Gliederung

Anmerkungen

In diesem Kapitel lernen wir einen Ansatz kennen, formale Sprachen zu beschreiben, indem man direkt ein Programm angibt, dass das Wortproblem für die zu beschreibende Sprache löst.

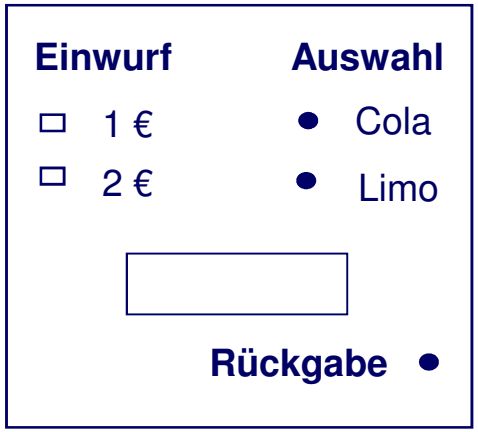
Wir müssen also festlegen,

- welche Programme wir aufschreiben können (also ein Berechnungsmodell einführen) und
- welche Sprache ein Programm in diesem Berechnungsmodell beschreibt (also für welche Sprache ein Programm das Wortproblem löst).

Doch beginnen wir erst einmal mit einem Beispiel für mögliche Abläufe in Systemen …

Beispiel

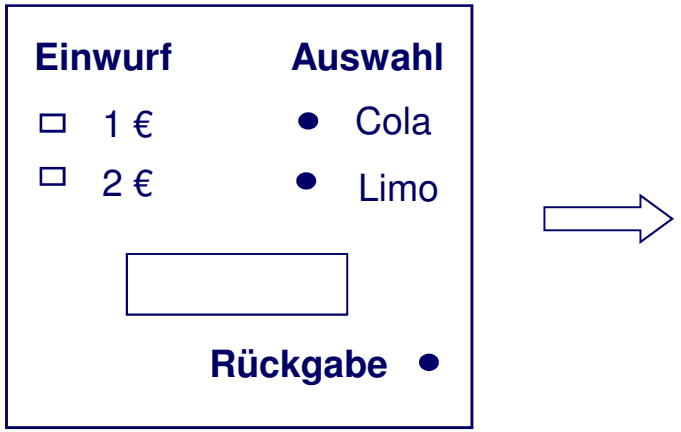
Benutzung eines Getränkeautomaten modellieren

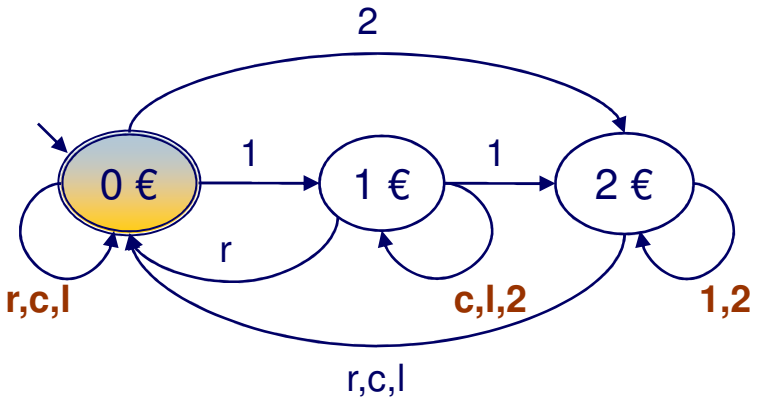


- Verkauf von Cola oder Limo für jeweils 2 €
- zulässige Münzen: 1 €, 2 €
- Möglichkeit zum Abbruch

Lösungsvorschlag

Eingaben: 1, 2, r (Rückgabe), c (Cola), l (Limo)
Zustände: 0 €, 1 €, 2 €





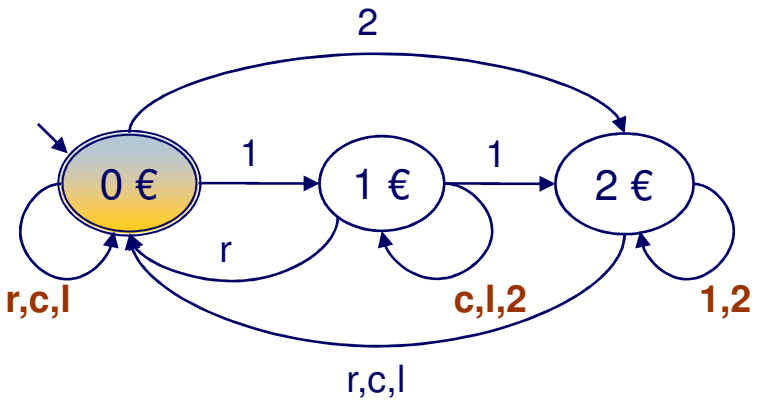
- *modelliert: nur Kunden-Eingaben und -Guthaben, ignoriert: Warenfluss (Ausgabe/Entnahme) und Geldrücknahme.*
- *In bestimmten Zuständen sind bestimmte Eingaben nicht möglich bzw. die entsprechenden Eingabeversuche bewirken nichts.*
- *In jedem Zustand kann jede Eingabe zumindest versucht werden.*

- Die abgeschlossene Kundenaktionsfolgen,
  d.h. vom Zustand „quitt“ (0 €) zum Zustand „quitt“ (0 €) …

  z.B.
  - ε
  - 11c
  - 21l
  - cc1cc1c
  - 1r2l
  - 2l2l2c

  (Gegenbeispiele:
  - 11
  - 2l1)

  bilden eine Sprache $\subseteq \{1,2,r,c,l\}^*$.



- *Das Automatendiagramm ermöglicht eine Prüfung, ob der Kunde eine Aktionsfolge aus der Sprache ausführt:*
  wenn von 0€ aus das Wort als Wegbeschreibung zu 0€ führt.

Gegenstand des Kapitels

- Wir schauen uns ein sehr einfaches Berechenbarkeitsmodell an, eigentlich das einfachste, das man in der Informatik betrachtet.
- Die Programme dieses Modells (so genannte endliche Automaten) werden benutzt, um bestimmte formale Sprachen zu beschreiben.
- Die Programme dieses Modells arbeiten in Echtzeit, d.h. die Eingabe wird einmal Zeichen für Zeichen von links nach rechts verarbeitet und sobald das letzte Zeichen verarbeitet wurde, steht das Ergebnis fest

Damit ist klar, dass diese Programme sehr effizient arbeiten.

Echtzeit / Nicht-Echtzeit (1)

Ein „kleines“ Spielproblem:

- Ein Schreib- und Formatier-Programm soll Zeichenketten außerhalb eines vorgegeben Wörterbuchs als „nicht akzeptiert“ markieren.
- Wir ignorieren hier einmal grammatische Abwandlungen, und gehen nur von einer festen Wörterliste aus, so dass z.B. neben „gehen“ auch „ging“ und „gegangen“ explizit im Wörterbuch steht.

Damit das funktioniert, muss natürlich „irgendwo im Programm“ das Wissen um die erlaubten Wörter vorhanden sein.

Echtzeit / Nicht-Echtzeit (2)

Wie man das Programm *besonders bequem* schreibt, ohne dass es vielleicht besonders schnell arbeitet, sollte klar sein:

- Nachdem das Wort durch ein folgendes Leerzeichen als beendet erkannt wird,
- sucht man das Wort von oben nach unten in einer Liste, die noch nicht einmal alphabetisch geordnet sein muss, und gibt (wenn gefunden) „akzeptiert“ oder (sonst) „nicht akzeptiert“ aus.

Wort
MUMM
MUS
MUSIK
MUT
MUTIG
MUTTER

- Wir gehen im Beispiel mal von einem kleinen Alphabet (E, G, I, K, M, R, S, T, U)
- und einem sehr kleinen Wortschatz (siehe links) aus.

Die benötigte Rechenzeit hängt offenbar von der Länge der eingegebenen Zeichenfolge und der Größe der Tabelle ab.

Echtzeit / Nicht-Echtzeit (3)

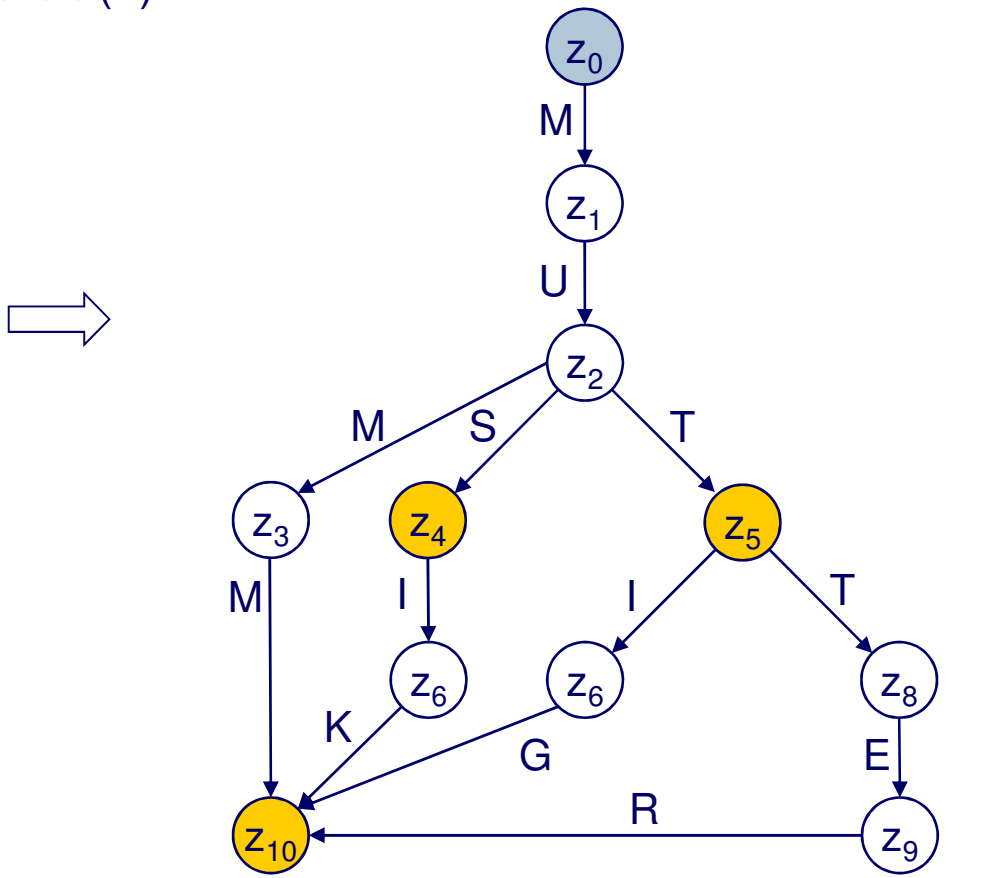
Ein Ansatz, dieses Problem *in Echtzeit* zu lösen – d.h. das Programm sagt sofort nach jedem Buchstaben, ob das bisher Getippte im Wörterbuch steht – basiert auf der folgenden Idee:

- Man verwendet einen gerichteten Graphen mit „Wurzel“, um die Informationen in der Tabelle zu repräsentieren.
- Jede Kante im Graphen erhält als Markierung einen Buchstaben. Jeder Weg im Graphen ab der Wurzel ist dadurch mit einer Buchstabenkette markiert.
- Jeder Knoten im Graphen enthält als Markierung die Information (bis hierher) „ akzeptiert“ (im Bild gelb) oder „nicht akz.“ (weiß).
- Die Buchstabenketten der Wege von der Wurzel bis zu einem „*akzeptiert*“-Knoten sind *genau alle* Wörter der Sprache.
- Die Buchstabenketten der Wege von der Wurzel bis zu einem „*nicht-akzeptiert*“-Knoten sind *alle keine* Wörter der Sprache.

Gibt es einen solchen Graphen?

Echtzeit / Nicht-Echtzeit (4)

Wort
MUMM
MUS
MUSIK
MUT
MUTIG
MUTTER



Gibt es einen solchen Graphen? *JA!*

Echtzeit / Nicht-Echtzeit (5)

Den auf der letzten Folie dargestellten Graphen kann man nun benutzen, um in Echtzeit herauszubekommen, ob ein Wort zu der Sprache gehört:

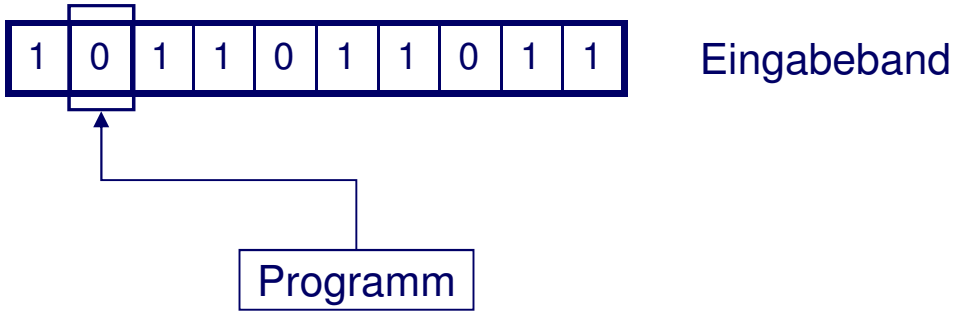
- *Begonnen* wird immer an der *Wurzel*.
- Wenn ein Buchstabe eingegeben wird,
  - wird der *Kante* gefolgt, die damit *markiert* ist ...
  - und angezeigt, ob der *erreichte Knoten* die bisher geschriebene Buchstabenfolge „akzeptiert“ oder „nicht akzeptiert,
  - bzw. wenn es *keine Kante* mit dem Buchstaben gibt, wird auch ab dann „nicht akzeptiert“.

Die benötigte Rechenzeit hängt nur von der Länge der eingegebenen Zeichenfolge ab

Der benutzte „Baum“ ist ein spezieller endlicher Automat.

- Endliche Automaten als einfaches Berechnungsmodell

  - Welche elementaren Operationen stehen zur Verfügung?
  - Wie viel Speicher steht zur Verfügung, und wie geht man damit um?
  - Wie wird die Eingabe eingegeben?
  - Wie wird die Ausgabe bestimmt?



- Informelle Beschreibung eines endlichen Automaten und seiner Verwendung

  - Eingabeband
    - linearer Speicher; besteht aus Zellen; je Zelle ein Zeichen
  - Lesekopf
    - ... bewegt sich von links nach rechts;
    - In jedem Schritt wird ein Zeichen gelesen.
  - Programm: steuert die Verarbeitung
    - ... befindet sich in unterschiedlichen Zuständen, anfangs im Anfangszustand.
    - Aktueller Zustand und aktuell gelesenes Zeichen legen den nächsten Zustand fest.
  - Ausgabe
    - indirekt: Der Zustand, in dem sich das Programm nach Abarbeitung des letzten Zeichens befindet, wird als Ausgabe interpretiert, genauer: ob er akzeptierend ist oder nicht.
    - alternativ: Ausgabe in akzeptierenden Zuständen : JA,

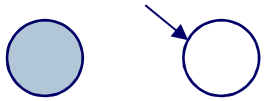
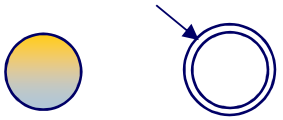
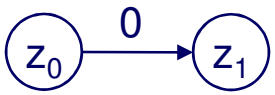
      sonst : NEIN.

Formale Beschreibung

- Komponenten eines endlichen Automaten $A = [Z,\Sigma,z_0,F,\delta]$
  - endliche Menge Z von Zuständen
    - ausgezeichneter Anfangszustand $z_0 \in Z$
    - Teilmenge $F \subseteq Z$ von akzeptierenden Zuständen
  - endliches Eingabealphabet Σ
  - vollständig definierte Zustandsüberführungsfunktion $\delta: Z \times \Sigma \rightarrow Z$

Verwendung, bezogen auf formale Beschreibung

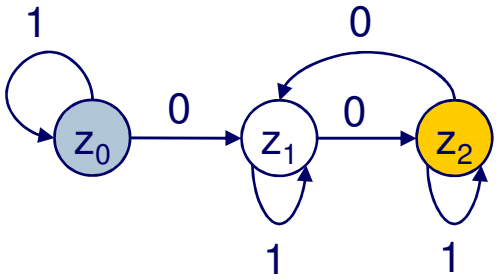
- Wörter $w \in \Sigma^*$ können auf dem Eingabeband stehen.
- Programm "=" Zustandsüberführungsfunktion δ
- Das erste Zeichen auf dem Eingabeband wird im Zustand z_0 verarbeitet.
- Unterteilung in F und $Z \setminus F$ wird benutzt, um das Ausgabeverhalten zu beschreiben.

- Darstellungsweise (in Bildern)
  - Anfangszustand 
  - Akzeptierender Zustand 
  - Anfangszustand, der gleichzeitig akzeptierend ist 
  - Zustandsüberführung, $\delta(z_0,0)=z_1$ 

Beispiel eines Automaten

- endlicher Automat $A = [Z,\Sigma,z_0,F,\delta]$ mit:
  - $Z = \{ z_0,z_1,z_2 \}$
  - $\Sigma = \{ 0,1 \}$
  - z_0
  - $F = \{ z_2 \}$
  - Für alle $z,z' \in Z$ und alle $x \in \Sigma$ gilt:
    $\delta(z,x) = z'$ gdw. es gibt eine mit x markierte Kante von z nach z'

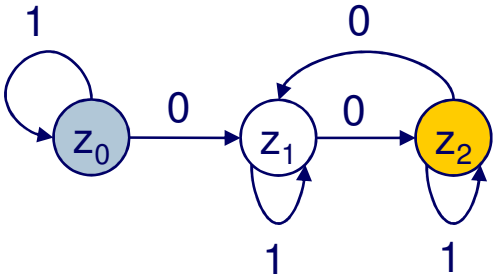
Graphische Beschreibung von δ bzw. $\delta(z,x)$, z_0 und F :



Tabellarische Beschreibung von δ bzw. $\delta(z,x)$:

z \ x	0	1
z_0	z_1	z_0
z_1	z_2	z_1
z_2	z_1	z_2

- Verarbeitungsbeispiel dieses Automaten

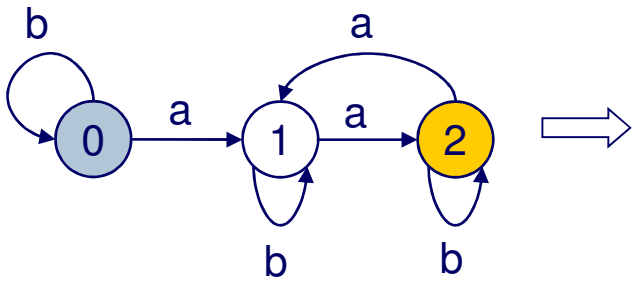


Verarbeitung der Eingabe w = 1010:

Zustand: z_0	Eingabeband mit Kopf:	1 0 1 0
	↓	
Zustand: z_0	Eingabeband mit Kopf:	1 0 1 0
	↓	
Zustand: z_1	Eingabeband mit Kopf:	1 0 1 0
	↓	
Zustand: z_1	Eingabeband mit Kopf:	1 0 1 0
	↓	
Zustand: z_2	Eingabeband mit Kopf:	1 0 1 0

... anders dargestellt: $(z_0,1010) \rightarrow_A (z_0,010) \rightarrow_A (z_1,10) \rightarrow_A (z_1,0) \rightarrow_A (z_2,\varepsilon)$

- Arbeitsweise des Automaten in Pseudoprogrammiersprache




```
string w; int z; char c;
z := 0;
read(w);
while ( w ≠ ”” )        {
        c := head(w);
        w := tail(w);
        if      ( z = 0 & c = 'a' ) then z := 1;
        else if ( z = 0 & c = ‘b' ) then z := 0;
        else if ( z = 1 & c = 'a' ) then z := 2;
        else if ( z = 1 & c = ‘b' ) then z := 1;
        else if ( z = 2 & c = 'a' ) then z := 1;
        else if ( z = 2 & c = ‘b' ) then z := 2;
                        };
print(z)
```

Begriff: beschriebene Sprache (zugrunde liegende Idee)

Sei $A = [Z,\Sigma,z_0,F,\delta]$ ein endlicher Automat

Die von A beschriebene Sprache (in Zeichen L(A)) enthält genau diejenigen Wörter aus Σ^*, für die gilt:

- Nachdem A, im Startzustand z_0 beginnend, das Wort w vollständig verarbeitet hat, erreicht A einen Zustand $z \in F$,

Die Sprache L(A) wird auch die von A akzeptierte Sprache genannt

Wir kümmern uns jetzt darum, wie man die Sprache L(A) formal definieren kann.
Die zugrunde liegende Idee ist typisch, um zu erklären, was ein Programm eines Berechnungsmodells leistet .

- Verhaltensaspekte eines endlichen Automaten:
  Fortsetzung der Zustandsüberführungsfunktion

  - Es sei $A = [Z,\Sigma,z_0,F,\delta]$ ein endlicher Automat;
  - es seien $w \in \Sigma^*$ und $z \in Z$.

Die Fortsetzung $\delta^*: Z \times \Sigma^* \rightarrow Z$ der Zustandsüberführungsfunktion δ ist wie folgt definiert:

- falls $w = \varepsilon$, so ist $\delta^*(z,w) = z$
- falls $w = x \circ w'$ mit $x \in \Sigma$ und $w' \in \Sigma^*$, so ist $\delta^*(z,w) = \delta^*(\delta(z,x),w')$

$\delta^*(z,w)$: *Wohin bringt mich vom Zustand z aus die „Wegbeschreibung“ w?*

- Verhaltensaspekte eines endlichen Automaten:
  Fortsetzung der Zustandsüberführungsfunktion

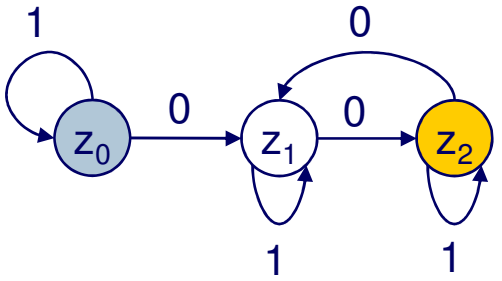
  - Es sei $A = [Z,\Sigma,z_0,F,\delta]$ ein endlicher Automat;
  - es seien $w \in \Sigma^*$ und $z \in Z$.

Die Fortsetzung $\delta^*: Z \times \Sigma^* \rightarrow Z$ der Zustandsüberführungsfunktion δ ist wie folgt definiert:

- falls $w = \varepsilon$, so ist $\delta^*(z,w) = z$
- falls $w = x \circ w'$ mit $x \in \Sigma$ und $w' \in \Sigma^*$, so ist $\delta^*(z,w) = \delta^*(\delta(z,x),w')$

- Beispiel

$$\begin{aligned}
\delta^*(z_1,01) &= \delta^*(\delta(z_1,0),1) \\
&= \delta^*(z_2,1) \\
&= \delta^*(z_2,1\varepsilon) \\
&= \delta^*(\delta(z_2,1),\varepsilon) \\
&= \delta^*(z_2,\varepsilon) \\
&= z_2
\end{aligned}$$

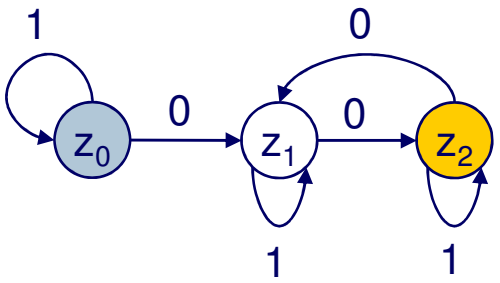


- Begriff: beschriebene (akzeptierte) Sprache (formale Definition)

  - sei $A = [Z,\Sigma,z_0,F,\delta]$ ein endlicher Automat
  - die von A beschriebene Sprache (kurz: L(A)) ist auch wie folgt definierbar:

$$L(A) = \{ w \in \Sigma^* \mid \delta^*(z_0,w) \in F \}.$$

- Beispiel



Für diesen endlichen Automaten A gilt offenbar:

$L(A) = \{ w \in \Sigma^* \mid$ w enthält gerade viele Nullen, mindestens zwei $\}$

- weitere Vorgehensweise

  Wir schauen uns jetzt an, wie man zu einer gegebenen Sprache L einen endlichen Automaten A mit L(A) = L konstruieren kann.

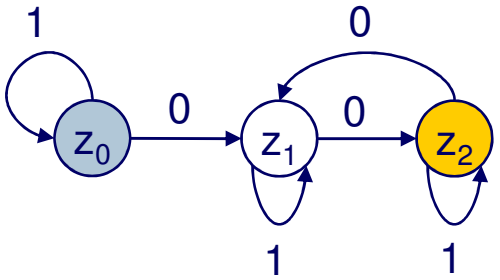
  Um uns das Leben leichter zu machen, führen wir noch einen Hilfsbegriff ein.

- Verhaltensaspekte eines endlichen Automaten:
  Wege nach z - $K_A(z)$ informell

  - Es sei $A = [Z,\Sigma,z_0,F,\delta]$ ein endlicher Automat.
  - Es sei $z \in Z$.
  - Zu jedem Zustand $z \in Z$ bezeichnet $K(A,z)$ die Menge der Wörter $w \in \Sigma^*$, für die gilt: nach Verarbeitung des letzten Buchstabens von w befindet sich der Automat A im Zustand z.
  - Da A meist „fest“ ist, schreiben wir für K(A,z) auch $K_A(z)$.

  $K_A(z)$: Welche Wörter bringen A von z_0 nach z?

- Beispiel



  - $K_A(z_0) = \{ w \in \Sigma^* \mid w \text{ enthält keine 0} \}$
  - $K_A(z_1) = \{ w \in \Sigma^* \mid w \text{ enthält ungerade viele 0‘en} \}$
  - $K_A(z_2) = \{ w \in \Sigma^* \mid w \text{ enthält gerade viele 0‘en, mindestens 2} \}$

$K_A(z)$ formal

- Es sei $A = [Z,\Sigma,z_0,F,\delta]$ ein endlicher Automat;
- es sei $z \in Z$.

Die Menge $K_A(z)$ ist formal wie folgt definiert: $K_A(z) = \{ w \in \Sigma^* \mid \delta^*(z_0,w) = z \}$.

Kurz, wie bereits gesagt:

$K_A(z)$: Welche Wörter bringen A von z_0 nach z?

Anders gesagt:

$K_A(z)$: Was wäre die Automatensprache, wenn genau nur z akzeptieren würde, d.h. F={z} wäre?

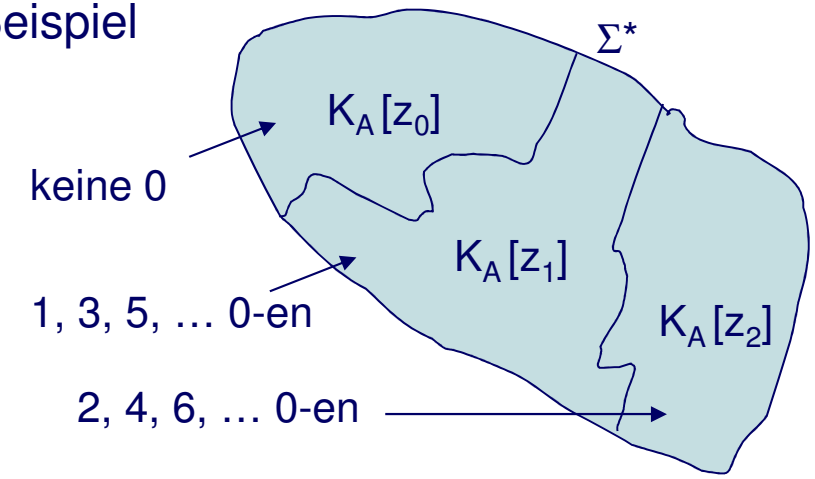
Eine wichtige Beobachtung

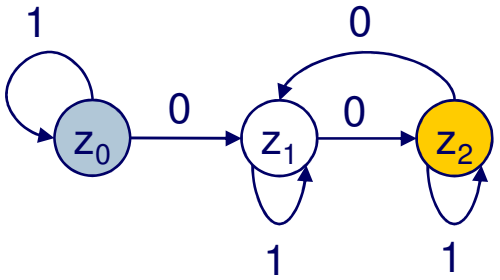
- Sei $A = [Z,\Sigma,z_0,F,\delta]$ ein endlicher Automat;
- dann gilt:

> - Für je zwei verschiedene Zustände $z, z' \in Z$ gilt, dass die Mengen $K_A(z)$ und $K_A(z')$ disjunkt sind.
> - Wenn man alle Mengen $K_A(z)$ vereinigt, enthält man die Menge Σ^*.

Die den Zuständen von A zugeordneten Mengen $K_A(z)$ definieren eine Klasseneinteilung auf Σ^*.

Beispiel





- Die von einem endlichen Automaten beschriebene Sprache

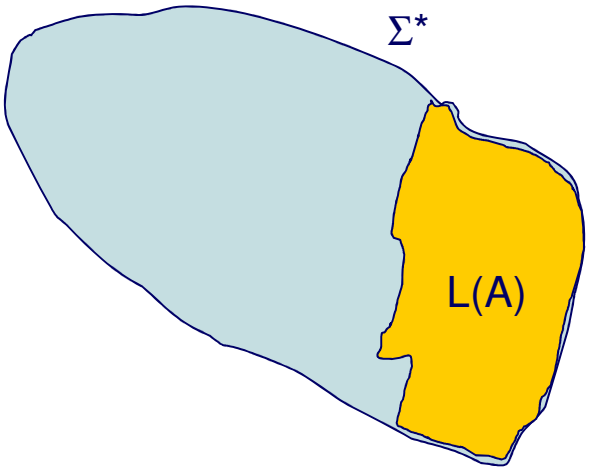
  - Es sei $A = [Z,\Sigma,z_0,F,\delta]$ ein endlicher Automat.

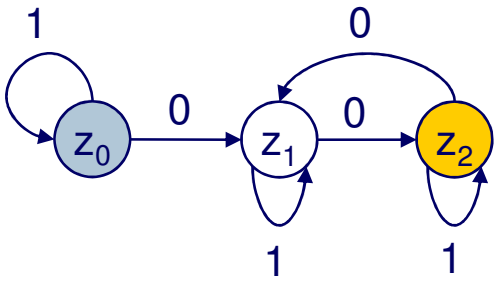
Die von A beschriebene Sprache L(A) ist wie folgt definiert:

$$L(A) = \bigcup_{z \in F} K_A(z) = \{ w \in \Sigma^* \mid w \in K_A(z), z \in F \} = \{ w \in \Sigma^* / \delta^*(z_0,w) \in F \}$$

L(A) : Was bringt A von z_0 in einen akzeptierenden Zustand?

- Beispiel





$L(A) = K[z_2] = \{ w \in \Sigma^* \mid w$ enthält gerade vielen 0'en, minimal 2 $\}$

- Umgekehrtes Beispiel: Sprache $\rightarrow$ Automat (intuitiv)

  Gesucht: ein endlicher Automat, der eine
  (irgendwie ohne Automat beschriebene)
  Sprache L über einem Alphabet Σ beschreibt.

- Allgemeine Tipps:

  - Welche nötigen bzw. verbotenen Eigenschaften der Wörter haben wir in einem betrachteten Zustand z erreicht?
  - Ist $K_A(z)$ …
    - komplett in der Sprache enthalten (→ z akzeptiert), oder
    - komplett nicht (→ z akzeptiert nicht), oder
    - teils/teils? (Dann sollten wir z durch mehrere Zustände ersetzen.)
  - Welcher nächste Schritt aus dem betrachteten Zustand verleiht dem Wort erwünschte bzw. unerwünschte Eigenschaften?
  - Welche nächsten Schritte verleihen den bisherigen Wörtern welche „sprachrelevanten Eigenschaften“ ? Danach ist zu unterscheiden.

- Umgekehrtes Beispiel: Sprache $\rightarrow$ Automat (intuitiv)

  Gesucht: ein endlicher Automat, der folgende Sprache L über $\Sigma = \{ 0,1 \}$ beschreibt:

  $L = \{ w \in \Sigma^* \mid w$ ist Binärdarstellung einer durch 3 teilbaren Zahl $\}$

- Spezielle Tipps:

  - Welche 0-1-Folgen stellen (üblicherweise) **keine** Binärzahl dar?
    Antwort: die leere Folge und alle Folgen mit Präfix 0 - außer der Folge „0“.
  - Zur Vereinfachung ordnen wir zunächst auch diesen Folgen einen (plausiblen) Zahlenwert zu.
  - Wenn ich an eine Binärzahl 0 oder 1 anhänge, was geschieht mit ihrem „Rest bei Division durch 3“?
  - Am Ende schließen wir die oben zur Vereinfachung ausnahmsweise zugelassenen Binärzahlen wieder aus.

- Umgekehrtes Beispiel: Sprache $\rightarrow$ Automat (intuitiv)

  Gesucht: ein endlicher Automat, der folgende Sprache L über $\Sigma = \{ 0,1 \}$ beschreibt:

  $L = \{ w \in \Sigma^* \mid w$ ist Binärdarstellung einer durch 3 teilbaren Zahl $\}$

- Zahlenwerte (für alle Folgen) und Divisionsreste

  - Zu einer Zeichenkette $w \in \Sigma^*$ bezeichnen wir mit nat(w) die natürliche Zahl, die w repräsentiert, wobei gilt:
    - wenn w = 0w‘gilt, so ist nat(w) = nat(w‘)
    - wenn w = ε, so ist nat(w) = 0
    - wenn w = 1 gilt, so ist nat(w) = 1
    - wenn w = w‘0 gilt, so ist nat(w) = 2*nat(w‘) + 0
    - wenn w = w‘1 gilt, so ist nat(w) = 2*nat(w‘) + 1
  - Es bezeichne (n mod 3) den Rest der natürlichen Zahl n bei Division durch 3 (mögliche Werte: 0,1 und 2)

- Umgekehrtes Beispiel: Sprache $\rightarrow$ Automat (intuitiv)

  Gesucht: ein endlicher Automat, der folgende Sprache L über $\Sigma = \{ 0,1 \}$ beschreibt:

  $L = \{ w \in \Sigma^* \mid w$ ist Binärdarstellung einer durch 3 teilbaren Zahl $\}$

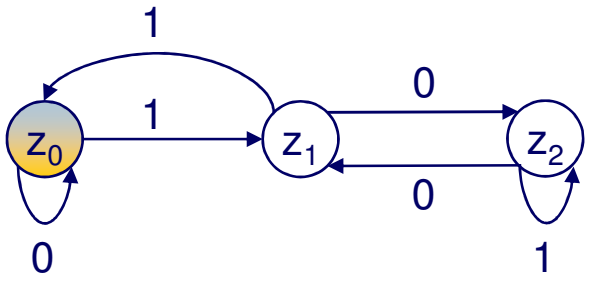
- 3-Reste bei Verlängerung der Binärzahl um 0 oder 1

(nat(w) mod 3)	(nat(w0) mod 3)	(nat(w1) mod 3)
0	0	1
1	2	0
2	1	2

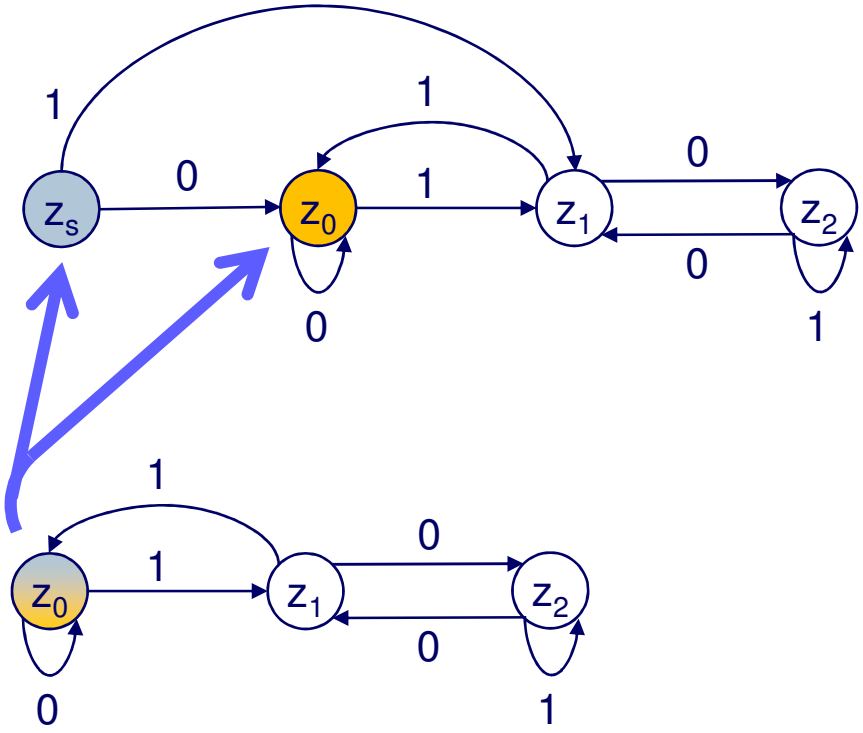
Warum? nat(w0) mod 3 = 2×nat(w) mod 3, d.h.
nat(w) mod 3 = 0 $\Rightarrow$ nat(w) = 3u $\Rightarrow$ nat(w0) mod 3 = 6u mod 3 = 3×(2u) mod 3 = 0
nat(w) mod 3 = 1 $\Rightarrow$ nat(w) = 3u+1 $\Rightarrow$ nat(w0) mod 3 = (6u+2) mod 3 *usw.*
usw.

- Umgekehrtes Beispiel: Sprache $\rightarrow$ Automat (intuitiv)
- 3-Reste-Tabelle als Automat:

(nat(w) mod 3)	(nat(w0) mod 3)	(nat(w1) mod 3)
0	0	1
1	2	0
2	1	2



- Umgekehrtes Beispiel: Sprache $\rightarrow$ Automat (intuitiv)
- leere Folge ausschließen:



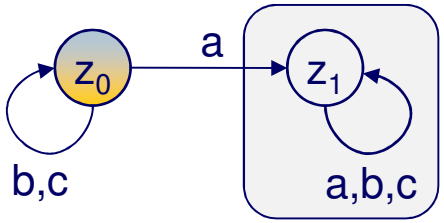
- führende Nullen ausschließen: *Wenn nach 0 noch etwas folgt, kann aus dem Wort „nichts mehr werden“.*

Intermezzo: Zwei gelegentlich anzutreffende Details in endlichen Automaten

- In den Beispielen sei $\Sigma = \{a,b,c\}$.

A_1

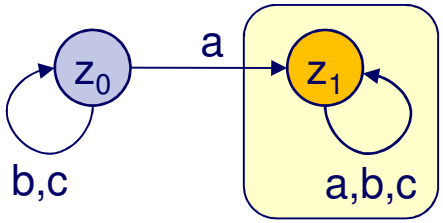
$L(A_1)$: alle Wörter die kein a enthalten



Error-State (Mülltonne)
egal wie's weitergeht: nie mehr „gut"

A_2

$L(A_2)$: alle Wörter die a enthalten



Paradies

egal wie's weitergeht: ab jetzt immer „gut"

- Umgekehrtes Beispiel: Sprache $\rightarrow$ Automat (intuitiv)
- führende Nullen ausschließen: