Äquivalenzsatz von Kleene

Niklas Rieken

28. Januar 2018

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 *Nicht-determinismus* ziemlich abgedreht (denkt mal drüber nach, wie man es anstellt aus einem Regex sofort einen DFA zu bauen).

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: $\varnothing, \varepsilon$ 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 $\varnothing, \varepsilon, a$ für die Sprachen $L(\varnothing) \coloneqq \varnothing, L(\varepsilon) \coloneqq \{\varepsilon\}, L(a) \coloneqq \{a\}$ und im rekursiven Fall $L(r \cdot e) \coloneqq L(r) \cdot L(e), L(r + e) \coloneqq L(r) \cup L(e), L(r^*) \coloneqq L(r)^*$.
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`, 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) $L_3 = \{w \in \{a, b, c\}^* \mid |w| > 0 \text{ und } w \text{ beginnt und endet mit dem selben Symbol}\}$.
$r_3 = a(a + b + c)^*a + b(a + b + c)^*b + c(a + b + c)^*c + a + b + c$. 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 $\varnothing, \varepsilon, a$, wie oben in der Definition der regulären Ausdrücke, verschieden von den der Sprache $\varnothing$, 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 $\Sigma \coloneqq (a + b + c)$ definieren.

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

Das Verfahren was hier beschrieben wird ist leider etwas umständlich aufgeschrieben, es gibt durchaus übersichtlichere Wege oder auch "graphische"Verfahren. Andererseits ist dieser Weg aber auch sehr genau und Fehler sind schneller nachzuvollziehen. 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 wie folgt: Rekursionsanfang:

$$r_\varnothing(p, q) = \begin{cases} \displaystyle\sum_{(p,a,q) \in \Delta} a, & p \neq q \\ \varepsilon + \displaystyle\sum_{(p,a,q) \in \Delta} a, & p = q. \end{cases}$$

Der Rekursionsschritt funktioniert nun folgendermaßen: Für ein $P \neq \varnothing$ wählen wir einen Zustand $s \in P$ und entfernen diesen mit folgender Umformung

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

wobei $P' \coloneqq P \setminus \{s\}$. 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.

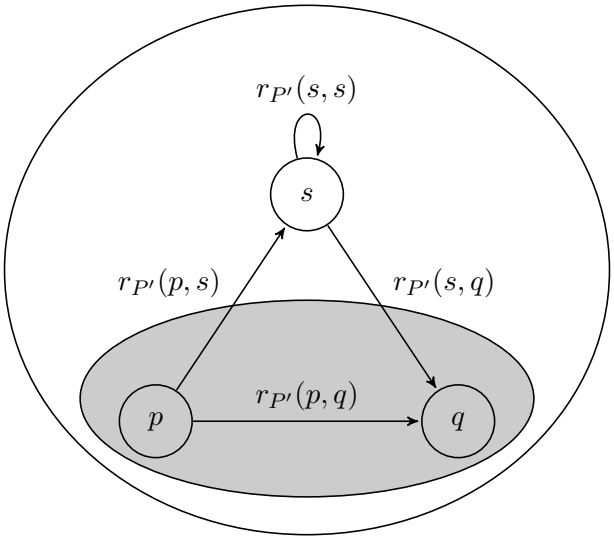
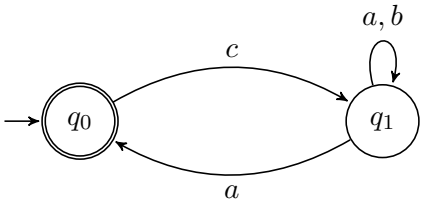


Abbildung 1: Rekursionsschritt um einen Zustand rauszuwerfen. Die grau getönte Fläche ist $P' \coloneqq P \setminus \{s\}$. Die größere Fläche ist P. Wichtig: Die Kanten in diesem Graphen sind keine Transitionen, sondern sind mit einem regulären Ausdruck beschriftet, der die Sprache zwischen den Knoten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



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:

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

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

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

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:

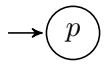


Abbildung 2: $r = \varnothing$

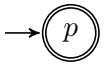


Abbildung 3: $r = \varepsilon$

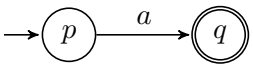


Abbildung 4: $r = a$

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 ist dargestellt wie die Komposition mit $+, \cdot und^*$ funktioniert. Es entstehen viele überflüssige ε-Transitionen, in der Hausaufgabe sollten

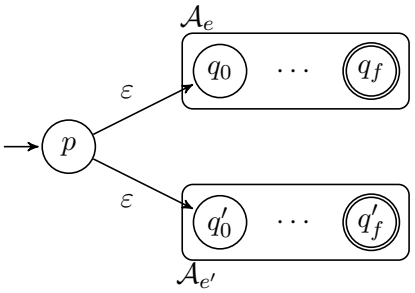


Abbildung 5: $r = e + e'$

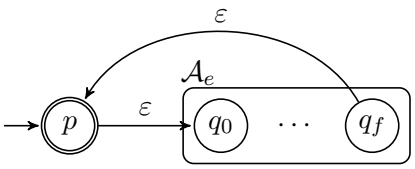


Abbildung 6: $r = e^*$

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.

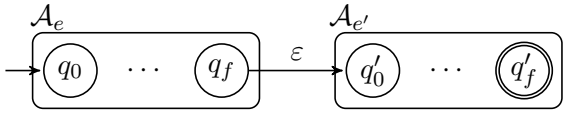


Abbildung 7: $r = e \cdot e'$

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 einen (eigentlich überflüssigen,

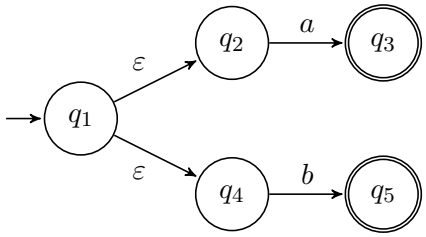


Abbildung 8: NFA für $(a + b)$

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:

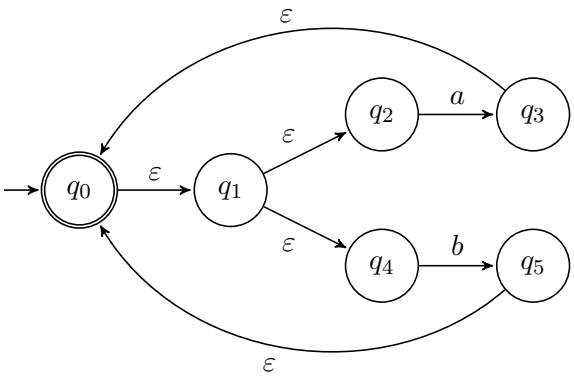


Abbildung 9: NFA für $(a + b)^*$

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 Hausaufgaben müsst ihr das nicht so ausführlich machen. Es reicht der Automat der Am Ende rauskommt. Ihr sollt aber das Verfahren aus der

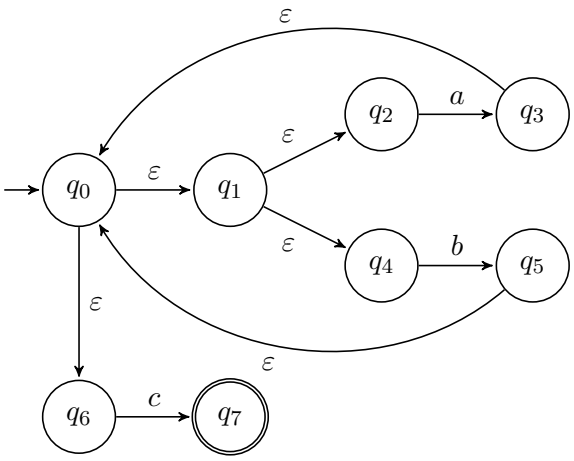


Abbildung 10: NFA für $(a + b)^*c$

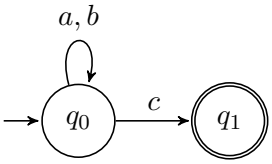


Abbildung 11: Ein minimaler NFA für die Sprache $L((a + b)^*c)$.

Vorlesung benutzen. Der Automat in Abbildung 11 ist zwar viel kleiner und äquivalent, aber nicht zulässig.