Automaten

by

Dr. Günter Kolousek

Automaten – Einführung

- Automat = virtuelle Maschine
- Ausführung eines Algorithmus
  - als Programm auf realer Maschine
  - virtuelle Maschine
- Automatentheorie
  - Teil der theoretischen Informatik
  - Beschreibung der Arbeitsweise von Automaten
  - Algorithmen mit Hilfe von Automaten untersuchen

Automaten – Zweck

- Überprüfung, ob reale Maschine gebaut werden kann
- Modellierung von zustandsabhängigen Systemen
  - Entwurf und Analyse digitaler Schaltkreise
  - Simulation und Implementierung realer Automaten (z.B. Getränkeautomat, Verkehrsampel, Bankomat,…)
  - Spezifikation und Implementation von Netzwerkprotokollen, GUIs (z.B. Wizard), Workflows
  - lexikalische Analyse (Compiler)
  - Überprüfung von Eingabeworten, Mustererkennung

Einteilung der Automaten

Chomsky-Hierarchie für Automaten

Typ	Automat	Sprache
Typ-0	Turingmaschine	unbeschränkt
Typ-1	linear beschränkter Automat	kontextsensitiv
Typ-2	Kellerautomat	kontextfrei
Typ-3	endlicher Automat	regulär

Definitionen

- Eingabealphabet = Menge aller Eingabezeichen: E
- endliche Menge aller Zustände: Z
  - d.h. $|Z| = n, n \in \mathbb{N}$
- Zustandsübergangsfunktion: δ
  - Automat im i.ten Zustand z_i
    - Startzustand: z_0
  - durch Eingabe des i-ten Zeichens e_i
    - Eingabewort $e = (e_0, e_1, \ldots, e_n)$
  - wird in den Zustand z_{i+1} übergeleitet.

Definitionen – 2

- Menge der Endzustände: F
  - $F \subseteq Z$
- Ausgabealphabet = Menge aller Ausgabezeichen: A
- Ausgabefunktion: γ
  - Automat im i.ten Zustand: z_i
  - durch Eingabe des i-ten Zeichens: e_i
  - ausgegeben wird das i-te Zeichen: a_i.

Beispiel Getränkeautomat

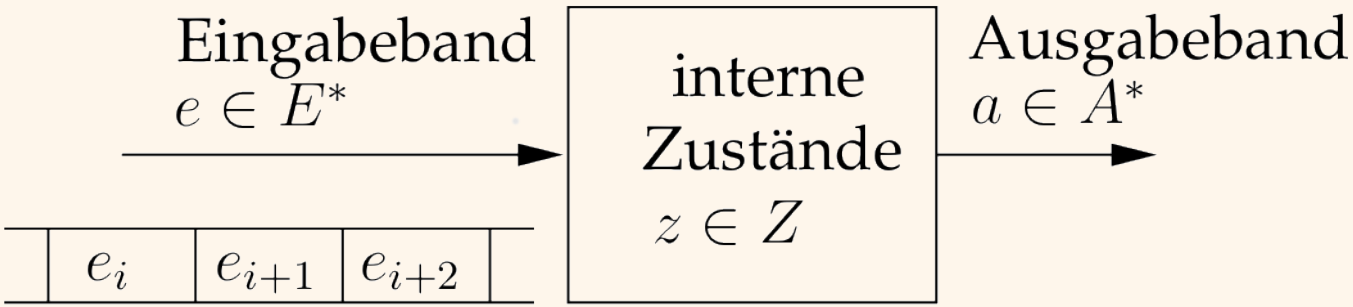
- Geldstück vom Betrag g
- Wahl zwischen zwei möglichen Getränken
  - Kaffee … k
  - Tee … t
- Auswahltaste ohne Geld: Signalton s
- Rückgabemöglichkeit durch Drücken von r

Beispiel Getränkeautomat – 2

- Eingabealphabet $E = \{g, k, t, r\}$
  - g … Einwurf eines Geldstückes
  - k … Kaffee-Auswahltaste k drücken
  - t … Tee-Auswahltaste t drücken
  - r … Rückgabetaste r drücken
- Zustandsmenge $Z = \{A, B\}$
  - A … Geldbetrag ausreichend
  - B … Automat bereit
- Ausgabealphabet $A = \{k, t, x, s\}$
  - k … Ausgabe Getränk Kaffee
  - t … Ausgabe Getränk Tee
  - x … Rückgabe - Geldbetrag x
  - s … Signalton

Arbeitsweise

- Vom Eingabewort e (am Eingabeband) wird ein Zeichen gelesen
- neuer Zustand wird bestimmt
- eventuell Ausgabe von Zeichen (am Ausgabeband)



Zustandsdiagramm

- Darstellungsweise der Zustandsübergangsfunktion δ
- Knoten (Zustand)
  - Startzustand
  - Normaler Zustand
  - Endzustand
    - $\rightarrow$ Fehlerzustand
- Kante (Zustandsübergang)
  - mit/ohne Ausgabe
  - Zusammenfassung mehrerer paralleler Kanten

Beispiele

- Zustandsdiagramm des Getränkeautomaten
- Zustandsdiagramm des erweiterten Getränkeautomaten
  - Getränkepreis 1 Euro
  - 1 Euro-Münze und 50 Cent-Münze
  - Restbetrag soll zurückgegeben werden

- $\rightarrow$ manchmal ist ”Fehlerzustand” sinnvoll

Zustandstabelle

- Darstellungsweise der Zustandsübergangsfunktion δ
- einfacher Getränkeautomat
  - $z_0 = B, F = \{B\}$
  - Tabelle

	g	k	t	r
$\underline{\text{B}}$	A/-	B/s	B/s	B/-
A	A/g	B/k	B/t	B/g

- Beispiel
  - ges.: Zustandstabelle des erweiterten Getränkeautomaten

Automaten – Überblick

1. Endlicher Automat ohne Ausgabe (EA)
2. Deterministischer EA (DEA)
3. Satz von der Existenz endlicher Automaten
4. Konstruktion eines EA aus einer rechtslinearen regulären Grammatik
5. Nichtdeterministischer EA (NEA)
6. Satz über die Äquivalenz von NEA und DEA
7. Konstruktion eines DEA aus NEA
8. Konstruktion eines minimalen DEA
9. Implementierung eines DEA
10. Endlicher Automat mit Ausgabe: Mealy & Moore
11. Kellerautomat
12. Turingmaschine

Endlicher Automat ohne Ausgabe

(engl. finite automaton)

- EA $= (E, Z, \delta, z_0, F)$
- EA: *endliche* Menge an Zuständen!
  - d.h. $|Z| = n$
- ohne Ausgabe $\rightarrow$ Akzeptor
  - Akzeptor: Eingabeworte entweder akzeptiert oder nicht akzeptiert
  - akzeptiert gdw. Eingabewort zur Gänze gelesen und Endzustand erreicht
  - hält, wenn Eingabewort zur Gänze gelesen oder kein weiterer Zustandsübergang möglich

EA – prinzipielle Arbeitsweise

```
def process(delta, z0, F, e):
    z = z0
    for e in e:
        waehle z aus delta(z, e)
        wenn kein Folgezustand:
            break
    else: # Eingabewort zur Gaenze gelesen!
        if z in F:
            return True
    return False
```

Von einem Knoten mehrere gleichbezeichnete Kanten?!

Deterministischer EA

- Keine gleichbezeichnete Kanten von einem Knoten
  - Zustandsübergangsfunktion δ liefert einen Zustand
    - $\delta : Z \times E \to Z, z_{i+1} = \delta(z_i, e_i)$
    - verwenden wir nicht: ϵ-DEA ... $\delta : Z \times (E \cup \{\epsilon\}) \to Z$
- δ liefert u.U. keinen Zustand
  - $\delta : Z \times E \to (Z \cup \{\varepsilon\})$
- Arbeitsweise

```
def process(delta, z0, F, e):
    z = z0
    for e in e:
        z = delta(z, e)
        if not z:
            break  # kein Zustandsuebergang!
    else:  # Eingabewort zur Gaenze gelesen!
        if z in F:
            return True
    return False
```

DEA – Beispiele

- Darstellung ganzer Zahlen
- Darstellung für Gleitkommazahlen (ohne Exponent).
  - erlaubt: `123 +0.5 -.3 .7`
  - nicht erlaubt: `3. 1.2.3 --5`
  - Eingabealphabet: $E = \{0, \ldots, 9, +, -, .\}$
  - Zustände: $Z = \{S, A, B, C, D\}$
  - Anfangsszustand: $z_0 = S$
  - Endzustände: $F = \{A, C\}$

Akzeptierte Wortmenge eines DEA

- erweiterte Zustandsübergangsfunktion $\hat{\delta}$ eines DEA
  - ermittelt ausgehend von einem Zustand beim Einlesen eines Wortes den erreichten Zustand
  - $v, w \in E^+, e \in E, w = ve$

$$\hat{\delta} : Z \times E^+ \to Z$$

$$\hat{\delta}(z, w) = \begin{cases} \delta(\hat{\delta}(z, v), e) & |w| > 1 \\ \delta(z, w) & |w| = 1 \end{cases}$$

  - Beispiel: $\hat{\delta}(z, abc) = \delta(\hat{\delta}(z, ab), c) = \delta(\delta(\hat{\delta}(z, a), b), c) = \delta(\delta(\delta(z, a), b), c)$
- Wortmenge, die von einem DEA akzeptiert wird: $T(DEA)$

$$T(DEA) = \begin{cases} \{w \in E^+ | \hat{\delta}(z_0, w) \in F\} & z_0 \notin F \\ \{w \in E^+ | \hat{\delta}(z_0, w) \in F\} \cup \{\epsilon\} & z_0 \in F \end{cases}$$

Satz von der Existenz EA

Zu jeder regulären Grammatik G gibt es einen endlichen Automaten A, für den gilt:

$$L(G) = T(A)$$

wobei:

- L(G) … Sprache, die durch eine Grammatik erzeugt werden kann
- T(A) … Menge der Worte, die vom Automaten akzeptiert werden

D.h. die von A akzeptierte Wortmenge T(A) stimmt mit der von der Grammatik G erzeugten Sprache L(G) überein.

Konstruktion: aus re-li reg. G

- G = (Φ, Σ, P, S)
- Algorithmus
  1. Zu jedem NT-Symbol aus Φ wird Knoten gebildet (keine Endknoten).
  2. Zusätzlicher Endknoten mit neuer Bezeichnung.
  3. Startknoten entspricht Startsymbol S.
  4. Kanten gemäß den Produktionen:
     - A $\rightarrow$ aB: Kante von A nach B mit der Beschriftung a.
     - A $\rightarrow$ a: Kante vom Knoten A zum Endknoten mit der Beschriftung a.
     - Bei S $\rightarrow \varepsilon$: Startknoten ist gleichzeitig Endknoten.

Beispiel

ges.: Automat für ganze Zahlen

$$
\begin{aligned}
G &= (\Phi, \Sigma, P, S) \\
\Phi &= \{S, Z\} \\
\Sigma &= \{+, -, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \\
P &= \{S \to +Z| - Z|0Z|1Z|2Z|3Z|4Z|5Z|6Z|7Z|8Z|9Z, \\
&\qquad S \to 0|1|2|3|4|5|6|7|8|9, \\
&\qquad Z \to 0Z|1Z|2Z|3Z|4Z|5Z|6Z|7Z|8Z|9Z, \\
&\qquad Z \to 0|1|2|3|4|5|6|7|8|9\} \\
S &= S
\end{aligned}
$$

Satz: Äquivalenz von re-li und li-li Gr

- Zu jeder rechtslinearen Grammatik G = (ϕ, Σ, P, S) existiert eine linkslineare Grammatik G' = (ϕ', Σ, P', S'), sodass L(G) = L(G') gilt, die Grammatiken also äquivalent sind.
- Zu jeder linkslinearen Grammatik G = (ϕ, Σ, P, S) existiert eine rechtslineare Grammatik G' = (ϕ', Σ, P', S'), sodass L(G) = L(G') gilt, die Grammatiken also äquivalent sind.

Konstruktion: aus li-li reg. G

- G = (Φ, Σ, P, S)
- Algorithmus
  1. Zu jedem NT-Symbol aus Φ wird Knoten gebildet (keinen Startknoten markieren!).
  2. Zusätzlicher Startknoten mit neuer Bezeichnung.
  3. Endknoten entspricht Startsymbol S.
  4. Kanten gemäß den Produktionen:
     - A $\rightarrow$ Ba: Kante von B nach A mit der Beschriftung a.
     - A $\rightarrow$ a: Kante vom neuem Startknoten zum Knoten A mit der Beschriftung a.
     - Bei S $\rightarrow \varepsilon$: neuer Startknoten ist gleichzeitig Endknoten.

Nichtdeterministischer EA

- Von einem Knoten: mehrere gleichbezeichnete Kanten
  - Zustandsübergangsgangsfunktion δ liefert Menge zurück
    - $\delta : Z \times E \to \mathcal{P}(Z)$
    - verwenden wir nicht: ϵ-NEA ... $\delta : Z \times (E \cup \{\epsilon\}) \to \mathcal{P}(Z)$
- D.h. der Automat muss eine der Kanten wählen
  - kann die ”falsche” sein $\to$ backtracking
- Arbeitsweise

```
def process(delta, z0, F, e):
    z = z0
    for e in e:
        zset = delta(z, e)
        if not zset:
            break  # kein Zustandsuebergang!
        z = zset.pop()  # u.U. falsch!
    else:  # Eingabewort zur Gaenze gelesen!
        if z in F:
            return True
    return False
```

NEA – Beispiel

Darstellung für Gleitkommazahlen (ohne Exponent)
erlaubt: `123 +0.5 -.3 .7`
nicht erlaubt: `3. 1.2.3 --5`

	$0-9$	$+,-$	$.$
S	$\{E\}$	$\{D, A\}$	$\{B\}$
A	$\{A\}$	-	$\{B\}$
B	$\{C\}$	-	-
C	$\{C\}$	-	-
D	$\{E\}$	-	-
E	$\{E\}$	-	-

`0.3`,... kann nicht verarbeitet werden. Modifikationen?

Satz: Äquivalenz von NEA und DEA

Zu jedem nicht-deterministischen endlichen Automaten gibt es einen äquivalenten deterministischen endlichen Automaten.

Es gilt somit: T(NEA) = T(DEA)

NEA zu DEA

- $Z_{NEA} = \{z_{(0)}, z_{(1)}, z_{(2)}, z_{(3)}, \ldots\}$
- $Z_{DEA} = \{z'_{(0)}, z'_{(1)}, z'_{(2)}, z'_{(3)}, \ldots\}$
- Zustandsmenge des DEA ist eine Teilmenge der Potenzmenge der Zustandsmenge des NEA

$$\mathcal{P}(Z_{NEA}) = \{\{\}, \{z_{(0)}\}, \{z_{(0)}, z_{(1)}\}, \{z_{(0)}, z_{(2)}\}, \{z_{(0)}, z_{(3)}\}, \ldots\}$$

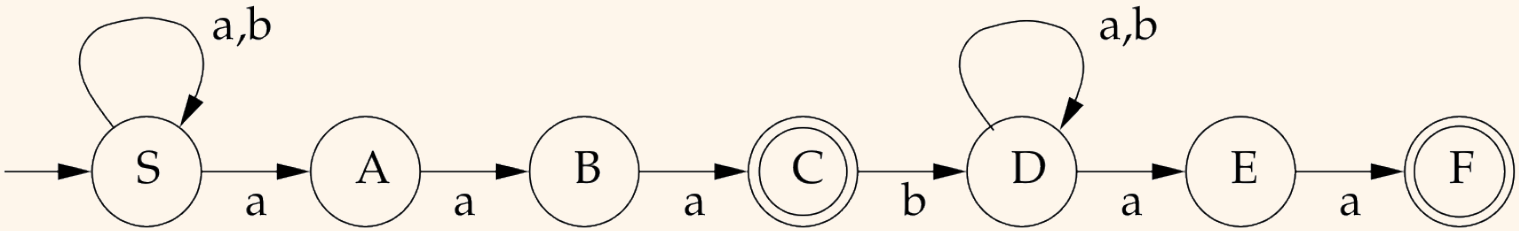
D.h.:

$$Z_{DEA} \subseteq \mathcal{P}(Z_{NEA})$$

NEA zu DEA – Konstruktion

- Basis: Zustandstabelle des nicht-deterministischen Automaten
- Verfahren
  1. Beim Startzustand beginnen
  2. Enthält die Zustandstabelle eine Teilmenge: neuer Knoten, der diese Teilmenge darstellt.
  3. Für neue ‘Teilmengen’-knoten ergibt sich das Verhalten aus der Summe aller Zustände der Teilmenge.
  4. Schritt 2,3 solange durchführen, bis alle Knoten, die in der Zustandstabelle vorkommen auch auf der linken Seite (Liste der Zustände) stehen.
  5. Bestimmen der Endknoten: Jene, die mindestens einen Endknoten des NEA enthalten.
  6. Die neuen Knoten zur besseren Lesbarkeit umbenennen.

NEA zu DEA – Beispiel



Minimaler DEA

Motivation und Definition

- Erinnerung: $Z_{DEA} \subseteq \mathcal{P}(Z_{NEA})$, d.h.:$|Z_{DEA}| \leq 2^{|NEA|} \rightarrow$ Anzahl der Zustände des konstruierten DEA ist u.U. sehr hoch!!!
- Optimal für die Implementierung: DEA mit minimaler Anzahl an Zuständen
- Definition: Äquivalenter minimaler DEA … äquivalenter DEA mit minimaler Anzahl an Zuständen (von allen äquivalenten DEAs)
- Ziel: Konstruktion eines äquivalenten minimalen DEAs

Minimaler DEA – 2

Äquivalente Zustände

- $z_{(i)}$ und $z_{(j)}$ sind äquivalent, wenn:
  $\forall w \in E^+ : \hat{\delta}(z_{(i)}, w) \in F \leftrightarrow \hat{\delta}(z_{(j)}, w) \in F$
- Beachte, dass **nicht** gefordert ist, dass

$$\hat{\delta}(z_{(i)}, w) = \hat{\delta}(z_{(j)}, w)$$

- man nennt 2 Zustände *unterscheidbar*, wenn diese nicht äquivalent sind
  - D.h. $z_{(i)}$ ist von $z_{(j)}$ unterscheidbar, wenn es mindestens ein w gibt, sodass einer der Zustände $\hat{\delta}(z_{(i)}, w)$ und $\hat{\delta}(z_{(j)}, w)$ akzeptiert und der andere nicht.
- Ermittlung äquivalenter Zustände mit dem *Table-filling* Algorithmus

Minimaler DEA – Konstruktion

1. Entferne alle Zustände, die vom Startzustand aus nicht erreicht werden können.

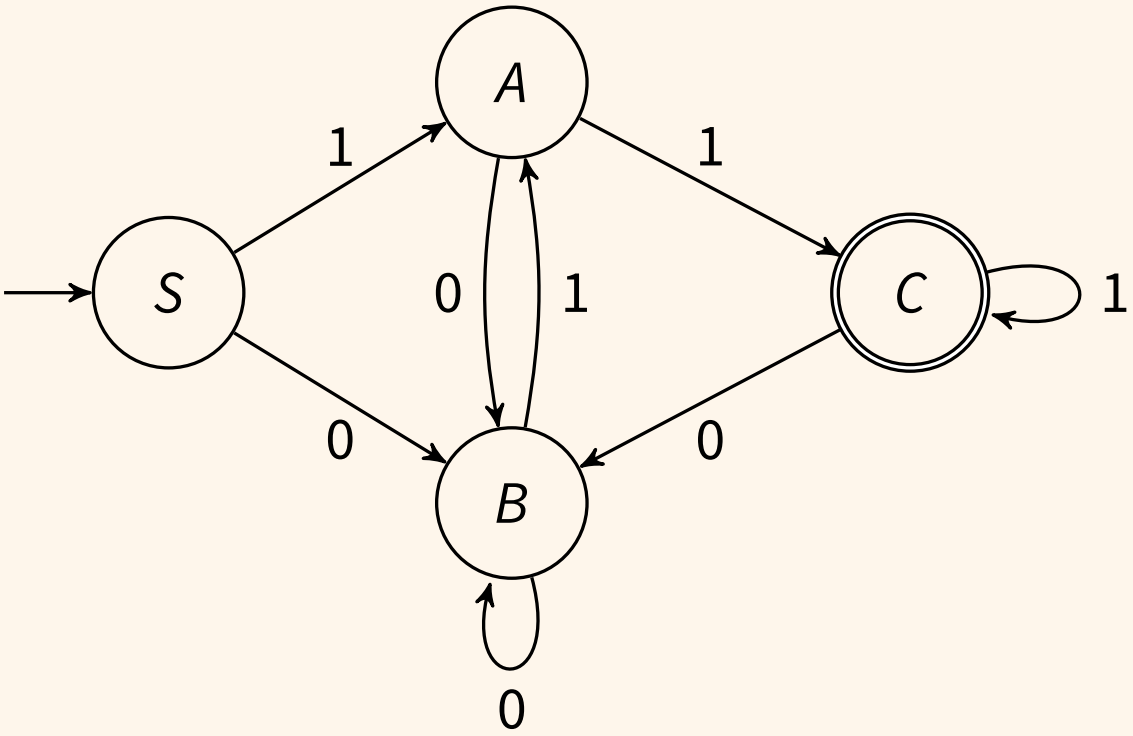
Table-filling Algorithmus:

2. Erstelle für die Menge von unterschiedlichen Paaren (keine Reihenfolge $\rightarrow$ Menge) an Zuständen $\{\{z_{(i)}, z_{(j)}\}, i \neq j\}$ eine Tabelle.
3. Markiere alle Paare, bei denen genau ein Zustand zu den akzeptierenden Zuständen gehört und der andere nicht, als nicht zusammenlegbar.
4. Wiederhole bis keine Änderungen mehr vorgenommen:
   - markiere alle Paare als nicht zusammenlegbar, für die es ein Eingabezeichen e gibt, so dass die mit e erreichten Folgezustände bereits markiert wurden.

Minimaler DEA – Konstruktion – 2

5. Partitioniere die Menge der Zustände Z auf Basis von Schritt 4 in Blöcke, die jeweils alle zu einem Zustand z äquivalenten Zustände enthalten.
6. Konstruiere den äquivalenten minimalen DEA unter Verwendung der erstellten Blöcke.

Beispiel



Beispiel – 2

1. Zustände entfernen… $\rightarrow$ nichts zu tun
2. Leere Tabelle erstellen

A			
B			
C			
	S	A	B

3. Initiale Paare markieren

A			
B			
C	X	X	X
	S	A	B

Beispiel – 3

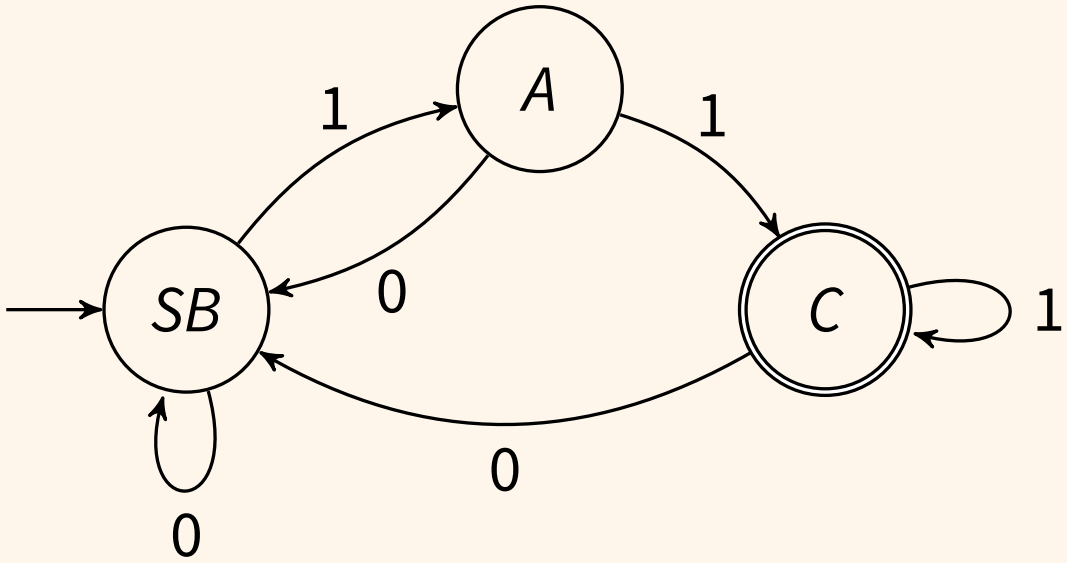
4. Wiederholen bis keine Änderungen…
   (a) Iteration 1
   - $\{A, S\}$: $A \xrightarrow{1} C, S \xrightarrow{1} A$ … $\{C, A\}$ bereits markiert $\rightarrow \{A, S\}$ markieren!
   - $\{B, S\}$: $B \xrightarrow{1} A, S \xrightarrow{1} A$ … nicht unterscheidbar $\rightarrow$ *nicht* markieren!
   - $\{B, A\}$: $B \xrightarrow{1} A, A \xrightarrow{1} C$ … $\{C, A\}$ bereits markiert $\rightarrow \{B, A\}$ markieren!

A	X		
B		X	
C	X	X	X
	S	A	B

   (b) Iteration 2: Einzige freie Stelle ist $\{B, S\} \rightarrow$ nicht unterscheidbar…

Beispiel – 4

5. Blöcke bilden. Nur $\{B, S\}$ ist nicht unterscheidbar $\rightarrow$ $Z = \{SB, A, C\}$
6. Zustandsdiagramm des äquivalenten minimalen DEA



Knoten umbenennen und fertig!

Implementierung eines DEA

1. switch-basiert

```
char z;  // current state
char e;  // current input symbol
switch (z) {
  case 'A':
    switch (e) {
      case 'a':
        z_new = 'B';
        break;
      case 'b': ...
      case 'c': ...
    }
    break;
  case 'B':
    switch (e) {
      case 'a': ...
      ...
    }
}
```

Implementierung eines DEA

`switch` basiert $\rightarrow$ einfach, effizient, aber unflexibel

2. Tabellen-basiert
   - Verwendung eines verschachtelten Dictionaries
     - u.U. auch zweidimensionales Array
   - äußeres Dictionary
     - Key … aktueller Zustand
     - Value … inneres Dictionary
   - inneres Dictionary
     - Key … aktuelles Eingabesymbol
     - Value … Folgezustand
   - Ermittlung des Folgezustandes

     ```
     z = delta_tab[z][e]
     ```

Endlicher Automat *mit* Ausgabe

- Zweck: Erzeugung von Ausgabewörtern aus gegebenen Eingabewörtern
- $M = (E, Z, A, \delta, \gamma, z_o, F)$
- Mealy-Automat
  - Ausgabefunktion
    - $a_i = \gamma(z_i, e_i)$,
    - $\gamma : Z \times E \to (A \cup \{\varepsilon\})$
- Moore-Automat
  - Ausgabefunktion
    - $a_i = \gamma(z_i)$
    - $\gamma : Z \to (A \cup \{\varepsilon\})$

Beispiel: Alarmanlage

- Zustände:
  - O … ausgeschaltet (off)
  - B … bereit
  - V … Voralarm (Bewegungsmelder an)
  - A … Alarm (Unterbrechungsmelder an)
- Eingangssymbole:
  - e … einschalten
  - a … ausschalten
  - v … Vorarlarm ausschalten
  - b … Alarm Bewegungsmelder
  - u … Alarm Unterbrechungsmelder
  - q … Alarm quittieren
- Ausgabesymbole:
  - b … Vorbeugende Maßnahmen: alle Warnmelder aktiviert
  - s … Sicherungsmaßnahmen: Schließen aller Tore
  - l … Alarmmaßnahmen: Warnsirene/Scheinwerfer an

Kellerautomat – Definition

(engl. push down automaton)

- KA = (E, Z, K, δ, z_0, k_0, F)
  - $\delta : Z \times (E \cup \{\varepsilon\}) \times K \to P(Z \times K^*)$
  - $e \in E^*$ … Eingabewort
  - $z \in Z^*$ … Wort, das alle Zustände in der Reihenfolge enthält, die der Automat einnimmt.
  - $k \in K^*$ … aktuelles Wort im Stack. Mit k_0 als oberstes Element.
- prinzipielle Arbeitsweise
  - $e_j \in E \cup \{\varepsilon\}$
  - $(z_{i+1}, l) \in \delta(z_i, e_j, k_0)$
  - dann
    - Zustand z_{i+1}
    - k_0 durch $l = l_0 ... l_n \in K^*, n \in \mathbb{N}_0$ ersetzt (in der Reihenfolge von oben nach unten).

Übergänge, Halten, Akzeptieren

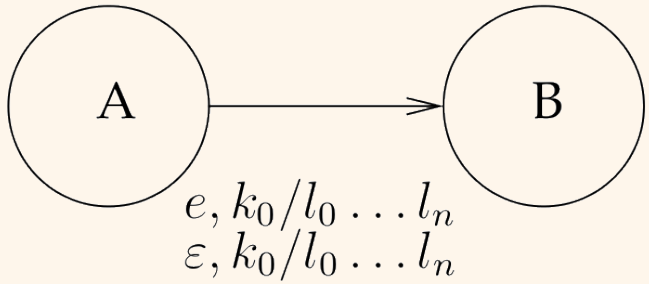
- 2 Arten von Zustandsübergängen
  - mit Lesen eines Eingabezeichens: $\delta(z_i, e_i, k_0) \neq \{\}$
  - ohne Lesen eines Eingabez. (spontan): $\delta(z_i, \varepsilon, k_0) \neq \{\}$
- Haltebedingungen
  - der Stack leer ist
  - Eingabewort gelesen & kein spontaner Übergang möglich
  - Eingabewort nicht gelesen ist & kein Übergang möglich
- Akzeptanzbedingungen
  - zustandsakzeptiert: Endzustand erreicht
  - kellerakzeptiert: Stack leer
  - akzeptiert: Endzustand erreicht und Stack leer

Zustandsübergangsfunktion

- Zustandstabelle

	Eingabesymbol	…	ε
Zustand,Kellersymbol	Menge von (Folgezustand, Wort aus K^*)	…	…
…	…	…	…

- Zustandsdiagramm



Beispiel

- $E = \{0, 1\}, Z = \{A, B\}, F = \{B\}, K = \{\bot, 0, 1\}$
- $z_0 = A, k_0 = \bot$
- δ gemäß folgender Zustandstabelle:

	0	1
$A, \bot$	$(B, \bot)$	$(A, 1\bot)$
$A, 0$	–	–
$A, 1$	(A, ε)	$(A, 11)$
$B, \bot$	$(B, 0\bot)$	$(A, \bot)$
$B, 0$	$(B, 00)$	(B, ε)
$B, 1$	–	–

- Akzeptierte Sprache: $L(KA) = \{w \in \{0, 1\}^* | \#0 \qquad \#1\}$

Konfigurationen und Züge

- Konfiguration
  - Tripel $(z_i, w, k) \in Z \times E^* \times K^*$
  - geben an:
    - den momentanen Zustand $z_i \in Z$
    - den noch zu lesenden Teil $w \in E^*$ des Eingabewortes
    - den Kellerinhalt $k \in K^*$
  - Startkonfiguration ist (z_0, e, k_0)
- Zug
  - Paar von Konfigurationen
  - entweder
    - $((z_i, w, k_0\ l), (z_{i+1}, w, q\ l)), k_0 \in K, q \in K^*$ mit $(z_{i+1}, q) \in \delta(z_i, \varepsilon, k_0)$
    - $((z_i, e_j\ w, k_0\ l), (z_{i+1}, w, q\ l))$ mit $(z_{i+1}, q) \in \delta(z_i, e_j, k_0)$
  - werden üblicherweise so angeschrieben:
    - $(z_i, w, p\ l) \vdash (z_{i+1}, w, q\ l)$ bzw.
    - $(z_i, e_j w, p\ l) \vdash (z_{i+1}, w, q\ l)$

Beispiele

- KA aus vorhergehendem Beispiel; $e = 11000$
  - Züge: $(A, 11000, \bot) \vdash (A, 1000, 1\bot) \vdash (A, 000, 11\bot) \vdash (A, 00, 1\bot) \vdash (A, 0, \bot) \vdash (B, \varepsilon, \bot)$
  - d.h.: e vollständig gelesen, Endzustand erreicht, Stack bis auf Startsymbol leer; Stack kann in diesem Fall nicht leer werden!
- KA aus vorhergehendem Beispiel; $e = 011000$
- KA aus vorhergehendem Beispiel; $e = 1101100$

Beispiel

- $E = \{0, 1\}, Z = \{A, B\}, F = \{B\}, K = \{\bot, 0, 1\}$
- $z_0 = A, k_0 = \bot$
- δ gemäß folgender Zustandstabelle:

	0	1	ε
$A, \bot$	$(A, 0\bot)$	$(A, 1\bot)$	(B, ε)
$A, 0$	$(A, 00)$	(A, ε)	–
$A, 1$	(A, ε)	$(A, 11)$	–

- $L(KA) = \{w \in \{0, 1\}^* | \#0 \qquad \#1\}$
- Dieser Automat ist nicht deterministisch!

Beispiel

- $E = \{0, 1\}, Z = \{A, B, C\}, F = \{C\}, K = \{\bot, 0, 1\}$
- $z_0 = A, k_0 = \bot$
- δ gemäß folgender Zustandstabelle:

	0	1	ε
$A, \bot$	$(A, 0\bot)$	$(A, 1\bot)$	$(B, \bot)$
$A, 0$	$(A, 00)$	$(A, 10)$	$(B, 0)$
$A, 1$	$(A, 01)$	$(A, 11)$	$(B, 1)$
$B, \bot$	–	–	$(C, \bot)$
$B, 0$	(B, ε)	–	–
$B, 1$	–	(B, ε)	–
$C, \bot$	–	–	–
$C, 0$	–	–	–
$C, 1$	–	–	–

- $L(KA) = \{w \in \{0, 1\}^* | w = w_1 w_2, w_1 = w_2 \quad \}$
- Dieser Automat ist nicht deterministisch!

Deterministischer KA (DKA)

- KA deterministisch, wenn
  - für alle $z_i \in Z$, $E_i \in E$ und $k_0 \in K$ gilt:
    - $\#\delta(z_i, E_i, k_0) + \#\delta(z_i, \varepsilon, k_0) \leq 1$
- D.h. für jeden Zustand und für jedes Zeichen an der Kellerspitze gibt es höchstens eine Möglichkeit, mit oder ohne Eingabe den Zustand zu wechseln und das Kellerzeichen zu ersetzen.
- Beispiel
  - DKA soll folgende Sprache akzeptieren:

$$L(KA) = \{w_1\$w_2 | w_1, w_2 \in \{0,1\}^*, w_1 = w_2^T\}$$

  - ges.: Zustandstabelle und Zustandsdiagramm

Turingmaschine (TM) – Überblick

- wahlfreier Zugriff auf den Arbeitsspeicher
  - vgl. Stack beim Kellerautomaten
- TM: *universelles* Maschinenmodell zur Realisierung von Algorithmen
  - hauptsächlich in der theoretischen Informatik
  - Turings Vorstellung
    - Endlicher Automat mit einem unendlichen Speicherband
    - Lese/Schreibkopf, der sich auf dem Band bewegen kann.
    - Felder des Bandes: Buchstaben des Bandalphabets
    - Zeichen unter dem Kopf: lesen/verändern
    - Kopf: um ein Feld nach links/rechts oder an Stelle
  - CPU läßt sich als TM auffassen (aber: endlicher Speicher)
- Linear beschränkter Automat: beschränkter Speicher!

Turingmaschine – Definition

- TM = $(Z, E, B, \delta, z_0, \#, F)$
  - B … Bandalphabet, $E \subseteq B$
  - δ: Z × B → $\mathcal{P}$ (Z × B × \l, r, n\)$
  - $\# \in B - E$ … Leerzeichen
  - l … Bewegung nach links
  - r … Bewegung nach rechts
  - n … keine Bewegung
  - # … Leerzeichen
- Arbeitsweise
  - TM im Zustand z_i, unter Kopf das Bandsymbol $b_m \in B$
  - TM im nächsten Schritt in den Zustand z_{i+1} über
  - schreibt anstelle von b_m ein Symbol $b_n \in B$
  - führt danach eine Bewegung $x \in \{l, r, n\}$
  - Am Anfang steht das Eingabewort **am Band** und der Schreib-/Lesekopf befindet sich am ersten Zeichen.
- Beispiel: $L(G) = \{a^n b^n c^n | n > 0\}$