Algorithmen mit konstantem Platzbedarf

HANDOUT

1 Endliche Automaten

- Wir wollen nachprüfen, dass ein Wort in einer Sprache liegt (d.h. gültig ist), ohne dafür die gesamte Sprache erzeugen zu müssen.
- Endliche Automaten arbeiten bereits nach dem Eingabe-Verarbeitung-Ausgabe-Prinzip eines "echten" Computers, wobei diese Schritte vergleichsweise simpel sind.

Definition 1 (Alphabet, Wort, Sprache) Ein *Alphabet* ist eine nichtleere endliche Menge. Ein *Wort* über einem Alphabet Σ ist eine endliche (möglicherweise leere!) Folge von Symbolen aus Σ. Wir schreiben für die Menge alle möglichen Worte über Σ auch Σ^*. Eine *Sprache* über Σ ist eine Menge von Worten über Σ, d.h. eine Teilmenge von Σ^*.

Die *Verkettung* $x \circ y$ zweier Worte x und y ist wieder ein Wort. Man schreibt auch einfach xy.

Definition 2 (deterministischer endlicher Automat (DFA)) Ein *deterministischer endlicher Automat* M besteht aus

- einem *Eingabealphabet* Σ,
- einer endlichen *Zustandsmenge* Q,
- einem *Startzustand* $q_0 \in Q$,
- einer Menge $Q_a \subseteq Q$ von *akzeptierenden Zuständen* und
- einer *Zustandsübergangsfunktion*

$$\delta : Q \times \Sigma \to Q.$$

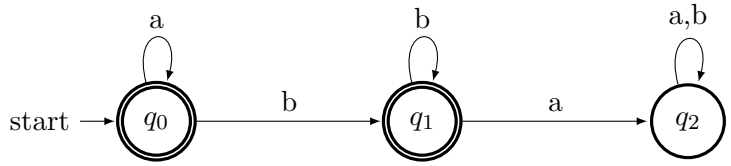
Der Automat beginnt im Anfangszustand $q := q_0$ und erhält ein Wort w als Eingabe. In jedem Schritt liest er das nächste Symbol σ von w und wechselt in den Zustand $q := \delta(q, \sigma)$. Wenn das Wort komplett gelesen ist, prüft er, ob q ein akzeptierender Zustand ist. Falls ja, wird w akzeptiert, sonst verworfen.

Definition 3 (erweiterte Zustandsübergangsfunktion) Wir erweitern die Zustandsübergangsfunktion δ zu einer Funktion $\delta^* : Q \times \Sigma^*$, die auf ganzen Worten operiert:

$$\delta^*(q, \lambda) = q$$
$$\delta^*(q, xw) = \delta^*(\delta(q, x), w).$$

Definition 4 (akzeptiertes Wort, akzeptierte Sprache) Der Automat M *akzeptiert* das Wort w, wenn $\delta^*(q_0, w) \in Q_a$ ist. Die Menge aller akzeptierten Worte heißt die *von M akzeptierte Sprache*, geschrieben $L(M) := \{w \in \Sigma^* \mid \delta^*(q_0, w) \in Q_a\}$.

Beispiel Graphdarstellung eines Automaten, der die Sprache $\{a^n b^m \mid n, m \geq 0\}$ akzeptiert:



Ein Kriterium dafür, was endliche Automaten *nicht* können, liefert das

Lemma 5 (Pumping-Lemma) Sei L eine von einem DFA akzeptierte Sprache. Dann existiert eine Wortlänge n so, dass für alle Wörter w der Länge mindestens n eine Zerlegung $w = x \circ y \circ z$ mit $|y| \geq 1$ und $|x \circ y| \leq n$ existiert, sodass für alle $i \in \mathbb{N}$ gilt

$$x \circ y^i \circ z \in L.$$

Bemerkung Dies ist ein *notwendiges* Kriterium einer Sprache für die Existenz eines Automaten, der diese Sprache akzeptiert. Es ist allerdings nicht hinreichend, d.h. selbst wenn die Aussage des Pumping-Lemma für eine Sprache gilt, muss diese Sprache noch nicht durch DFAs akzeptierbar sein. Wir können aber für eine bestimmte Sprache durch Finden von Gegenbeispielen zeigen, dass kein DFA existiert, der diese Sprache akzeptieren würde. Anders ausgedrückt:

Korollar 6 Sei L eine Sprache, für die für alle n ein Wort $w \in L$ der Länge $\geq n$ existiert, das für alle Zerlegungen $w = x \circ y \circ z$ mit $|y| \geq 1$ und $|x \circ y| \leq n$ ein Element der Form $x \circ y^i \circ z$ erzeugt, das nicht in L ist. Dann existiert kein DFA, der L akzeptiert.

Beispiel Sei $L := \{a^n b^n \mid n \in \mathbb{N}\}$ (beachte die identischen Potenzen zu a und b). Dann lässt sich L nicht durch einen DFA akzeptieren.

Beweis. Sei $1 \leq n \in \mathbb{N}$. Betrachte $w := a^n b^n$. In jeder Zerlegung von w wie oben hat y die Form a^p für ein $1 \leq p \in \mathbb{N}$ (beachte $|x \circ y| \leq n$, insbesondere $|y| \leq n$). Dann ist $\tilde{w} := x \circ y^2 \circ z = a^{n+p} b^n$ mit $n + p > n$, d.h. $\tilde{w} \notin L$. Wäre also L durch einen DFA akzeptierbar, so widerspräche dies dem Pumping-Lemma. □

Definition 7 (nichtdeterministischer endlicher Automat (NFA)) Ein *nichtdeterministischer endlicher Automat M* ist wie ein DFA aufgebaut, aber die Zustandsübergangsfunktion wird durch eine *Zustandsübergangsrelation* $\Delta \subseteq (Q \times \Sigma) \times Q$ ersetzt.

Ein NFA hat somit mehrere Möglichkeiten, in welchen Zustand er wechselt. Trotzdem sind NFAs nicht mächtiger als DFAs:

Konstruiere zu einem NFA $M = (\Sigma, Q, q_0, Q_a, \Delta)$ einen DFA, den *Potenzmengenautomat* $M' = (\Sigma, Q', q_0', Q_a', \delta')$:

- $Q' := \{P \mid P \subseteq Q\}$
- $q_0' := \{q_0\}$
- $Q_a' := \{P \in Q' \mid P \cap Q_a \neq \emptyset\}$
- $\delta'(P, x) := \{q \in Q \mid \exists p \in P : (p, x, q) \in \Delta\}$

Dann ist der nach dem Lesen eines Wortes w von M' angenommene Potenz-Zustand genau die Menge der Zustände, die M nach Lesen von w angenommen haben kann.

2 Reguläre Ausdrücke

- Gibt es noch andere einfache Charakterisierungen für Sprachen?
- Können wir vermeiden, für jede Sprache einen endlichen Automaten aufschreiben zu müssen?

Definition 8 ((reguläre) Grammatik) Eine *Grammatik* G ist ein Tupel (N, T, S, P) bestehend aus

- einem *Nonterminalalphabet* N,
- einem zu T disjunkten *Terminalalphabet* T,
- einem *Startsymbol* $S \in N$,
- und einer endlichen Menge P von *Ersetzungsregeln* der Form $l \to r$ mit $l, r \in (N \cup T)^*$. Dabei muss l mindestens ein Nonterminal enthalten.

Wir sagen G *erzeugt* eine Sprache L, wenn L genau die Worte aus T^* enthält, die mit Hilfe von Ersetzungsregeln in P aus S erzeugt werden. G heißt *regulär*, wenn alle Regeln ein einzelnes Nonterminal auf eine Folge von Terminalen mit höchstens einem Nonterminal am Ende abbilden. Wird L von einer regulären Grammatik erzeugt, so heißt L *regulär*.

Beispiel $(\{S, B\}, \{a, b\}, S, \{S \to aS, S \to B, B \to bB, B \to \lambda\})$ ist eine reguläre Grammatik, die die Sprache $\{a^n b^m \mid n, m \in \mathbb{N}\}$ erzeugt.

Satz 9 Jede von einem DFA akzeptierbare Sprache ist regulär.

Wie beschreiben wir reguläre Sprachen auf einfache Weise?

Definition 10 (reguläre Ausdrücke) Sei Σ ein Alphabet. Die Menge der *regulären Ausdrücke* über Σ ist eine rekursiv definierte Menge von Worten über dem Alphabet $\Sigma \cup \{(,), |, *\}$:

- λ ist ein regulärer Ausdruck.
- Für jedes $x \in \Sigma$ ist x ein regulärer Ausdruck.
- Sind R_1 und R_2 reguläre Ausrücke, so auch R_1R_2 und $(R_1 \mid R_2)$.
- Ist R ein regulärer Ausdruck, so auch $(R)*$.

Ein regulärer Ausdruck R beschreibt eine Sprache $L(R)$:

- $L(\lambda) = \{\lambda\}$.
- $L(x) = \{x\}$.
- $L(R_1R_2) = L(R_1) \circ L(R_2)$.
- $L((R_1 \mid R_2)) = L(R_1) \cup L(R_2)$.
- $L((R)*) = L(R)^*$, d.h. die Verkettung beliebig vieler Worte aus $L(R)$. Diesen Operator nennt man *Kleene-Stern*.

Die von einem regulären Ausdruck beschriebene Sprache ist regulär: $\{\lambda\}$ und $\{x\}$ sind offensichtlich regulär. Verkettung, Vereinigung und Kleene-Stern sind Operationen, die die Regularität erhalten.

Häufig werden in regulären Ausdrücken vereinfachte Schreibweisen zugelassen:

- `^` entspricht einem Zeilenanfang, `$` einem Zeilenende, `.` einem beliebigen Zeichen.
- `[abc]` entspricht `(a|b|c)`, `[^abc]` "jedem Zeichen außer `a`, `b` oder `c`".
- Statt `(0|1|...|9|.|_)` kann man auch `[0-9._]` schreiben.
- `R?` entspricht *höchstens einem*, `R+` *mindestens einem* Vorkommen von `R`.
- `R{a,b}` bedeutet, dass `R` mindestens `a` und höchstens `b` mal vorkommt.

Beispiel Regulärer Ausdruck zur Erkennung von Datumsangaben:

```
^(0?[1-9]|[12][0-9]|3[01])\.(0?[1-9]|1[012])\.[0-9]+$
```

Beispiel Regulärer Ausdruck für Email-Adressen:

```
^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$
```