Automatentheorie und ihre Anwendungen
Teil 1: endliche Automaten auf endlichen Wörtern

Wintersemester 2017/18 Thomas Schneider

AG Theorie der künstlichen Intelligenz (TdKI)

`http://tinyurl.com/ws1718-automaten`

Vorlesungsübersicht

Kapitel 1: endliche Automaten auf endlichen Wörtern

Kapitel 2: endliche Automaten auf endlichen Bäumen

Kapitel 3: endliche Automaten auf unendlichen Wörtern

Kapitel 4: endliche Automaten auf unendlichen Bäumen

Ziel dieses Kapitels

- Wiederholung der Definitionen & Resultate zu endlichen Automaten aus „Theoretische Informatik 1“
- Kennenlernen zweier Anwendungen endlicher Automaten

Überblick

1. Grundbegriffe
2. *Anwendung: Textsuche*
3. Abschlusseigenschaften
4. Reguläre Ausdrücke und *Anwendungen*
5. Charakterisierungen
6. Entscheidungsprobleme

Und nun ...

1. Grundbegriffe
2. *Anwendung: Textsuche*
3. Abschlusseigenschaften
4. Reguläre Ausdrücke und *Anwendungen*
5. Charakterisierungen
6. Entscheidungsprobleme

Wörter, Sprachen, ...

- **Symbole** $a, b, \ldots$
- **Alphabet** Σ: endliche nichtleere Menge von Symbolen
- **(endliches) Wort** w über Σ:
  endliche Folge $w = a_1 a_2 \ldots a_n$ von Symbolen $a_i \in \Sigma$
- **leeres Wort** ε
- **Wortlänge** $|a_1 a_2 \ldots a_n| = n, \quad |\varepsilon| = 0$
- **Menge aller Wörter** über Σ: Σ^*
- **Sprache** L über Σ: Teilmenge $L \subseteq \Sigma^*$ von Wörtern
- **Sprachklasse** $\mathcal{L}$: Menge von Sprachen

Endliche Automaten

Definition 1.1

Ein **nichtdeterministischer endlicher Automat (NEA)** über einem Alphabet Σ ist ein 5-Tupel $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$, wobei

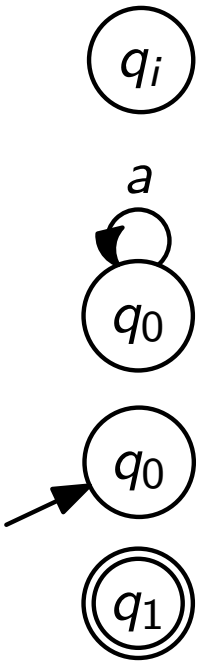
- Q eine endliche nichtleere **Zustandsmenge** ist,
- Σ ein Alphabet ist,
- $\Delta \subseteq Q \times \Sigma \times Q$ die **Überführungsrelation** ist, $\quad (*)$
- $I \subseteq Q$ die Menge der **Anfangszustände** ist,
- $F \subseteq Q$ die Menge der **akzeptierenden Zustände** ist.

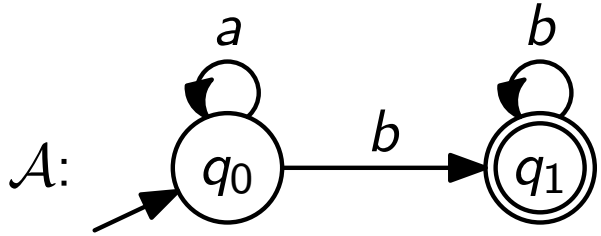
- $(*)$ bedeutet:
  Δ besteht aus Tripeln (q, a, q') mit $q, q' \in Q$ und $a \in \Sigma$
- $(q, a, q') \in \Delta$ bedeutet intuitiv:
  ist $\mathcal{A}$ in Zustand q und liest ein a, geht er in Zustand q' über.

Beispiel und graphische Repräsentation von NEAs

Betrachte $\mathcal{A} =$
$(\{q_0, q_1\},\ \{a, b\},\ \{(q_0, a, q_0), (q_0, b, q_1), (q_1, b, q_1)\},\ \{q_0\}, \{q_1\})$

- Zustände: q_0, q_1
- Alphabet $\{a, b\}$
- Übergänge: von q_0 mittels a zu q_0, ...
- Anfangszustand q_0
- einziger akzeptierender Zustand q_1





Berechnungen und Akzeptanz

Definition 1.2

Sei $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$ ein NEA.

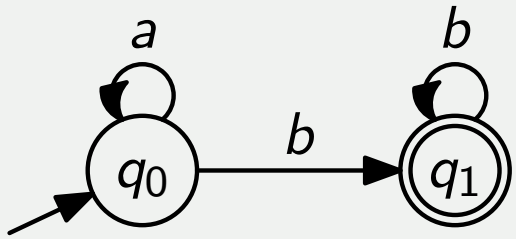
- Eine **Berechnung** (Run) von $\mathcal{A}$ auf $w = a_1 a_2 \dots a_n$ ist eine Folge
$$q_0 q_1 q_2 \dots q_n,$$
so dass für alle $i = 0, \dots, n-1$ gilt: $(q_i, a_{i+1}, q_{i+1}) \in \Delta$.
Man sagt/schreibt: w **überführt** q_0 in q_n, $q_0 \vdash_{\mathcal{A}}^{w} q_n$

- $\mathcal{A}$ **akzeptiert** $w = a_1 a_2 \dots a_n$,
wenn es eine Berechnung $q_0 q_1 q_2 \dots q_n$ von $\mathcal{A}$ auf w gibt mit $q_0 \in I$ und $q_n \in F$.

- Die von $\mathcal{A}$ **erkannte Sprache** ist
$L(\mathcal{A}) = \{w \in \Sigma^* \mid \mathcal{A} \text{ akzeptiert } w\}$.

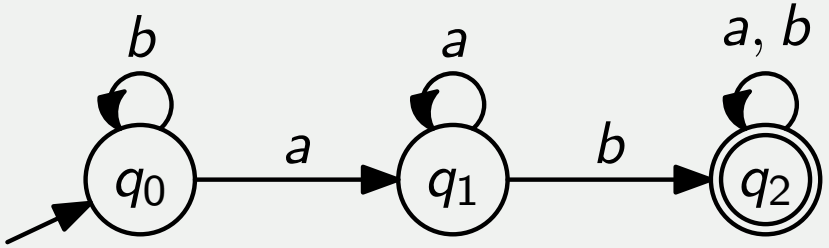
Beispiele

$\mathcal{A}_1$:



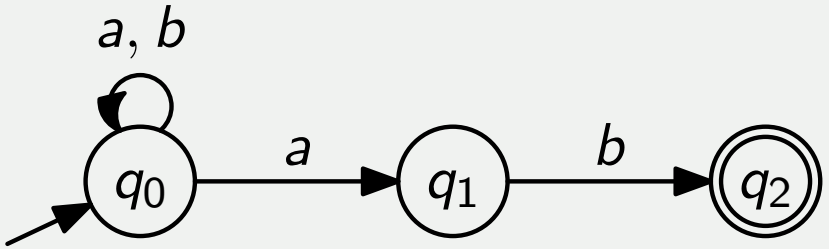
$$L(\mathcal{A}_1) = \{a^n b^m \mid n \geqslant 0, m \geqslant 1\}$$

$\mathcal{A}_2$:



$$L(\mathcal{A}_2) = \{w \in \{a, b\}^* \mid w \text{ enthält Teilwort } ab\}$$

$\mathcal{A}_3$:



$$L(\mathcal{A}_3) = \{w \in \{a, b\}^* \mid w \text{ endet auf } ab\}$$

Erkennbare Sprache

Definition 1.3

Eine Sprache $L \subseteq \Sigma^*$ ist **(NEA-)erkennbar**,
wenn es einen NEA $\mathcal{A}$ gibt mit $L = L(\mathcal{A})$.

Determinismus

Definition 1.4

Sei $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$ ein NEA.

Enthält Δ für jedes $q \in Q$ u. jedes $a \in \Sigma$ **genau 1** Tripel (q, a, q') und enthält I **genau 1** Zustand,

dann ist $\mathcal{A}$ ein **deterministischer endlicher Automat (DEA)**.

$\rightsquigarrow$ Nachfolgezustand für jedes Paar (q, a) eindeutig bestimmt

- Jeder DEA ist ein NEA,
  aber nicht umgekehrt (z. B. $\mathcal{A}_1, \mathcal{A}_3$ auf Folie 10).
- Auf Folie 10 ist nur $\mathcal{A}_2$ ein DEA;
  $\mathcal{A}_1$ kann mittels *Papierkorbzustand* zum DEA werden;
  bei $\mathcal{A}_3$ genügt auch das nicht. **T 1.1**

Potenzmengenkonstruktion

Frage: Sind DEAs und NEAs gleichmächtig?

Antwort: Ja!

Satz 1.5 (Rabin, Scott 1959)

Für jeden NEA $\mathcal{A}$ gibt es einen DEA $\mathcal{A}^d$ mit $L(\mathcal{A}^d) = L(\mathcal{A})$.

Beweisskizze: Sei $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$.

Wir konstruieren $\mathcal{A}^d = (Q^d, \Sigma, \Delta^d, I^d, F^d)$ wie folgt.

- $Q^d = 2^Q$ (Potenzmenge der Zustandsmenge)
- $I^d = \{I\}$
- $(S, a, S') \in \Delta^d$ gdw. $S' = \{q' \mid \exists q \in S : (q, a, q') \in \Delta\}$
- $F^d = \{S \subseteq Q \mid S \cap F \neq \emptyset\}$ T 1.2

Im schlimmsten Fall kann $\mathcal{A}^d$ im Vergleich zu $\mathcal{A}$ exponentiell viele Zustände haben (s. Hopcroft et al. 2001, S. 65).

Und nun ...

1 Grundbegriffe

2 *Anwendung: Textsuche*

3 Abschlusseigenschaften

4 Reguläre Ausdrücke und *Anwendungen*

5 Charakterisierungen

6 Entscheidungsprobleme

Stichwortsuche

Typisches Problem aus dem Internetzeitalter

Gegeben sind **Stichwörter** $w_1, \ldots, w_n \in \Sigma^*$
und **Dokumente** $D_1, \ldots, D_M \in \Sigma^*$.

Finde alle j, so dass D_j mindestens ein (alle) w_i als Teilwort hat.

- relevant z.B. für **Suchmaschinen**
- übliche Technologie: **invertierter Index (II)**

  speichert für jedes im Internet auftretende w_i
  eine Liste aller Dokumente D_j, die w_i enthalten
- IIs sind zeitaufwändig zu erstellen
  und setzen voraus, dass die D_j sich nur langsam ändern

Stichwortsuche ohne invertierte Indizes?

Ils versagen, wenn

- die (relevanten) Dokumente sich schnell ändern:
  - Suche in tagesaktuellen Nachrichtenartikeln
  - Einkaufshelfer sucht nach bestimmten Artikeln in aktuellen Seiten von Online-Shops
- die Dokumente nicht katalogisiert werden können:
  - Online-Shops wie Amazon generieren oft Seiten für ihre Artikel nur auf Anfragen hin.

$\rightsquigarrow$ Wie kann man dafür Stichwortsuche implementieren?

Ein Fall für endliche Automaten!

Gegeben sind **Stichwörter** $w_1, \ldots, w_n \in \Sigma^*$
und **Dokumente** $D_1, \ldots, D_M \in \Sigma^*$.

Finde alle j, so dass D_j mindestens ein w_i als Teilwort hat.

Ziel: konstruiere NEA $\mathcal{A}$, der

- ein D_j zeichenweise liest und
- in einen Endzustand geht gdw. er eins der w_i findet

Der Einfachheit halber legen wir fest, dass $\mathcal{A}$ ein Wort w akzeptiert, wenn $\mathcal{A}$ bereits nach Lesen eines *Teilworts* einen akz. Zustand erreicht.

Beispiel 1.6

$w_1 = \texttt{web}$ und $w_2 = \texttt{ebay}$

T 1.3

Implementation des NEAs $\mathcal{A}$

Eine Möglichkeit:

1. Determinisierung (Potenzmengenkonstruktion)
2. Simulation des resultierenden DEA $\mathcal{A}^d$

Wird $\mathcal{A}^d$ nicht zu groß?
($2^{27} > 134$ Mio. Zustände bei Stichw. „Binomialkoeffizient“, „Polynom“)

Nein,

- mit der leicht geänderten Definition von Akzeptanz
- und unserer Variante der Potenzmengenkonstruktion

wird $\mathcal{A}^d$ genauso viele Zustände haben wie $\mathcal{A}$!

T 1.4

Und nun ...

1 Grundbegriffe

2 *Anwendung: Textsuche*

3 Abschlusseigenschaften

4 Reguläre Ausdrücke und *Anwendungen*

5 Charakterisierungen

6 Entscheidungsprobleme

Operationen auf Sprachen sind Operationen auf Mengen

Wie können (NEA-erkennbare) Sprachen kombiniert werden?

- Boolesche Operationen

  Vereinigung $L_1 \cup L_2 = \{w \mid w \in L_1 \text{ oder } w \in L_2\}$

  Schnitt $L_1 \cap L_2 = \{w \mid w \in L_1 \text{ und } w \in L_2\}$

  Komplement $\overline{L} = \{w \in \Sigma^* \mid w \notin L\}$

- Wortoperationen

  Konkatenation $L_1 \cdot L_2 = \{vw \mid v \in L_1 \text{ und } w \in L_2\}$

  Kleene-Hülle $L^* = \bigcup_{i \geqslant 0} L^i,$

  wobei $L^0 = \{\varepsilon\}$ und $L^{i+1} = L^i \cdot L$ für alle $i \geqslant 0$

Frage

Unter welchen Op. sind die NEA-erkennbaren Sprachen abgeschlossen?

Abgeschlossenheit

Satz 1.7

Die Menge der NEA-erkennbaren Sprachen ist abgeschlossen unter den Operationen $\cup, \cap, \overline{\ }, \cdot, ^*$.

Das heißt:

Wenn L, L_1, L_2 NEA-erkennbar sind, dann sind auch

- $L_1 \cup L_2$
- $L_1 \cap L_2$
- $\overline{L}$
- $L_1 \cdot L_2$
- L^*

NEA-erkennbar.

Beweis: Siehe ThI 1.

Und nun ...

1 Grundbegriffe

2 *Anwendung: Textsuche*

3 Abschlusseigenschaften

4 Reguläre Ausdrücke und *Anwendungen*

5 Charakterisierungen

6 Entscheidungsprobleme

Reguläre Ausdrücke und Anwendungen

Reguläre Ausdrücke sind . . .

- bequeme Charakterisierung NEA-erkennbarer Sprachen
- besonders praktisch für Anwendungen

Reguläre Sprachen

Definition 1.8

Eine Sprache $L \subseteq \Sigma^*$ ist **regulär**, falls gilt:

- $L = \emptyset$ oder
- $L = \{\varepsilon\}$ oder
- $L = \{a\}$, $a \in \Sigma$, oder
- L lässt sich durch (endlichmaliges) Anwenden der Operatoren $\cup, \cdot, ^*$ aus den vorangehenden Fällen konstruieren.

Beispiele:

$(\{a\} \cup \{b\})^* \cdot \{a\} \cdot \{b\}$ (siehe $\mathcal{A}_3$ auf Folie 10)

$\{b\}^* \cdot \{a\} \cdot \{a\}^* \cdot \{b\} \cdot (\{a\} \cup \{b\})^*$ (s. $\mathcal{A}_2$ auf Folie 10)

Reguläre Ausdrücke

Definition 1.9

Ein **regulärer Ausdruck (RA)** r über Σ und die *zugehörige Sprache* $L(r) \subseteq \Sigma^*$ werden induktiv wie folgt definiert.

- $r = \emptyset$ ist ein RA mit $L(r) = \emptyset$
- $r = \varepsilon$ ist ein RA mit $L(r) = \{\varepsilon\}$
- $r = a$, für $a \in \Sigma$, ist ein RA mit $L(r) = \{a\}$
- $r = (r_1 + r_2)$ ist ein RA mit $L(r) = L(r_1) \cup L(r_2)$
- $r = (r_1 r_2)$ ist ein RA mit $L(r) = L(r_1) \cdot L(r_2)$
- $r = (r_1)^*$ ist ein RA mit $L(r^*) = (L(r))^*$

Beispiele: (wir lassen Klammern weg soweit eindeutig)

$(a + b)^* ab$ (siehe $\mathcal{A}_3$ auf Folie 10)

$b^* aa^* b(a + b)^*$ (siehe $\mathcal{A}_2$ auf Folie 10)

Reguläre und NEA-erkennbare Sprachen

Satz 1.10 (Kleene 1956)

Sei $L \subseteq \Sigma^*$ eine Sprache.

1. L ist regulär gdw. es einen RA r gibt mit $L = L(r)$.
2. L ist regulär gdw. L NEA-erkennbar ist.

Beweis.

1. Folgt offensichtlich aus Def. 1.8, 1.9.
2. Benutze Teil 1.

   „$\Rightarrow$“: Induktion über Aufbau von r.

   IA: gib Automaten an, die $\emptyset, \{\varepsilon\}, \{a\}$ erkennen.
   IS: benutze Abschlusseigenschaften (Satz 1.7)

   „$\Leftarrow$“: siehe Theoretische Informatik 1. □

Anwendungen regulärer Ausdrücke

- RAs werden verwendet, um „Muster“ von zu suchendem Text zu beschreiben.

  z. B.: suche alle Vorkommen von „PLZ Ort“:
  $(0 + \cdots + 9)^5 \sqcup (A + \cdots + Z)(a + \cdots + z)^*$

- Programme zum Suchen von Mustern im Text übersetzen RAs in NEAs/DEAs und simulieren diese.

- wichtige Klassen von Anwendungen:
  lexikalische Analyse, **Textsuche**

Komfortablere Syntax regulärer Ausdrücke

- UNIX und andere Anwendungen erweitern Syntax von RAs
- Hier: nur „syntaktischer Zucker“ – die Erweiterungen, die nicht aus den regulären Sprachen herausführen

- Alphabet Σ: alle ASCII-Zeichen
- RA . mit $L(.) = \Sigma$
- RA $[a_1 a_2 \ldots a_k]$, Abkürzung für $a_1 + a_2 + \cdots + a_k$
- RAs für Bereiche: z. B. `[a-z0-9]`, Abkü. für `[ab...z01...9]`
- Operator | anstelle +
- Operator ?: $r?$ steht für $\varepsilon + r$
- Operator +: $r+$ steht für rr^*
- Operator $\{n\}$: $r\{5\}$ steht für $rrrrr$
- Klammern und $*$ wie gehabt

PLZ-Ort-Beispiel:
`[0-9]{5}␣[A-Z][a-z]*`

Anwendung: lexikalische Analyse

- **Lexer** durchsucht Quelltext eines Programms nach **Token**:
  zusammengehörende Zeichenfolgen, z. B. Kennwörter, Bezeichner
- Ausgabe des Lexers: Token-Liste,
  wird an Parser weitergegeben
- Mit RAs: Lexer leicht programmier- und modifizierbar
- UNIX-Kommandos `lex` und `flex` generieren Lexer
  - Eingabe: Liste von Einträgen RA + Code
  - Code beschreibt Ausgabe des Lexers für das jeweilige Token
  - generierter Lexer wandelt alle RAs in **einen DEA** um,
    um Vorkommen der Tokens zu finden (siehe Folie 17)
  - anhand des Zustands des DEAs lässt sich bestimmen,
    welches Token gefunden wurde

Beispieleingabe für `lex`

```
else
  {return(ELSE)}

[A-Za-z][A-za-z0-9]*
  {<Trage gefundenen Bezeichner in Symboltabelle ein>;
  return(ID);}

>=
  {return(GE);}

=
  {return(EQ);}
```

(Lexer-Generator muss Prioritäten beachten:
`else` wird auch vom 2. RA erkannt, ist aber reserviert)

Anwendung: Finden von Mustern im Text

Beispiel: Suchen von Adressen (Str. + Hausnr.) in Webseiten

Solche Angaben sollen gefunden werden:

```
Parkstraße 5
Enrique-Schmidt-Straße 12a
Breitenweg 24A
Knochenhauergasse 30-32
```

aber auch solche:

```
Straße des 17. Juni 17
...boulevard, ...allee, ...platz, ...
Postfach 330 440
Am Wall 8
```

$\rightsquigarrow$ Ausmaß der Variationen erst während der Suche deutlich

$\rightsquigarrow$ Gesucht: einfach modifizierbare Beschreibung der Muster

Mustersuche mit regulären Ausdrücken

Mögliches Vorgehen:

(1) Beschreibung des Musters mit einem einfachen RA

(2) Umwandlung des RA in einen NEA

(3) Implementation des DEA wie auf Folie 17+18

(4) Test

(5) Wenn nötig, RA erweitern/ändern und Sprung zu Schritt 2

Adresssuche mit regulären Ausdrücken

So kann sich der RA entwickeln:

- Vorkommen von „straße“ etc.:[1]
  `straße|str\.|weg|gasse`
- Plus Name der Straße und Hausnummer:
  `[A-Z][a-z]*(straße|str\.|weg|gasse) [0-9]*`
- Hausnummern mit Buchstaben (12a), -bereiche (30–32):
  `[A-Z][a-z]*(straße|str\.|weg|gasse)`
  `([0-9]*[A-Za-z]?-)?[0-9]*[A-Za-z]?`
- und mehr:
  - Straßennamen mit Bindestrichen
  - „Straße“ etc. am Anfang
  - Plätze, Boulevards, Alleen etc.
  - Postfächer
  - ...

[1] Weil der UNIX-RA . für Σ reserviert ist, steht \. für $\{.\}$

Und nun ...

1 Grundbegriffe

2 *Anwendung: Textsuche*

3 Abschlusseigenschaften

4 Reguläre Ausdrücke und *Anwendungen*

5 Charakterisierungen

6 Entscheidungsprobleme

Pumping-Lemma

Wie zeigt man, dass L **nicht** NEA-erkennbar (regulär) ist?

Satz 1.11 (Pumping-Lemma)

Sei L eine NEA-erkennbare Sprache.

Dann gibt es eine Konstante $p \geq 0$,
so dass für alle Wörter $w \in L$ mit $|w| \geqslant p$ gilt:

Es gibt eine Zerlegung $w = xyz$ mit $y \neq \varepsilon$ und $|xy| \leqslant p$,
so dass $xy^i z \in L$ für alle $i \geq 0$.

Beweis: siehe ThI 1.

Anwendung des Pumping-Lemmas

Benutzen Kontraposition:

Wenn es **für alle** Konstanten $p \geq 0$
ein Wort $w \in L$ mit $|w| \geqslant p$ **gibt**, so dass es
für alle Zerlegungen $w = xyz$ mit $y \neq \varepsilon$ und $|xy| \leqslant p$
ein $i \geq 0$ **gibt** mit $xy^i z \notin L$,
dann ist L **keine** NEA-erkennbare Sprache.

T 1.5

Bemerkungen zum Pumping-Lemma

Die Bedingung in Satz 1.11 ist ...

- **notwendig** dafür, dass L NEA-erkennbar ist
- **nicht hinreichend**
  Bsp.: $\{a^n b^k c^k \mid n, k \geqslant 1\} \cup \{b^n c^k \mid n, k \geqslant 0\}$

$\rightsquigarrow$ Pumping-L. nur zum **Widerlegen** von Erkennbarkeit verwendbar, nicht zum Beweisen, dass L regulär ist

(Notwendige und hinreichende Variante: Jaffes Pumping-Lemma)

Der Satz von Myhill-Nerode

Ziel: notwendige **und** hinreichende Bedingung für Erkennbarkeit

Definition 1.12

Sei $L \subseteq \Sigma^*$ eine Sprache.

Zwei Wörter $u, v \in \Sigma^*$ sind ***L*-äquivalent** (Schreibweise: $u \sim_L v$), wenn für alle $w \in \Sigma^*$ gilt:

$$uw \in L \quad \text{genau dann, wenn} \quad vw \in L$$

$\sim_L$ heißt **Nerode-Rechtskongruenz** und ist Äquivalenzrelation (Reflexivität, Symmetrie, Transitivität sind offensichtlich)

Index von $\sim_L$: Anzahl der Äquivalenzklassen

Der Satz von Myhill-Nerode

Satz 1.13 (Myhill-Nerode)

$L \subseteq \Sigma^*$ is NEA-erkennbar gdw. $\sim_L$ endlichen Index hat.

Beweis: siehe ThI 1. **T 1.6**

Interessantes **„Nebenprodukt“** des Beweises:

Endlicher Index n von $\sim_L$
= minimale Anzahl von Zuständen in einem DEA, der L erkennt.

Und nun ...

1 Grundbegriffe

2 *Anwendung: Textsuche*

3 Abschlusseigenschaften

4 Reguläre Ausdrücke und *Anwendungen*

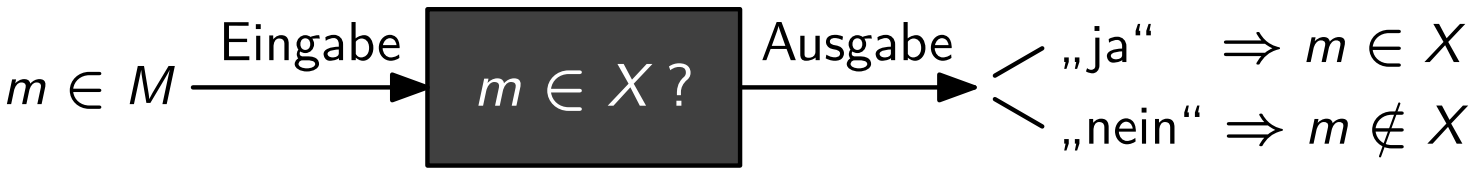
5 Charakterisierungen

6 Entscheidungsprobleme

Entscheidbarkeit

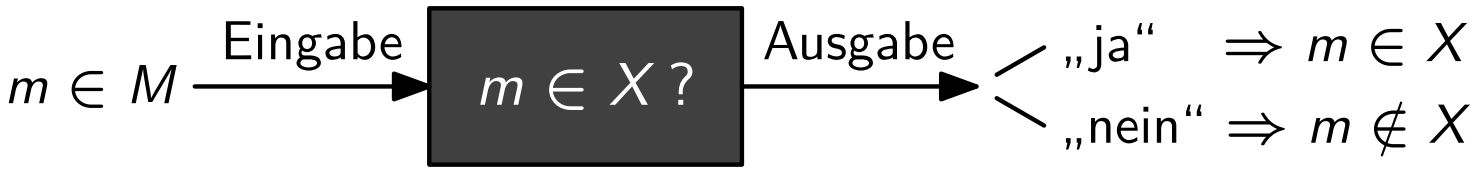
(Entscheidungs-)Problem

- ... ist eine Teilmenge $X \subseteq M$
  - Eingabe: $m \in M$; Frage: $m \in X$?
  - $m \in X$: **Ja-Instanzen;** $m \in M \setminus X$: **Nein-Instanzen**
- Beispiele:
  - X = Menge aller Primzahlen, $M = \mathbb{N}$
  - X = Menge aller NEAs $\mathcal{A}$ mit $L(\mathcal{A}) \neq \emptyset$,
    M = Menge aller NEAs
- man stelle sich eine Blackbox vor:



Entscheidbarkeit: X ist entscheidbar,
wenn es einen Algorithmus A gibt, der die Blackbox implementiert.

Komplexität



Komplexität:
zusätzliche Anforderungen an Zeit-/Speicherplatzbedarf von A

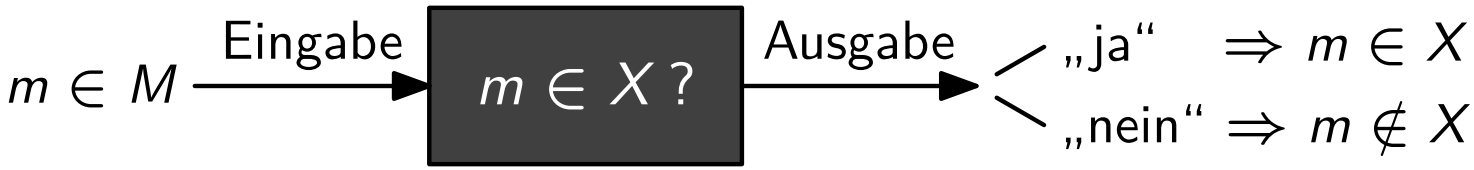
- **Polynomialzeit:** Anzahl Rechenschritte von A ist $\leqslant |m|^k$,
  $|m|$: Länge der Eingabe; k : beliebige Konstante
- **Polynomieller Platz:** von A benötigter Speicherplatz $\leqslant |m|^k$
- **Exponentialzeit:** Anzahl Rechenschritte von A ist $\leqslant 2^{|m|^k}$
- ...

Einige übliche Komplexitätsklassen

Name	Bedeutung	Beispiel-Problem
L	logarithm. Speicherplatz	Erreichbarkeit, ungerichtete Graphen
NL	nichtdetermin. log. Platz	Erreichbarkeit, gerichtete Graphen
P	Polynomialzeit	Primzahlen
NP	nichtdeterminist. Polyzeit	Erfüllbarkeit Aussagenlogik
PSpace	polynom. Speicherplatz	Erfüllbarkeit QBF
ExpTime	Exponentialzeit	Gewinnstrategie $n \times n$-Schach
NExpTime	nichtdet. Exponentialzeit	Clique f. schaltkreiscodierte Graphen
ExpSpace	exponentieller Platz	Äquiv. regulärer Ausdrücke mit „ $\cdot^2$ “
⋮	⋮	
	unentscheidbar	Erfüllbarkeit Prädikatenlogik

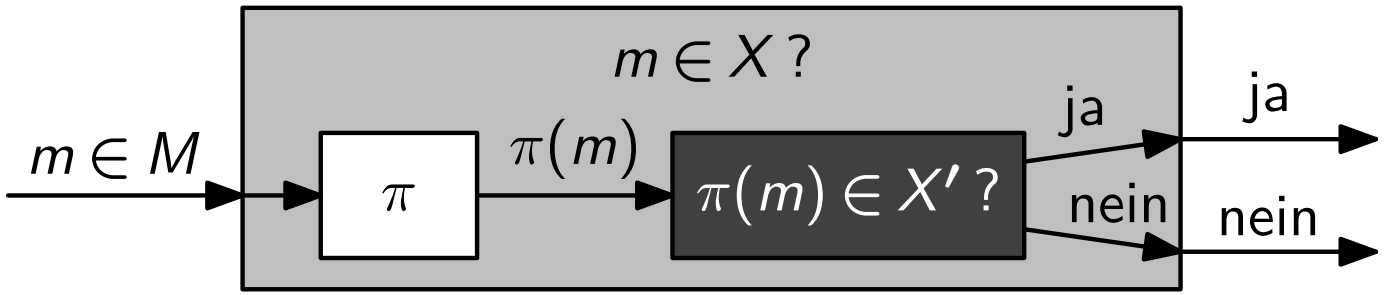
Komplementklassen: **coNL**, **coNP** etc.

Reduktion



(Polynomielle) Reduktion von $X \subseteq M$ nach $X' \subseteq M'$
ist eine (in Polyzeit) berechenbare Funktion $\pi : M \to M'$ mit

$$m \in X \qquad \text{gdw.} \qquad \pi(m) \in X'$$



Schreibweise: $X \leq X'$ bzw. $X \leq_P X'$ (X auf X' **reduzierbar**) T 1.7

Wenn alle Probleme aus Komplexitätsklasse $\mathcal{C}$ auf X reduzierbar, dann ist X **schwer für** $\mathcal{C}$.

Bestimmung der Komplexität

Normalerweise zeigt man, dass ein Problem $X \subseteq M$ ...

- **in** einer Komplexitätsklasse $\mathcal{C}$ liegt, indem man
  - einen Algorithmus A findet, der X löst
  - zeigt, dass A korrekt ist (*ja*/*nein*-Antworten) und terminiert
  - zeigt, dass A für jedes $m \in M$ **höchstens** die $\mathcal{C}$-Ressourcen braucht

    ... A kann z. B. eine Reduktion zu einem Problem aus $\mathcal{C}$ sein

- **schwer (hard) für** $\mathcal{C}$ ist, indem man
  - ein Problem $X' \subseteq M'$ findet, dass **schwer für** $\mathcal{C}$ ist
  - und eine Reduktion von X' nach X angibt

- **vollständig für** $\mathcal{C}$ ist, indem man zeigt, dass es
  - **in** $\mathcal{C}$ liegt und
  - **schwer für** $\mathcal{C}$ ist

Entscheidungsprobleme für endliche Automaten

- Betrachten wesentliche Eigenschaften von Sprachen (Sprachen repräsentiert durch NEAs oder reguläre Ausdr.)
  - Ist eine gegebene Sprache leer?
  - Ist ein gegebenes Wort w in einer Sprache L?
  - Beschreiben zwei Repräsentationen einer Sprache tatsächlich dieselbe Sprache?
- Wichtig für Anwendungen (siehe Einführung)
- Art der Repräsentation spielt manchmal eine Rolle: NEA, DEA, regulärer Ausdruck, Typ-3-Grammatik etc.

  Wir betrachten im Folgenden **NEAs und DEAs.**

Das Leerheitsproblem

Eingabe: NEA (oder DEA) $\mathcal{A}$

Frage: Ist $L(\mathcal{A}) = \emptyset$?

d. h. $\mathbf{LP_{NEA}} = \{\mathcal{A} \mid \mathcal{A} \text{ NEA}, L(\mathcal{A}) = \emptyset\}$,
$\mathbf{LP_{DEA}} = \{\mathcal{A} \mid \mathcal{A} \text{ DEA}, L(\mathcal{A}) = \emptyset\}$

Satz 1.14

$\mathrm{LP_{NEA}}$ und $\mathrm{LP_{DEA}}$ sind entscheidbar und **coNL**-vollständig.

Beweis.

- Entscheidbarkeit (in Polyzeit): siehe ThI 1
- **coNL**-Zugehörigkeit:
  Reduktion zu Erreichbarkeit in gerichteten Graphen, siehe T1.7
- **coNL**-Härte:
  Reduktion *von* Erreichbarkeit, analog □

Das Wortproblem

Eingabe: NEA (oder DEA) $\mathcal{A}$, Wort $w \in \Sigma^*$

Frage: Ist $w \in L(\mathcal{A})$?

d. h. $\mathbf{WP_{NEA}} = \{(\mathcal{A}, w) \mid \mathcal{A}$ NEA, $w \in L(\mathcal{A})\}$,
$\mathbf{WP_{DEA}} = \{(\mathcal{A}, w) \mid \mathcal{A}$ DEA, $w \in L(\mathcal{A})\}$

Satz 1.15

$\mathrm{WP_{NEA}}$ und $\mathrm{WP_{DEA}}$ sind entscheidbar.
$\mathrm{WP_{NEA}}$ ist **NL**-vollständig; $\mathrm{WP_{DEA}}$ ist **L**-vollständig.

Beweis.

- Entscheidbarkeit (in Polyzeit): siehe ThI 1
  (Reduktion zum LP: $w \in L(\mathcal{A})$ gdw. $L(\mathcal{A}) \cap L(\mathcal{A}_w) \neq \emptyset$)
- **NL**-Vollst.: $\approx$ Erreichbarkeit in gerichteten Graphen □

Das Äquivalenzproblem

Eingabe: NEAs (oder DEAs) $\mathcal{A}_1, \mathcal{A}_2$

Frage: Ist $L(\mathcal{A}_1) = L(\mathcal{A}_2)$?

d. h. $\ddot{\text{A}}\text{P}_{\text{NEA}} = \{(\mathcal{A}_1, \mathcal{A}_2) \mid \mathcal{A}_1, \mathcal{A}_2 \text{ NEAs}, L(\mathcal{A}_1) = L(\mathcal{A}_2)\}$,
$\ddot{\text{A}}\text{P}_{\text{DEA}} = \{(\mathcal{A}_1, \mathcal{A}_2) \mid \mathcal{A}_1, \mathcal{A}_2 \text{ DEAs}, L(\mathcal{A}_1) = L(\mathcal{A}_2)\}$

Satz 1.16

$\ddot{\text{A}}\text{P}_{\text{NEA}}$ und $\ddot{\text{A}}\text{P}_{\text{DEA}}$ sind entscheidbar.
$\ddot{\text{A}}\text{P}_{\text{NEA}}$ ist **PSPACE**-vollständig; $\ddot{\text{A}}\text{P}_{\text{DEA}}$ ist **NL**-vollständig.

Beweis.

- Entscheidbarkeit (in Polyzeit): siehe ThI 1
  (Red. zum LP: $L(\mathcal{A}_1) = L(\mathcal{A}_2)$ gdw. $L(\mathcal{A}_1) \triangle L(\mathcal{A}_2) = \emptyset$)
- Komplexität: Automat für $L(\mathcal{A}_1) \triangle L(\mathcal{A}_2)$ ist exponentiell in der Größe der Eingabe-NEAs; linear im Fall von DEAs

  Details: siehe [Holzer & Kutrib 2011] □

Das Universalitätsproblem

Eingabe: NEA (oder DEA) $\mathcal{A}$

Frage: Ist $L(\mathcal{A}) = \Sigma^*$?

d. h. $\mathbf{UP_{NEA}} = \{\mathcal{A} \mid \mathcal{A} \text{ NEA}, L(\mathcal{A}) = \Sigma^*\}$,
$\mathbf{UP_{DEA}} = \{\mathcal{A} \mid \mathcal{A} \text{ DEA}, L(\mathcal{A}) = \Sigma^*\}$

Satz 1.17

$\mathrm{UP_{NEA}}$ und $\mathrm{UP_{DEA}}$ sind entscheidbar.

Beweis: Übungsaufgabe

Überblick Entscheidungsprobleme für NEAs/DEAs

Problem	entscheidbar?	für DEAs effizient lösbar?	für NEAs effizient lösbar?
LP	✓	✓	✓
WP	✓	✓	✓
ÄP	✓	✓	✘*
UP	✓	✓	✘*

* unter den üblichen komplexitätstheoretischen Annahmen (z. B. PSPACE $\neq$ P)

Literatur für diesen Teil (Basis)

John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman.

Introduction to Automata Theory, Languages and Computation.

2. Auflage, Addison-Wesley, 2001.

Kapitel 1,2.

Verfügbar in SUUB (verschiedene Auflagen, auch auf Deutsch)

Meghyn Bienvenu.

Automata on Infinite Words and Trees.

Vorlesungsskript, Uni Bremen, WS 2009/10.

`http://www.informatik.uni-bremen.de/tdki/lehre/ws09/automata/automata-notes.pdf`

Literatur für diesen Teil (weiterführend)

Markus Holzer, Martin Kutrib.

Descriptional and computational complexity of finite automata – A survey.

Information and Computation 209:456–470, 2011.

Kapitel 3: sehr umfassender Überblick über Entscheidungsprobleme für endliche Automaten und deren Komplexität; viel Literatur

Verfügbar in SUUB (elektronisch)
`https://doi.org/10.1016/j.ic.2010.11.013`