

# Einführung in die Technische Informatik

## **Inhalt:**

1. Bezug zum Unterricht
2. Schaltwerke und endliche Automaten
  - 2.1 Erstes Beispiel: ein BCD-Ziffern-Erkennen
  - 2.2 Zustandsdiagramme
  - 2.3 Zustandstabellen
  - 2.4 Endliche Automaten mit Ausgabe
  - 2.5 Zweites Beispiel: ein Serienaddierwerk
  - 2.6 Aufgaben
  - 2.7 Die Simulation von Automaten durch Programme
  - 2.8 Beispiel: Simulation des BCD-Ziffern-Erkenners
  - 2.9 Endlichen Automaten und Schaltwerttabellen
  - 2.10 Blockschaltbild des Automaten

## Literaturhinweis:

Vossen/Witt: Grundlagen der Theoretischen Informatik mit Anwendungen, Vieweg  
Seiten 7-26

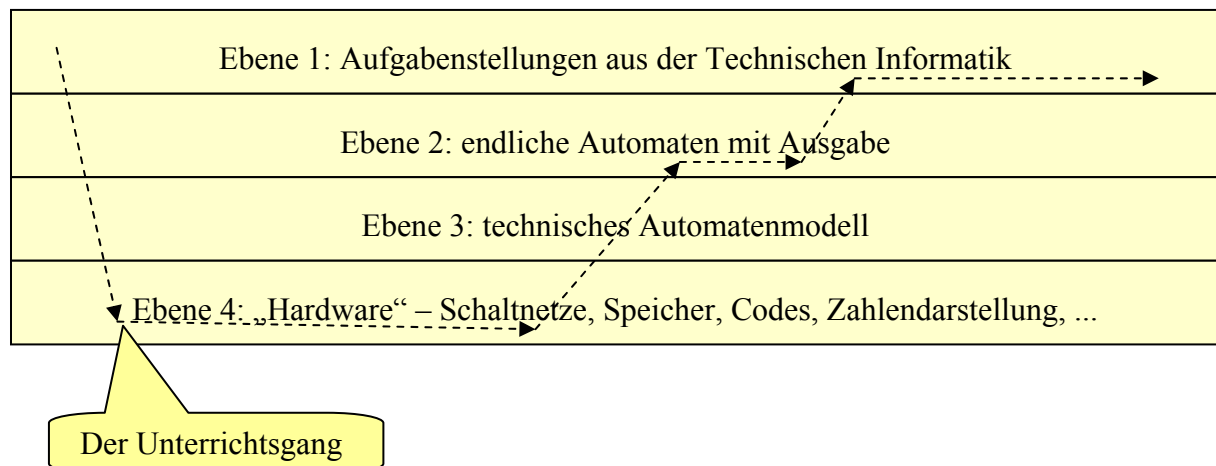
E. Modrow: Technische Informatik mit Delphi, emu-online 2004 ([www.emu-online.de](http://www.emu-online.de))

## 1. Bezug zum Unterricht

Ein Kurs zur technischen Informatik liegt – in der Oberstufe – meist in der Klassenstufe 13. Die teilnehmenden Schüler/innen verfügen zu diesem Zeitpunkt über solide Kenntnisse in der Programmierung und in der Definition und Anwendung von Datenstrukturen – verbunden mit einer konkreten (Modell-)Vorstellung von den programmtechnischen Abläufen bei der Abarbeitung von Programmen. Sie sollten auch einige Erfahrung in selbständiger (Projekt-)Arbeit gewonnen haben. Der Kurs zur technischen Informatik dient m. E. den folgenden Zielen:

- Vermittlung eines soliden Hardware-Computermodells, das das programmtechnische Software-Modell ergänzt und erweitert. („Wie bringt man Schalterkombinationen dazu, programmierbar zu sein?“, „Stack-Maschinen“)
- Anwendungsfall für Datenstrukturen und Algorithmen, insbesondere der OOP. („Entwicklung eines Logiksimulators“).
- Anwendungs- und Vorbereitungsbeispiel für Methoden der theoretischen Informatik („Spart Zeit, die man dann im 4. Semester dringend braucht!“).
- Konkretisierung der fundamentalen Idee des „Zustands“, des „zustandsabhängigen Systems“.
- Erfahrungen vermitteln im Umgang mit „echter“ Technik. („Basteln mit ICs“).

Aus unterrichtspraktischen Gründen hat es sich bewährt, relativ schnell über das Modell der endlichen Automaten zu deren Realisierung als Schaltwerke zu kommen und daraus eine Aufgabenliste zur Entwicklung von Schaltnetzen und Schaltwerken abzuleiten. Nachdem mit diesen traditionellen Themen der Technischen Informatik einige Erfahrungen gemacht wurden, können dann mithilfe der neu gewonnenen Kenntnisse programmierbare Schaltwerke selbst entwickelt und gebaut werden.

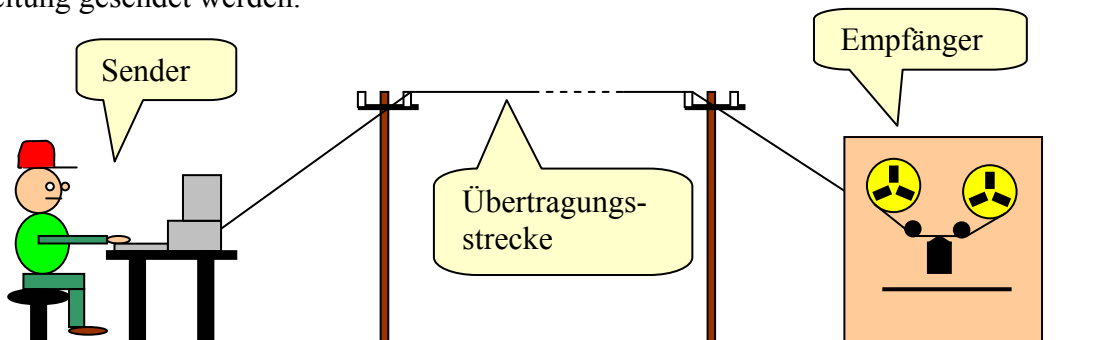


## 2. Schaltwerke und endliche Automaten<sup>1</sup>

In diesem Abschnitt werden einige grundlegende Begriffe der Automatentheorie eingeführt und ihre Bedeutungen für die Beschreibung von Schaltwerken erläutert. Daraus ergeben sich Aufgaben für den Kurs über Technische Informatik.

### 2.1 Erstes Beispiel: ein BCD-Ziffern-Erkenner

Als einführendes Beispiel wählen wir eine Datenübertragungsstrecke zwischen zwei Geräten, auf der natürliche Zahlen als *Binär-Codierte-Dezimal-Ziffern* (BCD-Ziffern) auf einer einzigen Leitung gesendet werden.



Was sind BCD-Ziffern?

Da es zehn verschiedene Dezimalziffern gibt (0...9), benötigt man für ihre Darstellung im Dualsystem vierstellige Dualzahlen. Mit vier Dualziffern lassen sich jedoch nicht nur die Zahlen 0 bis 9, sondern darüber hinaus noch die Zahlen 10 bis 15 beschreiben. Diese Kombinationen stellen in unserem Fall keine sinnvollen Informationen dar und sollten von dem Empfänger als Fehler gewertet werden, wenn sie nicht für spezielle Zwecke Verwendung finden. Wir wollen hier die „Tetrade“ 1111 als Zeichen für das Ende der Datenübertragung wählen - also als ein besonderes Steuerzeichen.

Als Sender und Empfänger von BCD-Zahlen kommen nun nicht nur Computer in Frage. Wir wollen deshalb das Problem, durch „Mitlesen“ das Ende der Übertragung und eventuelle Fehler zu erkennen und anzuzeigen, nicht mit Hilfe eines Programms lösen, sondern mithilfe einer Schaltung, die z. B. auf einer Platine in andere Geräte eingebaut wird.

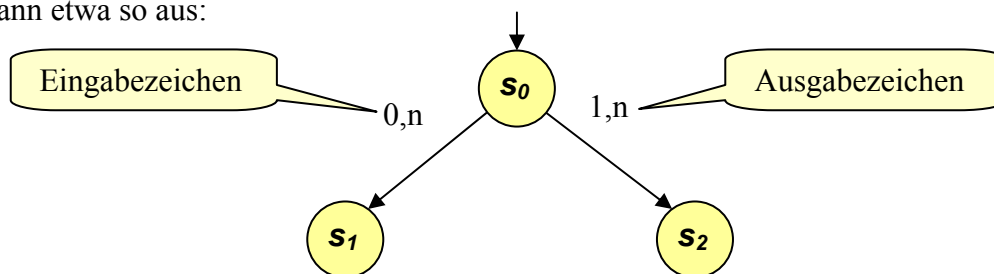
4-stellige Dualzahl	dargestelltes Zeichen
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	Fehler
1011	Fehler
1100	Fehler
1101	Fehler
1110	Fehler
1111	Ende

<sup>1</sup> nach Modrow, Automaten – Schaltwerke – Sprachen, Dümmler 1996

## 2.2 Zustandsdiagramme

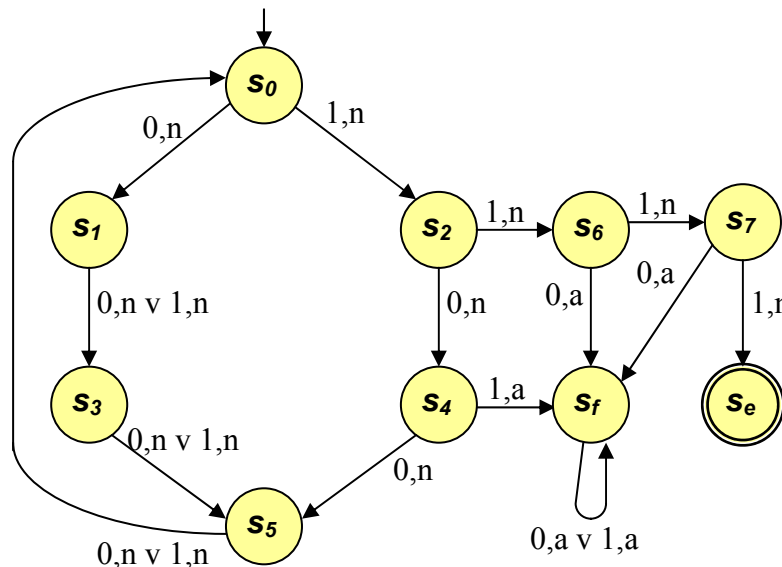
Wir wollen ein Verfahren entwickeln, mit dem wir systematisch Schaltwerke für vorgegebene Zwecke entwickeln können. Dazu müssen wir erst einmal Schaltwerke geeignet beschreiben. Betrachten wir z. B. unser gestelltes Problem genauer von Standpunkt des Empfängers aus, so sehen wir, dass für die Fehler- und Zeichenerkennung die gerade gesendete Dualziffer, ein *Bit* (*binary digit*), nicht ausreicht. Vielmehr hängt die Entscheidung, ob eine zulässige Information vorliegt, vom gerade gesendeten und den vorausgegangenen Bits ab. Die gesuchte Schaltung muss sich also „merken“ können, welche Bits schon gesendet worden sind. Wir können dieses Verhalten so interpretieren, dass die Schaltung von den gelesenen Bits in unterschiedliche *Zustände* überführt wird. Befindet sich die Schaltung in einem bestimmten Zustand, so wird sie von dem nächsten gesendeten Bit in einen von mehreren Folgezuständen überführt. Nach jeweils vier empfangenen Bits kann entschieden werden, ob ein Fehler vorliegt. Andernfalls wird das empfangene Zeichen ausgegeben und in den Anfangszustand zurückgekehrt. Falls die Tetrade *1111* empfangen wurde, geht die Schaltung in einen Endzustand über.

Für nicht zu umfangreiche Probleme bildet die Form des *Zustandsdiagramms* oder *Transitionsgraphen* eine übersichtliche und für das weitere Vorgehen hilfreiche Art, dieses Verhalten aufzuschreiben (und damit exakt die Arbeitsweise der Schaltung festzulegen). Jeder Zustand wird benannt, durch einen Kreis symbolisiert und bildet so einen *Knoten* des Zustandsdiagramms. Der *Anfangszustand* wird mit einem kleinen Pfeil zusätzlich gekennzeichnet. *Endzustände* erkennt man an einem Doppelkreis. Von jedem Zustand aus führen Pfeile - die *Kanten* des Transitionsgraphen - zu den möglichen Folgezuständen. Diese Kanten werden jeweils mit den Eingabezeichen beschriftet, die den entsprechenden Übergang auslösen. Hierbei sind Mehrfachnennungen möglich, falls unterschiedliche Eingabezeichen das System in denselben Folgezustand überführen (die Zeichen werden dann mit „*v*“ (oder) verknüpft). Produziert das System auch Ausgabezeichen, so müssen diese ebenfalls an der entsprechenden Kante neben dem Eingabezeichen notiert werden. Nennen wir beispielsweise den Anfangszustand unseres Schaltwerks  $s_0$  (*s* von *state*), so müssen von diesem Zustand genau zwei Kanten ausgehen, da es nur zwei verschiedene Eingabezeichen gibt - die Binärziffern *0* und *1*. Die entsprechenden Folgezustände wollen wir mit  $s_1$  und  $s_2$  bezeichnen. Da nach dem ersten Bit noch keine vernünftige Ausgabe erfolgen kann, bei diesem Automatentyp aber in jedem Fall eine Ausgabe erfolgen muss, nennen wir diese Ausgabe „*n*“ (nichts). Dieser Teil des Zustandsdiagramms sieht dann etwa so aus:



Unser gesuchter BCD-Ziffern-Erkennen lässt sich damit vollständig beschreiben, wenn wir für den Fehlerfall noch die Ausgabe „*a*“ (Alarm) einführen und die Zustände beliebig durchnummerieren. Den Fehlerzustand nennen wir  $s_f$ . Von ihm führt keine Kante fort, damit die als falsch erkannte Datenübermittlung dauerhaft durch das Alarmzeichen angezeigt wird. Der Übergang zum Endzustand  $s_e$  soll mit der Ausgabe des Zeichens „*e*“ (Ende) verbunden sein, damit auf diese Reaktion hin z. B. die Leitung zur Kostenersparnis automatisch abgeschaltet werden kann. Treten weder Fehler noch das Endezeichen auf, so erfolgt keine Ausgabe, da der Datenübermittlungsprozess nicht gestört werden soll.

An der oben angegebenen Tabelle sieht man, dass kein Fehler auftreten kann, wenn eine Tetrade mit einer 0 beginnt. In diesem Fall müssen wir nur „weiterzählen“, um feststellen zu können, wann die nächste Tetrade beginnt. Bei einer führenden 1 muss noch festgestellt werden, ob eine 8 oder 9 gesendet wird, oder ob es sich um Fehler bzw. Endezeichen handelt.



### 2.3 Zustandstabellen

Auch wenn die schrittweise Erstellung eines Graphen eine bewährte Arbeitsform ist, so nähert sich schon im eben beschriebenen Fall die Darstellung durch ein Zustandsdiagramm seinen Grenzen. Für noch kompliziertere Systeme ähnelt der Transitionsgraph leicht einem „Spaghettihaufen“. Es bietet sich deshalb eine andere Darstellungsform an: eine Tabelle. Sie hat nebenbei den Vorteil, dass sie für Programme wesentlich leichter auswertbar ist als ein Graph.

In eine Zustandstabelle werden in der Kopfzeile horizontal alle möglichen Eingabezeichen (die Elemente des **Eingabealphabets**) eingetragen. Links stehen untereinander alle Zustände des Systems. In die so entstehende Tabelle werden an den Kreuzungspunkten von Zeilen (die einem Ausgangszustand entsprechen) und Spalten (die einem Eingabezeichen entsprechen) die sich daraus ergebenden Folgezustände und Ausgabezeichen eingetragen. Endzustände erkennt man daran, dass in den entsprechenden Zeilen der Tabelle keine Einträge stehen, denn der Automat arbeitet nach Erreichung eines Endzustandes nicht mehr weiter.

	0	1	Eingabezeichen
S <sub>0</sub>	S <sub>1</sub> ,n	S <sub>2</sub> ,n	
S <sub>1</sub>	S <sub>3</sub> ,n	S <sub>3</sub> ,n	
S <sub>2</sub>	S <sub>4</sub> ,n	S <sub>6</sub> ,n	
S <sub>3</sub>	S <sub>5</sub> ,n	S <sub>5</sub> ,n	
S <sub>4</sub>	S <sub>5</sub> ,n	S <sub>f</sub> ,a	Folgezustand, Ausgabezeichen
S <sub>5</sub>	S <sub>0</sub> ,n	S <sub>0</sub> ,n	
S <sub>6</sub>	S <sub>f</sub> ,a	S <sub>7</sub> ,n	
S <sub>7</sub>	S <sub>f</sub> ,a	S <sub>e</sub> ,e	
S <sub>f</sub>	S <sub>f</sub> ,a	S <sub>f</sub> ,a	
S <sub>e</sub>	-	-	

Ausgangszustand

Eingabezeichen

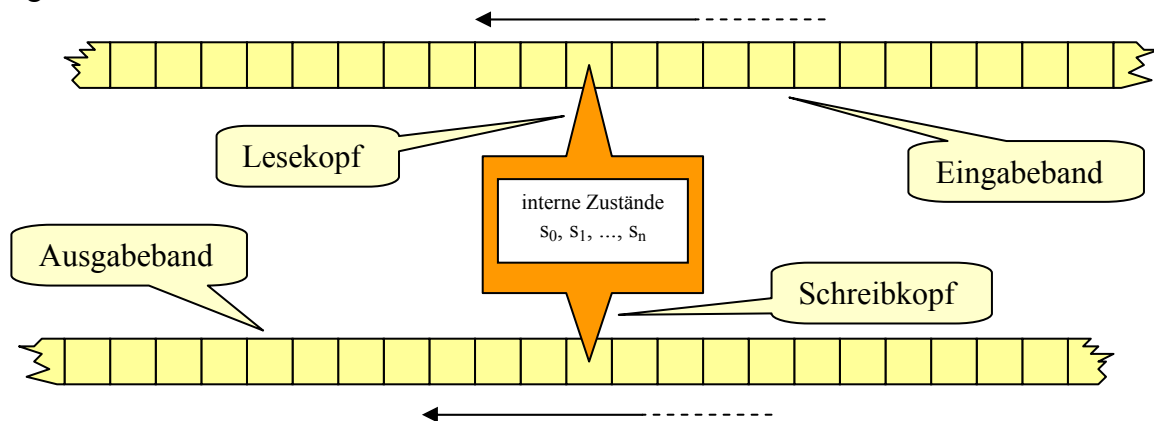
Folgezustand, Ausgabezeichen

## 2.4 Endliche Automaten mit Ausgabe

Fasst man die bisherigen Ergebnisse zusammen und abstrahiert von der Beschreibung des gesuchten Schaltwerks allgemein zu **zustandsabhängigen Systemen**, so kommt man zum Begriff des **endlichen Automaten**. Diese Modellmaschinen dienen dazu, Systeme zu beschreiben, die als Reaktion auf eine Eingabe eine (möglicherweise leere) Ausgabe produzieren, die auch noch vom momentanen Zustand des Systems abhängt. Die Automaten sind endlich, wenn sowohl die Menge der möglichen Eingabezeichen (das **Eingabealphabet  $E$** ), die Menge der möglichen Ausgabezeichen (das **Ausgabealphabet  $A$** ) und die **Zustandsmenge  $S$**  endlich sind. Typische Vertreter solcher Systeme sind Schaltwerke. Es lässt sich sogar jeder endliche Automat in ein Schaltwerk umsetzen, wie wir noch sehen werden. (Obwohl das nicht immer sinnvoll ist).

Als Modell für alle diese Systeme wählen wir die folgende Beschreibung:

Der Automat bestehe aus einer Maschine, die verschiedene interne Zustände einnehmen kann. Er verfüge darüber hinaus über einen Lesekopf, an dem ein Eingabeband in nur einer Richtung vorbeigeführt wird (wie bei einem Tonband oder einem Lochstreifenleser). Auf dem Eingabeband stehen nacheinander die Eingabezeichen, die in beliebiger Folge aus dem Eingabealphabet gewählt werden können. In Abhängigkeit vom zuletzt gelesenen Eingabezeichen und dem momentanen Zustand geht der Automat in einen Folgezustand über, nachdem er ein Ausgabezeichen, das zum Ausgabealphabet gehört, produziert hat. Dieses Ausgabezeichen wird von einem Schreibkopf auf ein Ausgabeband geschrieben, das sich ebenfalls nur in einer Richtung am Schreibkopf des Automaten vorbeibewegt, also nach jedem Schreibvorgang um eine Position weiterrückt.



Die Reaktion des Automaten auf das jeweilige Eingabezeichen wird präzise durch zwei Funktionen festgelegt: Die **Überföhrungsfunktion  $u$**  bestimmt aus Eingabezeichen und Zustand den Folgezustand. Die **Ausgabefunktion  $g$**  bestimmt aus Eingabezeichen und Zustand das Ausgabezeichen. Beide Funktionen sind sowohl im Zustandsdiagramm wie in der Zustandstabelle enthalten.

Ein endlicher Automaten  $M = (E, A, S, s_0, \Sigma, u, g)$  wird durch folgende Größen beschrieben:

- $E = \{e_1, e_2, \dots, e_r\}$  das Eingabealphabet
- $A = \{a_1, a_2, \dots, a_m\}$  das Ausgabealphabet
- $S = \{s_0, s_1, \dots, s_n\}$  die Zustandsmenge
- $s_0 \in S$  den Anfangszustand
- $\Sigma \in S$  die Menge der Endzustände
- $u: (e_i, s_j) \rightarrow s_k$  die Überföhrungsfunktion, mit  $(1 < i < r, 0 < j < n, 1 < k < n)$
- $g: (e_i, s_j) \rightarrow a_l$  die Ausgabefunktion, mit  $(1 < i < r, 0 < j < n, 1 < l < m)$

Alle diese Angaben sind schon im Zustandsdiagramm und der Zustandstabelle enthalten und müssen eigentlich nicht mehr extra herausgeschrieben werden. Wir wollen deshalb die Formalitäten nicht zu wichtig nehmen.

Die Arbeitsweise des Automaten kann übersichtlich in Form eines Struktogramms beschrieben werden:

<i>M in den Anfangszustand bringen</i>			
<i>den Lesekopf über das erste Zeichen des Eingabebandes bringen</i>			
<i>den Schreibkopf auf die erste freie Stelle des Ausgabebandes setzen</i>			
<i>ein Eingabezeichen lesen</i>			
<i>SOLANGE noch Eingabezeichen vorhanden sind TUE</i>			
<table border="1"> <tr> <td><i>ein Ausgabezeichen mithilfe der Ausgabefunktion bestimmen und auf das Ausgabeband schreiben</i></td> </tr> <tr> <td><i>den Folgezustand mithilfe der Überföhrungsfunktion bestimmen und in diesen Zustand wechseln</i></td> </tr> <tr> <td><i>ein Eingabezeichen lesen</i></td> </tr> </table>	<i>ein Ausgabezeichen mithilfe der Ausgabefunktion bestimmen und auf das Ausgabeband schreiben</i>	<i>den Folgezustand mithilfe der Überföhrungsfunktion bestimmen und in diesen Zustand wechseln</i>	<i>ein Eingabezeichen lesen</i>
<i>ein Ausgabezeichen mithilfe der Ausgabefunktion bestimmen und auf das Ausgabeband schreiben</i>			
<i>den Folgezustand mithilfe der Überföhrungsfunktion bestimmen und in diesen Zustand wechseln</i>			
<i>ein Eingabezeichen lesen</i>			

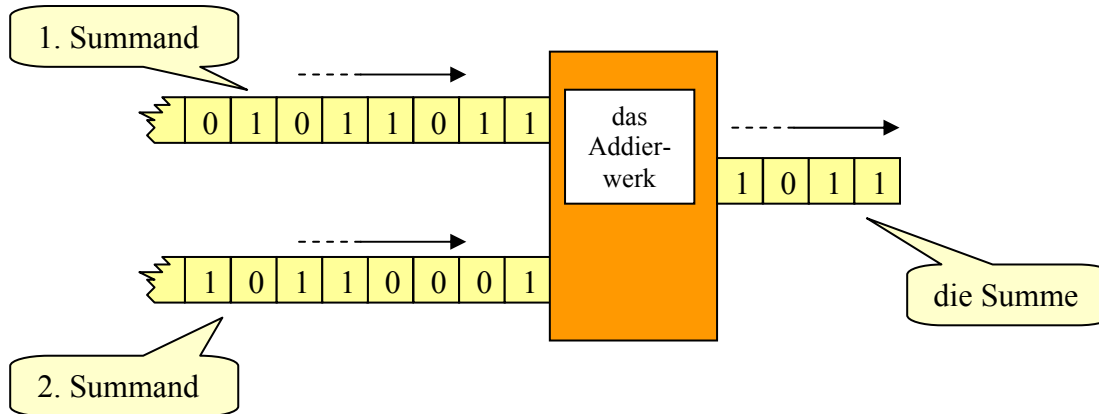
Etwas formaler geschrieben lautet das Struktogramm:

$s \leftarrow s_0$		
<i>lies(e)</i>		
<i>SOLANGE NICHT(s IN sigma) TUE</i>		
<table border="1"> <tr> <td><i>schreibe(g(s,e))</i></td> </tr> <tr> <td><math>s \leftarrow u(s,e)</math></td> </tr> </table>	<i>schreibe(g(s,e))</i>	$s \leftarrow u(s,e)$
<i>schreibe(g(s,e))</i>		
$s \leftarrow u(s,e)$		

In dieser Form kann es direkt in ein lauffähiges Programm übersetzt werden (s. u.). Das Automatenmodell führt also nicht nur zu Schaltwerken, sondern auch zu Programmen.

## 2.5 Zweites Beispiel: ein Serienaddierwerk

Bevor wir weiter auf die Entwicklung von Schaltwerken eingehen, wollen wir zur Übung noch ein anderes, sehr einfaches Beispiel betrachten: gesucht ist ein Serienaddierwerk. Es soll also ein Automat entwickelt werden, der zwei Dualzahlen stellenweise, beginnend mit den niederwertigsten Bits, addiert. Dabei werden die jeweils an dem Eingang des Automaten anliegenden beiden Dualziffern zum nächsten Bit des Ergebnisses zusammengefasst.



Die Addition von Dualzahlen erfolgt nun genauso wie die von Dezimalzahlen. Beginnend mit den am weitesten rechts stehenden Ziffern - den niederwertigsten Stellen - werden jeweils zwei Ziffern addiert, der im Zahlenbereich liegende Teil des Ergebnisses - die Einerziffer - notiert und der Übertrag zusammen mit der nächsten Stelle weiterverarbeitet.

$$\begin{array}{r}
 91 \quad 01011011 \\
 + 86 \quad + 01010110 \\
 \hline
 177 \quad \underline{\underline{10110001}}
 \end{array}$$

Ebenso wie bei dem BCD-Ziffern-Erkennen hängt auch hier das Ergebnis nicht nur von den beiden gerade zu addierenden Ziffern ab, sondern auch von einem Teilergebnis der vorhergehenden Addition - dem Übertrag. Da weitere zu merkende Größen nicht auftreten, können wir unseren Addierer durch einen Automaten beschreiben, der mit zwei Zuständen auskommt:

$s_0$ : eine Null als Übertrag gespeichert (der Anfangszustand)

$s_1$ : eine Eins als Übertrag gespeichert

Damit erhalten wir die Zustandsmenge  $S = \{s_0, s_1\}$  mit dem Anfangszustand  $s_0$ .

Wenden wir uns dem Eingabealphabet zu. In unserem Modell liest der Automat immer nur ein einziges Zeichen, unser Addierer dagegen erhält immer zwei Dualziffern als Eingabe. Wir müssen also die beiden Dualziffern als ein einziges Zeichen interpretieren! Das ist auch leicht möglich, wenn wir den vier auftretenden Kombinationen (00, 01, 10 und 11) an den Eingängen des Addierers vier einzelne Zeichen zuordnen. Dabei ist diese Zuordnung völlig willkürlich, z. B.:

- 00 <---> N (Null)
- 01 <---> E (Eins)
- 10 <---> A (Auch Eins)
- 11 <---> Z (Zwei)

Wir finden damit das Eingabealphabet  $E = \{N, E, A, Z\}$

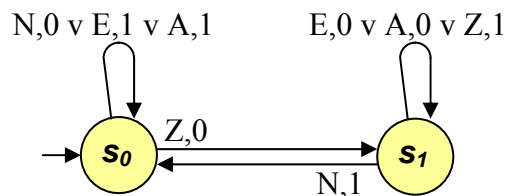
Die Arbeitsweise des Automaten ergibt sich, indem man die Anzahl der Einsen, die am Eingang des Automaten liegen, mit dem gespeicherten Wert des Übertrags (der sich aus dem Zustand ergibt) verarbeitet. Der Automat bleibt im Anfangszustand, wenn er die Zeichen N (00),

E (01) oder A (10) liest, da dann kein Übertrag bei der Addition auftritt. Entsprechend bleibt er im Zustand  $s_1$ , wenn er ein E, A oder Z liest, da diese Eingaben zusammen mit dem Übertrag zu einem neuen Übertrag führen.

Der Automat wechselt von  $s_0$  zu  $s_1$ , wenn er das Zeichen Z (11) liest und beim Zeichen N (00) in umgekehrter Richtung. Ein ausgezeichneter Endzustand ist aus der Aufgabenstellung nicht abzuleiten, da nach unserer Beschreibung ein Ende der Addition nicht angegeben wurde.

Das Ausgabealphabet besteht einfach aus den beiden Dualziffern.  $A = \{0, 1\}$

Das Zustandsdiagramm:



Da nur dann eine Ausgabe erfolgen kann, wenn vorher eine Eingabe vorgenommen wurde, kann bei  $n$  Eingabezeichen auch die Ausgabe nur  $n$  Zeichen umfassen. Die Eingabe sollte deshalb immer mit einer (führenden) Null abgeschlossen werden, damit der letzte für das Ergebnis zählende Übertrag auch noch ausgegeben wird.

## 2.6 Aufgaben

- Übersetzen Sie das Zustandsdiagramm des *Serienaddierers* in eine Zustandstabelle.
- Beschreiben Sie die Steuerung einer *Verkehrsampel* als endlichen Automaten. Eingabezeichen sind die Signale *F* (Farbe halten) und *W* (Farbe wechseln). Ausgabezeichen sind die Farbkombinationen der Ampel: *rot*, *rot-gelb*, *grün* und *gelb*, die dem zuletzt eingenommenen Zustand entsprechen sollen.
  - Erweitern Sie die Ampel um eine *Bedarfssteuerung*. Das Eingabezeichen *R* (verlängerte Rotphase) bewirke, dass die Rotphasen der Ampel doppelt so lang sind wie die Grünphasen. Beim Eingabezeichen *G* (verlängerte Grünphase) reagiere die Anlage entsprechend. Lassen Sie die Eingabezeichen *F* und *W* aus 2.a weg (die Ampel enthalte jetzt einen automatischen Takt).
- Beschreiben Sie einen einstelligen *Dualtaschenrechner* als endlichen Automaten. Der Rechner verfüge über die Grundrechenarten Addition und Multiplikation, sei aber in seinem Zahlenbereich extrem eingeschränkt: eingegeben werden nur die Dualzahlen *0* und *1*, als Ergebnis zur weiteren Verarbeitung werde nur die „Einerziffer“ des dualen Ergebnisses ausgegeben - also ebenfalls nur *0* oder *1*. Z. B. ergibt die Rechnung *1+1* eine *0*, da die führende *1* des Ergebnisses *2* (dual *10*) weggelassen wird. Einen Fehlerzustand kenne der Automat nicht. Erfolgen Eingaben, die in ihrer Reihenfolge keinen Sinn ergeben, so überschreibe die letzte Eingabe die vorhergehende: z. B. werde die Eingabefolge „*1+x10*“ als „*1x0*“ interpretiert.

4. Beschreiben Sie eine Fahrstuhlsteuerung als endlichen Automaten. Angefahren werden die drei Stockwerke  $E$  (Erdgeschoß),  $B$  (Beletage) und  $D$  (Dachgeschoß). Als Eingaben erhält das System die Anforderungssignale aus den drei Stockwerken als dreistellige Dualzahlen. Höchste Priorität hat immer die Anforderung aus dem Stockwerk, in dem sich der Fahrstuhl gerade befindet. Liegen Anforderungen aus einem höheren und einem niedrigeren Geschoß vor, so hat die Anforderung Priorität, die in der momentanen Fahrtrichtung des Fahrstuhls liegt. Steht der Fahrstuhl, so hat das Erdgeschoß Priorität. Nach Erreichen des nächsten Stockwerks hält der Fahrstuhl und die Anlage überprüft erneut die Anforderungen.
5. Entwickeln Sie einen *Prüfbitgenerator*. Da die meisten Zeichensätze weniger als 128 Zeichen enthalten, können sie als siebenstelligen Dualzahlen codiert werden. Rechner arbeiten aber mit Bytes, achtstelligen Dualdarstellungen von Zeichen. Das eigentlich überflüssige achte Bit kann als ein Prüfbit benutzt werden, das bei der Datenübertragung zur Fehlerermittlung benutzt wird. Eine Möglichkeit Prüfbits zu erzeugen ist die, an das zu übertragende Zeichen jeweils eine 1 oder 0 so anzuhängen, dass das gesendete Byte immer eine gerade (ungerade) Anzahl von Einsen enthält. Beschreiben Sie einen solchen Prüfbitgenerator als endlichen Automaten, der nacheinander 7 Bits liest, sie wieder ausgibt und an das achte Bit automatisch das Prüfbit anhängt.
6. a: Entwickeln Sie das Zustandsdiagramm eines Getränkeautomaten mit nur einer Sorte Fruchtsaft. Eingabezeichen sind  $G$  (Geld),  $W$  (Wahl) und  $R$  (Geld-Rückgabeknopf). Ausgegeben werden  $S$  (Saft),  $G$  (Geld) oder  $N$  (nichts).  
b: Erweitern Sie den Automaten aus 6.a so, dass er zwar immer noch nur eine Sorte Saft ausgibt, dafür aber sowohl 50-Cent- wie Eurostücke annimmt. (Der Preis des Getränkes betrage 1,50 €). Überzahlte Beträge werden einbehalten.  
c: Erweitern Sie den Automaten aus 6.a so, dass eine Wahl zwischen mehreren Sorten Saft möglich wird.
7. Behandeln Sie das Problem eines Kaffeeautomaten entsprechend Aufgabe 6. (Es gibt Kaffee mit und ohne Zucker und Milch, als Espresso etc.)
8. Entwickeln Sie eine *Zählschaltung modulo 3* (also einen Zähler, der bis zwei zählen kann: 0-1-2-0-1-2-0-...). Zusätzlich soll der Zähler durch die Eingabe von RESET in den Anfangszustand zurückgesetzt werden können. Der Zähler ändert seinen Zustand,
  - a: wenn ein Taktsignal  $T$  eintrifft.
  - b: wenn die Eingabe von 1 auf 0 wechselt.
9. Gesucht ist eine Sortierschaltung, die zwei Dualzahlen, die an den Eingängen  $E_1$  und  $E_2$ , beginnend mit der höchstwertigen Ziffer, anliegen, der Größe nach geordnet an den Ausgängen  $A_1$  und  $A_2$  ausgibt, also: Liegt am Eingang  $E_1$  die größere Zahl, dann wird diese am Ausgang  $A_1$  ausgegeben, sonst die andere Zahl.

## 2.7 Die Simulation von Automaten durch Programme

Das aus 2.4 bekannte Struktogramm zur Beschreibung der Arbeitsweise eines endlichen Automaten kann direkt in ein Java-Programm übersetzt werden, wenn die beiden Funktionen  $u(s,e)$  und  $g(s,e)$  entsprechend angepasst werden. Da Java keine Aufzählungstypen kennt, bezeichnen wir hier die Zustände der Einfachheit halber mit ganzen Zahlen: der Zustand  $s_0$  erhält den Namen  $,0'$ ,  $s_f$  den Namen  $,1'$  usw. Den Fehlerzustand  $s_f$  nennen wir  $,99'$  und den Endzustand  $s_e$ ,  $,100'$ .

Das Ein- und das Ausgabeband des Automaten wollen wir durch Zeichenketten (Strings) realisieren, die in zwei Textfeldern namens *tfEingabe* und *tfAusgabe* dargestellt werden. Der Takt wird durch das Anklicken eines Buttons gegeben.

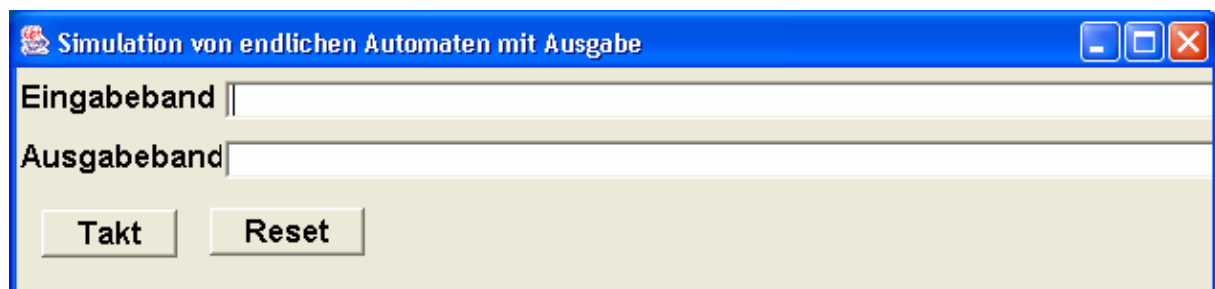
Zu Beginn muss der Automat in den Anfangszustand gebracht werden. Und da das erste Zeichen der Strings den Index  $,0'$  hat, setzen wir die Zählvariable  $i$ , die das zu verarbeitende Zeichen referenziert, anfangs auf  $,-1'$ . Außerdem sollten die Editierfelder leer sein. Diese Aufgaben können von einer Methode *neu()* erledigt werden.

```
private void neu()
{
    s = 0;
    i = -1;
    tfEingabe.setText("");
    tfAusgabe.setText("");
}
```

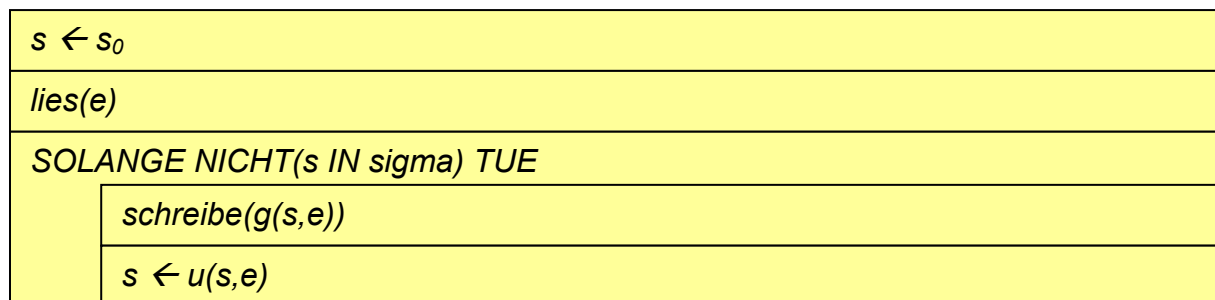
Drücken wir den Button *bReset*, dann wird diese Methode aufgerufen:

```
void bReset_actionPerformed(ActionEvent e)
{
    neu();
}
```

Unsere Oberfläche ist damit fertig. Das Programmfenster zieht so aus:



Betrachten wir noch einmal das Struktogramm:



Da die „Schleifendurchläufe“ bei uns durch mehrfaches Anklicken des *Takt*-Buttons ersetzt werden sollen, muss in dessen Eventhandler alles Erforderliche veranlasst werden.

```
void bTakt_actionPerformed(ActionEvent e)
{
    i++;
    if( i < tfEingabe.getText().length() )
    {
        rein = tfEingabe.getText().charAt(i);
        raus = g(s, rein);
        tfAusgabe.setText(tfAusgabe.getText()+raus);
        s = u(s, rein);
    }
}
```

Katastrophen  
verhindern

nächstes Zei-  
chen holen ...

... und verarbeiten.

Bei den Variablen *rein* und *raus* handelt es sich natürlich um Zeichen.

Mit dieser Methode lassen sich alle endlichen Automaten mit Ausgabe simulieren, denn die Unterschiede liegen (bei diesem Modell) alleine in den Funktionen. Die müssen wir jetzt schreiben.

Am Einfachsten ist es, für jeden Zustand und für jedes Eingabezeichen aufzuzählen, was zu tun ist. So lässt sich der Transitionsgraph einfach „abschreiben“.

Die Überföhrungsfunktion berechnet aus Zustand und Eingabezeichen einen Folgezustand - also eine ganze Zahl:

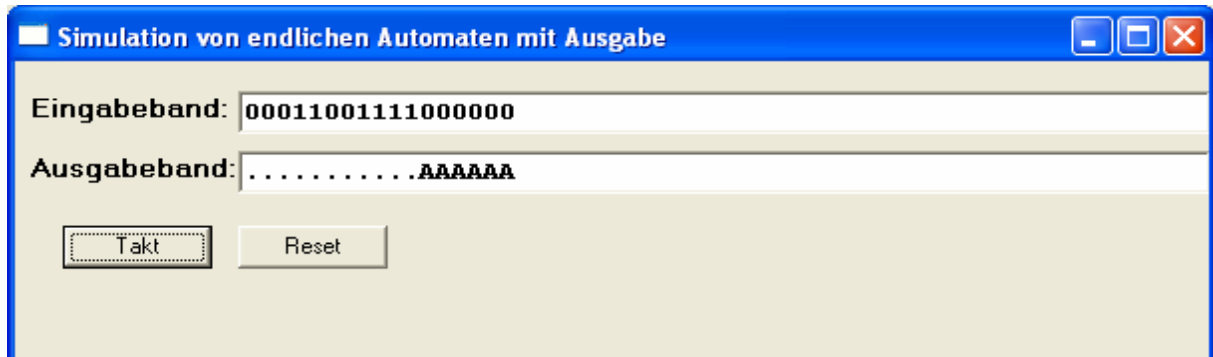
```
public int u(int s, char e)
{
    switch(s)
    {
        case 0: switch(e)
        {
            case ...: return ...;
            case ...: return ...;
            ...
        }
        case 1: switch(e)
        {
            case ...: return ...;
            case ...: return ...;
            ...
        }
        ...
    }
}
```

Entsprechend berechnet die Ausgabefunktion ein Zeichen:

```
public char u(int s, char e)
{
    switch(s)
    {
        case 0: switch(e)
        {
            case ...: return ...;
            case ...: return ...;
        }
        ...
    }
}
```

## 2.8 Beispiel: Simulation des BCD-Ziffernerkenners

Für diesen brauchen wir jetzt nur noch die beiden Funktionen anzugeben. Da es nur zwei Eingabezeichen gibt, arbeiten wir bei deren Unterscheidung mit der Alternative (*if*) statt mit der Mehrfachauswahl (*switch*):



```

...
int i=-1,s=0;
char rein,raus;
...

public int u(int s, char e)
{
    switch(s)
    {
        case 0: if(e=='0') return 1; else return 2;
        case 1: return 3;
        case 2: if(e=='0') return 4; else return 6;
        case 3: return 5;
        case 4: if(e=='0') return 5; else return 99;
        case 5: return 0;
        case 6: if(e=='0') return 99; else return 7;
        case 7: if(e=='0') return 99; else return 100;
        case 100: return 100;
        default: return 99;
    }
}

public char g(int s, char e)
{
    switch(s)
    {
        case 0: return '.';
        case 1: return '.';
        case 2: return '.';
        case 3: return '.';
        case 4: if(e=='0') return '.'; else return 'A';
        case 5: return '.';
        case 6: if(e=='0') return 'A'; else return '.';
        case 7: if(e=='0') return 'A'; else return 'E';
        case 99: return 'A';
        case 100: return 'E';
        default: return 'A';
    }
}
...

```

## 2.9 Endliche Automaten und Schaltwerttabellen

Wir können die Beschreibung von Schaltwerken durch endliche Automaten als Erzeugung von *Modellen* des gesuchten Systems auffassen. Die Simulation durch ein Programm entspricht dann dem *Test des Modells*. Nach diesem Schritt sollen wir einigermaßen sicher sein, dass unser Modell das Gewünschte leistet. Wir können dann auf dem Weg zur echten Technik fortschreiten.

Da wir ein digitales System erzeugen wollen, benötigen wir eine Umsetzung der den Automaten beschreibenden Größen (Eingabealphabet, ...) in eine digitale Darstellung. Wir *codieren* diese Größen.

### 1. Schritt: Codierung

Beschreiben müssen wir die Eingabezeichen, die Ausgabezeichen und die Zustände des Automaten. Als digitale Zeichen stehen uns nur die „0“ und die „1“ zur Verfügung. Wie kommen wir zu einer geeigneten Codierung?

Als Beispiel wollen wir ein Eingabealphabet codieren, das drei unterschiedliche Zeichen enthält. Um diese unterscheiden zu können, benötigen wir zwei Bits, da diese in vier unterschiedlichen Kombinationen (also einer zu viel) auftreten können. Man kann mit zwei Bits also maximal vier Zeichen codieren. Haben wir mehr Größen zu codieren, dann benötigen wir entsprechend mehr Bits. Bezeichnen wir mit  $P_2(n)$  die Zweierpotenz (also 2, 4, 8, ...), die größer oder gleich  $n$  ist, dann ergibt sich Anzahl  $m$  der benötigten Bits aus  $m = \text{ld}(P_2(n))$ . (In unserem Fall:  $P_2(3) = 4 \rightarrow m = \text{ld}(4) = 2$ )

Als Bezeichnungen für die Bits hat sich eingebürgert:

- die Bits der Eingabezeichen werden mit  $e_0, e_1, \dots$  bezeichnet
- die Bits der Ausgabezeichen werden mit  $a_0, a_1, \dots$  bezeichnet
- die Bits der Zustände werden mit  $z_0, z_1, \dots$  bezeichnet

Wie wir die Zuordnung zwischen Größen und Bitkombination wählen, ist eigentlich egal. Es bewährt sich allerdings meist, möglichst viele Nullen zu benutzen und „ähnliche“ Größen (z. B. Ein- und Ausgabezeichen oder bestimmte Zustände und Ausgabezeichen) auch „ähnlich“ zu codieren, damit die gefundenen Schaltungen sich dann auch „ähneln“, also relativ einfach werden, weil Teile mehrfach zu verwenden sind.

Unser Automat „berechnet“ aus der Kombination von aktuellem Zustand und dem zuletzt gelesenen Eingabezeichen des Eingabebands den Folgezustand und das nächste Ausgabezeichen. Diese Zuordnungen beschreiben wir durch die Überföhrungs- und die Ausgabefunktion. In der digitalen Version muss also die Überföhrungsfunktion alle  $n$  Bits des nächsten Zustands berechnen. Damit besteht sie aus  $n$  „Teilfunktionen“, die jeweils das entsprechende Bit liefern. Entsprechendes gilt für die Ausgabefunktion:

- die digitalen Überföhrungsfunktionen werden mit  $u_0, u_1, \dots$  bezeichnet.
- die digitalen Ausgabefunktionen werden mit  $g_0, g_1, \dots$  bezeichnet.

Liefert eine Überföhrungsfunktion in einem Fall z. B.  $u(s_1, e_3) = s_2$  und haben wir die Zustände und Zeichen einfach durch die entsprechenden Dualzahlen codiert, dann lautet die digitale Version dieses Falls  $u(01, 11) = 10$ . Da bei endlichen Automaten alle Mengen endlich sind (sic!), gibt es nur endlich viele Kombinationen aus Zustand und Eingabezeichen.

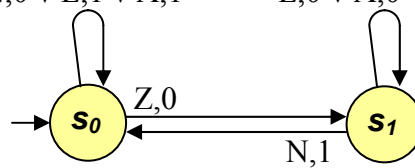
**Das nutzen wir jetzt schamlos aus!**

## 2. Schritt: Aufstellen der Schaltwerttabelle

Schreiben wir alle Kombinationen aus codierten Zuständen und codierten Eingabezeichen zusammen mit den codierten Folgezuständen und Ausgabezeichen in Form einer Tabelle auf, dann haben wir eine vollständige digitale Beschreibung des Automaten gefunden.

Zustand				Eingabezeichen			Folgezustand				Ausgabezeichen		
z <sub>0</sub>	z <sub>1</sub>	z <sub>2</sub>	...	e <sub>0</sub>	e <sub>1</sub>	...	u <sub>0</sub>	u <sub>1</sub>	u <sub>2</sub>	...	g <sub>0</sub>	g <sub>1</sub>	...

Als Beispiel wollen wir den Serienaddierer  $N,0 \vee E,1 \vee A,1$   $E,0 \vee A,0 \vee Z,1$  so beschreiben:



*Codierung der Mengen:*

Eingabezeichen	e <sub>0</sub>	e <sub>1</sub>
N	0	0
E	0	1
A	1	0
Z	1	1

Zustand	z
s <sub>0</sub>	0
s <sub>1</sub>	1

Ausgabezeichen	a
0	0
1	0

*Schaltwerttabelle:*

Zustand	Eingabezeichen		Folgezustand	Ausgabezeichen
z	e <sub>0</sub>	e <sub>1</sub>	u	g
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Man sieht schnell, dass die Zeilenzahl n der Tabellenlänge abhängt von

$$n = (\text{Anzahl der Zustände}) \times (\text{Anzahl der Eingabezeichen}).$$

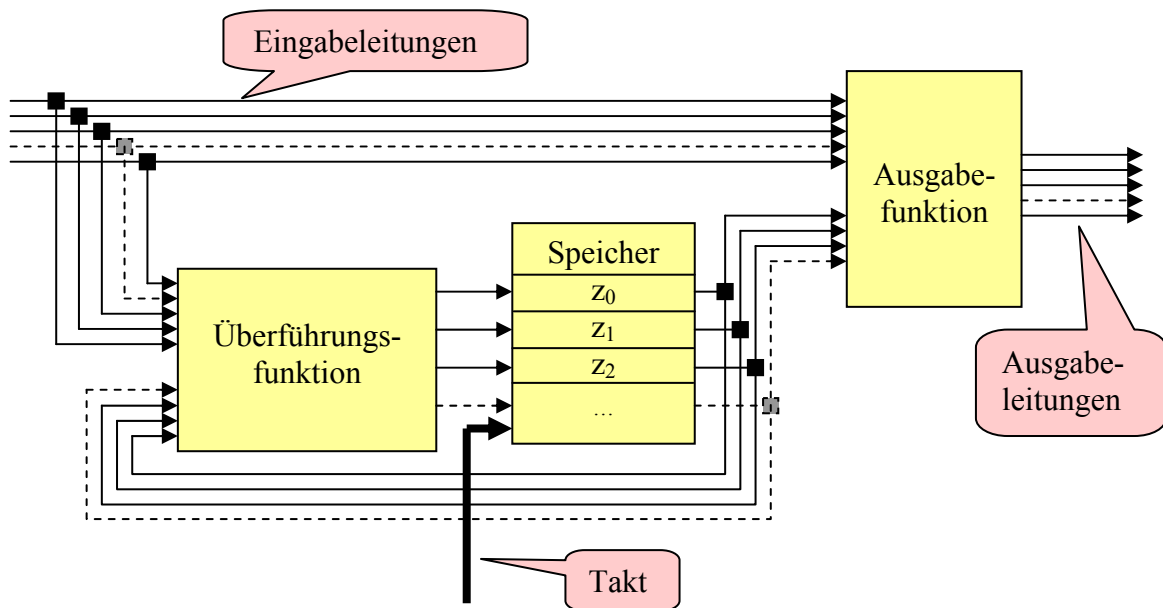
Aus unterrichtspraktischen Gründen ist also die Zahl der Automaten, die sich wirklich in dieser Form noch bearbeiten lassen, beschränkt. Das merkt man ziemlich schnell beim Aufstellen von Klausuren.

## 2.10 Blockschaltbild des Automaten

Fassen wir unsere bisherigen Kenntnisse zusammen:

Ein als Schaltwerk realisierter Automat benötigt

- **Speicher**, um den momentanen Zustand dauerhaft zu repräsentieren. Für jedes Bit des Zustands wird ein eigener Speicher benötigt,
- **Schaltfunktionen**, die aus momentanem Zustand und Eingabezeichen den Folgezustand und das Ausgabezeichen zu bestimmen,
- **Leitungen**, die die Bits des Eingabezeichens der Schaltung zuführen sowie die Bits des Ausgabezeichens aus der Schaltung leiten
- und einen **Takt**, der die Abläufe synchronisiert.



Es ist also unsere Aufgabe, Verfahren zu finden, um entsprechende Bauteile zu finden. Dann können wir systematisch Schaltwerke entwickeln.