Äquivalenzsatz von Kleene und Pumping-Argumente

Niklas Rieken

June 27, 2017

In diesem Dokument geht es um die Äquivalenz von regulären Ausdrücken und endlichen Automaten. Stephen C. Kleene zeigte, dass es zu jedem regulären Ausdruck einen DFA(!) gibt und umgekehrt. Der Beweis damals war aufgrund des fehlenden Konzept des *Nichtdeterminismus* ziemlich abgedreht (denkt mal drüber nach, wie man es anstellt aus einem Regexp sofort einen DFA zu bauen). Zum Schluss zeigen wir noch wie man beweisen kann, dass eine Sprache nicht regulär ist über ein sogenanntes Pumping-Argument. Es ist nicht genau das Pumping-Lemma, was in der Vorlesung meist recht ausführlich behandelt wird. Stattdessen beweisen wir etwas direkter, dass für eine nicht-reguläre Sprache kein endlicher Automat existieren kann. Das Dokument entstand aus einem Übungsblatt des Sommersemesters 2016.

Reguläre Ausdrücke

Reguläre Ausdrücke sind induktiv aufgebaut und beschreiben immer eine reguläre Sprache, welche mit den FA-erkennbaren Sprachen zusammenfallen (Satz von Kleene).

Reguläre Ausdrücke sind wie folgt definiert:
Basisfälle: $\emptyset, \varepsilon$ und a sind reguläre Ausdrücke für jedes $a \in \Sigma$.
Rekursionsschritt: Sind r und e reguläre Ausdrücke, so sind auch $(r{\cdot}e)$, $(r{+}e)$ und r^* reguläre Ausdrücke.
Semantisch stehen die Ausdrücke $\emptyset, \varepsilon, a$ für die Sprachen $L(\emptyset) := \emptyset, L(\varepsilon) := \{\varepsilon\}, L(a) := \{a\}$ und im rekursiven Fall $L(r \cdot e) := L(r) \cdot L(e), L(r + e) := L(r) \cup L(e), L(r^*) := L(r)^*$. Klammern und $\cdot$ werden gerne auch weggelassen, wenn der Ausdruck auch so klar ist.

Gelegentlich kennen Studenten reguläre Ausdrücke bereits durch Programme wie `grep, sed, vim`, womit sich Textdateien u.a. durchsuchen lassen mithilfe von POSIX-Ausdrücken. Diese sind gleichmächtig mit denen aus der Vorlesung, sollten aber nicht verwendet werden. Man möchte die Anzahl der Rekursionen auf das Minimum reduzieren, da dadurch auch die Beweise über reguläre Ausdrücke kürzer werden. Auf POSIX-Shortcuts wie $(r?)$ für $(r + \varepsilon)$, Umbenennungen wie $(r|e)$ für $(r + e)$ oder Schreibweisen

wie $[a-c]$ für $(a+b+c)$ solltet ihr also verzichten. Die einzige Ausnahme ist hier vielleicht r^+ für rr^*.

Aus den Tutoraufgaben hier zwei Beispiele nun:

a) $L_1 = \{w \in \{a,b,c\}^* \mid w \text{ hat gerade Länge }\}$.
$r_1 = ((a+b+c)(a+b+c))^*$. Das dieser Ausdruck die Sprache beschreibt lässt sich wie gewohnt über Induktion beweisen (zwei Richtungen für Gleichheit). Das ist aber hier nicht gefordert.

c) $L_3 = \{w \in \{a,b,c\}^* \mid |w| > 0$ und w beginnt und endet mit dem selben Symbol$\}$.
$r_3 = a(a+b+c)^*a + b(a+b+c)^*b + c(a+b+c)^*c + a + b + c$. Hier werden die Spezialfälle für $|w| = 1$ gerne übersehen.

Statt $(a+b+c)$ schreibt man gerne auch einfach Σ. Das ist aber eigentlich nicht ganz korrekt, da wir zwischen regulären Ausdrücken und den Sprachen die sie beschreiben unterscheiden. (r vs. $L(r)$). Streng genommen sind demnach auch die Zeichen $\emptyset, \varepsilon, a$, wie oben in der Definition der regulären Ausdrücke, verschieden von den der Sprache $\emptyset$, dem leeren Wort ε und dem Symbol $a \in \Sigma$. Wenn man also wie in Aufgabe a) z.B. $(\Sigma\Sigma)^*$ schreiben möchte, sollte man sich zumindest vorher Σ als regulären Ausdruck $\Sigma := (a+b+c)$ definieren.

Von Automaten zu regulären Ausdrücken (Floyd-Warshall-Algorithmus)

Das Verfahren was hier beschrieben wird ist leider ziemlich umständlich und technisch (im Vergleich zum Vorjahr), es ist aber das von Prof. Grohe präferierte Verfahren, deshalb müssen wir da jetzt durch. Die Idee ist eigentlich sogar recht einfach: Wir wollen vom Startzustand q_0 alle möglichen Pfade (d.h. über alle möglichen Zustände) zu akzeptierenden Zuständen finden (hier nur q_0). Allgemein suchen wir den regulären Ausdruck

$$r = \sum_{q \in F} r_Q(q_0, q) = r_Q(q_0, p_1) + \ldots + r_Q(q_0, p_k),$$

wobei $p_1, \ldots, p_k \in F$. Die Notation funktioniert folgendermaßen: Ein regulärer Ausdruck $r_P(p,q)$ mit $P \subseteq Q$ und $p, q \in Q$ beschreibt die Wörter mit denen man von p nach q kommt und dabei nur Zustände benutzt, die in P sind. Dies lässt sich rekursiv nun runterbrechen mit folgender Überlegung: Ein regulärer Ausdruck $r_\emptyset(p,q)$ ist $\sum_{(p,a,q)\in\Delta} a$, also alle Symbole $a \in \Sigma$ mit denen eine direkte Transition von p nach q möglich ist (oder noch anschaulicher: Alle Symbole die im Transitionsgraphen auf der Kante von p nach q stehen). Sollte keine Transition existieren, dann ist der reguläre Ausdruck entsprechend $\emptyset$ für $p \neq q$ und ε für $p = q$.
Der Rekursionsschritt funktioniert nun folgendermaßen: Für ein $P \neq \emptyset$

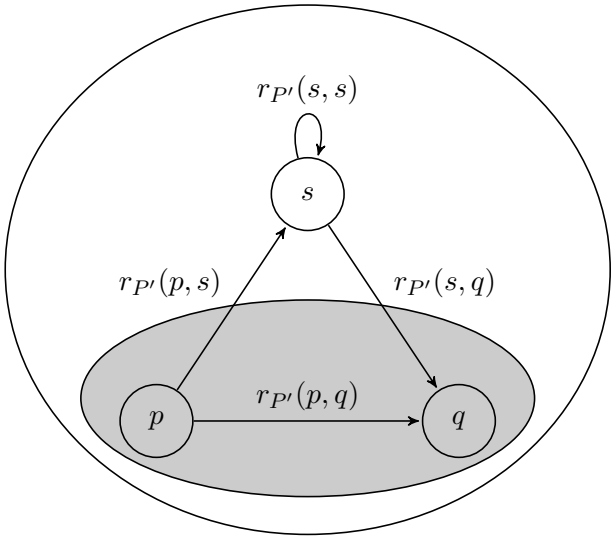


Figure 1: Rekursionsschritt um einen Zustand rauszuwerfen. Die grau getönte Fläche ist $P' := P \setminus \{s\}$. Die größere Fläche ist P. Wichtig: Die Kanten in diesem Graphen sind keine Transitionen, sondern sind mit einem regulären Ausdruck beschriftet, der die Sprache zwischen den Knoten beschreibt mit Zuständen aus dem Index!

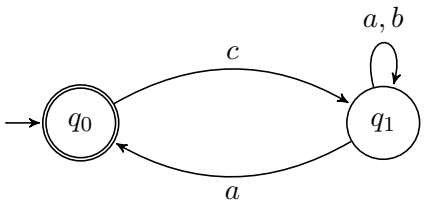
wählen wir einen Zustand $s \in P$ und entfernen diesen mit einer Umformung (WICHTIG!)

$$r_P(p,q) = r_{P'}(p,q) + r_{P'}(p,s) r_{P'}(s,s)^* r_{P'}(s,q),$$

wobei $P' := P \setminus \{s\}$. Auch dies einmal in einfache Worte gefasst: Um von p nach q zu kommen über Zustände ausschließlich aus P können wir entweder (links vom +) von p nach q gehen ohne einen Zustand s zu besuchen oder (rechts vom +) von p nach s gehen, Pfade von s nach s benutzen und schließlich von s nach q gehen, wobei auch hier nur Zustände in $P \setminus \{s\}$ benutzt werden.
Diesen rekursiven Schritt habe ich nochmal in Abbildung 1 veranschaulicht. Falls das Verfahren jemanden bekannt vorkommt, könnte derjenige damit richtig liegen. Es handelt sich bei dem Algorithmus um eine Version des *Floyd-Warshall-Algorithmus*, der in "Datenstrukturen und Algorithmen" in der Regel auch vorgestellt wird um günstigste Pfade in gewichteten Graphen zu finden.

Die Aufgabe selbst gehe ich jetzt nicht komplett durch, aber ein paar Rekursionsschritte sollten genügen um die idee zu verstehen. Wir suchen also einen regulären Ausdruck, der die Worte beschreibt mit denen man von q_0 nach q_0 kommt mit allen Zuständen, also den Ausdruck $r_Q(q_0, q_0)$. Wir wählen zunächst $s = q_1$ (eliminieren q_1):

$$r_Q(q_0,q_0) = r_{\{q_0\}}(q_0,q_0) + r_{\{q_0\}}(q_0,q_1) r_{\{q_0\}}(q_1,q_1)^* r_{\{q_0\}}(q_1,q_0)$$



Noch zu berechnen sind also $r_{\{q_0\}}(q_0, q_0), r_{\{q_0\}}(q_0, q_1), r_{\{q_0\}}(q_1, q_1), r_{\{q_0\}}(q_1, q_0)$. Wir berechnen $r_{\{q_0\}}(q_0, q_0)$ und eliminieren jetzt auch q_0 und können dann direkt den Basisfall einsetzen:

$$\begin{aligned} r_{\{q_0\}}(q_0, q_0) &= r_\emptyset(q_0, q_0) + r_\emptyset(q_0, q_0) r_\emptyset(q_0, q_0)^* r_\emptyset(q_0, q_0) \\ &\equiv \varepsilon + \varepsilon\varepsilon^*\varepsilon \\ &\equiv \varepsilon. \end{aligned}$$

Wir berechnen nun $r_{\{q_0\}}(q_0, q_1)$ und eliminieren wieder q_0:

$$\begin{aligned} r_{\{q_0\}}(q_0, q_1) &= r_\emptyset(q_0, q_1) + r_\emptyset(q_0, q_0) r_\emptyset(q_0, q_0)^* r_\emptyset(q_0, q_1) \\ &\equiv c + \varepsilon\varepsilon^* c \\ &\equiv c. \end{aligned}$$

Die letzten beiden lasse ich mal weg, damit die Aufgabe in späteren Jahrgängen noch verwendet werden kann. Die erhaltenen regulären Ausdrücke lassen sich aber später rekursiv wieder zusammensetzen und man erhält vereinfacht (falls ihr das noch selber machen wollt als Überprüfung):

$$r_Q(q_0, q_0) = \varepsilon + c(a + b + ac)^* a.$$

Von regulären Ausdrücken zu Automaten (Thompson-Konstruktion)

Die andere Richtung, von regulären Ausdrücken zu endlichen Automaten ist deutlich bequemer mit der *Thompson-Konstruktion*. Eine Variante davon wurde in der Vorlesung vorgestellt. Durch die rekursive Definition von regulären Ausdrücken lässt sich aus einem Ausdruck auch rekursiv ein ε-NFA aufbauen:
Basisfälle:

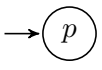


Figure 2: $r = \emptyset$

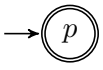


Figure 3: $r = \varepsilon$

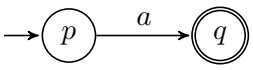


Figure 4: $r = a$

Rekursive Fälle: Seien $\mathcal{A}_e, \mathcal{A}_{e'}$ ε-NFAs für die regulären Ausdrücke e, e' mit Startzuständen q_0, q_0' und (verinfachte Darstellung mit nur einem Endzustand) akzeptierenden Zuständen q_f, q_f'. In den Abbildungen 5, 7 und 6

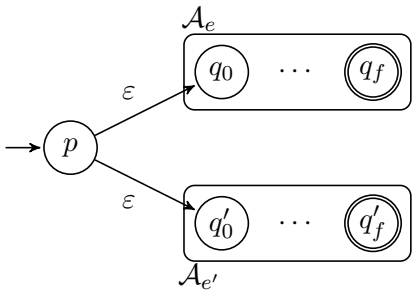


Figure 5: $r = e + e'$

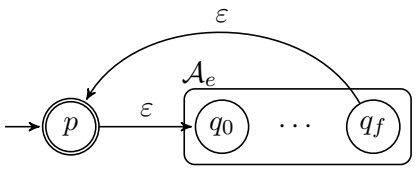


Figure 6: $r = e^*$

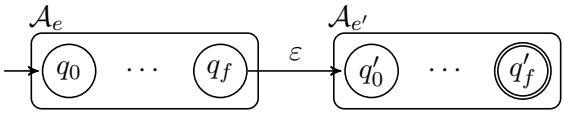


Figure 7: $r = e \cdot e'$

ist dargestellt wie die Komposition mit $+, \cdot und^*$ funktioniert. Es entstehen viele überflüssige ε-Transitionen, in der Hausaufgabe sollten diese aber besser nicht weggelassen werden. Die Konstruktion funktioniert immer und ist auch leichter zu implementieren als vielleicht effizientere Ansätze, die in FoSAP nicht behandelt werden.

Ein Minimalbeispiel:

$$r = (a + b)^* c.$$

Die Automaten für die Teilausdrücke a, b, c sind klar, wir starten mit $(a+b)$ in Abbildung 8. Um den Kleene'schen Abschluss zu bilden ergänzen wir

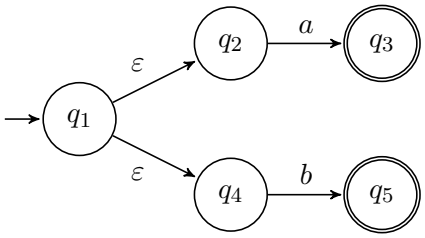


Figure 8: NFA für $(a + b)$

einen (eigentlich überflüssigen, aber nach Konstruktion erforderlichen) Zustand q_0^*, der neuer Startzustand wird und akzeptierend wird. Die ursprünglichen akzeptierenden Zustände werden nicht-akzeptierend und bekommen Transitionen zum neuen Startzustand. Zuletzt die Konkatenation von c: Dazu fügen wir eine ε-Transition vom (noch) akzeptierenden Zustand vom Automaten in Abbildung 9 zum Startzustand des trivialen Automaten für die Sprache $\{c\}$. Der fertige Automat ist in Abbildung 10. In den Hausauf-

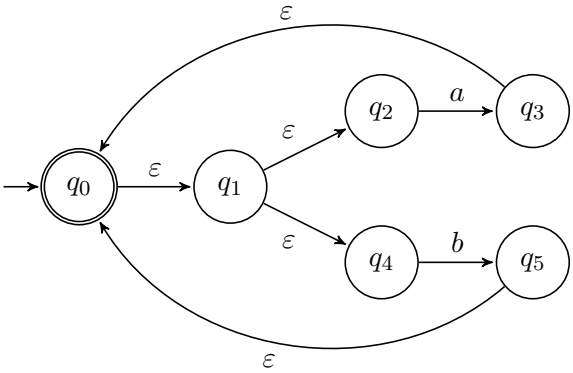


Figure 9: NFA für $(a + b)^*$

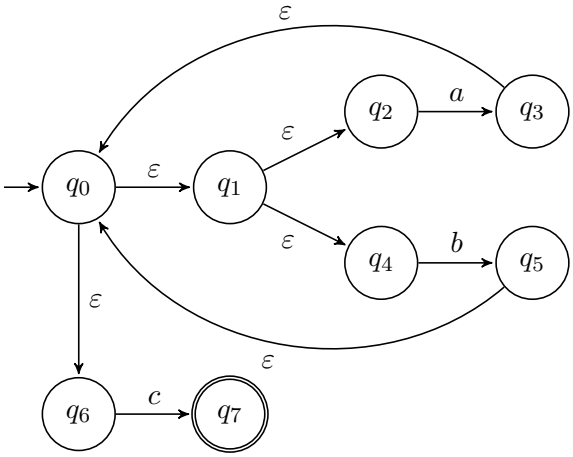


Figure 10: NFA für $(a + b)^*c$

gaben müsst ihr das nicht so ausführlich machen. Es reicht der Automat der Am Ende rauskommt. Ihr sollt aber das Verfahren aus der Vorlesung benutzen. Der Automat in Abbildung 11 ist zwar viel kleiner und äquivalent, aber nicht zulässig.

Nicht-reguläre Sprachen (Pumping-Argument)

Dies ist eine Aufgabe mit der Studenten in der Regel Probleme haben, da das Pumping-Lemma etwas sperrig ist und nicht sehr natürlich wirkt. Zur Erinnerung aber trotzdem nochmal:

Sei L eine reguläre Sprache. Dann existiert ein $n \in \mathbb{N}_+$, sodass für alle $w \in L$ mit $|w| \geq n$ eine Zerlegung $w = xyz$ existiert für die gilt:

(i) $|xy| \leq n$,

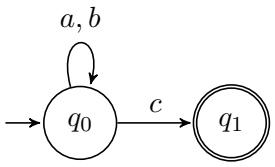


Figure 11: Ein minimaler NFA für die Sprache $L((a+b)^*c)$.

(ii) $y \neq \varepsilon$,

(iii) $xy^iz \in L$ für alle $i \in \mathbb{N}$.

Wir betrachten jetzt die Sprache

$$L = \{uav \,|\, u, v \in \{a, b\}^* \text{ mit } |u| = |v|\}$$

Der herkömmliche Weg, der auch in der Vorlesung vorgestellt wurde ist nun genau dieses Lemma benutzen und die Annahme, dass L regulär ist zu einem Widerspruch zu führen. Das ist aber genau das womit viele Schwierigkeiten haben, deswegen versuche ich das hier etwas anschaulicher:

Angenommen L ist regulär, dann gibt es einen endlichen Automaten $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$, der L erkennt. Angenommen $\mathcal{A}$ habe n Zustände. Wir betrachten das Wort $w = b^nab^n$. Das Wort ist in der Sprache und hat die Länge $2n + 1 > n$. Nach Lesen des Präfix b^n muss also spätestens eine Zustandswiederholung im akzeptierenden Lauf von $\mathcal{A}$ auf w aufgetreten sein (*pigeonhole principle, Schubfachprinzip*), d.h. vor dem Lesen vom a. Wir nehmen an, der Zustand, der sich wiederholt trat nach Lesen des i-ten und j-ten (oBdA $j > i$) b auf (s. Abbildung 12). Wir betrachten also den Lauf

$$r = (q_0, b, q_1, \ldots, b, q_i, b, \ldots, q_j, b, q_{j+1}, \ldots, b, q_n, a, q_{n+1}, b, \ldots, b, q_{2n+1}),$$

mit $q_{2n+1} \in F$ wobei $q_i = q_j$. Sei $j - i = k > 0$. Der Lauf

$$r = (q_0, b, \ldots, q_i, b, q_{j+1}, \ldots, b, q_{2n+1})$$

ist also ebenfalls möglich und auch akzeptierend mit dem Wort $w' = b^{n-k}ab^n$, aber $w' \notin L$, also erkennt $\mathcal{A}$ nicht die Sprache mit n Zuständen. Da n beliebig war gibt es also keinen endlichen Automaten, der L akzeptiert. Somit folgt, dass L nicht regulär ist.

Man sieht vielleicht, dass in dieser Lösung die Wörter und Zahlen aus dem Pumping Lemma wieder auftauchen, was natürlich kein Zufall ist (n, Das Präfix b^j entspricht xy, das Infix b^k mit $k > 0$ entspricht y und das Weglassen von b^k in w' entspricht der Betrachtung von xz in *iii)*).
Es wird in Zukunft noch weitere Aufgaben dieses Typs geben. Vor der Präsenzübung wird darauf auch im Tutorium nochmal eingegangen.

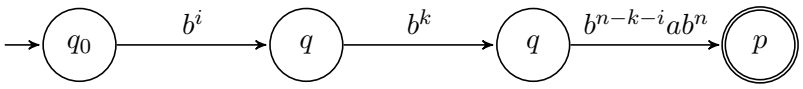


Figure 12: Zustandswiederholung in q. Es gibt also einen Lauf von q_0 nach q und einen Lauf von q nach p. Den Zwischenlauf von q nach q kann man also weglassen (oder auch beliebig oft wiederholen).