5.4 Endliche Automaten

Ein **endlicher Automat** ist ein mathematisches Modell eines Systems mit Ein- und Ausgaben. Ein solches System befindet sich immer in einem **internen Zustand**.

Beispiele

- Ein Register mit *n* Binärstellen befindet sich in einem von 2^n möglichen Zuständen.
- Ein Zigarettenautomat merkt sich die bisher eingeworfene Geldsumme als internen Zustand.
- Lexikalische Analysatoren von Compilern kann man als endliche Automaten modellieren.
- Auch ein Computer ist ein endlicher Automat. Allerdings ist hier das Modell wegen der großen Anzahl möglicher interner Zustände nicht besonders hilfreich.

Grundbegriffe

Ein **endlicher Automat** ist ein Fünftupel

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

Z = Menge der **Zustände**

E = Menge der **Eingabezeichen**

$\delta : Z \times E \rightarrow Z$ **Übergangsfunktion**

$z_0 \in Z$ **Anfangszustand**

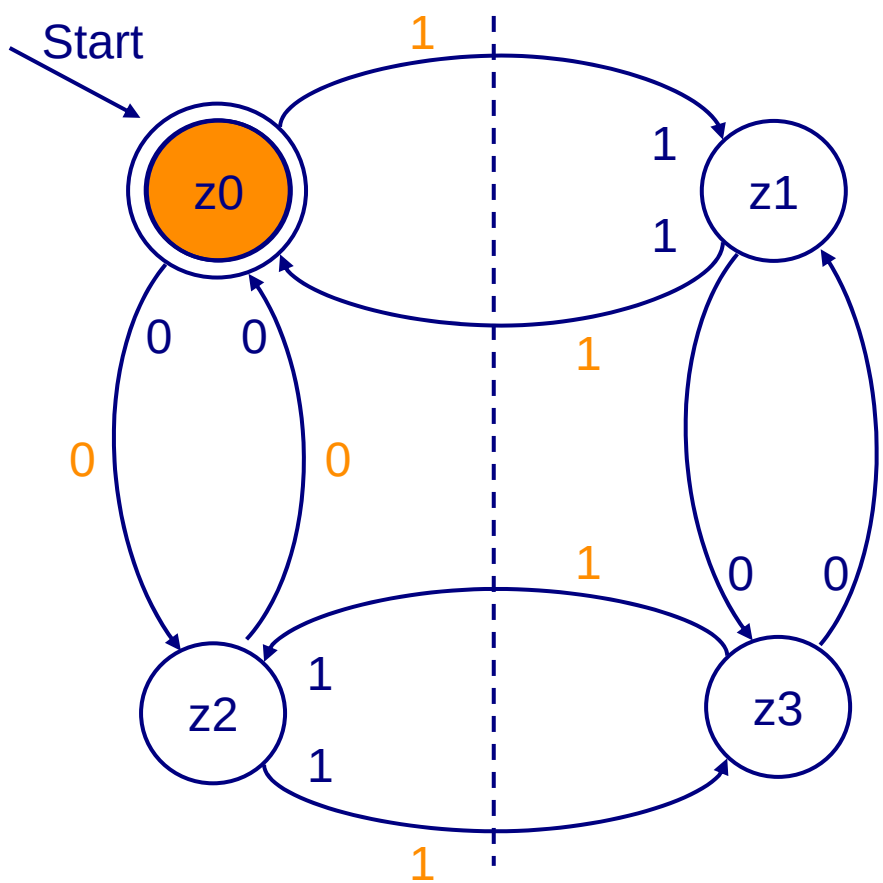
$F \subseteq Z$ **Menge der Endzustände.**

Da die Zustandsübergangsfunktion δ ein Paar (*z*,*a*) auf **genau einen** Folgezustand abbildet, sprechen wir auch von einem **deterministischen** endlichen Automaten, abgekürzt **DEA**.

Diagrammdarstellung

In der Regel veranschaulicht man endliche Automaten in Form von Diagrammen. Kreise stellen die Zustände dar, Pfeile zwischen den Kreisen repräsentieren die Übergänge (Transitionen). Jeder Übergang wird mit dem Eingabezeichen beschriftet, das ihn auslöst.

Beispiel 1: Zustandsdiagramm für einen DEA



Der Startzustand ist markiert. Der Doppelkreis bezeichnet den Endzustand.

Man überlege: **Welche speziellen Folgen von Nullen und Einsen akzeptiert dieser Automat?**

Ausgabe:

0 1 1 0 1 1

Zustandstabellen

Statt durch Graphen können Automaten auch durch **Zustandstabellen** beschrieben werden.

Wir betrachten wieder den Automaten aus Beispiel 1:

$$M = (Z, E, \delta, z_{0,} F)$$

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

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

$$F = \{z_0\}$$

Ausgangs-zustand	Eingabe	
	0	1
z_0	z_2	z_1
z_1	z_3	z_0
z_2	z_0	z_3
z_3	z_1	z_2

Erweiterung für Zeichenketten

Häufig möchte man Automaten aufschreiben, die nicht nur einzelne Zeichen, sondern ganze Zeichenketten als Eingabe akzeptieren. Man erweitert dazu die Übergangsfunktion für Zeichenketten wie folgt:

$\hat{\delta} : Z \times E' \rightarrow Z$, *E'* = Menge der Eingabe-Zeichenketten

$\delta(z, \varepsilon) = z$, ε = leere Zeichenkette

$\hat{\delta}(z, wa) = \delta(\hat{\delta}(z, w), a)$, *w* Zeichenkette (rekursive Definition)
In Zukunft schreiben wir auch für Zeichenketten nur noch δ statt $\hat{\delta}$.

Wir sagen: Ein Zeichenkette *s* wird vom Automaten **akzeptiert**, wenn $\delta(z_0, s) \in F$, wenn also *s* einen Pfad von Transitionen beschreibt, der mit dem Startzustand z_0 beginnt und in einem Endzustand des Automaten endet.

Endliche Automaten und Sprachen

Eine (formale) Sprache *L* ist eine Menge von zulässigen Zeichenketten. Die von einem Automaten *M* **akzeptierte Sprache** ist dann

$$L(M) = \{x \in E' \mid \delta(z_0, x) \in F\}$$

Nichtdeterministische endliche Automaten

Wenn wir in der Zustandsübergangsfunktion δ vorsehen, dass für ein Eingabesymbol *a* **mehr als ein** Folgezustand möglich ist, so sprechen wir von einem **nichtdeterministischen endlichen Automaten** (**NEA)**.

δ ist dann eine Abbildung auf **Mengen von Zuständen**.

Bei einer Eingabe von *a* wird *zufällig* einer der möglichen Folgezustände ausgewählt.

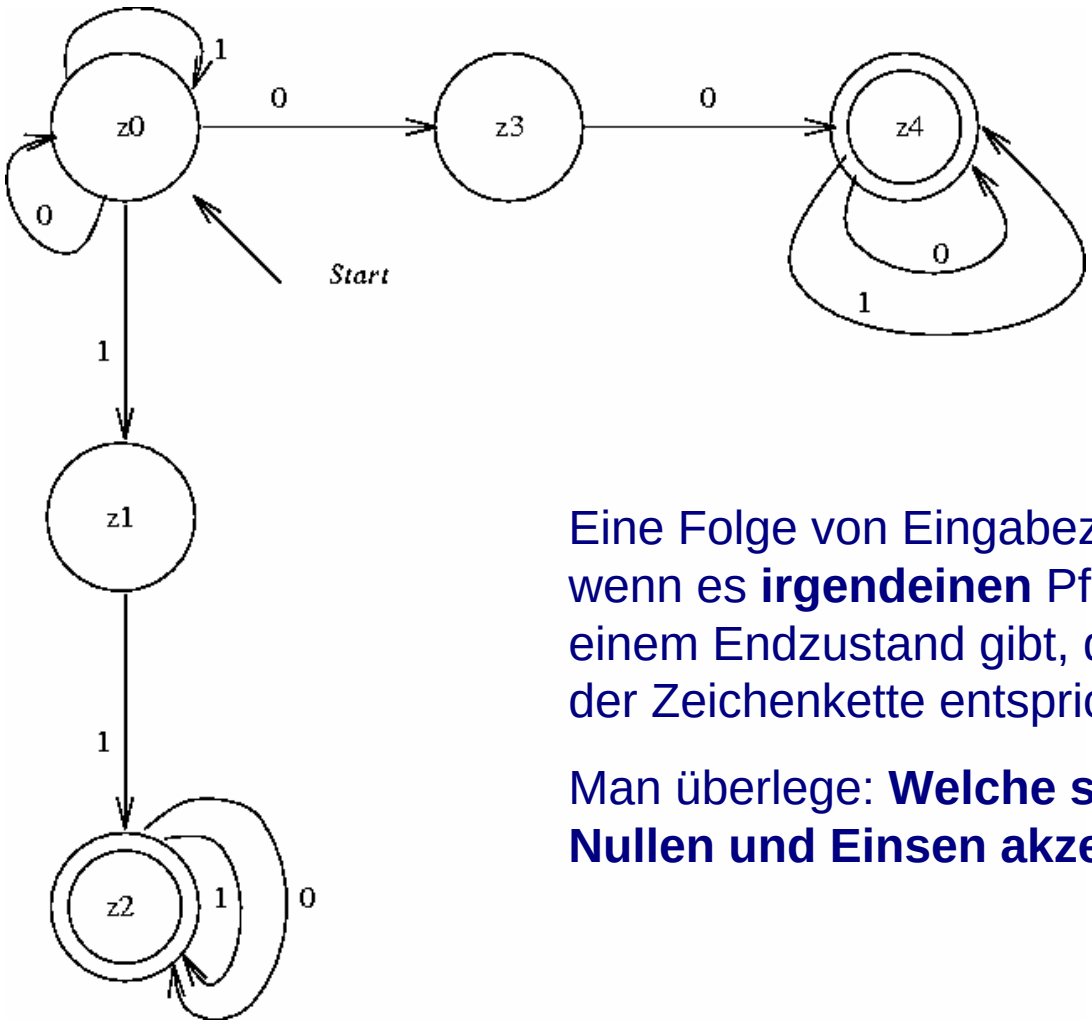
Formal ist ein NEA ein Quintupel (Z, E, δ, z_0, F), das genauso wie beim DEA definiert ist, nur gilt nun für δ:

$$\delta : Z \times E \rightarrow 2^Z$$

2^Z ist die Potenzmenge, also die Menge aller Teilmengen von *Z*. δ ist jetzt eine Abbildung von *Z x E* auf 2^Z. Das heißt, ein Eingabezeichen kann zu keinem, einem oder mehreren Folgezuständen des Automaten führen.

Wir werden sehen, dass zu jedem NEA ein DEA existiert, der die gleiche Sprache akzeptiert. Natürlich ist auch jeder DEA ein NEA.

Beispiel 2: Zustandsdiagramm für einen NEA



Eine Folge von Eingabezeichen wird akzeptiert, wenn es **irgendeinen** Pfad vom Startzustand zu einem Endzustand gibt, dessen Transitionsfolge der Zeichenkette entspricht.

Man überlege: **Welche speziellen Folgen von Nullen und Einsen akzeptiert dieser NEA?**

Zustandstabelle zum NEA aus Beispiel 2 (1)

Ausgangs-Zustand	Eingaben	
	0	1
z_0	$\{z_0, z_3\}$	$\{z_0, z_1\}$
z_1	$\varnothing$	$\{z_2\}$
z_2	$\{z_2\}$	$\{z_2\}$
z_3	$\{z_4\}$	$\varnothing$
z_4	$\{z_4\}$	$\{z_4\}$

Zustandstabelle zum NEA aus Beispiel 2 (2)

Erweiterung von δ für Zeichenketten als Eingabe

Eine Zeichenkette $s = a_1 a_2 a_3 ... a_n$ wird vom NEA (analog zum DEA) akzeptiert, wenn es **irgendeine** Transitionsfolge vom Startzustand z_0 aus gibt, die in einem Endzustand aus *F* endet.

$$\hat{\delta}(z, \varepsilon) = \{z\}$$

$$\hat{\delta}(z, wa) = \{p \mid \exists r \in \hat{\delta}(z, w) : p \in \delta(r, a)\}$$

In Zukunft schreiben wir auch hier δ statt $\hat{\delta}$.

Die Äquivalenz von NEA und DEA (1)

Um die Äquivalenz von NEA und DEA zu zeigen, konstruieren wir zu einem beliebigen NEA einen DEA, der die gleiche Sprache akzeptiert.

Sei $M = (Z, E, \delta, z_0, F)$ ein NEA. Wir definieren dann einen DEA wie folgt:

$Z' = 2^Z$ (die Potenzmenge von *Z*)

$E' = E$

$F' = \{ z \mid z \in Z' \wedge z \cap F \neq \varnothing \}$

Z', die Zustandsmenge des äquivalenten DEA, ist die Menge aller Teilmengen des NEA; wir repräsentieren also jede mögliche Teilmenge von Zuständen des NEA durch einen eigenen Zustand im DEA. Die Elemente aus *Z'* bezeichnen wir mit $[z_1 z_2 z_3 ... z_i]$, $z_1, z_2, ... z_i \in Z$

Die Äquivalenz von NEA und DEA (2)

$z_0' = [z_0]$

$\delta'([z_1,...,z_i],a) = [p_1,...,p_i]$ im DEA genau dann, wenn

$\delta(\{z_1,...,z_i\},a) = \{p_1,...,p_i\}$ im NEA

Gemäß Konstruktion ergibt sich dabei auch:

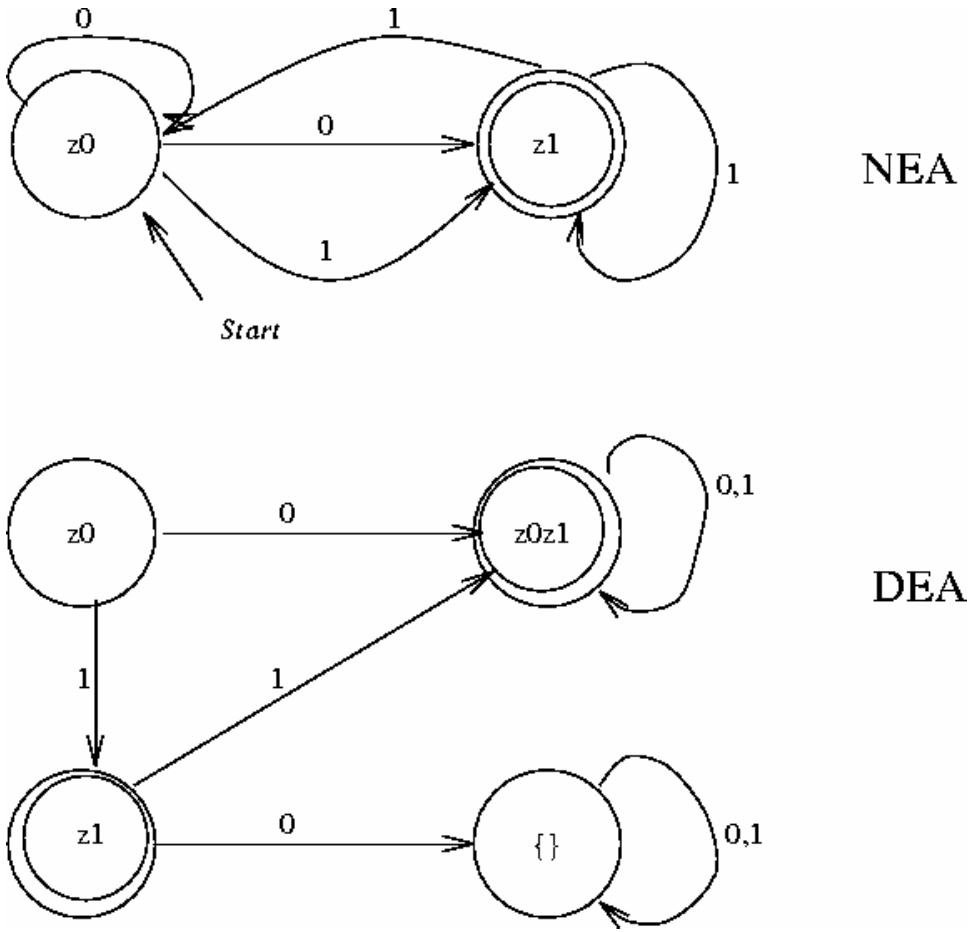
$\delta'(z_o',w) \in F'$ im DEA genau dann, wenn

$\exists z \in \delta(z_0,w): z \in F$ im NEA.

Der Beweis erfolgt durch Induktion über die Länge der Eingabe-Zeichenkette.

Beispiel 3: Äquivalenz von NEA und DEA

Man konstruiere zu dem folgenden NEA den zugehörigen DEA:



Endliche Automaten mit spontanen Übergängen

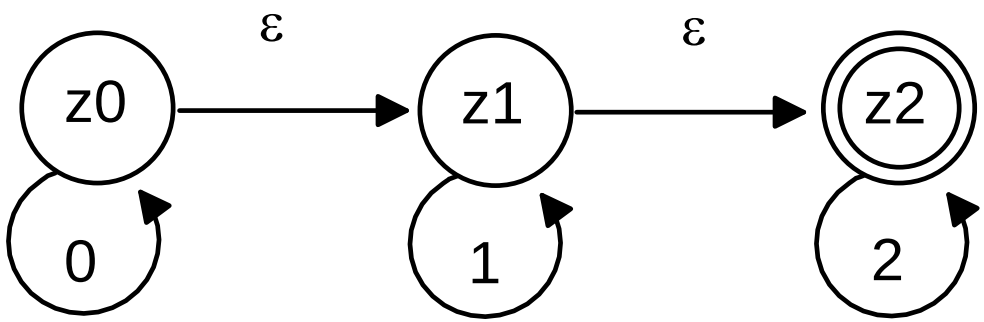
Bei der Programmentwicklung ist der Einsatz wieder verwendbarer Komponenten eine wichtige Strategie. Man kann viel Arbeit sparen, wenn man ein Programm aus bereits vorhandenen Unterprogrammen (Modulen) zusammen setzt.

Auch endliche Automaten kann man zu größeren, komplizierteren kombinieren. Diesen Zusammenbau kann man dadurch vereinfachen, dass man erlaubt, dass ein NEA auch **ohne Eingabe** seinen Zustand wechselt.

Um dies zu ermöglichen, definieren wir die Zustandsübergangsfunktion δ auch für die leere Zeichenkette ε. Das bedeutet, dass der Automat spontane Übergänge ausführen kann.

Eine Zeichenkette wird akzeptiert, wenn es ausgehend vom Startzustand einen Pfad zu einem Endzustand gibt, der auch mit ε markierte Kanten enthalten kann.

Beispiel 4: Endlicher Automat mit ε-Übergängen



Der Automat akzeptiert die Sprache, deren Worte aus (möglicherweise leeren) Folgen von Nullen, gefolgt von (möglicherweise leeren) Folgen von Einsen, gefolgt von (möglicherweise leeren) Folgen von Zweien bestehen.

Definition NEA mit ε-Übergängen

Ein NEA mit ε-Übergängen ist ein Quintupel

$$(Z, E, \delta, z_0, F),$$

dessen Komponenten wie üblich definiert sind, außer dass δ eine Abbildung von $Z \times (E \cup \{\varepsilon\})$ nach 2^Z ist.

$\delta(z, a)$ besteht aus den Zuständen *z*, für die eine mit *a* markierte Transition von *z* existiert, wobei *a* aus $E \cup \{\varepsilon\}$ ist.

Zustandstabelle für Beispiel 4

Ausgangs-zustand	Eingabe			
	0	1	2	ε
z_0	$\{z_0\}$	$\varnothing$	$\varnothing$	$\{z_1\}$
z_1	$\varnothing$	$\{z_1\}$	$\varnothing$	$\{z_2\}$
z_2	$\varnothing$	$\varnothing$	$\{z_2\}$	$\varnothing$

Die ε-Hülle

Definition

Die ε-Hülle eines Zustands *z* (ε-closure) ist die Menge aller Folgezustände von *z*, die ausschließlich durch ε-Übergänge erreicht werden können.

In Beispiel 4 ist die ε-Hülle von z_0 die Menge $\{z_0, z_1, z_2\}$.

z_0 ist in dieser Menge, da der Pfad von z_0 nach z_0 keine Kanten besitzt und damit trivialerweise unsere Bedingung erfüllt.

z_1 ist enthalten, weil der Pfad (z_0, z_1) nur ε-Übergänge enthält.

z_2 ist enthalten, weil (z_0, $z_{1,}$, z_2) nur mit ε markierte Kanten enthält.

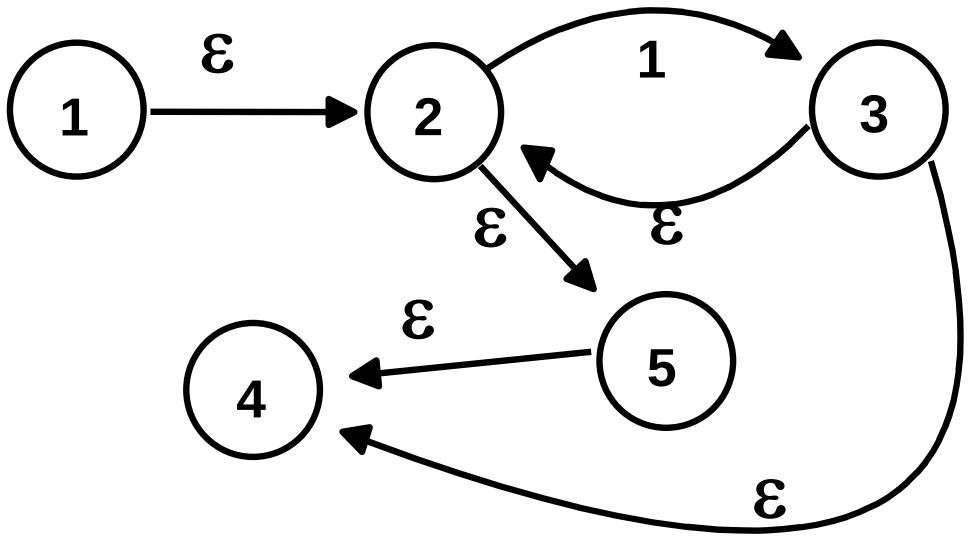
Die Berechnung der ε-Hülle

Wir geben einen rekursiven Algorithmus zur Berechnung der ε-Hülle an. Sei m eine Zustandsmenge. Dann berechnen wir die ε-Hülle(m) wie folgt:

Algorithmus ε-Hülle(*m*)

```
Falls m == ∅
dann  Ergebnismenge = ∅
sonst
      Wähle ein z ∈ m
      Berechne  m1 = { z' | z' = δ(z, ε) }
      Ergebnismenge = Ergebnismenge
            ∪ m1
            ∪ ε-Hülle(m1 \ m)
            ∪ ε-Hülle(m \ {z} )
```

Beispiel zur Berechung der ε-Hülle



ε-Hülle({1}) = {1,2,4,5}

Die Äquivalenz von NEA mit und ohne ε-Übergänge

Die ε -Übergänge sind für die Kombination von NEAs zwar hilfreich, sie erhöhen aber nicht die prinzipiellen Fähigkeiten dieser Automaten. Um dies zu beweisen, zeigt man, dass man zu jedem NEA **mit** ε -Übergängen einen Automaten ohne ε -Übergänge konstruieren kann, der die gleiche Sprache akzeptiert.

Auf die Details der Konstruktion des NEA ohne ε -Übergänge und den Äquivalenzbeweis sei hier verzichtet.

Äquivalenter NEA ohne ε -Übergänge zum Bsp. 4 (1)

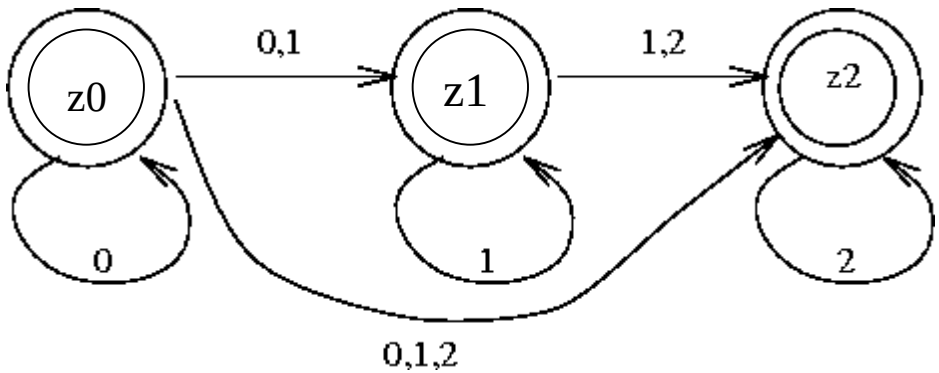
Zustandsübergangstabelle ohne ε für Beispiel 4

Ausgangs-zustand	Eingabe		
	0	1	2
z_0	$\{z_0, z_1, z_2\}$	$\{z_1, z_2\}$	$\{z_2\}$
z_1	$\varnothing$	$\{z_1, z_2\}$	$\{z_2\}$
z_2	$\varnothing$	$\varnothing$	$\{z_2\}$

Man beachte, dass zur Berechnung der Tabelleneinträge für Zustand z_i ε -Hülle(z_i) verwendet wurde, wo in der ursprünglichen Tabelle ε eine zulässige Eingabe war.

Äquivalenter NEA ohne ε-Übergänge zum Bsp. 4 (2)

Diagramm



Man beachte, dass jetzt *z0* und *z1* zulässige Endzustände sind.

Endliche Automaten mit Ausgabe

Die bisher betrachteten Automaten können auf eine Eingabe nur mit *akzeptiert* oder *nicht akzeptiert* reagieren – je nachdem, ob ein Endzustand aus *F* erreicht wird oder nicht. Man könnte auch sagen, ihr **Ausgabealphabet** besteht nur aus {0,1}.

Andere Typen von Automaten besitzen ein eigenes Ausgabealphabet und eine zusätzliche **Ausgabefunktion:**

- Ein **Moore-Automat** hat eine Ausgabefunktion, die jedem **Zustand** eine Ausgabe zuordnet.
- Ein **Mealy-Automat** hat eine Ausgabefunktion, die **bei jeder Transition** eine Ausgabe erzeugt.

Wir können zeigen, dass beide Typen grundsätzlich äquivalent sind.

Für endliche Automaten mit Ausgabe gibt es viele Anwendungen in der Informatik.

Moore-Automaten (1)

Ein **Moore-Automat** ist ein Sechs-Tupel

$$M = (Z, E, A, \delta, \lambda, z_0)$$

mit Z, E, δ, z_0 definiert wie beim DEA, *A* ist das Ausgabealphabet (die Menge der ausgebbaren Zeichen) und $\lambda : Z \rightarrow A$ eine Ausgabefunktion.

Eine Ausgabe von *M* als **Antwort** auf die Eingabe

$a_1 ..... a_n, n \geq 0$ ist

$\lambda(z_0)\lambda(z_1)...\lambda(z_n)$ mit

. $\delta(z_{i-1}, a_i) = z_i, 1 \leq i \leq n$

Auf die leere Eingabe liefert der Moore-Automat $\lambda(z_0)$.

Moore-Automaten (2)

Bemerkung

Ein DEA kann als Moore-Automat mit *A={0,1}* betrachtet werden, der für $z \in F$ die Ausgabe $\lambda(z) = 1$ liefert, für alle anderen *z* liefert er $\lambda(z) = 0$.

Beispiel: Restklassen mod 5 (1)

Wir betrachten binäre Zeichenketten und berechnen die Restklassen modulo 5. Hat die bisherige Zeichenkette den Wert *i* und folgt eine 0, so hat die resultierende Zeichenkette den Wert 2*i*, folgt eine 1, so hat sie den Wert 2*i*+1.

Für Folgeziffer 0 gilt:

i mod 5 = 0 $\Rightarrow$ 2i mod 5 = 0

i mod 5 = 1 $\Rightarrow$ 2i mod 5 = 2

i mod 5 = 2 $\Rightarrow$ 2i mod 5 = 4

i mod 5 = 3 $\Rightarrow$ 2i mod 5 = 6 mod 5 = 1

i mod 5 = 4 $\Rightarrow$ 2i mod 5 = 8 mod 5 = 3

Beispiel: Restklassen mod 5 (2)

Für Folgeziffer 1 gilt:

$i \bmod 5 = 0 \Rightarrow 2i+1 \bmod 5 = 1$

$i \bmod 5 = 1 \Rightarrow 2i+1 \bmod 5 = 3$

$i \bmod 5 = 2 \Rightarrow 2i+1 \bmod 5 = 5 \bmod 5 = 0$

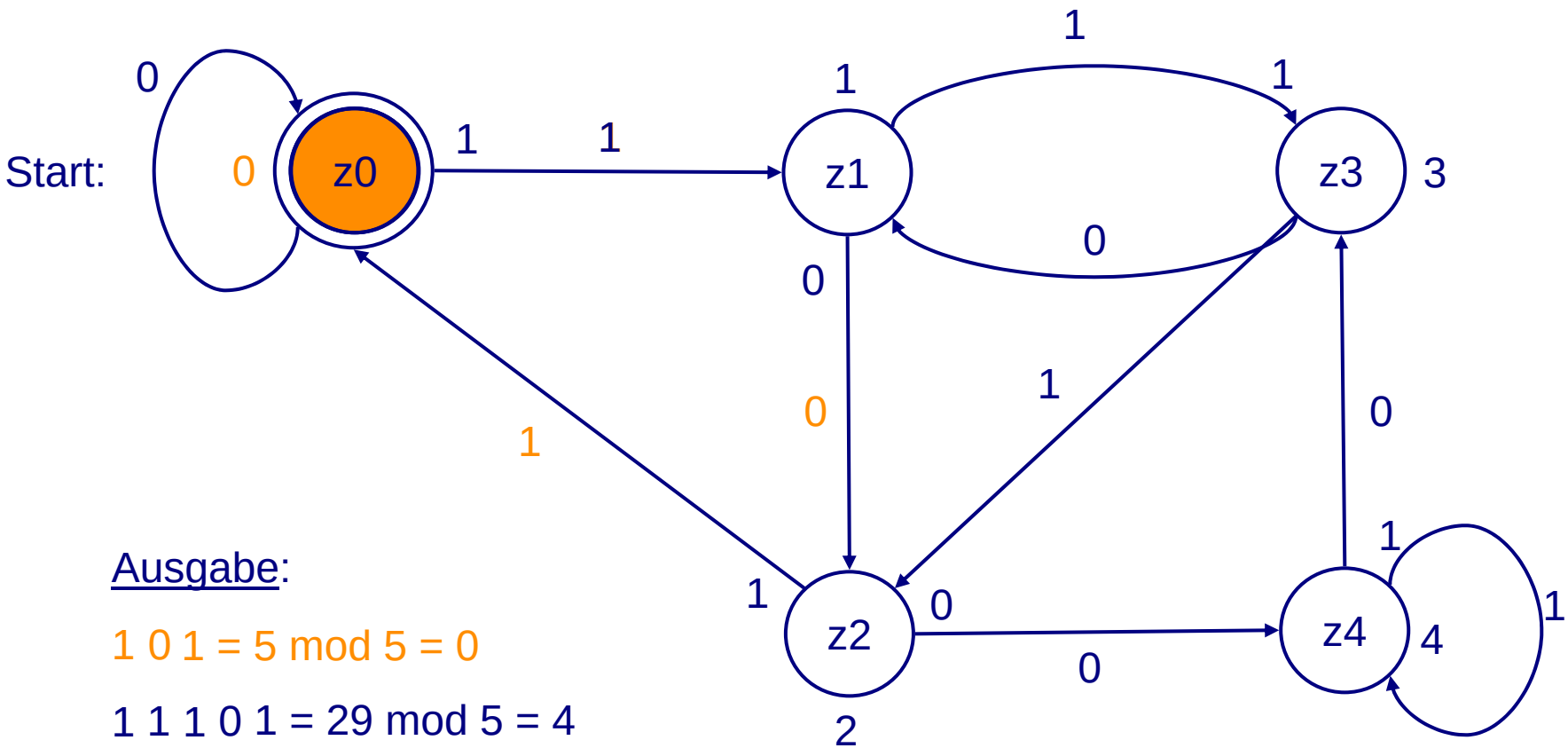
$i \bmod 5 = 3 \Rightarrow 2i+1 \bmod 5 = 7 \bmod 5 = 2$

$i \bmod 5 = 4 \Rightarrow 2i+1 \bmod 5 = 9 \bmod 5 = 4$

Wir benötigen also eine Maschine mit 5 Zuständen.

Moore-Automat für die Restklassen mod 5

Die Zustände sind zusätzlich mit den jeweiligen Ausgaben beschriftet.



Ausgabe:

1 0 1 = 5 mod 5 = 0

1 1 1 0 1 = 29 mod 5 = 4

Mealy-Automaten

Ein **Mealy-Automat** ist wie ein Moore-Automat ein Sechstupel

$$M = (Z, E, A, \delta, \lambda, z_0),$$

aber die Ausgabe ist jetzt abhängig von aktuellem **Zustand und Eingabe:**

$$\lambda : Z \times E \rightarrow A$$

Die Ausgabe wird hier also mit dem **Zustandsübergang** assoziiert.

Die **Ausgabe** vom *M* als Antwort auf die Eingabe $a_1 ... a_n$ ist $\lambda(z_0, a_1)\lambda(z_1, a_2)....\lambda(z_{n-1}, a_n)$, wobei $\delta(z_{i-1}, a_i) = z_i, 1 \leq i \leq n$.

Auf die leere Eingabe reagiert auch ein **Mealy-Automat** mit der leeren Ausgabe.

Äquivalenz von Moore- und Mealy-Automaten (1)

$T_M(w)$ sei die Ausgabe des Automaten *M* als Reaktion auf die Eingabezeichenkette *w*.

Bei einem Mealy-Automaten ist die Länge der Ausgabe um eins kleiner als bei einem Moore-Automaten, da die durchlaufene Kette der Zustände immer um eines größer ist als die durchlaufene Kette der Transitionen. Wir möchten aber dennoch die Äquivalenz von Mealy- und Moore-Automaten definieren.

M sei ein Mealy-Automat und *M'* ein Moore-Automat. Wenn wir mit *T*(*w*) die Ausgabe eines Automaten bei Eingabe der Zeichenkette *w* bezeichnen und wenn a_0 die Ausgabe des Zustands z_0 in *M'* ist, so definieren wir:

M und *M'* sind **äquivalent**, wenn für alle Zeichenketten *w* gilt:

$$a_0 T_M(w) = T_{M'}(w)$$

Äquivalenz von Moore- und Mealy-Automaten (2)

Satz

Zu einem Moore-Automaten $M_1 = (Z, E, A, \delta, \lambda, z_0)$ existiert ein äquivalenter Mealy-Automat M_2.

Beweis

Wir setzen $M_2 = (Z, E, A, \delta, \lambda', z_0)$ mit $\lambda'(z,a) = \lambda(\delta(z,a))$ für alle Zustände z und Eingabezeichen a. Dann produzieren beide Automaten für dieselben Eingabe-Zeichenketten stets dieselben Ausgabe-Zeichenketten.

Äquivalenz von Moore- und Mealy-Automaten (3)

Satz

Zu einem Mealy-Automaten $M_1 = (Z, E, A, \delta, \lambda, z_0)$ existiert ein äquivalenter Moore-Automat M_2.

Beweis

$M_2 = (Z \times A, E, A, \delta', \lambda', [z_0, b_0])$, b_0 beliebig aus A.

Die Zustände von M_2 sind Paare $[z,b]$, wobei b die Ausgabe von M_1 beim Übergang in den Zustand z ist. Wir setzen

$$\delta'([z,b],a) = [\delta(z,a), \lambda(z,a)] \text{ und}$$

$$\lambda'([z,b]) = b.$$

Dann sieht man, dass für beliebige Eingabe-Zeichenketten die Ausgaben von M_1 und M_2 identisch sind.

Ein Mealy-Automat für das Bit-Stuffing-Verfahren (1)

Beispiel

Zur Anwendung von fehlererkennenden und fehlerkorrigierenden Codes in der Kommunikationstechnik muss der binäre Datenstrom in einzelne Rahmen unterteilt werden.

Problem

Wie kann man solche Rahmen im kontinuierlichen Bitstrom begrenzen, wenn grundsätzlich **jedes beliebige** Bitmuster in den Nutzdaten vorkommen kann?

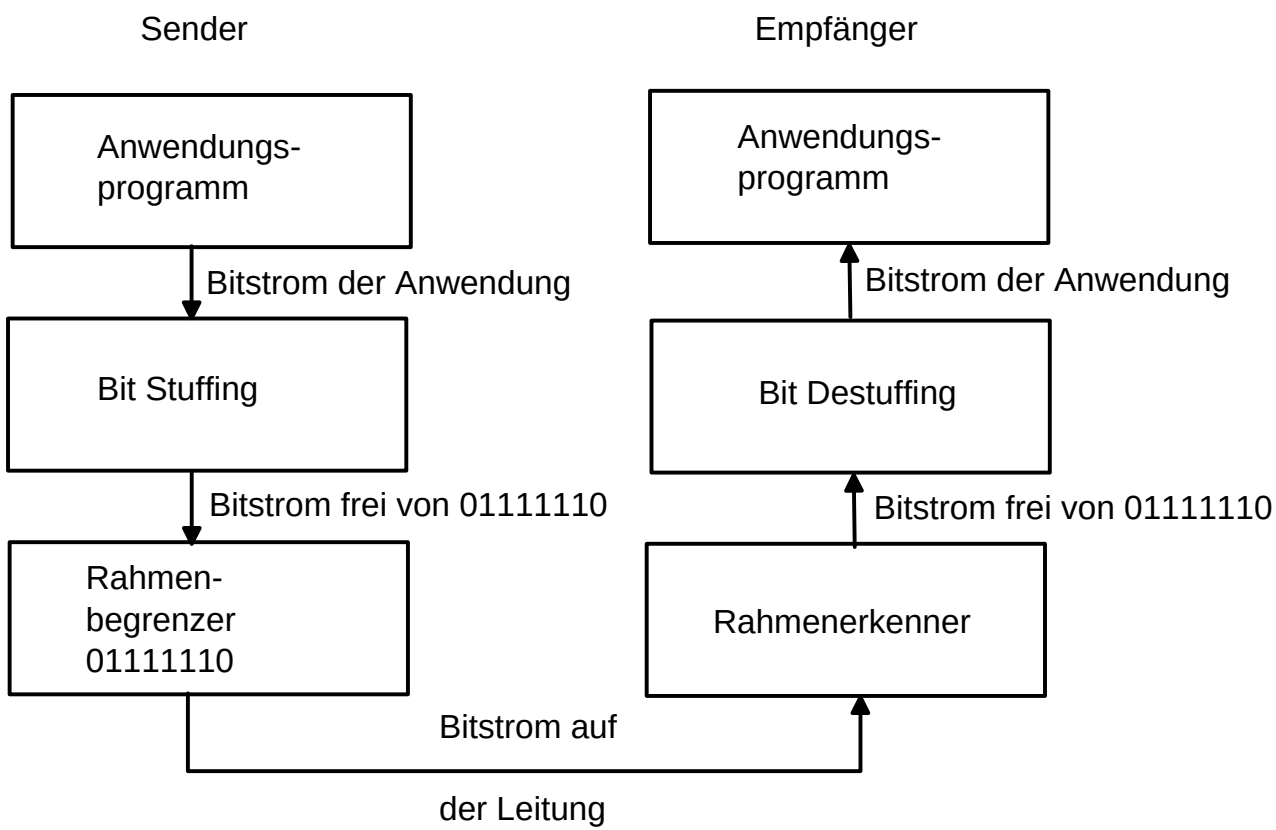
Ein Mealy-Automat für das Bit-Stuffing-Verfahren (2)

Lösung: Bit Stuffing (Bitstopfen)

Als Begrenzer wählt man 01111110. Der Sender fügt nach fünf Einsen im Nutzdatenstrom **immer** eine 0 ein. Wenn der Empfänger nach fünf Einsen eine Null sieht, entfernt er diese aus dem Datenstrom.

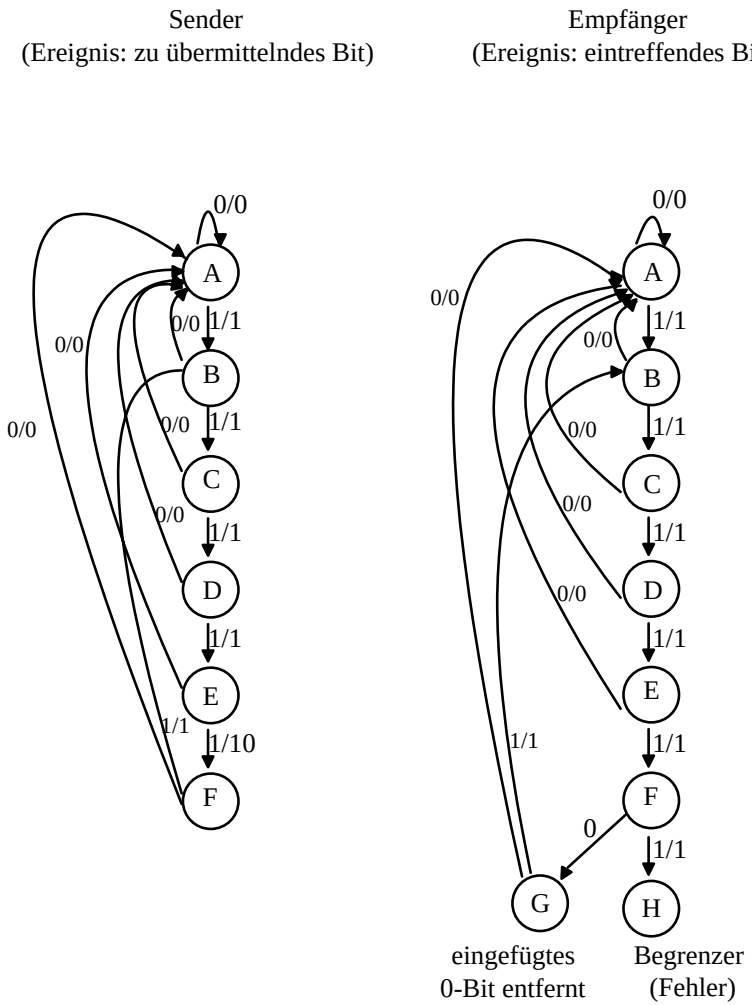
Sender	0	1	1	1	1	1		1	0	1	0	1	1	1	1	1		0	1	0
Leitung	0	1	1	1	1	1	0	1	0	1	0	1	1	1	1	1	0	0	1	0
Empfänger	0	1	1	1	1	1		1	0	1	0	1	1	1	1	1		0	1	0

Die Funktionsweise des Bit Stuffing



Ein Mealy-Automat für das Bit Stuffing

Mealy-Automaten für Sender und Empfänger bei einer Übertragung mit Bitstopfen



Notation: e/a
e = Ereignis
a = Ausgabe

Die Minimierung endlicher Automaten

In den vorangegangenen Abschnitten hatten wir die Äquivalenz von endlichen Automaten und regulären Sprachen betrachtet. Die Automaten, die wir auf diese Weise erhalten, besitzen eine Vielzahl von Zuständen, so dass die Frage entsteht, ob man sie nicht auch vereinfachen kann.

Wir werden nun einen Algorithmus vorstellen, der zu einem endlichen Automaten einen Automaten konstruiert, der dieselben Eingaben akzeptiert, aber die minimale dafür nötige Anzahl von Zuständen besitzt. Wir betrachten dabei zunächst nur deterministische endliche Automaten ohne Ausgabe. Da man zu jedem NEA einen äquivalenten DEA konstruieren kann, ist dies keine Einschränkung.

Außerdem soll $\delta(z,a)$ für alle Paare (z,a) vollständig definiert sein. Notfalls ist ein Fehlerzustand nachzurüsten.

Wir werden auch sehen, dass dieser Automat (bis auf Isomorphie) eindeutig bestimmt ist.

Aber um dieses Ergebnis zu beweisen, benötigen wir noch einige Tatsachen, die wir uns jetzt ansehen werden.

Die Äquivalenzrelation R_L (1)

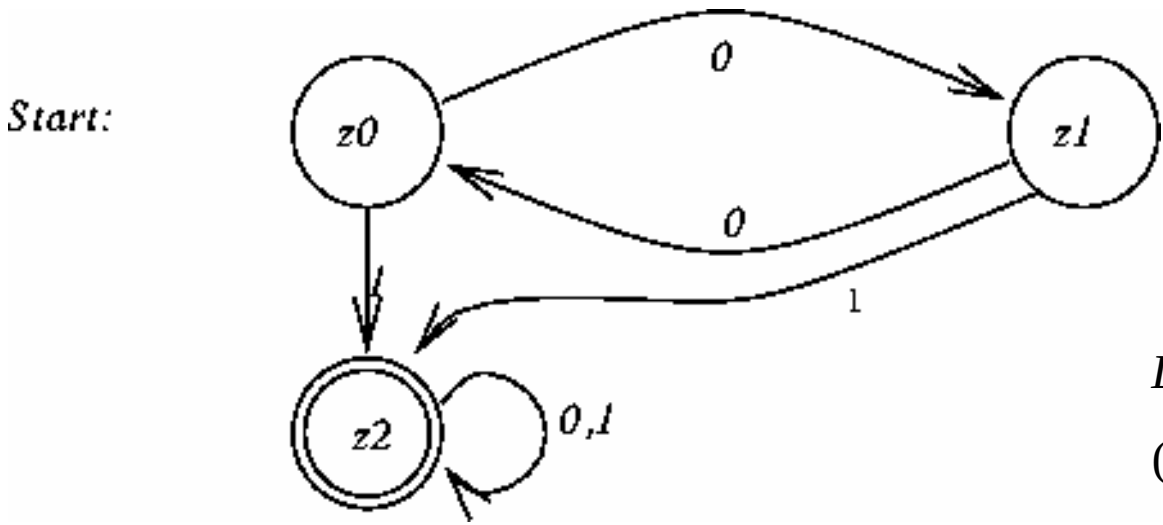
Wir assoziieren mit einer beliebigen Sprache L die Äquivalenzrelation R_L:

$$R_L = \{(x, y) \mid \forall z \in L : xz \in L \Leftrightarrow yz \in L\}$$

Im schlechtesten Fall besteht jede Äquivalenzklasse nur aus einem Element, denn $\forall x, z \in L : xz \in L \Leftrightarrow xz \in L\}$ gilt immer.

Die Äquivalenzrelation R_L (2)

Beispiel:



$$L_M = 0*1(0+1)*$$

$$(10,01) \in R_L$$

$$(0,000) \in R_L$$

$$(00,01) \notin R_L$$

Die Äquivalenzrelation R_M (1)

Sei $M = (Z, E, \delta, z_0, F)$ ein DEA. Wir definieren eine Relation R_M:

$$R_M = \{(x, y) \in E^* \times E^* : \delta(z_0, x) = \delta(z_0, y)\}$$

R_M ist offenbar eine **Äquivalenzrelation**, denn Reflexivität, Transitivität und Symmetrie folgen aus den Eigenschaften der Gleichheitsbeziehung.

R_M teilt E^* in Äquivalenzklassen auf, und zwar gehört zu jedem von z_0 erreichbaren Zustand eine Äquivalenzklasse.

Es gilt:

$$\delta(z_0, xz) = \delta(\delta(z_0, x), z) = \delta(\delta(z_0, y), z) = \delta(z_0, y_z)$$

also

$$xR_M y \Rightarrow \forall z \in E^* : xzR_M yz$$

Die Äquivalenzrelation R_M (2)

Die Relation bleibt erhalten, wenn man *x* und *y* von rechts mit *z* konkateniert. Solche Relationen heißen **rechts-invariant.**

Nun können wir einen wichtigen Satz formulieren, der die Grundlage für die Minimierung endlicher Automaten darstellt.

Der Satz von Myhill-Nerode

Die folgenden Aussagen sind äquivalent:

1. Die Menge $L \subseteq E^*$ wird von einem endlichen Automaten akzeptiert.

2. *L* ist die Vereinigung von einigen Äquivalenzklassen einer rechts-invarianten Äquivalenzrelation mit endlichem Index.

3. Die Äquivalenzrelation
   $R_L = \{(x, y) \in E^* \times E^* \mid \forall z \in E^* : xz \in L \Leftrightarrow yz \in L\}$
   hat einen endlichen Index, d. h. nur endlich viele Äquivalenzklassen.

Beweis

Zum Beweis sei auf die Literatur verwiesen, z. B.: **J.E. Hopcroft, J.D. Ullman**: Introduction to Automata Theory, Languages and Computation. Addison Wesley, 1979.

Minimale Automaten sind isomorph

Satz

Der Minimalautomat, der *L* akzeptiert, ist bis auf Isomorphie eindeutig bestimmt.

Beweis

Zum Beweis sei auf die Literatur verwiesen, siehe oben.

Die Minimierung von Mealy-Automaten

Das Verfahren zur Minimierung von Automaten mit Ausgabefunktion ist ganz analog. Bei Mealy-Automaten *M*, *M'* ist von Interesse, ob für die gleiche Eingabe irgendwann eine unterschiedliche Ausgabe erzeugt wird. Falls dies nie der Fall ist, betrachten wir die beiden Automaten als äquivalent.

Definition

Zwei **Mealy-Automaten** $M = (Z, E, A, \delta, \lambda, z_0)$ und $M' = (Z', E, A, \delta', \lambda', z'_0)$ heißen **äquivalent**, wenn sie für gleiche Eingaben immer gleiche Ausgaben erzeugen:

$$M \sim M' \Leftrightarrow (\forall w \in E^* : \lambda(z_0, w) = \lambda'(z'_0, w))$$

Wir suchen also zu dem zu minimierenden Automaten *M* den Automaten *M'*, der die wenigsten Zustände hat. Da uns zu einer Eingabe $w \in E^*$ nur $\lambda(z_0, w)$ interessiert, definieren wir eine Äquivalenzrelation auf *Z*, die alle Zustände miteinander assoziiert, die zu einer beliebigen Eingabe die gleiche Ausgabe erzeugen.

Die Minimierung des Bit-Stuffing-Automaten

Wir betrachten den Mealy-Automaten für das Bit Stuffing als Beispiel für die Minimierung. Wir stellen fest, dass er nicht minimal ist: Bei gleicher Eingabe produziert der Zustand F dasselbe Folgeverhalten wie der Zustand A, wobei jeweils auch dieselbe Ausgabe erfolgt:

- Bei Eingabe einer 0 wird eine 0 ausgegeben, der Folgezustand ist A.
- Bei Eingabe einer 1 wird eine 1 ausgegeben, der Folgezustand ist B.

Wir können deshalb den Zustand F streichen! Die Transition 1/10 von E aus führt dann nach A, die Transition 0/0 wie bisher ebenfalls nach A. Wir haben also einen reduzierten Automaten erzeugt, der für beliebige Eingabe-Zeichenketten stets dasselbe Verhalten zeigt wie der ursprüngliche Automat, der also äquivalent ist.