Grundlagen Theoretischer Informatik I

SoSe 2011 in Trier

Henning Fernau
Universität Trier
fernau@uni-trier.de

Grundlagen Theoretischer Informatik I Gesamtübersicht

- Organisatorisches; Einführung
- Logik & Beweisverfahren
- Mengenlehre
- reguläre Sprachen

Weitere Fragen an vorgegebenen DEA A: (evtl. zweiter DEA A')

- Ist $\mathrm{L}(\mathrm{A}) = \emptyset$? *Leerheitsproblem*
- Ist $\mathrm{L}(\mathrm{A}) = \mathrm{L}(\mathrm{A}')$? *Äquivalenzproblem*
- Ist $\mathrm{L}(\mathrm{A}) \subseteq \mathrm{L}(\mathrm{A}')$? *Teilmengenproblem*
- Ist $\mathrm{L}(\mathrm{A})$ endlich ? *Endlichkeitsproblem*

Leerheitsproblem

Wir haben schon zwei Methoden kennen gelernt, die Menge E der (vom Startzustand aus) erreichbaren Zustände zu berechnen. Die vom Automaten beschriebene Sprache ist leer gdw. E keine Endzustände enthält.

Betrachte zu DEA $A = (Q, \Sigma, \delta, q_0, F)$ die
erweiterte 1-Schritt-Zustandserreichbarkeitsrelation

$$R = \{(p, q) \in Q \times Q \mid \exists a \in \Sigma : \delta(p, a) = q\} \cup F \times \{q_f\},$$

wobei $q_f \notin Q$ und mit $Q' = Q \cup \{q_f\}$ gilt $R \subset Q' \times Q'$.
$L(A) = \emptyset$ gdw. $(q_0, q_f) \notin R^*$.
Die Existenz einer Punkt-zu-Punkt-Verbindung kann sogar in Linearzeit $O(|Q|)$ berechnet werden. (z.B.: Dijkstras Algorithmus)

Teilmengen- und Äquivalenzproblem

Beobachte: $\mathrm{L}(A) \subseteq \mathrm{L}(A')$ gdw. $\mathrm{L}(A) \setminus \mathrm{L}(A') = \emptyset$.

Daher:

1. Aus gegebenen DEAs A und A' berechne DEA A'' mit $\mathrm{L}(A'') = \mathrm{L}(A) \setminus \mathrm{L}(A')$.
Dies geht direkt mit *Produktautomatenkonstruktion* oder folgt mit den bekannten Mengenalgebragesetzen.

2. Entscheide ob $\mathrm{L}(A'') = \emptyset$ mit vorher skizziertem Verfahren.

Wegen $\mathrm{L}(A) = \mathrm{L}(A')$ gdw. $\mathrm{L}(A) \subseteq \mathrm{L}(A')$ und $\mathrm{L}(A') \subseteq \mathrm{L}(A)$ folgt damit die Entscheidbarkeit des Äquivalenzproblems.

Endlichkeitsproblem zu DEA $A = (Q, \Sigma, \delta, q_0, F)$

Wie im Beweis zum Pumping-Lemma sieht man:
Ist $L(A)$ unendlich, so gibt es einen Zustand q, einen (evtl. leeren) Weg vom Anfangszustand q_0 nach q, einen nicht-leeren Weg von q nach q und einen (evtl. leeren) Weg von q zu einem Endzustand.
Die Umkehrung gilt sogar trivialer Weise!
Bezeichnet R die 1-Schritt-Zustandserreichbarkeitsrelation, so berechne
E': die Menge der Zustände, die sowohl erreichbar als auch *co-erreichbar* sind (d.h., für alle $q \in E'$ gilt: $(q_0, q) \in R^*$ und $\exists q_f \in F : (q, q_f) \in R^*$).
Dann gilt: $L(A)$ ist unendlich gdw. $\exists q \in E' : (q, q) \in R^+$.

EA zur Mustersuche (Pattern Matching)

Beispiel: Finde Vorkommen des Musters (Pattern)

$$p = ababac$$

in einem Text $t \in \{a, b, c\}^*$.

Wir haben schon früher gesehen:
NEAs sind nützlich für diese Aufgabe.

In RA-artiger Notation beobachten wir:
Lemma: $t \in \Sigma^*$ enthält das Muster p gdw. $t \in \Sigma^*\{p\}\Sigma^*$.

Klar: Die Bedingung lässt sich sofort in NEA umsetzen.
Frage: Wie lassen sich hierzu DEAs nutzen ?

DEA zur Mustersuche

Vorteil wäre: Linearzeitalgorithmus zur Mustersuche.
Dagegen naiv: quadratischer Algorithmus zur Mustersuche; nämlich
Problem: Zurücksetzen bei “falschem Alarm”.

Ziel: Vermeide Potenzautomatenkonstruktion.

Wie geht das ?

Einige Hilfsbegriffe

u heißt *Teilwort* von $x \in \Sigma^*$ gdw. $x \in \Sigma^*\{u\}\Sigma^*$.

Mustersuche ist also die Suche nach Teilwörtern.

u heißt *Präfix* oder *Anfangswort* von $x \in \Sigma^*$ gdw. $x \in \{u\}\Sigma^*$.

u heißt *Suffix* oder *Endwort* von $x \in \Sigma^*$ gdw. $x \in \Sigma^*\{u\}$.

Ein Teilwort / Präfix / Suffix u von x heißt *echt* gdw. $\ell(u) < \ell(x)$.

Ein echtes Teilwort u von x, das sowohl Präfix als auch Suffix von x ist, heißt *Rand (der Breite $\ell(u)$)* von x.

Beispiel: Sei $x = abacab$.

Die echten Präfixe von x sind $\lambda, a, ab, aba, abac, abaca$;

die echten Suffixe von x sind $\lambda, b, ab, cab, acab, bacab$.

Ränder von x sind λ, ab; der Rand ab hat die Breite 2.

DEA-Konstruktionsidee für Mustersuche nach Knuth/Morris/Pratt (Matjasewitsch)

Frage: Wo darf DEA nach einem Mismatch wieder "einsetzen" ?
Idee: Verwende die im bisher gelesenen Präfix des Musters steckende Info !

Betrachte

Pos.	0	1	2	3	4	5	6	7	8
t	a	b	c	a	b	c	a	b	d
p	a	b	c	a	b	d			
Forts.				a	b	c	a	b	d

Die Symbole an den Positionen 0, . . . , 4 haben übereingestimmt. Der Vergleich $c - d$ an Position 5 ergibt einen Mismatch. Das Muster kann bis Position 3 weitergeschoben werden, und der Vergleich wird ab Position 5 des Textes fortgesetzt.

Wie weit dürfen wir schieben ?

Die *Schiebedistanz* richtet sich nach dem breitesten Rand des übereinstimmenden Präfixes des Musters.
Im Beispiel ist das übereinstimmende Präfix abcab; es hat die Länge $\mathrm{j} = 5$.
Sein breitester Rand ist ab mit der Breite $\mathrm{b} = 2$.
Die Schiebedistanz beträgt $\mathrm{j} - \mathrm{b} = 5 - 2 = 3$.

Die in der *Vorlaufphase* zu gewinnende Information besteht also darin, für jedes Präfix des Musters die Länge seines breitesten Randes zu bestimmen.

Eine wichtige Beobachtung für die Vorlaufphase:

Lemma: Seien r, s Ränder eines Wortes x mit $\ell(r) < \ell(s)$. Dann ist r ein Rand von s.

Beweis: Das Bild zeigt schematisch x mit den Rändern r und s.

Als Rand von x ist r Präfix von x und damit, weil kürzer als s, auch echtes Präfix von s.

Aber r ist auch Suffix von x und damit echtes Suffix von s. Also ist r Rand von s.

Ist s der breiteste Rand von x, so ergibt sich der nächstschmale Rand r von x als breitester Rand von s usw.

Ein weiterer wichtiger Begriff

Sei $x \in \Sigma^*$ und $a \in \Sigma$. Ein Rand r von x lässt sich durch a *fortsetzen*, wenn ra Rand von xa ist.

Bild $\rightsquigarrow$ Ein Rand r der Breite j von x lässt sich durch a fortsetzen, wenn $x[j] = a$.

Die Vorlaufphase

In der Vorlaufphase wird ein Array b der Länge $m + 1$ berechnet.
Der Eintrag $b[i]$ enthält für jedes Präfix der Länge i des Musters die *Breite seines breitesten Randes* $(i = 0, \ldots, m)$.
Das Präfix λ der Länge $i = 0$ hat keinen Rand; daher wird $b[0] = -1$ gesetzt.

Sind die Werte $b[0], \ldots, b[i]$ bereits bekannt, so ergibt sich $b[i+1]$, indem geprüft wird, ob sich ein Rand des Präfixes $p_0 \ldots p_{i-1}$ durch p_i fortsetzen lässt.
Dies ist der Fall, wenn $p_{b[i]} = p_i$ ist (Bild !).
Die zu prüfenden Ränder ergeben sich nach obigem Lemma in absteigender Breite aus den Werten $b[i]$, $b[b[i]]$ usw.

Ein Beispiel

Beispiel: Für das Muster $p = ababaa$ ergeben sich die Randbreiten im Array b wie folgt. Beispielsweise ist $b[5] = 3$, weil das Präfix $ababa$ der Länge 5 einen Rand der Breite 3 hat.

j	0	1	2	3	4	5	6
p[j]	a	b	a	b	a	a	
b[j]	−1	0	0	1	2	3	1

Die Vorlaufphase: In C-Code:

```
void kmpPreprocess()
{
    int i=0, j=-1;
    b[i]=j;
    while (i<m)
    {
        while (j>=0 && p[i]!=p[j]) j=b[j];
        i++; j++;
        b[i]=j;
    }
}
```

Knuth-Morris-Pratt Such–Algorithmus

Es werden sogar alle Treffer gemeldet.

```
void kmpSearch()
{
    int i=0, j=0;
    while (i<n)
    {
        while (j>=0 && t[i]!=p[j]) j=b[j];
        i++; j++;
        if (j==m)
        {
            report(i-j);
            j=b[j];
        }
    }
}
```

Sehen Sie den DEA ?

Ein Beispiel mit $p = ababaa$.

```
a b a b b a b a a ...
a b a b a a
    a b a b a a
      a b a b a a
        a b a b a a
          a b a b a a
            a b a ...
```

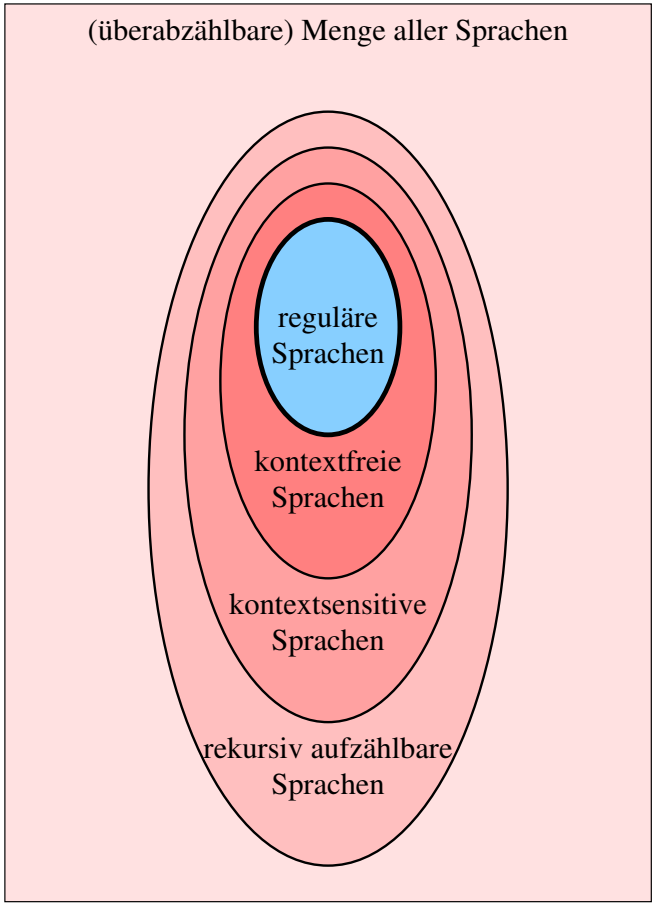
j	0	1	2	3	4	5	6
p[j]	a	b	a	b	a	a	
b[j]	−1	0	0	1	2	3	1

Automaten mit Ausgabe: Einige Bemerkungen

- Mealy-Automaten
- Moore-Automaten
- Die technische Sicht
- Die Sicht des Compilerbaus

Bei formalen Sprachen unterscheidet man zunächst vier Grundtypen:

Chomsky-Hierarchie:



- Typ-0-Sprachen: *rekursiv-aufzählbar*
  Beschreibung der Möglichkeiten (und der Grenzen) des Programmierens
- Typ-1-Sprachen: *kontextsensitiv*
- Typ-2-Sprachen: *kontextfrei*
  Syntax (“Grammatik”) von Programmiersprachen
- Typ-3-Sprachen: *regulär/rechtslinear*
  lexikalische Analyse bei Programmiersprachen, Suchalgorithmen in Texten,
  u.v.m.

Chomsky-Grammatik

Eine *Grammatik* ist ein 4-Tupel $G = (N, T, P, S)$ mit:

- N ist eine endliche Menge von *Variablen* oder *Nicht-Terminal-Zeichen*.
- T ist eine endliche Menge, das *Terminalalphabet*, wobei $N \cap T = \emptyset$.
- P ist eine endliche Menge von *Regeln* oder *Produktionen*
  $P \subseteq (N \cup T)^+ \times (N \cup T)^*$
- $S \in N$ ist das *Startsymbol*
- Entfällt die Unterscheidung zwischen N und T, d.h., es ist nur ein *(Gesamt-) Alphabet* Σ gegeben, und gilt $P \subseteq \Sigma^* \times \Sigma^*$, $|P| < \infty$, so heißt (Σ, P) ein *(sequentielles) Ersetzungssystem*.

Seien nun u, v aus $(N \cup T)^*$.
Wir schreiben $u \Rightarrow_G v$, falls

$$u = xyz, \quad v = xy'z$$

für $x, z \in (N \cup T)^*$, wobei $y \to y'$ eine Regel aus P ist.
Sprechweise: *v kann direkt aus u abgeleitet werden*.

In der Folge werden wir das tiefgestellte G auch weglassen.

Eine Folge von direkten Ableitungen

$$w_1 \Rightarrow w_2 \Rightarrow \ldots \Rightarrow w_n$$

heißt *Ableitung von w_n aus w_1*.

Ohne Angabe der 'Zwischenwörter' schreiben wir: $w_1 \Rightarrow^* w_n$

$\Rightarrow^*$ ist die reflexiv-transitive Hülle von $\Rightarrow$

$$L(G) := \{w \in T^* \mid S \Rightarrow^* w\}$$

Wir nennen $L(G)$ die *von der Grammatik G erzeugte Sprache*.

Endliche Automaten als Ersetzungssysteme

Ersetzungssysteme sind allgemeiner als Grammatiken.
So kann man endliche Automaten wie folgt beschreiben:
Ein Ersetzungssystem $E = (\Sigma, P)$ heißt *endlicher Automat*, falls
$\Sigma = Z \cup T$, $Z \cap T = \emptyset$, $P \subseteq ZT \times Z$.
Z: Zustände; T: Eingabezeichen
Sei $z_0 \in Z$ Anfangszustand und $Z_f \subseteq Z$ Endzustände, so *akzeptiert* (E, z_0, Z_f) die Sprache:

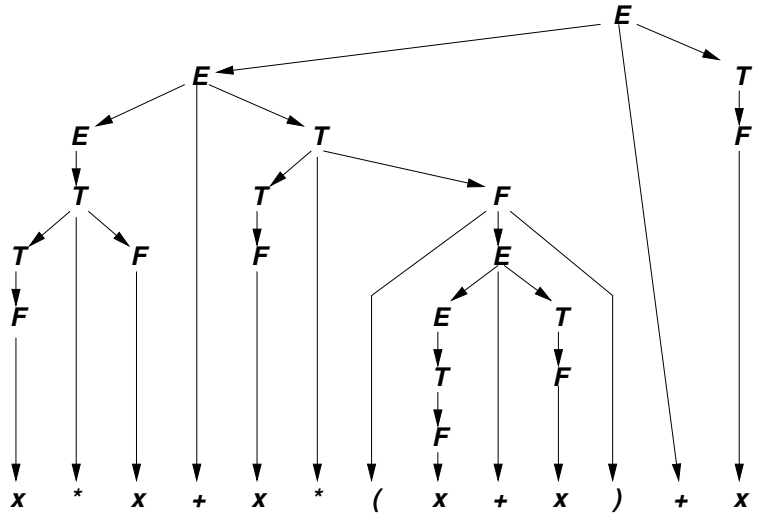
$$L_a(E, z_0, Z_f) = \{w \in T^* \mid \exists z_f \in Z_f : z_0 w \Rightarrow^* z_f\}$$

Aufgaben: 1. Geben Sie in diesem Formalismus einen endlichen Automaten an, der die Sprache $a^*bb^*c^*$ beschreibt.
2. Wie kann man in diesem Formalismus einen deterministischen endlichen Automaten beschreiben?

arithmetische Ausdrücke Sei $G = (\{E, T, F\}, \{(,), x, +, *\}, P, E)$ mit

$$P = \{E \to T, E \to E + T, T \to F, T \to T * F, F \to x, F \to (E)\}$$

Gegeben sei das Wort: $x * x + x * (x + x) + x$



$$P = \{E \to T, E \to E + T, T \to F, T \to T * F, F \to x, F \to (E)\}$$

Der *Syntaxbaum* beschreibt die einzelnen Ableitungsschritte vom Startsymbol bis zum Wort:

$$
\begin{aligned}
E &\Rightarrow E + T \\
&\Rightarrow T + T + T \\
&\Rightarrow F * F + T + T \\
&\Rightarrow x * x + T + T \\
&\Rightarrow x * x + F * F + T \\
&\Rightarrow x * x + x * (E) + T \\
&\Rightarrow x * x + x * (T + T) + T \\
&\Rightarrow x * x + x * (x + T) + T \\
&\Rightarrow x * x + x * (x + x) + T \\
&\Rightarrow x * x + x * (x + x) + x
\end{aligned}
\qquad
\begin{aligned}
&\Rightarrow E + T + T \\
&\Rightarrow T * F + T + T \\
&\Rightarrow x * F + T + T \\
&\Rightarrow x * x + T * F + T \\
&\Rightarrow x * x + x * F + T \\
&\Rightarrow x * x + x * (E + T) + T \\
&\Rightarrow x * x + x * (F + T) + T \\
&\Rightarrow x * x + x(x + F) + T \\
&\Rightarrow x * x + x * (x + x) + F
\end{aligned}
$$