# Formale Systeme Automaten und Prozesse

Abgabe: 01.06.2017

Georg C. Dorndorf Matr. Nr. 366511
Adrian C. Hinrichs Matr. Nr. 367129
Jan Bordihn Matr. Nr. 364705

| # 7 | # 8 | # 9 | $\sum$ |
|---|---|---|---|
| | | | |

# Aufgabe 7

## Aufgabe 7.a

Siehe Abbildung 1.

## Aufgabe 7.b

**Notation:** Im folgenden werden die Zustände des zu Abbildung 1 gehörenden Automatens mit den konkatenierten Indizes aus dem Zustandsnamen benannt (Bsp.: 134 := $q_1, q_3, q_4$)

| | | | | | | |
|---|---|---|---|---|---|---|
| 12 | $\times_4$ | | | | | |
| 123 | $\times_2$ | (12,1) $\times_4$ | | | | |
| *1234* | $\times_0$ | $\times_0$ | $\times_0$ | | | |
| 13 | $\times_3$ | $\times_5$ | $\times_8$ | $\times_0$ | | |
| *134* | $\times_0$ | $\times_0$ | $\times_0$ | $\times_8$ | $\times_0$ | |
| *124* | $\times_0$ | $\times_0$ | $\times_0$ | (13,123) (134,1234) $\times_8$ | $\times_0$ | $\times_9$ |
| | 1 | 12 | 123 | *1234* | 13 | *134* |

1. $\delta(12, a) = 123$
   $\delta(1, a) = 12 \Rightarrow$ unterscheidbar, wenn $\{123, 12\}$ unterscheidbar
   $\delta(12, b) = 13$
   $\delta(1, b) = 13 \Rightarrow$ nicht unterscheidbar

2. $\delta(123, a) = 1234$
   $\delta(1, a) = 12 \Rightarrow$ unterscheidbar

3. $\delta(13, a) = 124$
   $\delta(1, a) = 12 \Rightarrow$ unterscheidbar

4. $\delta(123, a) = 1234$
   $\delta(12, a) = 122 \Rightarrow$ unterscheidbar

5. $\delta(13, a) = 124$
   $\delta(12, a) = 123 \Rightarrow$ unterscheidbar

6. $\delta(13, a) = 124$
   $\delta(123, a) = 1234 \Rightarrow$ unterscheidbar, wenn $\{124, 1234\}$ unterscheidbar
   $\delta(13, b) = 134$
   $\delta(123, b) = 134 \Rightarrow$ nicht unterscheidbar

7. $\delta(134, a) = 124$
   $\delta(1234, a) = 1234 \Rightarrow$ unterscheidbar, wenn $\{124, 1234\}$ unterscheidbar
   $\delta(134, b) = 1234$
   $\delta(1234, b) = 134 \Rightarrow$ nicht nützlich.

8. $\delta(124, a) = 123$
   $\delta(1234, a) = 1234 \Rightarrow$ unterscheidbar

9. $\delta(124, a) = 123$
   $\delta(134, a) = 124 \Rightarrow$ unterscheidbar

Der Automat war also schon Minimal.

*QEF*

## Aufgabe 7.c

Sei $L$ die durch den deterministischen endlichen Automaten aus b) erzeugte Sprache. Nach Vorlesung ist die Anzahl der Zustände des minimalen Deterministischen Automaten, der ein Sprache $S$ impliziert, gleich der Anzahl der Äquivalenzklassen der Sprache $S$. Für L gilt also mit b) und der Vorlesung, dass es 7 Äquivalenzklassen bezüglich $L$ gibt. Daraus folgt nach Vorlesung, dass es 7 Wörter $w_1, \ldots, w_7$ gibt, für die $w_i \not\equiv_L$ mit $i \neq j$ gilt.

Diese Wörter lassen sich bestimmen indem man die Eingaben, die benötigt werden um im minimalen DFA der Sprache $L$ zu einem jeden Knoten zu kommen, betrachtet. So erhalten wir $\varepsilon \not\equiv_L a \not\equiv_L aa \not\equiv_L aaa \not\equiv_L b \not\equiv_L bb \not\equiv_L ba$ als möglichst viele bezüglich $L$ verschiedene Wörter.

# Aufgabe 8

Mit dem folgenden Programm können wir ein NFA modular erzeugen. Wenn wir das Programm über dem Alphabet {a,b} mit den Zuständen {1, ... , 34} und den Transitionen aus der Datei `H8.trans` für $\hat{\delta}(7, ababbbaa)$ nutzen erhalten wir folgenden Ausgabe.

```
Das Wort ababbbaa kann in folgenden
Zustaenden (13 Stueck) enden:
1,2,3,5,6,8,9,10,11,17,22,
23,25.
```

Um das Programm nicht abtippen zu müssen ist es auch — bis auf eine Änderung — unter `http://sandbox.onlinephpfunctions.com/code/7460674535b15893313cf724691d10d4a0942d32` (Kurzlink: `http://bit.ly/2qy8QHn`) verfügbar. Dort kann es zum Testen auch ausgeführt werden.

Quelltext siehe Listing 1

# Aufgabe 9

**Gegeben:** Sei $L$ die Sprache über dem Alphabet $\Sigma = \{a, b, c, d, e\}$[1]

[1] Die Aufgabenstellung Arbeitet mit Emoticons, da dies allerdings nur eine Bennenungsfrage ist und ich die Elemente des

**Zu zeigen:** Mittels des Satzes von Myhill–Nerode gilt, dass die Sprache $L$ nicht Regulär ist.

**Beweis:** Sei $A \in L$. Es gilt:

$$\begin{aligned} A &= \epsilon \text{ oder} \\ \exists\, x \in \Sigma : A &= x \text{ oder} \\ \exists\, B, C \in L \backslash \{\epsilon\} : A &= CBC^R \text{ und} && \text{(I)} \\ \exists\, D \in L : A &= DD \text{ und} && \text{(II)} \\ A &= A^R && \text{(III)} \end{aligned}$$

Ergo gilt:

$$A(B^*) \in L \overset{I \wedge II}{\iff} B = A \qquad \text{(IV)}$$

Ein Element aus $L$ wird also genau dann wieder zu einem Element aus $L$, wenn man es mit einem vielfachen dieses Elementes konkateniert. Also: Seien $A, B \in L, w \in \Sigma^*$

$$\begin{aligned} & Aw \in L \wedge Bw \in L && \text{(V)} \\ \Leftrightarrow\ & \exists\, x \in \mathbb{N} : AA^x \in L \wedge BA^x \in L && \text{(VI)} \end{aligned}$$

Genau dann, wenn $\exists\, y \in \mathbb{N} : B = A^y$ gilt, gilt also $A \equiv_L B$. Da aber unendlich viele Kombinationen der Buchstaben des Alphabetes existieren, müssen auch unendlich vile Palindrome existieren, die kein Vielfaches von einander sind. Ergo ist der Index von $\equiv_L$ unendlich groß.
Mit dem Satz von Myhill–Nerode folgt nun, dass $L$ nicht regulär ist.

*QED*

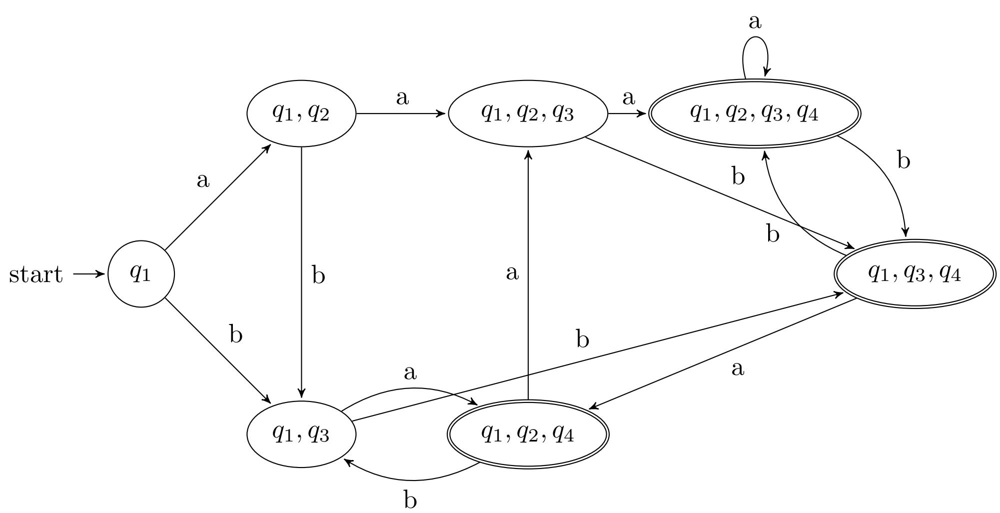


Abbildung 1: DFA zu Aufgabe 7

Alphabetes selten brauchen werde, sind sie hier lateinische Buchstaben.

Listing 1: Lösung zu Aufgabe 8 in PHP

```php
<?php
class Zustand {
    private $transitionen = array();
    private $zustandsId = 0;
    private $wortEndZustand = 0;

    public function __construct($zustandsId, $alphabet) {
        $this->zustandsId = $zustandsId;
        foreach ($alphabet as $key => $value) {
            $this->transitionen[$value] = array();
        }
    }

    public function addTransition($buchstabe, $endzustand) {
        if(!in_array($endzustand, $this->transitionen[$buchstabe])) {
            $this->transitionen[$buchstabe][] = $endzustand;
        }
    }

    private function getTransitionen($buchstaben) {
        if(isset($this->transitionen[$buchstaben])) {
            return $this->transitionen[$buchstaben];
        }
        return array();
    }

    public function simulate($wort) {
        $erreichbareZustaende = array();
        if($wort != "") {
            $naechsterBuchstabe = substr($wort,0,1);
            $restWort = substr($wort, 1);
            if($restWort == false) {
                $restWort = "";
            }
            $transitionen = $this->getTransitionen($naechsterBuchstabe);
            foreach ($transitionen as $key => $zustand) {
                if(is_a($zustand, 'Zustand')) {
                    $erreichbareZustaende = array_merge(
                       ↪ $erreichbareZustaende, $zustand->simulate(
                       ↪ $restWort));
                }
            }
        } else {
            if($wort == "" && $this->wortEndZustand == 0) {
                $erreichbareZustaende = array($this->zustandsId);
                $this->wortEndZustand = 1;
            }
        }
        return $erreichbareZustaende;
    }
}

class NFA {
    private $alphabet = array();
    private $zustaende = array();

    /*
     * Konstruktor der NFA Klasse.
     * Es bekommt die Zustaende & das Alphabet uebergeben.
```

```
     */
    public function __construct($zustaende, $alphabet) {
        $this->alphabet = $alphabet;
        foreach ($zustaende as $key => $value) {
            $this->zustaende[$value] = new Zustand($value, $alphabet);
        }
    }

    public function loadTransFile($dateiname) {
        $transdatei = fopen($dateiname, "r");
        if ($transdatei) {
            while (($zeile = fgets($transdatei)) !== false) {
                $zeile = explode('␣', $zeile);
                $this->addTransition($zeile[0],$zeile[1], trim($zeile[2]));
                   ↪ // trim entfernt die Zeilenumbrueche und evtl.
                   ↪ Leerzeichen
            }
        } else {
            echo "Die␣Transitionen␣Datei␣konnte␣nicht␣gefunden␣werden.";
        }
    }

    public function addTransition($startZustand, $buchstabe, $endzustand) {
        $this->zustaende[$startZustand]->addTransition($buchstabe, $this->
           ↪ zustaende[$endzustand]);
    }

    public function simulate($startZustand, $wort) {
        $erreichareZustaende = $this->zustaende[$startZustand]->simulate(
           ↪ $wort);
        sort($erreichareZustaende);
        echo "Das␣Wort␣$wort␣kann␣in␣folgenden␣Zustaenden␣(" . count(
           ↪ $erreichareZustaende) . "␣Stueck)␣enden:␣" . implode(',',
           ↪ $erreichareZustaende);
    }
}

$zustaende = array(1,2,3,4,5,6,7,8,9,10,11,12,13,14,
   ↪ 15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34);
$alphabet = array('a','b');
$nfa = new NFA($zustaende, $alphabet);
$nfa->loadTransFile("H8.trans");
$nfa->simulate(7,'ababbbaa');
```