# Einführung in die Computerlinguistik
# Automaten

Alexander Fraser / Robert Zangenfeind

Center for Information and Language Processing

2019-12-09

Die Grundfassung dieses Foliensatzes wurde von Prof. Dr. Stefan Evert erstellt, und von Prof. Dr. Hinrich Schütze erweitert.

Fehler und Mängel sind ausschließlich meine Verantwortung.

# Outline

# Outline

# Motivation: Morphologie

# Motivation: Morphologie

- Formen von “klein”:
  klein, kleine, kleinem, kleineren, kleiner, kleines,
  kleiner, kleinere, kleinerem, kleinereren, kleinerer, kleineres
  kleinst, kleinste, kleinstem, kleinsteren, kleinster, kleinstes

# Motivation: Morphologie

- Formen von “klein”:
  klein, kleine, kleinem, kleineren, kleiner, kleines,
  kleiner, kleinere, kleinerem, kleinereren, kleinerer, kleineres
  kleinst, kleinste, kleinstem, kleinsteren, kleinster, kleinstes
- Erkennung: Ist eine gegebene Form eine Form von “klein”?

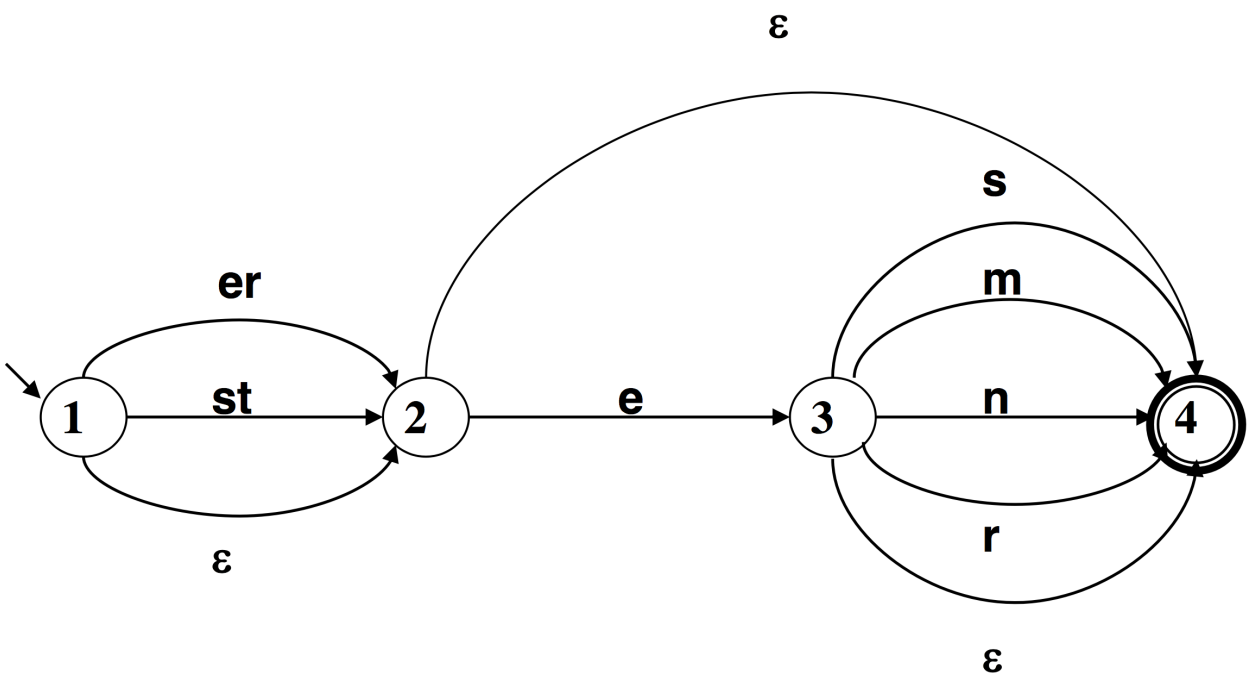
# Motivation: Morphologie

- Formen von “klein”:
  klein, kleine, kleinem, kleineren, kleiner, kleines,
  kleiner, kleinere, kleinerem, kleinereren, kleinerer, kleineres
  kleinst, kleinste, kleinstem, kleinsteren, kleinster, kleinstes
- Erkennung: Ist eine gegebene Form eine Form von “klein”?
- Generierung: Generiere für eine gegebene Spezifikation (z.B., stark, Nominativ, Singular, Maskulinum) eine Form (“kleiner”)

# Motivation: Morphologie

- Formen von “klein”:
  klein, kleine, kleinem, kleineren, kleiner, kleines,
  kleiner, kleinere, kleinerem, kleinereren, kleinerer, kleineres
  kleinst, kleinste, kleinstem, kleinsteren, kleinster, kleinstes
- Erkennung: Ist eine gegebene Form eine Form von “klein”?
- Generierung: Generiere für eine gegebene Spezifikation (z.B., stark, Nominativ, Singular, Maskulinum) eine Form (“kleiner”)
- Beide Aufgaben können mit Automaten gelöst werden.

# Automat für Adjektive (Buchstabenebene)



# Motivation: Syntax

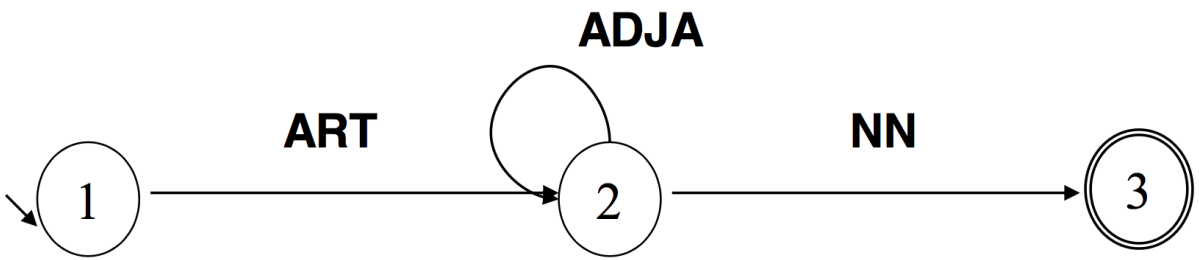
# Motivation: Syntax

- Ein wichtiger Teil der Syntax natürlicher Sprache kann durch Automaten dargestellt werden.

# Motivation: Syntax

- Ein wichtiger Teil der Syntax natürlicher Sprache kann durch Automaten dargestellt werden.
- Beispiel:
  - das Auto
  - das rote Auto
  - das alte rote Auto
  - das schnelle alte rote Auto
  - das kaputte schnelle alte rote Auto

# Automat für Nominalphrase (Lexemformebene)



# Praktische Motivation: Reguläre Ausdrücke

# Praktische Motivation: Reguläre Ausdrücke

- Die meisten Textverarbeitungssysteme verwenden Reguläre Ausdrücke.

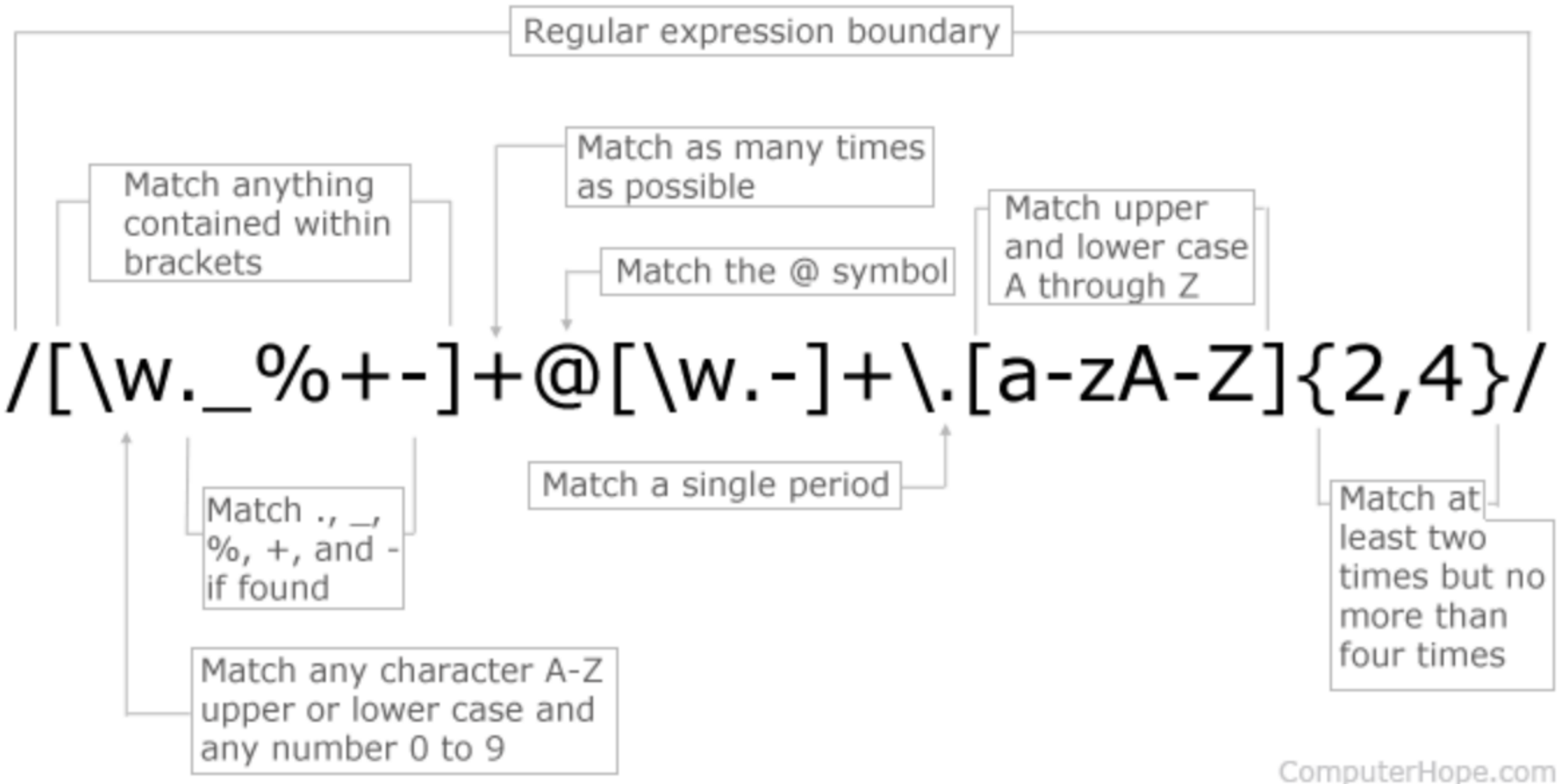
# Praktische Motivation: Reguläre Ausdrücke

- Die meisten Textverarbeitungssysteme verwenden Reguläre Ausdrücke.
- Vielfältige Verwendungsmöglichkeiten
- Entfernen von Markup (HTML etc.)
- Extraktion von Information
- Automatisches Ersetzen
- ...

# Praktische Motivation: Reguläre Ausdrücke

- Die meisten Textverarbeitungssysteme verwenden Reguläre Ausdrücke.
- Vielfältige Verwendungsmöglichkeiten
- Entfernen von Markup (HTML etc.)
- Extraktion von Information
- Automatisches Ersetzen
- ...
- Reguläre Ausdrücke ähneln einer eigenen Programmiersprache (sind aber auch Teil fast jeder Programmiersprache).

# Regulärer Ausdruck für Email-Adressen



# Theoretische Motivation: Was ist Sprache?

# Theoretische Motivation: Was ist Sprache?

- Ist Sprache endlich?

# Theoretische Motivation: Was ist Sprache?

- Ist Sprache endlich?
- Ist das Vokabular einer Sprache begrenzt?

# Theoretische Motivation: Was ist Sprache?

- Ist Sprache endlich?
- Ist das Vokabular einer Sprache begrenzt?
- Können Sätze beliebig lang sein?

# Theoretische Motivation: Was ist Sprache?

- Ist Sprache endlich?
- Ist das Vokabular einer Sprache begrenzt?
- Können Sätze beliebig lang sein?
- Ist Sprache rekursiv?

# Theoretische Motivation: Was ist Sprache?

- Ist Sprache endlich?
- Ist das Vokabular einer Sprache begrenzt?
- Können Sätze beliebig lang sein?
- Ist Sprache rekursiv?
- Ist Sprache hierarchisch?

# Theoretische Motivation: Was ist Sprache?

- Ist Sprache endlich?
- Ist das Vokabular einer Sprache begrenzt?
- Können Sätze beliebig lang sein?
- Ist Sprache rekursiv?
- Ist Sprache hierarchisch?
- Die Theorie der Formalen Sprachen hilft bei der Präzisierung und Beantwortung dieser Fragen.

# Theoretische Motivation: Was ist Sprache?

- Ist Sprache endlich?
- Ist das Vokabular einer Sprache begrenzt?
- Können Sätze beliebig lang sein?
- Ist Sprache rekursiv?
- Ist Sprache hierarchisch?
- Die Theorie der Formalen Sprachen hilft bei der Präzisierung und Beantwortung dieser Fragen.
- Zunächst nur Form, nicht Funktion/Bedeutung.

# Theoretische Motivation: Was ist Sprache?

- Ist Sprache endlich?
- Ist das Vokabular einer Sprache begrenzt?
- Können Sätze beliebig lang sein?
- Ist Sprache rekursiv?
- Ist Sprache hierarchisch?
- Die Theorie der Formalen Sprachen hilft bei der Präzisierung und Beantwortung dieser Fragen.
- Zunächst nur Form, nicht Funktion/Bedeutung.
- Form: Kann man das so sagen?

# Theoretische Motivation: Was ist Sprache?

- Ist Sprache endlich?
- Ist das Vokabular einer Sprache begrenzt?
- Können Sätze beliebig lang sein?
- Ist Sprache rekursiv?
- Ist Sprache hierarchisch?
- Die Theorie der Formalen Sprachen hilft bei der Präzisierung und Beantwortung dieser Fragen.
- Zunächst nur Form, nicht Funktion/Bedeutung.
- Form: Kann man das so sagen?
- Funktion: Was ist die Bedeutung des Gesagten?

# Motivation für Reguläre Sprachen / Automaten

- Morphologie
- Syntax
- Praktische Motivation: Reguläre Ausdrücke
- Theoretische Motivation: Was ist Sprache?

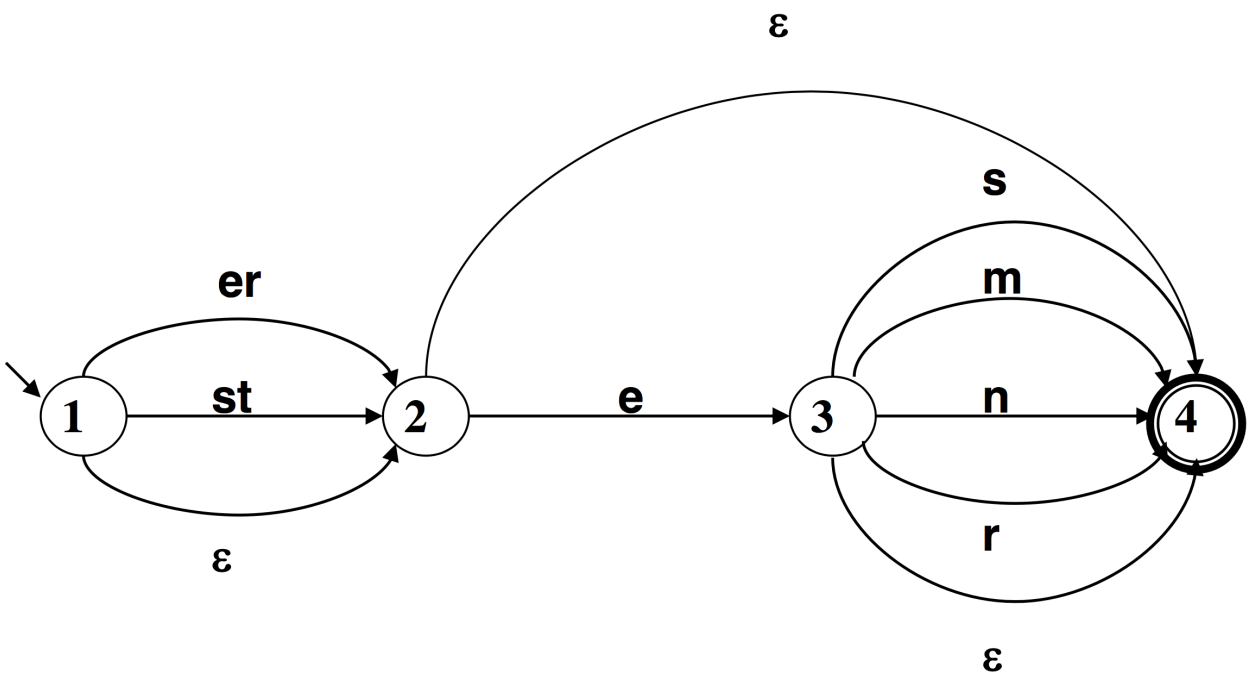
## Umfrage

Für wen sind neu:

- Automaten
- reguläre Ausdrücke
- formale Sprachen
  (einschließlich reguläre Sprachen)

## Beispiel / Übung

Denken Sie sich eine Endung des Adjektivs “klein” aus und finden Sie den Pfad, mit dem der Automat diese Endung erkennt/akzeptiert. Epsilon = “Nullübergang”. Endzustand: Doppelkreis.



# Automatendiagramme

# Automatendiagramme

- Bogen, der mit einem Symbol annotiert ist:
  Der Automat liest das Symbol und geht zum nächsten Zustand.

# Automatendiagramme

- Bogen, der mit einem Symbol annotiert ist:
  Der Automat liest das Symbol und geht zum nächsten Zustand.
- Bogen, der mit einem Epsilon annotiert ist:
  Der Automat liest nichts und geht zum nächsten Zustand.

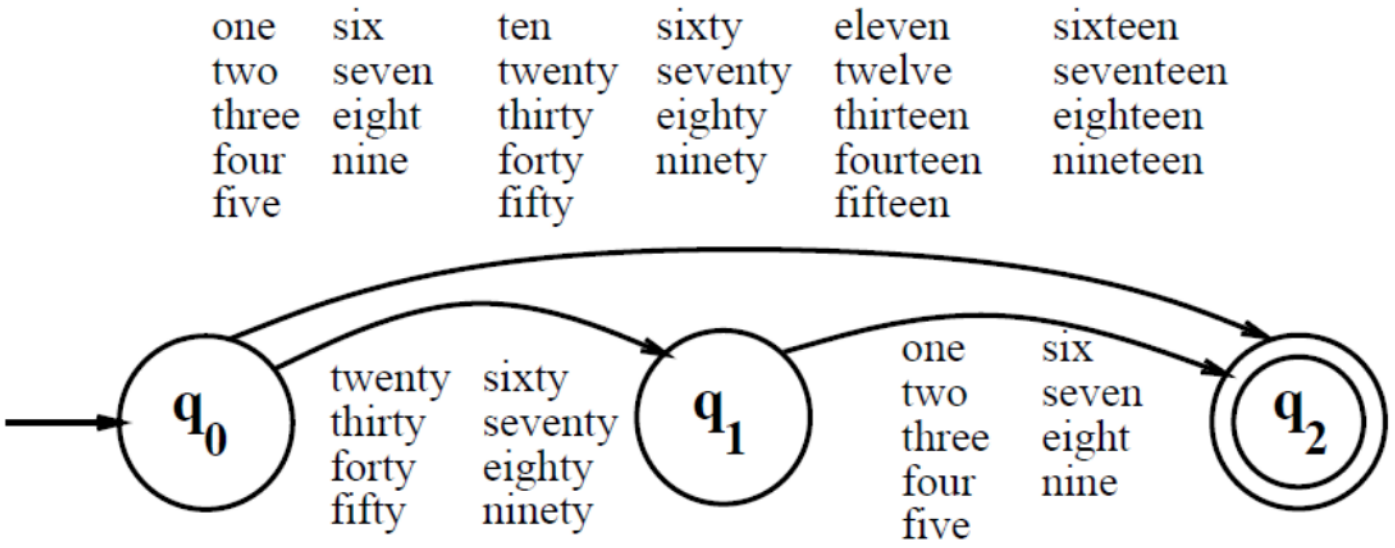
# Automatendiagramme

- Bogen, der mit einem Symbol annotiert ist:
  Der Automat liest das Symbol und geht zum nächsten Zustand.
- Bogen, der mit einem Epsilon annotiert ist:
  Der Automat liest nichts und geht zum nächsten Zustand.
- Doppelkreis: Endzustand.
  Der Automat "akzeptiert" einen String, wenn er nach Lesen des Strings in einem Endzustand ist.

# Automatendiagramme

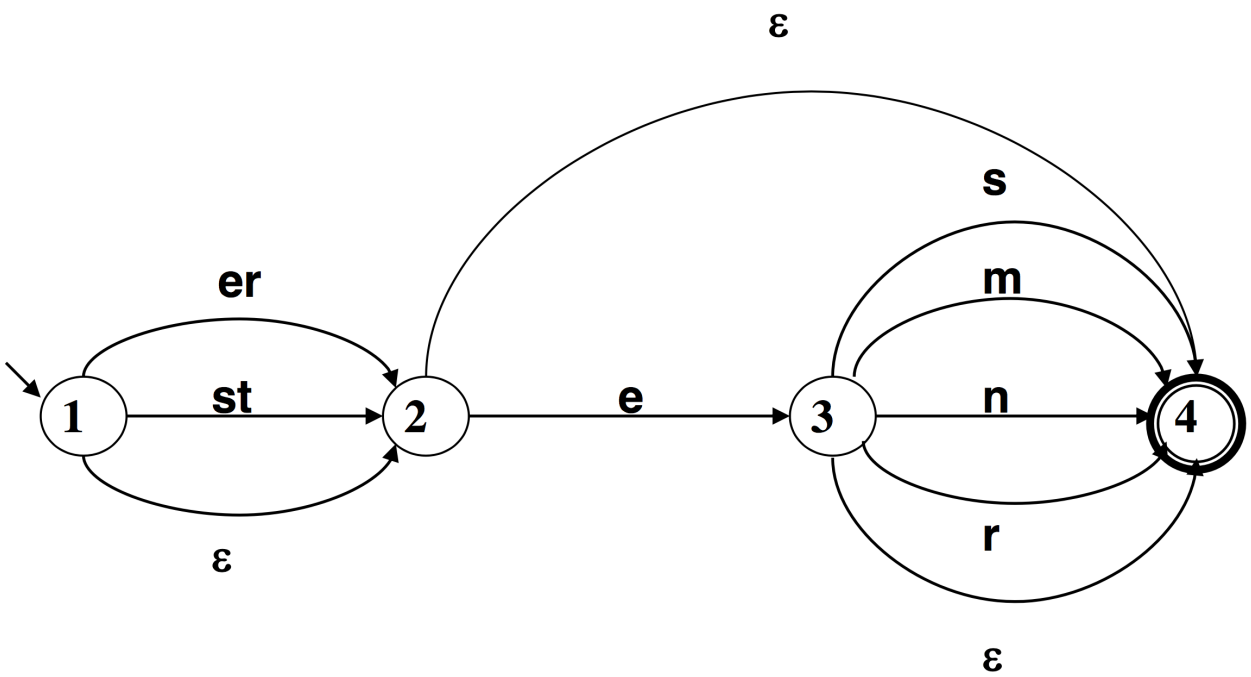
- Bogen, der mit einem Symbol annotiert ist:
  Der Automat liest das Symbol und geht zum nächsten Zustand.
- Bogen, der mit einem Epsilon annotiert ist:
  Der Automat liest nichts und geht zum nächsten Zustand.
- Doppelkreis: Endzustand.
  Der Automat "akzeptiert" einen String, wenn er nach Lesen des Strings in einem Endzustand ist.
- Ein einfacher Pfeil (der nicht bei einem Zustand beginnt) zeigt auf den Startzustand.

# Beispiel


one
two
three
four
five
six
seven
eight
nine
ten
twenty
thirty
forty
fifty
sixty
seventy
eighty
ninety
eleven
twelve
thirteen
fourteen
fifteen
sixteen
seventeen
eighteen
nineteen
$q_0$
twenty
thirty
forty
fifty
sixty
seventy
eighty
ninety
$q_1$
one
two
three
four
five
six
seven
eight
nine
$q_2$

## Beispiel / Übung

Denken Sie sich eine Endung des Adjektivs “klein” aus und finden Sie den Pfad, mit dem der Automat diese Endung erkennt/akzeptiert. Epsilon = “Nullübergang”. Endzustand: Doppelkreis.



# Outline

# Fahrplan

- Reguläre Sprachen
- Reguläre Ausdrücke
- deterministische Automaten
- nichtdeterministische Automaten
- Reguläre Sprachen
  = Reguläre Ausdrücke
  = deterministische Automaten
  = nichtdeterministische Automaten

## Caveat

In dieser Vorlesung gibt es einiges, das Sie als Hintergrund wissen sollten, das aber nicht abgefragt wird. Versuchen Sie einfach, möglichst viel zu verstehen, aber vollständiges Verständnis ist bei vielem nicht nötig.

# Grundbegriffe

Das Alphabet $\Sigma$ ist eine endliche Menge, deren Elemente Symbole oder Zeichen heißen. Wir verwenden für Zeichen üblicherweise die Buchstaben $a, b, c, \ldots \in \Sigma$.

Wir können die Zeichen aus einem gegebenen Alphabet $\Sigma$ zu endlichen Symbolfolgen oder Wörtern zusammensetzen, wobei Wiederholungen erlaubt sind. Wir bezeichnen Wörter mit $u, v, w, \ldots$ oder griechischen Buchstaben.

Die Länge $|w|$ eines Wortes $w$ ist die Anzahl der Zeichen, aus denen es besteht. Eine besondere Stellung nimmt dabei das leere Wort $\epsilon$ ein, das als einziges die Länge $|\epsilon| = 0$ hat.

Die Menge aller Wörter über einem Alphabet $\Sigma$ (einschließlich des leeren Wortes) wird mit $\Sigma^*$ bezeichnet.

# Buchstabenebene vs. Lexemformebene

# Buchstabenebene vs. Lexemformebene

## Buchstabenebene

Zeichen sind Buchstaben.
Ein Wort der regulären Sprache ist ein Wort der natürlichen Sprache.

# Buchstabenebene vs. Lexemformebene

## Buchstabenebene

Zeichen sind Buchstaben.
Ein Wort der regulären Sprache ist ein Wort der natürlichen Sprache.

## Lexemformebene

Zeichen sind Lexemformen (natürlichsprachliche Wörter).
Ein Wort der regulären Sprache ist eine Phrase oder ein Satz der natürlichen Sprache.

# Indexnotation

Formal läßt sich ein Wort $w \in \Sigma^*$ der Länge $|w| = n$ als Abbildung $w : \{1, \ldots, n\} \to \Sigma$ auffassen.

Wir verwenden die Notation $w(i)$ für das $i$-te Zeichen eines Wortes: $w(1)$ bezeichnet also das erste, $w(n)$ das letzte Zeichen von $w$. Alternativ: $w_i$, $w_1$, $w_n$

# Verkettung = Konkatenation

Die Verkettung oder Konkatenation zweier Wörter $v, w \in \Sigma^*$ wird $v \circ w$ oder kurz $vw$ geschrieben. $v \circ w \in \Sigma^*$ entsteht durch "Aneinanderhängen" der Wörter $v$ und $w$. Wir können jedes Wort als Verkettung seiner Zeichen auffassen:
$w = w(1) \circ w(2) \circ \cdots \circ w(n)$ (mit $n = |w|$).

Das leere Wort $\epsilon$ wird als neutrales Element (bezüglich der Konkatenation) bezeichnet, da $\epsilon \circ w = w \circ \epsilon = w$ für alle $w \in \Sigma^*$ gilt.

Weiterhin ist stets $|v \circ w| = |v| + |w|$.

# Potenz, Umkehrung, Palindrom

Die $i$-te Potenz $w^i$ eines Wortes $w$ bezeichnet dessen $i$-fache Wiederholung. Formal läßt sich dieser Sachverhalt in eine rekursive Definition fassen: $w^0 := \epsilon$ und $w^{i+1} := w^i \circ w$ für $i \in \mathbb{N}_0$.

Die Umkehrung (oder Spiegelung) $w^R$ von $w = w(1)w(2)\cdots w(n)$ ist definiert als $w^R := w(n)w(n-1)\cdots w(1)$ (z.B. ist $(\text{stefan})^R = \text{nafets}$).

Ein Palindrom ist ein Wort $w$, für das $w^R = w$ gilt.

Einfache Eigenschaften der Umkehrung sind $(v \circ w)^R = w^R \circ v^R$ und $\left|w^R\right| = |w|$.

# Präfix, Suffix

Gibt es zu $v, w \in \Sigma^*$ ein Wort $u \in \Sigma^*$ mit $w = v \circ u$ (bzw. $w = u \circ v$), so heißt $v$ Präfix (bzw. Suffix) von $w$. Ein Präfix $v$ von $w$ wird abkürzend mit $v \leq w$ gekennzeichnet.

# Kleenesche Hülle

Der Abschluß $A^*$ (oder Abschluß unter Konkatenation) einer Wortmenge $A \subseteq \Sigma^*$ ist die kleinste Teilmenge von $\Sigma^*$, die folgende Bedingungen erfüllt:

- $A \subseteq A^*$
- $\epsilon \in A^*$
- $v, w \in A^* \Rightarrow v \circ w \in A^*$

Der Abschluß einer Wortmenge wird als Kleenesche Hülle bezeichnet, der Operator $^*$ als Kleene Star. Anschaulich besteht $A^*$ aus allen möglichen Verkettungen von beliebig vielen Wörtern aus $A$. Es gilt folglich

$$A^* = \{w_1 \circ w_2 \circ \cdots \circ w_n \mid n \in \mathbb{N} \text{ und } w_1, \ldots, w_n \in A\} \cup \{\epsilon\}$$

# Definition: Formale Sprache

Eine formale Sprache über dem Alphabet $\Sigma$ ist eine (beliebige) Teilmenge $A \subseteq \Sigma^*$.

Wir unterscheiden zwischen endlichen und unendlichen Sprachen, je nachdem ob $|A|$ endlich oder unendlich ist.

Da das Alphabet $\Sigma$ endlich ist, ist $\Sigma^*$ und somit jede unendliche formale Sprache abzählbar unendlich.

# Mengenoperationen

Auf formale Sprachen $A, B \subseteq \Sigma^*$ lassen sich die folgenden Mengenoperationen anwenden:
Vereinigung $A \cup B$,
Durchschnitt $A \cap B$,
Differenz $A \setminus B$,
sowie das Komplement $\mathcal{C}A = \Sigma^* \setminus A$
sind alle wieder formale Sprachen über $\Sigma$.

Zusätzliche Operationen für formale Sprachen sind Kleenesche Hülle $A^*$ und die Verkettung $A \circ B := \{v \circ w \mid v \in A \wedge w \in B\}$.
Formal ist $A \circ B$ als Teilmenge von $\Sigma^*$ definiert:

$$A \circ B := \{w \in \Sigma^* \mid \exists u \in A \, \exists v \in B \; : w = u \circ v\}.$$

# Sprachklassen

Eine Sprachklasse $\mathcal{L}$ über einem Alphabet $\Sigma$ ist eine Menge von formalen Sprachen $A \subseteq \Sigma^*$.

Da jede Sprache $A$ selbst eine Teilmenge von $\Sigma^*$ ist, ist $\mathcal{L}$ also eine Menge von Teilmengen, d.h. eine Teilmenge der Potenzmenge von $\Sigma^*$: $\mathcal{L} \subseteq \mathcal{P}(\Sigma^*)$. Auch $\mathcal{P}(\Sigma^*)$ stellt somit eine Sprachklasse dar; sie umfaßt alle möglichen formalen Sprachen über $\Sigma$.

Wir stellen an jede betrachtete Sprachklasse $\mathcal{L}$ bestimmte Forderungen. Sie sind zugleich Minimalforderungen an die Mächtigkeit der entsprechenden Grammatikformalismen.

Hier: Die Sprachklasse der regulären Sprachen

# Forderungen an eine Sprachklasse $\mathcal{L}$

# Forderungen an eine Sprachklasse $\mathcal{L}$

- (i) Jede endliche Sprache gehört zu $\mathcal{L}$.

# Forderungen an eine Sprachklasse $\mathcal{L}$

- (i) Jede endliche Sprache gehört zu $\mathcal{L}$.
- Mit zwei Sprachen $A, B \in \mathcal{L}$ gehört auch deren (ii) Vereinigung $A \cup B$ zu $\mathcal{L}$ und (iii) Verkettung $A \circ B$ zu $\mathcal{L}$.

# Forderungen an eine Sprachklasse $\mathcal{L}$

- (i) Jede endliche Sprache gehört zu $\mathcal{L}$.
- Mit zwei Sprachen $A, B \in \mathcal{L}$ gehört auch deren (ii) Vereinigung $A \cup B$ zu $\mathcal{L}$ und (iii) Verkettung $A \circ B$ zu $\mathcal{L}$.
- (iv) Mit jeder Sprache $A \in \mathcal{L}$ gehört auch deren Kleenesche Hülle $A^*$ zu $\mathcal{L}$.

# Forderungen an eine Sprachklasse $\mathcal{L}$

- (i) Jede endliche Sprache gehört zu $\mathcal{L}$.
- Mit zwei Sprachen $A, B \in \mathcal{L}$ gehört auch deren (ii) Vereinigung $A \cup B$ zu $\mathcal{L}$ und (iii) Verkettung $A \circ B$ zu $\mathcal{L}$.
- (iv) Mit jeder Sprache $A \in \mathcal{L}$ gehört auch deren Kleenesche Hülle $A^*$ zu $\mathcal{L}$.

Forderung (i) stellt sicher, dass jeder Grammatikformalismus extensionale Definitionen ermöglicht. Forderungen (ii) und (iii) leiten sich aus der Kombination der Regeln zweier Grammatiken her, wobei die Regelsätze entweder Alternativen sind (ii) oder durch Verkettung kombiniert werden (iii). Forderung (iv) schließlich bildet den Schlüssel zu unendlichen Sprachen. Während (iii) bereits sicherstellt, dass Verkettungen endlicher Länge $\in \mathcal{L}$ sind, wird diese Eigenschaft durch (iv) auf Verkettungen beliebiger Länge ausgedehnt.

# Reguläre Sprachen

Die einfachste Sprachklasse, die obigen Forderungen genügt, bilden die sogenannten regulären Sprachen. Wir bezeichnen die Klasse aller regulären Sprachen über einem gegebenen Alphabet $\Sigma$ mit $\mathrm{Reg}(\Sigma) \subseteq \mathcal{P}(\Sigma^*)$.

# Reguläre Sprachen: Definition

Wir können $\mathrm{Reg}(\Sigma)$ analog zu den Forderungen (i) – (v) rekursiv definieren:

# Reguläre Sprachen: Definition

Wir können $\mathrm{Reg}(\Sigma)$ analog zu den Forderungen (i) – (v) rekursiv definieren:

- (i) $\emptyset, \{\epsilon\} \in \mathrm{Reg}(\Sigma)$

# Reguläre Sprachen: Definition

Wir können $\mathrm{Reg}(\Sigma)$ analog zu den Forderungen (i) – (v) rekursiv definieren:

- (i) $\emptyset, \{\epsilon\} \in \mathrm{Reg}(\Sigma)$
- (ii) $\forall a \in \Sigma \ : \ \{a\} \in \mathrm{Reg}(\Sigma)$

# Reguläre Sprachen: Definition

Wir können $\mathrm{Reg}(\Sigma)$ analog zu den Forderungen (i) – (v) rekursiv definieren:

- (i) $\emptyset, \{\epsilon\} \in \mathrm{Reg}(\Sigma)$
- (ii) $\forall a \in \Sigma \ : \ \{a\} \in \mathrm{Reg}(\Sigma)$
- (iii) $A, B \in \mathrm{Reg}(\Sigma) \ \Rightarrow \ A \cup B \in \mathrm{Reg}(\Sigma)$

# Reguläre Sprachen: Definition

Wir können $\mathrm{Reg}(\Sigma)$ analog zu den Forderungen (i) – (v) rekursiv definieren:

- (i) $\emptyset, \{\epsilon\} \in \mathrm{Reg}(\Sigma)$
- (ii) $\forall a \in \Sigma \ : \ \{a\} \in \mathrm{Reg}(\Sigma)$
- (iii) $A, B \in \mathrm{Reg}(\Sigma) \ \Rightarrow \ A \cup B \in \mathrm{Reg}(\Sigma)$
- (iv) $A, B \in \mathrm{Reg}(\Sigma) \ \Rightarrow \ A \circ B \in \mathrm{Reg}(\Sigma)$

# Reguläre Sprachen: Definition

Wir können $\mathrm{Reg}(\Sigma)$ analog zu den Forderungen (i) – (v) rekursiv definieren:

- (i) $\emptyset, \{\epsilon\} \in \mathrm{Reg}(\Sigma)$
- (ii) $\forall a \in \Sigma \ : \ \{a\} \in \mathrm{Reg}(\Sigma)$
- (iii) $A, B \in \mathrm{Reg}(\Sigma) \ \Rightarrow \ A \cup B \in \mathrm{Reg}(\Sigma)$
- (iv) $A, B \in \mathrm{Reg}(\Sigma) \ \Rightarrow \ A \circ B \in \mathrm{Reg}(\Sigma)$
- (v) $A \in \mathrm{Reg}(\Sigma) \ \Rightarrow \ A^* \in \mathrm{Reg}(\Sigma)$

# Reguläre Sprachen: Definition

Wir können $\mathrm{Reg}(\Sigma)$ analog zu den Forderungen (i) – (v) rekursiv definieren:

- (i) $\emptyset, \{\epsilon\} \in \mathrm{Reg}(\Sigma)$
- (ii) $\forall a \in \Sigma \ : \ \{a\} \in \mathrm{Reg}(\Sigma)$
- (iii) $A, B \in \mathrm{Reg}(\Sigma) \ \Rightarrow \ A \cup B \in \mathrm{Reg}(\Sigma)$
- (iv) $A, B \in \mathrm{Reg}(\Sigma) \ \Rightarrow \ A \circ B \in \mathrm{Reg}(\Sigma)$
- (v) $A \in \mathrm{Reg}(\Sigma) \ \Rightarrow \ A^* \in \mathrm{Reg}(\Sigma)$

Forderung (i) wird durch die Regeln (i), (ii) und (iii) erfüllt, da mit ihnen jede endliche Sprache erzeugt werden kann. Jede reguläre Sprache $A \in \mathrm{Reg}(\Sigma)$ läßt sich durch wiederholte Anwendung der obigen Regeln darstellen.

# Beispiel

Die Sprache $A := \{ab^k c^l \mid k \in \mathbb{N}_0,\ l \in \{0,1\}\}$ ist regulär, denn

$$A = \{a\} \circ \big(\{b\}\big)^* \circ \big(\{c\} \cup \{\epsilon\}\big).$$

# Formale Sprachen: Theoretische Motivation

Das Hauptinteresse der Theorie formaler Sprachen besteht darin, eine unendliche Sprache durch eine endliche Grammatik zu beschreiben. Hierzu werden geeignete Beschreibungsformalismen benötigt. Die Gesamtheit aller Sprachen $A$, die sich in einem bestimmten Formalismus beschreiben lassen, wird als Sprachklasse $\mathcal{A}$ bezeichnet. Ein wesentlicher Aspekt der Forschungsarbeit ist somit die Untersuchung der Eigenschaften verschiedener Sprachklassen.

# Zusammenfassung

- Eine Sprache ist eine endliche oder unendliche Menge von Wörtern (= Strings)
- Wir interessieren uns für Sprachen, weil wir oft eine endliche oder unendliche Liste von Wörtern matchen wollen
- Die Reguläre Sprachen sind sehr interessant, weil wir eine Reguläre Sprache kompakt darstellen können, z.B., als Regulärer Ausdruck, deterministischer endlicher Automat (DEA), oder nichtdeterministischer endlicher Automat (NEA)

# Fahrplan

- Reguläre Sprachen
- Reguläre Ausdrücke
- deterministische Automaten
- nichtdeterministische Automaten
- Reguläre Sprachen
  = Reguläre Ausdrücke
  = deterministische Automaten
  = nichtdeterministische Automaten

# Outline

# Reguläre Ausdrücke (1)

Eine etwas kompaktere Darstellung von regulären Sprachen als die gerade eingeführte Mengenschreibweise erlauben reguläre Ausdrücke.

Ein regulärer Ausdruck über einem Alphabet $\Sigma$ besteht aus den Zeichen von $\Sigma$ sowie den sogenannten Metazeichen $($, $)$, $|$ und $*$.

Um Verwechslungen von regulären Ausdrücken mit Wörtern über $\Sigma$ zu vermeiden, werden reguläre Ausdrücke auf diesen Folien durch Unterstreichung gekennzeichnet (z.B. ist $aab$ ein Wort, $\underline{aab}$ ein regulärer Ausdruck). Wir verwenden $\underline{r}, \underline{s}, \underline{t}, \ldots$ als Variablen für reguläre Ausdrücke.

# Reguläre Ausdrücke (2)

Wir können reguläre Ausdrücke über $\Sigma$ als Wörter über dem Alphabet $\Sigma \cup M$ auffassen, wobei $M := \{(, ), |, *\}$ die Menge der Metazeichen ist und $\Sigma \cap M = \emptyset$ angenommen wird.

Die Menge aller zulässigen regulären Ausdrücke über $\Sigma$ bezeichnen wir mit $R(\Sigma)$.

Jeder Ausdruck $\underline{r} \in R(\Sigma)$ beschreibt eine formale Sprache über $\Sigma$, die wir mit $\mathcal{L}\left[\underline{r}\right]$ bezeichnen.

# Abbildung: Regulärer Ausdruck $\mapsto$ Reguläre Sprache

Die Menge aller regulären Ausdrücke $R(\Sigma)$ und die Abbildung $\mathcal{L} : R(\Sigma) \to \mathcal{P}(\Sigma^*)$, die jedem regulären Ausdruck die durch ihn beschriebene Sprache zuordnet, werden wiederum rekursiv definiert:

| | | |
|---|---|---|
| $\underline{\epsilon} \in R(\Sigma)$ | $\mathcal{L}\left[\underline{\epsilon}\right] := \{\epsilon\}$ | |
| $\forall a \in \Sigma \ : \ \underline{a} \in R(\Sigma)$ | $\mathcal{L}\left[\underline{a}\right] := \{a\}$ | |
| $\underline{r}, \underline{s} \in R(\Sigma) \ \Rightarrow \ \underline{(r)\,(s)} \in R(\Sigma)$ | $\mathcal{L}\left[\underline{(r)\,(s)}\right] := \mathcal{L}\left[\underline{r}\right] \circ \mathcal{L}\left[\underline{s}\right]$ | Konkatenation |
| $\underline{r}, \underline{s} \in R(\Sigma) \ \Rightarrow \ \underline{r \mid s} \in R(\Sigma)$ | $\mathcal{L}\left[\underline{r \mid s}\right] := \mathcal{L}\left[\underline{r}\right] \cup \mathcal{L}\left[\underline{s}\right]$ | Disjunktion |
| $\underline{r} \in R(\Sigma) \ \Rightarrow \ \underline{(r)*} \in R(\Sigma)$ | $\mathcal{L}\left[\underline{(r)*}\right] := \left(\mathcal{L}\left[\underline{r}\right]\right)^*$ | Kleene-Hülle |

# Reguläre Ausdrücke = Reguläre Sprachen

Aus dieser Definition ist unmittelbar ersichtlich, dass reguläre Ausdrücke gerade die regulären Sprachen beschreiben, bzw. genauer:

**Satz**

$\mathrm{Reg}(\Sigma) = \mathcal{L}\left[R(\Sigma)\right] \cup \{\emptyset\}$.

# Vereinfachte Notation

Zur Vereinfachung der Notation vereinbaren wir, dass Klammern weggelassen werden dürfen, wo sie aufgrund der Assoziativität von Disjunktion und Konkatenation bzw. aufgrund der Präzedenzregeln (Kleenesche Hülle vor Konkatenation vor Disjunktion) nicht benötigt werden (ebenso dürfen zusätzliche Klammern eingefügt werden, um die Lesbarkeit zu verbessern). Beispielsweise ist wegen der Assoziativität der Konkatenation

$$\underline{((a)(b))(c)} \equiv \underline{(a)((b)(c))} \equiv: \underline{abc}$$

# Äquivalenz regulärer Ausdrücke

Zwei reguläre Ausdrücke $\underline{r}, \underline{s} \in R(\Sigma)$ sind äquivalent ($\underline{r} \equiv \underline{s}$), wenn $\mathcal{L}\left[\underline{r}\right] = \mathcal{L}\left[\underline{s}\right]$.

# Normalform

Es ist nicht immer offensichtlich, ob zwei reguläre Ausdrücke äquivalent sind, z.B.

$$\underline{(ab*)*} \equiv \underline{(a(a*b*)*) \mid \epsilon} \equiv \underline{(a(a \mid b)* \mid \epsilon)}$$

Daher ist es wünschenswert, reguläre Ausdrücke auf eine sogenannte "Normalform" zurückzuführen. Jeder zu einem regulären Ausdruck $\underline{r} \in R(\Sigma)$ äquivalente Ausdruck der Form $\underline{s_1 \mid s_2 \mid \ldots \mid s_n}$, wobei keiner der Ausdrücke $\underline{s_i}$ das Metazeichen $\mid$ enthält, heißt disjunktive Normalform von $\underline{r}$. Die disjunktive Normalform ist im allgemeinen nicht eindeutig bestimmt.

# POSIX

Für Software-Anwendungen erweist sich die bisher verwendete Standardnotation regulärer Ausdrücke oft als unhandlich. Sie wurde daher nach und nach um weitere Metazeichen ergänzt. Inzwischen hat sich als Standard die sogenannte POSIX-Notation mit den Metazeichen

$$M := \{(, ), |, *, ., ?, +, \{, ,, \}, [, -, ], \hat{}, \$ \dots\}$$

etabliert.

# POSIX: Backslash

Das Alphabet $\Sigma$ besteht stets aus dem kompletten Zeichenvorrat der Anwendung (ASCII, Unicode, ...), daher ist zwangsläufig $M \cap \Sigma \neq \emptyset$ (und sogar $M \subseteq \Sigma$). Soll ein Zeichen aus $M$ "wörtlich", d.h. als Zeichen von $\Sigma$ interpretiert werden, so wird ein Backslash \ vorangestellt ($, ^, , und - werden nur in bestimmten Kontexten als Metazeichen interpretiert).

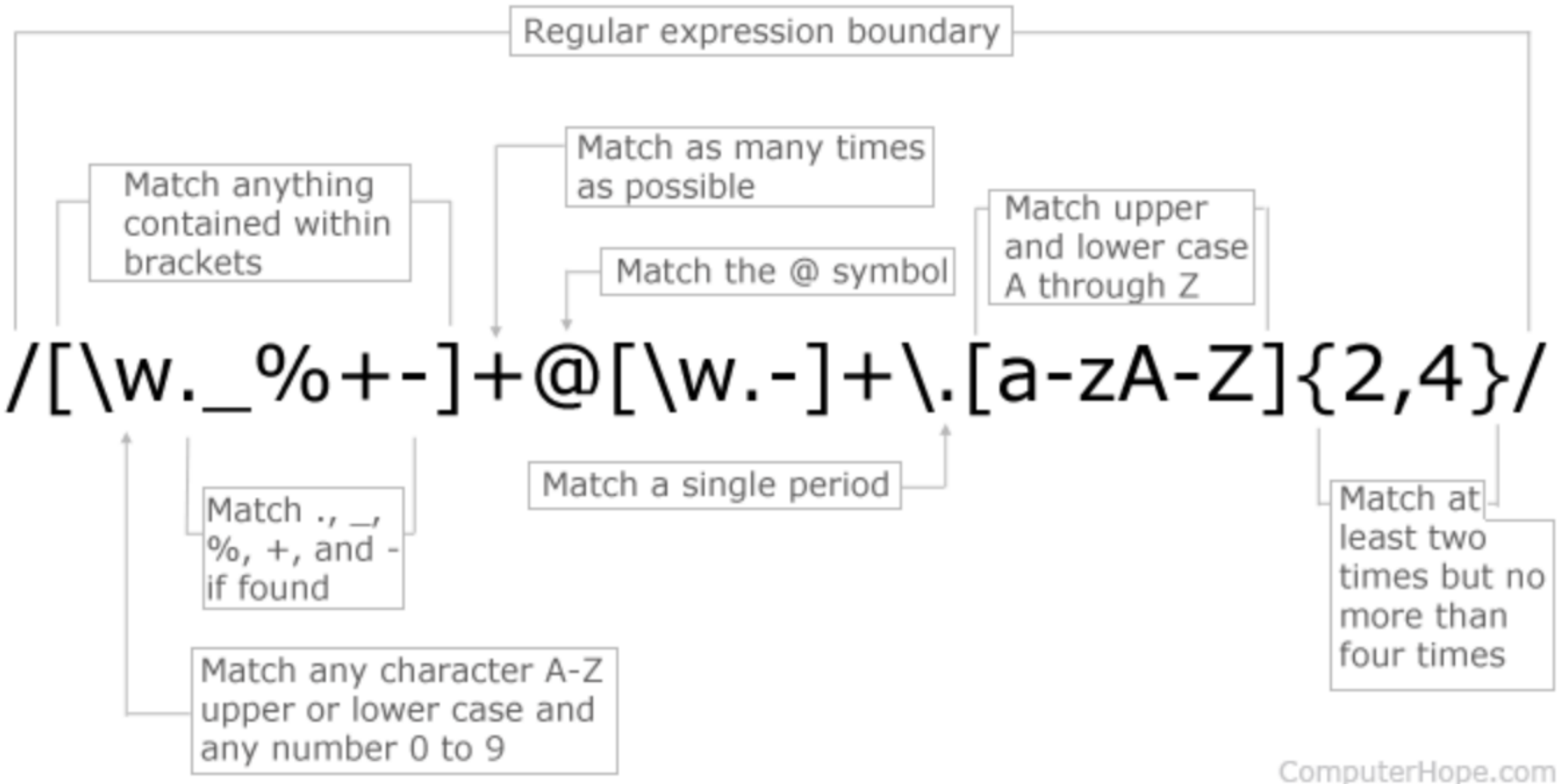
# POSIX: Epsilon und Anfang-vom-String

Man beachte, dass der leere reguläre Ausdruck $\underline{\epsilon}$ nicht durch ein spezielles Zeichen sondern durch einen leeren String repräsentiert wird.

Auch wichtig: POSIX erlaubt einen Match an eine beliebige Stelle in einem String. $\hat{}$ matcht den Anfang vom String. \$ matcht das Ende vom String.

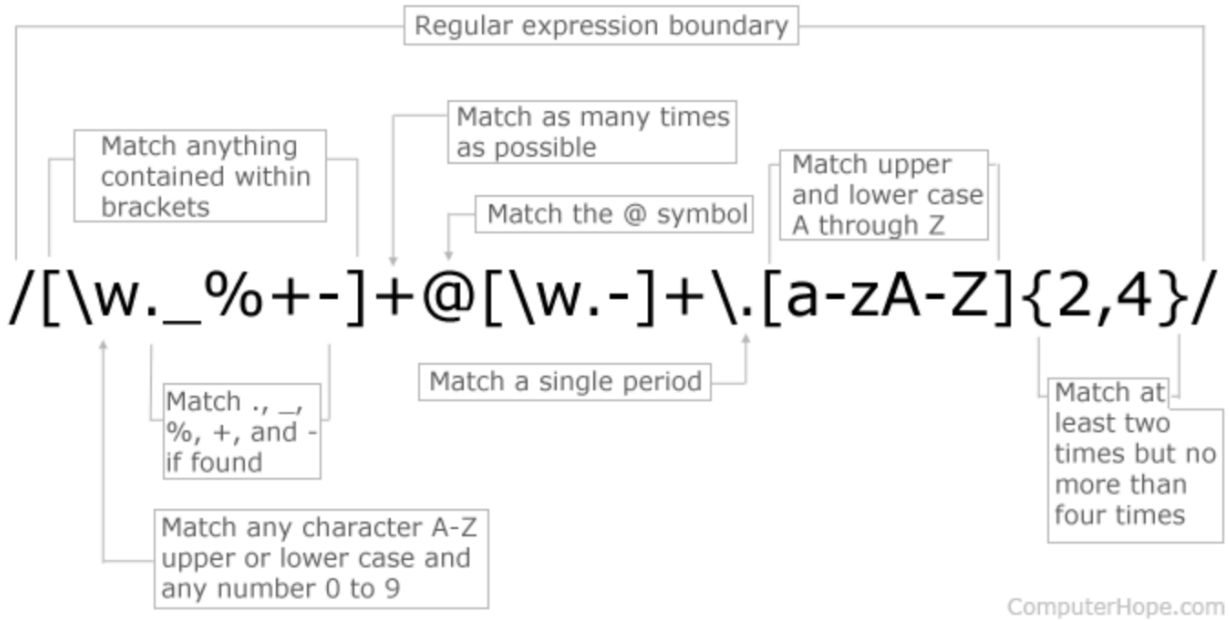
So würde beispielsweise der POSIX-Ausdruck $\hat{}[ac] + (baa|)\$$ in der Standardnotation $\underline{(a|c)(a|c)*(baa|\epsilon)}$ lauten.

# Regulärer Ausdruck für Email-Adressen



## Übung

(i) Modifiziere den Ausdruck, so dass die neuen top level domains richtig erkannt werden. (ii) Denken Sie sich eine valide Email-Adresse aus, die vom neuen Ausdruck nicht erkannt wird. (iii) Gibt es eine nicht valide Email-Adresse, die vom neuen Ausdruck trotzdem als richtig erkannt wird?



# Fahrplan

- Reguläre Sprachen
- Reguläre Ausdrücke
- deterministische Automaten
- nichtdeterministische Automaten
- Reguläre Sprachen
  = Reguläre Ausdrücke
  = deterministische Automaten
  = nichtdeterministische Automaten

# Outline

# Endlicher Automat = finite state automaton

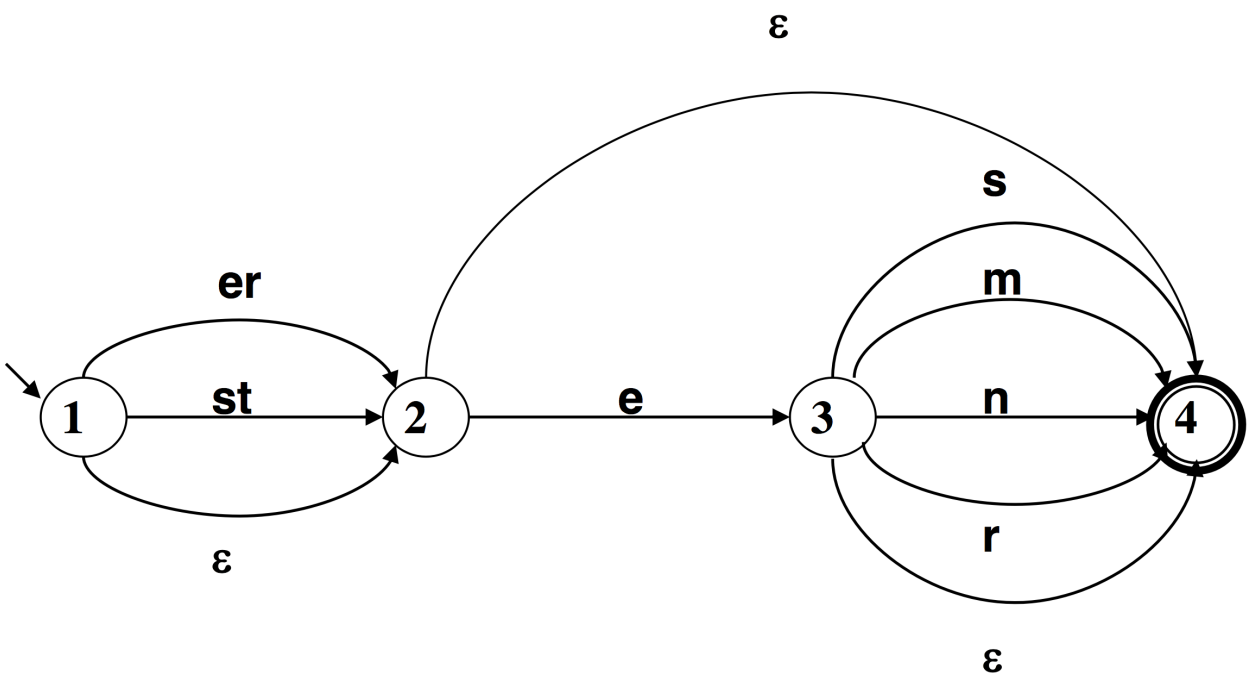
Ein endlicher Automat (EA) $M$ über dem Alphabet $\Sigma$ ist formal ein Quintupel $M = (Q, \Sigma, s, F, \delta)$.

Dabei bezeichnet $Q$ eine endliche Menge von Zuständen, $s \in Q$ den Startzustand des Automaten und $F \subseteq Q$ die Menge der Endzustände.

Die Übergangsrelation $\delta \subseteq Q \times \Sigma \times Q$ beschreibt, welche Zustandsübergänge der EA ausführen kann: ist $(q, a, q') \in \delta$, so kann $M$ durch Einlesen des Zeichens $a$ (Label) vom Zustand $q$ (Ausgangszustand) in den Zustand $q'$ (Zielzustand) wechseln.

Wir schreiben Übergangsregeln in der Form $q \xrightarrow{a} q'$ und bezeichnen sie kurz als Bewegungen (oder Transitionen).

# Automat für Adjektive (Buchstabenebene)



# Deterministisch vs nicht-deterministisch

Gibt es für jeden Ausgangszustand $q$ und jedes Zeichen $a$ höchstens einen Zielzustand (d.h. höchstens eine Übergangsregel $q \xrightarrow{a} q'$), und keine $\epsilon$-Bewegungen (nächste Folie), dann bezeichnen wir $M$ als deterministisch (DEA), andernfalls als nichtdeterministisch (NEA).

Für einen deterministischen EA können wir die Übergangsrelation als (partielle) Abbildung $\delta : Q \times \Sigma \to Q$ auffassen.

# Epsilon-Bewegungen

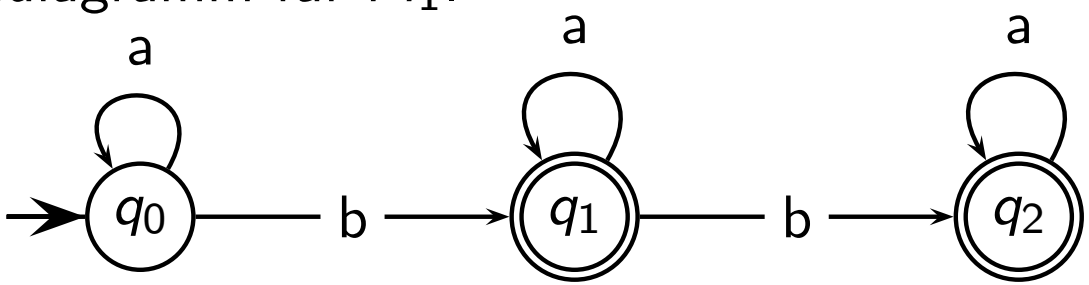
Bei einem NEA erlauben wir Epsilon-Bewegungen, also Zustandsübergänge ohne Einlesen eines Zeichens. Diese werden als Bewegungen mit dem Label $\epsilon$ dargestellt ($q \xrightarrow{\epsilon} q'$), also ist hier $\delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Q$.

# Beispiel

Deterministischer EA $M_1 := (Q_1, \Sigma, s_1, F_1, \delta_1)$ über $\Sigma = \{a, b\}$ mit

$$Q_1 := \{q_0, q_1, q_2\}$$
$$s_1 := \ q_0$$
$$F_1 := \{q_1, q_2\}$$
$$\delta_1 := \{q_0 \xrightarrow{a} q_0,\ q_0 \xrightarrow{b} q_1,\ q_1 \xrightarrow{a} q_1,\ q_1 \xrightarrow{b} q_2,\ q_2 \xrightarrow{a} q_2\}.$$

Endliche Automaten lassen sich anschaulich durch Übergangsdiagramme darstellen. Die Abbildung zeigt das Übergangsdiagramm für $M_1$.



# Ein Automat akzeptiert ein Eingabewort, wenn ...

Ein Wort $w \in \Sigma^*$ wird von dem EA $M$ akzeptiert, wenn dieser sich nach Einlesen aller Zeichen von $w$ in einem Endzustand $q \in F$ befindet.

Umgekehrt wird $w$ nicht akzeptiert, wenn $M$ sich zum Schluß nicht in einem Endzustand befindet oder schon während der Verarbeitung von $w$ mit einem Fehler abbricht, weil keine zu dem nächsten einzulesenden Zeichen passende Bewegung definiert ist.

Gewöhnlich wird kein Unterschied zwischen den beiden Arten des Nichtakzeptierens gemacht.

# Simulation eines Automaten

Die schrittweise Abarbeitung eines Eingabewortes $w$ auf dem Rechner nennt man Simulation des Automaten $M$.

Der Zustand, in dem sich $M$ jeweils befindet, wird als aktiver Zustand bezeichnet.

Wichtig: bei einem nichtdeterministischen EA können mehrere Zustände gleichzeitig aktiv sein!

Die Menge aller akzeptierten Wörter $w \in \Sigma^*$ bezeichnet man als die von $M$ akzeptierte Sprache $\mathcal{L}\left[M\right]$.

# Konfiguration eines Automaten

Zur formalen Definition von $\mathcal{L}\left[M\right]$ führen wir den Begriff der Konfiguration $(q, w) \in Q \times \Sigma^*$ ein, wobei $q$ den gerade aktiven Zustand und $w$ die noch zu verarbeitende Eingabe bezeichnet.

Der Übergang $(q, a \circ w) \vdash (q', w)$ von einer Konfiguration $(q, a \circ w)$ in die Konfiguration $(q', w)$ (d.h. der Wechsel vom Zustand $q$ in den Zustand $q'$ unter Einlesen des Zeichens $a$) ist erlaubt, wenn eine entsprechende Bewegung $q \xrightarrow{a} q' \in \delta$ definiert ist.

Im Falle eines NEA erlaubt jede $\epsilon$-Bewegung $q \xrightarrow{\epsilon} q'$ einen Übergang $(q, w) \vdash (q', w)$ ohne Einlesen eines Zeichens.

# Pfad

Ein Pfad von $(q, w)$ nach $(q', w')$ ist eine Kette von Übergängen

$$(q, w) = (q_0, w_0) \vdash (q_1, w_1) \vdash \cdots \vdash (q_n, w_n) = (q', w').$$

Wir schreiben kurz $(q, w) \vdash^* (q', w')$, wenn (mindestens) ein solcher Pfad existiert.

# Automaten-basierte Definition von Sprache

Wenden wir $M$ auf das Eingabewort $w$ an, so startet der Automat in der Konfiguration $(s, w)$ und $w$ wird genau dann akzeptiert, wenn es einen Pfad zu einem geeigneten Endzustand $q \in F$ gibt, der $w$ vollständig abarbeitet: $(s, w) \vdash^* (q, \epsilon)$. Also können wir definieren:

$$\mathcal{L}\left[M\right] := \{w \in \Sigma^* \mid \exists q \in F : \ (s, w) \vdash^* (q, \epsilon)\}$$

Zwei EA $M_1$ und $M_2$ über demselben Alphabet $\Sigma$ heißen äquivalent ($M_1 \equiv M_2$), wenn $\mathcal{L}\left[M_1\right] = \mathcal{L}\left[M_2\right]$.

# NEA = DEA

Es ist unmittelbar klar, dass die DEA Spezialfälle der NEA sind. Somit gibt es zu jedem DEA $M$ einen äquivalenten NEA. Um die Äquivalenz von NEA und DEA nachzuweisen, müssen wir also noch zeigen, dass es zu jedem NEA einen äquivalenten DEA gibt.

# Theorem

## Lemma

*Sei $M = (Q, \Sigma, s, F, \delta)$ ein NEA. Dann gibt es einen zu $M$ äquivalenten DEA $M' = (Q', \Sigma, s', F', \delta')$, d.h. es gilt $\mathcal{L}[M'] = \mathcal{L}[M]$.*

# Equivalence of NFAs and DFAs

# Equivalence of NFAs and DFAs

- The $\epsilon$-closure of a state $q$, denoted $E(q)$, is the set of all states, including $q$ itself, that can be reached using only $\epsilon$-moves.

# Equivalence of NFAs and DFAs

- The $\epsilon$-closure of a state $q$, denoted $E(q)$, is the set of all states, including $q$ itself, that can be reached using only $\epsilon$-moves.
- Let $\{Q, \Sigma, q_0, F, \delta\}$ be an NFA.

# Equivalence of NFAs and DFAs

- The $\epsilon$-closure of a state $q$, denoted $E(q)$, is the set of all states, including $q$ itself, that can be reached using only $\epsilon$-moves.
- Let $\{Q, \Sigma, q_0, F, \delta\}$ be an NFA.
- Define a DFA $\{Q', \Sigma', q_0', F', \delta'\}$ as follows:

# Equivalence of NFAs and DFAs

- The $\epsilon$-closure of a state $q$, denoted $E(q)$, is the set of all states, including $q$ itself, that can be reached using only $\epsilon$-moves.
- Let $\{Q, \Sigma, q_0, F, \delta\}$ be an NFA.
- Define a DFA $\{Q', \Sigma', q_0', F', \delta'\}$ as follows:
  - $Q' = \mathcal{P}(Q)$.

# Equivalence of NFAs and DFAs

- The $\epsilon$-closure of a state $q$, denoted $E(q)$, is the set of all states, including $q$ itself, that can be reached using only $\epsilon$-moves.
- Let $\{Q, \Sigma, q_0, F, \delta\}$ be an NFA.
- Define a DFA $\{Q', \Sigma', q_0', F', \delta'\}$ as follows:
  - $Q' = \mathcal{P}(Q)$.
  - $\Sigma' = \Sigma$.

# Equivalence of NFAs and DFAs

- The $\epsilon$-closure of a state $q$, denoted $E(q)$, is the set of all states, including $q$ itself, that can be reached using only $\epsilon$-moves.
- Let $\{Q, \Sigma, q_0, F, \delta\}$ be an NFA.
- Define a DFA $\{Q', \Sigma', q_0', F', \delta'\}$ as follows:
  - $Q' = \mathcal{P}(Q)$.
  - $\Sigma' = \Sigma$.
  - $q_0' = E(q_0)$.

# Equivalence of NFAs and DFAs

- The $\epsilon$-closure of a state $q$, denoted $E(q)$, is the set of all states, including $q$ itself, that can be reached using only $\epsilon$-moves.
- Let $\{Q, \Sigma, q_0, F, \delta\}$ be an NFA.
- Define a DFA $\{Q', \Sigma', q_0', F', \delta'\}$ as follows:
  - $Q' = \mathcal{P}(Q)$.
  - $\Sigma' = \Sigma$.
  - $q_0' = E(q_0)$.
  - For any state $S \in \mathcal{P}(Q)$ and any $a \in \Sigma$, define $\delta'(S, a)$ to be

$$\delta'(S, a) = \bigcup_{q \in S} E(\delta(q, a)).$$

# Equivalence of NFAs and DFAs

- The $\epsilon$-closure of a state $q$, denoted $E(q)$, is the set of all states, including $q$ itself, that can be reached using only $\epsilon$-moves.
- Let $\{Q, \Sigma, q_0, F, \delta\}$ be an NFA.
- Define a DFA $\{Q', \Sigma', q_0', F', \delta'\}$ as follows:
  - $Q' = \mathcal{P}(Q)$.
  - $\Sigma' = \Sigma$.
  - $q_0' = E(q_0)$.
  - For any state $S \in \mathcal{P}(Q)$ and any $a \in \Sigma$, define $\delta'(S, a)$ to be

$$\delta'(S, a) = \bigcup_{q \in S} E(\delta(q, a)).$$

  - The final states in $F'$ are those states that contain final states of the NFA.

# “Automatensprachen” = reguläre Sprachen

Endliche Automaten akzeptieren genau diejenigen Sprachen, die durch reguläre Ausdrücke beschrieben werden können, also die Klasse der regulären Sprachen. Zum Nachweis dieser Behauptung müssen wir die Äquivalenz von EA und regulären Ausdrücken zeigen: zu jedem regulären Ausdruck $\underline{r}$ gibt es (mindestens) einen endlichen Automaten $M$ mit $\mathcal{L}\left[M\right] = \mathcal{L}\left[\underline{r}\right]$; umgekehrt gibt es zu jedem endlichen Automaten $M$ einen regulären Ausdruck $\underline{r}$ mit $\mathcal{L}\left[\underline{r}\right] = \mathcal{L}\left[M\right]$.

# Äquivalenzbeweis

Zu jedem regulären Ausdruck $\underline{r}$ gibt es (mindestens) einen endlichen Automaten $M$ mit $\mathcal{L}\left[M\right] = \mathcal{L}\left[\underline{r}\right]$: Konstruktionsverfahren.

Zu jedem endlichen Automaten $M$ gibt es einen regulären Ausdruck $\underline{r}$ mit $\mathcal{L}\left[\underline{r}\right] = \mathcal{L}\left[M\right]$. Diese Richtung ist komplexer und sehr "technisch", d.h. intuitiv schwer nachvollziehbar.

# Fahrplan

- Reguläre Sprachen
- Reguläre Ausdrücke
- deterministische Automaten
- nichtdeterministische Automaten
- Reguläre Sprachen
  = Reguläre Ausdrücke
  = deterministische Automaten
  = nichtdeterministische Automaten

# Outline

# Breadth first search

# Breadth first search

- Breadth first traversal is accomplished by enqueueing each level of a tree sequentially as the root of any subtree is encountered. There are 2 cases in the iterative algorithm.

# Breadth first search

- Breadth first traversal is accomplished by enqueueing each level of a tree sequentially as the root of any subtree is encountered. There are 2 cases in the iterative algorithm.
- Root case: The traversal queue is initially empty so the root node must be added before the general case.

# Breadth first search

- Breadth first traversal is accomplished by enqueueing each level of a tree sequentially as the root of any subtree is encountered. There are 2 cases in the iterative algorithm.
- Root case: The traversal queue is initially empty so the root node must be added before the general case.
- General case: Process any items in the queue, while also expanding their children. Stop if the queue is empty. The general case will halt after processing the bottom level as leaf nodes have no children.

# Breadth first search

- Breadth first traversal is accomplished by enqueueing each level of a tree sequentially as the root of any subtree is encountered. There are 2 cases in the iterative algorithm.
- Root case: The traversal queue is initially empty so the root node must be added before the general case.
- General case: Process any items in the queue, while also expanding their children. Stop if the queue is empty. The general case will halt after processing the bottom level as leaf nodes have no children.
- Input: A search problem. A search-problem abstracts out the problem specific requirements from the actual search algorithm.

# Breadth first search

- Breadth first traversal is accomplished by enqueueing each level of a tree sequentially as the root of any subtree is encountered. There are 2 cases in the iterative algorithm.
- Root case: The traversal queue is initially empty so the root node must be added before the general case.
- General case: Process any items in the queue, while also expanding their children. Stop if the queue is empty. The general case will halt after processing the bottom level as leaf nodes have no children.
- Input: A search problem. A search-problem abstracts out the problem specific requirements from the actual search algorithm.
- Output: An ordered list of actions to be followed to reach from start state to the goal state.

Wikipedia: `https://de.wikipedia.org/wiki/Breitensuche`

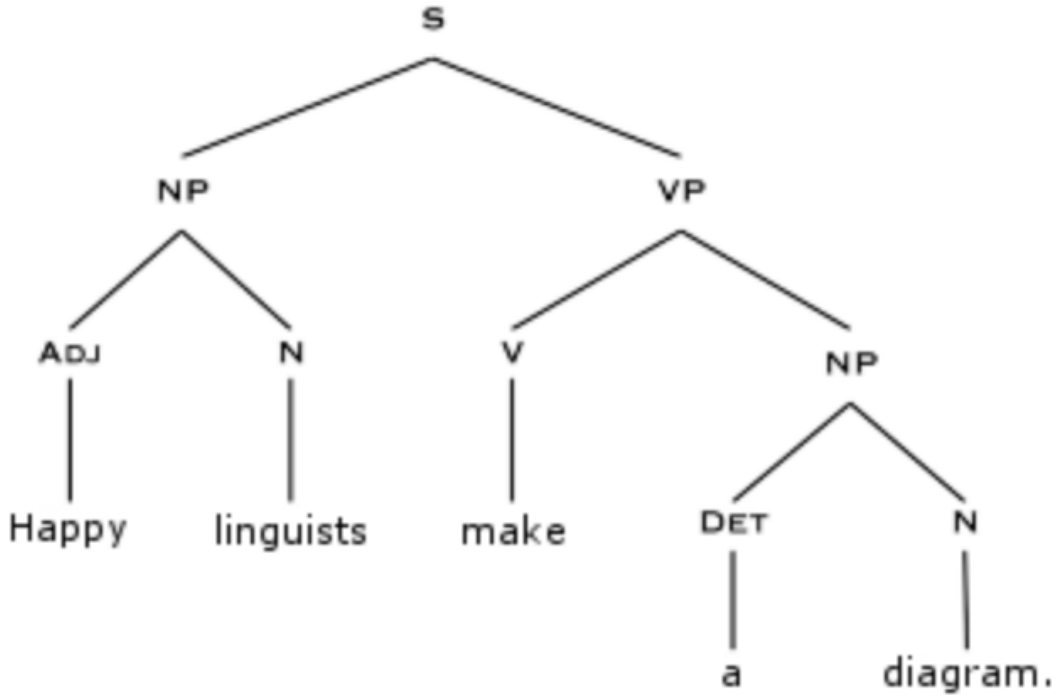
# Depth first search (DFS)

```
procedure DFS(G,v):
    process v
    label v as discovered
    for all edges from v to w in G.adjacentEdges(v) do
        if vertex w is not labeled as discovered then
            recursively call DFS(G,w)
```

Wikipedia: `https://de.wikipedia.org/wiki/Tiefensuche`

## Übung

Give order of nodes for BFS and DFS



# Zum Schluss: Besonders klausurrelevant

- Reguläre Ausdrücke (POSIX)
- Formale Definition des DEA
- Formale Definition des NEA
- Unterschiede DEA vs NEA
- Automaten zeichnen
- Depth first search (DFS)
- Breadth first search (BFS)
- Nicht relevant: Determinisierung von NEAs

# Outline

In der folgenden Aufgabe werden Sie mit dem Tiger-Corpus arbeiten. Das Tiger-Corpus enthält ca. 50000 deutsche Sätze, die mit morphologischen und syntaktischen Informationen wie z.B. Wortarten annotiert sind. Auf Moodle ist ein Ausschnitt aus dem Tiger-Corpus verlinkt (tiger.txt), in dem in jeder Zeile ein Wort steht.

| | |
|---|---|
| . | irgendein beliebiges Zeichen |
| .* | kein oder beliebig viele Zeichen |
| [a-e] | a, b, c, d oder e |
| [^a-e] | alle Zeichen außer a, b, c, d und e |
| (maus\|hund) | Zeichenfolge maus oder hund |
| (ab)* | kein oder beliebig viele ab |
| (ab)+ | mindestens ein oder oder mehr ab |
| (ab)? | kein oder ein ab |
| ^ | Anfang einer Zeile |
| $ | Ende einer Zeile |

Unter Linux können Sie mit dem Kommando grep mit regulären Ausdrücken in Dateien suchen (unter Windows geht das mit Notepad++ in der erweiterten Suchoption). Das Kommando grep -P "abc" datei.txt gibt z.B. jede Zeile von datei.txt aus, in der abc vorkommt.

Versuchen Sie, einen regulären Ausdruck zu finden, mit dem Sie alle Wortformen (Präsens, Präteritum, mit Partizipialendungen) des Wortes reden (aber nicht z.B. von mitreden oder Gerede) im Tiger-Corpus finden, und geben Sie ihn an. Unter Linux können Sie außerdem wc verwenden, um herauszufinden, wie viele Vorkommen es sind.

Einige Sonderzeichen (z.B. wenn nach "+" gesucht werden soll) müssen mit einem Backslash ( nach links geneigter Schrägstrich: "\") escaped werden; dazu und generell zu regulären Ausdrücken gibt es viele vollständige und ausführliche Hilfeseiten online.

Spielen Sie ein wenig mit grep und anderen Unix-Befehlen! der man-Befehl (z.B. man grep) liefert zu fast allen Unix-Befehlen Erklärungen und eine Auflistung möglicher Optionen. Übung macht den Meister, und mit ein wenig Tüftelei kann man in einer Zeile mächtige Befehle zusammenstellen.

Mit dem Operator | können mehrere Unix-Befehle hintereinandergeschaltet werden (man spricht auch von "pipen" oder einer "Pipeline"), sodass das Ergebnis des einen Befehls vom nächsten Befehl aufgenommen und weiterverarbeitet wird.

Mit den Operatoren $>$ und $>>$ kann man das Ergebnis eines Befehls in eine Datei schreiben bzw. an eine Datei anhängen, um nicht alles auf der Konsole ausgegeben zu bekommen.

In regulären Ausdrücken werden (runde Klammern) für zusammengehörige Zeichenfolgen (="Gruppen") verwendet und [eckige Klammern], um Mengen/Klassen von Zeichen zu deklarieren. Gruppen sind innerhalb von Klassen nicht aussagekräftig. Dies kann zu Verwirrung führen. Um mehrere Gruppen zu quantifizieren (z.B. mit +), kann man diese zu einer "veroderten Gruppe" machen: ((gruppe1)|(gruppe2)|(gruppe3))+ z.B. matcht alles, was mindestens eine der drei Gruppen enthält, egal in welcher Reihenfolge oder Zusammensetzung.

Um sicherzustellen, dass alle Wortformen von reden gematcht werden, sollten Sie auch wirklich alle Wortformen testen und nicht darauf vertrauen, dass wirklich alle Wortformen von reden im Tiger-Corpus vorkommen.

grep 'red' tiger.txt | uniq | sort | uniq -c | sort -gr > output.txt

# grep

- “minus P” option: perl-compatible regular expression (recommended for exercises)
  `grep -P 'dunke?l' tiger.txt`
- “minus i”: case insensitive
  `grep -Pi 'darum' tiger.txt`
- “minus v”: complement
  `grep -Pi 'darum' tiger.txt | grep -Pv 'darum'`
- To read up on definition of these options:
  `man grep`

## Übung

Zeichnen Sie das Diagramm des folgenden Automaten $M = (Q, \Sigma, s, F, \delta)$. Alphabet $\Sigma = \{a, b, c\}$. Zustandsmenge $Q = \{1, 2, 3, 4, 5\}$. Startzustand: 5. Endzustandsmenge $F = \{1, 3\}$. Übergangsrelation $\delta = \{(1, b, 3), (1, c, 4), (2, b, 4)\} \cup \{(4, b, 3), (4, c, 4), (5, a, 2), (5, b, 1), (5, c, 4)\}$

## Übung

### BFS and DFS for “bba”