
Lektion 11

Nachdem in den vorangehenden Kapiteln mit regulären Ausdrücken und kontextfreien Grammatiken zwei Methoden vorgestellt wurden, wie Sprachen definiert werden können, geht es in den folgenden Kapiteln nun darum, wie geprüft werden kann, ob ein Wort zu einer Sprache gehört.

Eine Möglichkeit, die es auf einfache Art erlaubt zu prüfen, ob ein Wort zu einer Sprache gehört, sind sog. *endliche Automaten*. In Kapitel 7 wird zunächst die deterministische Variante davon eingeführt, in Kapitel 8 dann auch die nichtdeterministische Erweiterung.

Kap. 7 Deterministische endliche Automaten

Inhalt

- ▶ Definition deterministischer endlicher Automaten (DEA)
- ▶ Akzeptierte Sprache eines Automaten
- ▶ Minimierung von Automaten
- ▶ Nachweis der Äquivalenz von Automaten

7.1 Einführung

Beispiel 7.1 - 0/1-Folgen prüfen

Es soll geprüft werden, ob eine (möglicherweise sehr lange) 0/1-Folge, d.h. ein Wort aus $\{0,1\}^*$, eine ungerade Anzahl von Nullen und auch eine ungerade Anzahl von Einsen enthält. Es ist also zu prüfen, ob ein gegebenes Wort zur Sprache

$$L = \{ w \in \{0,1\}^* \mid |w|_0 \text{ ist ungerade} \wedge |w|_1 \text{ ist ungerade} \}$$

gehört.

Lösungsansatz 1

Wäre die Folge als ein String in Java gegeben, dann wäre es sehr einfach, ein Programm dafür zu erstellen, z.B. in folgender Weise.

```
public static boolean test01(String word) {  
    int count0 = 0;  
    int count1 = 0;
```

```

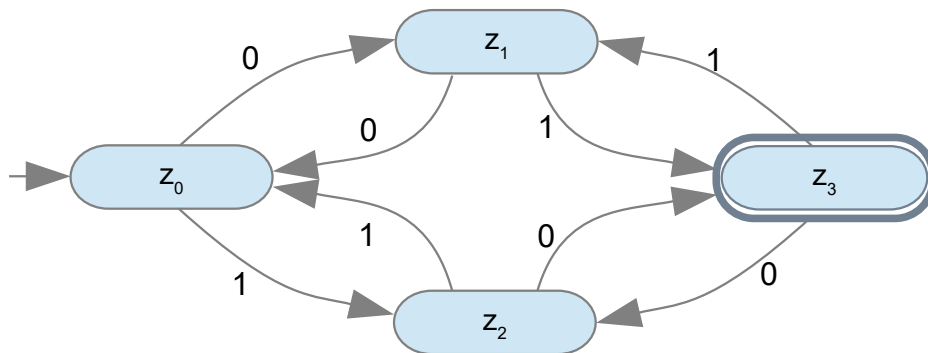
for (int i = 0; i < word.length(); i++) {
    switch(word.charAt(i)) {
        case '0':
            count0++; break;
        case '1':
            count1++; break;
        default:
            throw new RuntimeException("unexpected character");
    }
}
return (count0 % 2 == 1) && (count1 % 2 == 1);
}

```

Die Lösung funktioniert, zumindest solange die Zeichenkette nicht so lang ist, dass es einen Überlauf beim Zählen gibt.

Lösungsansatz 2

Auf abstrakter Ebene könnte man das Problem auch auf folgende spielerische Weise lösen. Es ist folgender "Spielplan" gegeben:



Ist ein Wort gegeben, das geprüft werden soll, z.B. 011100, dann startet man an der Position z_0 , die mit Pfeil markiert ist. Es werden dann alle Zeichen des Wortes nacheinander durchgegangen und für jedes Zeichen folgt man jeweils dem Pfeil mit dem entsprechenden Zeichen. Z.B. würde man mit der ersten 0 zur Position z_1 wechseln und mit der folgenden 1 nach z_3 .

Landet man am Ende an Position z_3 , dann erfüllt das Wort die Eigenschaft, dass sowohl die Zahl der Nullen als auch die der Einsen ungerade ist.

Die korrekte Funktionsweise kann man sich folgendermaßen klar machen: Hat man einen Zustand erreicht, dann war folgendes der Fall:

- z_0 : die Zahl der Nullen ist *gerade* und die Zahl der Einsen ist *gerade*
- z_1 : die Zahl der Nullen ist *ungerade* und die Zahl der Einsen ist *gerade*
- z_2 : die Zahl der Nullen ist *gerade* und die Zahl der Einsen ist *ungerade*
- z_3 : die Zahl der Nullen ist *ungerade* und die Zahl der Einsen ist *ungerade*

Diese Vorgehensweise hat den Vorteil, dass sie auch mit beliebig langen Folgen funktioniert. (Natürlich könnte man das entsprechend auch in Java mit zwei boolean-Variablen implementieren).

Das, was hier als "Spielplan" bezeichnet wurde, ist die Darstellung eines sog. **endlichen deterministischen Automaten (DEA)**. DEAs sind abstrakte Maschinen zur Prüfung von Wörtern.

7.2 Definition deterministischer endlicher Automaten

Definition 7.2 - Deterministischer endlicher Automat

Ein **deterministischer endlicher Automat (DEA)** (engl. *deterministic finite automaton, DFA*)

$$A = (Z, \Sigma, \delta, z_0, E)$$

besteht aus:

Z *endliche Menge von Zuständen*

Σ *Eingabealphabet*

$\delta: Z \times \Sigma \rightarrow Z$ *Zustandsübergangsfunktion (totale Funktion)*

$z_0 \in Z$ *Startzustand*

$E \subseteq Z$ *Menge von akzeptierenden Zuständen
(auch Endzustände genannt)*

Anmerkungen

▶ Bezeichnung "*endlicher*" Automat, da die Menge der Zustände endlich ist

▶ Übergangsfunktion δ beschreibt folgendes:

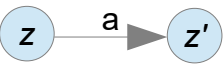

$\delta(z, a)$ ist der Zustand, den man mit Zeichen a von Zustand z aus erreicht

▶ Wenn $\delta(z, a) = z'$, d.h. wenn man mit Zeichen a vom Zustand z nach Zustand z' kommt, notieren wir das auch so:

$$z \xrightarrow{a} z'$$

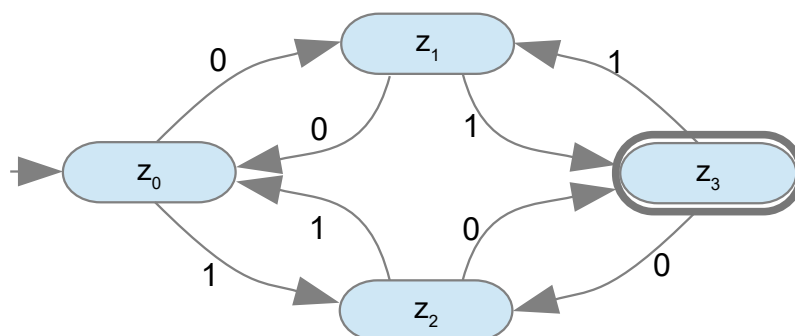
Darstellung eines DEA als Übergangsdiagramm

Der "Spielplan" oben zur Prüfung der 0/1-Folgen beschreibt auf grafische Weise einen DEA mit vier Zuständen $Z = \{z_0, z_1, z_2, z_3\}$. Die dort verwendete Darstellungsweise als gerichteter Graph wird meist zur anschaulichen Darstellung von Automaten verwendet:

Darstellung	Bedeutung
	Zustand
	Zustandsübergang $\delta(z,a) = z'$
	Startzustand
	akzeptierender Zustand (Endzustand)

Beispiel 7.3 - DEA als Übergangsdiagramm

Das Beispiel



von oben stellt also den Automaten $A = (Z, \Sigma, \delta, z_0, E)$ dar, wobei

$$Z = \{z_0, z_1, z_2, z_3\}$$

endliche Zustandsmenge

$$\Sigma = \{0, 1\}$$

Eingabealphabet

$$\delta : Z \times \Sigma \rightarrow Z$$

Zustandsübergangsfunktion

$$\delta(z_0, 0) = z_1$$

$$\delta(z_0, 1) = z_2$$

$$\delta(z_1, 0) = z_0$$

$$\delta(z_1, 1) = z_3$$

$$\delta(z_2, 0) = z_3$$

$$\delta(z_2, 1) = z_0$$

$$\delta(z_3, 0) = z_2$$

$$\delta(z_3, 1) = z_1$$

z_0 ist Startzustand

$$E = \{z_3\}$$

Menge der akzeptierenden Zustände (hier nur einer)

Aufgabe 7.4 - DEA zeichnen

Gegeben ist folgende Definition für einen deterministischen endlichen Automaten:

$$A = (Z, \Sigma, \delta, z_0, E) \text{ mit}$$

$$Z = \{z_0, z_1, z_2, z_3\}$$

$$\Sigma = \{a, b\}$$

$$\delta(z_0, a) = z_1$$

$$\delta(z_0, b) = z_2$$

$$\delta(z_1, a) = z_3$$

$$\delta(z_1, b) = z_1$$

$$\delta(z_2, a) = z_1$$

$$\delta(z_2, b) = z_3$$

$$\delta(z_3, a) = z_3$$

$$\delta(z_3, b) = z_3$$

$$E = \{z_2, z_3\}$$

Stellen Sie den Automaten anschaulich in Diagrammform dar.

Deterministisches Verhalten

Unter **Determinismus** (von lateinisch *determinare*, "festlegen") versteht man, dass das Verhalten eines Systems und seine Reaktion auf Ereignisse durch Vorbedingungen eindeutig vorbestimmt sind, d.h. dass das System also nichts "zufällig" machen kann und somit immer eindeutig voraussehbar ist, wie sich das System verhalten wird, wenn der aktuelle Systemzustand bekannt ist.

Bei DEAs beschreibt die Zustandsübergangsfunktion δ die möglichen Reaktionen in Form von Zustandsübergängen. Da δ eine Funktion ist und es für jeden Zustand und jedes Zeichen des Alphabets genau einen Nachfolgezustand gibt (der auch der gleiche Zustand wieder sein darf), ist das Verhalten vollständig deterministisch.

Sie werden im nächsten Kapitel dann eine Variante von endlichen Automaten kennenlernen, die auch nichtdeterministisches Verhalten zulässt.

7.2.1 Zustandsübergangstabellen

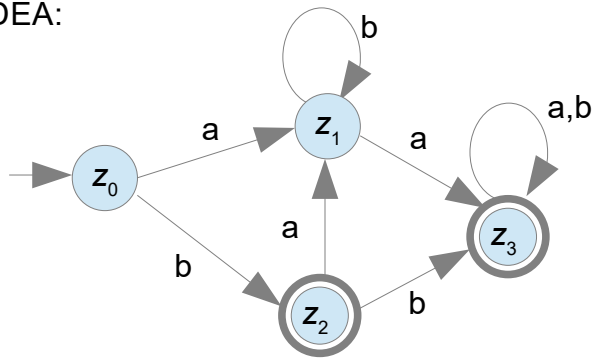
Tabellarische Darstellung der Zustandsübergänge

Die Funktion δ beschreibt die möglichen Zustandsübergänge. Für jeden Zustand und jedes Zeichen des Alphabets gibt es genau einen Nachfolgezustand, so dass dies auch in Tabellenform als sog. *Zustandsübergangstabelle* dargestellt werden kann.

Welcher Zustand Startzustand ist und welche Zustände akzeptierende Zustände sind, muss dann noch separat angegeben werden

Beispiel 7.5 - Zustandsübergangstabelle

► DEA:



► Zustandsübergangstabelle:

Zustand	Nachfolgezustand	
	a	b
z₀	z ₁	z ₂
z₁	z ₃	z ₁
z₂	z ₁	z ₃
z₃	z ₃	z ₃

Dass δ eine Funktion ist, erkennt man daran, dass es niemals mehr als einen Zustand pro Eintrag in der Tabelle gibt. Da die Zustandsübergangsfunktion total sein muss, darf es keine leeren Einträge in der Tabelle geben darf.

7.3 Akzeptierte Sprache eines DEA

Ist ein Wort $w = a_1 \dots a_n$ gegeben, kann nun in naheliegender Weise der Automat damit durchlaufen werden, indem man Zeichen für Zeichen jeweils zum Nachfolgezustand wechselt, der durch die Zustandsübergangsfunktion bestimmt ist.

Definition 7.6 - Zustandsübergang mit Worts w

Für ein Wort $w \in \Sigma^*$ gibt

$$z \xrightarrow{w} z'$$

an, dass man mit Wort w vom Zustand z zu Zustand z' kommt.

□ Für jeden Zustand z gilt:

$$z \xrightarrow{\epsilon} z$$

d.h. mit dem leeren Wort bleibt man im gleichen Zustand.

□ Für ein Wort $w = a_1 a_2 \dots a_n$ (mit $a_i \in \Sigma$) gilt

$$z_0 \xrightarrow{w} z_n,$$

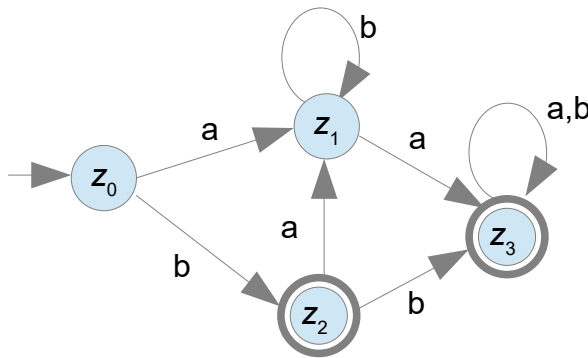
wenn es Zustände z_1 bis z_{n-1} gibt, so dass eine Folge von Zustandsübergängen

$$z_0 \xrightarrow{a_1} z_1 \xrightarrow{a_2} z_2 \dots z_{n-1} \xrightarrow{a_n} z_n$$

von z_0 zu z_n führt.

Aufgabe 7.7 - Zustandsübergänge mit Wörtern

Folgender DEA ist gegeben:



Was trifft zu?

- (1) $z_0 \xrightarrow{ab} z_1$
- (2) $z_2 \xrightarrow{\epsilon} z_2$
- (3) $z_2 \xrightarrow{abab} z_2$
- (4) $z_1 \xrightarrow{bbbbbaa} z_3$

Anmerkung

- Da das Verhalten, das durch die Zustandsübergangsfunktion δ beschrieben wird, für einzelne Zeichen deterministisch ist, ist auch das Verhalten für die Verarbeitung von ganzen Wörtern deterministisch. Startet man bei einem Zustand z , dann gibt es einen eindeutigen Zustand z' , bei dem man mit dem Wort w am Ende landen wird.

Ein Automat kann nun in naheliegender Weise als Prüfverfahren verwendet werden: Um ein Wort zu prüfen, beginnt man im Startzustand. Erreicht man mit dem Wort am Ende einen akzeptierenden Zustand (Endzustand), dann ist die Prüfung erfolgreich, d.h. das Wort gehört dann zur Sprache des Automaten.

Definition 7.8 - Akzeptierte Sprache eines DEA

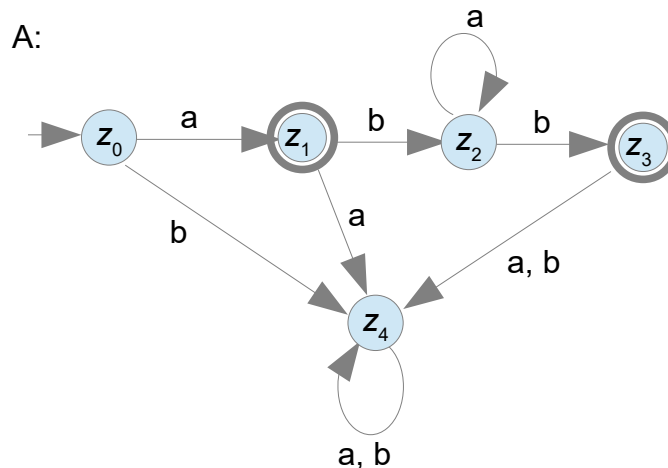
Die durch einen DEA $A = (Z, \Sigma, \delta, z_0, E)$ **akzeptierte Sprache** $L(A)$ ist die Menge

$$L(A) = \{w \in \Sigma^* \mid z_0 \xrightarrow{w} z' \wedge z' \in E\}.$$

Die Sprache $L(A)$ des Automaten A ist die Menge aller Wörter w , mit denen man ausgehend vom Startzustand z_0 einen akzeptierenden Zustand erreicht.

Beispiel 7.9 - Sprache eines DEA

Welche Sprache akzeptiert folgender Automat über Alphabet $\Sigma = \{a, b\}$? D.h. mit welchen Wörtern erreicht man vom Startzustand z_0 aus einen akzeptierenden Zustand (Endzustand)?



- ▶ Mit dem Wort a erreicht man Endzustand z_1 .
- ▶ Mit jedem Wort $aba^k b$ (für $k \geq 0$), d.h. erst a , dann b , dann beliebig viele a und nochmal ein b , erreicht man den Endzustand z_3 .
- ▶ Sobald man in den Zustand z_4 gekommen ist, kann man keinen Endzustand mehr erreichen.

Zusammengefasst ergibt sich also:

$$L(A) = \{ a \} \cup \{ aba^k b \mid k \geq 0 \}$$

Die Sprache könnte man noch bequemer als regulären Ausdruck beschreiben:

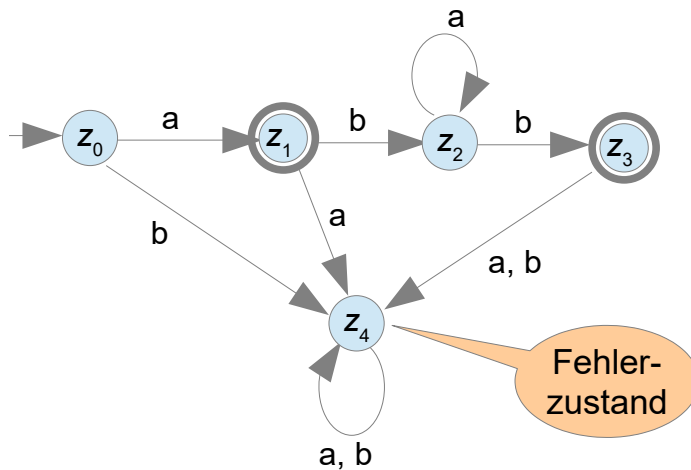
$$a \mid aba^*b$$

bzw. äquivalent

$$a (\varepsilon \mid ba^*b).$$

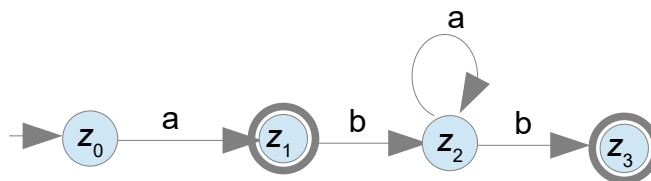
Anmerkung - Fehlerzustände

Nach Definition 7.2 für DEAs ist δ eine totale Funktion, d.h. für jeden Zustand und zu jedem Zeichen muss es einen eindeutigen Nachfolgezustand geben. Will man beschreiben, dass es ab einer bestimmten Stelle nicht mehr erfolgreich weitergehen kann, kann man einen "Fehlerzustand" ("Papierkorbzustand") einführen, d.h. einen Zustand, der kein Endzustand ist und nur mit Zustandsübergängen zu sich selbst versehen ist. (Es kann auch mehrere solche Fehlerzustände geben.)



- Bei komplexeren Automaten werden in praktischen Anwendungen Fehlerzustände im Übergangsdiagramm oft nicht dargestellt, um das Diagramm übersichtlich zu halten. Gibt es zu einem Zustand für ein Zeichen keinen Nachfolger, ist dies dann so zu verstehen, dass man in diesem Fall in einen Fehlerzustand gehen würde.

Der Automat oben würde in der vereinfachten Darstellung so aussehen:

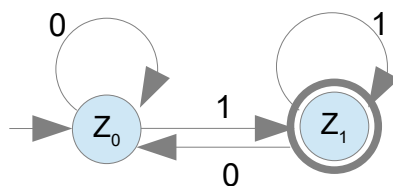


Bei Übungs- und Klausuraufgaben sollten Sie DEAs immer vollständig zeichnen, d.h. mit Fehlerzuständen und Zustandsübergängen für alle Zeichen des Alphabets. Nur wenn es explizit in der Aufgabenstellung erwähnt ist, können die Fehlerzustände weggelassen werden.

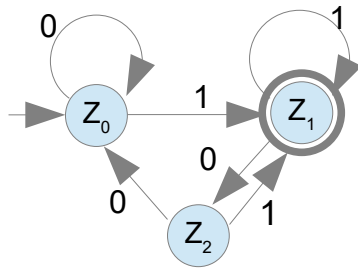
Aufgabe 7.10 - Sprache von DEAs

Welche Sprache wird von folgenden DEAs mit Eingabealphabet $\Sigma = \{0, 1\}$ akzeptiert?

(1) DEA A_1 :



(2) DEA A_2 :



Aufgabe 7.11 - DEAs für Sprache definieren

Gegeben ist das Alphabet $\Sigma = \{a, b\}$. Definieren Sie DEAs, die folgende Sprachen akzeptieren:

- (1) $L_1 = \{ aab, aba \}$
- (2) $L_2 = \{ (ab)^n \mid n > 0 \}$
- (3) $L_3 = \{ w \in \Sigma^* \mid |w|_b \text{ ist ungerade} \}$

7.4 Ausblick: Implementierung von DEAs

Sind die Zustände und Zustandsübergänge eines Automaten bekannt, kann der Automat einfach und systematisch in einer Programmiersprache implementiert werden. Zwei typische Formen der Implementierung werden nachfolgend vorgestellt.

- ▶ **Tabellengesteuerte Implementierung:** Dabei wird die Zustandsübergangstabelle direkt als zweidimensionales Array gespeichert.
- ▶ **Implementierung mittels Fallunterscheidungen:** Dabei wird durch Fallunterscheidungen im Code ausprogrammiert, welcher Nachfolgezustand sich abhängig von aktuellem Zustand und Eingabesymbol ergibt.

Beispiel 7.12 - Tabellengesteuerte Implementierung

Java-Implementierung für den DEA aus *Beispiel 7.9 - Sprache eines DEA*.

- ▶ Die Zustandsübergangstabelle wird hier als zweidimensionales Array `delta` gespeichert.
- ▶ Es gibt eine Variable `zustand`, die den aktuellen Zustand speichert. Abhängig vom nächsten Eingabezeichen wird dann in der Tabelle direkt nachgeschlagen, was der Folgezustand ist.
- ▶ Wenn am Ende des Worts ein Endzustand erreicht worden ist, wird das Wort akzeptiert. Sobald der Fehlerzustand erreicht wird, wird die Prüfung erfolglos beendet.

```

final static int ERROR = -1; // Fehlerzustand

//      Eingabe:  a      b
private static int delta[][] = new int[][]
    { {1,      ERROR}, // Zustand 0
      {ERROR, 2  }, // Zustand 1
      {2,      3  }, // Zustand 2
      {ERROR, ERROR}}; // Zustand 3

private static boolean istEndzustand(int z) {
    return z==1 || z == 3 ;
}

public static boolean akzeptiere1(String wort){
    int pos = 0;
    int zustand = 0;

    while (pos < wort.length()) {
        char zeichen = wort.charAt(pos);
        pos++;

        //Nachfolgezustand aus Tabelle ablesen
        zustand = delta[zustand][zeichen-'a'];

        if (zustand == ERROR) {
            return false; //Wort kann nicht akzeptiert werden
        }
    }

    return istEndzustand(zustand); //Endzustand am Ende erreicht?
}

```

Beispiel 7.13 - Implementierung mittels Fallunterscheidungen

Auch hier wird der DEA aus *Beispiel 7.9 - Sprache eines DEA* implementiert. Wie vorher gibt es eine Variable `zustand` für den aktuellen Zustand des Automaten.

Durch geschachtelte switch-case-Anweisungen werden alle Kombinationen von Zustand und Eingabesymbol abgeprüft, um den Nachfolgezustand bzw. Fehlerfälle zu bestimmen.

```

public static boolean akzeptiere2(String wort){
    int pos = 0;
    int zustand = 0;

    while (pos < wort.length()) {
        char zeichen = wort.charAt(pos);
        pos++;

        switch(zustand) {
            case 0:
                switch(zeichen) {
                    case 'a':
                        zustand = 1; break;
                    default:
                        return false; // Fehler
                }
                break;
            case 1:
                switch(zeichen) {

```

```

        case 'b':
            zustand = 2; break;
        default:
            return false;          // Fehler
    }

    break;
case 2:
    switch(zeichen) {
        case 'a':
            zustand = 2; break;
        case 'b':
            zustand = 3; break;
        default:
            return false;          // Fehler
    }

    break;
case 3:
    switch(zeichen) {
        default:
            return false;          // Fehler
    }
}

return istEndzustand(zustand);
}

```

Anmerkungen

- ▶ Die tabellengesteuerte Implementierung hat den Vorteil, dass sie sehr kompakt ist und Änderungen am Automaten leicht durch Anpassung der Tabelle vorgenommen werden können. Die längere, ausprogrammierte Version mit Fallunterscheidung hat den Vorteil, dass sie noch ein klein wenig schneller ist, wenn es darum geht, das letzte Quäntchen an Geschwindigkeit herauszukitzeln (was z.B. für den Scanner eines Compilers wichtig sein kann).
- ▶ Die Details der Implementierung sind für Theoretische Informatik nicht so wichtig. Was Sie sich aber auf jeden Fall merken sollten:

DEAs können einfach, schematisch und effizient implementiert werden.

- ▶ Da DEAs ganz nach "Schema F" implementiert werden können, wird in Anwendungen der Code für DEAs oft nicht selbst geschrieben, sondern man spart sich diese Fleißarbeit und lässt sich den Code von einem Werkzeug generieren.

Zusammenfassung Lektion 11

Diese Fragen sollten Sie nun beantworten können

- ▶ Was sind deterministische endliche Automaten (DEA)?
- ▶ Wie können DEAs dargestellt werden?
- ▶ Wie ist die akzeptierte Sprache eines DEA definiert?
- ▶ Wie können DEAs implementiert werden?