

Theoretische Informatik

[github/bircni](#)

Inhaltsverzeichnis

	Seite
1 Mathematische Grundbegriffe	1
1.1 Mengen	1
1.2 Relationen	2
1.3 Funktionen	2
1.4 Unendliche Mengen	2
2 Grundbegriffe der Graphentheorie	3
2.1 Gerichtete Graphen	3
2.2 Ungerichtete Graphen	4
2.3 Wege, Zyklen und Kreise	6
2.4 Zusammenhang	6
2.5 Euler-Zyklen und -Wege	6
2.6 Hamilton-Kreise	7
2.7 Bäume	7
2.8 Binärbäume	8
3 Formale Sprachen	9
3.1 Alphabete und Wörter	9
3.2 Sprachen	10
3.3 Exkurs: XML	10
4 Reguläre Ausdrücke	12
4.1 Syntax reguläre Ausdrücke	12
4.2 Semantik reguläre Ausdrücke	12
4.3 Äquivalenz regulärer Ausdrücke	13
4.4 Anwendung regulärer Ausdrücke	14
5 Kontextfreie Grammatiken	15
5.1 Kontextfreie Grammatiken	15
5.2 Weitere Formen der Syntaxbeschreibung	16
5.3 Chomsky-Grammatiken	18
5.4 XML-Bäume	18
6 Deterministische endliche Automaten	20
6.1 Definition DEA	20
6.2 Akzeptierte Sprache eines DEA	20

6.3	Minimalautomaten	20
6.4	Äquivalenz von Automaten	23
7	Nichtdeterministische endliche Automaten	24
7.1	NEAs mit ϵ -Übergängen	24
7.2	Transformation von ϵ -NEAs in DEAS	25
8	Reguläre Sprachen	26
8.1	Reguläre Ausdrücke und endliche Automaten	26
8.2	Umwandung von ϵ -NEAs in reguläre Ausdrücke	27
8.3	Chomksy-Typ-3-Grammatiken und endliche Automaten	28
8.4	Klasse der regulären Sprache	29
8.5	Abschlusseigenschaften regulärer Sprachen	29
9	Kellerautomaten	30
9.1	Nichtdeterministische Kellerautomaten	30
9.2	Kellerautomaten und kontextfreie Grammatiken	31

1 Mathematische Grundbegriffe

1.1 Mengen

- Eigenschaften von Mengen

Eine Menge ist eine Sammlung von Elementen, diese können alles mögliche sein.

Wesentliche Eigenschaften von Mengen:

- Mengen haben keine Anordnungsreihenfolge
- Ein Element kann höchstens einmal in einer Menge enthalten sein
Bsp. $a, a, b = a, b$

- Grundmengen von Zahlen

- \mathbb{N} : Natürliche Zahlen
- \mathbb{Z} : Ganze Zahlen
- \mathbb{Q} : Rationale Zahlen
- \mathbb{R} : Reelle Zahlen

- Mengenoperationen

- \emptyset : leere Menge
- $x \in A$: x ist Element der Menge A
- $|A|$: Kardinalität der Menge, bzw. Anzahl der Elemente
- $A \subseteq B$: A ist Teilmenge von B oder gleich der Menge von B
- $A \subset B$: Menge A ist echt in Menge B enthalten
- $A \cup B$: **Vereinigung**: Menge aller Elemente in A oder B
- $A \cap B$: **Schnittmenge**: Menge aller Elemente, sowohl in A als auch in B
- $A \setminus B$: **Differenzmenge**: alle Elemente, die in A aber nicht in B enthalten sind
- $A \times B$: **kartesisches Produkt**: Menge aller Paare, die aus A und B gebildet werden können
- \mathcal{A} : **Potenzmenge**: Menge aller Teilmengen von A

- Kardinalität der Potenzmenge

Für die Potenzmenge \mathcal{M} einer endlichen Menge M gilt: $|\mathcal{M}| = 2^{|M|}$

1.2 Relationen

Eine Relation ist eine Beziehung zwischen Elementen einer Menge.

Eine zweistellige Relation R über einer Grundmenge M ist eine Menge von Paaren (x,y) mit $x \in M$ und $y \in M$, d.h. $R \subseteq M \times M$

1.3 Funktionen

- Eine Funktion $f : A \rightarrow B$ ist eine zweistellige Relation $f \subseteq A \times B$ mit der Eigenschaft:
 $(a, b_1) \in f, (a, b_2) \in f \Rightarrow b_1 = b_2$
- Ist $(a, b) \in f$, dann heißt b Funktionswert zu a , $f(a) = b$
- Eine Funktion $f : A \rightarrow B$ heißt **total**, wenn es für jedes Argument $a \in A$ einen Funktionswert $f(a) = b \in B$ gibt. Sonst heißt die Funktion **partiell**.
- Gibt es für ein $a \in A$ keinen Funktionswert, dann ist f an dieser Stelle **undefiniert**
 $f(a) = \perp$

1.4 Unendliche Mengen

- **abzählbar unendlich**: Gleich mächtig, wie die Menge der natürlichen Zahlen.
Also: für jedes Element gibt es eine Position und für jede Position i gibt es auch einen Wert $m_i \in M$
- **überabzählbar unendlich**: unendlich und nicht abzählbare Menge
Keine 1-zu-1-Zuordnung zu den natürlichen Zahlen
Bsp: Die Menge der unendlich langen 0/1 Folgen, die Potenzmenge der natürlichen Zahlen

2 Grundbegriffe der Graphentheorie

2.1 Gerichtete Graphen

2.1.1 Definition

- Ein gerichteter Graph $G = (V, E)$ besteht aus
 V Menge von Knoten
 $E \subseteq V \times V$ Menge von Kanten
- Für eine Kante $e = (u, v)$ ist u der Ausgangs- und v der Zielknoten
- Existiert eine Kante $e = (u, v)$, dann ist v ein Nachbar von u . Sie sind adjazent.
- Eine Kante mit gleichem Ausgangs- und Zielknoten heißt Schlinge/Schleife.

2.1.2 Diagrammdarstellung von Graphen

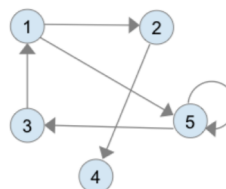
- Knoten werden als Kreise dargestellt.
- Kanten werden als Pfeil vom Ausgangs- zum Zielknoten dargestellt.
- Die Positionierung von Knoten ist irrelevant, Kanten müssen keine geraden Linien sein.

2.1.3 Weitere Darstellungsmöglichkeiten von Graphen

Graphen können als Adjazenzmatrix oder in Adjazenzlisten dargestellt werden:

	1	2	3	4	5
1	0	1	0	0	1
2	0	0	0	1	0
3	1	0	0	0	0
4	0	0	0	0	0
5	0	0	1	0	1

Knoten	Adjazenzliste
1	[2, 5]
2	[4]
3	[1]
4	[]
5	[3, 5]



2.1.4 Knotengrad

Für einen Knoten eines gerichteten Graphen $G = (V, E)$ ist

- der **Eingangsgrad** die Anzahl der Zielknoten v
- der **Ausgangsgrad** die Anzahl der Ausgangsknoten v
- der **Grad** die Summe von Ausgangsgrad und Eingangsgrad von v

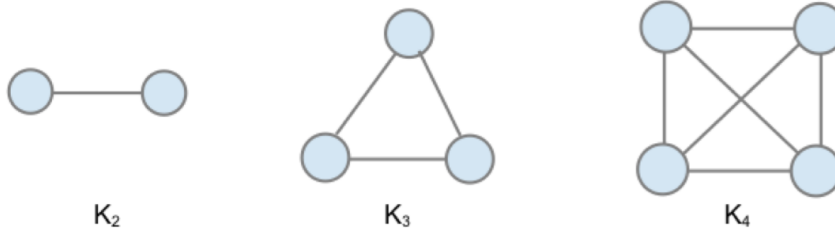
2.2 Ungerichtete Graphen

2.2.1 Definition

- Ein ungerichteter Graph $G = (V, E)$ besteht aus einer Knotenmenge V und einer Kantenmenge E
- Der Grad eines Knotens ist die Anzahl der von v ausgehenden Kanten.
(Schlingen werden doppelt gezählt)

2.2.2 Vollständige Graphen

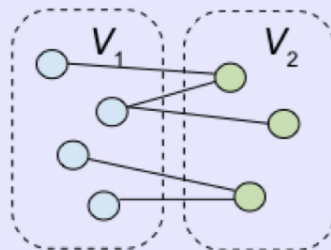
Ein ungerichteter Graph heißt **vollständiger Graph**, wenn es zw. je zwei verschiedenen Knoten eine Kante gibt.



Ein vollständiger Graph mit n Knoten hat $\frac{n(n-1)}{2}$ Kanten.

2.2.3 Bipartite Graphen

- Ein Graph $G = (V, E)$ heißt **bipartit**, wenn die Knotenmenge V in zwei disjunkte Teilmengen V_1, V_2 mit $V = V_1 \cup V_2$ aufgeteilt werden kann, so dass jede Kante zwei Knoten aus verschiedenen Teilmengen verbindet.



- Ein **bipartiter Graph** heißt **vollständig**, wenn es von jedem Knoten aus V_1 eine Kante zu jedem Knoten aus V_2 gibt.

2.2.4 Planare Graphen

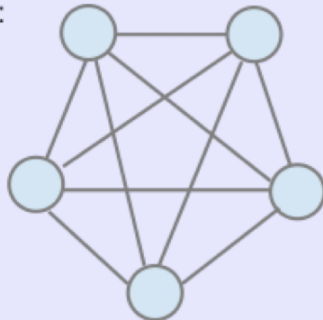
Definition:

Ein Graph G heißt **planar**, wenn er in einer Ebene so gezeichnet werden kann, dass sich keine Kanten kreuzen.

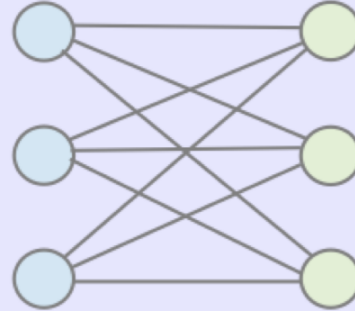
Kuratowski-Graphen

- (1) Der **vollständige Graph K_5** mit 5 Knoten und der **vollständige bipartite Graph $K_{3,3}$** mit zweimal drei Knoten sind **nicht planar**.

K_5 :



$K_{3,3}$:



- (2) Ein endlicher Graph ist genau dann planar, wenn er keinen Teilgraphen erhält, der durch Unterteilung von K_5 oder $K_{3,3}$ entstanden ist. Unterteilung bedeutet hier, dass beliebig oft (auch null mal) neue Knoten auf einer Kante eingefügt werden.

- ▶ Wenn in einem Graphen also irgendwie K_5 oder $K_{3,3}$ als Teil enthalten ist, ist der Graph nicht planar.
- ▶ Folglich sind also beispielsweise alle vollständigen Graphen mit mehr als 5 Knoten auch nicht planar, da immer K_5 als Teilgraph enthalten ist.

2.3 Wege, Zyklen und Kreise

2.3.1 Definition

- Ein **Zyklus** ist ein Weg der Länge $n \geq 0$ - Ausgangs und Endknoten stimmen überein
- Ein **Kreis** ist ein Zyklus, bei dem kein Knoten doppelt besucht wird
- Ein gerichteter Graph heißt **azyklischer Graph**, wenn er keinen Zyklus enthält

2.4 Zusammenhang

2.4.1 Zusammenhang bei ungerichteten Graphen

Ein ungerichteter Graph heißt **zusammenhängend**, wenn es von jedem Knoten zu jedem anderen mind. einen Weg gibt.

Ein Teilgraph des ungerichteten Graphen heißt **Zusammenhangskomponente**, wenn folgende Bedingungen gelten:

- G' ist zusammenhängend
- G hat keinen größeren Teilgraphen G'' , der zusammenhängend ist und G' als Teilgraph enthält

Die Knotenmenge von G' ist eine Teilmenge der Knotenmenge von G

2.4.2 Zusammenhang bei gerichteten Graphen

stark zusammenhängend: für jede Kombination von v_1, v_2 gibt es jeweils einen Weg
(**schwach**) **zusammenhängend**: wenn er als ungerichteter Graph zusammenhängend wäre

2.5 Euler-Zyklen und -Wege

2.5.1 Definition

- Ein **Euler-Weg** für G ist ein Weg in dem jede Kante genau einmal durchlaufen wird
- Ein **Euler-Zyklus** für G ist ein Zyklus in dem jede Kante genau einmal durchlaufen wird
- G hat einen **Euler-Zyklus** genau dann, wenn alle Knoten einen **geraden Grad** haben
- G hat einen **Euler-Weg**, wenn genau zwei Knoten ungeraden Grad haben und alle anderen einen geraden

2.6 Hamilton-Kreise

Ein Zyklus, bei dem jeder Knoten genau einmal besucht wird, nennt man **Hamilton-Kreis**.

2.7 Bäume

2.7.1 Definition

Ein Baum ist ein ungerichteter, zusammenhängender Graph ohne Schlingen.

Alle Knoten mit Grad 1 nennt man **Blätter** des Baums, die anderen heißen **innere Knoten**

Jeder nicht leere Baum mit **n Knoten** hat **n-1 Kanten**

2.7.2 Spannbäume

Ein Spannbaum ist ein Teilgraph eines ungerichteten Graphen, der ein Baum ist und alle Knoten dieses Graphen enthält.

Spannbäume existieren nur in zusammenhängenden Graphen.

2.7.3 Wurzelbäume

Ein gewurzelter Baum hat einen ausgezeichneten Knoten als **Wurzel**

Jeder Knoten kann beliebig viele Nachfolger haben, diese nennt man Kinder

Jeder Knoten hat genau einen Elternknoten

Blätter sind Knoten ohne Kinder, **innere Knoten** sind Knoten mit Kindern

Definition:

Ein gerichteter, zusammenhängender, azyklischer Graph ist ein Wurzelbaum, wenn

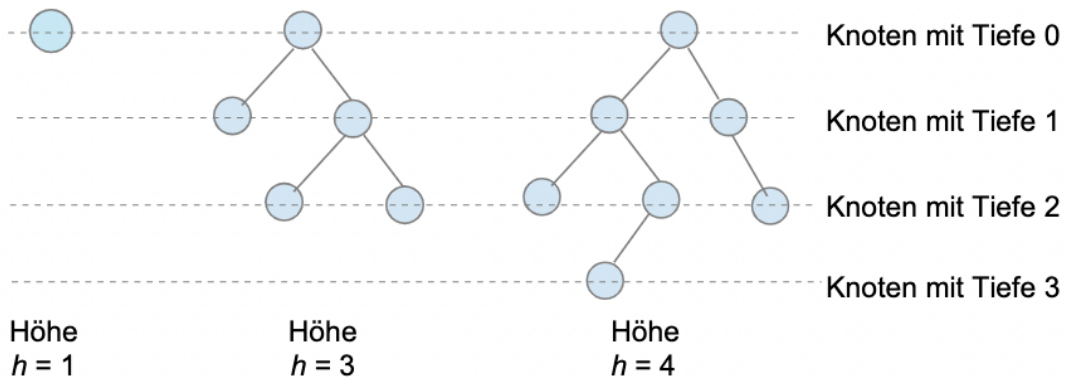
- es genau einen Knoten w mit Eingangsgrad 0 gibt (Wurzel)
- alle anderen Knoten den Eingangsgrad 1 haben

2.7.4 Tiefe von Knoten und Höhe von Bäumen

Die **Tiefe** eines Knoten ist die Länge des Pfades vom Wurzelknoten zum Knoten k

Ein leerer Baum hat die Höhe $h=0$

Ein Baum aus einem einzigen Knoten hat die Höhe $h=1$



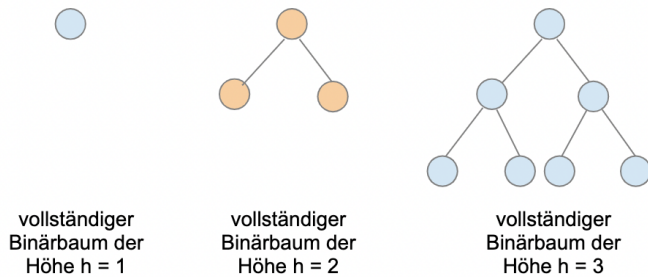
2.8 Binärbäume

Ein Binärbaum ist ein Wurzelbaum, dessen Knoten maximal den Ausgangsgrad 2 haben, d.h. ein Knoten hat maximal 2 Kinder

2.8.1 Vollständige Binärbäume

Ein Binärbaum heißt **vollständig**, wenn

- alle innere Knoten den Ausgangsgrad 2 haben
- alle Blätter die gleiche Tiefe haben



Eigenschaften:

Ein vollständiger Binärbaum der Höhe $h > 0$ hat

- $2^{(h-1)}$ **Blätter**
- $2^h - 1$ **Knoten**

2.8.2 Traversierung von Binärbäumen

- Preorder-Traversierung (W-L-R)
- Inorder-Traversierung (L-W-R)
- Postorder-Traversierung (L-R-W)

3 Formale Sprachen

3.1 Alphabete und Wörter

Ein **Alphabet** Σ ist eine endliche, nicht leere Menge von Zeichen.

3.1.1 Wörter

Definition

- Ein Wort über einem Alphabet ist eine endlich lange Folge von Zeichen aus Σ
- $|w|$ bezeichnet die Länge des Worts w
- ϵ bezeichnet das leere Wort mit Länge 0, $|\epsilon| = 0$
- $|w|_a$ bezeichnet die Anzahl der Vorkommen des Zeichens a in w
- Σ^* bezeichnet die Menge aller Wörter, die mit Zeichen von Σ gebildet werden können

3.1.2 Konkatenation

Die Konkatenation (Aneinanderhängen) von zwei Wörtern wird als $v \cdot w$ (vw) notiert.

- **Assoziativität:**
 $u \cdot (v \cdot w) = (u \cdot v) \cdot w$
- ϵ ist **neutrales Element:**
 $\epsilon \cdot w = w \cdot \epsilon = w$
- Addition der Längen:
 $|v \cdot w| = |v| + |w|$
- **n-fache Konkatenation:**
 $w^0 = \epsilon$
 $w^n = w \cdot w^{n-1}$ für $n > 0$
 $|w^n| = n \cdot |w|$

3.2 Sprachen

Die **Syntax** einer Sprache beschreibt die Regeln wie "Äußerungen" der Sprache gebildet werden können.

Die **Semantik** beschreibt die Bedeutung der formulierbaren Äußerung.

Die **Pragmatik** beschäftigt sich mit der Nutzung der Sprache.

Eine Sprache L über dem Alphabet Σ ist eine Menge von Wörtern über Σ

Einige Sprachen über Alphabet $\Sigma = \{a, b\}$:

- ▶ $L_1 = \{\epsilon, ab, ba\}$ endliche Sprache, besteht nur aus drei Wörtern.
- ▶ $L_2 = \{b, bb, bbb, bbbb, \dots\}$
Sprache beliebig langer Folgen von b , unendliche Sprache
- ▶ $L_3 = \{\}$ leere Sprache, enthält gar kein Wort
- ▶ $L_4 = \{\epsilon\}$ Sprache, die nur das leere Wort enthält
- ▶ $L_5 = \{\epsilon, a, b, aa, bb, aaa, aba, bab, bbb, aaaa, abba, baab, bbbb, \dots\}$

3.2.1 Operationen für Sprachen

Definition

- Die **Konkatenation** $L_1 \cdot L_2$ zweier Sprachen L_1 und L_2 ist
 $L_1 \cdot L_2 = \{vw \mid v \in L_1, w \in L_2\}$
- Für eine Sprache L ist L^n wie folgt definiert:
 $L^0 = \epsilon$
 $L^n = L \cdot L^{n-1}$, für $n > 0$
- Die **Kleene'sche Hülle** L^* einer Sprache L ist definiert durch
 $L^* = L^0 \cup L^1 \cup L^2 \cup \dots$

3.3 Exkurs: XML

Serialisierung = Abbildung strukturierter Daten in eine sequentielle Darstellungsform

- **XHTML**: Beschreibung von Web-Seiten
- **SVG**: zweidimensionale Vektorgrafik
- **ODF**: genormtes Austauschformat für Bürodokumente
- **MathML**: Dokumentenformat zur Darstellung mathematischer Formeln
- **MusicXML**: offenes Dateiformat zum Austausch von Musiknoten

- **WSDL:** Schnittstellen-Beschreibung für Web-Services

3.3.1 Wohlgeformtheit

Ein Text ist ein wohlgeformter XML-Text, wenn er folgende Regeln erfüllt:

- Ein XML-Text besteht aus genau einem XML-Element
- Ein XML-Element beginnt mit einem Anfangstag `<tagi` und endet mit dem gleichnamigen Endtag `>/tagi`
- Elementare Texte können beliebige Zeichenfolgen sein, die aber keine Tags enthalten

3.3.2 Weitere XML-Details

- ▶ Anfangstags können mit Attributen versehen werden, um zusätzliche Informationen mit einem Knoten des Baums zu verbinden.

```
<person alter="24"> ... </person>
```

- ▶ Steht zwischen Anfangstag und Endtag kein Inhalt, kann das Element zu einem Tag mit `< ... />` zusammengefasst werden

```
<image name="hochschule.jpg" />
```

entspricht

```
<image name="hochschule.jpg"></image>
```

- ▶ Eine XML-Datei beginnt immer mit Angaben zu XML-Version, Zeichencode und ggf. weiteren Metainformationen, z.B. Strukturdefinitionen.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE adressBuch SYSTEM "adressBuch.dtd">
...
```

Danach folgt dann das Wurzelement als eigentlicher Inhalt.

4 Reguläre Ausdrücke

4.1 Syntax reguläre Ausdrücke

Die Menge der **regulären Ausdrücke** über einem Alphabet ϵ ist rekursiv so definiert:
Folgendes sind elementare reguläre Ausdrücke:

- (1) \emptyset
- (2) ϵ
- (3) a für jedes Zeichen $a \in \epsilon$

Sind R und S beliebige reguläre Ausdrücke, dann sind auch folgendes reguläre Ausdrücke:

- (4) $R \cdot S$
- (5) $R|S$
- (6) R^*
- (7) (R)

Notationskonvention

- Operator $*$ bindet stärker als Operator \cdot
- Operator \cdot bindet stärker als Operator $|$

4.2 Semantik reguläre Ausdrücke

Die von einem regulären Ausdruck R dargestellte Sprache $L(R)$ ist folgendermaßen definiert:

- (1) $L(\emptyset) = \quad \quad \quad //$ leere Sprache, enthält gar kein Wort
- (2) $L(\epsilon) = \epsilon \quad \quad \quad //$ Sprache, die nur das leere Wort enthält
- (3) $L(a) = a \quad \quad \quad //$ Sprache, die nur das Wort aus dem einzelnen Zeichen a enthält
- (4) Ist $L(R) = L_1$ und $L(S) = L_2$ dann ist $L(R \cdot S) = L_1 \cdot L_2$
- (5) Ist $L(R) = L_1$ und $L(S) = L_2$ dann ist $L(R|S) = L_1 \cup L_2$
- (6) Ist $L(R) = L_1$ dann ist $L(R^*) = L_1^*$

(7) Ist $L(R) = L_1$ dann ist $L((R)) = L_1$

Die Bedeutung zusammengesetzter Ausdrücke kann so erklärt werden:

- $R|S$ ist die Menge aller Wörter, die in der Sprache R oder in der Sprache S enthalten sind. Sprechweise: "R oder S"
- RS ist die Menge aller Wörter, so dass der erste Teil des Worts in der Sprache von R liegt und der Rest in S
- R^* ist die Menge aller Wörter $w = u_1u_2...u_n$ die sich aus bel. vielen Teilwörtern zusammensetzen

R^+ ist die Abkürzung für RR^* $R^?$ ist die Abkürzung für $R|\epsilon$

4.3 Äquivalenz regulärer Ausdrücke

Zwei reguläre Ausdrücke R und S heißen äquivalent, falls $L(R) = L(S)$, d.h. wenn beide Ausdrücke die gleiche Sprache darstellen.

Eigenschaften

Für beliebige reguläre Ausdrücke R, S und T gilt:

- $R|S = S|R$
- $(R|S)|T = R|(S|T)$
- $RS|RT = R(S|T)$
- $RT|ST = (R|S)T$
- $R|R = R$
- $R|\emptyset = \emptyset|R = R$
- $(RS)T = R(ST)$
- $R\epsilon = \epsilon R = R$
- $R\emptyset = \emptyset R = \emptyset$
- $R^{**} = R^*$
- $R^* = \epsilon|RR^*$ bzw. $R^* = \epsilon|R^+$
- $R^* = \epsilon|R^*$

4.4 Anwendung regulärer Ausdrücke

boolean	matches (regex) prüft, ob der String entsprechend dem regulären Ausdruck regex aufgebaut ist.
String	replaceAll (String regex, String replacement) ersetzt alle Vorkommen des Musters regex durch replacement
String	replaceFirst (String regex, String replacement) ersetzt das erste Vorkommen des Musters regex durch replacement
String []	split (String regex, int limit) Splits this string around matches of the given regular expression

5 Kontextfreie Grammatiken

5.1 Kontextfreie Grammatiken

Eine kontextfreie Grammatik $G = (N, T, P, S)$ besteht aus:

- **N** endliche Menge von **nichtterminalen Symbolen**
- **T** endliche Menge von **terminalen Symbolen**
- **P** endliche Menge von **Produktionen** $L \rightarrow r$
- **S** Startsymbol $S \in N$ eines der nichtterminalen Symbole

Beispiel:

- $N = \{A, D, S\}$
- $T = \{a, b, c, d\}$
- Produktionen = $\{S \rightarrow AD, A \rightarrow aAc, A \rightarrow b, D \rightarrow dD, D \rightarrow \epsilon\}$
- Startsymbol S

5.1.1 Ableitbarkeit und Sprache einer Grammatik

Ableitbarkeit:

w ist aus u ableitbar, wenn $u = w$ oder wenn es eine Folge von Ableitungsschritten von u nach w gibt

$*$ = "beliebig viele" Schritte

Die von $G = (N, T, P, S)$ erzeugte Sprache ist: $L(G) = \{w \in T^* | S \Rightarrow w\}$

5.1.2 Links- und Rechtsableitungen

Linksableitung:

In jedem Ableitungsschritt wird das am weitesten links stehende nichtterminale Symbol ersetzt

Rechtsableitung

In jedem Ableitungsschritt wird das am weitesten rechts stehende nichtterminale Symbol ersetzt

5.1.3 Ableitungsbäume

Ein Baum ist ein Ableitungsbaum zu einer kontextfreien Grammatik G für ein Wort w , wenn:

- Die Wurzel des Baums ist mit dem Startsymbol S markiert
- Die inneren Knoten sind mit einem nichtterminalen Symbol markiert
- Die Blätter des Baums sind mit einem terminalen Symbol oder mit ϵ markiert

Ein Wort w ist genau dann aus dem Startsymbol ableitbar, wenn ein Ableitungsbaum für das Wort w existiert. $S \Rightarrow^* w$

5.1.4 Mehrdeutigkeit und Äquivalenz von Grammatiken

Eine kontextfreie Grammatik heißt mehrdeutig, wenn es für mindestens ein Wort mindestens zwei unterschiedliche Ableitungen gibt.

Zwei Grammatiken sind äquivalent, wenn sie dieselbe Sprache erzeugen $L(G_1) = L(G_2)$

5.1.5 Kontextfreie Sprachen

Eine Sprache L ist kontextfrei, wenn es eine kontextfreie Grammatik gibt, die die Sprache darstellt

5.2 Weitere Formen der Syntaxbeschreibung

5.2.1 Backus-Naur-Form (BNF)

Notation:

- In Produktionen: $::=$ statt \rightarrow
- Nichtterminale Symbole werden mit $(...)$ geklammert
- Terminale Symbole werden mit $"..."$ eingeschlossen

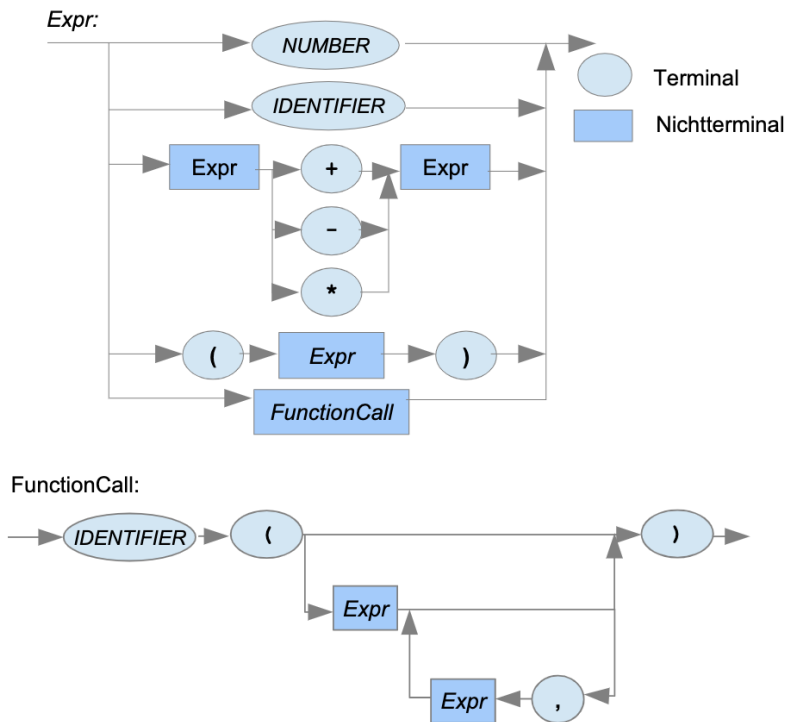
5.2.2 Erweiterte-Backus-Naur-Form (EBNF)

EBNF-Notation	Erläuterung
M^*	Wiederholung: M beliebig oft (auch 0 mal)
M^+	Wiederholung: M ein- oder mehrfach
$M_1 \mid M_2$	Alternative: M_1 oder M_2
$[M]$	Optional: Teil: M kann entfallen
(M)	Klammern zur Strukturierung

Beispiel 6.38 - EBNF-Grammatik

$S \rightarrow (bA \mid aa)^*$
 $A \rightarrow a[Ab]a$

5.2.3 Syntaxdiagramme



5.2.4 Zusammenhang EBNF und Syntaxdiagramme

EBNF-Konstrukt	Bedeutung	Darstellung im Syntaxdiagramm
$M \mid N$	Alternative	
M^*	Wiederholung ≥ 0 mal	
M^+	Wiederholung ≥ 1 mal	
$[M]$	optionaler Teil	

5.3 Chomsky-Grammatiken

5.3.1 Typ-i-Grammatiken

Eine Chomsky-Grammatik des Typs i besteht aus:

- einer endlichen Menge N nichtterminaler Symbole
- einer endlichen Menge T terminaler Symbole ($N \cap T = \emptyset$)
- einem Startsymbol $S \in N$

sowie einer endlichen Menge P von Produktionen des Typs i

- **Typ 0 (uneingeschränkt)**
Produktion hat die Form $u \rightarrow w$ mit
 $u \in (N \cup T)^*$, $w \in (N \cup T)^*$ wobei $|u| \leq |w|$
- **Typ 1 (kontextsensitiv, monoton)**
Produktion hat die Form $u \rightarrow w$ mit
 $u \in (N \cup T)^*$, $w \in (N \cup T)^*$ wobei $|u| \leq |w|$
Zusätzlich ist die Produktion $S \rightarrow \epsilon$ erlaubt, sofern S nicht auf der rechten Seite der Produktion vorkommt.
- **Typ 2 (kontextfrei)**
Produktion hat die Form $A \rightarrow w$ mit
 $A \in N$, $w \in (N \cup T)^*$
- **Typ 3 (rechtslinear)**
Produktionen können entweder die Form $A \rightarrow aB$
oder die Form $A \rightarrow \epsilon$ haben mit $(A, B \in N, a \in T)$

5.4 XML-Bäume

5.4.1 Document Type Definition (DTD)

Eine DTD-Spezifikation hat die Form:

<! DOCTYPE wurzelement[Definition der XML-Elemente]>

Jedes XML-Element wird durch einen Eintrag definiert

<! ELEMENT elementname regAusdruck>

Der Ausdruck "regAusdruck" beschreibt welchen Inhalt das XML-Element haben kann

- *#PCDATA* elementarer Text, enthält keine weiteren XML-Elemente
- *EMPTY* leeres Element
- *A,B,C* Sequenz: A gefolgt von B gefolgt von C
- *A—B—C* alternativ A oder B oder C

- Y^* (evtl. leere) Folge von Y , beliebig oft
- Y^+ nicht-leere Folge von Y , d.h. mind einmal
- $A?$ A ist optional

Ein XML-Dokument heißt **valid**, wenn es die strukturellen Vorgaben der DTD einhält.

6 Deterministische endliche Automaten

6.1 Definition DEA

Ein deterministischer endlicher Automat (DEA) $A = (Z, \Sigma, \delta, z_0, E)$ besteht aus:


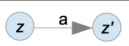

Z endliche Menge von Zuständen

Σ Eingabealphabet

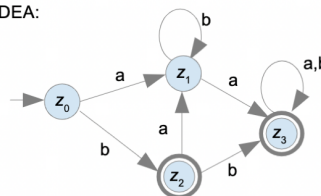
δ Zustandsübergangsfunktion (totale Funktion)

z_0 Startzustand ($z_0 \in Z$)

E Menge von akzeptierenden Zuständen

Darstellung	Bedeutung
	Zustand
	Zustandsübergang $\delta(z, a) = z'$
	Startzustand
	akzeptierender Zustand (Endzustand)

► DEA:



► Zustandsübergangstabelle:

Zustand	Nachfolgezustand	
	a	b
z₀	z ₁	z ₂
z₁	z ₃	z ₁
z₂	z ₁	z ₃
z₃	z ₃	z ₃

6.2 Akzeptierte Sprache eines DEA

Die durch einen DEA $A = (Z, \Sigma, \delta, z_0, E)$ akzeptierte Sprache $L(A)$ ist die Menge $L(A) = \{w \in \Sigma^* \mid z_0 \rightarrow z' \cap z' \in E\}$.

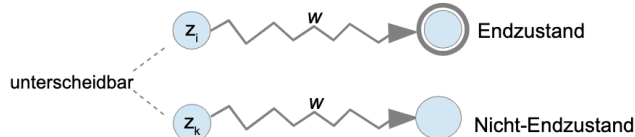
6.3 Minimalautomaten

Ein DEA ist ein Minimalautomat falls der Automat die Sprache L akzeptiert und es keinen anderen Automaten A' gibt, der

- äquivalent zu A ist
- weniger Zustände als A hat

6.3.1 Äquivalenz von Zuständen

- ▶ Zwei Zustände z_1 und z_2 sind **äquivalent**, wenn von beiden Zuständen aus mit genau den gleichen Wörtern Endzustände erreichbar sind (es müssen aber nicht die identischen Endzustände sein!). Hat man zwei äquivalente Zustände dann ist das zukünftige Verhalten bei der Analyse eines Worts für beide Zustände gleich: Es ist egal, ob man von dem einen oder anderen Zustand ausgeht, man wird genau mit den gleichen (Rest-)Wörtern Endzustände erreichen.
- ▶ Zwei Zustände z_1 und z_2 sind somit **unterscheidbar**, wenn es mindestens ein Wort w gibt, so dass von z_1 aus mit w ein Endzustand erreicht werden kann, aber von z_2 aus nicht (bzw. umgekehrt).

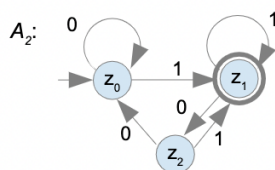


Wir sagen in diesem Fall, dass die Zustände z_1 und z_k *durch das Wort w unterscheidbar* sind.

Sind zwei Zustände unterscheidbar, macht es also einen Unterschied, ob man bei der Analyse eines Worts in den einen oder in den anderen Zustand gelangt.

Beispiel 7.16 - Äquivalente und unterscheidbare Zustände

Welche Zustände sind bei folgendem DEA A_2 zueinander äquivalent bzw. voneinander unterscheidbar?



- ▶ **z_0 und z_1 sind unterscheidbar:** Betrachte Wort ε . Man bleibt mit dem Wort in z_0 bzw. in z_1 . z_1 ist Endzustand, z_0 aber nicht. Also sind z_0 und z_1 durch das Wort ε unterscheidbar, d.h. nicht äquivalent.
- ▶ **z_1 und z_2 sind unterscheidbar:** Gleiche Argumentation gilt hier auch: z_1 ist Endzustand, z_2 aber nicht, also sind z_1 und z_2 durch das Wort ε unterscheidbar, also nicht äquivalent.
- ▶ **z_0 und z_2 sind äquivalent:** Wir betrachten alle möglichen Wörter $w \in \{0,1\}^*$. Es gibt folgende drei Fälle: Es kann das leere Wort ε sein, das Wort kann mit 0 anfangen oder es kann mit 1 anfangen.
 - Fall $w=\varepsilon$: Beide Zustände sind keine Endzustände, somit durch ε nicht unterscheidbar.
 - Fall $w=0w'$: Nach Transition mit 0 befindet sich Automat A_2 in Zustand z_0 , egal ob man bei z_0 oder z_2 gestartet ist. Danach geht es für w' somit gleich weiter, somit sind beide Zustände durch keine Wort $0w'$ unterscheidbar.
 - Fall $w=1w'$: nach Transition mit 1 befindet sich Automat A_2 im Zustand z_1 , egal ob man bei z_0 oder z_2 gestartet ist. Danach geht es für w' folglich gleich weiter. Somit sind beide Zustände durch keine Wort $1w'$ unterscheidbar.

Die Zustände z_0 und z_2 sind durch kein Wort $w \in \{0,1\}^*$ unterscheidbar, also sind sie äquivalent, d.h. $z_0 \equiv z_2$.

6.3.2 Berechnung äquivalenter Zustände

Ausgangspunkt: Wir bilden eine Tabelle in Dreiecksform für alle Paare $\{z_i, z_k\}$ von Zuständen. Jeder Zustand ist zu sich selbst äquivalent, bei allen anderen Paaren ist noch unklar, ob sie unterscheidbar oder äquivalent sind.

z_0	\equiv				
z_1		\equiv			
z_2			\equiv		
z_3				\equiv	
z_4					\equiv
	z_0	z_1	z_2	z_3	z_4

In diese Tabelle wird eingetragen, ob schon bekannt ist, ob zwei Zustände unterscheidbar oder äquivalent sind.

Eintrag bei $\{z_i, z_k\}$:

- \equiv (für $i = k$): jeder Zustand ist zu sich selbst äquivalent
- X z_i und z_k sind unterscheidbar
- leer es ist noch unbekannt, ob z_i und z_k unterscheidbar sind

Vorgehensweise:

1. Phase (Endzustand/Nicht-Endzustand): Markiere alle Paare $\{z_i, z_k\}$, bei denen entweder z_i oder z_k Endzustand ist, aber nicht beide, mit X als unterscheidbar.

2. Phase (Propagierungsregel anwenden):

Wiederhole für jedes noch nicht markierte Paare $\{z_i, z_k\}$:

Wiederhole für alle Zeichen $a \in \Sigma$:

Falls $z_i \xrightarrow{a} z_i'$,

$z_k \xrightarrow{a} z_k'$

und $\{z_i', z_k'\}$ in Tabelle schon mit X markiert

dann markiere auch $\{z_i, z_k\}$ mit X als unterscheidbar.

so lange, bis sich in einem kompletten Durchlauf durch alle leeren Einträge keine neue Markierung mehr ergibt.

Ergebnis: Für alle am Ende nicht markierten Paare $\{z_i, z_j\}$ gilt, dass sie nicht unterscheidbar und somit äquivalent sind, d.h. $z_i \equiv z_j$.

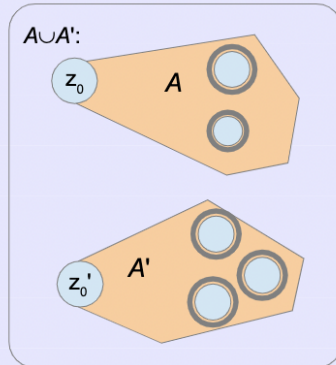
6.4 Äquivalenz von Automaten

Zwei DEAs A_1 und A_2 heißen äquivalent, wenn beide die gleiche Sprache akzeptieren, d.h. $L(A_1) = L(A_2)$

Äquivalenznachweis für DEAs

Zwei DEAs A und A' seien gegeben.

- (3) Bilde einen gemeinsamen Automaten $A \cup A'$ durch disjunkte Vereinigung der Zustände von A und A' (d.h. gleich benannte Zustände ggf. dabei umbenennen).



- (4) Berechne die äquivalenten Zustände von $A \cup A'$ mit Algorithmus 7.19.
- (5) Ergibt sich, dass die Startzustände z_0 von A und z_0' von A' zueinander äquivalent sind, dann sind auch die Automaten A und A' äquivalent, da von beiden Zuständen aus mit genau den gleichen Wörtern Endzustände erreicht werden können.

7 Nichtdeterministische endliche Automaten

7.1 NEAs mit ϵ -Übergängen

Ein nichtdeterministischer endlicher Automat mit ϵ -Übergängen

$$A = (Z, \Sigma, \Delta, z_0, E)$$

Z endliche Zustandsmenge

Σ Eingabealphabet

$z_0 \in Z$ Startzustand

Δ Zustandsübergangsrelation

E Menge von akzeptierenden Zuständen

7.1.1 Sprache eines ϵ -NEA

Für einen ϵ -NEA $A = (Z, \Sigma, \Delta, z_0, E)$ ist die **auf Wörter erweiterte Zustandsübergangsrelation**

$$\rightarrow \subseteq Z \times \Sigma^* \times Z$$

folgendermaßen definiert:

- Mit dem leeren Wort ϵ kann man den gleichen Zustand erreichen, d.h. für alle Zustände $z \in Z$ gilt

$$z \xrightarrow{\epsilon} z$$

- Gibt die Zustandsübergangsrelation Δ an, dass der Automat mit Zeichen a von einem Zustand z_1 nach z_2 wechseln kann und kann er mit dem Wort w von z_2 aus Zustand z_3 erreichen, dann kann der Automat mit Wort aw von z_1 aus Zustand z_3 erreichen:

$$\text{falls } (z_1, a, z_2) \in \Delta \text{ und } z_2 \xrightarrow{w} z_3, \text{ dann } z_1 \xrightarrow{aw} z_3$$

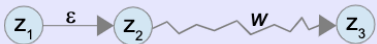
(für alle $z_1, z_2, z_3 \in Z, a \in \Sigma, w \in \Sigma^*$)



- Gibt die Zustandsübergangsrelation Δ an, dass der Automat mit ϵ -Übergang von z_1 nach z_2 wechseln kann, und kann er mit dem Wort w von z_2 den Zustand z_3 erreichen, dann kann man mit Wort w auch von z_1 nach z_3 kommen:

$$\text{falls } (z_1, \epsilon, z_2) \in \Delta \text{ und } z_2 \xrightarrow{w} z_3, \text{ dann } z_1 \xrightarrow{w} z_3$$

(für alle $z_1, z_2, z_3 \in Z, w \in \Sigma^*$)



7.2 Transformation von ϵ -NEAs in DEAS

7.2.1 Teilmengenkonstruktion

Gegeben ist ein ϵ -NEA

$$A_N = (Z_N, \Sigma, \Delta, z_{N0}, E_N).$$

Der zugehörige äquivalente DEA

$$A_D = (Z_D, \Sigma, \delta_D, z_{D0}, E_D)$$

wird folgendermaßen definiert:

- **Zustandsmenge Z_D :** Menge aller Teilmengen von Z_N (= Potenzmenge von Z_N).
- **Zustandübergangsfunktion δ_D :** Ist $X = \{z_1, \dots, z_k\}$ ein Zustand des DEA, dann ist $\delta_D(X, a)$ die Menge aller ϵ -NEA-Zustände, die ausgehend von einem der Zustände z_i durch einen direkten a -Übergang und dann beliebig viele ϵ -Übergänge im ϵ -NEA erreichbar sind.
- **Startzustand z_{D0} :** Menge aus dem Startzustand z_{N0} und allen Zuständen, die über ϵ -Übergänge von z_{N0} aus erreichbar sind.
- **Endzustände E_D :** Allen Zustandsmengen, die mindestens einen ϵ -NEA-Endzustand enthalten, sind Endzustände des DEA.

Ist A_D der gebildete DEA zu einem ϵ -NEA A_N , dann gilt $L(A_D) = L(A_N)$, beide Automaten akzeptieren die gleiche Sprache.

7.2.2 Verbesserte Teilmengenkonstruktion

Gegeben ist ein ϵ -NEA $A_N = (Z_N, \Sigma, \Delta, z_{N0}, E_N)$. Der zugehörige äquivalente DEA $A_D = (Z_D, \Sigma, \delta_D, z_{D0}, E_D)$ wird folgendermaßen konstruiert:

1. Initialisierung: Bestimme Startzustand X_{D0} :

X_{D0} besteht aus Startzustand z_{N0} und allen Zuständen, die über ϵ -Übergänge von z_{N0} aus direkt oder indirekt erreichbar sind.

Setze $Z_D = \{X_{D0}\}$, X_{D0} ist unmarkiert.

2. Verarbeitung der noch unmarkierten Zustände:

Wiederhole, solange noch ein unmarkierter Zustand in Z_D vorhanden ist:

(1) Wähle einen unmarkierten Zustand X aus Z_D aus.

(2) Für jedes Zeichen $a \in \Sigma$ berechne den Nachfolgezustand:

$X' = \{z' \in Z_N \mid z' \text{ ist über erst einen } a\text{-Übergang und dann ggf. beliebig viele } \epsilon\text{-Übergänge danach von einem Zustand } z \in X \text{ erreichbar}\}$

Nimm X' zu Z_D als unmarkierten Zustand dazu, falls noch nicht enthalten und setze

$$\delta(X, a) = X'$$

(3) Markiere Zustand X als fertig.

3. Endzustände: Alle Mengen aus Z_D , die mindestens einen Endzustand des ϵ -NEA enthalten, sind Endzustände des DEA.

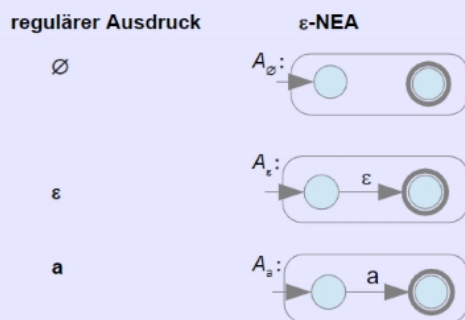
8 Reguläre Sprachen

8.1 Reguläre Ausdrücke und endliche Automaten

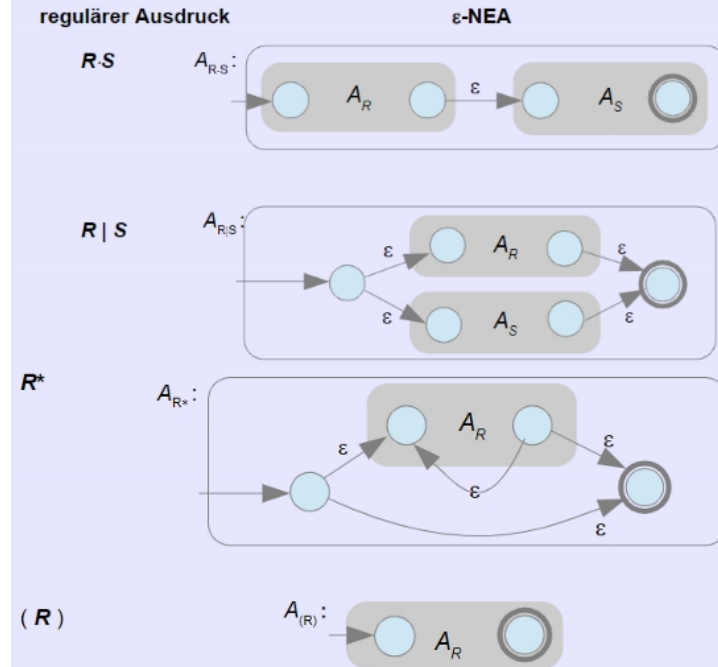
8.1.1 Umwandlung regulärer Ausdrücke in ϵ -NEAs

Induktive Konstruktion eines ϵ -NEA zu regulärem Ausdruck

- Für elementare reguläre Ausdrücke kann ein Automat direkt angegeben werden:



- Für zusammengesetzte reguläre Ausdrücke wird der Automat aus den Automaten für die Teilausdrücke gebildet (hier als graue Fläche dargestellt)



Jeder konstruierte Automat hat genau einen Endzustand.
Der Startzustand hat keine eingehenden Zustandsübergänge.
Der Endzustand hat keine ausgehenden Zustandsübergänge.

8.1.2 Strukturelle Induktion

Induktive Definition von Mengen:

gegeben:

- Eine endliche Menge $G = \{a_1, a_2, \dots, a_n\}$ von Grundelementen
- Eine endliche Menge $R = \{r_1, \dots, r_k\}$ von Konstruktionsregeln
Eine induktiv definierte Menge besteht aus allen Elementen, die durch Konstruktionssschritte konstruiert werden können.

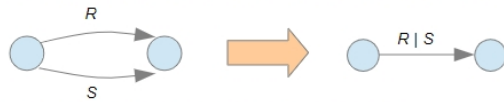
8.2 Umwandlung von ϵ -NEAs in reguläre Ausdrücke

Zu jedem ϵ -NEA kann ein regulärer Ausdruck konstruiert werden.

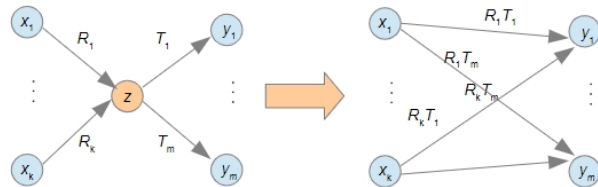
8.2.1 Eliminationsregeln & Eliminationsverfahren

Eliminationsregel 1: Parallele Zustandsübergänge zusammenlegen

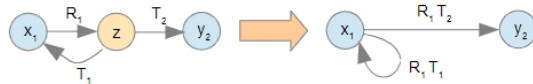
- ▶ Parallele Zustandsübergänge, die mit regulären Ausdrücken R bzw. S markiert sind, können zu einem Übergang mit $R|S$ zusammengefasst werden.



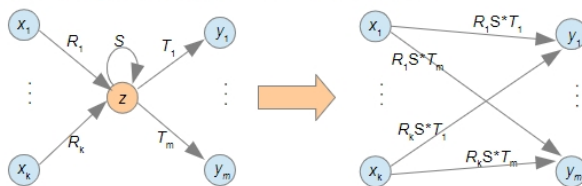
Eliminationsregel 2a: Zustand ohne Schleife eliminieren



- ▶ Jede Kombination von eingehendem und ausgehendem Zustandsübergang muss durch einen neuen Zustandsübergang abgedeckt werden, wenn z eliminiert wird.
- ▶ Ein Zielzustand y_j darf auch mit einem Ausgangszustand x_i identisch sein, z.B. wie hier:



Eliminationsregel 2b: Zustand mit Schleife eliminieren



Eliminationsverfahren 9.15 - ϵ -NEA in regulären Ausdruck umwandeln

1. Vorbereitung:



- Der Automat wird um einen neuen Startzustand z_0 und einen neuen Endzustand z_ϵ erweitert.
 - Ein ϵ -Übergang von z_0 zum ursprünglichen Startzustand wird eingeführt.
 - z_ϵ wird einziger Endzustand. Es werden ϵ -Übergänge von jedem ursprünglichen Endzustand nach z_ϵ eingeführt. Die ursprünglichen Endzustände sind nicht mehr Endzustände.
2. Die Eliminationsregeln 1, 2a und 2b (s.o.) werden solange angewendet, bis alle Zustände außer den neuen Zuständen z und z' eliminiert sind und keine parallelen Übergänge mehr zusammengelegt werden können.

Ergebnis: Es gibt nur zwei Möglichkeiten, wie der Automat am Ende aussehen kann:

- ▶ Es gibt einen Zustandsübergang, der mit einem regulären Ausdruck R markiert ist.



Der reguläre Ausdruck R beschreibt dann die Sprache des Automaten.

- ▶ Es gibt keinen Zustandsübergang zwischen Start- und Endzustand:



Das ist dann der Fall, wenn der Automat gar nichts akzeptiert. In diesem Fall ist \emptyset der reguläre Ausdruck, der die Sprache des Automaten darstellt.

8.3 Chomsky-Typ-3-Grammatiken und endliche Automaten

ϵ -NEA für Chomsky-Typ-3-Grammatiken bilden

Der ϵ -NEA wird in folgender Weise definiert:

- Zustände: Menge der nichtterminalen Symbole der Grammatik
- Startzustand: Startsymbol der Grammatik
- Für jede Produktion $A \rightarrow aB$ wird ein entsprechender Zustandsübergang definiert
- Gibt es eine Produktion $A \rightarrow \epsilon$, dann ist der Zustand A ein Endzustand

Definition: Chomsky-Typ-3-Grammatik

- Als Zustände des DEA werden als nichtterminale Symbole der Grammatik verwendet

- Der Startzustand ist das Startsymbol
- Für jeden Zustandsübergang $\delta(z_1, a) = z_2$ wird eine Produktion $z_1 \rightarrow az_2$ gebildet
- Für jeden Endzustand z wird eine ϵ -Produktion eingeführt: $z \rightarrow \epsilon$

8.4 Klasse der regulären Sprache

Definition:

Eine Sprache L heißt regulär, wenn es einen regulären Ausdruck R gibt, der die Sprache L beschreibt, d.h. für den $L = L(R)$ gilt.

Eigenschaft:

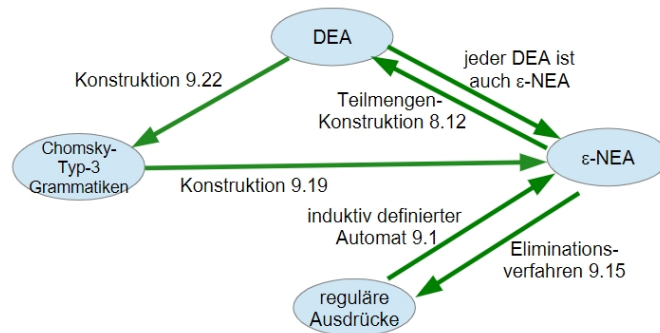
Klasse der **regulären Sprachen**

= Klasse der von **regulären Ausdrücken** dargestellten Sprachen

= Klasse der von **DEAs** akzeptierten Sprachen

= Klasse der von **ϵ -NEAs** akzeptierten Sprachen

= Klasse der durch **Chomsky-Typ-3-Grammatiken** generierbaren Sprachen

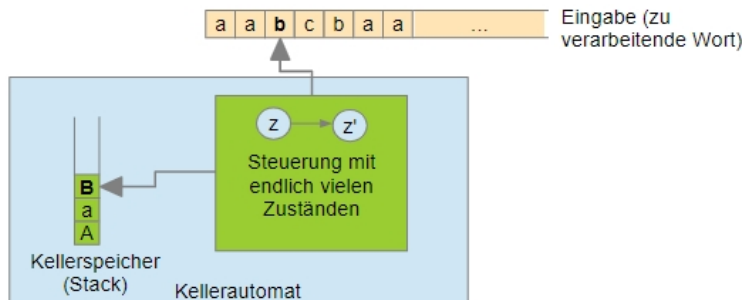


8.5 Abschlusseigenschaften regulärer Sprachen

Sind L_1 und L_2 reguläre Sprachen, dann sind auch folgende Sprachen regulär:

$L_1 \cup L_2$	Vereinigung der Sprachen
$L_1 \cdot L_2$	Konkatenation der Sprachen
L_1^*	Kleene'sche-Hülle der Sprache
$L_1 \cap L_2$	Durchschnitt der Sprachen
$\overline{L_1}$	Komplement der Sprache (alle Wörter, die nicht in L_1 enthalten sind)

9 Kellerautomaten



- ▶ Der Kellerspeicher (Stack) kann eine unbegrenzte Folge von Einträgen (Kellersymbolen) speichern.
- ▶ Die Steuerung hat endlich viele Zustände. Die Steuerung legt den nächsten Ausführungsschritt fest, abhängig von:
 - aktuellem Zustand,
 - oberstem Symbol auf dem Keller und
 - ggf. nächstem Symbol der Eingabe. ϵ -Schritte unabhängig von der Eingabe sind auch möglich.
- ▶ Ein Ausführungsschritt bewirkt, dass
 - in einen Nachfolgezustand gewechselt wird,
 - zum nächsten Symbol der Eingabe weiter gegangen wird, sofern es kein ϵ -Schritt ist, und
 - der oberste Kellereintrag entfernt und durch eine Folge neuer Einträge (die auch leer sein kann) ersetzt wird.

9.1 Nichtdeterministische Kellerautomaten

Definition:

Ein (nichtdeterministischer) **Kellerautomat** $K = (Z, \Sigma, \Gamma, \Delta, z_0, k_0, E)$ ist definiert durch:

Z	endliche Menge von Zuständen
Σ	endliches Eingabealphabet
Γ	endliches Kelleralphabet
Δ	Zustandsübergangsrelation
$z_0 \in Z$	Startzustand
$k_0 \in \Gamma$	Kellerstartsymbol
$E \subseteq Z$	Menge der Endzustände

9.1.1 Sprache eines Kellerautomaten

Definition:

Die vom Kellerautomat K **akzeptierte Sprache** $L(K)$ ist
 $L(K) = \{w \in \Sigma^* \mid K \text{ akzeptiert } w\}$.

9.2 Kellerautomaten und kontextfreie Grammatiken