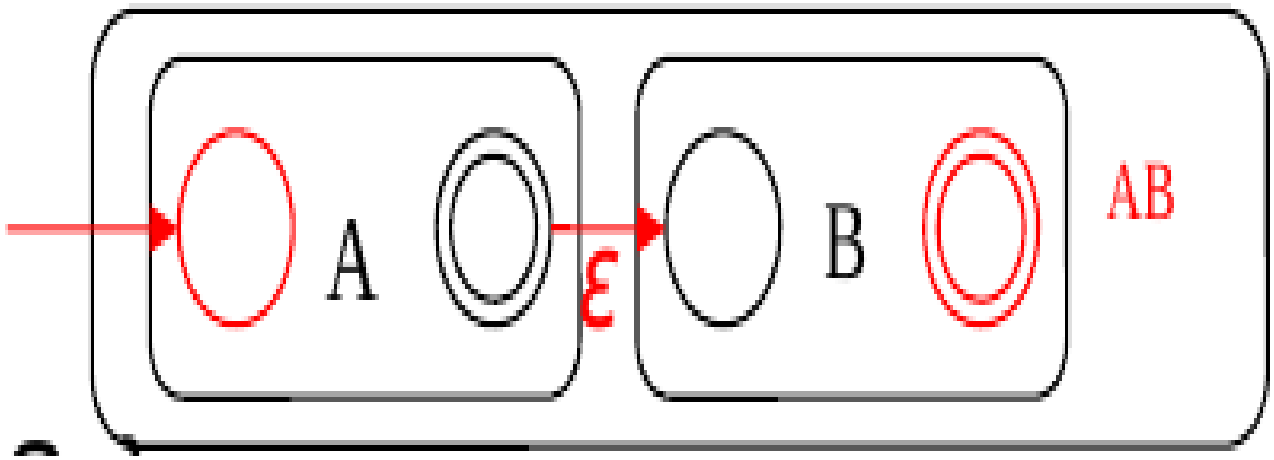
Ein FSA für a (mit $L(a) = \{a\}$)



Ein FSA für ε (mit $L(\varepsilon) = \{\varepsilon\}$.

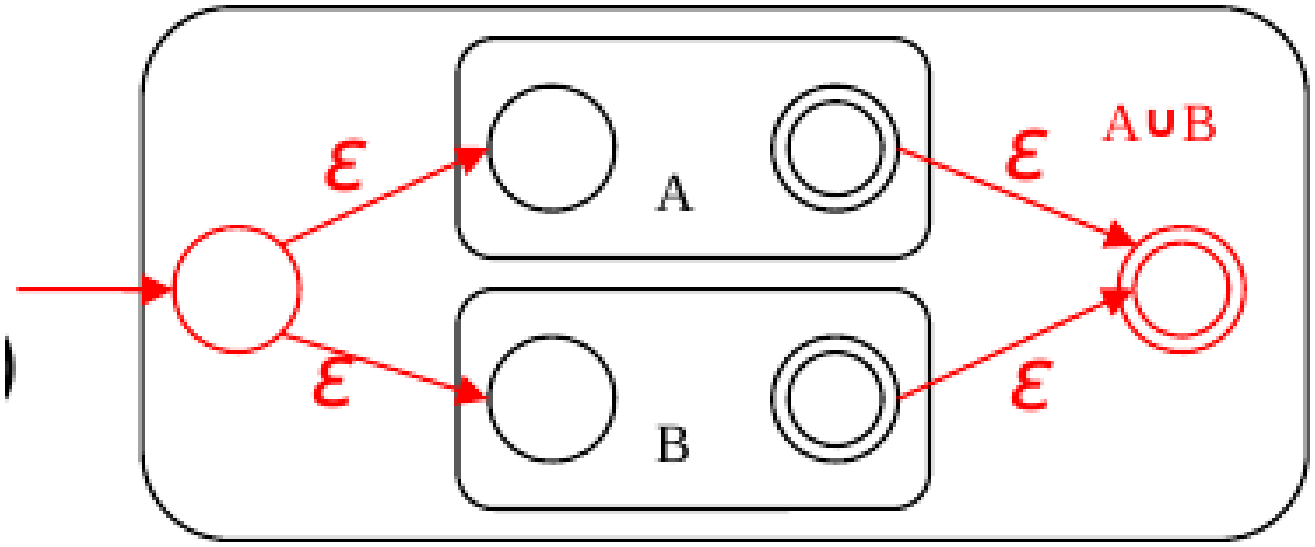


Thompson II: Konkatenation



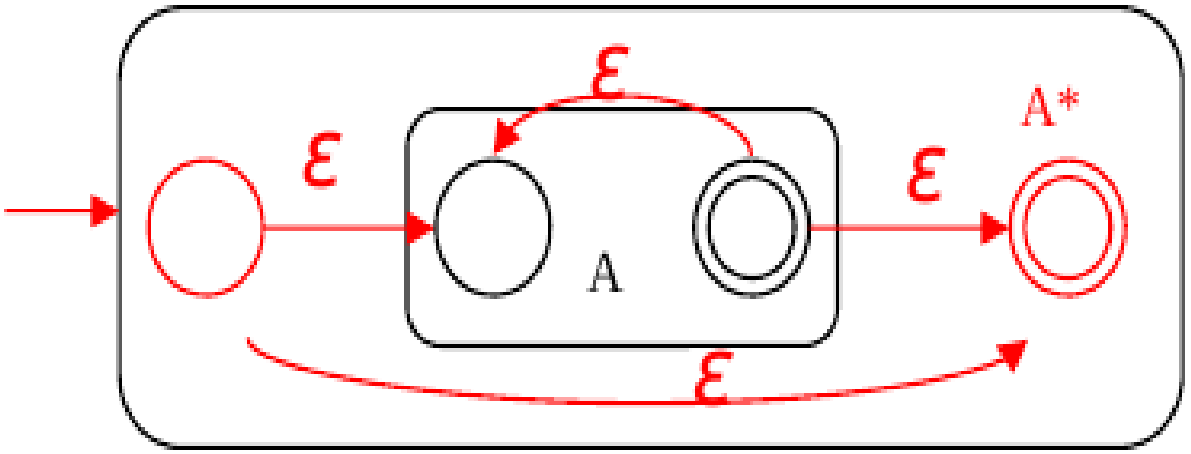
- Verbinde alle Endzustände von A mit dem Anfangszustand von B
- $S_{AB} = S_A, F_{AB} = F_B$
- $\delta_{AB} = \delta_A \cup \delta_B \cup \{(q_i, \varepsilon, S_B) | q_i \in F_A\}$

Thompson III: Vereinigung



- $S_{A \cup B} = q_0$ (neuer Startzustand)
- $F_{A \cup B} = q_{end}$ (neuer Endzustand)
- $\delta_{AB} = \delta_A \cup \delta_B \cup (q_0, \varepsilon, S_A) \cup (q_0, \varepsilon, S_B) \cup \{(q_z, \varepsilon, q_{end}) | q_z \in F_A \vee q_z \in F_B\}$

Thompson IV: Kleene-Star



- $S_{A^*} = q_0$ (neuer Startzustand)
- $F_{A^*} = q_{end}$ (neuer Endzustand)
- $\delta_{A*} = \delta_A \cup (q_0, \varepsilon, S_A) \cup (q_0, \varepsilon, q_{end}) \cup \{(q_i, \varepsilon, S_A) | q_i \in F_A\} \cup \{(q_z, \varepsilon, q_{end}) | q_z \in F_A\}$

Konvertierung RE $\rightarrow$ Automat: Beispiel

Regulärer Ausdruck: $(ab|c)*$

Konvertierung RE $\rightarrow$ Automat: Beispiel

Regulärer Ausdruck: $(ab|c)*$

Konvertierung RE $\rightarrow$ Automat: Beispiel

Regulärer Ausdruck: $(ab|c)*$

Nicht ganz genau nach Thompson. Was entspricht nicht genau dem Thompson-Algorithmus?

Äquivalenz REs und endliche Automaten

Der Thompson-Algorithmus verwandelt jeden regulären Ausdruck in einen EA.

Der Kleene-Algorithmus (Zustandseliminationsalgorithmus) verwandelt jeden DEA in einen regulären Ausdruck.

Siehe Hopcroft et al (2013): *Introduction to automata theory, languages and computation*

EAs und reguläre Ausdrücke/Sprachen sind also vollkommen äquivalent

Abschlusseigenschaften

Man kann zeigen: Sind L, L_1, L_2 reguläre Sprachen, dann sind auch

- die Vereinigung $L_1 \cup L_2$
- das Komplement $\bar{L} := \Sigma^* - L$
- die Intersektion $L_1 \cap L_2$
- die Differenz $L_1 - L_2$
- die Inversion (Spiegelworte) L_1^{rev}

reguläre Sprachen.

Schöne Folien zu Beweisen hierzu unter `https://www.cs.uni-potsdam.de/ti/lehre/downloads/TI-I/slides-2.5-anim.pdf`

Fazit Automaten

- Endliche Automaten sind aq̈uivalent zu regulären Sprachen. Manche Eigenschaften regulärer Sprachen lassen sich leichter mit Automaten zeigen.
- Deterministische Automaten: leichte automatische Verarbeitung, manchmal nicht leicht zu schreiben
- Nicht-deterministische Automaten lassen sich in deterministische umwandeln

ToDo und Literatur

- Übungsblatt 1/2
- Jurafsky und Martin, Kapitel 2.2 (zweite Edition!!!)
- Hopcroft et al (2013): *Introduction to automata theory, languages and computation*
- (optional): Ein kurzes Video zum Pumping Lemma. `https://studyflix.de/informatik/pumping-lemma-1445`

Endliche Automaten: Tools

- SFST (Stuttgarter FST-Library, Helmut Schmid) Open Source, UTF-8 kompatibel
  `http://www.cis.uni-muenchen.de/~schmid/tools/SFST/`
  sowie `/resources/platforms/sfst-1.3` `pysfst` Python-Wrapper
- Geertjan van Noord's finite-state tools
  `http://www.let.rug.nl/~vannoord/Fsa/`
- OpenFST: `www.openfst.org`;
  `https://github.com/tmbdev/pyopenfst` (Python- Wrapper)
- Xerox finite-state tools: XFST Tools (provided with Beesley and Karttunen: Finite-State Morphology, CSLI Publications und: SCL Resources),
  `http://www.stanford.edu/~laurik/fsmbook/home.html`
  sowie `/resources/platforms/xfst-8.1.4`

Formale Grammatiken: Definition

Eine formale (generative) Grammatik $G = \langle NT, \Sigma, R, S \rangle$ besteht aus

- einer endlichen Menge NT von Nicht-Terminalen (nicht überlappend mit Zeichenketten, die man aus G generiert)
- einer endlichen Menge von Terminalen Σ (nicht überlappend mit NT)
- Einem Startsymbol $S \in NT$
- Einer endlichen Menge von Produktionsregeln R von der Form

$$(\Sigma \cup NT)^* NT (\Sigma \cup NT)^* \rightarrow (\Sigma \cup NT)*$$

Die von G generierten Wörter L(G) sind die Ketten von Terminalen, die man vom Startsymbol ausgehend erreichen kann.

Formale Grammatiken: Beispiel

- $NT = \{S, A\}$
- $\Sigma = \{a, b\}$
- R :

$$\begin{aligned} \mathrm{S} &\rightarrow \mathrm{A} \\ \mathrm{A} &\rightarrow \mathrm{a\ A\ b} \\ \mathrm{A} &\rightarrow \mathrm{a\ b} \end{aligned}$$

erzeugt die Sprache $L(G) = a^n b^n$ mit $n \geq 1$

Erweiterte (rechts)reguläre Grammatiken Definition

Auch genannt Typ-3 Grammatiken. Folgende Einschränkungen für die Regeln R:

- Links darf nur ein Nicht-Terminal stehen
- Rechts entweder
  - ε ($NT \rightarrow \varepsilon$)
  - eine Sequenz von Terminalen ($NT \rightarrow w$ mit $w \in \Sigma^*$)
  - eine Sequenz von Terminalen gefolgt von einem Nicht-Terminal ($NT \rightarrow wB$ mit $w \in \Sigma^*$ und $B \in NT$)

Äquivalent werden linksreguläre Grammatiken definiert.

Ist das obige Beispiel regulär?

- $NT = \{S, A\}$
- $\Sigma = \{a, b\}$
- R:

$$\mathrm{S} \rightarrow \mathrm{A}$$
$$\mathrm{A} \rightarrow \mathrm{a\ A\ b}$$
$$\mathrm{A} \rightarrow \mathrm{a\ b}$$

Nein! Dies ist aber i.a. kein Beweis, dass die erzeugte Sprache nicht-regulär ist; vielleicht haben wir nur die Regeln nicht geschickt geschrieben. Im obigem Beispiel ist die Sprache aber wirklich nicht regulär.

Ist das obige Beispiel regulär?

- $NT = \{S, A\}$
- $\Sigma = \{a, b\}$
- R :

$$\begin{aligned} \mathrm{S} &\rightarrow \mathrm{A} \\ \mathrm{A} &\rightarrow \mathrm{a\ A\ b} \\ \mathrm{A} &\rightarrow \mathrm{a\ b} \end{aligned}$$

Nein! Dies ist aber i.a. kein Beweis, dass die erzeugte Sprache nicht-regulär ist; vielleicht haben wir nur die Regeln nicht geschickt geschrieben. Im obigem Beispiel ist die Sprache aber wirklich nicht regulär.

Eine reguläre Grammatik

- $NT = \{S, A, B\}$
- $\Sigma = \{a, b\}$
- R:

$$\begin{aligned} \mathrm{S} &\to \mathrm{aA} \\ \mathrm{A} &\to \mathrm{aA} \\ \mathrm{A} &\to \mathrm{bB} \\ \mathrm{B} &\to \mathrm{bB} \\ \mathrm{B} &\to \varepsilon \end{aligned}$$

erzeugt welche Sprache? (5 Minuten Zeit)

erzeugt die Sprache $a^n b^m$ mit $n, m \geq 1$

Eine reguläre Grammatik

- $NT = \{S, A, B\}$
- $\Sigma = \{a, b\}$
- R:

$$\begin{aligned} \mathrm{S} &\to \mathrm{aA} \\ \mathrm{A} &\to \mathrm{aA} \\ \mathrm{A} &\to \mathrm{bB} \\ \mathrm{B} &\to \mathrm{bB} \\ \mathrm{B} &\to \varepsilon \end{aligned}$$

erzeugt welche Sprache? (5 Minuten Zeit)
erzeugt die Sprache $a^n b^m$ mit $n, m \geq 1$

Äquivalenz (deterministische) endliche Automaten und rechtsreguläre Grammatiken I

Wenn M ein deterministischer endlicher Automat ist, dann existiert eine reguläre Grammatik G, so dass $L(G) = L(M)$

Konstruktion:
Sei $M = \langle Q, \Sigma, q_0, F, \delta \rangle$ ein DEA. Wir definieren $G = \langle NT, \Sigma_G, S, R \rangle$ wie folgt.

- $NT := Q$
- $\Sigma_G := \Sigma$
- Für jeden Übergang $\delta(K_1, a) = K_2$ definieren wir eine Produktionsregel $K_1 \to aK_2$ (wobei $K_1, K_2 \in Q, a \in \Sigma$)
- $S := q_0$
- Für alle Endzustände $K \in F$, definieren wir eine Produktionsregel $K \to \varepsilon$

Äquivalenz (deterministische) endliche Automaten und rechtsreguläre Grammatiken II

Für jede reguläre Grammatik G, gibt es einen deterministischen endlichen Automaten M so dass L(M) = L(G)

Konstruktion:
Sei $G = \langle NT, \Sigma_G, S, R \rangle$ eine reguläre Grammatik. Wir definieren $M = \langle Q, \Sigma, q_0, F, \delta \rangle$ wie folgt:

- $Q := NT$
- $\Sigma := \Sigma_G$
- Für jede Produktionsregel $K_1 \rightarrow aK_2$ führen wir einen Übergang $\delta(K_1, a) = K_2$ ein
- $q_0 := S$
- F ist die Menge aller Nichtterminale X mit der Produktionsregel $X \rightarrow \varepsilon$

Finale Zusammenfassung und Äquivalenzen

- Endliche (deterministische und nicht-deterministische) Automaten erkennen Wörter einer Sprache
- Reguläre Ausdrücke beschreiben Struktur der Wörter
- reguläre (Typ-3) Grammatiken erzeugen Wörter einer regulären Sprache

Finale Zusammenfassung und Äquivalenzen

Alle Modelle sind äquivalent.

- $\varepsilon - NEA \rightarrow$ DEA: Powersetconstruction (optional)
- DEA $\rightarrow \varepsilon - NEA$: folgt trivialerweise aus Def
- DEA $\rightarrow$ reg. Grammatik: Verwandle Übergangsfunktion in Regeln
- reg. Grammatik $\rightarrow$ DEA: Verwandle Regeln in Übergangsfunktion
- reg. Ausdrücke $\rightarrow$ NEA: Thompson-Algorithmus
- DEA $\rightarrow$ reg. Ausdrücke: Kleene-Algorithmus (optional)

Finale Zusammenfassung und Äquivalenzen

Alle Modelle sind äquivalent.

- $\varepsilon - NEA \rightarrow$ DEA: Powersetconstruction (optional)
- DEA $\rightarrow \varepsilon - NEA$: folgt trivialerweise aus Def
- DEA $\rightarrow$ reg. Grammatik: Verwandle Übergangsfunktion in Regeln
- reg. Grammatik $\rightarrow$ DEA: Verwandle Regeln in Übergangsfunktion
- reg. Ausdrücke $\rightarrow$ NEA: Thompson-Algorithmus
- DEA $\rightarrow$ reg. Ausdrücke: Kleene-Algorithmus (optional)

Finale Zusammenfassung und Äquivalenzen

Alle Modelle sind äquivalent.

- $\varepsilon - NEA \rightarrow$ DEA: Powersetconstruction (optional)
- DEA $\rightarrow \varepsilon - NEA$: folgt trivialerweise aus Def
- DEA $\rightarrow$ reg. Grammatik: Verwandle Übergangsfunktion in Regeln
- reg. Grammatik $\rightarrow$ DEA: Verwandle Regeln in Übergangsfunktion
- reg. Ausdrücke $\rightarrow$ NEA: Thompson-Algorithmus
- DEA $\rightarrow$ reg. Ausdrücke: Kleene-Algorithmus (optional)