Übung "Grundbegriffe der Informatik"

11. Übung

Karlsruher Institut für Technologie

Matthias Janke, Gebäude 50.34, Raum 249
email: matthias.janke ät kit.edu

Matthias Schulz, Gebäude 50.34, Raum 247
email: schulz ät ira.uka.de

Besondere Zustände

Akzeptor $A = (Z, z_0, X, f, F)$

Der Endzustand F, Müllzustand J

- $\forall x \in X \forall z \in F : f(z, x) \in F$
- $J \cap F = \emptyset \wedge \forall x \in X \forall z \in J : f(z, x) \in J$

Besondere Zustände

Akzeptor $A = (Z, z_0, X, f, F)$

Der Endzustand F, Müllzustand J

- $\forall x \in X \forall z \in F : f(z, x) \in F$
  $\rightarrow$ Zustand aus F irgendwann erreicht $\Rightarrow$ Wort wird akzeptiert.
- $J \cap F = \emptyset \wedge \forall x \in X \forall z \in J : f(z, x) \in J$
  $\rightarrow$ Zustand aus J irgendwann erreicht $\Rightarrow$ Wort wird abgelehnt. (Müllzustände)

Besondere Zustände

Beispiel zur Verwendung des Müllzustandes:

Die Sprache L_3 ist definiert als die Menge aller Wörter w über dem Alphabet $\{a, b\}$, für die gilt:

- $N_a(w) = N_b(w)$.
- Für alle Präfixe v von w gilt: $N_a(v) \geq N_b(v)$ und $N_a(v) - N_b(v) \leq 3$.

Besondere Zustände

Die Sprache L_3 ist definiert als die Menge aller Wörter w über dem Alphabet $\{a, b\}$, für die gilt:

- $N_a(w) = N_b(w)$.
- Für alle Präfixe v von w gilt: $N_a(v) \geq N_b(v)$ und $N_a(v) - N_b(v) \leq 3$.

Beispiele für Worte der Sprache L_3: $ababab$,ϵ,$baabab$,$aabaabbb$

Besondere Zustände

Die Sprache L_3 ist definiert als die Menge aller Wörter w über dem Alphabet $\{a, b\}$, für die gilt:

- $N_a(w) = N_b(w)$.
- Für alle Präfixe v von w gilt: $N_a(v) \geq N_b(v)$ und $N_a(v) - N_b(v) \leq 3$.

Beispiele für Worte der Sprache L_3: $ababab$,ϵ,$baabab$,$aabaabbb$

Besondere Zustände

Die Sprache L_3 ist definiert als die Menge aller Wörter w über dem Alphabet $\{a, b\}$, für die gilt:

- $N_a(w) = N_b(w)$.
- Für alle Präfixe v von w gilt: $N_a(v) \geq N_b(v)$ und $N_a(v) - N_b(v) \leq 3$.

Beispiele für Worte der Sprache L_3: $ababab$,ϵ,$baabab$,$aabaabbb$

Besondere Zustände

Die Sprache L_3 ist definiert als die Menge aller Wörter w über dem Alphabet $\{a, b\}$, für die gilt:

- $N_a(w) = N_b(w)$.
- Für alle Präfixe v von w gilt: $N_a(v) \geq N_b(v)$ und $N_a(v) - N_b(v) \leq 3$.

Beispiele für Worte der Sprache L_3: $ababab$,ϵ,$baabab$,$aabaabbb$

Besondere Zustände

Die Sprache L_3 ist definiert als die Menge aller Wörter w über dem Alphabet $\{a, b\}$, für die gilt:

- $N_a(w) = N_b(w)$.
- Für alle Präfixe v von w gilt: $N_a(v) \geq N_b(v)$ und $N_a(v) - N_b(v) \leq 3$.

Beispiele für Worte der Sprache L_3: $ababab$,ϵ,$baabab$,$aabaabbb$

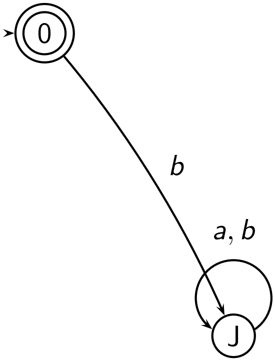
Besondere Zustände

- $N_a(w) = N_b(w)$.
- Für alle Präfixe v von w gilt: $N_a(v) \geq N_b(v)$ und $N_a(v) - N_b(v) \leq 3$.

Zweite Bedingung verhindert, dass mit b begonnen werden darf.

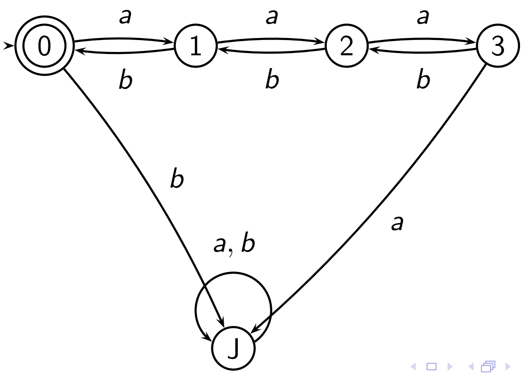
Besondere Zustände

- $N_a(w) = N_b(w)$.
- Für alle Präfixe v von w gilt: $N_a(v) \geq N_b(v)$ und $N_a(v) - N_b(v) \leq 3$.



Besondere Zustände

- $N_a(w) = N_b(w)$.
- Für alle Präfixe v von w gilt: $N_a(v) \geq N_b(v)$ und $N_a(v) - N_b(v) \leq 3$.



Reguläre Ausdrücke

Von formaler Sprache zu regulärem Ausdruck.

$$L = \{\mathtt{a}^k\mathtt{b}^m \mid k, m \in \mathbb{N}_0 \land k \bmod 2 = 0 \land m \bmod 3 = 1\}$$

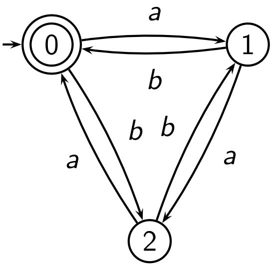
Geben Sie für L einen regulären Ausdruck R_L an mit $\langle R_L \rangle = L$.

Von formaler Sprache zu regulärem Ausdruck.

$$L = \{\mathtt{a}^k\mathtt{b}^m \mid k, m \in \mathbb{N}_0 \wedge k \bmod 2 = 0 \wedge m \bmod 3 = 1\}$$

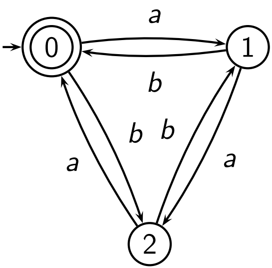
Geben Sie für L einen regulären Ausdruck R_L an mit $\langle R_L \rangle = L$.
Lösung: $(\mathtt{aa}) * \mathtt{b}(\mathtt{bbb})*$

Akzeptoren $\leftrightarrow$ Reguläre Ausdrücke



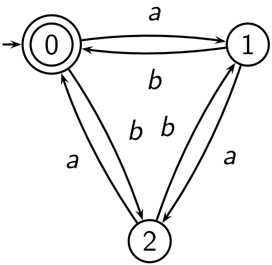
Regulärer Ausdruck für $L(A)$?

Akzeptoren $\leftrightarrow$ Reguläre Ausdrücke



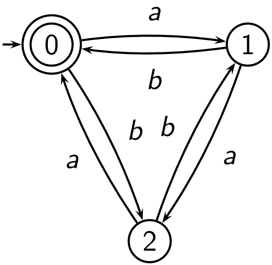
1. $F = \{z_0\} \Rightarrow R = (R')*$

Akzeptoren $\leftrightarrow$ Reguläre Ausdrücke



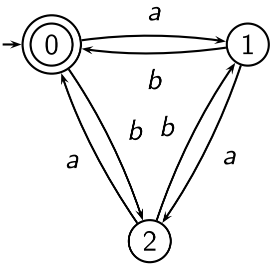
2. Erstes Zeichen $a \to$ 1. Zustand 1

Akzeptoren $\leftrightarrow$ Reguläre Ausdrücke



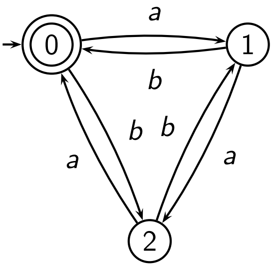
2. Erstes Zeichen $a \rightarrow$ 1. Zustand 1
Danach beliebig oft zwischen 1 und 2 hin und her $\rightarrow (ab)*$

Akzeptoren $\leftrightarrow$ Reguläre Ausdrücke



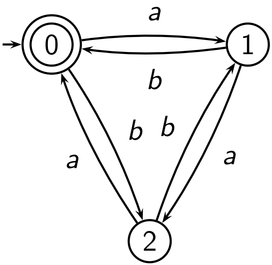
2. Erstes Zeichen $a \rightarrow$ 1. Zustand 1
Danach beliebig oft zwischen 1 und 2 hin und her $\rightarrow (ab)*$
Dann mit b oder aa zurück nach 0.

Akzeptoren $\leftrightarrow$ Reguläre Ausdrücke



3. Erstes Zeichen $b \rightarrow$ 1. Zustand 2
Danach beliebig oft zwischen 2 und 1 hin und her $\rightarrow (ba)*$
Dann mit a oder bb zurück nach 0.

Akzeptoren $\leftrightarrow$ Reguläre Ausdrücke



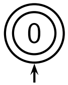
4. Zusammensetzen: $R = (a(ab) * (b \mid aa) \mid b(ba) * (a \mid bb))*$

Akzeptoren $\leftrightarrow$ Reguläre Ausdrücke

Rückwärts: $R = (a(ab) * (b \mid aa) \mid b(ba) * (a \mid bb))*$
Akzeptor konstruieren.

Akzeptoren $\leftrightarrow$ Reguläre Ausdrücke

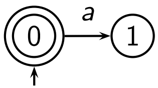
Rückwärts: $R = (a(ab)*(b \mid aa) \mid b(ba)*(a \mid bb))*$
Akzeptor konstruieren.



1. $R = (R')*$, also ist Anfangszustand akzeptierend.

Akzeptoren $\leftrightarrow$ Reguläre Ausdrücke

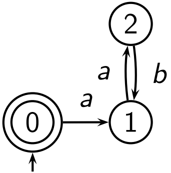
Rückwärts: $R = (a(ab)*(b \mid aa) \mid b(ba)*(a \mid bb))*$
Akzeptor konstruieren.



2. Mit a lande ich in anderem Zustand.

Akzeptoren $\leftrightarrow$ Reguläre Ausdrücke

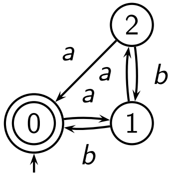
Rückwärts: $R = (\mathbf{a}(ab) * (b \mid aa) \mid b(ba) * (a \mid bb))*$
Akzeptor konstruieren.



3. Mit *ab* komme ich in Zustand 1 zurück, also Zwischenzustand 2 einfügen.

Akzeptoren $\leftrightarrow$ Reguläre Ausdrücke

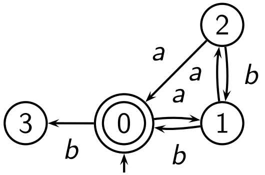
Rückwärts: $R = (a(\mathbf{ab})*(b \mid aa) \mid b(ba)*(a \mid bb))*$
Akzeptor konstruieren.



4. Nach 0 komme ich danach mit b oder aa (über gleichen Zwischenzustand).

Akzeptoren $\leftrightarrow$ Reguläre Ausdrücke

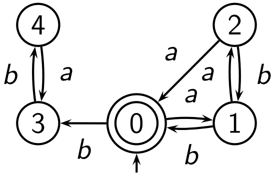
Rückwärts: $R = (a(ab)*(b \mid aa)|b(ba)*(a \mid bb))*$
Akzeptor konstruieren.



5. Mit b als erstem Zeichen komme ich in neuen Zustand.

Akzeptoren $\leftrightarrow$ Reguläre Ausdrücke

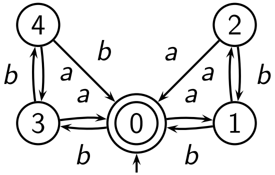
Rückwärts: $R = (a(ab)*(b \mid aa) \mid \mathbf{b}(ba)*(a \mid bb))*$
Akzeptor konstruieren.



6. Mit *ba* komme ich nach 3 zurück über Zustand 4.

Akzeptoren $\leftrightarrow$ Reguläre Ausdrücke

Rückwärts: $R = (a(ab)*(b \mid aa) \mid b(\mathbf{ba})*(a \mid bb))*$
Akzeptor konstruieren.



7. Mit *a* oder *bb* komme ich nach 0 zurück.

Akzeptoren $\leftrightarrow$ Reguläre Ausdrücke

Akzeptor konstruieren: Jeder Zustand entspricht

"Menge an Stellen im Regulären Ausdruck,
an denen man bei Zusammensetzung von w sein kann."

$R = (aab \mid ab)*$

$z_0 =$ Anfang
$z_1 = f(z_0, a) =$ Erstes oder Drittes a im regulären Ausdruck
$z_2 = f(z_1, a) =$ Zweites a
$\ldots$

Akzeptoren $\leftrightarrow$ Reguläre Ausdrücke

Idee für reguläre Ausdrücke:

Zustände des Akzeptors durchnummerieren.

$\langle R_{ij}^k \rangle$ sei Menge aller Wörter w, so dass man von i bei Eingabe von w nach j kommt und dabei nur Zustände aus $\mathbb{G}_k$ durchläuft.

Akzeptoren $\leftrightarrow$ Reguläre Ausdrücke

Idee für reguläre Ausdrücke:

Zustände des Akzeptors durchnummerieren.

$\langle R_{ij}^k \rangle$ sei Menge aller Wörter w, so dass man von i bei Eingabe von w nach j kommt und dabei nur Zustände aus $\mathbb{G}_k$ durchläuft.

R_{ij}^0 sind alle einfach.

Akzeptoren $\leftrightarrow$ Reguläre Ausdrücke

Idee für reguläre Ausdrücke:

Zustände des Akzeptors durchnummerieren.

$\langle R_{ij}^k \rangle$ sei Menge aller Wörter w, so dass man von i bei Eingabe von w nach j kommt und dabei nur Zustände aus $\mathbb{G}_k$ durchläuft.

R_{ij}^{k+1}: Gehe von i nach k über Zustände aus $\mathbb{G}_k$.
Gehe beliebig oft von k nach k über Zustände aus $\mathbb{G}_k$.
Gehe von k nach j über Zustände aus $\mathbb{G}_k$.

Oder gehe direkt von i nach j über Zustände aus $\mathbb{G}_k$.

Akzeptoren $\leftrightarrow$ Reguläre Ausdrücke

Idee für reguläre Ausdrücke:

Zustände des Akzeptors von 0 bis $n-1$ durchnummerieren.

$\langle R_{ij}^k \rangle$ sei Menge aller Wörter w, so dass man von i bei Eingabe von w nach j kommt und dabei nur Zustände aus $\mathbb{G}_k$ durchläuft.

$R_{ij}^{k+1} = R_{ik}^k (R_{kk}^k) * R_{kj}^k \mid R_{ij}^k$

Sei 0 Anfangszustand und $j_0, \ldots, j_m$ akzeptierende Zustände.

Dann ist $R = R_{0j_0}^n \mid \ldots \mid R_{0j_m}^n$.

Akzeptoren $\leftrightarrow$ Rechtslineare Grammatiken (RLG)

$A = (Z, z_0, X, f, F)$.

Idee 1: $G = (Z, X, z_0, P)$ so dass gilt:

$z_0 \Rightarrow^* wz \iff f^*(z_0, w) = z$.

Akzeptoren $\leftrightarrow$ Rechtslineare Grammatiken (RLG)

$A = (Z, z_0, X, f, F)$.

Idee 1: $G = (Z, X, z_0, P)$ so dass gilt:

$z_0 \Rightarrow^* wz \iff f^*(z_0, w) = z$.

Also: $z_0 \Rightarrow^* wz \Rightarrow wxf(z, x)$ muss Ableitung sein.

Akzeptoren $\leftrightarrow$ Rechtslineare Grammatiken (RLG)

$A = (Z, z_0, X, f, F)$.

Idee 1: $G = (Z, X, z_0, P)$ so dass gilt:

$z_0 \Rightarrow^* wz \iff f^*(z_0, w) = z$.

Also: $z_0 \Rightarrow^* wz \Rightarrow wxf(z, x)$ muss Ableitung sein,
also $\forall z \in Z \forall x \in X : z \to xf(z, x)$ muss Produktion sein.

Akzeptoren ↔ Rechtslineare Grammatiken (RLG)

$A = (Z, z_0, X, f, F)$.

Idee 2: Ableitung $z_0 \Rightarrow^* wz$ soll mit w enden **können**, falls $z \in F$ gilt.

Akzeptoren $\leftrightarrow$ Rechtslineare Grammatiken (RLG)

$A = (Z, z_0, X, f, F)$.

Idee 2: Ableitung $z_0 \Rightarrow^* wz$ soll mit w enden **können**, falls $z \in F$ gilt.

Also $z_0 \Rightarrow^* wz \Rightarrow w$ soll möglich sein, wenn $z \in F$ gilt.

Akzeptoren $\leftrightarrow$ Rechtslineare Grammatiken (RLG)

$A = (Z, z_0, X, f, F)$.

Idee 2: Ableitung $z_0 \Rightarrow^* wz$ soll mit w enden **können**, falls $z \in F$ gilt.

Also $z_0 \Rightarrow^* wz \Rightarrow w$ soll möglich sein, wenn $z \in F$ gilt.

Also $z \to \epsilon$ soll Produktion sein, falls $z \in F$ gilt.

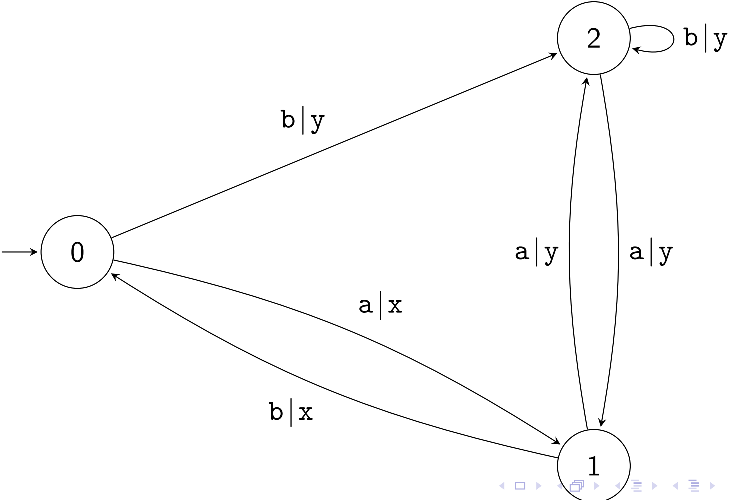
Akzeptoren $\leftrightarrow$ Rechtslineare Grammatiken (RLG)

$A = (Z, z_0, X, f, F)$.

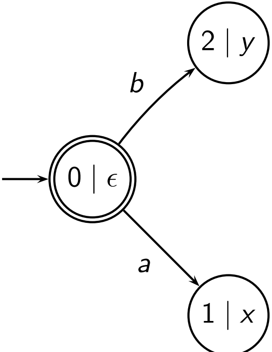
Also: $G = (Z, X, z_0, P)$ mit
$P = \{z \to xf(z,x) \mid z \in Z, x \in X\} \cup \{z \to \epsilon \mid z \in F\}$

Mealy-Automat ↔ Moore-Automat

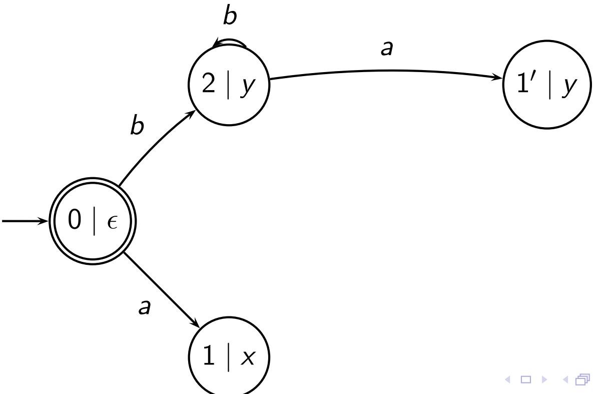
Gegeben ist folgender Mealy-Automat:



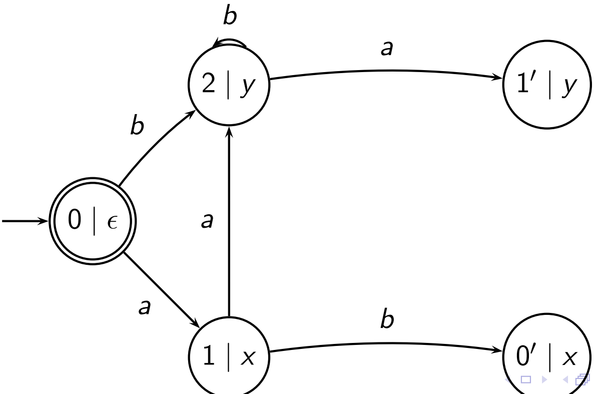
Mealy-Automat $\leftrightarrow$ Moore-Automat



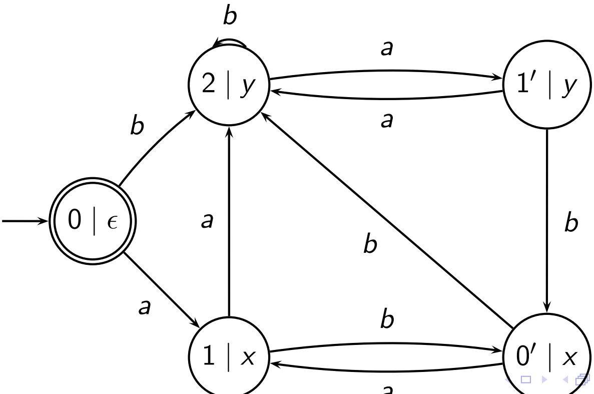
Mealy-Automat ↔ Moore-Automat



Mealy-Automat ↔ Moore-Automat



Mealy-Automat ↔ Moore-Automat



Programmieren mit Automaten

http://www.swisseduc.ch/informatik/karatojava/kara/

Mittels grafischer “Entwicklungumgebung” kann über endliche Automaten das Verhalten eines Marienkäfers programmiert werden.