

## Übung zur Vorlesung Formale Sprachen, Automaten und Prozesse

### Aufgabe T11

1. Betrachten Sie  $L = \{a^n b^n \mid n \geq 0\}$ . Gelten  $a \equiv_L aa$ ,  $b \equiv_L bb$  oder  $ab \equiv_L ba$ ?
2. Es sei nun  $L = (ab)^*$ . Gelten  $a \equiv_L b$ ,  $aa \equiv_L a$  oder  $\epsilon \equiv_L ab$ ? Wie lauten in diesem Fall die Äquivalenzklassen von  $\equiv_L$ ?

### Lösungsvorschlag:

1.
  - $a \not\equiv_L aa$ , da  $ab \in L$ , aber  $aab \notin L$ .
  - $b \equiv_L bb$ , da für alle  $u \in \Sigma^*$  gilt:  $bu \notin L$  gdw.  $bbu \notin L$ .
  - $ab \not\equiv_L ba$ , da  $ab \in L$  aber  $ba \notin L$ .
2.
  - $a \not\equiv_L b$ , da  $ab \in L$ , aber  $bb \notin L$ .
  - $aa \not\equiv_L a$ , da  $aab \notin L$ , aber  $ab \in L$ .
  - $\epsilon \equiv_L ab$ , da für alle  $u \in \Sigma^*$  gilt:  $u \in L$  gdw.  $abu \in L$ .

Die Äquivalenzklassen von  $\equiv_L$  sind:

- $[\epsilon]_{\equiv_L} = L$
- $[a]_{\equiv_L} = La$
- $[b]_{\equiv_L} = \Sigma^* \setminus (L \cup La) = (ab)^*(b + aa)(a + b)^*$

Zuerst zeigen wir, dass  $L \subseteq [\epsilon]_{\equiv_L}$ . Sei  $w \in L$ . Für alle  $w' \in \{a, b\}^*$  gilt:  $\epsilon w' = w' \in L$  gdw.  $ww' \in LL = L$ . Es folgt  $w \equiv_L \epsilon$ , und somit  $w \in [\epsilon]_{\equiv_L}$ . Wir zeigen außerdem  $[\epsilon]_{\equiv_L} \subseteq L$ : Sei  $w \in [\epsilon]_{\equiv_L}$ . Es gilt  $w \equiv_L \epsilon$ . Da  $\epsilon \in L$ , gilt auch  $w \in L$ . Aus beidseitiger Inklusion folgt  $[\epsilon]_{\equiv_L} = L$ .

Zeigen wir nun  $La \subseteq [a]_{\equiv_L}$ . Sei  $w \in La$ . Für alle  $w' \in \{a, b\}^*$  gilt:  $ww' \in L$  gdw.  $w' \in b(ab)^*$  gdw.  $aw' \in L$ . Es folgt  $w \equiv_L a$ . Wir zeigen außerdem  $[a]_{\equiv_L} \subseteq La$ : Sei  $w \in [a]_{\equiv_L}$ . Es gilt  $w \equiv_L a$ . Da  $a \in La$ , gilt auch  $w \in La$ . Es folgt  $[a]_{\equiv_L} = La$ .

Es bleibt zu zeigen, dass  $[b]_{\equiv_L} = \Sigma^* \setminus (L \cup La) = (ab)^*(b + aa)(a + b)^*$ . Die Äquivalenzklasse  $[b]_{\equiv_L}$  enthält genau die Wörter, die man nicht so ergänzen kann, dass sie in  $L$  sind. Also die Wörter, die zwei aufeinanderfolgende, gleiche Buchstaben enthalten. Diese Wörter sind durch  $(ab)^*(b + aa)(a + b)^*$  charakterisiert. Es folgt  $[b]_{\equiv_L} = \Sigma^* \setminus (L \cup La)$ .

Da  $[\epsilon]_{\equiv_L} \cup [a]_{\equiv_L} \cup [b]_{\equiv_L} = \Sigma^*$ , kann es keine weiteren Äquivalenzklassen geben.

### Aufgabe T12

Zeigen Sie mit dem Satz von Myhill–Nerode, daß folgende Sprachen nicht regulär sind:

1.  $L = \{ ww \mid w \in \{a, b\}^* \}$
2.  $L = \{ a^n b^m \mid |n - m| < 5 \}$

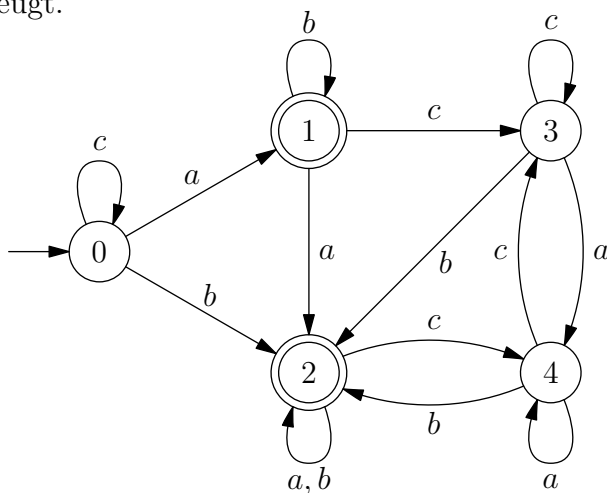
### Lösungsvorschlag:

Die hier verwendete Beweismethode benutzt jeweils eine unendliche Menge  $N$  und zeigt dann, daß alle Elemente von  $N$  in verschiedenen Äquivalenzklassen von  $\equiv_L$  liegen. Daraus resultiert dann, daß  $\equiv_L$  einen unendlichen Index hat, und  $L$  somit nach dem Satz von Myhill-Nerode nicht regulär sein kann.

1. Seien  $N = a^*b$  und  $u, v \in N$  mit  $u = a^i b$  und  $v = a^j b$  für irgendwelche  $i \neq j$ . Somit gilt  $uu \in L$ , aber  $vu = a^j b a^i b \notin L$ . Dies bedeutet, daß  $u$  und  $v$  in verschiedenen Äquivalenzklassen von  $\equiv_L$  liegen.
2. Seien  $N = a^*$  und  $u, v \in N$  mit  $u = a^i$  und  $v = a^j$  für irgendwelche  $i > j$ . Somit ist  $a^i b^{i+4} \in L$ , aber  $a^j b^{i+4} \notin L$ . Analog zu oben liegen  $a^i$  und  $a^j$  somit in verschiedenen Äquivalenzklassen.

### Aufgabe T13

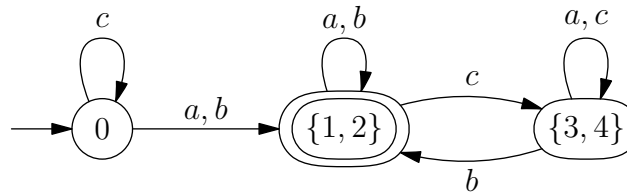
Minimieren Sie folgenden DFA und finden Sie anschließend einen regulären Ausdruck, der seine Sprache erzeugt.



### Lösungsvorschlag:

Der Markierungsalgorithmus zur Minimierung des Automaten ergibt folgende Tabelle.

	0	1	2	3	4
0		X	X	X	X
1	X			X	X
2	X			X	X
3	X	X	X		
4	X	X	X		



Wir können nun den regulären Ausdruck ablesen. Wir erhalten:

$$c^*(a + b)((a + b) + c(a + c)^*b)^*$$

### Aufgabe T14

Richtig oder falsch?

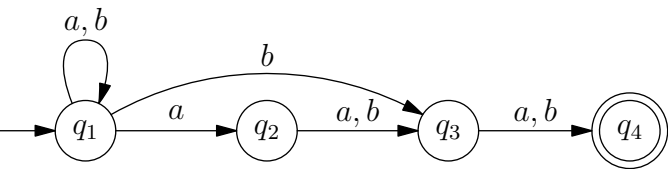
- Es gibt eine reguläre Sprache, die von einem NFA mit 10 Zuständen akzeptiert wird, aber von keinem DFA mit 100 Zuständen.
- Es gibt eine reguläre Sprache, die von einem NFA mit 10 Zuständen akzeptiert wird, aber von keinem DFA mit 1217 Zuständen.
- Zwei minimale DFAs, welche jeweils komplementäre Sprachen akzeptieren, haben gleich viele Zustände.
- Zwei minimale NFAs, welche jeweils komplementäre Sprachen akzeptieren, haben gleich viele Zustände.

### Lösungsvorschlag:

- Richtig: Wie in der Vorlesung gezeigt wurde, gibt es einen NFA mit  $n+2$  Zuständen, der die Sprache  $(a+b)^*a(a+b)^n$  akzeptiert, gleichzeitig aber jeder DFA wenigstens  $2^n$  Zustände benötigt.
- Falsch: Wenn ein NFA mit 10 Zuständen eine Sprache akzeptiert, dann hat der zugehörige Potenzmengenautomat höchstens  $2^{10} = 1024 < 1217$  Zustände.
- Richtig: Wenn  $M$  ein minimaler Automat für  $L = L(M)$  ist, dann erhält man einen Automaten  $\overline{M}$  mit  $\overline{L} = L(\overline{M})$ , indem man Nichtendzustände und Endzustände vertauscht (siehe Vorlesung), wobei die Größe des Automaten sich nicht ändert. Wenn  $\overline{M}$  nun nicht minimal wäre, dann existiert  $\overline{M}'$  mit  $\overline{L} = L(\overline{M}) = L(\overline{M}')$  und  $|\overline{M}'| < |\overline{M}|$ . Der Komplementäutomat von  $\overline{M}'$  wiederum, nennen wir ihn  $M'$ , hat nun weniger Zustände als  $M$ , erkennt aber dennoch dieselbe Sprache. Ein Widerspruch zur Minimalität von  $M$ .
- Falsch: Betrachte  $L = \{\epsilon\}$  und  $\overline{L} = \Sigma^* - \{\epsilon\}$ . Dann enthält  $\overline{L}$  alle Wörter, die wenigstens ein Zeichen enthalten.  $L$  kann man mit einem NFA erkennen, der nur einen einzigen Zustand, einen Endzustand, aber keine Transitionen enthält. Für  $\overline{L}$  benötigt man hingegen auf jeden Fall mindestens zwei Zustände, da  $\epsilon$  nicht akzeptiert werden darf.

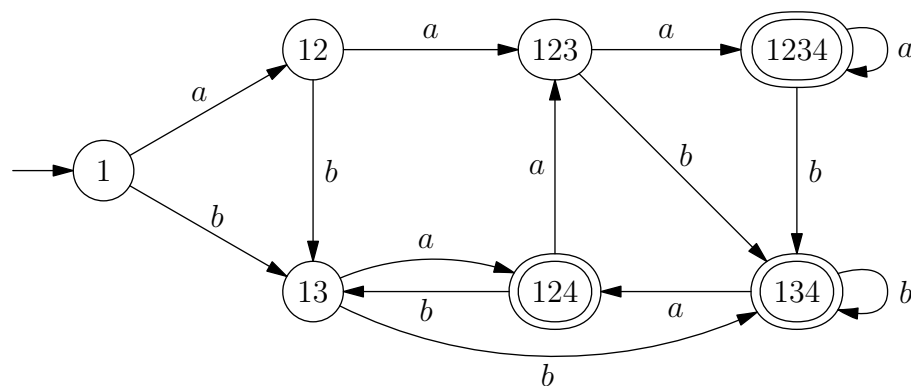
### Aufgabe H7 (5+5+5 Punkte)

- a) Bilden Sie den Potenzautomaten von nebenstehendem NFA  $M$ .
- b) Minimieren Sie ihn mit dem Markierungsalgorithmus.
- c) Geben Sie möglichst viele Wörter  $w_1, \dots, w_n$  an, so daß  $w_i \not\equiv_L w_j$  für  $i \neq j$ . Wie haben Sie die Wörter gefunden und warum erfüllen Sie diese Bedingung?



### Lösungsvorschlag:

- a) Mit der Potenzmengenkonstruktion erhält man folgenden DFA.



- b) Der Übersicht halber definieren wir uns  $A = 1, B = 12, C = 13, D = 123, E = 124, F = 1234, G = 134$ . Endzustände sind dann  $\{E, F, G\}$ .

	A	B	C	D	E	F	G
A		X	X	X	X	X	X
B	X		X	X	X	X	X
C	X	X		X	X	X	X
D	X	X	X		X	X	X
E	X	X	X	X		X	X
F	X	X	X	X	X		X
G	X	X	X	X	X	X	

Da alle Zustände unterscheidbar sind, ist der DFA bereits minimal.

- c) Da die Zustände alle unterscheidbar sind braucht man nur sieben Wörter zu finden, die vom Startzustand jeweils in einen der sieben Zustände führen. Zum Beispiel  $\{\epsilon, a, b, aa, aaa, ba, bb\}$ . Da  $\sim$  eine Verfeinerung von  $\equiv_L$  ist, kann es keine größere Menge geben.

## Aufgabe H8 (15 Punkte)

Programmieren Sie einfache NFAs in einer objektorientierten, gängigen Programmiersprache, am besten in Java. Erstellen Sie hierzu eine generische Klasse  $NFA\langle S, A \rangle$ , welche einen geeigneten Konstruktor und folgende Methoden besitzt:

1.  $addTransition(S\ q, A\ a, S\ p)$  fügt eine Transition vom Zustand  $q$  zum Zustand  $p$  mit dem Symbol  $a$  hinzu.
2.  $Set\langle S \rangle\ simulate(S\ q, List\langle A \rangle\ w)$  simuliert den NFA vom Zustand  $q$  beginnend und das Wort  $w$  lesend. Sie berechnet dabei  $\hat{\delta}(q, w)$ .

Die Parameter  $A$  und  $S$  stehen hierbei für den Typ des Alphabets und der Zustände. Natürlich sollte Ihr Programm gut lesbar und einfach zu verstehen sein, so daß es auch von anderen Personen gewartet und angepaßt werden könnte.

Folgendes Programmstück würde einen NFA konstruieren und dann simulieren:

```
import java.util.*;
public class MyNFA {
    static public void main(String args[]) {
        Set<Integer>states = new HashSet<Integer>();
        states.add(1); states.add(2); states.add(3); states.add(4);
        NFA<Integer, String>M = new NFA<Integer, String>(states);
        M.addTransition(1, "rechts", 2); M.addTransition(1, "runter", 3);
        M.addTransition(2, "links", 1); M.addTransition(2, "runter", 4);
        M.addTransition(3, "rechts", 4); M.addTransition(3, "hoch", 1);
        M.addTransition(4, "links", 3); M.addTransition(4, "hoch", 2);
        List<String>eingabe = new LinkedList<String>();
        eingabe.add("rechts"); eingabe.add("runter"); eingabe.add("links");
        Set<Integer>reachable = M.simulate(1, eingabe);
        System.out.println(reachable);
    }
}
```

Verwenden Sie Ihr Programm, um  $\hat{\delta}(7, ababbbaa)$  für den NFA zu berechnen, dessen Transitionen Sie auf der Webseite neben dem Aufgabenblatt finden. Er hat die Zustandsmenge  $\{1, \dots, 34\}$  und das Eingabealphabet  $\{a, b\}$ . Was ist das Ergebnis?

Ihr Programm sollte halbwegs effizient sein, insbesondere im Spezialfall eines deterministischen Automaten. Achten Sie darauf, den Code möglichst modular zu gestalten.

## Lösungsvorschlag:

Eine einfache Implementierung der Klasse  $NFA\langle S, A \rangle$  findet sich in Abbildung 1

Ein Programm, welches die Transitionen eines Automaten mit Zustandsmenge  $\{1, \dots, 34\}$  einliest und auf dem Wort  $ababbbaa$  simuliert, ist in Abbildung 2 enthalten. Lassen wir das Programm laufen, erhalten wir die Ausgabe  $[1, 2, 3, 5, 6, 8, 9, 10, 11, 17, 22, 23, 25]$ .

## Aufgabe H9 (5 Punkte)

Sei  $L$  die Sprache aller Palindrome über dem Alphabet  $\{\text{😊}, \text{😍}, \text{🐼}, \text{🐶}, \text{🐱}\}$ . Zeigen Sie mithilfe des Satzes von Myhill–Nerode, daß diese Sprache nicht regulär ist.

```

import java.util.*;

public class NFA<S, A>{
    Set<S> Q;
    Map<S, HashMap<A, HashSet<S>>> > delta;

    public NFA(Set<S> Q) {
        this.Q = Q;
        this.delta = new HashMap<S, HashMap<A, HashSet<S>>> > ();
        for(S q : Q) {
            delta.put(q, new HashMap<A, HashSet<S>>());
        }
    }

    public Set<S> simulateOneStep(Set<S> qset, A a) {
        Set<S> H = new HashSet<S>();
        for(S p : qset) {
            Map<A, HashSet<S>> rho = delta.get(p);
            if(rho != null && rho.get(a) != null) {
                H.addAll(rho.get(a));
            }
        }
        return H;
    }

    public Set<S> simulate(S q, List<A> word) {
        Set<S> R = new HashSet<S>();
        R.add(q);
        for(A a : word) {
            R = simulateOneStep(R, a);
        }
        return R;
    }

    public void addTransition(S q, A a, S p) {
        Map<A, HashSet<S>> rho = delta.get(q);
        HashSet<S> target = rho.get(a);
        if(target != null) {
            target.add(p);
        }
        else {
            target = new HashSet<S>();
            target.add(p);
            rho.put(a, target);
        }
    }
}

```

Abbildung 1: Implementierung eines einfachen NFAs

```

import java.util.*;

public class H8 {
    static public void main(String args[]) {
        Set<Integer> states = new HashSet<Integer>();
        for(int q = 1; q ≤ 34; q++) {
            states.add(q);
        }
        NFA<Integer, Character> M = new NFA<Integer, Character>(states);
        Scanner stdin = new Scanner(System.in);
        while(stdin.hasNextLine()) {
            String line = stdin.nextLine();
            if(line.length() == 0) break;
            String s[] = line.split(" ");
            int q = Integer.parseInt(s[0]);
            char a = s[1].charAt(0);
            int p = Integer.parseInt(s[2]);
            M.addTransition(q, a, p);
        }
        List<Character> word = new ArrayList<Character>();
        String w = "ababbbaa";
        for(int i = 0; i < w.length(); i++) {
            word.add(w.charAt(i));
        }
        System.out.println(M.simulate(7, word));
    }
}

```

Abbildung 2: Programm, das Transitionen einliest und auf dem gegebenen Wort simuliert.

### Lösungsvorschlag:

Wir konstruieren eine unendliche Menge  $N$  und zeigen, dass alle Elemente aus  $N$  in verschiedenen Äquivalenzklassen von  $\equiv_L$  liegen. Daraus folgt, dass  $\equiv_L$  einen unendlichen Index hat und somit mit dem Satz von Myhill–Nerode nicht regulär ist.

Sei  $N = \{\text{😄}^n \text{💩} \mid n \in \mathbb{N}\}$  und  $u, v \in N$ . Es gilt  $u = \text{😄}^i \text{💩}$  und  $v = \text{😄}^j \text{💩}$  für irgendwelche  $i \neq j$ . Des weiteren gilt  $uu^r = \text{😄}^i \text{💩} \text{😄}^i \in L$ , aber  $vu^r = \text{😄}^j \text{💩} \text{😄}^i \notin L$ . Das bedeutet, dass  $u$  und  $v$  in verschiedenen Äquivalenzklassen bezüglich  $\equiv_L$  liegen.