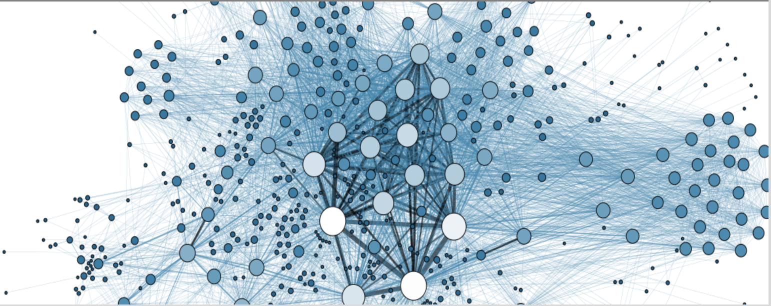
Karlsruher Institut für Technologie

Grundbegriffe der Informatik
Tutorium 33

Lukas Bach, lukas.bach@student.kit.edu | 02.02.2017



Lukas Bach, lukas.bach@student.kit.edu

Gliederung

Karlsruher Institut für Technologie

Lukas Bach, lukas.bach@student.kit.edu

Mealy-Automat

Karlsruher Institut für Technologie

Mealy-Automat

Ein Mealy-Automat ist ein Tupel $A = (Z, z_0, X, f, Y, h)$ mit...

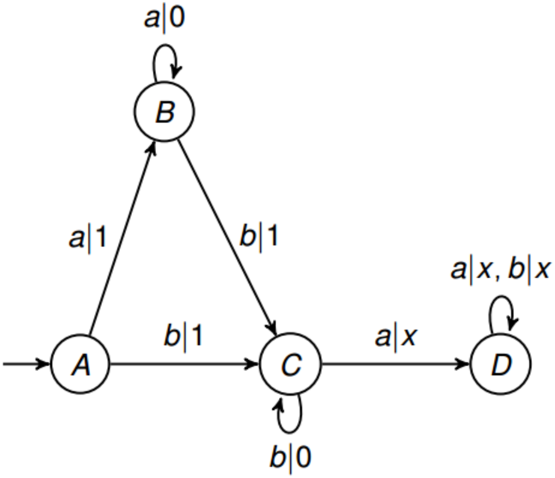
- endliche Zustandsmenge Z
- Anfangszustand $z_0 \in Z$
- Eingabealphabet X
- Zustandsübergangsfunktion $f : Z \times X \to Z$
- Ausgabealphabet Y
- Ausgabefunktion $h : Z \times X \to Y^*$

Darstellung als Graph

- Zustände $\to$ Knoten
- Startzustand $\to$ Pfeil an diesen Knoten (ohne Anfang)
- Zustandsüberführungsfunktion $\to$ Kanten mit Beschriftung
- Ausgabefunktion $\to$ zusätzliche Kantenbeschriftung

Lukas Bach, lukas.bach@student.kit.edu

Beispiel Mealy-Automat



Lukas Bach, lukas.bach@student.kit.edu

Moore-Automat

Karlsruher Institut für Technologie

Moore-Automat

Ein Moore-Automat ist ein Tupel $A = (Z, z_0, X, f, Y, h)$ mit...

- endliche Zustandsmenge Z
- Anfangszustand $z_0 \in Z$
- Eingabealphabet X
- Zustandsübergangsfunktion $f : Z \times X \to Z$
- Ausgabealphabet Y

$\to$ Bis hierhin alles wie bei Mealy!

- Ausgabefunktion $h : Z \to Y^*$

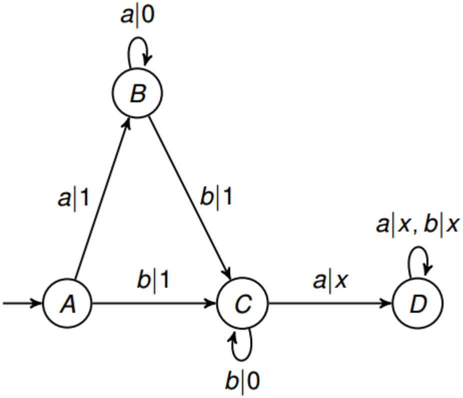
Bemerkung

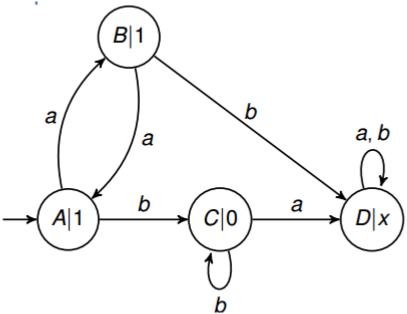
Für jeden Mealy-Automaten kann man einen Moore-Automaten konstruieren, der genau die gleiche Aufgabe erfüllt, und umgekehrt.

Lukas Bach, lukas.bach@student.kit.edu

Umwandlung Mealy- in Moore-Automat

Links Mealy-Automat, rechts Moore Automat.





Lukas Bach, lukas.bach@student.kit.edu

Endliche Akzeptoren

- Sonderfall von Moore-Automaten
- Bei einem Akzeptor will man nur wissen, ob die Eingabe akzeptiert wurde oder nicht (also reicht ein Bit als Ausgabealphabet)
- Statt der Ausgabefunktion h schreibt man einfach die Menge der akzeptierenden Zustände $F \subseteq Z$ auf
- Zustände, die nicht akzeptieren, heißen ablehnend
- Im Graphen werden akzeptierende Zustände einfach mit einem doppelten Kringel gekennzeichnet 

Akzeptierte Wörter und Sprachen

Lukas Bach, lukas.bach@student.kit.edu

Karlsruher Institut für Technologie

Akzeptierte Wörter

Ein Wort $w \in X^*$ wird vom endlichen Akzeptor akzeptiert, wenn man ausgehend vom Anfangszustand bei Eingabe von w in einem akzeptierenden Zustand endet.

Bemerkung

- Wird ein Wort nicht akzeptiert, dann wurde es abgelehnt

Akzeptierte formale Sprache

Die von einem Akzeptor A akzeptierte formale Sprache $L(A)$ ist die Menge aller von ihm akzeptierten Wörter.

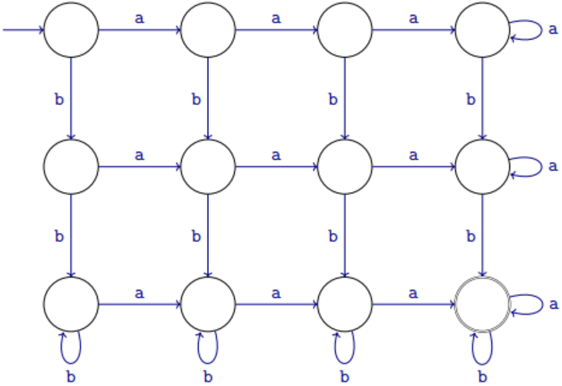
Lukas Bach, lukas.bach@student.kit.edu

Endliche Akzeptoren

Aufgabe zu endlichen Akzeptoren

Konstruiere einen endlichen Akzeptor, der die Sprache $L_1(A) = \{w \in \{a, b\}^* : (N_a(w) \geq 3 \wedge N_b(w) \geq (2)\}$ erkennt.

Lösung



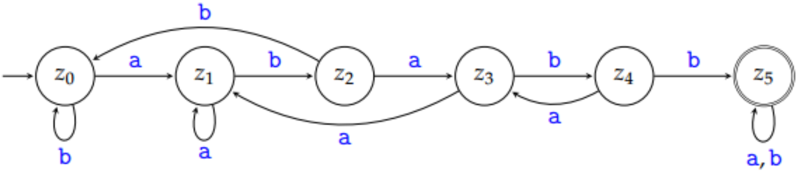
Lukas Bach, lukas.bach@student.kit.edu

Endliche Akzeptoren

Aufgabe zu endlichen Akzeptoren

Konstruiere einen endlichen Akzeptor, der die Sprache $L_2(A) = \{w_1 ababbw_2 | w_1, w_2 \in \{a, b\}^*\}$ erkennt.

Lösung



Aufgabe

Konstuiere einen endlichen Akzeptor der die Sprache $L_3 = \{w \in \{a, b\}^* | w \notin L_1\}$ akzeptiert.

Lösung

Ablehnende Zustände wereden zu akzeptierenden und andersrum.

Lukas Bach, lukas.bach@student.kit.edu

Endliche Akzeptoren

Aufgaben zu endlichen Akzeptoren

- Gebe für den unten stehenden Automaten an, welche Sprache dieser akzeptiert.
- Gebe für die folgende Sprache über dem Alphabet $\{a, b\}$ einen endlichen Akzeptor an: $L = \{w \in \Sigma^* | N_a(w) \bmod 3 > N_b(w) \bmod 2\}$

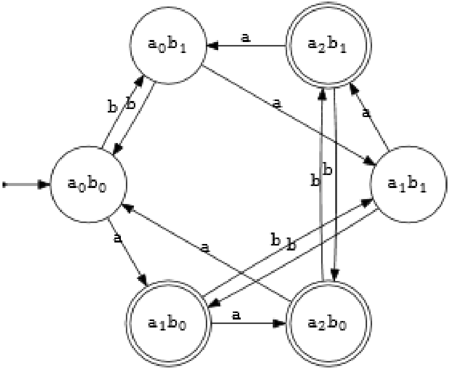
Lukas Bach, lukas.bach@student.kit.edu

Lösungen

Lösung 1

$L = \{w \in \Sigma^* || w| \bmod 2 = 1\}$ (Worte ungerader Länger)

Lösung 2



Lukas Bach, lukas.bach@student.kit.edu

Endliche Akzeptoren

Wann wird das leere Wort ε von einem endlichen Akzeptor akzeptiert?
$\varepsilon \in L(A)$ gilt genau dann, wenn der Startzustand akzeptiert wird.

Regulärer Ausdruck

Lukas Bach, lukas.bach@student.kit.edu

Regulärer Ausdruck

- Alphabet $Z = \{|, (,), *, \emptyset\}$ von "Hilfssymbolen"
- Alphabet A enthalten keine Zeichen aus Z
- Ein **regulärer Ausdruck** (RA) über A ist eine Zeichenfolge über dem Alphabet $A \cup Z$, die gewissen Vorschriften genügt.
- Vorschriften
  - $\emptyset$ ist ein RA
  - Für jedes $x \in A$ ist x ein RA
  - Wenn R_1 und R_2 RA sind, dann auch $(R_1|R_2)$ und $(R_1 R_2)$
  - Wenn R ein RA ist, dann auch $(R*)$

Lukas Bach, lukas.bach@student.kit.edu

Klammerregeln

- "Stern- vor Punktrechnung"
- "Punkt- vor Strichrechnung"

$\longrightarrow$ $R_1|R_2R_3*$ Kurzform für $(R_1|(R_2(R_3*)))$

- Bei mehreren gleichen Operatoren ohne Klammern links geklammert

$\longrightarrow$ $R_1|R_2|R_3$ Kurzform für $((R_1|R_2)|R_3)$

Aufgabe

Entferne so viele Klammern wie möglich, ohne die Bedeutung des RA zu verändern.

- $(((((ab)b)*)*)|(\emptyset*)) \longrightarrow (abb)**|\emptyset*$
- $((a(a|b))|b) \longrightarrow a(a|b)|b$

Lukas Bach, lukas.bach@student.kit.edu

Alternative Definition

Wir können die Syntax von regulären Ausdrücken auch über eine kontextfreie Grammatik definieren.

Aufgabe

Vervollständigt die folgende Grammatik.

$G = (\{R\}, \{|, (,), *, \emptyset\} \cup A, R, P)$
mit $P = \{R \to \emptyset, R \to x$ (mit $x \in A$),
$R \to (R|R), R \to (RR)$,
$R \to (R*)$
$R \to \varepsilon\}$

Wieso brauchen wir ε?

Lukas Bach, lukas.bach@student.kit.edu

Durch R beschriebene Sprache

Notation

- Spitze Klammern $\langle, \rangle$

Regeln

- $\langle \emptyset \rangle = \{\}$
- $\langle x \rangle = \{x\}$ für jedes $x \in A$
- $\langle R_1 | R_2 \rangle = \langle R_1 \rangle \cup \langle R_2 \rangle$
- $\langle R_1 R_1 \rangle = \langle R_1 \rangle \cdot \langle R_2 \rangle$
- $\langle R* \rangle = \langle R \rangle *$

Lukas Bach, lukas.bach@student.kit.edu

Charakterisierung regulärer Sprachen

Satz

Für jede formale Sprache L sind äquivalent:

1. L kann von einem endlichen Akzeptor erkannt werden.
2. L kann durch einen regulären Ausdruck beschrieben werden
3. L kann von einer rechtslinearen Grammatik erzeugt werden.

Solche Sprachen heißten regulär.

Lukas Bach, lukas.bach@student.kit.edu

Anwendung von regulären Ausdrücken

Zum selbst probieren:
http://regexr.com/

Achtung: Reguläre Ausdrücke in praktischer Programmierung funktionieren zwar ähnlich, haben aber eine andere Syntax und können teils mehr!

Lukas Bach, lukas.bach@student.kit.edu

Rechtslineare Grammatiken

Karlsruher Institut für Technologie

Definition

Eine rechtslineare Grammatik ist eine reguläre Grammatik $G = (N, T, S, P)$ mit der Einschränkung, dass alle Produktionen die folgende Form haben:

- $X \longrightarrow w$ mit $w \in T^*$ oder
- $x \longrightarrow wY$ mit $w \in T^*$, $Y \in N$

Lukas Bach, lukas.bach@student.kit.edu

Aufgabe zu rechtslinearen Grammatiken

Gebe zu $L = \{w \in \{0,1\}^* | \exists k \in \mathbb{N}_0 : Num_2(w) = 2^k + 1\}$ jeweils einen regulären Ausdruck R und eine rechtslineare Grammatik G an, sodass $L = \langle R \rangle = L(G)$ gilt.

Lösung

- $R = (0 * 10)|(0 * 1(0) * 1) = 0 * 10|0 * 10 * 1$
- $G = (\{S, A\}, \{0, 1\}, S, \{S \to 0S|10|1A, A \to 0A|1\})$

Lukas Bach, lukas.bach@student.kit.edu

Informationen

Zum Tutorium

- Lukas Bach
- Tutorienfolien auf:
  - `http://gbi.lukasbach.com`
- Tutorium findet statt:
  - Donnerstags, 14:00 - 15:30
  - 50.34 Informatikbau, -107

Mehr Material

- Ehemalige GBI Webseite:
  - `http://gbi.ira.uka.de`
  - Altklausuren!

Zur Veranstaltung

- Grundbegriffe der Informatik
- Klausurtermin:
  - 06.03.2017, 11:00
  - Zwei Stunden Bearbeitungszeit
  - 6 ECTS für Informatiker und Informationswirte, 4 ECTS für Mathematiker und Physiker

Zum Übungsschein

- Übungsblatt jede Woche
- Ab 50% insgesamt hat man den Übungsschein
- Keine Voraussetzung für die Klausur, aber für das Modul