Theoretische Informatik für Medieninformatiker

Jan Johannsen

Lehrveranstaltung im Sommersemester 2013

Organisatorisches:

Jede Lehrveranstaltungsstunde gliedert sich in einen Vorlesungsteil, dessen Länge je nach Inhalt variiert, und einen Übungsteil.

Im Übungsteil werden gemeinsam die Präsenzübungen bearbeitet.

Die Hausaufgaben orientieren sich an den Präsenzübungen und können entweder elektronisch über UniWorx oder auf Papier zu Beginn der nächsten Vorlesung abgegeben werden. Bei Papierabgabe machen Sie bitte eine Leerabgabe bei UniWorx.

Für die Hausaufgaben gibt es Punkte. Wer mindestens 50 % der Punkte für die Hausaufgaben erreicht, erhält dafür Bonuspunkte, die in der Klausur angerechnet werden. Die maximal erreichbare Zahl an Bonuspunkten entspricht einer von sechs Klausuraufgaben, also ca. 16% der Klausur.

Zur Vorlesung existiert ein Forum auf `die-informatiker.net`. Ich werde dieses lesen und mich bemühen, dort gestellte Fragen zu beantworten.

Inhalt

Teil I: Automaten und Formale Sprachen

Teil II: Berechenbarkeit

Teil III: Komplexität

Zum Inhalt von Teil I siehe unten.

Im Teil II geht es um die Frage nach den prinzipiellen Grenzen dessen, was Computer leisten können. Wir werden zeigen, dass es Probleme gibt, deren Lösung mit algorithmischen Methoden grundsätzlich nicht möglich ist. Solche Probleme nennt man *unentscheidbar*, dazu gehören z. B. Fragen nach der Korrektheit von Programmen.

Im Teil III dagegen geht es um die Untersuchung, welche Probleme von Computern *effizient* gelöst werden können. Dies führt zur Theorie der NP-Vollständigkeit und dem bekannten P-NP-Problem, für dessen Lösung ein Preis von 1.000.000 US-Dollar ausgesetzt ist.

Für beide Themen gilt: um positive Ergebnisse zu erzielen, reicht eine informelle Argumentationsweise durch Angabe eines Algorithmus. Um jedoch *negative* Ergebnisse zu erzielen, braucht man formale Maschinenmodelle, über die dann Unmöglichkeitsaussagen bewiesen werden können.

Im Teil I befassen wir uns zur Eingewöhnung daher zunächst mit einem äußerst einfachen Maschinenmodell, dem *endlichen Automaten*.

Literatur

- John E. Hopcroft, Rajeev Motwani und Jeffrey D. Ullman: *Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie*, 3. Auflage, Pearson Studium, 2011.
- Alexander Asteroth und Christel Baier: *Theoretische Informatik*, Pearson Studium 2002.
- Uwe Schöning: *Theoretische Informatik – kurz gefasst*, 5. Auflage, Spektrum Akademischer Verlag, 2008.

Teil I

Automaten und Formale Sprachen

Übersicht

Übersicht

Zeichenketten

Ein Alphabet Σ ist eine endliche Menge (von Symbolen).

Eine Zeichenkette über Σ ist eine endliche Folge

$$w = a_1 a_2 \dots a_n$$

von Symbolen $a_i \in \Sigma$.

Σ^* ist die Menge der Zeichenketten über Σ.

Beispiel: für $\Sigma = \{0, 1\}$ ist

$$\Sigma^* = \{ \epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, \dots \}$$

Zeichenketten werden auch als *Wörter* oder *Strings* über Σ bezeichnet.

Man könnte den Begriff mathematisch präzisieren, dies ist aber für unsere Zwecke irrelevant, wir beiben bei einem intuitiven Verständnis von Zeichenkette.

Länge und Konkatenation

Für $w = a_1 \dots a_n$ bezeichnet $|w| = n$ die Länge von w.

Die leere Zeichenkette ϵ hat die Länge $|\epsilon| = 0$.

Konkatenation: ist $v = a_1 \dots a_n$ und $w = b_1 \dots b_m$, dann ist

$$v \cdot w := a_1 \dots a_n b_1 \dots b_m$$

Schreibweise: statt $v \cdot w$ auch vw

Es gilt: $\epsilon \cdot w = w \cdot \epsilon = w$

Außerdem: $|v \cdot w| = |v| + |w|$

Konkatenation ist Operation auf Σ^*. Diese ist

- *assoziativ*: $(u \cdot v) \cdot w = u \cdot (v \cdot w)$
- aber *nicht kommutativ*: $v \cdot w \neq w \cdot v$

Das leere Wort ϵ ist *neutrales Element* bzgl. der Konkatenation.

$|.|$ ist ein Homomorphismus von Σ^* nach $\mathbb{N}$:

- Verträglichkeit mit der Operation: $|vw| = |v| + |w|$
- Erhaltung des neutralen Elements: $|\epsilon| = 0$

Definitionen

Definition: $\Sigma^n := \{ w \in \Sigma^* ; |w| = n \}$

Also ist $\Sigma^0 = \{\epsilon\}$, und wir identifizieren Σ^1 mit Σ.

Definition: $\Sigma^+ := \Sigma^* \setminus \{\epsilon\}$

$$\Sigma^* = \bigcup_{n \geq 0} \Sigma^n \qquad \text{und} \qquad \Sigma^+ = \bigcup_{n \geq 1} \Sigma^n$$

Definitionen

Theorie für MI

Endliche Automaten
Wörter und Sprachen
Deterministische endliche Automaten
Nichtdeterministische endliche Automaten
Äquivalenz der Automatenmodelle
Reguläre Ausdrücke
Pumping Lemma
Kontextfreie Sprachen
Pushdown-Automaten

$|w|_a$ bezeichnet die Anzahl der Vorkommen von a in w.

$$|01011|_0 = 2 \qquad |01011|_1 = 3$$

Eine Sprache ist eine Teilmenge $L \subseteq \Sigma^*$.

Beispiele für Sprachen $L \subseteq \{0,1\}^*$

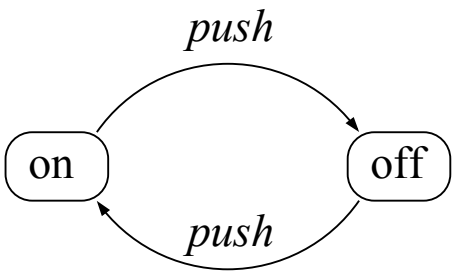
- $L = \{ w ; |w|_0 = |w|_1 \} =$
  $\{ \epsilon , 01 , 10 , 0011 , 0101 , 0110 , 1001 , 1010 , 1100 , \ldots \}$
- $L = \{ w ; (w)_2 \text{ ist Primzahl} \} =$
  $\{ 10 , 11 , 101 , 111 , 1011 , 1101 , 10001 , 10011 , 10111 , \ldots \}$
- $L = \{ 0^n 1^n ; n \geq 0 \} =$
  $\{ \epsilon , 01 , 0011 , 000111 , 00001111 , 0000011111 , \ldots \}$

Motivation 1

Modellierung von Systemen

- zu jedem Zeitpunkt in einem Zustand
- ein Ereignis bewirkt Übergang in anderen Zustand

Einfaches Beispiel: Schalter



Das Beispiel zeigt einen einfachen Druckschalter, der zwei mögliche Zustände "**on**" und "**off**" hat, und durch Drücken eines Knopfes ("*push*") vom einen in den anderen Zustand übergeht.

Typische Fragestellungen sind, welche Folgen von Ereignissen vom System sinnvoll verarbeitet werden können.

Im Beispiel beobachten wir, dass der Schalter, wenn er anfänglich auf **off** steht, nach ungerade vielen *push* im Zustand **on** und nach gerade vielen *push* im Zustand **off** ist.

Motivation 2

Klassifikation von Strings

- Suche nach Vorkommen von Schlüsselwörtern
- Überprüfung syntaktischer Korrektheit

Anwendungen

- Textsuche (Editor, Suchmaschine)
- Lexikalische Analyse

Lexikalische Analyse ist die erste Phase der Übersetzung von Programmen in Maschinencode, bei der Code in Teile zerlegt wird wie Schlüsselwörter, Bezeichner, Konstanten etc.

Motivierendes Beispiel

Suche nach Vorkommen des Schlüsselwortes `if`:

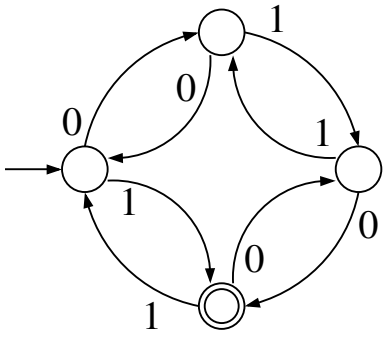
`dgrinfsmiidexifblixgifhdtuc`

Es ist leicht einzusehen, dass ein Zustandsmodell für die Suche nach Schlüsselwörtern nützlich ist. Wird in einem Programmtext nach dem Schlüsselwort “if” gesucht, so bleibe ich im Normalzustand, solange ich kein i gelesen habe. Durch das Lesen eines i geht man in einen Zustand erhöhter Aufmerksamkeit über, da es der Beginn eines “if” sein könnte. Folgt dann aber kein f gehe ich wieder in den Normalzustand. Folgt ein weiteres i, verbleibe ich im aufmerksamen Zustand, und folgt ein f, so habe ich ein Vorkommen von “if” gefunden und bin im Erfolgszustand.

Deterministische Automaten

Ein deterministischer endlicher Automat (DEA) A mit Alphabet Σ besteht aus:

- Zustände Q (endlich viele!)
- Übergangsfunktion $\delta : Q \times \Sigma \to Q$
- Anfangszustand $q_0 \in Q$
- Endzustände $F \subseteq Q$



Kurzschreibweise: $A = (Q, \Sigma, \delta, q_0, F)$

Die Kurzschreibweise liest ein Informatiker am besten so:
A ist ein Objekt der Klasse DEA, dessen Instanzvariablen sind Q, Σ, δ, q_0, F mit der oben beschriebenen Bedeutung.

Ein Automat repräsentiert einen Algorithmus zur Entscheidung, ob ein Wort $w \in \Sigma^*$ korrekt ist, also von A akzeptiert ist. Die Menge der akzeptierten Wörter ist eine Sprache, die von A akzeptierte Sprache $L(A)$.

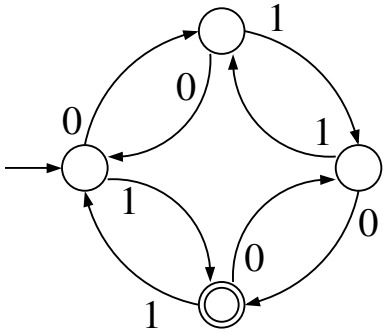
Die Arbeitsweise des Automaten ist intuitiv wie folgt: anfänglich befindet sich A im Anfangszustand q_0. Dann wird das eingegebene Wort w Symbol für Symbol gelesen, und bei jedem gelesenen Symbol entsprechend der Übergangsfunktion der Zustand gewechselt. Ist, nachdem w komplett gelesen wurde, der Zustand ein Endzustand in F, so wird w akzeptiert, andernfalls nicht.

Die nächsten Folien präzisieren diese Arbeitsweise formal.

Deterministische Automaten

Alternative Notation: Übergangstabelle

		0	1
$\rightarrow$	q_0	q_1	q_2
	q_1	q_0	q_3
$*$	q_2	q_3	q_0
	q_3	q_2	q_1



In der Tabellenschreibweise wird der Anfangszustand durch einen Pfeil $\rightarrow$ markiert, und die Endzustände durch einen Stern $*$.

Erweiterte Übergangsfunktion

Automat soll ganze Wörter $w \in \Sigma^*$ abarbeiten.

Dazu wird Übergangsfunktion δ auf Σ^* erweitert:

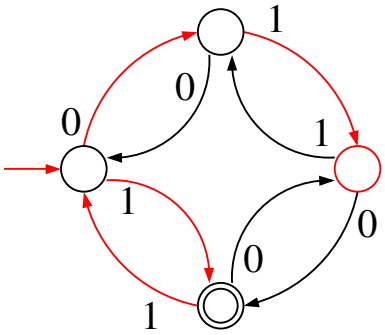
$\hat{\delta} : Q \times \Sigma^* \to Q$ ist induktiv definiert durch

$$\begin{aligned} \hat{\delta}(q, \epsilon) &= q \\ \hat{\delta}(q, wa) &= \delta(\hat{\delta}(q, w), a) \end{aligned}$$

Der Funktionswert $\hat{\delta}(q, w)$ gibt den Zustand an, in dem sich A befindet, wenn er im Zustand q gestartet das Wort w liest.

Beispiel

	0	1
$\rightarrow q_0$	q_1	q_2
q_1	q_0	q_3
$* q_2$	q_3	q_0
q_3	q_2	q_1



Berechne $\hat{\delta}(\, q_0\,,\, 1101\,)$:

$$
\begin{aligned}
\hat{\delta}(q_0, \epsilon) &= q_0 \\
\hat{\delta}(q_0, 1) &= \delta(\,\hat{\delta}(q_0, \epsilon)\,,\, 1\,) &&= \delta(q_0, 1) &&= q_2 \\
\hat{\delta}(q_0, 11) &= \delta(\,\hat{\delta}(q_0, 1)\,,\, 1\,) &&= \delta(q_2, 1) &&= q_0 \\
\hat{\delta}(q_0, 110) &= \delta(\,\hat{\delta}(q_0, 11)\,,\, 0\,) &&= \delta(q_0, 0) &&= q_1 \\
\hat{\delta}(q_0, 1101) &= \delta(\,\hat{\delta}(q_0, 110)\,,\, 1\,) &&= \delta(q_1, 1) &&= q_3
\end{aligned}
$$

Theorie für MI

Endliche Automaten
Wörter und Sprachen
Deterministische endliche Automaten
Nichtdeterministische endliche Automaten
Äquivalenz der Automatenmodelle
Reguläre Ausdrücke
Pumping Lemma
Kontextfreie Sprachen
Pushdown-Automaten

Sprache des Automaten

Definition

Die von $A = (Q, \Sigma, \delta, q_0, F)$ akzeptierte Sprache ist:

$$L(A) := \{\, w \in \Sigma^* \,;\, \hat{\delta}(q_0, w) \in F \,\}$$

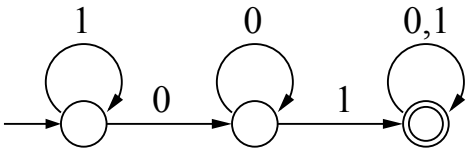
Konstruktion eines Automaten

Beispiel: Menge L der Wörter, die 01 enthalten, also

$$L = \{\, u01v \,;\, u, v \in \{0,1\}^* \,\}$$

- q_0 Anfangszustand, noch keine 0 gelesen
- q_1 0 gelesen, noch keine 1
- q_2 01 gelesen, fertig

	0	1
$\rightarrow q_0$	q_1	q_0
q_1	q_1	q_2
$*\, q_2$	q_2	q_2



Konstruktion eines Automaten

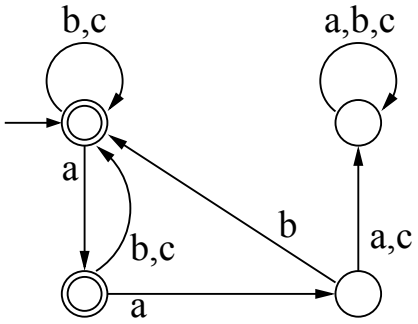
Theorie für MI

Endliche Automaten
Wörter und Sprachen
Deterministische endliche Automaten
Nichtdeterministische endliche Automaten
Äquivalenz der Automatenmodelle
Reguläre Ausdrücke
Pumping Lemma
Kontextfreie Sprachen
Pushdown-Automaten

Noch ein Beispiel: Menge L der Wörter aus $\{a, b, c\}^*$ mit:
auf jedes Teilwort aa folgt ein b

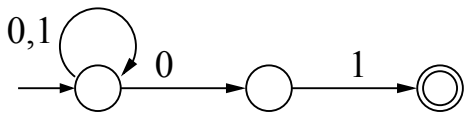
- q_0 Anfangszustand, noch kein a gelesen
- q_1 a gelesen,
- q_2 aa gelesen
- q_3 Mülleimer

	a	b	c
$*\rightarrow q_0$	q_1	q_0	q_0
$*\, q_1$	q_2	q_0	q_0
q_2	q_3	q_0	q_3
q_3	q_3	q_3	q_3



Nichtdeterministische Automaten

Ein nichtdeterministischer Automat:



Ein nichtdeterministischer endlicher Automat (NEA) ist definiert wie ein DEA, nur mit

- Übergangsfunktion $\delta : Q \times \Sigma \to 2^Q$

Im Beispiel oben:

$$\delta(q_0, 0) = \{q_0, q_1\} \qquad \delta(q_0, 1) = \{q_0\} \qquad \delta(q_1, 0) = \emptyset$$

Ein nichtdeterministischer Automat kann in einem Zustand q für ein gelesenes Symbol a mehrere Übergänge zur Auswahl haben.

Daher liefert die Übergangsfunktion als Wert $\delta(q, a)$ eine *Menge* von Zuständen, nämlich die Menge der möglichen Folgezustände.

Im Gegensatz zu einem DEA kann ein NEA auch in Sackgassen geraten, wenn nämich für Zustand q und Symbol a kein Übergang definiert ist, also wenn $\delta(q, a) = \emptyset$.

Für jedes Wort w gibt es also verschiedene mögliche Abläufe des Automaten, die in verschiedenen Zuständen enden können. Für die Akzeptanz reicht es, wenn einer dieser Abläufe in einem Endzustand endet. Dies wird wiederum auf den folgenden Folien präzisiert.

Erweiterte Übergangsfunktion

Wie beim DEA wird die Übergangsfunktion δ auf Σ^* erweitert:

$\hat{\delta} : Q \times \Sigma^* \to 2^Q$ ist induktiv definiert durch

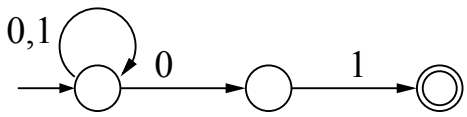
$$\begin{aligned} \hat{\delta}(q, \epsilon) &= \{q\} \\ \hat{\delta}(q, wa) &= \bigcup_{q' \in \hat{\delta}(q,w)} \delta(q', a) \end{aligned}$$

Beispiel

Theorie für MI

Endliche Automaten
Wörter und Sprachen
Deterministische endliche Automaten
Nichtdeterministische endliche Automaten
Äquivalenz der Automatenmodelle
Reguläre Ausdrücke
Pumping Lemma
Kontextfreie Sprachen
Pushdown-Automaten

	0	1
$\to q_0$	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\emptyset$	$\{q_2\}$
$* q_2$	$\emptyset$	$\emptyset$



Berechne $\hat{\delta}(\, q_0 \,, 011\,)$:

$$\begin{aligned} \hat{\delta}(q_0, 0) &= \bigcup_{q \in \hat{\delta}(q_0, \epsilon)} \delta(q, 0) &&= \delta(q_0, 0) &&= \{q_0, q_1\} \\ \hat{\delta}(q_0, 01) &= \bigcup_{q \in \hat{\delta}(q_0, 0)} \delta(q, 1) &&= \delta(q_0, 1) \cup \delta(q_1, 1) &&= \{q_0, q_2\} \\ \hat{\delta}(q_0, 011) &= \bigcup_{q \in \hat{\delta}(q_0, 01)} \delta(q, 1) &&= \delta(q_0, 1) \cup \delta(q_2, 1) &&= \{q_0\} \end{aligned}$$

Sprache des Automaten

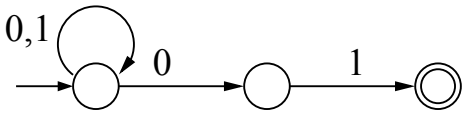
Definition

Die vom NEA $A = (Q, \Sigma, \delta, q_0, F)$ akzeptierte Sprache ist:

$$L(A) := \{ w \in \Sigma^* ; \hat{\delta}(q_0, w) \cap F \neq \emptyset \}$$

Beispiel: Der Automat A akzeptiert die Sprache

$$\{ w ; w = u01 \text{ für ein } u \}$$



Wir wollen beweisen, dass der Automat tatsächlich die genannte Sprache akzeptiert.

Dazu sind drei Behauptungen zu beweisen, die für jeden der Zustände charakterisieren, mit welchen Eingabewörtern sie erreicht werden.

1. $q_0 \in \hat{\delta}(q_0, w)$ für alle w.
2. $q_1 \in \hat{\delta}(q_0, w)$ genau dann, wenn w mit 0 endet.
3. $q_2 \in \hat{\delta}(q_0, w)$ genau dann, wenn w mit 01 endet.

Damit folgt die Aussage, da q_2 der einzige Endzustand ist.

1. ist klar, da ich von q_0 mit jedem Symbol in q_0 übergehen kann.

Zu 2.: Ist $w = v0$, dann ist $q_0 \in \hat{\delta}(q_0, v)$ nach 1., also kann danach mit der 0 in q_1 übergegangen werden.
Umgekehrt ist q_1 nur erreichbar mit dem Übergang von q_0 bei Eingabe 1, also muss das letzte Symbol eine 0 sein, wenn in q_1 geendet wird.

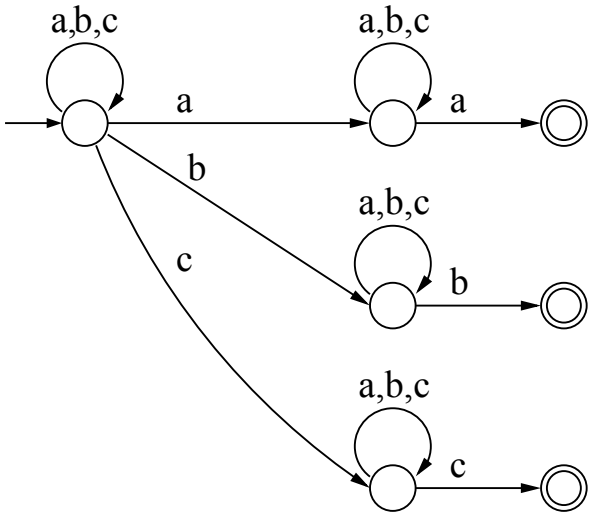
Zu 3.: Ist $w = u01$, dann ist $q_0 \in \hat{\delta}(q_0, u)$ nach 1., also kann danach mit der 0 in q_1 und schließlich mit der 1 in q_2 übergegangen werden.

Umgekehrt ist q_2 nur über den Übergang mit 1 von q_1 erreichbar, ist also q_2 der Zustand nach Lesen von w, muss $w = v1$ für ein v sein, und nach v ist der Zustand q_1. Dann muss aber nach 2. v auf 0 enden, also $v = u0$ für ein u, und somit ist $w = u01$.

Konstruktion eines NEA

Beispiel: Menge L der Wörter aus $\{a, b, c\}^*$ mit:
das letzte Symbol ist mindestens einmal vorher vorgekommen.

Also: $cbabba \in L$ aber $cbbcba \notin L$



Arbeitsweise des NEA

ababcbab