RWTH Aachen **SS 2017**
Lehrgebiet Theoretische Informatik Blatt 4
Rossmanith–Dreier–Hark–Kuinke 22.5.2017

Lösungsvorschlag zur Vorlesung Formale Sprachen, Automaten und Prozesse

Aufgabe T11

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 duch $(ab)^*(b+aa)(a+b)^*$ 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

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.

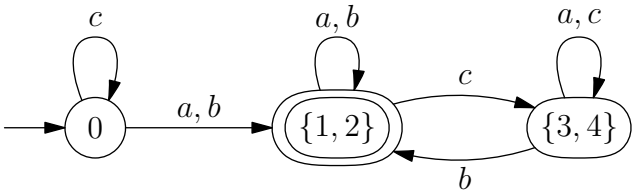
1. Seien $N = a^*b$ und $u, v \in N$ mit $u = a^ib$ und $v = a^jb$ für irgendwelche $i \neq j$. Somit gilt $uu \in L$, aber $vu = a^jba^ib \notin L$. Dies bedeutet, daß u und v in verschiedenen Äquivalenzklassen von $\equiv_L$ liegen.

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

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

a) 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.

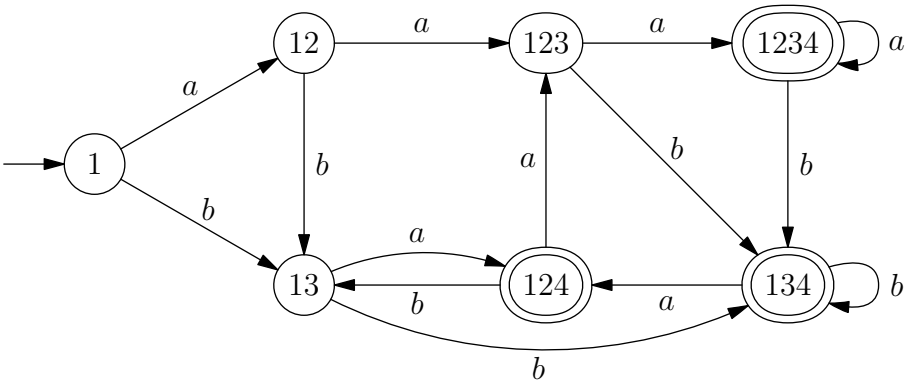
b) Falsch: Wenn ein NFA mit 10 Zuständen eine Sprache akzeptiert, dann hat der zugehörige Potenzmengenautomat höchstens $2^{10} = 1024 < 1217$ Zustände.

c) 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 Komplementärautomat 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.

d) 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 ϵ nicht akzeptiert werden darf.

Aufgabe H7 (5+5+5 Punkte)

a) Mit der Potenzmengenkonstruktion erhält man folgenden DFA.



b) Der Übersichthalber 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)

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

Ein Programm, welches die Transitionen eines Automaten mit Zustandsmenge $\{1, \ldots, 34\}$ einliest und auf dem Wort *ababbbaa* simuliert, ist in Abbildung **??** 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)

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

```
import java.util.*;

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

  public NFA(Set<S> Q) {
    for(S q : Q) {
      addState(q);
    }
  }

  public NFA() { }

  public void addState(S q) {
    delta.put(q, new HashMap<A, HashSet<S>>());
    Q.add(q);
  }

  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.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);
    java.util.Scanner stdin = new java.util.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.