

HA-Lösung

TA-Lösung

Einführung in die theoretische Informatik – Aufgabenblatt 3

Beachten Sie: Soweit nicht explizit angegeben, sind Ergebnisse stets zu begründen!

Hausaufgaben: Abgabe bis zum **04.05.2016** (Mittwoch) um **12:00**

Einige der folgenden Aufgaben können auch online gelöst werden. Melden Sie sich auf <http://automatatutor.com> an und schreiben Sie sich in den Kurs **90THEO201** mit dem Passwort **3LUIYINJ** ein. Bitte geben Sie dennoch die Lösungen **handschriftlich** ab. Bitte beachten Sie, dass dies nur ein Zusatzangebot darstellt. Wenn eine Lösung als korrekt befunden wird, bedeutet dies **nur**, dass die richtige Sprache akzeptiert wird. Der Algorithmus kann dennoch falsch ausgeführt worden sein.

Aufgabe 3.1

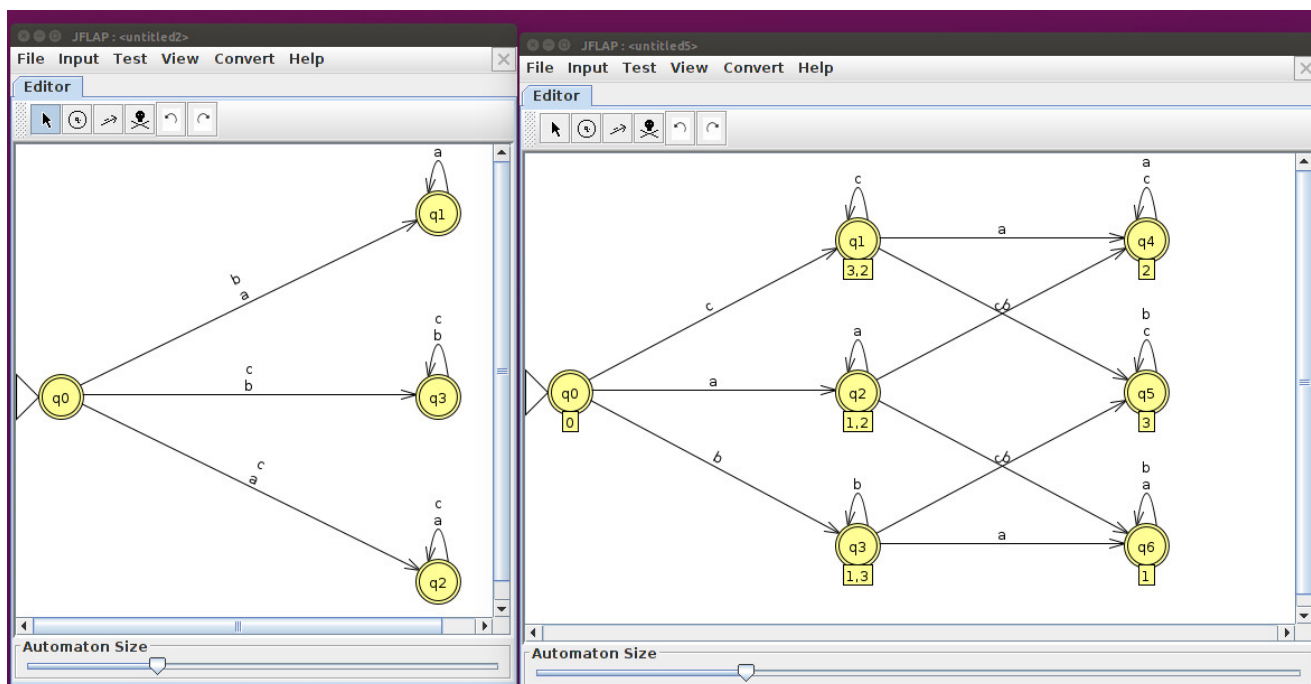
2P+2P

Mit $|w|_x$ wird die Anzahl der Vorkommen des Buchstabens $x \in \Sigma$ in $w \in \Sigma^*$ bezeichnet.

Sei $\Sigma = \{a, b, c\}$ und $L = \{w \in \Sigma^* \mid \exists x \in \Sigma: |w|_x = 0\}$.

- Konstruieren Sie einen NFA N mit genau 4 Zuständen und $L(N) = L$.
- Determinisieren Sie den NFA N aus (a) mittels der Potenzmengenkonstruktion aus den Folien (ab Folie 31).

Lösung:



Hinweis: Aus Gründen der Übersichtlichkeit ist der Fehlerzustand nicht eingezeichnet. Alle fehlenden Kanten führen zu diesem.

(a) Konstruieren Sie zunächst **mit dem Verfahren aus der Vorlesung** (ab Folie 50) einen ε -NFA N mit $L(N) = L((a|b)^*a)$. Halten Sie sich strikt an die Konstruktion aus den Folien. Als NFA für $x \in \Sigma$ verwenden Sie hierfür den (bis auf Knotenbenennung) kanonischen NFA $(\{q_0, q_1\}, \Sigma, \{(q_0, x, q_1)\}, q_0, \{q_1\})$.

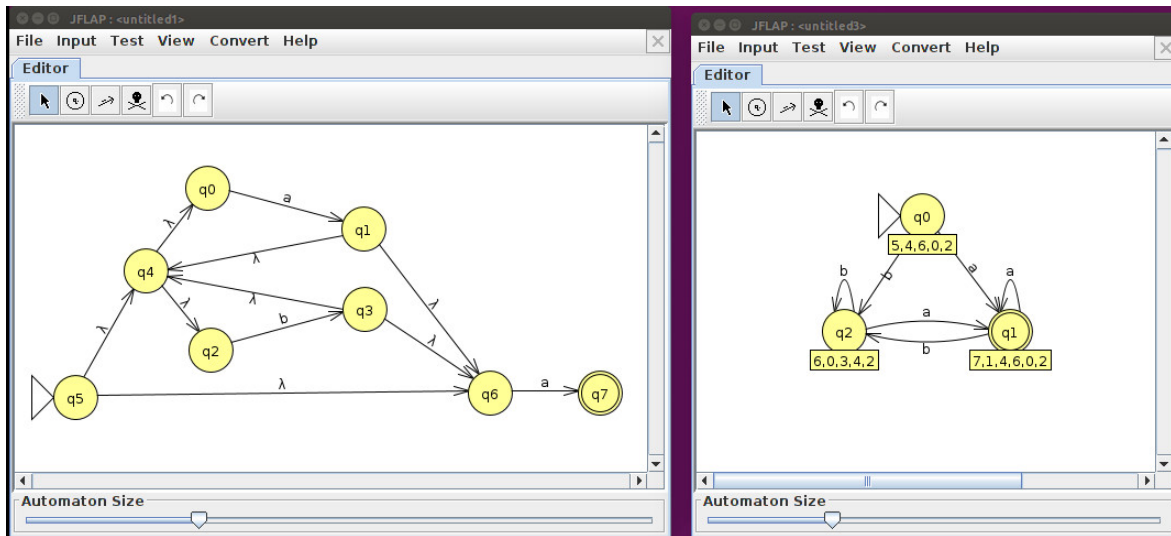
(b) Die Potenzmengenkonstruktion lässt sich auf ε -NFAs wie folgt erweitern:

Gegeben ein ε -NFA $N = (Q, \Sigma, \delta, q_0, F)$ sei $N' = (Q', \Sigma, \delta', S_0, F')$ wie folgt definiert:

- $S_0 := \bigcup_{i \geq 0} \delta(q_0, \varepsilon^i)$.
- Q' und $\delta': Q' \rightarrow Q'$ sind induktiv wie folgt definiert:
 - $S_0 \in Q'$.
 - Falls $S \in Q'$, dann $\delta'(S, a) := \bigcup_{q \in S, i \in \mathbb{N}_0, j \in \mathbb{N}_0} \delta(q, \varepsilon^i a \varepsilon^j)$ für jedes $a \in \Sigma$ und $\delta'(S, a) \in Q'$.
 - Ansonsten enthält Q' keine weiteren Elemente.
- $F' := \{S \in Q' \mid S \cap F \neq \emptyset\}$.

Wenden Sie **diese** Konstruktion an, um aus dem ε -NFA aus (a) einen DFA zu $L((a|b)^*a)$ zu konstruieren.

Lösung:



Aufgabe 3.3

Zeigen oder widerlegen Sie:

Für jeden NFA $N = (Q, \Sigma, \delta, q_0, F)$ (nur ein Startzustand), gibt es einen NFA $N' = (Q', \Sigma, \delta', q'_0, F')$ mit $L(N) = L(N')$ und

- (a) der Startzustand hat keine eingehenden Kanten.
- (b) kein Endzustand hat eine ausgehende Kante.
- (c) für jeden Zustand gilt: alle eingehenden Kanten sind mit demselben Zeichen beschriftet.
- (d) für jeden Zustand gilt: alle ausgehenden Kanten sind mit demselben Zeichen beschriftet.

Lösung:

(a) Ja: Man erstellt von jedem Startzustand q eine Kopie \hat{q} , die nur die ausgehenden Kanten von q erbt. Danach wird q zu einem normalen Zustand, während \hat{q} zu einem Startzustand wird. Formal:

$$N' = (Q \uplus \{\hat{q} \mid q \in I\}, \Sigma, \delta \uplus \{(\hat{q}, a, q') \mid (q, a, q') \in \delta\}, \hat{q}_0, F)$$

(b) Nein: Betrachte einen NFA N für $L = \{\epsilon, a\}$.

(c) Ja: man spaltet jeden Zustand $q \in Q$ nach dem Buchstaben der eingehenden Kanten auf, d.h. aus q macht man die Zustände (q, a) für jedes $a \in \Sigma$.

$$\text{Danach setzt man } \delta'((q, a), x, (q', x)) := \delta(q, x, q').$$

(d) Nein: Betrachte einen NFA N mit $L(N) = \{a, b\} = \Sigma$. Dann gilt $\delta(q_0, a) \cap F \neq \emptyset$ und $\delta(q_0, b) \cap F \neq \emptyset$.

Tutoraufgaben: Besprechung in KW18

Aufgabe 3.1 Pumping Lemma

Entscheiden Sie für jede der folgenden Sprachen, ob diese regulär sind.

- (a) $L_1 = \{w \in \{0,1\}^* \mid w = w^R\}$
- (b) $L_2 = \{w \in \{0,1\}^* \mid |w|_0 \geq |w|_1\}$
- (c) $L_3 = \{\varepsilon, a, a^{n-m} \mid m \in \mathbb{N}, n \in \mathbb{N}, m > 1, n > 1\}$

Lösung:

- (a) Angenommen, L_1 wäre regulär. Dann können wir das Pumping-Lemma anwenden.

Sei $n \in \mathbb{N}$ eine Pumping-Lemma-Zahl. Dann gilt $0^n 10^n \in L_1$. Es gibt also für $z = 0^n 10^n$ eine Zerlegung $z = uvw$ mit $v \neq \varepsilon$ und $|uv| \leq n$, so dass $(*) uv^i w \in L_1$ für alle $i \in \mathbb{N}_0$. Deshalb $uv = 0^k$ für $0 < k \leq n$. Also $u = 0^i$ und $v = 0^j$ mit $i + j = k$ und $j > 0$.

Dann ist aber $uv^2w = 0^i 0^{2j} 0^{n-k} 10^n \notin L_1$, denn $i + 2j + n - k > n$. Dies steht im Widerspruch zu $(*)$, d.h. unsere ursprüngliche Annahme, dass L_1 regulär ist, ist falsch.

- (b) Angenommen, L_2 wäre regulär. Dann können wir das Pumping-Lemma anwenden.

Sei also $n \in \mathbb{N}$ eine Pumping-Lemma-Zahl. Dann gilt $z = 0^n 1^n \in L_2$. Es gibt daher eine Zerlegung $z = uvw$ mit $v \neq \varepsilon$, $|uv| \leq n$ und $uv^i w \in L_2$ für alle $i \in \mathbb{N}_0$. Aus unserer Wahl von z folgt, dass $u = 0^i$ und v^j mit $j > 0$ gilt.

Allerdings ist $uv^0w = 0^i 0^{n-i-j} 1^n \notin L_2$, ein Widerspruch zum Pumping-Lemma. L_2 ist somit nicht regulär.

- (c) Angenommen, L_3 wäre regulär.

Da die regulären Sprachen unter Komplement abgeschlossen sind, ist auch $\overline{L_3} = \{a^n \mid n \in \mathbb{N}, n > 1, n \text{ ist prim}\}$ regulär. Dann können wir das Pumping-Lemma anwenden.

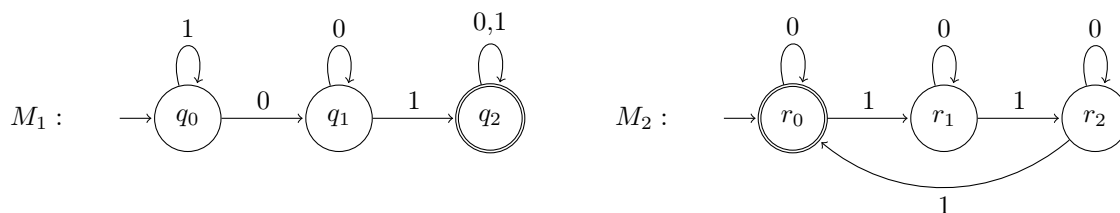
Sei n eine Pumping-Lemma-Zahl. Da die Menge der Primzahlen bekanntlich unendlich ist, gibt es eine Primzahl p bzw. ein $z = 1^p \in L$, so dass z mindestens n Zeichen enthält, also $|z| \geq n$. Sei also $z \in L$ mit $|z| \geq n$ und sei $z = uvw$ mit $|uv| \leq n$, v nicht leer und $uv^i w \in L$ für alle $i \geq 0$. Wegen $uw = uv^0w \in L$ ist $|uw|$ eine Primzahl. Wir setzen $x = uv^{|uw|}w$.

Es gilt $x \in L$, d. h. $|x|$ ist eine Primzahl. Es folgt aber auch $|x| = |uw| + |uv||v| = |uw|(1 + |v|)$, d. h. $|x|$ ist wegen $|v| \neq 0$ keine Primzahl. Widerspruch. L_3 ist somit nicht regulär.

Aufgabe 3.2 Produktkonstruktion

Gegeben seien zwei DFAs M_1 und M_2 über dem gleichen Eingabealphabet. In der Vorlesung haben Sie ein direktes Verfahren (ohne Umweg über reguläre Ausdrücke oder NFAs) gesehen, um einen DFA M mit $L(M) = L(M_1) \cap L(M_2)$ zu konstruieren.

- (a) Wenden Sie dieses Verfahren auf die beiden folgenden Automaten an:



- (b) Diese Produktkonstruktion lässt sich in Python wie folgt implementieren:

```
class DFA(object):
    def __init__(self, states, alphabet, transitions, initial_state, final_states):
        # ...

    def is_initial_state(self, state):
        return state == self.initial_state

    def is_final_state(self, state):
        return state in self.final_states

    def get_successor(self, state, letter):
        return self.transitions.get((state, letter), None)
    pass

def product_construction(dfa_left, dfa_right, boolean_op):
    alphabet = dfa_left.alphabet | dfa_right.alphabet
```

```

dfa_states = set()
dfa_final_states = set()
dfa_transitions = dict()
pair_to_dfa_state = dict()

def _create_new_state(P):
    new_dfa_state = State('%s,%s' % (P[0], P[1]))
    if boolean_op.is_final_state(dfa_left.is_final_state(P[0]), dfa_right.is_final_state(P[1])):
        dfa_final_states.add(new_dfa_state)
    dfa_states.add(new_dfa_state)
    pair_to_dfa_state[P] = new_dfa_state
    return new_dfa_state

P = (dfa_left.initial_state, dfa_right.initial_state)
todo = [P]

dfa_initial_state = _create_new_state(P)

while len(todo) > 0:
    P = todo.pop()
    cur_dfa_state = pair_to_dfa_state[P]
    for a in alphabet:
        Suc = (dfa_left.get_successor(P[0], a), dfa_right.get_successor(P[1], a))
        suc_dfa_state = pair_to_dfa_state.get(Suc)
        if suc_dfa_state is None:
            suc_dfa_state = _create_new_state(Suc)
            todo.append(Suc)
        dfa_transitions[(cur_dfa_state, a)] = suc_dfa_state
    pass
return DFA(dfa_states, alphabet, dfa_transitions, dfa_initial_state, dfa_final_states)

class BooleanOp(object):
    @staticmethod
    def is_final_state(left_final, right_final):
        return _____
    pass

```

Definieren Sie `is_final_state` so, dass obige Funktion gerade einen DFA zu $L(A) \cap L(B)$ konstruiert.

- (c) Definieren Sie `is_final_state` so, dass obige Funktion gerade einen DFA zu $L(A) \Delta L(B)$ konstruiert, wobei Δ die symmetrische Differenz zweier Mengen bezeichnet.
- (d) Überlegen Sie sich, warum man annehmen darf, dass ein DFA höchstens einen Zustand hat, von welchem aus kein Endzustand erreichbar ist (üblicherweise als ablehnender Zustand oder “trap state” oder “rejecting state” bezeichnet).

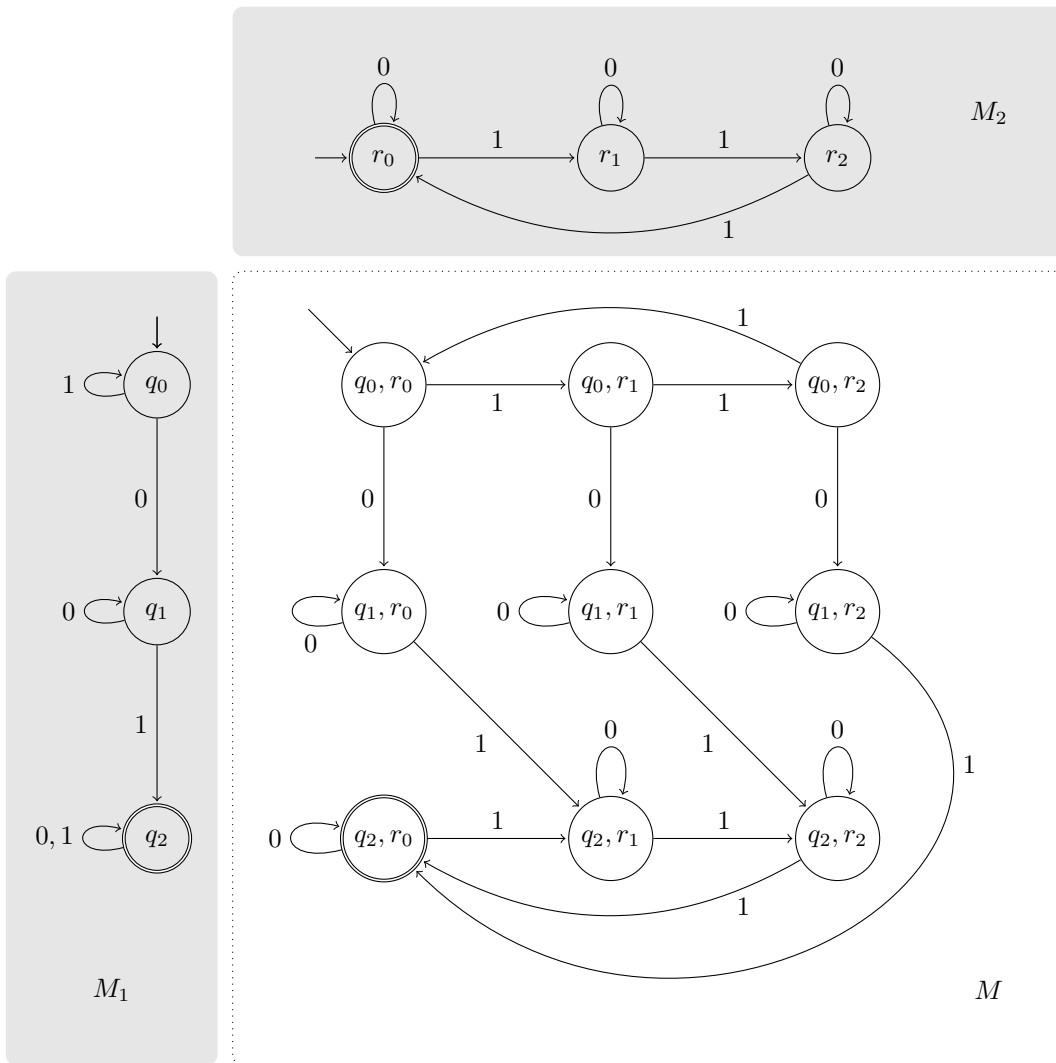
Wir erweitern die Klasse `DFA` entsprechend um die Methode `is_rejecting_state` inklusive der notwendigen Anpassungen an Attributen und dem Konstruktor. Entsprechend erweitern wir die Signatur der Funktion `product_construction` um das Argument `is_rejecting_in_product`.

ändern Sie die Funktion `product_construction` so ab, dass der erzeugte DFA höchstens einen ablehnenden Zustand hat und unnötige Zustände erst gar nicht konstruiert werden.

Definieren Sie dann `is_rejecting_in_product` sowohl für den Schnitt als auch die symmetrische Differenz.

Lösung:

- (a) Wir konstruieren den Vereinigungsautomaten aus M_1 und M_2 durch Ausnutzen der Tabellennotation:



(b) $(q, q') \in F_{M_1 \times M_2} \Leftrightarrow (q \in F_1 \wedge q' \in F_2)$. Code: Siehe unten.

(c) $(q, q') \in F_{M_1 \times M_2} \Leftrightarrow (q \in F_1 \oplus q' \in F_2)$. Code: Siehe unten.

(d) Siehe unten.

```
class IntersectionOp(object):
    @staticmethod
    def is_final_state(left_final, right_final):
        return left_final and right_final
    pass
```

(a) Schnitt

```
class SymDiffOp(object):
    @staticmethod
    def is_final_state(left_final, right_final):
        return left_final != right_final
    pass
```

(b) Symmetrische Differenz

```
def is_rejecting_in_product(left_rejecting,
                             right_rejecting):
    return left_rejecting or right_rejecting
```

(a) Schnitt

```
def is_rejecting_in_product(left_rejecting,
                             right_rejecting):
    return left_rejecting and right_rejecting
```

(b) Symmetrische Differenz

Aufgabe 3.3 Suffix Sprachen

Sei $L \subseteq \Sigma^*$ eine Sprache über dem Alphabet Σ . Mit $L_{sf} = \{v \in \Sigma^* \mid \exists u \in \Sigma^* : uv \in L\}$ sei die Sprache der Suffixe von L bezeichnet.

Beispiel: Für $L = \{abc, d\}$ ergibt sich $L_{sf} = \{\varepsilon, c, bc, abc, d\}$.

(a) Sei L regulär. Zeigen Sie, dass auch L_{sf} regulär ist, indem Sie aus einem NFA $N = (Q, \Sigma, \delta, q_0, F)$ mit $L = L(N)$ einen ε -NFA N' mit $L_{sf} = L(N')$ konstruieren.

- (b) Konstruieren Sie direkt, ohne den Umweg über NFAs einen regulären Ausdruck r mit $L(r) = L((ab | b)^* cd)_{\text{sf}}$.
- (c) Geben Sie ein allgemeines Verfahren an, dass einen gegebenen regulären Ausdruck r direkt in einen regulären Ausdruck r' mit $L(r') = L(r)_{\text{sf}}$ umschreibt.

Lösung:

- (a) **Konstruktion:** O.B.d.A. nehmen wir an, dass N keine von q_0 unerreichbaren Zustände hat. Sei $q'_0 \notin Q$ ein neuer Zustand, von dem man zu einer beliebigen Zustand in N springen kann und somit u überspringen kann. Wir definieren:

$$N' = (Q \cup \{q'_0\}, \Sigma, \delta \cup \{(q'_0, \epsilon, q) \mid q \in Q\}, q'_0, F)$$

Korrektheit:

- (i) $L_{\text{sf}} \subseteq L(N')$: Sei $v \in L_{\text{sf}}$. Dann existiert ein $u \in \Sigma^*$ mit $uv \in L$. Somit gibt einen akzeptierenden Lauf auf N :

$$q_0 \xrightarrow{u^{(1)}} q_1 \cdots \xrightarrow{u^{(|u|)}} q_{|u|} \xrightarrow{v^{(1)}} q_{|u|+1} \cdots \xrightarrow{v^{(|v|)}} q_{|uv|} \in F$$

Wir konstruieren nun aus dem akzeptierenden Lauf auf N für uv einen akzeptierenden Lauf auf N' für v , in dem aus dem Zustand q'_0 direkt zu $q_{|u|}$ springen:

$$q'_0 \xrightarrow{\epsilon} q_{|u|} \xrightarrow{v^{(1)}} q_{|u|+1} \cdots \xrightarrow{v^{(|v|)}} q_{|uv|} \in F$$

- (ii) $L_{\text{sf}} \supseteq L(N')$: Analog.

(b) $r = \epsilon | d | cd | (ab | b | \epsilon)(ab | b)^* cd$

- (c) **Konstruktion:** Wir definieren folgende rekursive Prozedur:

$$\begin{aligned} \text{suff}(\emptyset) &= \emptyset & \text{suff}(\alpha\beta) &= \text{suff}(\beta) | \text{suff}(\alpha)\beta \\ \text{suff}(\epsilon) &= \epsilon & \text{suff}(\alpha | \beta) &= \text{suff}(\alpha) | \text{suff}(\beta) \\ \text{suff}(a) &= a | \epsilon & \text{suff}(\alpha^*) &= \text{suff}(\alpha)\alpha^* \end{aligned}$$

Korrektheit: Wir zeigen $L(\text{suff}(\gamma)) = L(\gamma)_{\text{sf}}$ mit struktureller Induktion über γ :

- $\gamma = \emptyset$: $L(\text{suff}(\emptyset)) = \emptyset = \emptyset_{\text{sf}} = L(\emptyset)_{\text{sf}}$
- $\gamma = \epsilon$: $L(\text{suff}(\epsilon)) = \{\epsilon\} = \{\epsilon\}_{\text{sf}} = L(\epsilon)_{\text{sf}}$
- $\gamma = a$: $L(\text{suff}(a)) = \{\epsilon, a\} = \{a\}_{\text{sf}} = L(a)_{\text{sf}}$.
- $\gamma = \alpha\beta$: $L(\text{suff}(\alpha\beta)) = L(\text{suff}(\beta)) \cup L(\text{suff}(\alpha))L(\beta) \stackrel{\text{IH}}{=} L(\beta)_{\text{sf}} \cup L(\alpha)_{\text{sf}}L(\beta) = L(\alpha\beta)_{\text{sf}}$
- $\gamma = \alpha | \beta$: $L(\text{suff}(\alpha | \beta)) = L(\text{suff}(\alpha)) \cup L(\text{suff}(\beta)) \stackrel{\text{IH}}{=} L(\alpha)_{\text{sf}} \cup L(\beta)_{\text{sf}} = L(\alpha | \beta)_{\text{sf}}$
- $\gamma = \alpha^*$: $L(\text{suff}(\alpha^*)) = L(\text{suff}(\alpha))L(\alpha^*) \stackrel{\text{IH}}{=} L(\alpha)_{\text{sf}}L(\alpha^*) = L(\alpha^*)_{\text{sf}}$

Knobelaufgabe

Freiwillige Abgabe per E-Mail an theo-uebungsleitung@in.tum.de. Der Preis diese Woche für die ersten drei richtigen Gesamtlösungen ist jeweils eine Fritz Cola. Alle richtigen Einsendungen bis zum 11.05.2016 um 12:00 Uhr erhalten die Bonuspunkte. Wir bewerten von jedem Studenten nur die erste Einsendung und behalten uns das Recht vor unleserliche oder unverständliche Abgaben nicht zu korrigieren.

Aufgabe 3.1 Präfixfreie Sprachen

Bonus: 2P

Erinnerung: Eine Sprache $L \subseteq \Sigma^*$ ist *präfixfrei*, falls $\forall u \in L: u\Sigma^* \cap L = \{u\}$.

Sei $M = (Q, \Sigma, \delta, q_0, F)$ ein DFA und $L_q = L_q(M)$ die von M akzeptierte Sprache, wenn q statt q_0 als Startzustand gewählt wird. (Damit: $L(M) = L_{q_0}(M)$.)

(a) Zeigen Sie, dass folgende Behauptung falsch ist:

L ist präfixfrei **gdw.** $L_q(M) = \{\varepsilon\}$ für alle $q \in F$.

(b) Korrigieren Sie die rechte Seite der vermeintlichen Äquivalenz aus (a) so, dass eine korrekte Charakterisierung von DFAs gegeben ist, welche präfixfreie Sprachen akzeptieren.

Aufgabe 3.2 Strukturelle Eigenschaften von Automaten

Bonus: 2P

Wir erweitern NFAs und ε -NFAs um die Möglichkeit, mehrere Startzustände $I \subseteq Q$ zu besitzen. Ein NFA bzw. ε -NFA $N = (Q, \Sigma, \delta, I, F)$ akzeptiert ein Wort $w \in \Sigma^*$ genau dann, wenn $\emptyset \neq F \cap \bigcup_{q_0 \in I} \delta(q_0, w)$, d.h. wenn es mindestens einen akzeptierenden Ablauf von irgendeinem Startzustand aus gibt.

Zeigen oder widerlegen Sie:

(a) Für jeden NFA $N = (Q, \Sigma, \delta, I, F)$, gibt es einen ε -NFA $N' = (Q', \Sigma, \delta', I', F')$ mit $L(N) = L(N')$ und für jeden Zustand gilt: alle ausgehenden Kanten sind mit demselben Zeichen beschriftet.

(b) Für jeden NFA $N = (Q, \Sigma, \delta, I, F)$, gibt es einen NFA $N' = (Q', \Sigma, \delta', I', F')$ mit $L(N) = L(N')$ und für jeden Zustand gilt: alle ausgehenden Kanten sind mit demselben Zeichen beschriftet.