Technische Universität München
Fakultät für Informatik
Lehrstuhl für Effiziente Algorithmen
Christian Ivicevic

Sommersemester 2015
Studentisches Tutorium
Übungsblatt 1
20. April 2015

Theoretische Informatik

– Sprachen, Grammatiken, Automaten 1 (DFA) –

Rückblick auf Diskrete Strukturen - Wie beweist man richtig?

In der Übung haben wir festgestellt, dass Sprachen nur Mengen an Wörtern sind, auf denen man bereits bekannte Operationen anwenden kann. Daher sind auch mengentheoretische Beweise problemlos möglich.

Beispiel 1. Seien $A, B, C \subset \Sigma^*$ gegebene Sprachen. Es gilt

$$A(B \cap C) \subset AB \cap AC.$$

Beweis. Wir wählen ein beliebiges Wort $w \in A(B \cap C)$ aus der linken Seite und beweisen, dass dieses ebenso auf der rechten Seite enthalten ist. Nun erkennen wir, dass dieses Wort zerlegt werden kann in zwei Teilwörter $w = uv$ mit $u \in A$ als *Präfix* und $v \in B \cap C$ als *Suffix*. Es ist zu erkennen, dass $uv \in AB$ und $uv \in AC$ gleichzeitig gelten muss, mithin gilt $uv \in AB \cap AC$ was zu beweisen war. □

Beispiel 2. Seien $A, B \subset \Sigma^*$ gegebene Sprachen. Es gilt für alle $n \in \mathbb{N}_0$

$$A \subset B \implies A^n \subset B^n. \tag{*}$$

Beweis. Wir beweisen die Behauptung mithilfe vollständiger Induktion über n.

- *Induktionsanfang*: Mit $n = 0$ gilt $A^0 = \{\varepsilon\} \subset \{\varepsilon\} = B^0$.
- *Induktionshypothese*: Die Gleichung (*) gilt für ein beliebiges, aber festes, $n \in \mathbb{N}_0$.
- *Induktionsschritt*: Für $n \to n + 1$ erhalten wir

  $$A^{n+1} = AA^n \overset{(1)}{\subset} BA^n \overset{(2)}{\subset} BB^n = B^{n+1}.$$

  In Schritt (1) haben wir die gegebene Voraussetzung $A \subset B$ aus der Aufgabe eingesetzt und daraufhin bei (2) die Induktionshypothese angewendet um den gesuchten Ausdruck zu erhalten.

□

Kontextfreie Sprachen (CFL) und Grammatiken (CFG)

Sprachen lassen sich natürlich extensional aufzählen, jedoch ist es oftmals nötig eine andere Darstellung zu finden, sofern man mit unendlichen Sprachen arbeitet. Hierzu kann man sogenannte Grammatiken nutzen, die sich aus Ableitungen zusammensetzen mit denen man Strukturen bilden kann, die man Stück für Stück in vollständige Wörter überführen kann.

Definition (Kontextfreie Grammatik)**.** *Eine* kontextfreie Grammatik *(engl. context-free grammar oder kurz CFG)* $G = (V, \Sigma, P, S)$ *ist ein Tupel bestehend aus*

- *der Menge der* Variablen *(Nichtterminalzeichen)* V*,*
- *einem endlichen* Alphabet *(Terminalzeichen)* Σ*,*
- *einer endlichen Menge an* Produktionen *(Ableitungsregeln)* P *der Form* $\ell \to r$ *mit* $\ell \in V$ *und* $r \in (V \cup \Sigma)^*$*,*
- *und einem* Startsymbol $S \in V$*.*

Die von G erzeugte kontextfreie Sprache *(engl. context-free language oder kurz CFL) ist*

$$L(G) = \{w \in \Sigma^* \mid S \to_G^* w\},$$

also die Menge aller Wörter (die keine Variablen mehr enthalten), die mit endlich vielen Ableitungen von S *aus erzeugt werden können.*

Beispiel 3. Wir betrachten die Grammatik

$$G_1 = (\{S, T\}, \{a, b\}, \{S \to aS \mid T,\ T \to b\}, S).$$

Bei dieser Grammatik lässt sich nun durch mehrmaliges Ableiten beispielweise

$$S \to aS \to aaS \to aaT \to aab$$

erzeugen. Wir schreiben an dieser Stelle formal $S \to_{G_1}^* aab$ und daher gilt $aab \in L(G_1)$.

Beispiel 4. Es soll eine Grammatik $G_2 = (V_2, \Sigma_2, P_2, S)$ konstruiert werden, für die gilt

$$L(G_2) = \{a^n \mid n \in \mathbb{N}\}.$$

Mit $V_2 = \{S\}, \Sigma_2 = \{a\}, P_2 = \{S \to a \mid aS\}$ ist eine gültige Grammatik definiert, die die gegebene Sprache erzeugt. Man achte insbesondere darauf, dass das leere Wort ε durch $n \neq 0$ nicht erzeugt werden kann in diesem Beispiel.

Beispiel 5. Es soll eine Grammatik $G_3 = (V_3, \Sigma_3, P_3, S)$ konstruiert werden, für die gilt

$$L(G_3) = \{a^n b^n \mid n \in \mathbb{N}_0\}.$$

Mit $V_3 = \{S\}, \Sigma_3 = \{a, b\}, P_3 = \{S \to \varepsilon \mid aSb\}$ haben wir eine Grammatik gefunden. Die Besonderheit ist, dass ein zugehöriger Automat[1] der prüft ob ein Wort in der vorgegeben Sprache enthalten ist, sich beim Einlesen merken kann[2], wieviele a's gelesen wurden um die passende Anzahl an b's zu erwarten.

[1] Im späteren Verlauf der Vorlesung nutzen wir Pushdown Automatas (PDA) zum erkennen von CFLs.

[2] Für die Klausur merken! Diese Sprache ist nicht regulär, weil einfache Automaten gedächtnislos sind.

Beispiel 6. Für die Sprache $\{a^n b^n c^n \mid n \in \mathbb{N}_0\}$ lässt sich keine kontextfreie Grammatik konstruieren[3], da der zugehörige Automat nur einen Zähler haben kann. Bei der vorliegenden Sprache müsste man aber an zwei Stellen mitzählen. Hierzu benötigt man das Konzept von kontextsensitiven Sprachen, das bisher nicht in der Vorlesung nicht detailliert angesprochen wurde.

Reguläre Sprachen und Automaten

Wir haben uns bisher kurze Beispiele zu kontextfreien Grammatiken und zugehörigen Sprachen angesehen und bemerkt, dass ein zugehöriger Automat, der solche Sprachen akzeptiert, Informationen damit speichern kann im Sinne eines Zählers. Vereinfachen wir das und arbeiten mit Automaten, die das nicht können, landen wir bei regulären Sprachen, die ein Untermenge der kontextfreien Sprachen sind[4]. Wir werden an dieser Stelle *deterministisch endliche Automaten* kennenlernen, die reguläre Sprachen akzeptieren - eine Sprache ist für uns zunächst regulär, wenn es einen solchen zugehörigen Automaten gibt. Um das Gegenteil zu beweisen, d.h. dass eine Sprache nicht regulär ist, nutzen wir im späteren Verlauf der Vorlesung das Pumping-Lemma. Vorerst widmen wir uns aber ausschließlich den Automaten zu regulären Sprachen und betrachten dazu sowohl die formale Definition als auch einige Beispiele.

Definition (Deterministisch endlicher Automat)**.** *Ein* deterministisch endlicher Automat *(engl. deterministic finite automaton oder kurz DFA)* $M = (Q, \Sigma, \delta, q_0, F)$ *ist ein Tupel bestehend aus*

- *einer endlichen* Zustandsmenge Q*,*
- *einem endlichen* Eingabealphabet Σ*,*
- *einer totalen und eindeutigen* Zustandsübergangsfunktion $\delta : Q \times \Sigma \to Q$*,*
- *einem* Startzustand $q_0 \in Q$*,*
- *und einer endlichen Menge an* Endzuständen $F \subset Q$*.*

Die von M akzeptierte (erkannte) Sprache *ist*

$$L(M) = \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \in F\}$$

wobei die Erweiterung[5] $\hat{\delta} : Q \times \Sigma^* \to Q$ *rekursiv definiert ist durch*

$$\begin{aligned} \hat{\delta}(q, \varepsilon) &= q, \\ \hat{\delta}(q, aw) &= \hat{\delta}(\delta(q, a), w) \qquad \forall q \in Q, a \in \Sigma, w \in \Sigma^*. \end{aligned}$$

[3] Für die Klausur merken! Das muss später mit dem Pumping-Lemma bewiesen werden können.

[4] Dies ist dem erweiterten Gedächtnis für die Klausur zu entnehmen.

[5] Diese Erweiterung entspricht der Funktion `foldl` in funktionalen Programmiersprachen um δ auf jedes Zeichen des Texts anzuwenden, der nichts anderes als `[char]` ist. Daher ist es nicht verkehrt eigene Simulationen in Haskell oder F# zu schreiben um evtl. bei den Hausaufgaben eigene Automaten testen zu können.

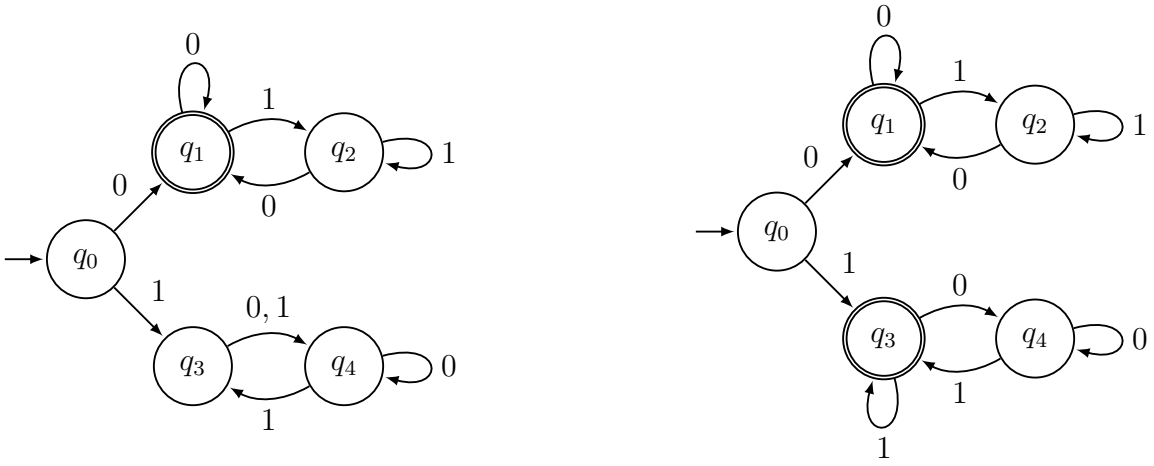

Abbildung 1: Automaten zu Beispiel 7 (links) und 8 (rechts)

Beispiel 7. Wir betrachten nun den Automaten $A = (Q, \Sigma, \delta, q_0, F)$ aus Abbildung 1 (links). Aus der Grafik können wir nun den Automaten in extensionaler Mengenschreibweise angeben:

$$\begin{aligned} Q &= \{q_0, \ldots, q_4\} \\ \Sigma &= \{0, 1\} \\ \delta &= \{((q_0, 0), q_1), ((q_0, 1), q_3), ((q_1, 0), q_1), \ldots\} \\ q_0 &= q_0 \\ F &= \{q_1\} \end{aligned}$$

Weiterhin können wir den Automaten per Hand simulieren, indem wir die δ-Funktion auswerten. So gilt unter Anderem

$$\begin{aligned} \delta(\delta(q_0, 0), 1) &= \delta(q_1, 1) = q_2, && \text{(Beispiel 1)} \\ \hat{\delta}(q_0, 10) &= \hat{\delta}(\delta(q_0, 1), 0) = \hat{\delta}(q_3, 0) && \text{(Beispiel 2)} \\ &= \hat{\delta}(\delta(q_3, 0), \varepsilon) = \hat{\delta}(q_4, \varepsilon) \\ &= q_4. \end{aligned}$$

An dieser Stelle ist noch zu erwähnen, dass es besonders wichtig ist die genauen Schritt beim Auswerten der $\hat{\delta}$-Funktion anzugeben und nicht abzukürzen sofern eine solche Aufgabe in der Klausur gestellt wird.

Beispiel 8. Wir betrachten nun das Alphabet $\Sigma = \{0, 1\}$ und die Sprache

$$\begin{aligned} L = \{w \in \Sigma^* \mid\ & w \text{ beginnt mit 0 und endet gleichzeitig mit 0 oder} \\ & \text{beginnt mit 1 und endet gleichzeitig mit 1}\}. \end{aligned}$$

Um einen Automaten B mit $L(B) = L$ zu konstruieren können wir uns bei Abbildung 1 (links) inspirieren lassen und erhalten den Automaten aus gleichen Abbildung (rechts).

Zusätzliche Bemerkungen

In der Vorlesung wurde ein Satz vorgestellt, dessen Beweis scheinbar etwas unverständlich war - diesen Satz werden wir hier aufgreifen und detaillierter beschreiben.

Satz. *Die Menge der Sprachen über einem nicht-leeren Alphabet ist überabzählbar.*

Beweis. Der Satz soll durch einen Widerspruch bewiesen werden. Hierzu behaupten wir, dass die Anzahl abzählbar wäre und wir ein Verfahren beschreiben mit dem wir absolut alle Sprachen aufzählen können und dies führen wir zu einem Widerspruch, so dass die Behauptung, dass es abzählbar viele Sprachen gäbe nicht stimmen kann. Wir werden eine (unendlich) große Tabelle erzeugen in der wir festhalten welche der Wörter $w_1, w_2, \ldots$ in einer der Sprachen $L_1, L_2, \ldots$ enthalten sind oder nicht. Dies kann beispielsweise wie folgt aussehen:

	w_1	w_2	w_3	$\cdots \quad = \Sigma^*$
L_1	0	1	1	$\cdots$
L_2	1	0	0	$\cdots$
L_3	1	1	1	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Wäre nun die betrachtete Menge abzählbar müsste in dieser Tabelle jede nur mögliche Sprache enthalten sein. Jedoch ist dies nicht der Fall, denn betrachtet man die Diagonale und bildet eine Sprache aus ihrem Komplement, dann hat man eine Sprache L gefunden, die definitiv noch nicht enthalten ist, da L sich in mindestens einem Wort w_i in Bezug zu jeder Sprache L_i unterscheidet. D.h. folgende Zeile ist sicher noch nicht in der Tabelle vorzufinden:

L	1	1	0	$\cdots$

Formal gilt für diese Sprache $L = \{w_i \mid w_i \notin L_i\}$ also auch $L \neq L_i$, da $w_i \in L \Leftrightarrow w_i \notin L_i$. Fügt man nun diese Sprache der Tabelle hinzu, hat man diese erweitert und kann dieses Diagonalargument erneut nutzen um eine weitere Sprache zu konstruieren, die nicht in der Tabelle enthalten war. Sukzessives Fortführen dieses Verfahrens liefert also Unstimmigkeiten. Dies ist ein Widerspruch dazu, dass wir bereits alle Sprachen in der Tabelle haben! □