Automatentheorie

Endliche Automaten, Kellerautomaten und Turingmaschinen

fachhochschule münchen
munich university of applied sciences

Inhaltsübersicht und Literatur

- Der Begriff des Automaten
- Endliche Automaten mit Ausgabe
- Technische Realisierung von Automaten
- Erkennende endliche Automaten
  - Sprache endlicher Automaten
  - Äquivalenz deterministischer und nicht-deterministischer endlicher Automaten
- Minimierung von endlichen Automaten
- Kellerautomaten
- Turingmaschinen und das Halteproblem

Literatur:

- Uwe Schöning: Theoretische Informatik - kurzgefaßt, 4. Auflage, Spektrum Akademischer Verlag 2001
- Hopcroft/Motwani/Ullman: Introduction to Automata Theory, Languages, and Computation, 2nd edition, Addison-Wesley 2001

Definition eines Automaten

Definition:

Ein abstrakter Automat mit Ausgabe ist ein 5-Tupel

$$A = (X , Y , Z , f_z , f_a)$$

X = Ausprägungsmenge der Eingabevariablen (Eingabealphabet)

Y = Ausprägungsmenge der Ausgabevariablen (Ausgabealphabet)

Z = Ausprägungsmenge der Zustände

f_z = Zustandsfunktion, $f_z : X \times Z \rightarrow Z$

f_a = Ausgabefunktion, $f_a : X \times Z \rightarrow Y$

Der Automat heißt endlich, wenn die Mengen X, Y und Z endlich sind.

Eingaben, Ausgaben und Zustände

Die drei Ausprägungsmengen in der Definition eines Automaten können folgendermaßen interpretiert werden:

1. **Eingabe:**
   - **Ein Automat muß von außen bedient werden können.**

2. **Interne Zustände:**
   - **Jeder Automat befindet sich immer in einem bestimmten Zustand. Unter Einwirkung der Eingaben kann der Automat eine Reihe von Zuständen durchlaufen.**

3. **Ausgabe:**
   - **Im Laufe seiner Arbeit produziert der Automat Informationen, d. h. er gibt Ausgabedaten aus.**

Zustandsfunktion eines Automaten

Die Zustandsfunktion f_z ist eine Abbildung

$$f_z : X \times Z \rightarrow Z$$

der Produktmenge X x Z (kartesisches Produkt) in die Menge Z der Zustände, die wie folgt interpretiert werden kann:

Ist der Automat zu einem Zeitpunkt t im Zustand z(t), und liegt zum Zeitpunkt t die Eingabe x(t) an, so geht der Automat im nächsten Zeitpunkt $t + \Delta t$ in den neuen Zustand $z (t + \Delta t)$ über, also

$$z (t + \Delta t) = f_z (x(t) , z(t))$$

Da uns der genaue zeitliche Ablauf der Arbeit eines Automaten nicht interessiert, lassen wir diese Zeitabhängigkeit in der Schreibweise immer weg.

Ausgabefunktion eines Automaten

Die Ausgabefunktion f_a ist eine Abbildung

$$f_a : X \times Z \rightarrow Y$$

der Produktmenge $X \times Z$ in die Menge Y der möglichen Ausgabewerte.

Im allgemeinen ist die Ausgabe eines Automaten somit abhängig vom Zustand und von der Eingabe (sog. Mealy-Automat). Der Zustand z(t) zum Zeitpunkt t und die Eingabe x(t) zum Zeitpunkt t bestimmen den Ausgabewert y(t).

Ist die Ausgabe unabhängig von der Eingabe, also $f_a : Z \rightarrow Y$, so spricht man auch von einem Moore-Automaten.

Man unterstellt, daß die Ausgabe y(t) beim Übergang von z(t) nach $z(t + \Delta t)$ erzeugt wird, also:

$$y(t) = f_a(x(t), z(t))$$

Beispiel:

Ein Verkaufsautomat gibt nach Einwurf eines Geldstückes x(t) in Abhängigkeit vom Füllstand z(t) das Geldstück x(t) oder die Ware y(t) aus.

Nicht-deterministische Automaten

Als Verallgemeinerung der bisher definierten (deterministischen) Automaten, in denen die Zustands- und die Ausgabefunktion echte Funktionen, d.h. mit einem eindeutigen Funktionswert sind, betrachtet man auch Automaten, deren Zustands- und/oder Ausgabe-"funktion" Relationen sind, also mehrere Werte annehmen können.

Definition:

Ein nicht-deterministischer Automat ist ein 5-Tupel

$$A = (X , Y , Z , f_z , f_a)$$

mit Mengen X (Eingaben), Y (Ausgaben), Z (Zuständen) und Relationen

$$f_z : \quad X \times Z \rightarrow \text{Potenzmenge (Z)}$$

$$f_a : \quad X \times Z \rightarrow \text{Potenzmenge (Y)}$$

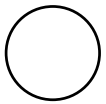
Beschreibung von Automaten

Zur Beschreibung des Verhaltens von Automaten dienen

a) Zustandsdiagramme (state diagram)

b) Zustandstabellen (state table)

Elemente zur Darstellung von Zustandsdiagrammen:



Zustandsknoten

akt. Eingabe / akt. Ausgabe
Zustandsvektor

Aufbau von Zustandstabellen

	Zustände (1. Angabe: neuer Zustand, 2. Angabe: Ausgabewert)				
Eingaben	z_1	z_2	z_3	. . .	z_n
x_1	z_{11},y_{11}	z_{12},y_{12}	z_{13},y_{13}	. . .	z_{1n},y_{1n}
x_2	z_{21},y_{21}	z_{22},y_{22}	z_{23},y_{23}	. . .	z_{2n},y_{2n}
...	...	...	...		...
x_m	z_{m1},y_{m1}	z_{m2},y_{m2}	z_{m3},y_{m3}	. . .	z_{mn},y_{mn}

1. Automatenbeispiel: Mausefalle

Eine Mausefalle als Beispiel für einen einfachen, endlichen Automaten mit Ausgabe

Eingabealphabet $X = \{ M , \overline{M} \}$

mit Bedeutung

M = "Maus kommt"

$\overline{M}$ = "Maus kommt nicht"

Zustandsmenge $Z = \{ G , \overline{G} \}$

mit Bedeutung

G = "Falle gespannt"

$\overline{G}$ = "Falle nicht gespannt"

Ausgabealphabet $Y = \{ T , \overline{T} \}$

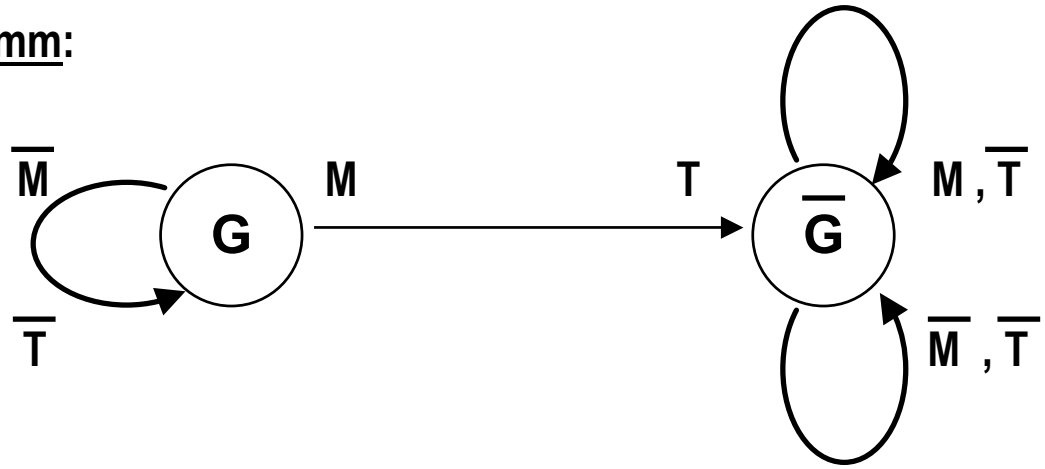
mit Bedeutung

T = "Maus tot"

$\overline{T}$ = "Maus nicht tot"

Mausefalle: Zustandsdiagramm und -Tafel

Zustandsdiagramm:



Zustandstafel:

	Zustände	
Eingaben	G	$\overline{G}$
M	$\overline{G}$, T	$\overline{G}$, $\overline{T}$
$\overline{M}$	G , $\overline{T}$	$\overline{G}$, $\overline{T}$

2. Automatenbeispiel: Blumenautomat

Ein Blumenautomat hat 3 Fächer, jedes Fach enthält einen Strauß Nelken.

<u>Zustandsmenge Z</u> :

$Z = \{ z_3 , z_2 , z_1 , z_0 \}$ = { 3 Sträuße , 2 Sträuße , 1 Strauß , 0 Sträuße }

<u>Eingabemenge X</u> :

Ein Nelkenstrauß kostet 2.-- € , der Automat akzeptiert nur Zwei-Euro-Stücke, wechselt also nicht.

$X = \{ x_1 , x_2 , x_3 \}$ = { 2-€-Stück, unpassendes Geld, keine Eingabe }

<u>Ausgabemenge Y</u> :

$Y = \{ y_1 , y_2 , y_3 \}$ = { 1 Nelkenstrauß, Geldstück, keine Ausgabe }

Blumenautomat: Zustands- und Ausgabefunktion

- Zustandsfunktion $f_z : X \times Z \rightarrow Z$

mögl. Eingaben	mögliche Zustände			
	z_3	z_2	z_1	z_0
x_1	z_2	z_1	z_0	z_0
x_2	z_3	z_2	z_1	z_0
x_3	z_3	z_2	z_1	z_0

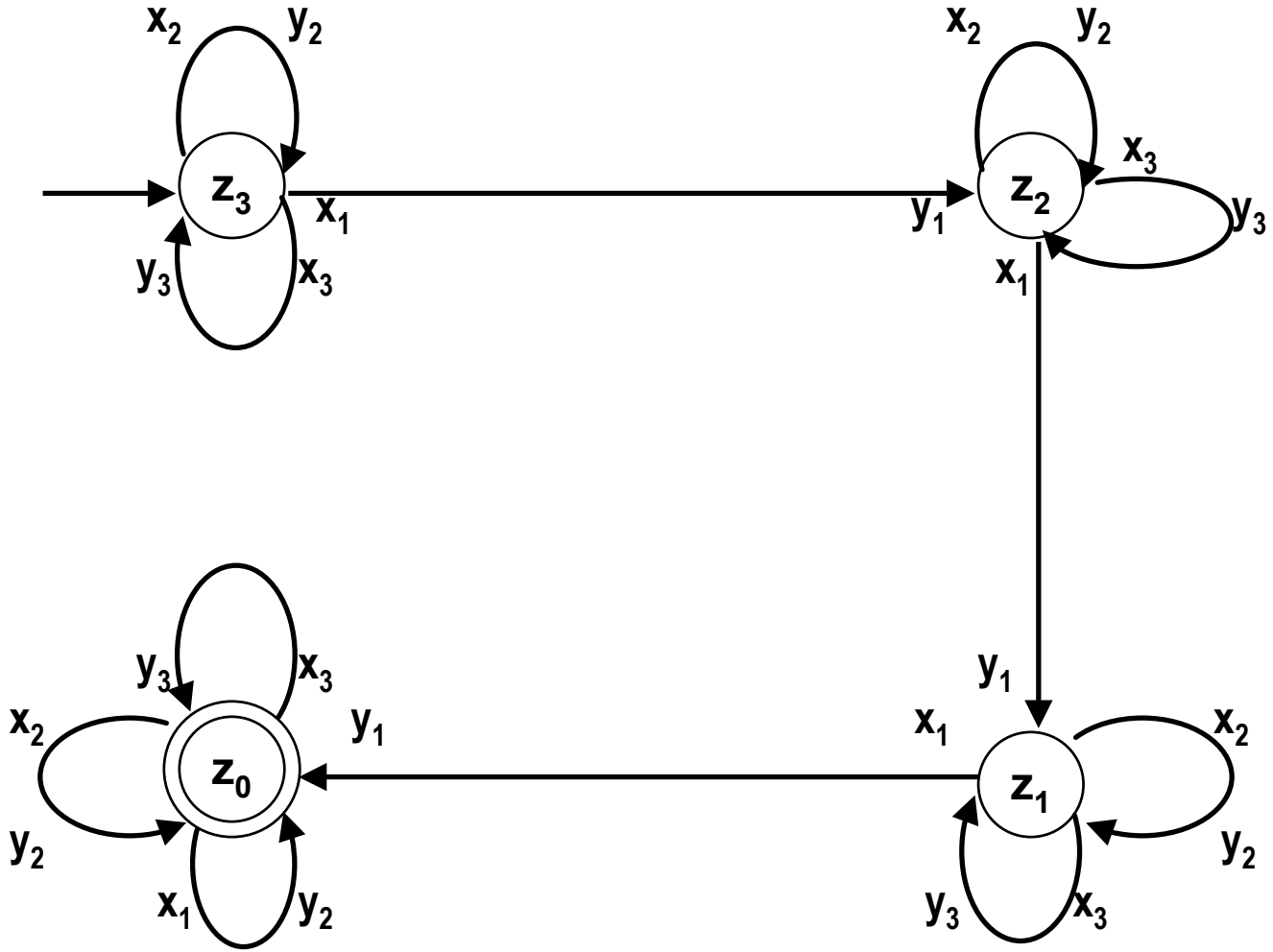
- Ausgabefunktion $f_a : X \times Z \rightarrow Y$

	mögliche Zustände			
Eingabe	z_3	z_2	z_1	z_0
x_1	y_1	y_1	y_1	$y_2 = x_1$
x_2	$y_2 = x_2$	$y_2 = x_2$	$y_2 = x_2$	$y_2 = x_2$
x_3	y_3	y_3	y_3	y_3

Blumenautomat : Zustandsfolgetabelle

Eingabe	Zustand	Folgezustand	Ausgabe
x_1	z_3	z_2	y_1
x_1	z_2	z_1	y_1
x_1	z_1	z_0	y_1
x_1	z_0	z_0	y_2
x_2	z_3	z_3	y_2
x_2	z_2	z_2	y_2
x_2	z_1	z_1	y_2
x_2	z_0	z_0	y_2
x_3	z_3	z_3	y_3
x_3	z_2	z_2	y_3
x_3	z_1	z_1	y_3
x_3	z_0	z_0	y_3

Blumenautomat : Zustandsdiagramm



Stochastische Automaten

- **Ein stochastischer Automat ist ein 5-Tupel**

$$A_S = (X, Y, Z, P(z(t+\Delta t)/x(t),z(t)) , P(y(t)/x(t),z(t)))$$

wobei $P(z(t+\Delta t)/x(t),z(t))$ die Wahrscheinlichkeitsfunktion für einen neuen Zustand $z(t + \Delta t)$ bei einer Eingabe $x(t)$ und dem alten Zustand $z(t)$ ist, sowie $P(y(t)/x(t),z(t))$ die Wahrscheinlichkeitsfunktion für eine Ausgabe $y(t)$ bei einer Eingabe $x(t)$ und dem alten Zustand $z(t)$.

- **Sonderformen:**
  - **Stochastischer Automat mit determinierter Ausgabefunktion und indeterminierter Zustandsfunktion**

$$A_S = (X, Y, Z, P(z(t + \Delta t)/x(t),z(t) , f_a))$$

  - **Die bedingten Wahrscheinlichkeiten werden durch unbedingte ersetzt**

$$A_S = (X, Y, Z, P(z(t + \Delta t)), P(y(t)))$$

Bei dieser Form bleiben die unabhängigen Variablen von Zustands- und Ausgabefunktion außer Betracht.

Inzidenzmatrizen für stochastische Automaten

Bei einem deterministischen Automaten ist der Übergang vom Zustand z_i in den Zustand z_J ein sicheres Ereignis. Im Fall eines endlichen stochastischen Automaten geschieht der Übergang mit einer Wahrscheinlichkeit

$$0 \leq p_{IJ} \leq 1 \quad \text{mit} \quad \sum_{j=1}^{n} p_{IJ} = 1$$

die meist auch noch vom Eingabewert x abhängt. Diese Wahrscheinlichkeiten werden dann in die Inzidenzmatrizen $I(x_k)$ (eine Matrix pro Eingabewert) eingetragen.

z_i \ z_j	z_1	z_2	. . .	z_n
z_1	$p(x_k)_{11}$	$p(x_k)_{12}$		$p(x_k)_{1n}$
z_2	$p(x_k)_{21}$	$p(x_k)_{22}$		$p(x_k)_{2n}$
:	:	:		:
z_n	$p(x_k)_{1n}$	$p(x_k)_{2n}$		$p(x_k)_{nn}$

$= I(x_k)$

Blumenautomat : Inzidenzmatrizen

Ableitung der Inzidenzmatrizen I (eine Matrix für jeden Eingabewert):

1. **Für jeden Zustand z_i den Folgezustand z_J bei diesem Eingabewert feststellen.**
2. **Das entsprechende Matrixelement i_{iJ} wird mit 1 und der zugehörigen Ausgabe markiert.**

$z_i \backslash z_J$	z_3	z_2	z_1	z_0
z_3	0	$1,y_1$	0	0
z_2	0	0	$1,y_1$	0
z_1	0	0	0	$1,y_1$
z_0	0	0	0	$1,y_2$

$= I(x_1)$

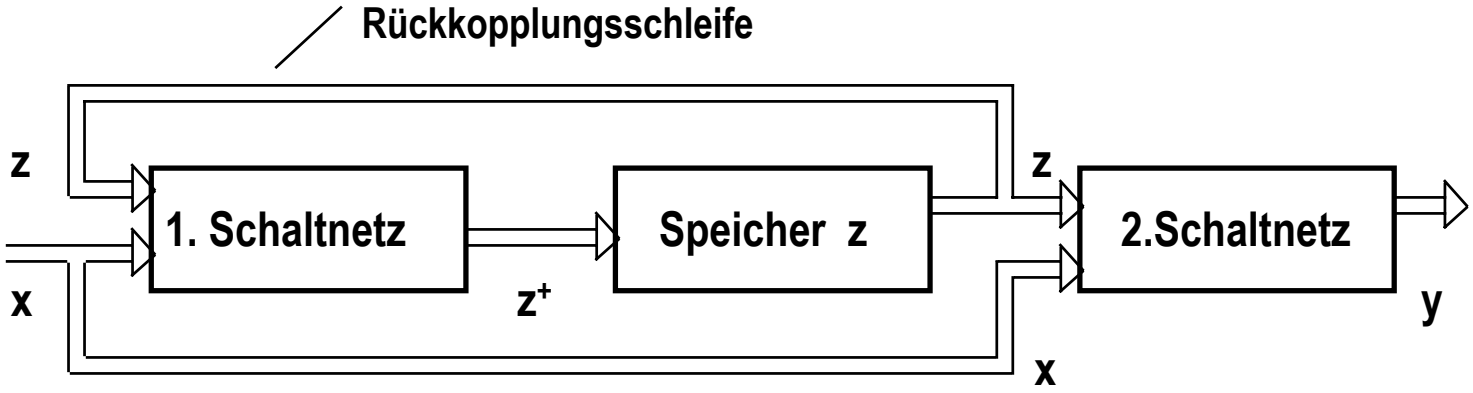
$z_i \backslash z_J$	z_3	z_2	z_1	z_0
z_3	$1,y_2$	0	0	0
z_2	0	$1,y_2$	0	0
z_1	0	0	$1,y_2$	0
z_0	0	0	0	$1,y_2$

$= I(x_2)$

$z_i \backslash z_J$	z_3	z_2	z_1	z_0
z_3	$1,y_3$	0	0	0
z_2	0	$1,y_3$	0	0
z_1	0	0	$1,y_3$	0
z_0	0	0	0	$1,y_3$

$= I(x_3)$

Technische Realisierung von Automaten



Erklärung:

1. Schaltnetz: Übergangsschaltnetz oder "next state decoder"
Aus der Eingabe x und dem Zustand z wird der Folgezustand z^+ abgeleitet.

2. Schaltnetz : Ausgabe- (Ausgangs-) Schaltnetz "output decoder"
Aus den Variablen x und z wird die Ausgangsvariable y abgeleitet.

Erkennende, endliche Automaten

Die Fragestellung bei einem Automaten mit Ausgabe lautet:

Welche Ausgabe produziert der endliche Automat bei welcher Eingabe?

Im Gegensatz dazu fragt man bei einem endlichen Automaten ohne Ausgabe (Senke):

Führt eine Eingabe(folge) den Automaten in einen definierten Endzustand, d.h. wird eine Eingabe(folge) erkannt? Man spricht dann auch von **erkennenden Automaten**.

Definition:

Ein erkennender endlicher Automat ist ein 5-Tupel

$$A = (X , Z , Z_E , z_0 , f_z)$$

wobei:

$X = \{ x_1 , x_2 , \ldots , x_n \}$ Eingabealphabet

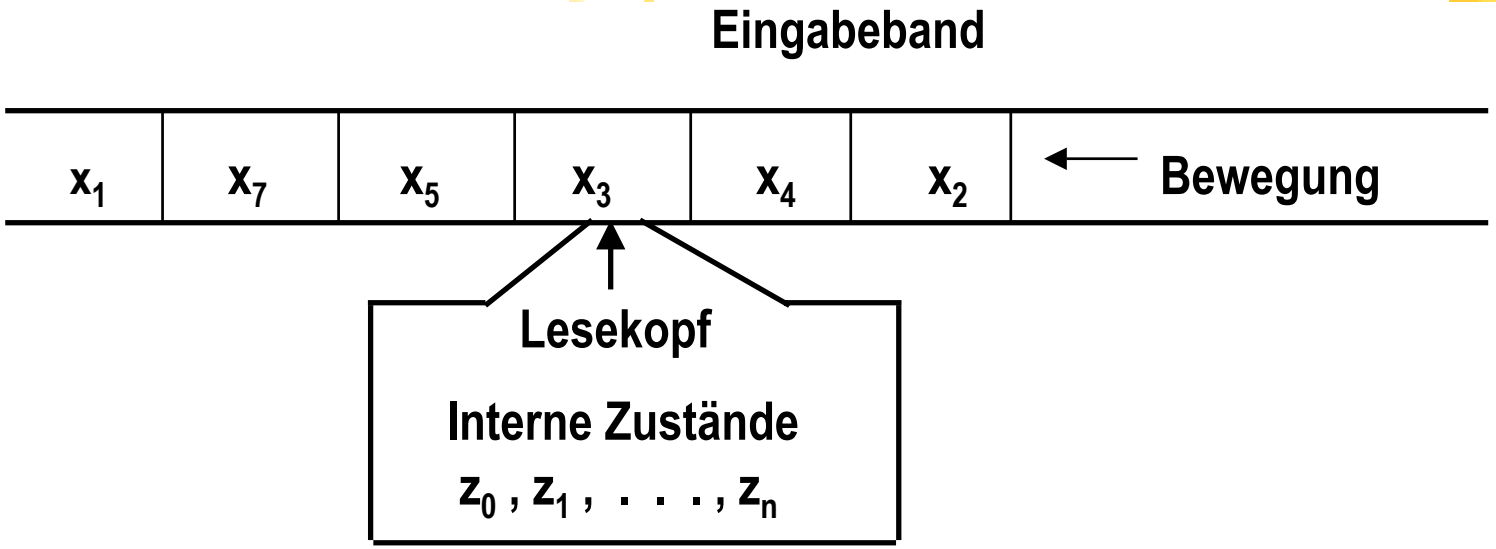
$Z = \{ z_0 , z_1 , \ldots , z_m \}$ Zustandsmenge

$Z_E = \{ z_{e1}, \ldots , z_{er} \} \subseteq Z$ Menge der (zulässigen) Endzustände

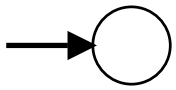
$z_0 \in Z$: Anfangszustand, wobei $z_0 \notin Z_E$

$f_z : X \times Z \rightarrow Z$ Zustandsfunktion

Modell eines erkennenden Automaten



Im Zustandsdiagramm führt man zusätzlich folgende Notationen ein:



Anfangszustand z_0
(mit Bedingung $z_0 \notin Z_E$)

Endzustand $z_e \in Z_E$

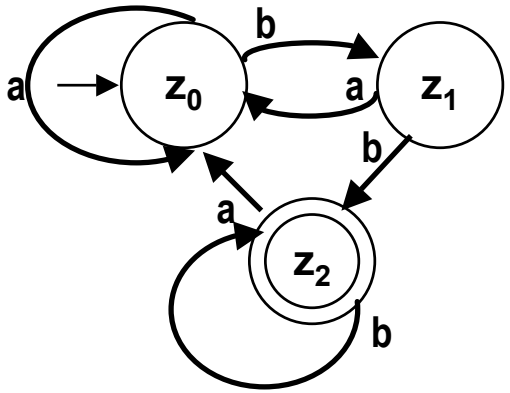
Beispiel 1

Gegeben sei ein erkennender, endlicher Automat A durch:

$X = \{ a, b \}, \quad Z = \{ z_0, z_1, z_2 \}, \quad Z_E = \{ z_2 \}$

sowie die Überführungsfunktion in Form des Zustandsdiagrammes:

Eingabe	Zustand nach Abarbeitung
a a b	z_1
b a b a	z_0
a b b	z_2
a b b a b b	z_2



Beispiel 2

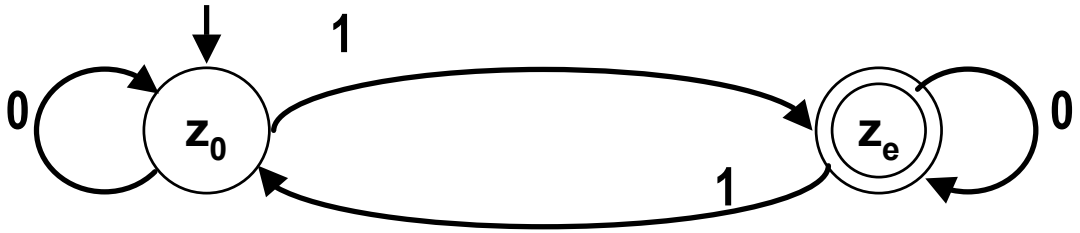
Gegeben sei ein erkennender, endlicher Automat durch:

$X = \{ 0 , 1 \} , Z = \{ z_0 , z_e \} , \; Z_E = \{ z_e \}$

und die nebenstehende Überführungsfunktion f_z in Form einer Zustandstafel

	0	1
z_0	z_0	z_e
z_e	z_e	z_0

Frage: Was leistet der Automat, welche Eingaben führen in den zulässigen Endzustand?



	Eingabe	Zustand nach Abarbeitung
1.	0 0 1 0 1	z_0
2.	0 0 1 0 0	z_e
3.	0 0 1 1 0 1 0	z_e

Analyse versus Synthese von Automaten

Zwei grundlegende Fragestellungen sind im Zusammenhang mit Automaten von Interesse:

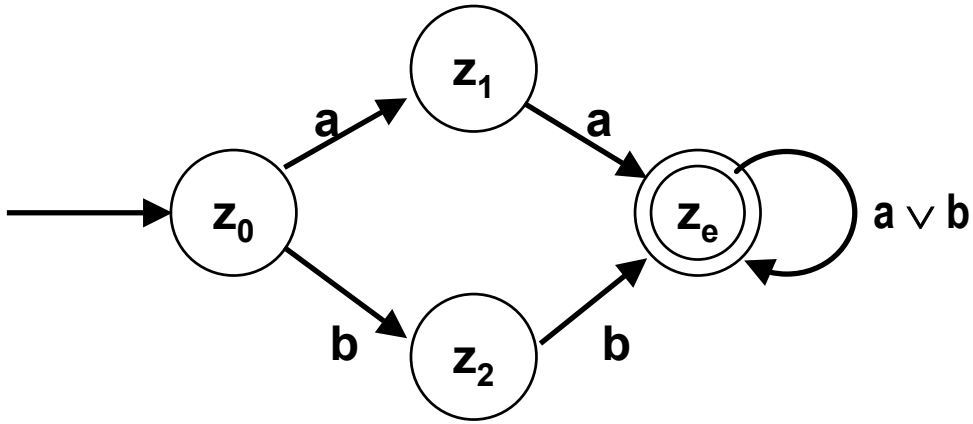
- **Wie verhält sich ein vorgegebener Automat → Analyseproblem (bisherige Betrachtungsweise)**
- **Wie muß ein Automat für eine vorgegebene Aufgabe beschaffen sein → Konstruktions- / Syntheseproblem**

Konstruktion eines Automaten - 1. Entwurf

Aufgabe: Wie müssen die Zustandsmenge Z und die Überführungsfunktion f_z definiert werden, damit bei $X = \{ a , b \}$ genau solche Eingaben auf den (einzigen) Endzustand $z_e \in Z$ führen, in denen zwei aufeinanderfolgende a oder b vorkommen ?

Bemerkung: Es ist nicht bekannt, wieviele Zustände notwendig sein werden.

Entwurf:

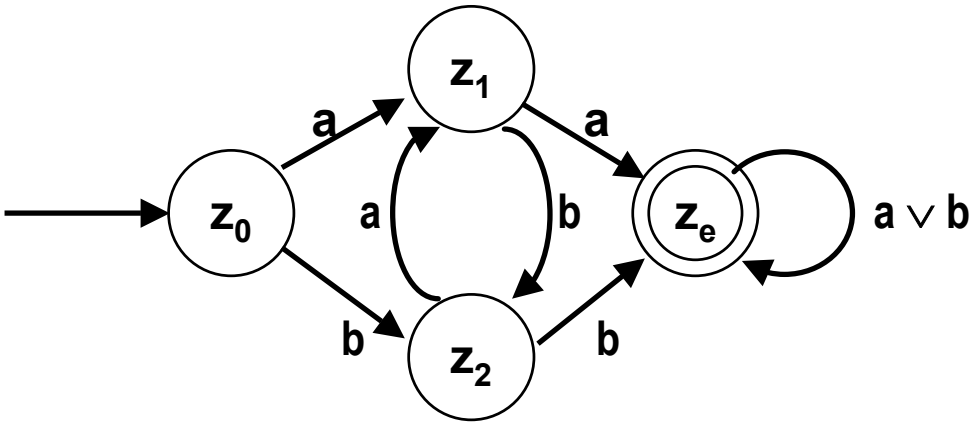


Fehler:

Die Bedingung $X \times Z \rightarrow Z$ ist in z_1 und z_2 nicht erfüllt, es gehen keine Kanten mit der Bewertung b von z_1 bzw. mit der Bewertung a von z_2 aus. Hier kann aa oder bb nur am Anfang einer Eingabe stehen.

Konstruktion eines Automaten - 2. Entwurf

Richtiger, vollständiger Entwurf:



Die (Teil-)Folge aa oder bb kann nun am Anfang, in der Mitte, oder am Ende des Eingabewortes erkannt werden.

Wichtige Forderung:

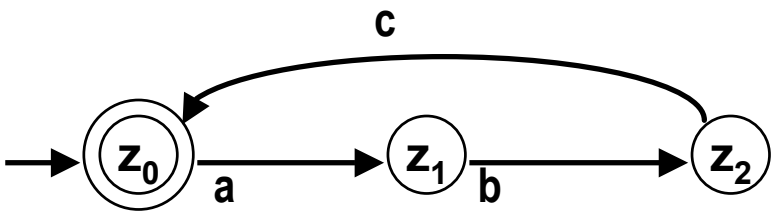
In jedem einzelnen Zustand müssen alle Eingabemöglichkeiten, also alle Elemente der Eingabemenge berücksichtigt werden.

Konstruktion eines Automaten - 2. Beispiel

Aufgabe: Es soll ein endlicher Automat entworfen werden, der für $X = \{ a , b , c \}$ ausgehend von z_0 genau nach Eingabe von abc, abcabc, abcabcabc, . . . in einen Endzustand gelangt.

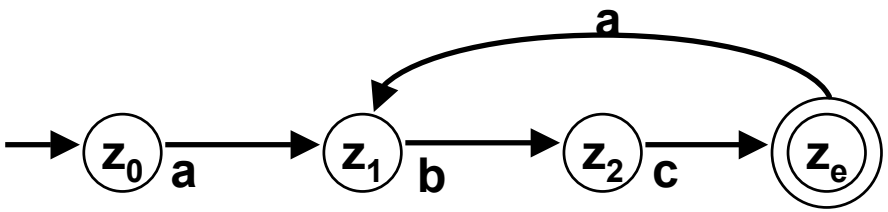
Entwürfe:

a)



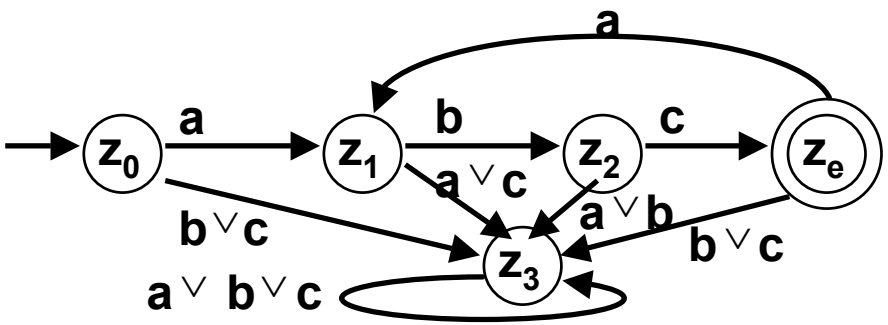
falsch, u.a. da $z_0 \in Z_E$

b)



noch unvollständig

c)



vollständig richtig !

Bemerkung:

aus z_3 ist der Endzustand z_e nicht mehr erreichbar

Der endliche Automat als Akzeptor

Sei X^* die Menge der Wörter über dem Eingabealphabet X.

Die Zustandsfunktion $f_z : X \times Z \rightarrow Z$ läßt sich kanonisch fortsetzen zu einer Funktion

$$f_z^* : X^* \times Z \rightarrow Z$$

$$f_z^* (x_1 x_2 \ldots x_n , z) = f_z (x_n , f_z (x_{n-1} , \ldots , f_z (x_2 , f_z (x_1 , z))\ldots)$$

indem die sukzessiven Zustandsübergänge des Automaten bei aufeinanderfolgender Eingabe von $x_1, x_2 , \ldots , x_n$ genommen werden.

Insbesondere interessiert uns, in welchen Zustand der Automat ausgehend vom Startzustand z_0 bei einem Eingabewort $x \in X^*$ übergeht.

Definition: Sei $A = (X, Z, f_z, z_0, Z_E)$ ein endlicher Automat und $x \in X^*$ ein Eingabewort. Der Automat A akzeptiert (oder erkennt) x, wenn

$$f_z^* (x, z_0) \in Z_E ,$$

d.h. wenn A bei der Verarbeitung von x in einen Endzustand übergeht.

Sprache eines endlichen Automaten

Definition:

Die Menge aller Eingabewörter $x \in X^*$, die ein endlicher Automat $A = (X, Z, f_z, z_0, Z_E)$ akzeptiert, heißt die Sprache $L(A)$ von A, also

$$L(A) = \{x \in X^* \mid f_z^*(x, z_o) \in Z_E\}$$

Bisherige Beispiele:

a) $L(A) = \{ x \in \{a,b\}^* \mid x \text{ endet mit bb}\}$

b) $L(A) = \{ x \in \{0,1\}^* \mid x \text{ enthält eine ungerade Zahl von EINSen } \}$

c) $L(A) = \{ x \in \{a,b\}^* \mid x \text{ enthält aa oder bb } \}$

d) $L(A) = \{ x \in \{a,b,c\}^* \mid x \text{ ist eine Verkettung von abc } \}$

Satz: Die Sprache $L(A) \subset X^*$ eines endlichen Automaten kann eine endliche oder unendliche Menge sein.

Satz: Es gibt (jede Menge) Teilmengen von X^*, die nicht Sprache L(A) eines endlichen Automaten sind.

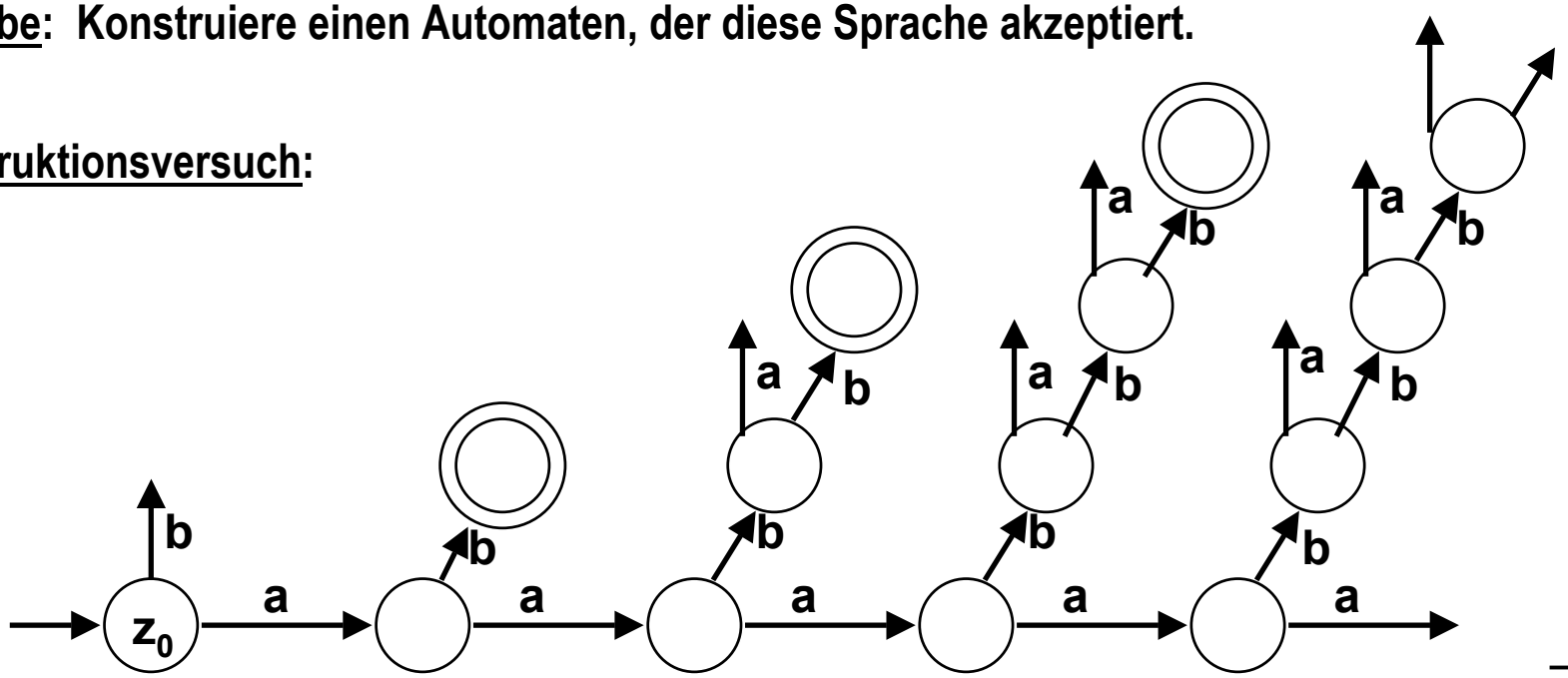
Eine interessante Sprache

Notation: $x = \underbrace{a \ldots a}_{m} \underbrace{b \ldots . b}_{n} = a^m b^n$ z. B. $x = aaaaabbb = a^5 b^3$

Gegeben sei $X = \{ a, b \}$ und die Sprache $L = \{ x \mid x = a^m b^m , m \in N \}$.

Aufgabe: Konstruiere einen Automaten, der diese Sprache akzeptiert.

Konstruktionsversuch:



Pumping-Lemma für endliche Automaten

Der folgende Satz ist unmittelbar einsichtig:

Satz: Hat ein endlicher Automat n Zustände und besteht ein Eingabewort x aus p > n Zeichen, dann muß die durchlaufene Zustandsfolge einen Zyklus enthalten.

Aus dem Satz folgt:

Pumping-Lemma für endliche Automaten:

Sei L die Sprache eines endlichen Automaten. Dann gibt es eine Konstante n, so daß sich jedes Wort $z \in L$ der Länge $|z| \geq n$ schreiben läßt als $z = u\ v\ w$, mit $|u\ v| \leq n$ und $|v| \geq 1$, so daß für alle $i \geq 0$ auch $u\ v^i\ w \in L$.

Folgerung: Die Sprache $L = \{ x \mid x = a^m b^m , m \in N \}$ ist nicht Sprache eines endlichen Automaten.

Nichtdeterministische Automaten

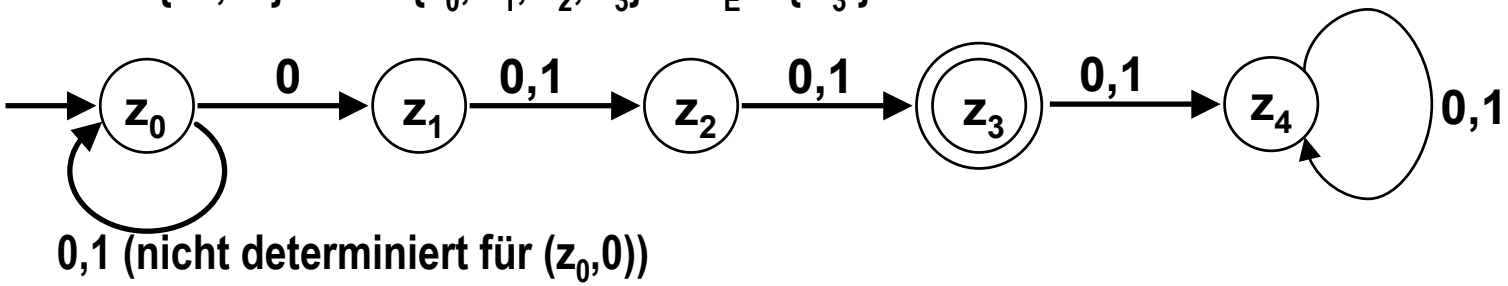
Deterministischer Automat:

$f_z : X \times Z \rightarrow Z$, d.h. zu jedem Tupel (x_i , z_i) gibt es genau einen Folgezustand.

Nichtdeterministischer Automat:

Bei gegebenem Eingabezeichen und Zustand kann der Automat in verschiedene Zustände übergehen, d.h. es ist kein eindeutiger Folgezustand gegeben.

Beispiel: $X = \{ 0 , 1 \}$ $Z = \{z_0, z_1, z_2, z_3\}$ $Z_E = \{ z_3 \}$



Übergangsrelation $\subseteq X \times Z \times Z$:

$\{ (0, z_0 , z_0), (0, z_0 , z_1), (1, z_0 , z_0), (0, z_1 , z_2),$
$(1, z_1 , z_2), (0, z_2 , z_3), (1, z_2 , z_3), (0, z_3 , z_4),$
$(1, z_3 , z_4), (0, z_4 , z_4), (1, z_4 , z_4) \}$

Nichtdeterministischer endlicher Automat

Definition:

Ein nicht-deterministischer, erkennender endlicher Automat ist ein 5-Tupel

$$A = (X , Z , Z_E , S , f_z),$$

$X = \{ x_1 , x_2 , \ldots , x_n \}$	Eingabealphabet
$Z = \{ z_0 , z_1 , \ldots , z_m \}$	Zustandsmenge
$Z_E = \{ z_{e1}, \ldots , z_{er} \} \subseteq Z$	Menge der (zulässigen) Endzustände
$S \subseteq Z$	Menge der Anfangszustände
$f_z : X \times Z \rightarrow P(Z) \setminus \varnothing$	Zustandsfunktion (P(Z) ist die Potenzmenge von Z)

Die Zustandsfunktion kann fortgesetzt werden zu einer Funktion

$$f_z^* : X^* \times P(Z) \rightarrow P(Z)$$

indem man für $Z' \subseteq Z$ und $x \in X^*$, $x = x_i x'$ rekursiv definiert:

$$f_z^* (\Lambda , Z') = Z' , \quad f_z^* (x_i x' , Z') = \bigcup_{z \in Z'} f_z^* (x' , f_z (x_i , z))$$

Sprache eines nichtdeterministischen Automaten

Definition:

Die Sprache L(A) eines nichtdeterministischen Automaten A ist die Menge von Eingabewörtern x, für die es, ausgehend von einem Startzustand, eine Folge von Zuständen gibt, die in einem erlaubten Endzustand endet. Formal:

$$L(A) = \{x \in X^* \mid f_z^*(x, S) \cap Z_E \neq \varnothing\}$$

Bemerkungen:

- Ein deterministischer Automat ist schneller bei der Überprüfung, ob $x \in L(A)$ ist.
  Anwendung: Compiler, Mustererkennung (pattern matching).

- Ein nichtdeterministischer Automat ist platzsparender.

Äquivalenzsatz für endliche Automaten

Satz: Zu jedem nichtdeterministischen endlichen Automaten gibt es einen deterministischen endlichen Automaten, der dieselbe Sprache akzeptiert.

Beweis:

Sei $A = (X , Z , Z_E , S , f_z)$ ein nicht-deterministischer endlicher Automat. Wir definieren einen deterministischen endlichen Automaten, dessen Zustände alle Teilmengen von Z sind:
$A' = (X , P(Z) , Z_E' , z_o , f_z')$ wie folgt:

$$z_o = S, \qquad Z_E' = \{Z' \in P(Z) \mid Z' \cap Z_E \neq \varnothing \},$$

$$f_z' (x , Z') = \bigcup_{z \in Z'} f_z (x , z) = f_z^* (x , Z') \quad \text{für } Z' \in P(Z)$$

Dann gilt für alle $x = x_1 x_2 \ldots x_r \in X^*$:

$x \in L(A) \Leftrightarrow f_z^* (x , S) \cap Z_E \neq \varnothing$

$\Leftrightarrow$ es gibt Folge von Teilmengen $Z_1, Z_2 , \ldots , Z_r$ von Z mit
$f_z' (x_1 , S) = Z_1 , \; f_z' (x_2 , Z_1) = Z_2 , \ldots , \; f_z' (x_r , Z_{r-1}) = Z_r$
und $Z_r \cap Z_E \neq \varnothing$

$\Leftrightarrow f_z'^* (x , S) \in Z_E' \Leftrightarrow x \in L(A')$

Minimierung von Automaten

Ein kleinster Automat (d.h. einer mit einer minimalen Anzahl von Zuständen), der ein gegebenes Verhalten zeigt, wird als minimaler Automat bezeichnet.

Definition: Äquivalenz zweier Automaten

Zwei Automaten A_1 , A_2 heißen äquivalent , wenn sie

a) als übersetzende Automaten das gleiche Ein-Ausgabe-Verhalten haben

b) als erkennende Automaten die gleiche Sprache akzeptieren.

Definition: Minimale Automaten

Ein Automat heißt minimal , wenn es keinen äquivalenten Automaten mit weniger Zuständen gibt.

Konstruktion minimaler Automaten (1)

Bei fast allen Automatentypen gibt es zu einem Automaten mehrere verschiedene, äquivalente minimale Automaten. Bei deterministischen endlichen Automaten ist der minimale Automat eindeutig bestimmt und läßt sich nach folgenden Regeln konstruieren:

<u>Regeln</u>:

1. **Entferne alle Zustände, die vom Anfangszustand z_0 aus nicht erreichbar sind.**

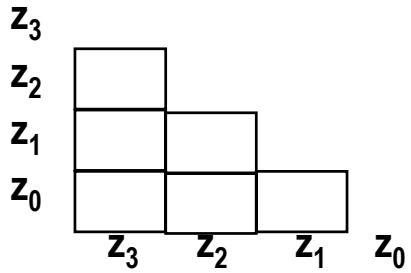
2. **Fasse äquivalente Zustände zu einem Zustand zusammen.**
   Dabei sind zwei Zustände z und z' äquivalent, wenn für alle Wörter $x \in X^*$ gilt:

$$f_z^*(x, z) \in Z_E \quad \Leftrightarrow \quad f_z^*(x, z') \in Z_E$$

Die nächste Folie beschreibt einen Algorithmus zur Bestimmung äquivalenter Zustände.

Konstruktion minimaler Automaten (2)

Der Algorithmus zum Auffinden äquivalenter Zustände basiert darauf, rekursiv Paare von Zuständen zu finden, die *nicht* äquivalent sind, und diese zu markieren. Die am Ende nicht markierten Zustände sind äquivalent. Eine mögliche Darstellung der Zustandspaare ist z.B.:



1. Markiere alle Paare aus einem Endzustand und einem Nicht-Endzustand.
2. Überprüfe für jedes nicht markierte Paar von Zuständen (z_i,z_k), ob für eines der Eingabezeichen x_n das Zustandspaar $(f_z(x_n,z_i),f_{z(}x_n,z_k))$ markiert ist. Falls ja, markiere das Paar (z_i,z_k).
3. Wiederhole 2. bis sich keine Änderung mehr ergibt.

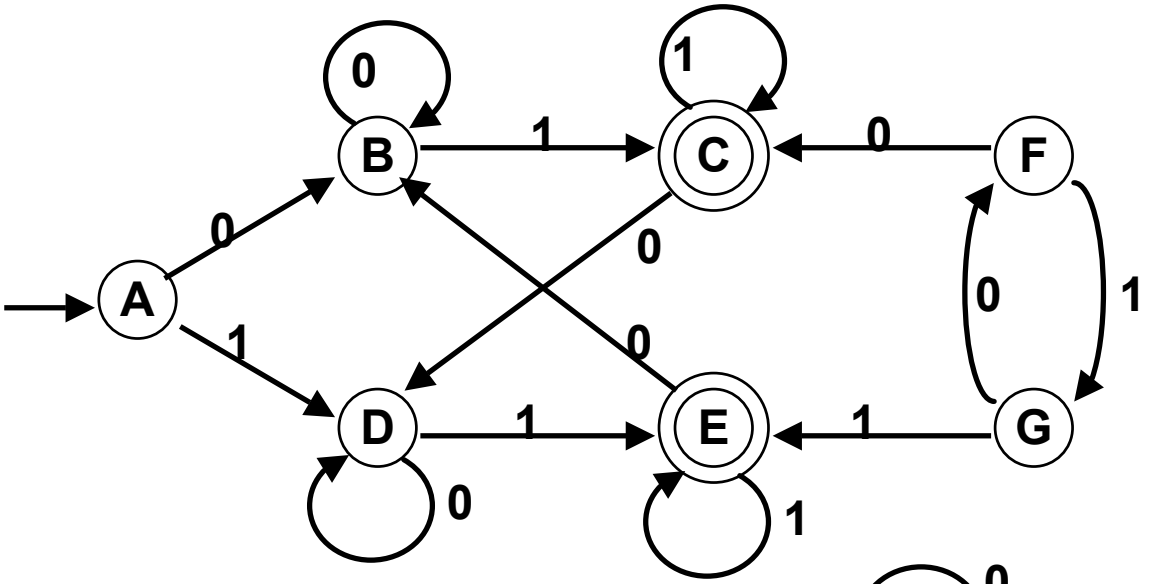
Jedes nicht markierte Paar besteht aus zwei äquivalenten Zuständen.

Minimierung endlicher Automaten - Beispiel

Beispiel: Gegeben sei ein erkennender endlicher Automat $A = (X , Z , f_z , z_0 , Z_E)$ durch:

$X = \{ 0 , 1 \}$, $Z = \{ A , B , C , D , E , F , G \}$, $z_0 = \{ A \}$, $Z_E = \{ C , E \}$

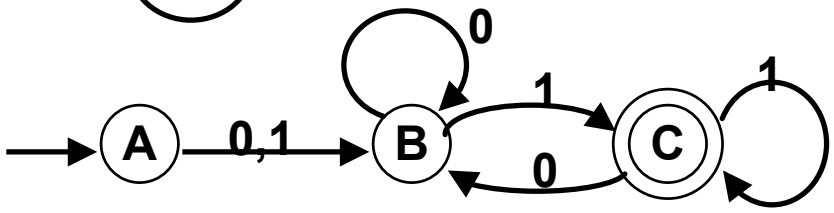
f_z :



Minimalversion :

F, G sind nicht erreichbar

B ist äquivalent zu D, C zu E



Grenzen endlicher Automaten

Wir haben gesehen, daß es Sprachen gibt (sogar sehr einfache), die nicht von einem endlichen Automaten erkannt werden, z.B.

$$L = \{ x \mid x = a^m b^m , m \in N \}$$

Auch die nicht-deterministischen endlichen Automaten erweitern nicht die Klasse der Sprachen endlicher Automaten.

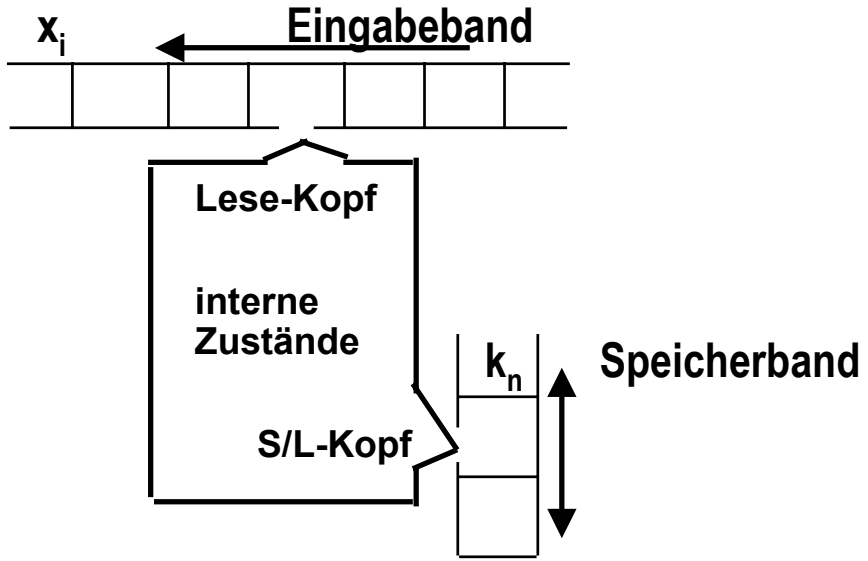
Das Problem liegt darin, daß ein endlicher Automat kein Gedächtnis hat: er merkt sich z.B. nicht, wieviele a's schon aufgetreten sind.

Wir suchen also nach einer Klasse von Automaten mit einer gewissen Speicherfähigkeit.

Modellvorstellung eines Kellerautomaten

Ein Kellerautomat ist ein endlicher Automat erweitert um einen Keller (Stack):

Modell:



Erklärung:

- **Mit Hilfe des Kellers kann der Automat sich beliebig viele Zeichen "merken".**
- **Zustandsübergänge sind auch abhängig vom zuletzt gemerkten Zeichen. Dabei werden entweder ein oder mehrere neue Zeichen oben auf den Keller gelegt, oder das oberste Zeichen wird vom Speicherband heruntergenommen und "vergessen".**

Definition eines Kellerautomaten

Definition: Ein **Kellerautomat** ist ein 7-Tupel $KA = (X , Z , K , z_0 , k_0 , Z_E , f_K)$, wobei

$X = \{ x_1 , x_2 , \ldots , x_n \}$ Eingabealphabet

$Z = \{ z_1 , z_2 , \ldots , z_m \}$ Zustandsmenge

$z_0 \in Z$ Anfangszustand

$K = \{ k_1 , k_2 , \ldots , k_r \}$ Kelleralphabet

$k_0 \notin K$ unterstes Kellerzeichen

$Z_E \subseteq Z$ Menge der (zulässigen) Endzustände

$f_K : (X \cup \{\Lambda\}) \times Z \times K \rightarrow P_e (Z \times K^*)$ Überführungsfunktion

(Λ = leeres Wort, P_e bezeichne die Menge der endlichen Teilmengen)

Bemerkungen:

- Die Definition beschreibt einen nicht-deterministischen Kellerautomaten. Der Kellerautomat heißt deterministisch, wenn die Mengen $f_K(x_i,z_k,k_l) \cup f_K(\Lambda,z_k,k_l)$ für alle x_i, z_k und k_l einelementig sind.
- Bei nicht-deterministischen Kellerautomaten wird bei der Definition gelegentlich darauf verzichtet, die Spezifikation einer Menge Z_E von Endzuständen zu verlangen.

Interpretation der Überführungsfunktion

- Die Überführungsfunktion gibt abhängig von
  - dem momentanen Eingabezeichen,
  - dem momentanen Zustand und
  - dem obersten Kellerzeichen

  an,
  - in welchen Zustand der Automat übergeht und
  - welches Kellerwort das oberste Kellerzeichen ersetzt.

  Im nicht-deterministischen Falle sind dabei Folgezustand und / oder neues Kellerwort nicht eindeutig bestimmt.
- Spezialfälle der Ersetzung des obersten Kellerzeichens sind:
  - Ersetzen mit dem leeren Wort Λ: Löschen des obersten Zeichens (POP-Operation)
  - Abhängig vom obersten Zeichen k_x Ersetzen mit k_i k_x (PUSH k_i)
- Daß f_K auch für (Λ, z, k) definiert ist bedeutet, daß unabhängig vom Eingabezeichen eine "spontane" Änderung des Zustandes mit Ersetzen des obersten Kellerzeichens möglich ist (aber ohne Bewegen des Eingabebandes).

Arbeitsweise eines Kellerautomaten

Am Anfang befindet sich der KA im Zustand z_0 , der Lese-Kopf über dem linken Zeichen des Eingabewortes x, der Schreib-Lese-Kopf über dem untersten Kellerfeld mit dem Inhalt k_0 .

Wenn sich im Lauf der Abarbeitung von x unter dem Lese-Kopf das Zeichen x_i , unter dem SL-Kopf das Zeichen k_h befindet und der Kellerautomat im Zustand z_J ist, so geschieht folgendes :

1. Ist $f_K (x_i , z_J , k_h) = f_K (\Lambda , z_J , k_h) = \varnothing$, so bleibt der Automat stehen.
2. Ist $(z_r , k) \in f_K (x_i , z_J , k_h)$, so kann der Kellerautomat in den Zustand z_r übergehen, k_h wird ersetzt durch $k \in K^*$, das Eingabeband wird um ein Feld nach links bewegt und der SL-Kopf über das letzte Zeichen von k positioniert (bzw. für $k = \Lambda$ um ein Feld nach unten bewegt, sofern nicht schon am Kellerboden).
3. Ist $(z_r , k) \in f_K (\Lambda , z_J , k_h)$ wird wie bei 2. vorgegangen, aber das Eingabeband nicht bewegt.
4. Beim Versuch, das Kellerbodenzeichen zu entfernen, bleibt der Kellerautomat stehen.

Sprache eines Kellerautomaten

Definition:

Ein Eingabewort $x \in X^*$ wird von einem Kellerautomaten akzeptiert, wenn der Kellerautomat nach Abarbeiten von x in einem Endzustand ist und der Keller leer ist (bzw. im nicht-deterministischen Fall: wenn es eine Abarbeitungsfolge gibt, die dies erreicht).

Definition:

Die Sprache L(KA) eines Kellerautomaten KA ist die Menge der von ihm akzeptierten Wörter $x \in X^*$.

Bemerkung:

Für *nicht*-deterministische Kellerautomaten ist es äquivalent, das Akzeptieren eines Wortes durch Erreichen eines Endzustandes zu definieren oder dadurch, daß der Keller leer ist (oder beides).

Für deterministische Kellerautomaten ist dies nicht der Fall.

Deterministische vs. nichtdeterministische Kellerautomaten

Satz: Es gibt nicht-deterministische Kellerautomaten, deren Sprache nicht Sprache eines deterministischen Kellerautomaten ist.

Beispiel: Die Sprache

$$L = \{a_1 a_2 \ldots a_n a_n \ldots a_2 a_1 \; ; \; a_i \in \{0,1\} \}$$

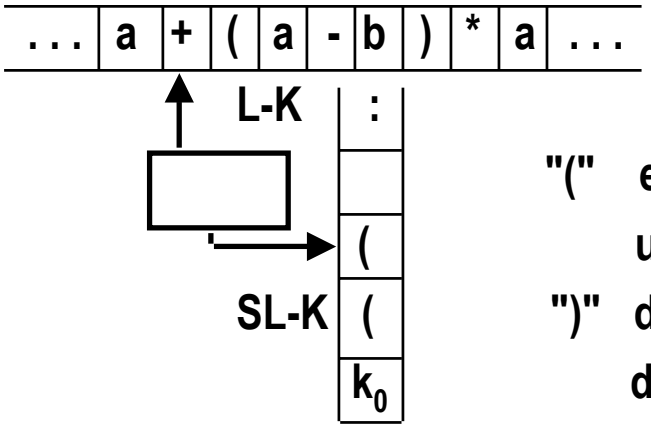
wird von einem nicht-deterministischen, jedoch von keinem deterministischen Kellerautomaten erkannt.

Im Gegensatz zu den endlichen Automaten ist also bei den Kellerautomaten die Klasse der von nicht-deterministischen Kellerautomaten erkannten Sprachen echt größer als die der Sprachen, die von deterministischen Kellerautomaten erkannt werden.

Kellerautomat - 1. Anwendungsbeispiel

- Syntaktische Analyse von Programmiersprachen, z.B. erkennender Automat für korrekt geklammerte arithmetische Ausdrücke.

Das Eingabeband enthält den zu überprüfenden arithmetischen Ausdruck.



"(" eine "(" wird in den Keller geschrieben,
und der SL-Kopf ein Feld nach oben bewegt.

")" die zuletzt geöffnete Klammer wird abgearbeitet und
der SL-Kopf ein Feld nach unten bewegt.

Wenn der arithmetische Ausdruck abgearbeitet ist, muß der Keller bis auf k_0 leer sein.

Fehlermöglichkeiten:

- Mehr "(" als ")" , d.h. der Keller ist nicht leer.
- Mehr ")" als "(" , d.h. Versuch k_0 zu löschen.

Kellerautomat - 2. Anwendungsbeispiel

Ein Kellerautomat, der die Sprache $L(KA) = \{ x \mid x = a^n b^n , n \in N \}$ erkennt, wird definiert durch:

$X = \{ a , b \} , Z = \{ z_1 , z_2 , z_3, z_F \} , z_0 = z_1 , z_E = \{ z_3 \} , K = \{ k , a \}$

f_K :

$f_K (a , z_1 , k) = (z_1 , ak)$	$f_K (b , z_1 , k) = (z_F , k)$
$f_K (a , z_1 , a) = (z_1 , aa)$	$f_K (b , z_2 , k) = (z_F , k)$
$f_K (b , z_1 , a) = (z_2 , \Lambda)$	$f_K (a , z_2 , *) = (z_F , *)$
$f_K (b , z_2 , a) = (z_2 , \Lambda)$	$f_K (* , z_F , *) = (z_F , *)$
$f_K (\Lambda , z_2 , k) = (z_3 , k)$ („künstlich“ in Endzustand bringen)	

Eingabeband	Zustand	Speicherband
(a) a a b b b Λ	z_1	(k) Λ Λ Λ Λ Λ
a (a) a b b b Λ	z_1	k (a) Λ Λ Λ Λ
a a (a) b b b Λ	z_1	k a (a) Λ Λ Λ
a a a (b) b b Λ	z_1	k a a (a) Λ Λ
a a a b (b) b Λ	z_2	k a (a) Λ Λ Λ
a a a b b (b) Λ	z_2	k (a) Λ Λ Λ Λ
a a a b b b (Λ)	z_2	(k) Λ Λ Λ Λ Λ
a a a b b b (Λ)	z_3	(k) Λ Λ Λ Λ Λ

Grenzen von Kellerautomaten

Auch mit Kellerautomaten lassen sich nicht alle Sprachen erkennen. Eine Sprache, die von keinem Kellerautomaten erkannt wird ist

$$L = \{ a^n b^n c^n ;\ n \geq 1 \}$$

Dies folgt aus dem

Pumping-Lemma für Kellerautomaten:

Sei L die Sprache eines Kellerautomaten. Dann gibt es eine Konstante n, so daß sich jedes Wort $z \in L$ der Länge $|z| \geq n$ schreiben läßt als $z = uvwxy$, $|vx| \geq 1$, und so daß für alle $i \geq 0$ auch $uv^iwx^iy \in L$.

Der Beweis hierfür wird nicht im Rahmen der Automatentheorie geführt, sondern in der Theorie der formalen Sprachen.

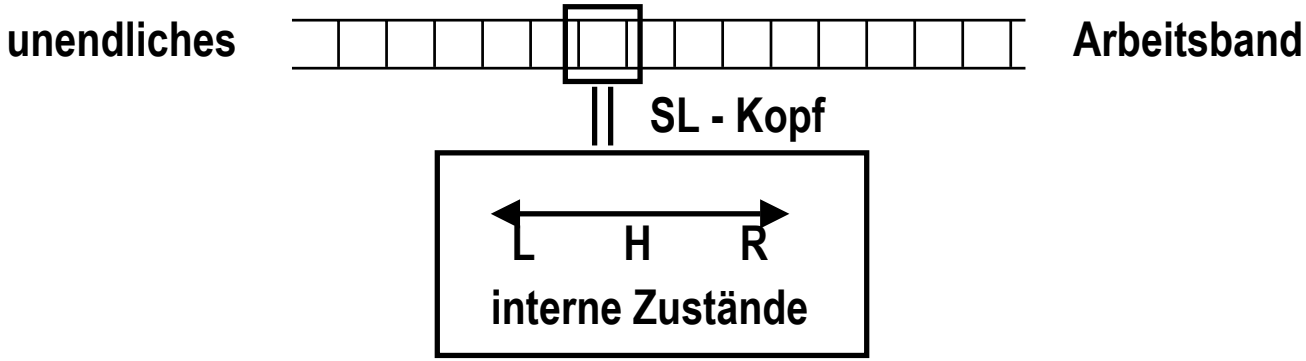
Die Einschränkungen der Kellerautomaten beruhen auf der Art der Verwendung des Speicherbandes:

- Nur das oberste Kellerelement ist im Zugriff.
- Das Speicherband wird nach jeder Bewegung nach unten gelöscht.

Modellvorstellung einer Turingmaschine

Turingmaschinen sind ein universelles Automatenmodell, das 1936 von Alan M. Turing (1912-1954) eingeführt wurde. Er verwendete es für seine Betrachtungen über die Berechenbarkeit von Problemen.

<u>Aufbau einer Turingmaschine</u> (Modell)



Im Gegensatz zu Kellerautomaten gibt es keinen Unterschied mehr zwischen Eingabe- und Speicherband.

Definition einer Turingmaschine

Definition: Eine **Turingmaschine** ist ein 6-Tupel $T = (X , Z , B , z_0 , Z_E , u)$ wobei

$X = \{ x_0 , x_1 , \ldots , x_r \}$ Menge der Bandzeichen, Bandalphabet

$Z = \{ z_0 , z_1 , \ldots , z_n \}$ Menge der internen Zustände

$B = \{ R , L , H \}$ Menge der Kopfbewegungen

$z_0 \in Z$ Anfangszustand

$Z_E \subseteq Z$ Menge der (zulässigen) Endzustände

$u : (X \cup \Lambda) \times Z \rightarrow (X \cup \Lambda) \times Z \times B$ Überführungsfunktion

(Λ = leeres Zeichen, Blank)

Für eine nicht-deterministische Turingmaschine definieren wir die Überführungsfunktion wie folgt:

$u : (X \cup \Lambda) \times Z \rightarrow P((X \cup \Lambda) \times Z \times B)$

Arbeitsweise einer Turingmaschine

- Zu Beginn befindet sich die Turingmaschine T im Anfangszustand $z_0 \in Z$ und der Schreib-Lese-Kopf über dem ersten (linkesten) Zeichen des Eingabewortes.

- Die Überführungsfunktion bestimmt abhängig vom unter dem SL-Kopf stehenden Bandzeichen x_i und dem aktuellen Zustand z_J
  - welches Zeichen x_k anstelle von x_i aufs Band geschrieben wird,
  - den Folgezustand z_L ,
  - die Kopfbewegung (R = ein Feld nach rechts , L = ein Feld nach links, H = keine Bewegung) .

- T durchläuft auf diese Weise eine Reihe derartiger Schritte. Die Turingmaschine hält an, wenn sie einen Endzustand $z_i \in Z_E$ erreicht.

- Die Überführungsfunktion u entspricht einem einfachen Maschinenprogramm mit minimalem Befehlsvorrat.

1. Beispiel einer Turingmaschine

$X = \{ 0 , 1 \} , Z = \{ z_0 , z_1 \} , Z_E = \{ z_1 \}$

Gesucht ist eine Turingmaschine, deren SL-Kopf auf einem Feld mit 1 stehend nach rechts das erste Feld sucht, das eine 0 enthält, in dieses Feld eine 1 schreibt, und darüber stehen bleibt.

u:
Maschinentafel

	0	1
z_0	$1z_1H$	$1z_0R$
z_1	$0z_1H$	$1z_1H$

Bandbeschriftung:

0	0	1	$z_0/1$	1	1	0	0
0	0	1	1	$z_0/1$	1	0	0
0	0	1	1	1	$z_0/1$	0	0
0	0	1	1	1	1	$z_0/0$	0
0	0	1	1	1	1	$z_1/1$	0

Halt

2. Beispiel einer Turingmaschine

Die folgende Turingmaschine interpretiert eine Eingabe x $\in$ {0,1}* als Binärzahl, addiert 1 hinzu und positioniert den Schreib-/Lesekopf wieder über dem most significant bit der Zahl:

$T = \{ \{0,1\}, \{z_0, z_1, z_2, z_e\}, \{R,L,H\}, \{z_0\}, \{z_e\}, u \}$

$u(z_0, 0) = (z_0, 0, R)$
$u(z_0, 1) = (z_0, 1, R)$
$u(z_0, \Lambda) = (z_1, \Lambda, L)$

$u(z_1, 0) = (z_2, 1, L)$
$u(z_1, 1) = (z_1, 0, L)$
$u(z_1, \Lambda) = (z_e, 1, H)$

$u(z_2, 0) = (z_2, 0, L)$
$u(z_2, 1) = (z_2, 1, L)$
$u(z_2, \Lambda) = (z_e, \Lambda, R)$

<u>Übung</u>: Beschreiben Sie die einzelnen Schritte, die diese Turingmaschine bei der Eingabe 101 durchläuft.

Sprache einer Turingmaschine

Definitionen:

- **Eine Turingmaschine T akzeptiert ein Wort $x \in X^*$, wenn sie ausgehend von der Bandbeschriftung x nach endlich vielen Arbeitsschritten in einem Endzustand anhält (bzw. im nicht-deterministischen Fall: wenn es eine solche Folge von Arbeitsschritten gibt).**
- **Die von T akzeptierte Sprache $L(T) \subseteq X^*$ ist die Menge aller von T akzeptierten Wörter.**

Satz (Zusammenhang zur Theorie der Sprachen und Grammatiken):

- **Zu jeder Grammatik gibt es eine Turingmaschine, die die von dieser Grammatik erzeugte Sprache akzeptiert.**
- **Umgekehrt gibt es zu jeder Turingmaschine T eine Grammatik, die die von T akzeptierte Sprache erzeugt.**

Satz: **Zu jeder nicht-deterministischen Turingmaschine gibt es eine deterministische Turingmaschine, die dieselbe Sprache akzeptiert.**

Das Halteproblem für Turingmaschinen

Definition :

Unter dem Halteproblem für Turingmaschinen versteht man die Frage nach der Existenz eines Algorithmus, mit dem man für jede Turingmaschine T und jede Bandbeschriftung x entscheiden kann, ob T bei der Bearbeitung von x nach endlich vielen Schritten anhält oder nicht.

Satz: Das Halteproblem für Turingmaschinen ist nicht entscheidbar.

Das heißt es gibt keinen Algorithmus, der für eine beliebige Turingmaschine, angesetzt auf irgendeine Bandbeschriftung, feststellt („entscheidet"), ob die Turingmaschine anhält oder nicht.

Turingberechenbarkeit + Church'sche These

Die Bedeutung der Turingmaschinen bezüglich der Berechenbarkeit liegt in der 1936 von A. Church formulierten Church'schen These:

> **Jede im intuitiven Sinne berechenbare Funktion ist Turing-berechenbar.**

Dabei bedeutet "im intuitiven Sinn", daß das Verfahren von einer genügend großen Zahl von Menschen als berechenbar anerkannt wird. Dies ist keine mathematische Formulierung, und somit kann die Church'sche These auch nicht mathematisch bewiesen werden. Sie konnte aber bislang auch nicht widerlegt werden,denn anschaulich besagt die Church'sche These:

> **Zu jedem algorithmischen Verfahren (unabhängig von der Darstellungsart) läßt sich eine Turingmaschine konstruieren, die die gleiche Funktion berechnet.**

Weitere Möglichkeiten zur Präzisierung des Algorithmenbegriffs berechenbarer Funktionen: Registermaschinen, μ -rekursive Funktionen, formale Logik (Prolog)

Linear beschränkte Automaten

Die Definition einer Turingmaschine ging von einem unendlich langen Arbeitsband aus.

Definition:

Eine Turingmaschine heißt **linear beschränkt** (oder ein linear beschränkter Automat, LBA), wenn sie im Laufe ihrer Arbeit nie den Bereich des Arbeitsbandes verläßt, auf dem das Eingabewort steht.

Bemerkung: Eine linear beschränkte Turingmaschine muß eine Möglichkeit haben, das Ende des Eingabewortes zu erkennen.

Satz: Die Menge der von nicht-deterministischen, linear beschränkten Automaten akzeptierten Sprachen ist genau die Menge der kontextsensitiven Sprachen. (vgl. Kapitel „formale Sprachen“)

LBA-Problem: Es ist bis heute nicht bekannt, ob es zu jedem nicht-deterministischen LBA einen deterministischen LBA gibt, der dieselbe Sprache akzeptiert.