Theoretische Informatik

Vorlesungsskript

Mitschrift von Falk-Jonatan Strube

Vorlesung von Dr. Boris Hollas

29. März 2016

Inhaltsverzeichnis

Inhalte

Grundlage: Grundkurs Theoretische Informatik [1]

- Formale Sprachen
  - Reguläre Sprachen
    - Endliche Automaten
    - Reguläre Ausdrücke
  - Nichtreguläre Sprachen
  - Kontextfreie Sprachen
    - Kellerautomaten
    - Grammatiken
- Berechenbarkeit
  - Halteproblem
- Komplexitätsklassen
  - P
  - NP
  - NP-vollständige Probleme

1 Automaten und Formale Sprachen

Def.: Ein Alphabet ist eine Menge $\Sigma \neq \emptyset$ (Symbole in Σ – müssen nicht einzelne Buchstaben sein, auch Wörter usw. [bspw. „if“ oder „else“ im Alphabet der Programmiersprache C]).

Def.: Für $w_1, ..., w_n \in \Sigma$ ist $w = w_1...w_n$ ein Wort der Länge n.
Σ^n beschreibt alle Worte mit der Länge genau n
Das Wort ε ist das *leere Wort*.
Die Menge aller Wörter bezeichnen wir mit Σ^* (einschließlich dem leeren Wort).

Bsp.: $\Sigma = \{a, b, c\} \quad \rightarrow \Sigma^* = \{\varepsilon, a, b, c, aa, ab, ac, aaa, ...\}$

Def.: Für Wörter $a, b \in \Sigma^*$ ist ab die Konkatenation dieser Wörter.
Für ein Wort w ist w^n die n-fache Konkatenation von w, wobei $w^0 = \varepsilon$.

Bemerkung: Für alle $w \in \Sigma^*$ gilt $\varepsilon w = w = w\varepsilon$. ε ist also das neutrale Element der Konkatenation.

Def.: Eine *formale Sprache* ist eine Teilmenge von σ^*.

Def.: Für Sprachen A, B ist $AB = \{ab \mid a \in A,\ b \in B\}$ sowie $A^n = \prod_{i=1}^{n} A$, wobei $A^0 = \{\varepsilon\}$.

Bemerkung: $\emptyset, \varepsilon, \{\varepsilon\}$ sind unterschiedliche Dinge (leere Menge, leeres Wort, Menge mit leerem Wort).

Bemerkung: Σ^* lässt sich ebenfalls definieren durch $\Sigma^* = \bigcup_{n \geq 0} \Sigma^n$.
Ferner ist $\Sigma^+ = \Sigma^* - \{\varepsilon\}$.

1.1 Reguläre Sprachen

1.1.1 Deterministische endliche Automaten (DFA)

Bsp.:
ABB1
(Pfeil zeigt auf Startzustand, Endzustand ist doppelt umrandet)
Dieser DFA akzeptiert alle Wörter über $\Sigma = \{a, b, c\}$, die abc enthalten.
Deterministisch: Es gibt genau ein Folgezustand. Von jedem Knoten aus gibt es genau eine Kante für jedes Zeichen, nicht mehrere und nicht keine.

Bsp.:
ABB2
Dieser DFA erkennt die Sprache $\{a, b, c\}^*$.

Def.: Ein DFA ist ein Tupel $\mathcal{M} = (Z, \Sigma, \delta, z_0,)$

- Z: Menge der Zustände
- Σ: Eingabealphabet
- δ: Überführungsfunktion $Z \times \Sigma \to Z$. Dabei bedeutet $\delta(z, a) = z'$, dass $\mathcal{M}$ im Zustand z für das Zeichen a in den Zustand z' wechselt.
- $z_0 \in Z$: Startzustand
- E: Menge der Endzustände

δ: ABB3

Def.: Die erweiterte Überführungsfunktion $\hat{\delta} : Z \times \Sigma^* \to Z$ ist definiert durch

$$\hat{\delta}(z, w) = \begin{cases} z & \text{für } w = \varepsilon \\ \hat{\delta}(\delta(z, a), x) & \text{für } w = ax \text{ mit } a \in \Sigma, x \in \Sigma^* \end{cases}$$

Dazu vergleichbarer C-Code:

```
int δ̂(int z, char* w){
  if ( strlen(w) == 0 )
    return z;
  else
    return δ̂(δ(z, w[0]), w[1]);
```

Veranschaulichung:
ABB4
Die erweiterte Überführungsfunktion bestimmt den Zustand nach dem vollständigen Lesen eines Wortes.

Bsp.:
ABB5

$$\begin{aligned} \hat{\delta}(z_0, aaba) &= \hat{\delta}(\delta(z_0, a), aba) = \\ \hat{\delta}(z_0, aba) &= \hat{\delta}(\delta(z_0, a), ba) = \\ \hat{\delta}(z_0, ba) &= \hat{\delta}(\delta(z_0, b), a) = \\ \hat{\delta}(z_E, a) &= \hat{\delta}(\delta(z_E, a), \varepsilon) = \\ \hat{\delta}(z_E, \varepsilon) &= z_E \end{aligned}$$

Die von $\mathcal{M}$ *akzeptierte Sprache* ist $L(\mathcal{M}) = \{w \in \Sigma^* \mid \hat{\delta}(z_0, w) \in E\}$

1.1.2 Nichtdeterministischer endliche Automaten (NFA)

ABB23
NFA, der alles akzeptiert, was abc enthält:
ABB24
Beispiel: Wort $abaabcab$

Beispiel: NFA, der alle Wörter akzeptiert, die auf 001 enden:
ABB25
Akzeptierte Worte unter anderem: 01011001, 001001
Ein Wort wird vom NFA akzeptiert, wenn es einen Weg, ausgehend von einem Startzustand, gibt, mit den ein End-Zustand erreicht wird.
Der NFA „weiß“ nicht, welcher Pfad zu durchlaufen ist; diesen muss der Benutzer ermitteln (wie bei einer Straßenkarte).
Ein NFA lässt sich formalisieren durch ein Tupel $\mathcal{M} = (Z, \Sigma, \delta, S, E)$

- Z: Zustände
- Σ: Eingabealphabet
- δ: $Z \times \Sigma \to \mathcal{P}(Z)$ Überführungsfunktion (bildet ab in Potenzmenge von Z)
- S: Menge der Startzustände
- E: Menge der Endzustände

Dabei bedeutet $\delta(z, a) \ni z'$, dass der NEA im Zustand z für die Eingabe a die Möglichkeit besitzt, in den Zustand z' zu wechseln.

1.1.3 Umwandlung eines NFA in einen DFA

Wir wollen den NFA
ABB 27
in einen DFA umwandeln. Der Startzustand des DFA besteht aus den Startzuständen des NFA:
ABB 28
Betrachten die Folgezustände für $a \in \Sigma$:
ABB 29
nächster Schritt:
ABB 30
nächster Schritt:
ABB 31
weitere Schritte:
$z_0, b : \{z_0\}$
$z_1, b : \{z_2\}$
ABB 32
$z_0, c : \{z_0\}$
$z_1, c : \{\}$
ABB 33
usw.:
ABB 34
Wenn ein Zustand des DFA einen Endzustand des NFA enthält, so ist es ein Endzustand.
Der auf diese Weise erhaltene DFA kann Zustände enthalten, die sich zu einem Zustand zusammen fassen lassen. Mit dem Algorithmus Minimalautomat lässt sich ein DFA konstruieren, der minimal bezüglich der Anzahl seiner Zustände ist. Der Minimalautomat ist eindeutig, d.h. Minimalautomaten unterscheiden sich höchstens in der Benennung der Zustände.

1.1.4 Reguläre Ausdrücke

Def.: Sei Σ ein Alphabet. Ein *regulärer Ausdruck* E sowie die durch E *erzeugte Sprache* $L(E)$ sind induktiv definiert:

- $\emptyset$ ist ein regulärer Ausdruck und $L(\emptyset) = \emptyset$.
  Bsp.:
  ABB35

- Für $a \in \Sigma \cup \{\varepsilon\}$ ist a ein regulärer Ausdruck und $L(a) = \{a\}$.

- Für reguläre Ausdrücke E_1, E_2 sind $(E_1|E_2),\ (E_1E_2),\ (E_1^*)$ reguläre Ausdrücke (hier: $|$ =„oder“) und $L(E_1|E_2) = L(E_1) \cup L(E_2),\ L(E_1E_2) = L(E_1)L(E_2),\ L(E_1^*) = L(E_1)^*$ die davon erzeugten Sprachen:

Ausdruck	Sprache
$E_1\|E_2$	$L(E_1\|E_2) = L(E_1) \cup L(E_2)$
E_1E_2	$L(E_1E_2) = L(E_1)L(E_2)$
E_1^*	$L(E_1^*) = L(E_1)^*$

  Hinweis: $E^+ = EE^*$, $E? = \varepsilon|E$

Bsp.:

- $L((0|1)^*) = (L(0|1))^* = (L(0) \cup L(1))^* = (\{0\} \cup \{1\})^* = \{0,1\}^*$

- Regulärer Ausdruck über $\Sigma = \{a, b, c\}$, der die gleiche Sprache erzeugt wie der DFA aus dem letzten Automaten-Beispiel:
  $L((a|b|c)^*abc(a|b|c)^*) = \{a, b, c\}^*\{abc\}\{a, b, c\}^*$

Satz: Reguläre Ausdrücke erzeugen genau die regulären Sprachen.

Skizze: Umwandlung eines regulären Ausdrucks in einen endlichen Automaten.

- $\emptyset$: ABB 40

- $a \in \Sigma$: ABB 41 1.
  ε: ABB 41 2.

- Seien E_1, E_2 reguläre Ausdrücke und $\mathcal{M}_1, \mathcal{M}_2$ DFAs mit $L(E_1) = L(\mathcal{M}_1), L(E_2) = L(\mathcal{M}_2)$.
  - $E_1|E_2$: $\mathcal{M}_1, \mathcal{M}_2$ sind zusammen ein NFA, der $L(\mathcal{M}_1) \vee L(\mathcal{M}_2)$ erkennt.
    Bsp.: $E_1 = a, E_2 = b$
    ABB 42
  - E_1E_2: $\mathcal{M}_1, \mathcal{M}_2$ müssen hintereinander geschaltet werden, wobei ggf. neue Kanten eingefügt werden müssen. Dazu betrachtet man die Kante nach der neuen Verbindung und erzeugt dem entsprechend die Übergangskanten.
    ABB 43
  - E_1^*: Es müssen Kanten zurück zum Startzustand eingefügt werden
    ABB 44

Der Beweis für die umgekehrte Richtung (DFA $\rightarrow$ reg. Ausdruck) ist schwierig.

Bsp.:

- $E = 0(0|1)^*$
  ABB 45

Mitschrift von
Falk-Jonatan Strube

- ABB 46
  Beobachtungen:
  - um zum Endzustand zu kommen, braucht man eine 1.
  - vor der 1 kann ε stehen, oder beliebig viele 0en der 1en.

  $\Rightarrow E = (0|1)^*1$

1.1.5 Das Pumping-Lemma

Wenn ein DFA ein Wort akzeptiert, das mindestens so lang ist wie die Anzahl seiner Zustände, dann muss er einen Zustand zweimal durchlaufen (Schubfachprinzip). Daraus folgt, dass der DFA dabei eine Schleife durchläuft.

Bsp.:
ABB 47
Für $x = abcdecfg$ durchläuft der Automat eine Schleife: $x = ab\boxed{cde}cfg$. Daher akzeptiert der DFA auch alle Wärter $ab(cde)^kcfg$ für $k \geq 0$.

Satz: (Pumping Lemma)
Für jede reguläre Sprache L gibt es ein $n > 0$ (n: Anzahl Zustände des Minimalautomaten), so dass es für alle Wörter $x \in L$ mit $|x| \geq n$ eine Zerlegung $x = uvw$ gibt (in vorherigem Bsp.: $u = ab,\ v = cde\ w = cfg$), so dass gilt:

1.) $|v| \geq 1$, $|uv| \geq n$ $\quad (u, w$ können auch ε sein)

2.) $uv^kw \in L$ für alle $k \geq 0$.

Ohne Einschränkung ist n die Anzahl Zustände des Minimalautomaten.
$\Rightarrow \forall$ regulären Sprachen $L \quad \exists\, n > 0 \quad \forall\, x \in L,\ |x| \geq n \quad \exists\, u, v, w$ mit $x = uvw$ und $|v| \geq 1, |uv| \geq n \quad \forall\, k \geq 0\ uv^kw \in L$.
Das Pumping-Lemma lässt sich nutzen, um zu zeigen, dass eine Sprache nicht regulär ist.

Bsp.: Wir zeigen, dass $L = \{a^nb^n | n \in \mathbb{N}\}$ nicht regulär ist.
Problemstellung: Der Automat kann sich das n nicht „merken“, um nach n as wieder n bs zu erzeugen.
Beweis (Widerspruch):

- Angenommen, L sei regulär.
- Nach Pumping-Lemma gibt es dann ein $n > 0$, so dass sich alle $x \in L$ mit $|x| \geq n$ gemäß Pumping-Lemma zerlegen lassen.
- Sei $x = a^nb^n$.
- Angenommen v enthalte ein b, dann wäre $|uv| > n$.
  Aus $|uv| \geq n$ folgt aber, dass v kein b enthält. aus $|v| \geq 1$ folgt, dass v mindestens ein a enthält.
- Das Wort uw enthält daher weniger as als bs und kann somit nicht in L enthalten sein (denn w enthält b^n, da v mindestens ein a enthält, ist durch uw mindestens ein a „verloren gegangen“: $uw = a^{n/|v|}b^n$) und ist deshalb nicht in L enthälten, Widerspruch ↯ #

Vorgehen:

- ist regulär

Mitschrift von
Falk-Jonatan Strube

- Def. Pumping Lemma
- x finden (trivial)
- durch 1.) einschränken
- durch 2.) zum Widerspruch führen

Literatur

[1] Boris Hollas. *Grundkurs Theoretische Informatik: Mit Aufgaben und Anwendungen*. Springer Berlin, 2015.