Formale Systeme, Automaten, Prozesse, SS 2020
Prof. Dr. P. Rossmanith
J. Dreier, T. Hartmann, H. Lotze, D. Mock

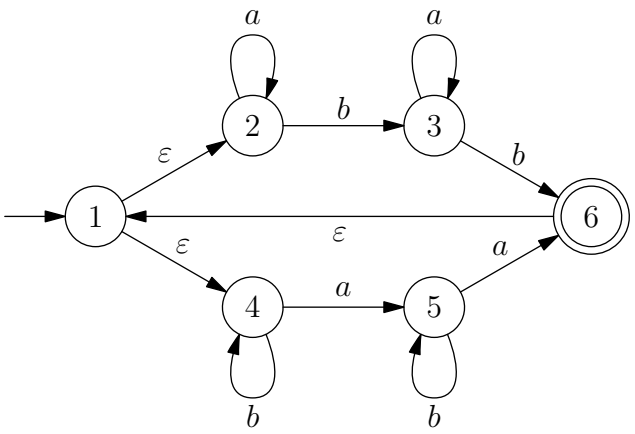
Datum: 27.04.2020

Übungsblatt mit Lösungen 03

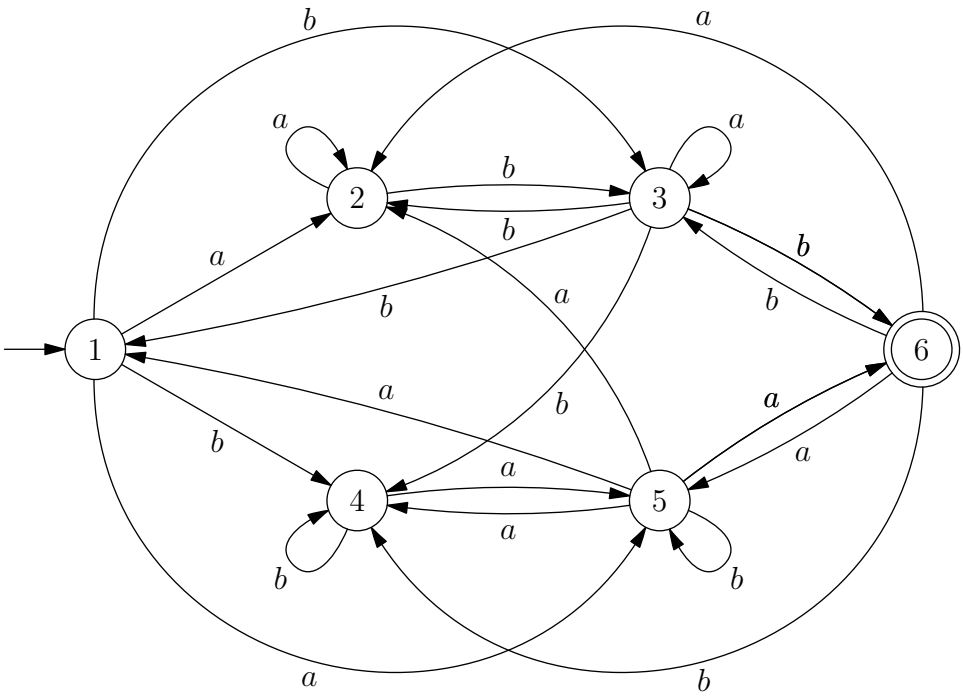
Schreiben Sie Ihre Matrikelnummern und Ihre Tutoriumsnummer auf Ihre Abgabe!

Aufgabe T9

Wandeln Sie den folgenden NFA mit ϵ-Übergängen in einen NFA (ohne ϵ-Übergänge) um.



Lösungsvorschlag



Aufgabe T10

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 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 T11

Gegeben sei der reguläre Ausdruck $r = (a^* + b)^*$.

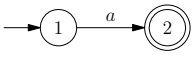
a) Geben sie mit Hilfe der Thompson-Konstruktion einen ϵ-NFA an, der $L(r)$ erkennt.

b) Gegeben sei ein ϵ-NFA mit einem gerichteten Kreis C, der nur aus ϵ-Transitionen besteht. Erklären Sie, wie man die Zustände in C zu einem neuen Zustand q_C kontrahieren kann, um einen neuen ϵ-NFA zu erhalten, der dieselbe Sprache erkennt. Benutzen Sie diese Konstruktion, um den in a) konstruierten ϵ-NFA zu verkleinern.

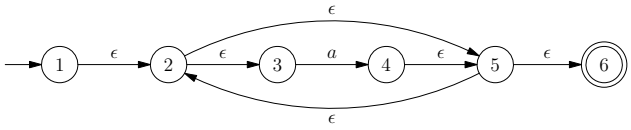
c) Konstruieren Sie aus dem ϵ-NFA in b) einen äquivalenten NFA (ohne ϵ-Übergänge).

Lösungsvorschlag

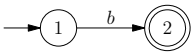
a) Wir beginnen mit einem ϵ-NFA für den Ausdruck a.



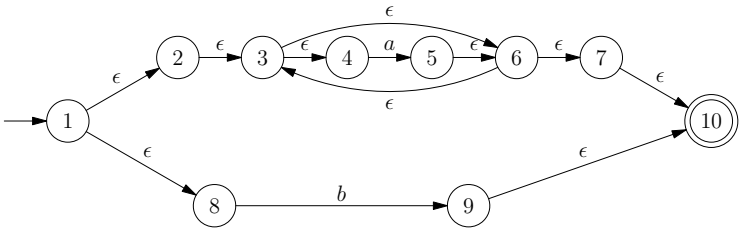
Nun erstellen wir einen ϵ-NFA für a^*.



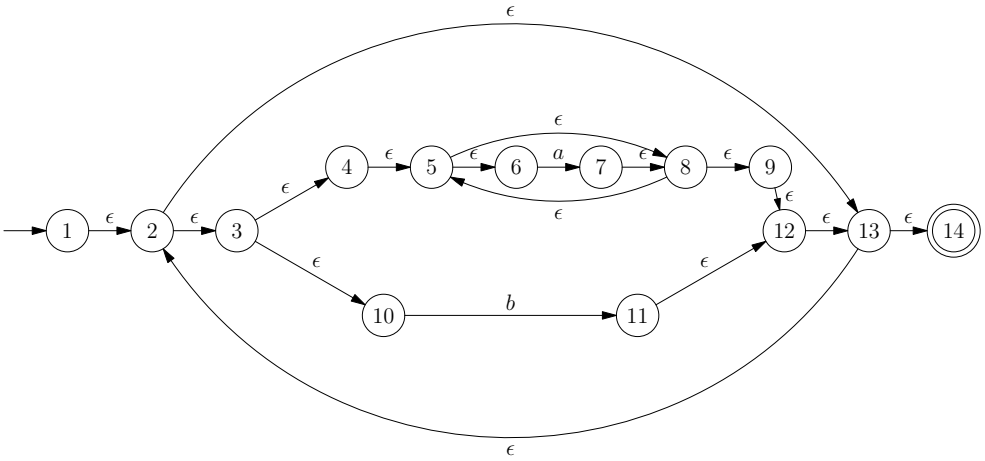
Wir wiederholen die Prozedur für b.



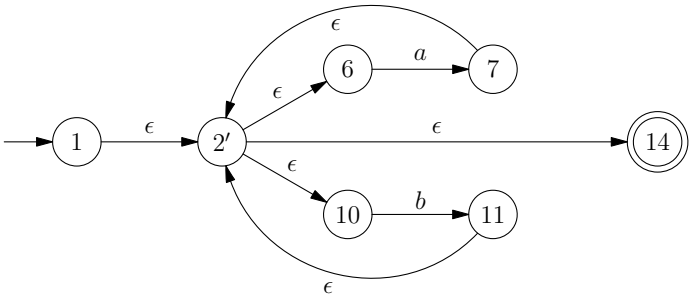
Wir erhalten einen ϵ-NFA für $(a^* + b)$.



Schließlich ergibt sich ein ϵ-NFA für $(a^* + b)^*$.



b) Eine Begründung kann wie folgt aussehen. Sei $C = (q_1 \dots q_n,\ q_1)$ ein solcher Kreis. Jeder Lauf des Automaten, der einen Zustand q_i enthält, kann so erweitert werden, dass nach q_i der gesamte Kreis anhand der ϵ-Transitionen einmal durchlaufen wird, d.h. er die Form $\dots,\ q_i, q_{i+1 \bmod n}, \dots q_i,\ \dots$ hat ohne den letzten Zustand des Laufes zu verändern. Im zusammengeschrumpften Automaten entspricht dies dann dem Lauf $\dots, q_C, \dots$. Also induziert jeder akzeptierende Lauf im zusammengeschrumpften Automaten einen akzeptierenden Lauf im Ausgangsautomaten und umgekehrt. Insbesondere ändert sich die Menge der akzeptierten Wörter nicht, d.h. die Automaten sind äquivalent. Ein ϵ-Kreis ist hier $(2, 3, 4, 5, 8, 9, 12, 13)$, sodass wir diese acht Zustände zu einem neuen Zustand $2'$ zusammenschrumpfen können.



c) Wir berechnen die ϵ-Hüllen der Zustände.

Zustand	ϵ-Hülle
1	$\{1, 2', 6, 10, 14\}$
$2'$	$\{2', 6, 10, 14\}$
6	$\{6\}$
7	$\{2', 6, 7, 10, 14\}$
10	$\{10\}$
11	$\{2', 6, 10, 11, 14\}$
14	$\{14\}$

Jeder Zustand, der den Zustand 14 in seiner ϵ-Hülle enthält, wird im NFA ohne ϵ-Transitionen zu einem Endzustand. Mit Hilfe der Tabelle ergibt sich beim Eliminieren der ϵ-Transitionen die unten angegebene Übergangsfunktion δ. Für eine einfachere Schreibweise seien $A = \{2', 6, 7, 10, 14\}$ und $B = \{2', 6, 10, 11, 14\}$.

Der resultierende NFA ohne ϵ-Transitionen ist somit

$$(\{1, 2', 6, 7, 10, 11, 14\}, \{a, b\}, \delta, 1, \{1,\ 2',\ 7,\ 11,\ 14\}).$$

$$\begin{gathered}
\delta(1, a) = A \\
\delta(1, b) = B \\
\delta(2', a) = B \\
\delta(2', b) = B \\
\delta(6, a) = A \\
\delta(7, a) = A \\
\delta(7, b) = B \\
\delta(10, b) = B \\
\delta(11, a) = A \\
\delta(11, b) = B
\end{gathered}$$

Die graphische Darstellung des Automaten sieht wie folgt aus, wobei nicht beschriftete Transitionen den Wert „a, b“ haben.

Aufgabe H7 (10 Punkte)

Entwerfen Sie einen effizienten Algorithmus, der für einen gegebenen regulären Ausdruck R entscheidet, ob $L(R)$ endlich viele Wörter enthält.

Was ist die Laufzeit Ihres Algorithmus?

Überprüfen Sie dazu rekursiv, ob ein regulärer Ausdruck (i) leer ist, (ii) nicht-leere Wörter enthält oder (iii) endlich viele Wörter enthält.

Lösungsvorschlag

Für einen regulären Ausdruck betrachten wir die folgenden drei Eigenschaften:

(i) $L(R)$ ist leer

(ii) $L(R)$ enthält ein nicht-leeres Wort

(iii) $L(R)$ enthält endlich viele Wörter

Für atomare Ausdrücke gilt: $\emptyset$ erfüllt genau (i) und (iii), $a \in \Sigma$ erfüllt genau (ii) und (iii), und ε erfüllt genau (iii). Induktiv gilt weiterhin:

AB erfüllt (i) genau wenn A oder B (i) erfüllen.

AB erfüllt (ii) genau wenn A oder B (ii) erfüllen und A und B nicht (i) erfüllen.

AB erfüllt (iii) genau wenn AB (i) erfüllt, oder A oder B (iii) erfüllen,

$A + B$ erfüllt (i) genau wenn A und B (i) erfüllen.

$A + B$ erfüllt (ii) genau wenn A oder B (ii) erfüllen.

$A + B$ erfüllt (iii) genau wenn A und B (iii) erfüllen.

A^* erfüllt nie (i).

A^* erfüllt (ii) genau wenn A (ii) erfüllt.

A^* erfüllt (iii) genau wenn A nicht (ii) und A nicht (iii) erfüllt.

Wir entscheiden ob $L(R)$ unendlich viele Worte enthält, indem wir rekursiv über den Aufbau des regulären Ausdrucks (i), (ii) und (iii) berechnen. Dies geht in linearer Zeit.

Aufgabe H8 (10 Punkte)
Bestimmen Sie die Äquivalenzklassen $\equiv_{L_1}$ und $\equiv_{L_2}$. (Beweisen Sie Ihre Behauptung.)

1) $L_1 = \{a, b\}^* c$ über dem Alphabet $\Sigma = \{a, b, c\}$
2) $L_2 = \{\, a^{n^2} \mid n \geq 0 \,\} = \{\epsilon, a, aaaa, aaaaaaaaa, \ldots\}$ über dem Alphabet $\Sigma = \{a\}$.

Lösungsvorschlag

1) Die Äquivalenzklassen von L_1 sind $(ab)^*$, $(ab)^*c$ und $(ab)^*c(abc)^+$. Diese Mengen sind Teilmengen einer Klasse, denn

   - Für alle $u \in (ab)^*$ und $w \in \Sigma^*$ gilt, dass $uw \in L_1$ gdw. $w \notin c(ab)^+$.
   - Für alle $u \in (ab)^*c$ und $w \in \Sigma^*$ gilt, dass $uw \in L1$ gdw. $c = \varepsilon$.
   - Für alle $u \in (ab)^*c(abc)^+$ und $w \in \Sigma^*$ gilt, dass $uw \notin L_1$.

   Zudem bilden diese Mengen verschieden Klassen, denn

   - Für $u \in (ab)^*$, $v \in (ab)^*c$, gilt $uc \in L_1$ aber $vc \notin L_1$.
   - $u \in (ab)^*c(abc)^+$ ist nicht äquivalent zu Wörtern aus $(ab)^*$ und $(ab)^*c$, denn für kein $w \in \Sigma^*$ gilt $uw \in L_1$.

   Da außerdem $(ab)^* \cup (ab)^*c \cup (ab)^*c(abc)^+ = \Sigma^*$, bilden diese drei Mengen die Äquivalenzklassen von L_1.

2) Die Äquivalenzklassen von L_2 sind $\{\{a^n\} \mid n \geq 0\}$, also jedes Wort aus Σ^* bildet ihre eigene Äquivalenzklasse. Betrachte verschiedene Wörter $a^i, a^j \in \Sigma^*$, also mit $0 \leq i < j$. Sei x^2 und y^2 die nächst größere Quadratzahl von i bzw. j. Falls $x^2 - i \neq y^2 - j$, dann sei $k = \min i, j$. Falls $k = i$, dann ist $a^i a^{x^2-k} = a^{x^2} \in L_2$ aber $a^j a^{x^2-k} = a^{x^2+i-j} \notin L_2$ da $x^2 + i - j < y^2$. Für $k = j$ ist analog auch nur genau einer der Wörter $a^i a^{x^2-k}$, $a^j a^{x^2-k}$ Teil von L_2.

   Es bleibt der Fall übrig, dass $d = x^2 - i = y^2 - j$. Dann gilt $x^2 \neq y^2$. Betrachten wir die Differenz zur der jeweils nächst größeren Quadratzahl. Falls $(x+1)^2 - x^2 = (y+1)^2 - y^2$, dann folgt der Widerspruch $x = y$. Da $x < y$, folgt also $(x+1)^2 - x^2 < (y+1)^2 - y^2$. Daher ist $a^i a^{(x+1)^2-x^2+d} = a^{x^2} \in L_2$ aber $a^j a^{(x+1)^2-x^2+d} \notin L_2$, denn $y^2 < j + (x+1)^2 - x^2 + d < (y+1)^2$.

Aufgabe H9 (10 Punkte)

Programmieren Sie einfache NFAs in Java, C, C++, Python oder Haskell. (Ihre Tutorin bzw. Ihr Tutor kann Ihnen eine andere Programmiersprache erlauben, wenn er bzw. sie es möchte.) 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 $\{1, \ldots, 34\}$ und das Eingabealphabet $\{a, b\}$. Was ist das Ergebnis? Zusätzlich finden Sie auf der Website eine Datei mit vier langen Eingabewörtern, welche durch Zeilenumbrüche voneinander getrennt sind. Berechnen Sie für jedes Wort wieder die resultierende Zustandsmenge, wenn man vom Zustand 7 aus beginnt. Messen Sie hierfür die Laufzeit ihres Programms.

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 $\{1, \ldots, 34\}$ einliest und auf dem Wort *ababbbaa* simuliert, ist in Abbildung 2 enthalten. Lassen wir das Programm für *abababbaa* laufen, erhalten wir die Ausgabe

$$[1, 2, 3, 5, 6, 8, 9, 10, 11, 17, 22, 23, 25, 27, 28, 29, 31, 32, 34].$$

Für die Wörter aus der Datei erhalten wir:

$$[32, 1, 34, 2, 3, 5, 6, 8, 9, 10, 11, 17, 20, 21, 25, 27, 28, 29, 31]$$
$$[32, 1, 34, 3, 4, 5, 6, 8, 9, 12, 13, 17, 18, 19, 25, 27, 29, 30, 31]$$
$$[32, 1, 34, 3, 4, 5, 6, 8, 9, 17, 18, 19, 25, 27, 29, 30, 31]$$
$$[32, 1, 34, 3, 4, 5, 6, 8, 9, 17, 18, 19, 25, 27, 29, 30, 31]$$

```
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);
    }
  }
}
```

Abb. 1: Implementierung eines einfachen NFAs

```
import java.util.*;
import java.io.File;
import java.util.Scanner;
import java.io.FileNotFoundException;

public class H09_2020{
  static public void main(String args[]) {
    File nfaFile = new File("2020_H09.trans");
    File inputWords = new File("2020_H09_input");
    String w = "abababbaa";
    Set<Integer> states = new HashSet<Integer>(34);
    for(int q = 1; q <= 34; q++) {
      states.add(q);
    }
    NFA<Integer, Character> M = null;
    try {
      M = parseNFA(states, new Scanner(nfaFile));
    }catch(FileNotFoundException e) {
      e.printStackTrace();
      return;
    }

    System.out.println(M.simulate(7, toList(w)));
    System.out.println("Lange Wörter:");
    try {
      Scanner sc = new Scanner(inputWords);
      while(sc.hasNextLine()) {
        String line = sc.nextLine();
        System.out.println(M.simulate(7, toList(line)));
      }
      sc.close();
    }catch(FileNotFoundException e) {
      e.printStackTrace();
    }
  }

  static public NFA<Integer, Character> parseNFA(Set<Integer> states, Scanner scanner) {
    NFA<Integer, Character> M = new NFA<>(states);
    while(scanner.hasNextLine()) {
      String line = scanner.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);
    }
    return M;
  }

  static public List<Character> toList(String w) {
    List<Character> word = new ArrayList<Character>(w.length());
    for(int i = 0; i < w.length(); i++) {
      word.add(w.charAt(i));
    }
    return word;
  }
}
```

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