

# Reguläre Grammatiken

---

- Eine formale Grammatik ist ein 4-Tupel  $G = \{V, \Sigma, P, S\}$  mit
  - $V$ : Menge der Nichtterminalsymbole (Variablen)
  - $\Sigma$ : Menge der Terminalsymbole (Alphabet)
  - $P$ : Menge von Produktionsregeln
  - $S$ : Ein Startsymbol aus der Menge  $V$
- In der allgemeinsten Form ist für jede Produktionsregel lediglich vorgeschrieben, dass auf der linken Seite mindestens ein Nichtterminal vorkommt
  - Damit lassen sich Grammatiken bilden, für die nicht entscheidbar ist, ob ein vorgegebenes Wort zur zugehörigen Sprache  $L(G)$  gehört oder nicht  $\Rightarrow$  Einfachere Grammatiken sind wünschenswert

- **Reguläre Grammatiken** schränken die erlaubten Produktionsregeln sehr stark ein:
  - Auf der linken Seite darf nur genau ein Nichtterminalsymbol stehen
  - Auf der rechten Seite darf entweder nur ein Terminalsymbol oder ein Terminalsymbol gefolgt von einem Nichtterminalsymbol stehen (**rechtsregulär**<sup>1</sup>)
- Eine Sprache heißt regulär, wenn es eine reguläre Grammatik gibt, die die Sprache erzeugt

---

<sup>1</sup>Alternativ kann eine Grammatik auch **linksregulär** sein, in diesem Fall steht das Nichtterminalsymbol links vom Terminalsymbol

## Aufgabe 1

- Entscheide, welche der folgenden Grammatiken regulär sind:

a)  $A \rightarrow Aa|Bb|a$

$$B \rightarrow Ba|b$$

b)  $aAb \rightarrow aaab|aBb$

$$B \rightarrow c$$

c)  $A \rightarrow abA|ab$

$$B \rightarrow bB|b$$

d)  $C \rightarrow cD|cE$

$$D \rightarrow d$$

$$E \rightarrow e$$

a) (links)regulär b) nicht regulär c) nicht regulär d) (rechts)regulär

- Reguläre Sprachen sind genau diejenigen Sprachen, die von einem deterministischen, endlichen Automaten (DEA) akzeptiert werden
- Anders gesagt:
  - Existiert ein DEA, der eine Sprache akzeptiert, so ist diese Sprache regulär
  - Zu jeder regulären Grammatik kann man einen DEA angeben, der  $L(G)$  akzeptiert

## Aufgabe 2

- Gib eine reguläre Grammatik an, die die von diesem Automaten akzeptierte Sprache erzeugt

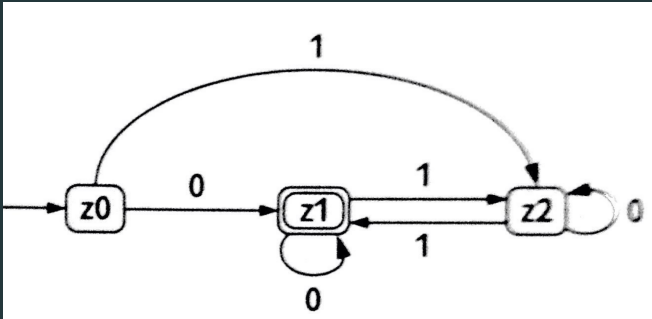


Abbildung 1: Ein einfacher DEA, entnommen aus Klett Informatik Oberstufe

## Aufgabe 3

- Entwirf einen Automaten (als Zustandsübergangsdiagramm), der die von der folgenden Grammatik erzeugte Sprache akzeptiert:

$$S \rightarrow aA|bB$$

$$A \rightarrow aA|\varepsilon$$

$$B \rightarrow aA|bB|\varepsilon$$

## Aufgabe 4

- Entwirf einen Automaten (als Zustandsübergangsdiagramm), der die von der folgenden Grammatik erzeugte Sprache akzeptiert:

$$S \rightarrow aA|bB$$

$$A \rightarrow bB|a$$

$$B \rightarrow aA|bB$$



- Eine reguläre Sprache kann auch durch einen **regulären Ausdruck** angegeben werden
- Ein regulärer Ausdruck kann aus folgenden Teilen aufgebaut werden:
  - einem oder mehreren Terminalsymbolen
  - dem Zeichen | als Trennzeichen alternativer Teilausdrücke
  - dem Zeichen \*, das eine beliebig häufige Wiederholung (auch kein Mal) einer Zeichenfolge anzeigt
  - Klammern zur Festlegung der Auswertungsreihenfolge

## Beispiele für reguläre Ausdrücke

- $a^* | b^*$ : Sprache aller Wörter, die aus beliebig vielen a's **oder** beliebig vielen b's bestehen (auch  $\varepsilon$  ist Teil der Sprache)
- $ac^*a$ : Sprache aller Wörter, die mit a anfangen, mit a enden und dazwischen beliebig viele c's enthalten (auch aa ist Teil der Sprache)
- $a(b | c)^*$ : Sprache aller Wörter, die mit a beginnen, gefolgt von einer beliebig langen Folge aus b's oder c's (kein XOR!) (auch a ist Teil der Sprache)

## Aufgabe 5

- Konstruiere mit Excorciser zu einigen regulären Ausdrücke die zugehörigen Automaten

- Reguläre Ausdrücke (in einer erweiterten Form) stehen in vielen Programmiersprachen und Editoren zur Verfügung und erlauben beispielsweise:
  - Suche nach bestimmten Zeichenfolgen, die gleich aufgebaut, aber nicht identisch sind (bspw. zwei Ziffern hinter einem Punkt)
  - Einfache Gültigkeitsprüfung von eingegebenen Zeichenfolgen, (bspw. bei Mailadressen)

- Mit der Methode `String.matches(String regex)` kann geprüft werden, ob ein String zu einem regulären Ausdruck “passt”
- Beispiel:
  - `"hallo".matches("[a-z]*");` liefert `true` zurück
  - `"Hallo".matches("[a-z]*");` liefert `false` zurück

## Aufgabe 6

- Experimentiere mit einigen regulären Ausdrücken in Java
  - Erstelle dazu ein neues BlueJ-Projekt mit einer einzelnen Methode und rufe darin die oben genannte Methode auf verschiedenen Strings und mit unterschiedlichen regulären Ausdrücken auf
- Implementiere eine Methode, die prüft, ob eine Mailadresse dem folgenden Aufbau(von links nach rechts) entspricht:
  - Eine beliebige Folge von Kleinbuchstaben und Ziffern
  - Genau ein Mal das Zeichen @
  - Eine beliebige Folge von Kleinbuchstaben und Ziffern
  - Ein Punkt .
  - Eine Folge von 2 oder 3 Kleinbuchstaben
- Hilfreich ist vielleicht die [hier verlinkte Dokumentation](#)