Äquivalenzsatz von Kleene
Rossmanith-Edition

Niklas Rieken

January 28, 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 *Nichtdeterminismus* ziemlich abgedreht (denkt mal drüber nach, wie man es anstellt aus einem Regex sofort einen DFA zu bauen). Dieses Dokument entstand auf Grundlage eines Übungsblattes aus dem Sommersemester 2016 und wurde für die Notation von Professor Rossmanith 2017 etwas überarbeitet.

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 $L(\varnothing) := \varnothing, L(\varepsilon) := \{\varepsilon\}, L(a) := \{a\}$ 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`, 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 eines alten Übungsblattes hier zwei Beispiele nun:

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

c) $L_3 = \{w \in \{a,b,c\}^+ \mid \text{erstes und letztes Symbol sind gleich}\}$.
$r_3 = a(a+b+c)^*a + b(a+b+c)^*b + c(a+b+c)^*c + a + b + c$. 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 $\Sigma := (a+b+c)$ definieren.

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

Das Verfahren was hier beschrieben wird ist etwas schreibaufwändig (im Vergleich zum State-Elimination-Algorithmus von Prof. Thomas), es ist aber das von Prof. Grohe präferierte Verfahren und wird in ähnlicher Form auch bei Prof. Rossmanith durchgeführt, deshalb müssen wir da jetzt durch. Die Idee ist eigentlich sogar recht einfach: Wir wollen vom Startzustand q_1 alle möglichen Pfade (d.h. über alle möglichen Zustände) zu akzeptierenden Zuständen finden (hier im Beispiel aus Abbildung 2 nur q_1). Allgemein suchen wir den regulären Ausdruck

$$r = \sum_{q_j \in F} r^n_{1j} = r^n_{1j_1} + \ldots + r^n_{1j_\ell}$$

wobei $q_{j_1}, \ldots, q_{j_\ell} \in F$. Die Notation funktioniert folgendermaßen: Ein regulärer Ausdruck r^k_{ij} mit $0 \leq k \leq n := |Q|$ und $q_i, q_j \in Q$ beschreibt die Wörter mit denen man von q_i nach q_j kommt und dabei nur die Zustände $q_1, \ldots, q_k$ benutzt. Dies lässt sich rekursiv nun runterbrechen durch folgende Basisfälle:

$$r^0_{ij} = \begin{cases} \displaystyle\sum_{a \in \Sigma : \delta(q_i,a) = q_j} a, & i \neq j \\ \varepsilon + \displaystyle\sum_{a \in \Sigma : \delta(q_i,a) = q_j} a, & i = j. \end{cases}$$

Anschaulich meint das: Alle Symbole die im Transitionsgraphen auf der Kante von q_i nach q_j stehen durch + verbunden. Sollte keine Transition existieren, dann ist der reguläre Ausdruck entsprechend $\varnothing$ für $q_i \neq q_j$ und ε für $q_i = q_j$.
Der Rekursionsschritt funktioniert nun folgendermaßen: Für ein $k > 0$ "eliminieren" wir den Zustand q_k durch folgende Rekursion:

$$r^k_{i,j} = r^{k-1}_{ij} + r^{k-1}_{ik} r^{k-1*}_{kk} r^{k-1}_{kj}.$$

Auch dies einmal in einfache Worte gefasst: Um von q_i nach q_j zu kommen, ausschließlich mit den Zuständen $q_1, \ldots, q_k$ können wir entweder (links vom +) von q_i nach q_j gehen

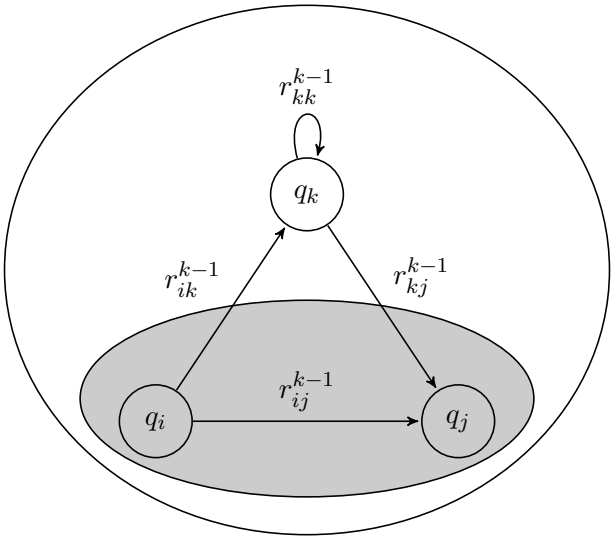

Figure 1: Rekursionsschritt um einen Zustand rauszuwerfen. Die grau getönte Fläche besteht nur aus den Zuständen $q_1, \ldots, q_{k-1}$. Die größere Fläche enthält zusätzlich den Zustand q_k, sonst keinen weiteren. Wichtig: Die Kanten in diesem Graphen sind keine Transitionen, sondern sind mit einem regulären Ausdruck beschriftet, der die Sprache zwischen den Knoten beschreibt mit Zuständen aus dem Index!

ohne den Zustand q_k zu besuchen oder (rechts vom +) von q_i nach q_k gehen, Pfade von q_k nach q_k benutzen und schließlich von q_k nach q_j gehen, wobei wir in jedem Zwischenschritt nur noch Transitionen über Zustände $q_1, \ldots, q_{k-1}$ benutzen.
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

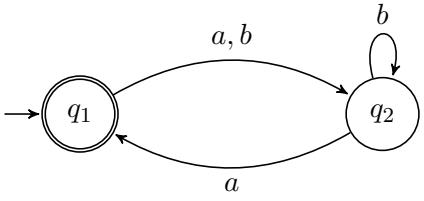

Figure 2: Ein DFA

die Worte beschreibt mit denen man von q_1 nach q_1 kommt mit allen Zuständen, also den

Ausdruck r_{11}^2. Wir eliminieren nun q_2

$$r_{11}^2 = r_{11}^1 + r_{12}^1 r_{22}^{1*} r_{21}^1.$$

Noch zu berechnen sind also $r_{11}^1, r_{12}^1, r_{22}^1, r_{21}^1$.

$$\begin{aligned}
r_{11}^1 &= r_{11}^0 + r_{11}^0 r_{11}^{0*} r_{11}^0 \\
r_{12}^1 &= r_{12}^0 + r_{11}^0 r_{11}^{0*} r_{12}^0 \\
r_{22}^1 &= r_{22}^0 + r_{21}^0 r_{11}^{0*} r_{12}^0 \\
r_{21}^1 &= r_{21}^0 + r_{21}^0 r_{11}^{0*} r_{11}^0.
\end{aligned}$$

Wir können nun für die Ausdrücke $r_{11}^0, r_{12}^0, r_{22}^0, r_{21}^0$ den Basisfall anwenden:

$$\begin{aligned}
r_{11}^0 &= \varepsilon + \varnothing \equiv \varepsilon \\
r_{12}^0 &= a + b \\
r_{22}^0 &= \varepsilon + b \\
r_{21}^0 &= a.
\end{aligned}$$

Dies können wir nun wieder rekursiv einsetzen, wir vereinfachen außerdem auch noch. In den Hausaufgaben ist dies nicht notwendig:

$$\begin{aligned}
r_{11}^1 &= \varepsilon + \varepsilon\varepsilon^*\varepsilon && \equiv \varepsilon \\
r_{12}^1 &= (a+b) + \varepsilon\varepsilon^*(a+b) && \equiv a+b \\
r_{22}^1 &= (\varepsilon + b) + (a\varepsilon^*(a+b)) && \equiv \varepsilon + b + a(a+b) \\
r_{21}^1 &= a + a\varepsilon^*\varepsilon && \equiv a.
\end{aligned}$$

Und weiter:

$$r_{11}^2 = \varepsilon + (a+b)(\varepsilon + b + a(a+b))^* a \equiv \varepsilon + (a+b)(b + a(a+b))^* a.$$

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: In Abbilungen 3, 4 und 5. Rekursive Fälle: Seien $\mathcal{A}_e, \mathcal{A}_{e'}$ ε-NFAs für die regulären Ausdrücke e, e' mit Startzuständen

Figure 3: $r = \varnothing$

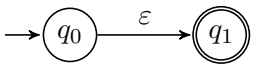

Figure 4: $r = \varepsilon$

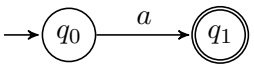

Figure 5: $r = a$

q_0, q_0' und akzeptierenden Zuständen q_f, q_f' (in dieser Variante haben wir ein paar nette

Invarianten: 1. Es gibt nur einen Endzustand egal wie groß oder verschachtelt der reguläre Ausdruck ist, 2. der Startzustand hat keine eingehende Transition, 3. Der Endzustand hat keine ausgehende Transition). In den Abbildungen 6, 8 und 7 ist dargestellt wie die Komposition mit $+, \cdot$ und * funktioniert.

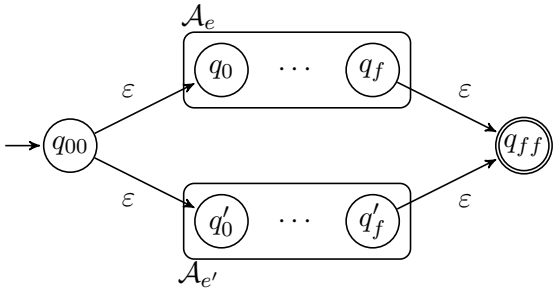

Figure 6: $r = e + e'$

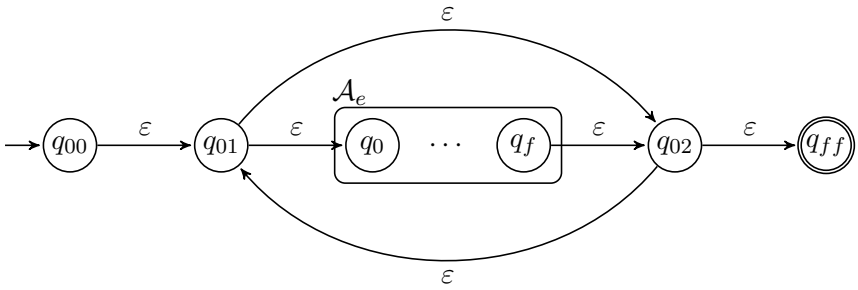

Figure 7: $r = e^*$

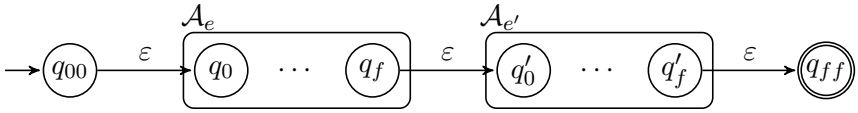

Figure 8: $r = e \cdot e'$

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 9. Nicht beschriftete Transitionen sind implizit ε-Transitionen.

Um den Kleene'schen Abschluss zu bilden ergänzen wir einige (eigentlich überflüssige,

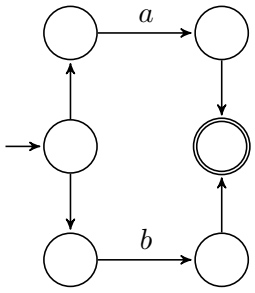

Figure 9: NFA für $(a + b)$

aber nach Konstruktion erforderliche) Zustände. Man sieht wie der Automat aus Abbildung 9 eingebettet ist in den neuen Automaten in Abbildung 10.

Zuletzt die Konkatenation von c: Dazu fügen wir eine ε-Transition vom (noch) akzep-

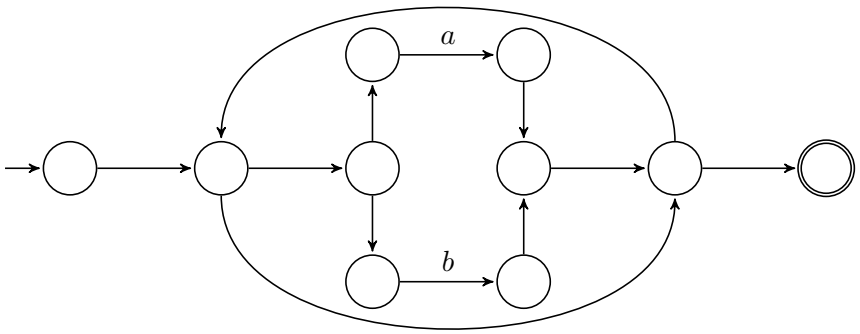

Figure 10: NFA für $(a + b)^*$

tierenden Zustand vom Automaten in Abbildung 10 zum Startzustand des trivialen Automaten für die Sprache $\{c\}$. Der fertige Automat ist in Abbildung 11.

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 Vorlesung benutzen. Der Automat in Abbildung 12 ist zwar viel kleiner und äquivalent, aber nicht zulässig.

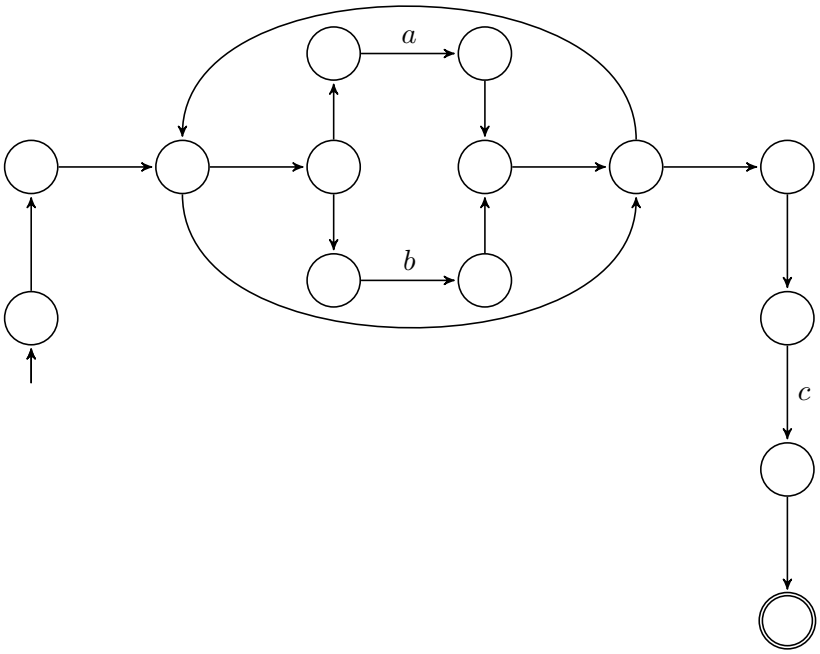

Figure 11: NFA für $(a + b)^* c$

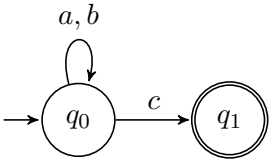

Figure 12: Ein minimaler NFA für die Sprache $L((a + b)^* c)$.