Technische Universität München
Fakultät für Informatik
Dr. M. Schneider, Dr. M. Pizka,
Dr. A. Vilbig, Dr. R. Letz,
G. Bauer

SS 2003
Lösungsvorschläge zu Blatt 2
5. Mai 2003

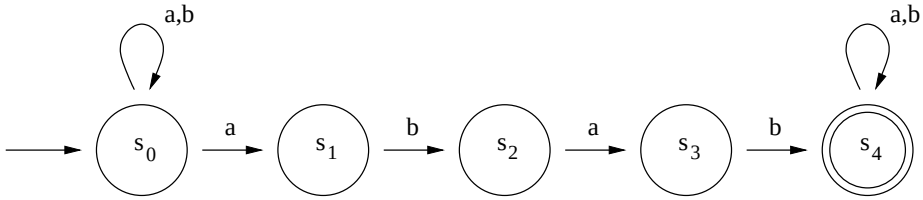
Übungen zur Vorlesung Einführung in die Informatik IV

Aufgabe 4 Chomsky-3 Sprachen

Wir betrachten die Sprache L aller Wörter über der Zeichenmenge $T = \{a, b\}$, die das Wort *abab* als Teilwort enthalten.

(a) L wird beschrieben durch den regulären Ausdruck $[a|b]^*abab[a|b]^*$.

(b) L wird akzeptiert durch den Automat $A = (S, T, s_0, S_Z, \delta)$ wobei $S = \{s_0, \ldots, s_4\}$, $T = \{a, b\}$, $S_Z = \{s_4\}$ und die Übergangsrelation δ wie folgt gegeben ist:



(c) L wird akzeptiert durch die Grammatik $G = (T, N, \rightarrow, Z)$ wobei

$$
\begin{aligned}
T &= \{a, b\} \\
N &= \{s_0, \ldots, s_4, Z\} \\
\rightarrow &= \{a \rightarrow s_0, \\
&\quad b \rightarrow s_0, \\
&\quad a \rightarrow s_1, \\
&\quad s_0a \rightarrow s_0 \\
&\quad s_0b \rightarrow s_0 \\
&\quad s_0a \rightarrow s_1 \\
&\quad s_1b \rightarrow s_2 \\
&\quad s_2a \rightarrow s_3 \\
&\quad s_3b \rightarrow s_4 \\
&\quad s_4a \rightarrow s_4 \\
&\quad s_4b \rightarrow s_4 \\
&\quad s_4 \rightarrow Z\}
\end{aligned}
$$

Beispiel:

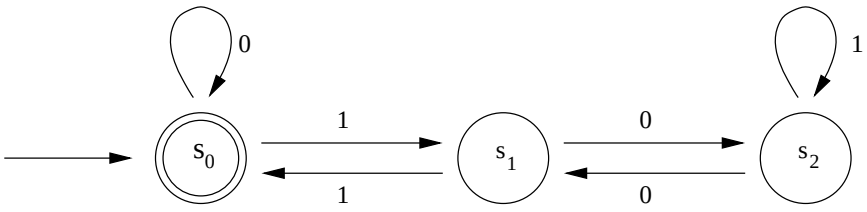
$$
\begin{aligned}
&\underline{a}ababababba\\
\rightarrow\ &\underline{s_0}ababababba\\
\rightarrow\ &\underline{s_0}bababbba\\
\rightarrow\ &\underline{s_0}ababbba\\
\rightarrow\ &\underline{s_1}babbba\\
\rightarrow\ &\underline{s_2}abbba\\
\rightarrow\ &\underline{s_3}bbba\\
\rightarrow\ &\underline{s_4}bba\\
\rightarrow\ &\underline{s_4}a\\
\rightarrow\ &\underline{s_4}\\
\rightarrow\ &\underline{Z}
\end{aligned}
$$

Aufgabe 5 Endliche Automaten

(a) Die Menge der durch 3 teilbaren (positiven) Binärzahlen wird akzeptiert durch den deterministischen endlichen Automat $A = (S, T, s_0, S_Z, \delta)$, wobei $S = \{s_0 \ldots s_4\}$, $T = \{0, 1\}$, $S_z = \{s_0\}$.

Die Eingabe wird beginnend mit dem höchstwertigen Bit gelesen. Jeder Zustand s_i entspricht hier dem Divisionsrest i mod 3 der bisher gelesenen Eingabe. Dadurch ergibt sich die Übergangsrelation wie folgt:

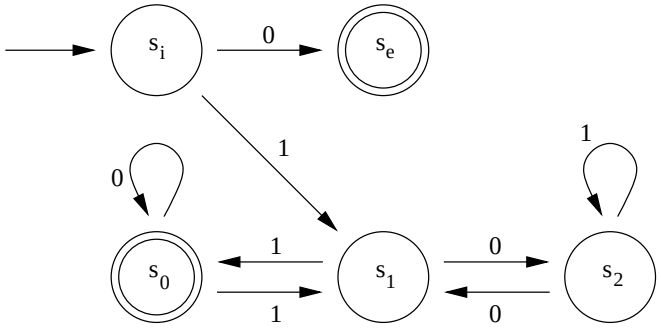
$$
\begin{aligned}
\delta(s_0, 0) &= s_0 \quad & 2 \cdot (0 \bmod 3) + 0 &= 0 \bmod 3\\
\delta(s_0, 1) &= s_1 \quad & 2 \cdot (0 \bmod 3) + 1 &= 1 \bmod 3\\
\delta(s_1, 0) &= s_2 \quad & 2 \cdot (1 \bmod 3) + 0 &= 2 \bmod 3\\
\delta(s_1, 1) &= s_0 \quad & 2 \cdot (1 \bmod 3) + 1 &= 0 \bmod 3\\
\delta(s_2, 0) &= s_1 \quad & 2 \cdot (2 \bmod 3) + 0 &= 1 \bmod 3\\
\delta(s_2, 1) &= s_2 \quad & 2 \cdot (2 \bmod 3) + 1 &= 2 \bmod 3
\end{aligned}
$$



Da Binärzahlen keine führenden Nullen haben sollen und auch das leere Wort nicht akzeptiert werden soll, modifizieren wir den Automat, indem wir einen Startwert s_i und einen weiteren Endzustand s_e einführen.

Bemerkung: Der angegebene Automat ist nicht total. Durch Einführung von Fangzuständen kann er in einen totalen verwandelt werden.

(b) Sei $A = (S, T, s_0, S_Z, \delta)$ der endliche Automat, der die Sprache L akzeptiert. Aus diesem erhält man einen endlichen Automaten A^R, der L^R akzeptiert, indem man einen neuen Anfangsustand s_0^R einführt, von dem aus es ε-Übergänge zu allen Zuständen in S_Z gibt.



Die Übergangsfunktion (bzw. Relation) δ^R besteht außer diesen ε-Übergängen aus der Umkehrfunktion (bzw. Relation) von δ (d.h. alle Pfeile werden umgedreht). Der neue akzeptierende Zustand ist s_0.

A^R hat also die Form $(S \cup \{s_0^R\}, T, \delta^R, s_0^R, \{s_0\})$.

Die eingeführten ε-Übergänge können dann nach dem in der Vorlseung beschriebenen Verfahren wieder beseitigt werden.

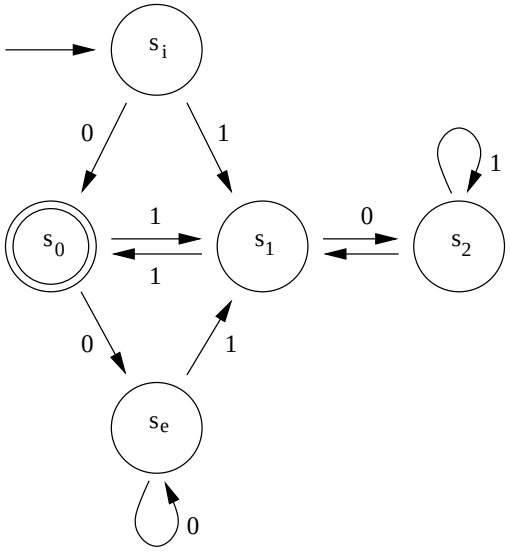
Falls der Automat nur einen Endustand hat, braucht man keinen neuen Zustand s_0^R, sondern kann einfach Anfangs- und Endzustände vertauschen.

(c) Der gesuchte Automat akzeptiert genau die Menge der rückwärts gelesenen Wörter, der von dem Automaten in Aufgabe (a) akzeptierten Sprache.

Man erhält ihn, indem man die in Aufgabe (b) beschriebene Konstruktion anwendet.

Dabei erhält man wieder den gleichen Automaten, denn alle Pfeile sind bidirektional und Anfangs- und Endzustand sind identisch (woraus wir schließen können, dass eine Binärzahl genau dann durch 3 teilbar ist, wenn sie rückwärts gelesen durch 3 teilbar ist).

Wie in Aufgabe (a) müssen wir den Automaten modifizieren, so dass Binärzahlen mit führenden Nullen und das leere Wort nicht akzeptiert werden.

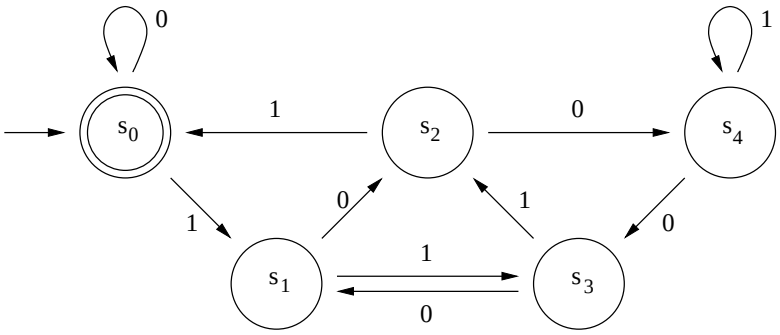


Aufgabe 6 (H) Endliche Automaten

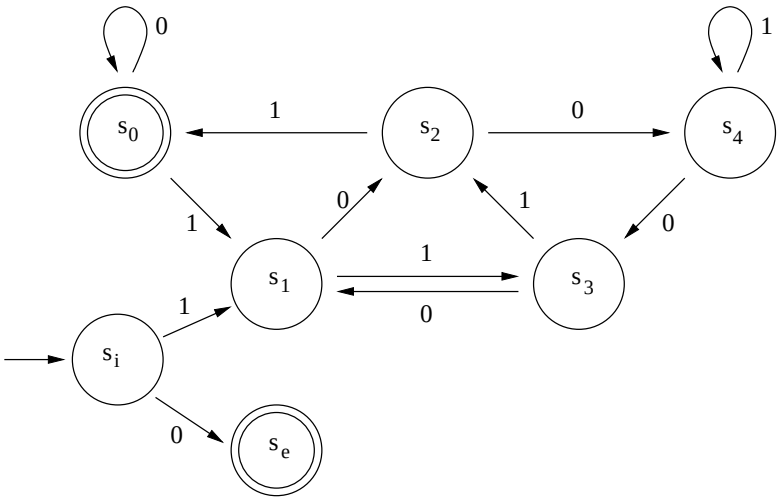
Die Menge der durch 5 teilbaren (positiven) Binärzahlen wird akzeptiert durch den deterministischen endlichen Automat $A = (S, T, s_0, S_Z, \delta)$, wobei $S = \{s_0 \ldots s_4\}$, $T = \{0,1\}$, $S_z = \{s_0\}$.

Die Eingabe wird beginnend mit dem höchstwertigen Bit gelesen. Jeder Zustand s_i entspricht hier dem Divisionsrest i mod 5 der bisher gelesenen Eingabe. Dadurch ergibt sich die Übergangsrelation wie folgt:

$$
\begin{aligned}
\delta(s_0,0) &= s_0 \quad & 2\cdot(0 \bmod 5)+0 &= 0 \bmod 5\\
\delta(s_0,1) &= s_1 \quad & 2\cdot(0 \bmod 5)+1 &= 1 \bmod 5\\
\delta(s_1,0) &= s_2 \quad & 2\cdot(1 \bmod 5)+0 &= 2 \bmod 5\\
\delta(s_1,1) &= s_3 \quad & 2\cdot(1 \bmod 5)+1 &= 3 \bmod 5\\
\delta(s_2,0) &= s_4 \quad & 2\cdot(2 \bmod 5)+0 &= 4 \bmod 5\\
\delta(s_2,1) &= s_0 \quad & 2\cdot(2 \bmod 5)+1 &= 0 \bmod 5\\
\delta(s_3,0) &= s_1 \quad & 2\cdot(3 \bmod 5)+0 &= 1 \bmod 5\\
\delta(s_3,1) &= s_2 \quad & 2\cdot(3 \bmod 5)+1 &= 2 \bmod 5\\
\delta(s_4,0) &= s_3 \quad & 2\cdot(4 \bmod 5)+0 &= 3 \bmod 5\\
\delta(s_4,1) &= s_4 \quad & 2\cdot(4 \bmod 5)+1 &= 4 \bmod 5
\end{aligned}
$$



Dieser Automat kann auch noch modifiziert werden zu einem Automat, der das leere Wort und Binärzahlen mit führenden Nullen nicht akzeptiert.



Aufgabe 7 **Reguläre Grammatik, regulärer Ausdruck**

Reguläre Grammatik:

(1) aZ → Z
(2) aB → Z
(3) b → Z
(4) bZ → B

(a) Ableitung des Wortes abaaabab (jeweils vom rechten Wortende):

abaaabab $\rightarrow_3$ abaaabaZ $\rightarrow_1$ abaaabZ $\rightarrow_4$ abaaaB $\rightarrow_2$ abaaZ $\rightarrow_1$ abaZ $\rightarrow_1$ abZ $\rightarrow_4$ aB $\rightarrow_2$ Z

(b) Reguläre Ausdrücke, die dieselbe Sprache beschreiben: [a|[ab]]*b oder [a*[ab]*]*b

Aufgabe 8 (P) Reguläre Sprache, endlicher Automat

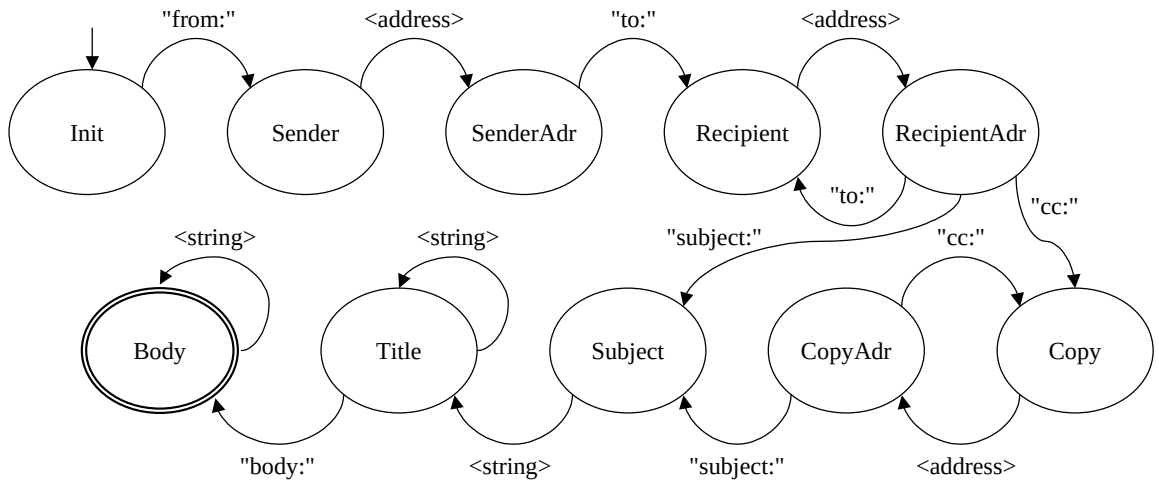
Teilaufgabe a)

Aufbau und Struktur einer gültigen EMail-Nachricht wird durch folgenden regulären Ausdruck beschrieben

'from:' <address> ('to:' <address>)+ ('cc:' <address>)* 'subject:' <string>+ 'body:' <string>*

Hierbei sind <address> und <string> keine Terminalzeichen im eigentlichen Sinne, sondern vielmehr wiederum reguläre Ausdrücke für EMail-Adressen und beliebige Zeichenketten (ohne Schlüsselwörter), deren innere Struktur aus Gründen der Übersichtlichkeit an dieser Stelle vernachlässigt wird. Ihr jeweiliger Inhalt wird später in der Implementierung des Parsers einfach als Ganzes in die Nachricht übertragen.

Teilaufgabe b)



Teilaufgabe c)

Der Lösungsvorschlag setzt den oben aufgeführten Automaten in einer tabellarischen Repräsentation um. Die eingeführten Integer-Konstanten für Zustände und Transitionen erleichtern in diesem Zusammenhang die Fortschaltung des Automaten, auch wenn offensichtlich auch andere Implementierungen, etwa über eigene Klassen für Bestandteile des Automaten, an dieser Stelle möglich sind. Weiterhin wird im folgenden auf eine robuste und ausführliche Fehlerbehandlung und -toleranz bei ungültigen Nachrichten verzichtet – diese lässt sich beispielsweise durch Einführung zusätzlicher Transitionen und die Verfolgung ihres Schaltens verbessern.

```
/*
```

```
 * DFA-based (deterministic finite state automaton) implementation of a parser
 * for a text file with an email message defined by following grammar:
 * <message>    ::= <sender> <recipient>+ <carboncopy>* <subject> <body>
 * <sender>     ::= "from:" <address>
 * <recipient>  ::= "to:" <address>
 * <carboncopy> ::= "cc:" <address>
 * <subject>    ::= "subject:" <string>+
 * <body>       ::= "body:" <string>*
 * <address>    ::= <id>'@'<id>
 * <id>         ::= ['a'..'z','A'..'Z','0'..'9','.','-','_']+
 * <string>     ::= ['a'..'z','A'..'Z','0'..'9','ä'..'Ü','.',',','_']+
 *
 * (assuming that <id> and <string> do not contain key tokens)
 */

import java.util.*;
import java.io.*;

public class DFAMailer {
    private Message msg;
    // Integer constants for states and labels of automaton
    private final int Init = 0, Sender = 1, SAddress = 2, Recipient = 3,
        RAddress = 4, Copy = 5, CAddress = 6, Subject = 7, STitle = 8,
        Body = 9, Error = -1;
    private final int FROM = 0, TO = 1, CC = 2, SUBJECT = 3, BODY = 4,
        ADDRESS = 5, STRING = 6, ERROR = -1;
    // Tabular representation of automaton
    private final int NumStates = 10, NumLabels = 7;
    private int[][] automaton;
    private int currentState;

    public DFAMailer() {
        // Initialize automaton with default state
        automaton = new int [NumStates][NumLabels];
        for (int i = 0; i < NumStates; i++)
            for (int j = 0; j < NumLabels; j++)
                automaton[i][j] = Error;
        // Set appropriate transitions
        automaton[Init]     [FROM]    = Sender;
        automaton[Sender]   [ADDRESS] = SAddress;
        automaton[SAddress] [TO]      = Recipient;
        automaton[Recipient][ADDRESS] = RAddress;
        automaton[RAddress] [TO]      = Recipient;
        automaton[RAddress] [CC]      = Copy;
        automaton[RAddress] [SUBJECT] = Subject;
        automaton[Copy]     [ADDRESS] = CAddress;
        automaton[CAddress] [CC]      = Copy;
        automaton[CAddress] [SUBJECT] = Subject;
        automaton[Subject]  [STRING]  = STitle;
        automaton[STitle]   [STRING]  = STitle;
        automaton[STitle]   [BODY]    = Body;
```

```
    automaton[Body]     [STRING]  = Body;
}

public static void main(String[] args) {
    if (args.length != 1) {
        System.out.println("Usage: java DFAMailer <message file>");
        System.exit(0);
    }
    DFAMailer mailer = new DFAMailer();
    mailer.processMessage(args[0]);
}

private void processMessage(String fileName) {
    String buffer = new String();
    try {
        BufferedReader reader = new BufferedReader(new FileReader(fileName));
        while (reader.ready()) {
            buffer += reader.readLine() + " ";
        }
        reader.close();
    }
    catch (IOException ex) {
        ex.printStackTrace();
        System.exit(1);
    }
    parseMessage(buffer);
    System.out.println("*** Message ***\n" + msg);
}

private void parseMessage(String str) {
    String titleString = new String();
    String bodyString = new String();
    msg = new Message();
    // Set initial state and iterate over all tokens
    currentState = Init;
    StringTokenizer tokenizer = new StringTokenizer(str);
    while (tokenizer.hasMoreTokens()) {
        String token = tokenizer.nextToken();
        int label = encodeToken(token);
        // Check for invalid token and inform user about expected labels
        if (automaton[currentState][label] == Error) {
            System.out.print("-> Error in state \"" + decodeState(currentState)
                + "\" with token \"" + token + "\", expected one of:");
            for (int j = 0; j < NumLabels; j++)
                if (automaton[currentState][j] != Error)
                    System.out.print(" \"" + decodeLabel(j) + "\"");
            System.out.print("\n");
        } else {
            // Update content of message according to current state and label
            if (label == ADDRESS) {
                switch (currentState) {
```

```
                    case Sender:    msg.setSender(token); break;
                    case Recipient: msg.getRecipients().add(token); break;
                    case Copy:      msg.getCarbonCopies().add(token); break;
                }
            }
            if (label == STRING) {
                switch (currentState) {
                    case Subject:
                    case STitle:  titleString += token + " "; break;
                    case Body:    bodyString += token + " "; break;
                }
            }
            if (label == BODY) {
                msg.setSubject(titleString);
            }
            // Perform transition to next state
            currentState = automaton[currentState][label];
        }
    }
    // Check for correct terminal state and update message body accordingly
    if (currentState == Body)
        msg.setBody(bodyString);
    else
        System.out.println("-> Error: Parsing stopped in non-final state \""
           + decodeState(currentState) + "\"");
}

// Utility method for encoding of a token as corresponding integer constant
private int encodeToken(String token) {
    if (token.equals("from:")) return FROM;
    if (token.equals("to:")) return TO;
    if (token.equals("cc:")) return CC;
    if (token.equals("subject:")) return SUBJECT;
    if (token.equals("body:")) return BODY;
    if (isAddress(token)) return ADDRESS;
    if (token.length() > 0) return STRING;
    return ERROR;
}

// Utility method for decoding of a state integer constant
private String decodeState(int num) {
    switch (num) {
        case 0: return "Init";
        case 1: return "Sender";
        case 2: return "SAddress";
        case 3: return "Recipient";
        case 4: return "RAddress";
        case 5: return "Copy";
        case 6: return "CAddress";
        case 7: return "Subject";
        case 8: return "STitle";
```

```
            case 9: return "Body";
            default: return "Error";
        }
    }

    // Utility method for decoding of a label integer constant
    private String decodeLabel(int num) {
        switch (num) {
            case 0: return "from:";
            case 1: return "to:";
            case 2: return "cc:";
            case 3: return "subject:";
            case 4: return "body:";
            case 5: return "<address>";
            case 6: return "<string>";
            default: return "error";
        }
    }

    // Simple check for a valid email address
    private boolean isAddress(String adr) {
        StringTokenizer tokenizer = new StringTokenizer(adr, "@");
        if (tokenizer.countTokens() != 2) {
            return false;
        }
        return true;
    }
}
```