

Theoretische Informatik

Grundlagen der Automatentheorie

Technische Hochschule Rosenheim

SS 2019

Prof. Dr. J. Schmidt

Inhalt

- Definition von endlichen Automaten
- Darstellung von endlichen Automaten
- Die akzeptierte Sprache von endlichen Automaten
- Erweiterung der Definition von endlichen Automaten
- Kellerautomaten

DEFINITION UND DARSTELLUNG VON AUTOMATEN

Automaten

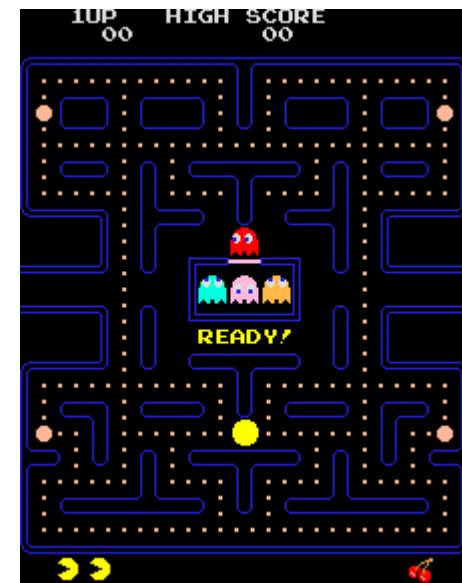
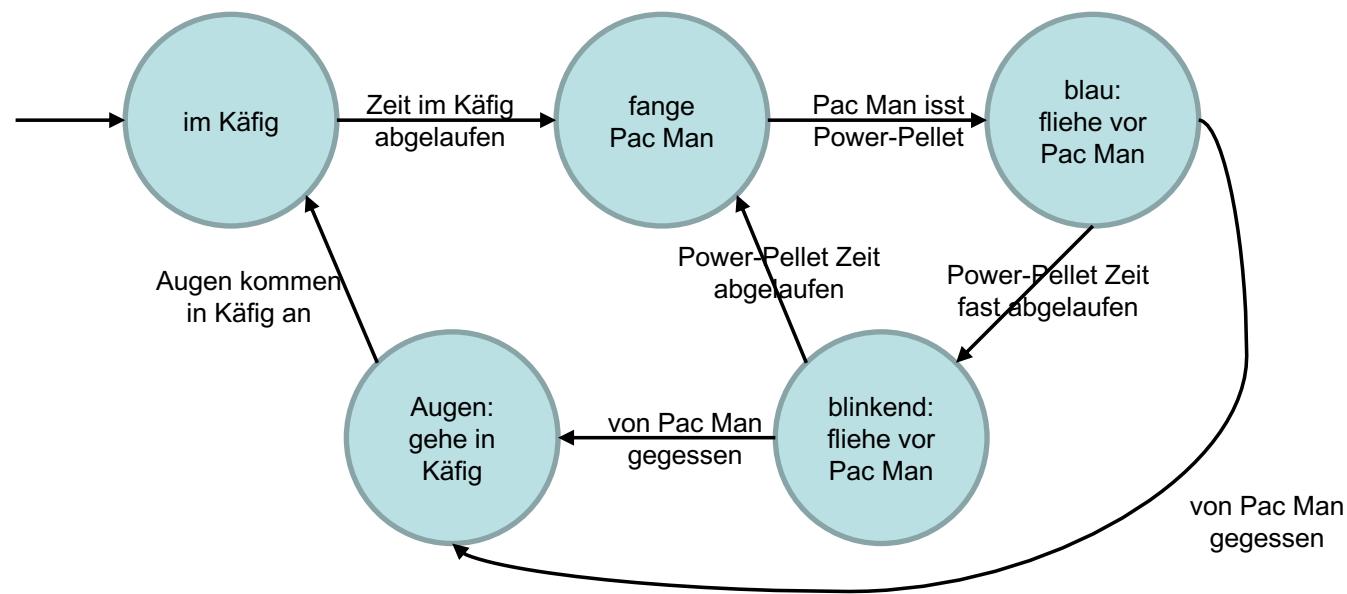
- Automat
 - Maschine, die ihr Verhalten bis zu einem gewissen Grade selbst steuert
 - z.B.: Kaffeautomat
- Für Anwendungen in Wissenschaft und Technik: mathematische Präzisierung des Begriffs notwendig
- Abstraktion:
 - Eingangs-/Ausgangsvariablen sind Ein-/Ausgabezeichen
 - interne Zustände
 - Zustandsübergänge

Automaten und Informatik

- gut zur Analyse und formalisierten Darstellung komplexer Zusammenhänge und Abläufe geeignet
 - zur Modellbildung (Zustände und Zustandsübergänge eines Systems)
- enger Zusammenhang mit
 - Berechenbarkeitstheorie
 - formalen Sprachen
(und damit auch den Grundlagen der Programmiersprachen)
- Beschreibung und Realisierung integrierter Schaltkreise
 - Minimierung von Zuständen und Schaltfunktionen
 - Fragen der Verbindungsoptimierung (Kreuzungsfreiheit)
 - Minimierung der Anzahl der Eingabe- und Ausgabeleitungen
(bestimmen in erster Linie die Kosten)

Beispiel: Pac Man

- Zustandsmodellierung: deterministischer endlicher Automat
- extrem hilfreich, sehr übersichtlich
 - welche Zustandsübergänge sind wann zugelassen?
 - sehr viele Algorithmen für endliche Automaten verfügbar
- Beispiel: Geister in Pac Man



Deterministische Automaten

Ein **deterministischer Automat** $A(S, T, f)$ ist definiert durch:

- Eine abzählbare Menge $S = \{s_1, s_2, s_3, \dots\}$ von **Zuständen**.
(oft auch: $Q = \{q_1, q_2, q_3, \dots\}$)
- Ein **Alphabet** (d.h. eine abzählbare, geordnete Menge)
 $T = \{t_1, t_2, t_3, \dots\}$ von **Eingabezeichen**.
(oft auch: $\Sigma = \{\sigma_1, \sigma_2, \sigma_3, \dots\}$)
- Eine eindeutige **Übergangsfunktion** $f: S \times T \rightarrow S$.
(oft auch: $\delta: Q \times \Sigma \rightarrow Q$)

Dabei ist $S \times T$ das **kartesische Mengenprodukt**, d.h. die Menge aller geordneten Paare (s, t) mit $s \in S$ und $t \in T$.

Übergangsfunktion

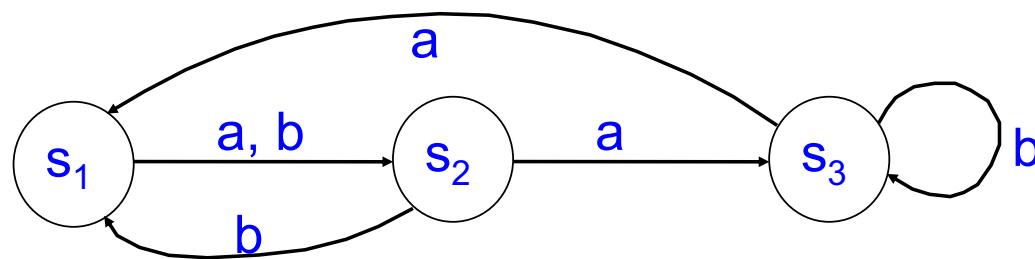
- geschrieben: $f(s_i, t_j) = s_k$ oder $s_i, t_j \rightarrow s_k$
- ist eindeutig (daher: deterministisch)
- aber nicht notwendigerweise umkehrbar

Endliche Automaten

- Sind T und S endlich:
endlicher Automat (finite automaton)
- **DEA: deterministischer endlicher Automat**
(englisch **DFA**)

Übergangsdiagramme

- Übergangsdiagramm:
 - gerichtete Graphen
 - Knoten = Zustände
 - Kanten = Übergänge
 - von jedem Knoten gehen so viele Pfeile aus, wie es Eingabezeichen gibt
 - Zustände werden in Knoten, Eingabezeichen an Kanten notiert
- Beispiel



	s_1	s_2	s_3
s_1			
s_2			
b			

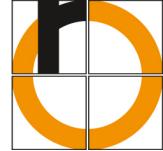
Übergangstabellen

- bei DEA:
 - ⊕ Menge der möglichen Übergänge zwischen den Zuständen endlich
 - ⊕ höchstens z^n (n Anzahl der Zustände in S , z : #Zeichen in T)
 - ⊕ Übertragungsfunktion lässt sich dann in Form einer endlichen **Übergangstabelle** darstellen
- Beispiel
 - ⊕ Übergangstabelle eines DEA mit den Eingabezeichen $T = \{a,b\}$ und den Zuständen $S = \{s_1, s_2, s_3\}$
 - ⊕ Übergangsfunktion wird durch Tabelleneinträge definiert:

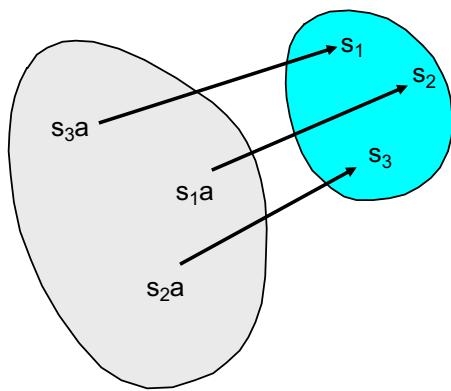
	s_1	s_2	s_3
a	s_2	s_3	s_1
b	s_2	s_1	s_3

- ⊕ bei NEA: Menge von Folgezuständen

Nichtdeterministische Automaten

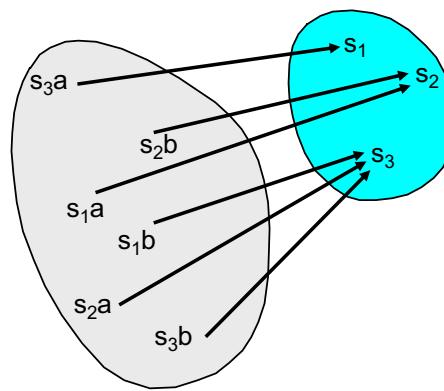


- Automat ist nichtdeterministisch, wenn Übergangsabbildung $f: S \times T \rightarrow S$ nicht eindeutig ist
- f ist dann keine Funktion, sondern eine Relation

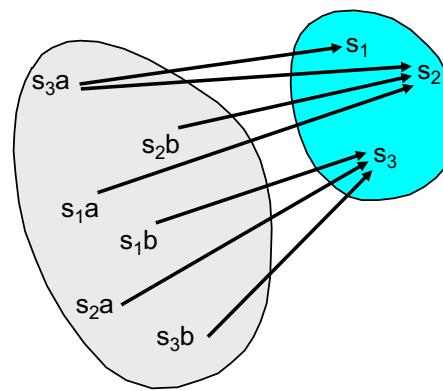


eineindeutig

deterministisch



eindeutig



nicht eindeutig

nichtdeterministisch

Nichtdeterministische Automaten

- Abkürzung: **NEA** bzw. NFA
- für mindestens einen Zustand gibt es nicht nur einen einzigen möglichen Folgezustand bei einem bestimmten Eingabezeichen, sondern eine Menge von Folgezuständen
- Verhalten vorstellbar als
 - „zufällige“ Auswahl eines Folgezustands (aber, so dass es am Ende passt!)
 - parallele Verarbeitung aller möglichen Folgezustände
- Überraschend:
 - ein NEA ist nicht leistungsfähiger als ein DEA

Automaten mit Ausgabe

Automat mit Ausgabe $A(S, T, f, Y, g)$ besteht aus

- Alphabet $Y = \{y_1, y_2, y_3, \dots\}$ von **Ausgabezeichen** und
- einer nicht notwendigerweise eindeutigen Abbildung g in die Menge der Ausgabezeichen Y

Mealy-Automat: $g: S \times T \rightarrow Y$, Ausgabe hängt ab vom

- Eingabezeichen
- aktuellen Zustand

Moore-Automat: $g: S \rightarrow Y$, Ausgabe hängt nur ab vom

- aktuellen Zustand

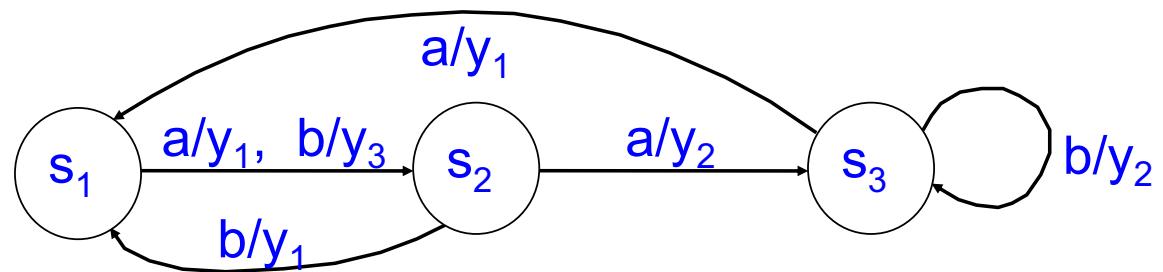
Automaten mit Ausgabe

- zu jeder Folge von Eingabezeichen wird eine Folge von Ausgabezeichen generiert
- Automaten mit Ausgabe = übersetzende Automaten (Transduktoren)
- Sind S , T und Y des übersetzenden Automaten endlich:
endlicher Übersetzer

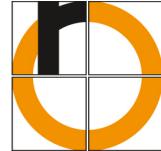
Automat mit Ausgabe

- entweder zweite Tabelle für Ausgabefunktion
- oder Erweiterung der Übergangstabelle um Ausgabezeichen
- Beispiel (Mealy)
 - zusätzlich zu vorher noch Ausgabezeichen $Y = \{y_1, y_2, y_3\}$
 - erweiterte(s) Übergangstabelle/-diagramm:

	s_1	s_2	s_3
a	s_2, y_1	s_3, y_2	s_1, y_1
b	s_2, y_3	s_1, y_1	s_3, y_2



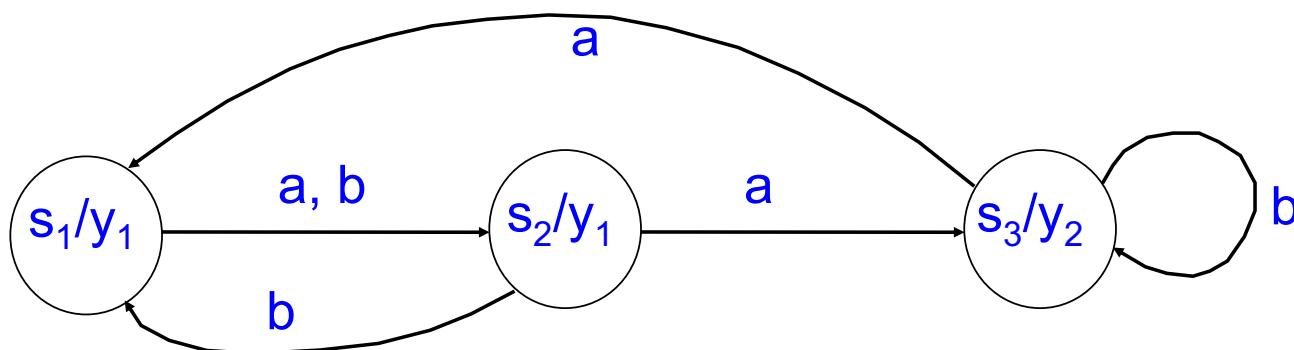
Modifiziertes Beispiel: Moore-Automat



	s_1	s_2	s_3
a	s_2, y_1	s_3, y_1	$s_1, y_{\cancel{2,1}}$
b	s_2, y_1	s_1, y_1	s_3, y_2

oder

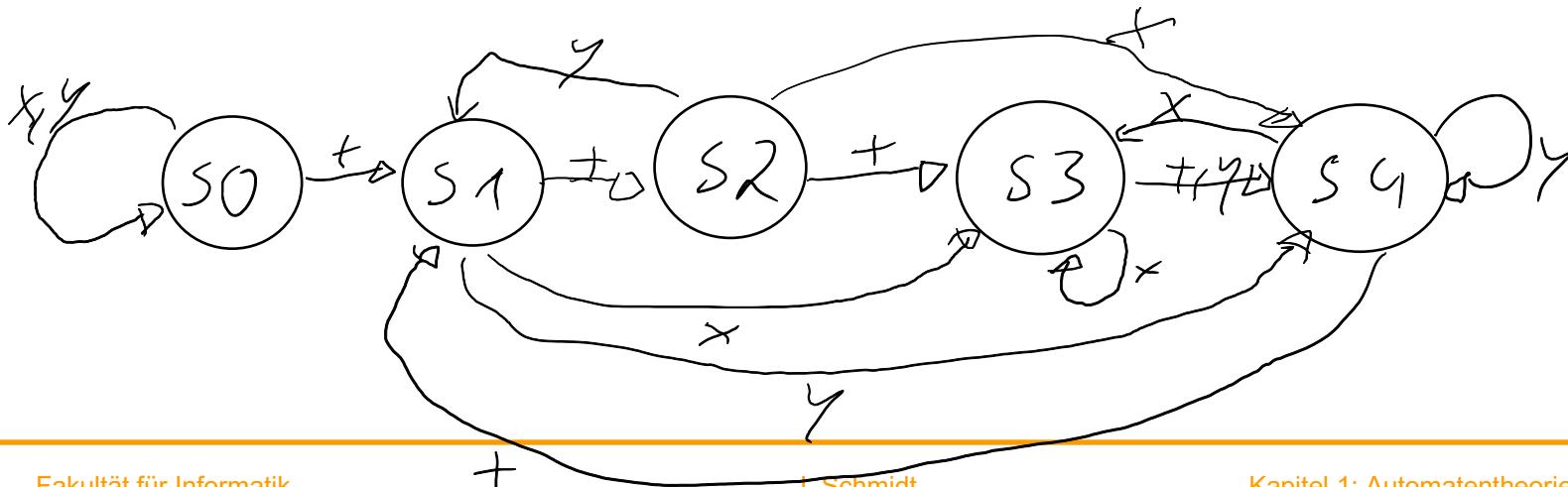
	s_1/y_1	s_2/y_1	s_3/y_2
a	s_2	s_3	s_1
b	s_2	s_1	s_3



Aufgabe

Zeichnen Sie das Übergangsdiagramm eines Automaten mit folgender Übergangstabelle:

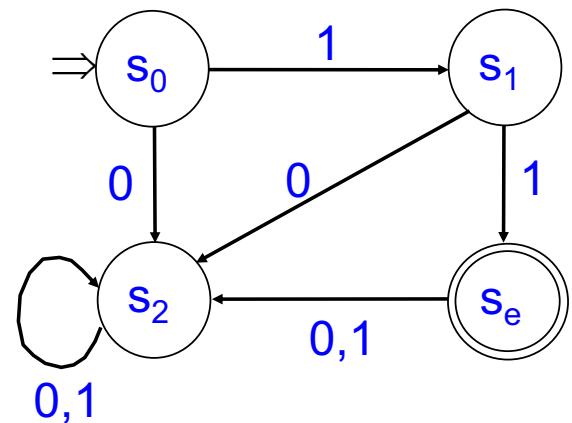
	s0	s1	s2	s3	s4
x	s0	s3	s4	s3	s3
y	s0	s4	s1	s4	s4
+	s1	s2	s3	s4	s1



AKZEPTIERTE SPRACHE VON AUTOMATEN

Besondere Zustände

- **Anfangszustand** $s_0 \in S$, markiert mit Pfeil \Rightarrow
- mindestens ein **akzeptierender Zustand** oder **Endzustand** $s_e \in S$, zusammengefasst in Menge der Endzustände $E \subseteq S$ markiert durch **Doppelkreis**
- Automat dann charakterisiert durch
 - $A(S, T, f, s_0, E)$ ohne Ausgabe
 - $A(S, T, Y, f, g, s_0, E)$ mit Ausgabe



Akzeptierte Sprache

➤ Von A akzeptierte Sprache $L(A)$

- Menge aller aus den Zeichen des Alphabets T der Eingabezeichen bildbaren Wörter, die einen Automaten $A(S, T, f, s_0, E)$ vom Anfangszustand s_0 in einen Endzustand $s_e \in E$ überführen
- Mengenschreibweise: $L(A) := \{w \in T^* \mid (s_0, w) \rightarrow s_e \in E\}$

➤ Mächtigkeit $|L|$: Anzahl der Wörter der Sprache

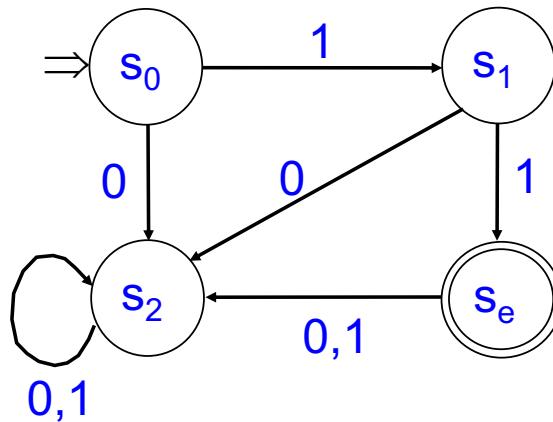
Erkennende Automaten

- Automaten mit Sprache L können von Wort $w \in T^*$ erkennen, ob es zu L gehört oder nicht
- Vorgehen:
 - ⊕ Automat ist im Startzustand s_0
 - ⊕ verwende einzelne Zeichen des Wortes w als Eingabe
 - ⊕ wenn sich Automat nach dem letzten Zeichen in einem Endzustand befindet, dann gilt $w \in L$, sonst $w \notin L$
- solche Automaten heißen
erkennende Automaten oder **Akzeptoren**
- Abgrenzung zu übersetzenden Automaten:
diese transformieren ein Eingabewort in ein Ausgabewort

Erkennende Automaten

Beispiel 1

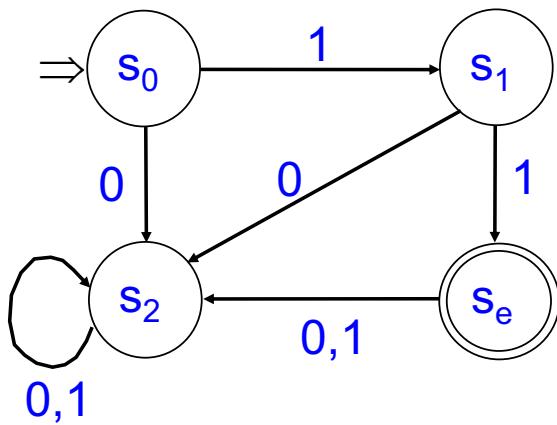
- Automat mit $T = \{0, 1\}$ und $S = \{s_0, s_1, s_2, s_e\}$
- akzeptierte Sprache L enthält nur ein Wort: 11
- $|L|=1$



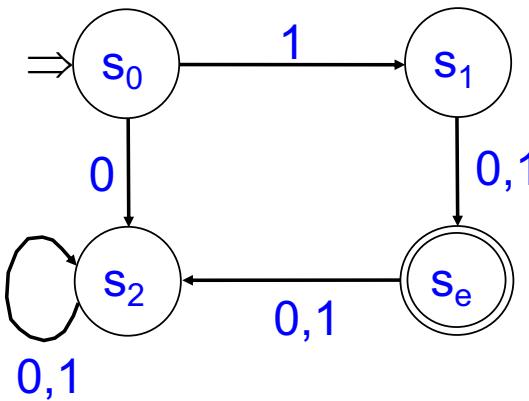
Erkennende Automaten

Beispiel 2

- Erweiterung: Automat akzeptiert 2 Wörter: 11 und 10



akzeptiertes Wort: 11

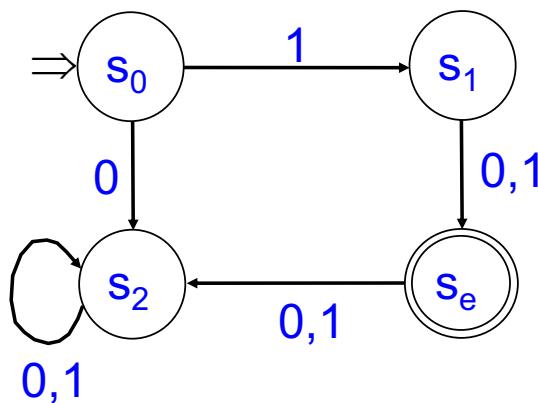


akzeptierte Wörter: 11 und 10

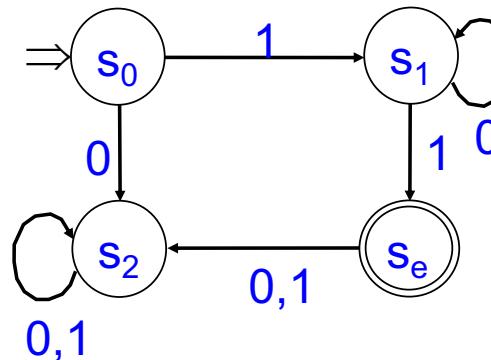
Erkennende Automaten

Beispiel 3

- Erweiterung: Automat akzeptiert unendliche viele Wörter der Form 10^n1
- 0^n : Zeichenkette aus n aufeinander folgenden Nullen
- mit $n \in \mathbb{N}_0$, so darf n auch 0 sein (d.h., das Wort enthält keine 0)
- $L(A, s_a, s_e) = \{10^n1 \mid n \in \mathbb{N}_0\}$, $|L| = \infty$

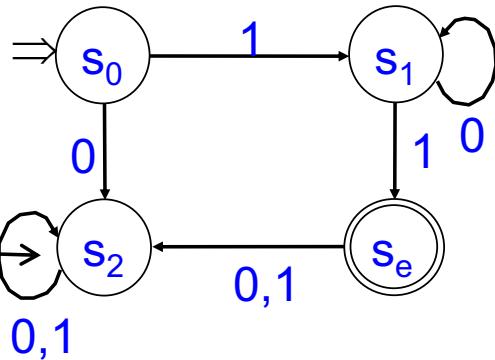


akzeptierte Wörter: 11 und 10



akzeptierte Wörter: 10^n1

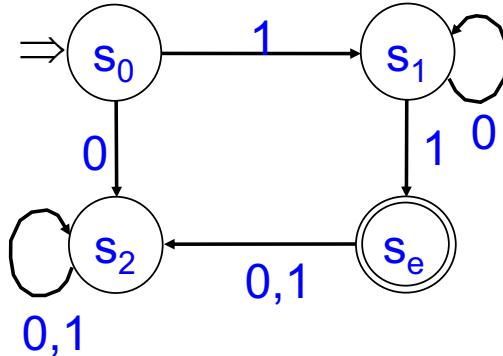
Fangzustände



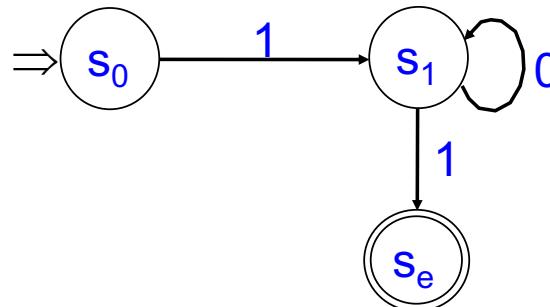
Fangzustand:

- Automat bleibt für jede beliebige Eingabe in diesem Zustand
- Bezeichnung: s_f

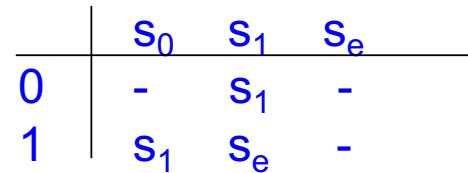
Unvollständige Automaten



vollständiger Automat



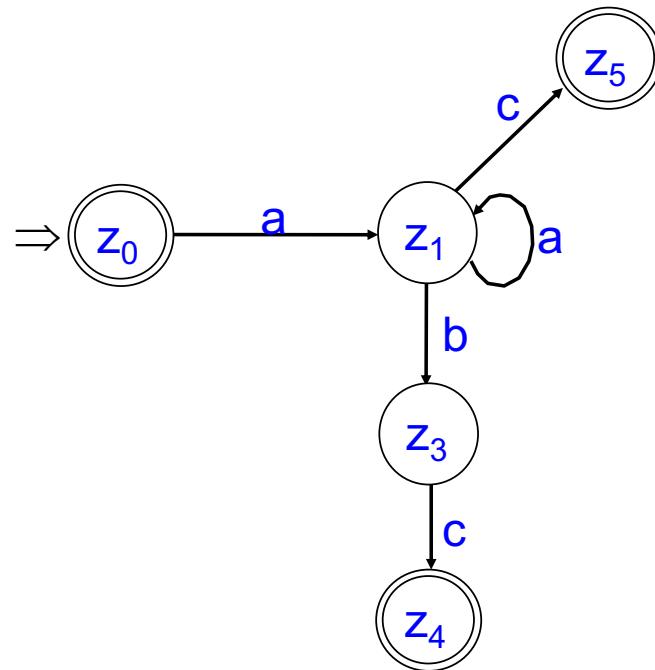
unvollständiger Automat



- **Fangzustände** werden oft **weggelassen** (Übersichtlichkeit)
- Vereinbarung: alle nicht berücksichtigten Übergänge führen in einen Fangzustand
- Automat mit Ausgabe: ggf. Ausgabe von Fehlermeldung
- Bezeichnung: **unvollständiger Automat**

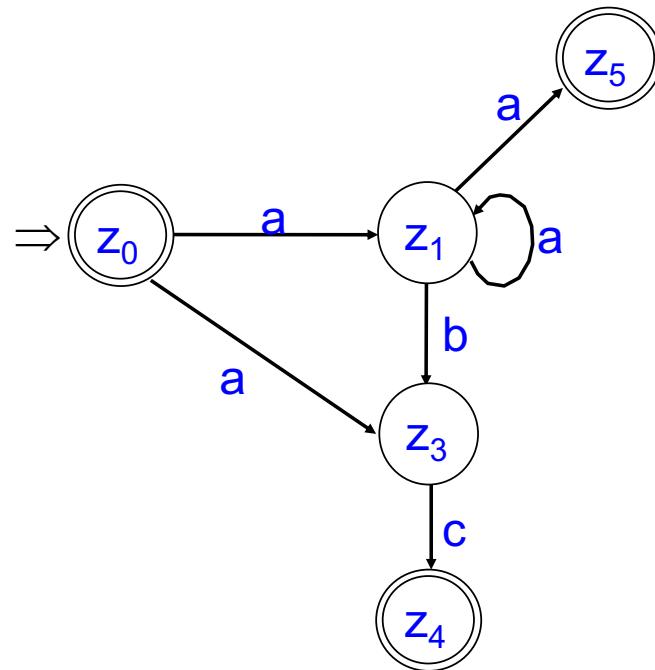
Aufgabe

Welche Sprache akzeptiert der folgende Automat?



Aufgabe

Welche Sprache akzeptiert der folgende Automat?



Das Wortproblem

- **Wortproblem:** Erkennung, ob ein gegebenes Wort zur Sprache L des erkennenden Automaten gehört oder nicht
- Anwendungsbeispiele:
 - Pattern Matching:
Finden einer bestimmten Zeichenkette in einem Text
 - Lexikalische Analyse bei Compilern:
 - Entscheidung, ob eine Zeichenfolge vorgegebenen Regeln einer Programmiersprache folgt
 - Zerlegung in elementare Einheiten, z.B. Bezeichner, Schlüsselwörter, ...

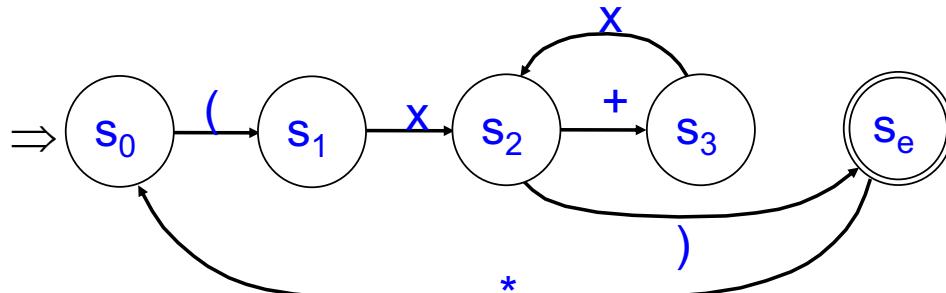
Das Wortproblem

Beispiel

Analyse von Klammerterminen

- Prüfung, ob Terme der Art $(x+x)^*(x+x+x)$ korrekt gebildet sind
- Alphabet (Eingabezeichen) $T = \{(,), +, *, x\}$
- Zustände: $S = \{s_0, s_1, s_2, s_3, s_e\}$

	s_0	s_1	s_2	s_3	s_e
(-	-	-	-
)	-	-	s_e	-	-
+	-	-	s_3	-	-
*	-	-	-	-	s_a
x	-	s_2	-	s_2	-



ERWEITERUNG DER DEFINITION VON AUTOMATEN

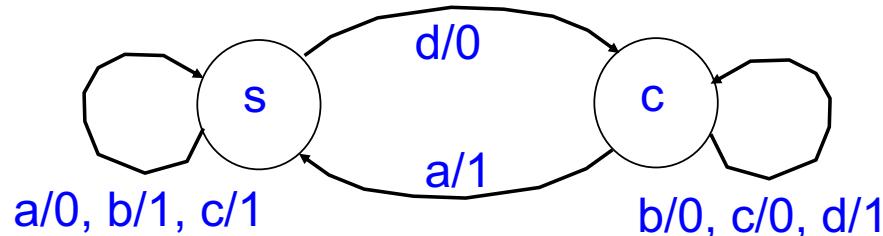
Mehrere Eingänge

- Manche Probleme erfordern mehr als einen Eingang
 - Automat verfügt nach Definition nur über einen Eingang
 - erscheint erst als nicht ausreichend
-
- Lösung:
 - ❖ Mengen der simultan einzugebenden Zeichen als neue Zeichen eines erweiterten Alphabets auffassen
 - ❖ Automat konstruieren, der wiederum über nur einen Eingang verfügt

Mehrere Eingänge Beispiel

- Addition von zwei Binärziffern, $T = \{0, 1\}$
- es müssen 2 Operanden gleichzeitig verarbeitet werden
- generiere neues Alphabet $T = \{a,b,c,d\}$ mit Zuordnungen:
a: $0 + 0 = 0$, b: $0 + 1 = 1$,
c: $1 + 0 = 1$, d: $1 + 1 = 0$ mit Übertrag (Carry) 1
- Ausgabe: Additionsergebnis (0 oder 1)
- Zustände: Übertrag (c) oder nicht (s), d.h. $S=\{s, c\}$

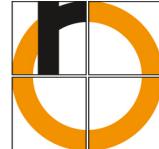
	s	c
a	s,0	s,1
b	s,1	c,0
c	s,1	c,0
d	c,0	c,1



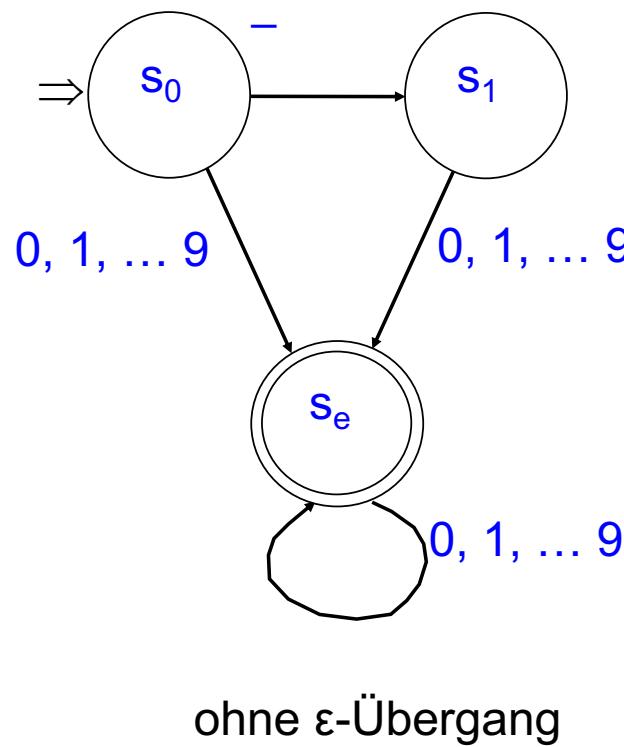
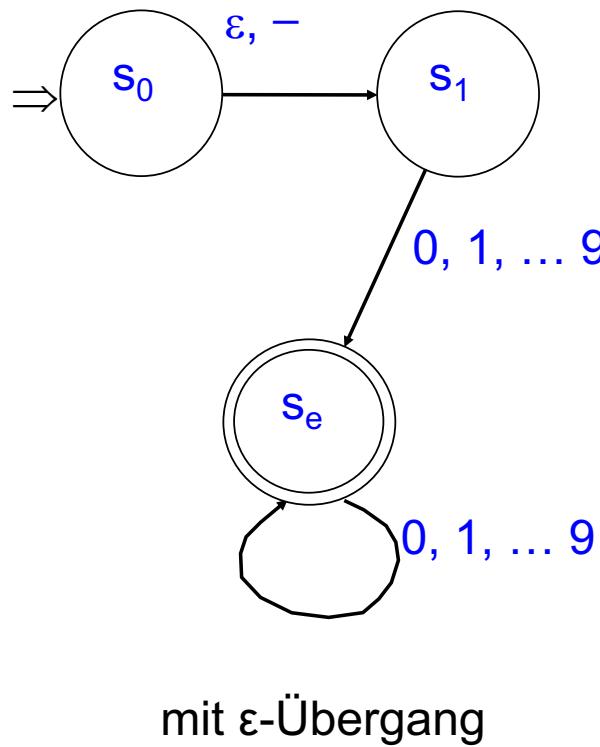
ϵ -Übergänge

- DEA ist **deterministisch**: Übergang vom aktuellen Zustand in den Folgezustand ist eindeutig
- DEA und NEA sind auch **kausal**: Ursache für den Übergang ist durch das Auftreten eines bestimmten Eingabezeichens eindeutig bestimmt
- Kausalität kann eingeschränkt werden: Zulassen **spontaner Zustandsübergänge** (nicht durch Eingabezeichen verursacht)
- Bezeichnung: ϵ -Übergänge (auch: λ -Übergänge)
- wird in Übergangstabellen/-graphen durch das leere Zeichen ϵ angedeutet
- ϵ gehört nicht zur Menge T der Eingabezeichen, sondern ist Bestandteil der Übergangsfunktion
- **Jeder Automat mit ϵ -Übergängen kann durch einen DEA mit derselben Sprache ohne solche Übergänge ersetzt werden**

ε -Übergänge Beispiel



Automat für Integer-Zahlen mit optional vorangestelltem Minus-Zeichen



Transformation NEA → DEA

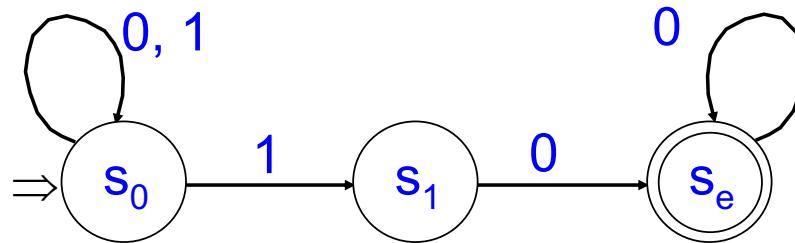
- Zu **jedem** erkennenden NEA kann ein DEA konstruiert werden, der die gleiche Sprache akzeptiert
- Für den Beweis erhielten M. Rabin und D. Scott 1976 den Turing Award
- Grundidee:
 - geg.: NEA mit n Zuständen
 - ermittle alle Übergänge von Teilmengen der Zustandsmenge S
 - Menge aller Teilmengen: **Potenzmenge**; enthält 2^n Elemente
 - der entstehende DEA kann dadurch mehr Zustände und Übergänge aufweisen als der ursprüngliche NEA
 - meist wächst aber die Anzahl der Zustände nur wenig oder bleibt sogar gleich

Transformation NEA → DEA

Beispiel

- NEA mit $T = \{0,1\}$ und $S = \{s_0, s_1, s_e\}$
- akzeptierte Sprache: $L = \{ x10^n \mid x \in T^*, n \in \mathbb{N} \}$

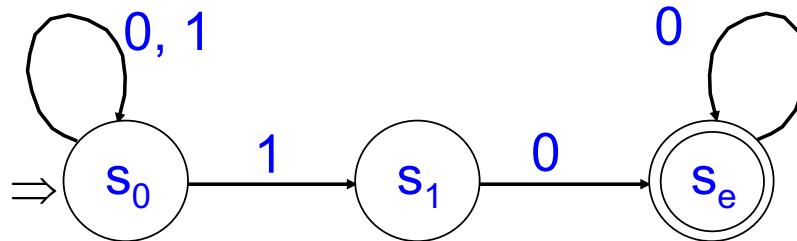
	s_0	s_1	s_e
0	s_0	s_e	s_e
1	s_0, s_1	-	-



Transformation NEA → DEA

Beispiel

	s_0	s_1	s_e
0	s_0	s_e	s_e
1	s_0, s_1	-	-



- Beginne mit allen von der Teilmenge $\{s_0\}$ aus direkt erreichbaren Mengen von Folgezuständen

	$\{s_0\}$
0	$\{s_0\}$
1	$\{s_0, s_1\}$

- Nimm alle noch nicht betrachteten Teilmengen als neuen Zustand (neue Spalte) hinzu

	$\{s_0\}$	$\{s_0, s_1\}$
0	$\{s_0\}$	$\{s_0, s_e\}$
1	$\{s_0, s_1\}$	$\{s_0, s_1\}$

	$\{s_0\}$	$\{s_0, s_1\}$	$\{s_0, s_e\}$
0	$\{s_0\}$	$\{s_0, s_e\}$	$\{s_0, s_e\}$
1	$\{s_0, s_1\}$	$\{s_0, s_1\}$	$\{s_0, s_1\}$

Transformation NEA → DEA

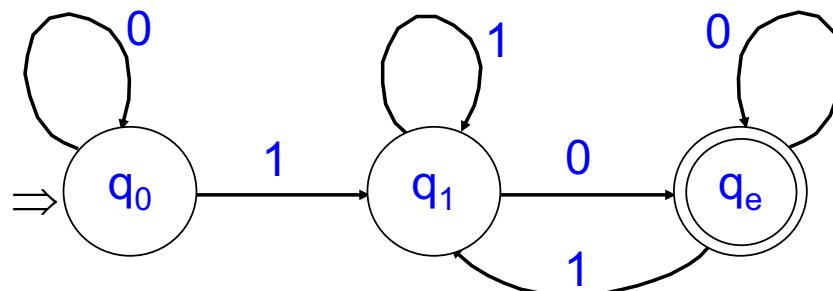
Beispiel

- Umbenennung der Zustände (übersichtlicher)

	$\{s_0\}$	$\{s_0, s_1\}$	$\{s_0, s_e\}$
0	$\{s_0\}$	$\{s_0, s_e\}$	$\{s_0, s_e\}$
1	$\{s_0, s_1\}$	$\{s_0, s_1\}$	$\{s_0, s_1\}$

	q_0	q_1	q_e
0	q_0	q_e	q_e
1	q_1	q_1	q_1

- von 8 möglichen Zuständen der Potenzmenge werden nur 3 benötigt. Potenzmenge:
 $\{ \{ \}, \{s_0\}, \{s_1\}, \{s_e\}, \{s_0,s_1\}, \{s_0,s_e\}, \{s_1,s_e\}, \{s_0,s_1,s_e\} \}$
- Resultierender Graph des DEA,
 $S = \{q_0, q_1, q_e\}$, $L = \{ x10^n \mid x \in T^*, n \in \mathbb{N} \}$



Transformation NEA → DEA

Konstruktionsvorschrift

1. geg.: unvollständige Darstellung des NEA ohne Fangzustände.
2. Schreibe alle Eingabezeichen in die erste Spalte einer Tabelle.
3. Schreibe die Teilmenge, die alle Anfangszustände des NEA enthält, als Kopf in die zweite Spalte der Tabelle. Dies ist der Anfangszustand DEA.
4. Trage in die folgenden Zeilen der zweiten Spalte die Teilmengen ein, die vom Anfangszustand bei Eingabe des entsprechenden Eingabezeichens erreicht werden.
5. Schreibe die in Spalte eins neu aufgetretenen Teilmengen in die folgenden Spaltenköpfen.
6. Trage in die Zeilen der folgenden Spalten die Teilmengen ein, die von den in den Spaltenköpfen stehenden Teilmengen aus mit den jeweiligen Eingabezeichen erreicht werden.
7. Verfare mit den in der aktuellen Spalte neu aufgetretenen Teilmengen analog durch Anhängen und Ausfüllen weiterer Spalten, bis keine neuen Teilmengen mehr auftreten.
8. Die gefundenen Teilmengen entsprechen den Zuständen des gesuchten DEA.
9. Die Übergänge des DEA ergeben sich direkt aus den Tabelleneinträgen.
10. Alle Teilmengen, die einen Endzustand des NEA enthalten, sind Endzustände des DEA.
11. Der Übersichtlichkeit halber wird man die als Zustände des DEA resultierenden Zustandsteilmengen des NEA mit neuen Namen belegen.

Dieses Ausschöpfungsverfahren endet auf jeden Fall nach endlich vielen Schritten, nämlich maximal nach 2^n , da dies der Anzahl der Teilmengen der Potenzmenge entspricht.
→ Zeitkomplexität $O(2^n)$

Äquivalenz von Automaten

Minimale Automaten

Äquivalenz

- Zwei Automaten heißen **äquivalent**, wenn sie identisches Ein-/Ausgabeverhalten zeigen, bzw. (im Falle erkennender Automaten) dieselbe Sprache akzeptieren

Minimaler Automat

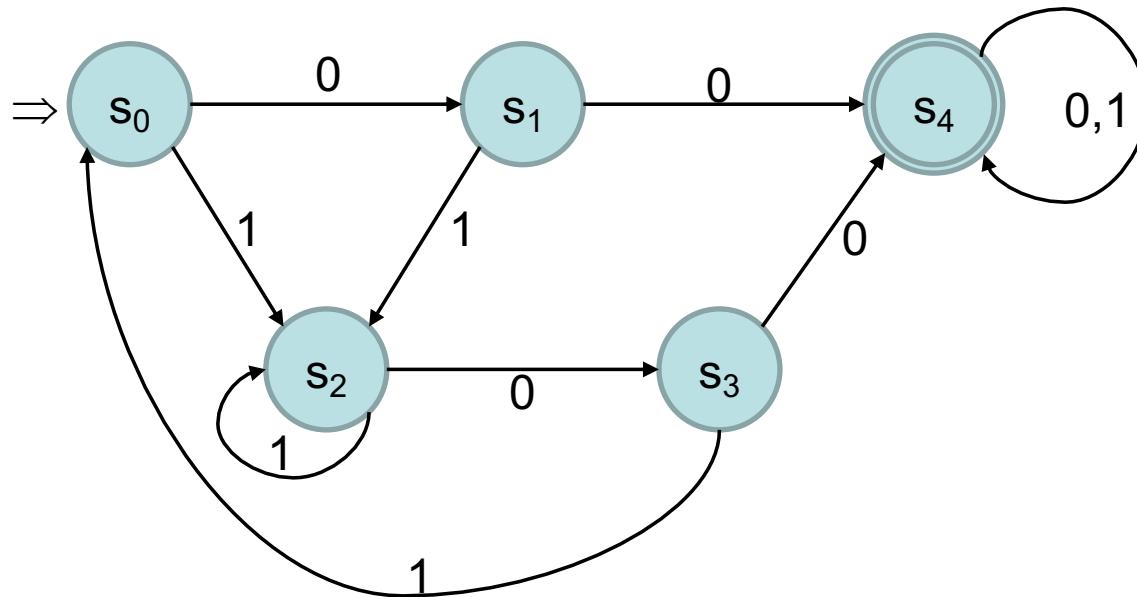
- derjenige unter den äquivalenten Automaten, der die kleinste Anzahl an Zuständen hat
- für DEAs lässt sich stets der zugehörige minimale Automat konstruieren
- praktisch bedeutsam z.B. in der Schaltungstechnik oder im Chip-Design

Minimalautomat – Konstruktion

- Idee: Bildung von Äquivalenzklassen
- Iteration, bis sich keine neuen Klassen mehr bilden
- Ausgangspunkt: Zustandsübergangstabelle
- Ende: alle Zustände einer Klasse geben einen neuen Zustand

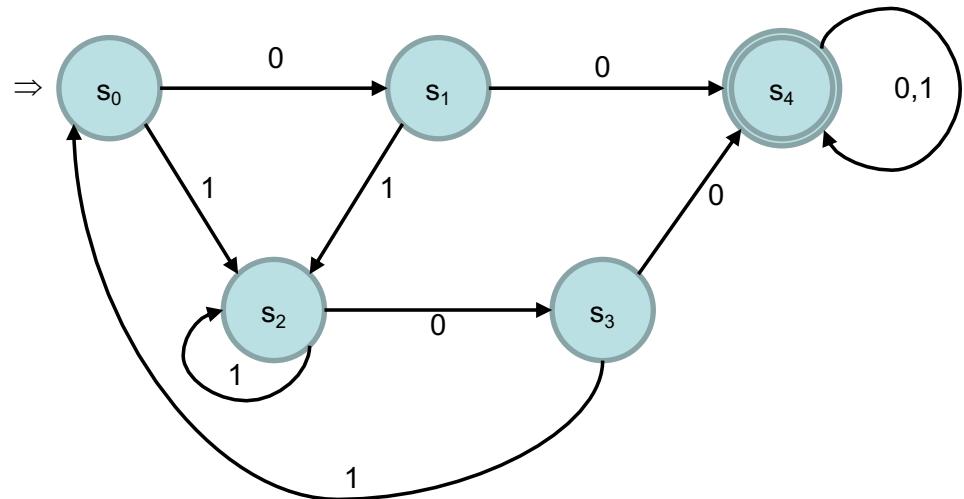
Minimalautomat – Konstruktion

Beispiel:



	s₀	s₁	s₂	s₃	s₄
0	s ₁	s ₄	s ₃	s ₄	s ₄
1	s ₂	s ₂	s ₂	s ₀	s ₄

Minimalautomat – Konstruktion



1. Abtrennung der Endzustände in einer einzigen eigenen Partition

	P1				P2
	s ₀	s ₁	s ₂	s ₃	s ₄
0	s ₁ , P1	s ₄ , P2	s ₃ , P1	s ₄ , P2	s ₄ , P2
1	s ₂ , P1	s ₂ , P1	s ₂ , P1	s ₀ , P1	s ₄ , P2

Minimalautomat – Konstruktion

	P1				P2
	s ₀	s ₁	s ₂	s ₃	s ₄
0	s ₁ , P1	s ₄ , P2	s ₃ , P1	s ₄ , P2	s ₄ , P2
1	s ₂ , P1	s ₂ , P1	s ₂ , P1	s ₀ , P1	s ₄ , P2

2. Weitere Unterteilung jeder Partition, bis diese eine Äquivalenzklasse bilden

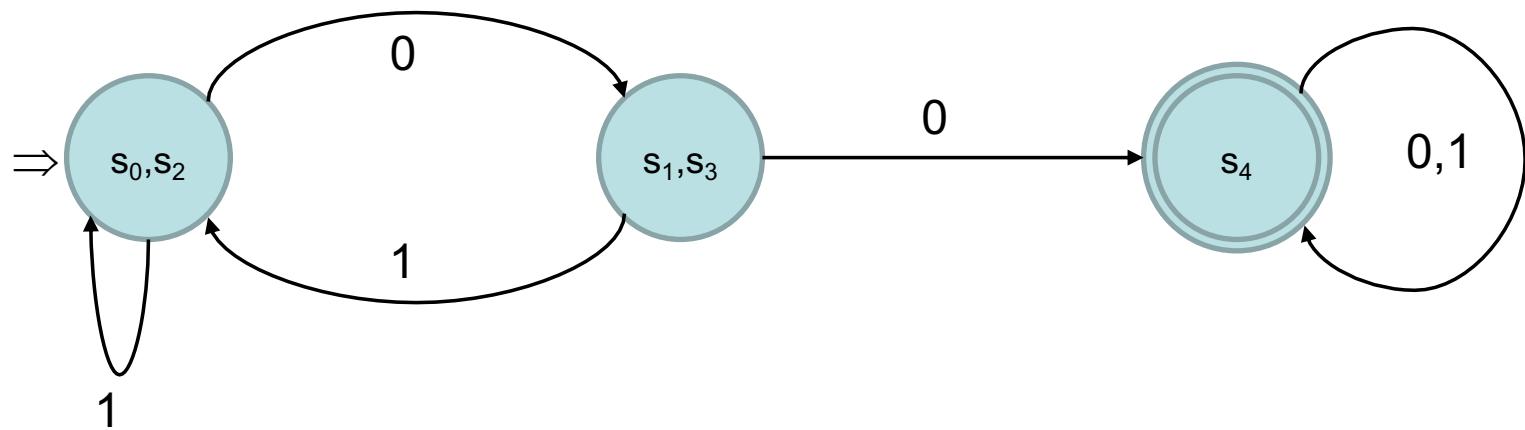
	P1		P2		P3
	s ₀	s ₂	s ₁	s ₃	s ₄
0	s ₁ , P2	s ₃ , P2	s ₄ , P3	s ₄ , P3	s ₄ , P3
1	s ₂ , P1	s ₂ , P1	s ₂ , P1	s ₀ , P1	s ₄ , P3

Minimalautomat – Konstruktion

	P1		P2		P3
	s_0	s_2	s_1	s_3	s_4
0	$s_1, P2$	$s_3, P2$	$s_4, P3$	$s_4, P3$	$s_4, P3$
1	$s_2, P1$	$s_2, P1$	$s_2, P1$	$s_0, P1$	$s_4, P3$

Ergebnis:

- alle Zustände einer Äquivalenzklasse können zusammengefasst werden
- es ergibt sich hier ein Automat mit 3 Zuständen:



Automaten und reguläre Sprachen

- Die von Automaten akzeptierten Sprachen sind mit den **regulären Sprachen** identisch
- Reguläre Sprache:
Teilmengen der Worthalbgruppe über einem Alphabet T , die sich erzeugen lassen durch:
 - Konkatenation,
 - Bilden von Vereinigungsmengen
 - Bildung der Kleeneschen Hülle

Konkatenation

- betrachte Alphabet $T = \{t_1, t_2, t_3, \dots, t_n\}$ mit endlich vielen Zeichen t_k .
- **Kleenesche Hülle** T^* des Alphabets T : $T^* := T^0 \cup T^1 \cup T^2 \cup T^3 \cup \dots$
 - enthält alle aus dem Alphabet bildbaren Wörter
 - einschließlich des leeren Worts ϵ
 - ist unendlich groß
- **Positive Hülle** T^+ des Alphabets T :
 - Kleenesche Hülle ohne das leere Wort
- Operation: **Konkatenation** (Verkettung, Hintereinanderschreiben) von Wörtern (Symbol: \circ)
- das leere Wort ϵ ist das neutrale Element bzgl. dieser Operation
- Beispiel:
 - Alphabet $T = \{a, b\}$
 - $T^* = \{\epsilon, a, b, aa, bb, ab, ba, aaa, \dots\}$
 - $ab \circ abb = ababb, (aa \circ bb) \circ aba = aabbaba$

KELLERAUTOMATEN

Definition

- Endliche Automaten verfügen nicht über Speicher
- jetzt: Erweiterung um einseitig unbegrenzten Speicher
 - Kellerspeicher, Stack
 - Zugriff nur auf oberstes Element möglich
 - Bezeichnung: **Kellerautomat** (Push-down Automaton – PDA)
- Zustandsübergänge sollen nichtdeterministisch sein
- zusätzlich zum Automaten:
 - endliche Menge von Kellerzeichen K
(oft auch: Γ)
 - unterstes Kellerzeichen $\#$
 - Erweiterung der Zustandsübergangsfunktion
 - Übergang zusätzlich abhängig vom obersten Kellerzeichen
 - Zeichen können in der obersten Kellerposition gespeichert werden
- zu Beginn der Berechnung steht im Keller das Zeichen $\#$

Akzeptierte Sprache

- Wörter können akzeptiert werden durch
 - **Endzustände** (wie bei endlichen Automaten), unabhängig vom Kellerinhalt
 - **leeren Keller** – auch das unterste Kellerzeichen # ist dann weg
- für nichtdeterministische Kellerautomaten sind diese beiden Arten äquivalent
- Die von nichtdeterministischen Kellerautomaten akzeptierten Sprachen sind genau die **kontextfreien Sprachen**

Beispiel Klammerausdrücke

- DEA/NEA können keine geschachtelten Klammerausdrücke auf Korrektheit prüfen
- dies ist erst mit Kellerautomaten möglich, da hierfür Speicher benötigt wird
- Beispiel: Schachtelung der Blockstruktur in C mit { }
 - korrekte Schachtelung: { { } { } }
 - falsche Schachtelung: { } } { } {

Beispiel Klammerausdrücke

1. Der Kellerautomat befindet sich im Anfangszustand, der Kellerspeicher auf der Anfangsposition (d.h. Zeichen #)
2. Tritt { als Eingabezeichen auf, so wird es eingekellert, d.h. auf die oberste Kellerposition geschrieben
3. Tritt } als Eingabezeichen auf, so gibt es zwei Möglichkeiten:
 - a) Der Keller befindet sich auf der Anfangsposition. Fehler!
Die Schachtelung ist falsch
 - b) Der Keller befindet sich nicht in der Anfangsposition: Das zuletzt in den Keller geschriebene Zeichen wird gelesen und damit aus dem Keller eliminiert. Jede } löscht eine {
4. Sind alle Eingabezeichen abgearbeitet, so gibt es wieder zwei Möglichkeiten:
 - a) Der Keller befindet sich auf der Anfangsposition: Die analysierte Blockstruktur ist korrekt.
Es wird das Zeichen # gelesen, der Keller ist leer.
 - b) Der Keller befindet sich nicht auf der Anfangsposition: Die analysierte Blockstruktur ist fehlerhaft.

Beispiel 2: Spiegelung

- Eingabealphabet $T = \{a, b\}$
- Kelleralphabet $K = \{A, B, \#\}$
- Sprache $L = \{x_1 x_2 \dots x_n x_n \dots x_2 x_1 \mid x_i \in T\}$
- Zustände $S = \{s_0, s_1\}$
- Notation Zustandsübergang: $a, \# / A\#$ bedeutet:
 - a wurde als Eingabe gelesen
 - # wurde als oberstes Zeichen vom Keller genommen
 - schreibe A# auf Keller

$a, \# / A\#$

$a, A / AA$

$a, B / AB$

$b, \# / B\#$

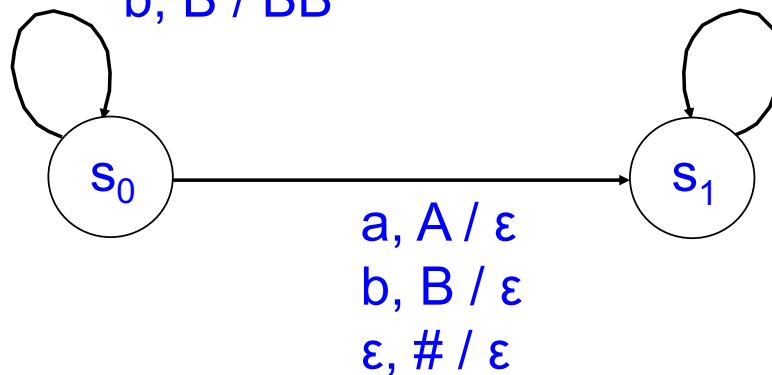
$b, A / BA$

$b, B / BB$

$a, A / \varepsilon$

$b, B / \varepsilon$

$\varepsilon, \# / \varepsilon$



Beispiel 2: Spiegelung Konfigurationen

Abarbeitung des Wortes aabbaa

Eingabe

ϵ

a

a

b

b

a

a

Konfiguration

$(s_0, aabbaa, \#) \xrightarrow{\quad} (s_1, aabbaa, \epsilon)$

$(s_0, abbaa, A\#) \xrightarrow{\quad}$

$(s_0, bbaa, AA\#) \xrightarrow{\quad} (s_1, bbaa, \#) \xrightarrow{\quad} (s_1, bbaa, \epsilon)$

$(s_0, baa, BAA\#) \xrightarrow{\quad}$

$(s_0, aa, BBAA\#) \xrightarrow{\quad}$

$(s_1, aa, AA\#) \xrightarrow{\quad}$

$(s_0, a, ABBAA\#) \xrightarrow{\quad}$

$(s_1, a, A\#) \xrightarrow{\quad}$

$(s_0, \epsilon, AABBAA\#)$

$(s_1, \epsilon, BBAA\#)$

$(s_1, \epsilon, \#) \xrightarrow{\quad} (s_1, \epsilon, \epsilon)$

Anmerkungen

- der PDA für $L = \{x_1 x_2 \dots x_n x_n \dots x_2 x_1 \mid x_i \in \{a, b\}\}$ ist nichtdeterministisch
- es gibt **keinen** deterministischen PDA (DPDA), der diese Sprache akzeptiert
- das nichtdeterministische Verhalten ist notwendig, um die Wortmitte zu „erraten“
- erst durch Markieren der Wortmitte wird es möglich, einen DPDA zu konstruieren, z.B.:
 $L = \{x_1 x_2 \dots x_n 8 x_n \dots x_2 x_1 \mid x_i \in \{a, b\}\}$

PDA – DPDA

- im Gegensatz zu DEA/NEA sind PDA und DPDA **nicht äquivalent** – **PDAs sind mächtiger als DPDAs**
- bei DPDAs ist das Akzeptieren durch leeren Keller verschieden vom Akzeptieren durch Endzustand
 - ⊕ Akzeptieren durch Endzustand ist mächtiger
- DPDAs akzeptieren nur eine echte Untermenge der kontextfreien Sprachen, die **deterministisch kontextfreien Sprachen**
- diese sind äquivalent zu den LR(k) Sprachen ($k > 0$)
 - ⊕ spielen im Compilerbau für die Syntaxanalyse eine große Rolle

Zusammenfassung (1)

- Definition und Darstellung von endlichen Automaten
 - Zustände, Alphabet, Übergänge
 - deterministisch (DEA), nichtdeterministisch (NEA) – äquivalent
 - Automaten mit Ausgabe
 - Übergangstabelle/-graph, baumartiger Graph
- Die akzeptierte Sprache von endlichen Automaten
 - Endzustände (Doppelkreis)
 - Wort akzeptiert, wenn alle Eingabezeichen gelesen und Automat sich in einem Endzustand befindet
 - Fangzustände
 - Automat bleibt immer in diesem Zustand
 - werden wegen Übersichtlichkeit oft weggelassen (unvollständiger Automat)
 - durch DEA/NEA akzeptierte Sprachen identisch zu regulären Sprachen
- Erweiterung der Definition von endlichen Automaten
 - ein einziger Eingang genügt
 - ϵ -Übergänge: Automat nicht mehr kausal
 - Konstruktion einer DEA aus einem NEA

Zusammenfassung (2)

➤ Kellerautomaten

- zusätzlich: Speicher nach Kellerprinzip (Stack), Alphabet für Kellerzeichen
- Wörter werden akzeptiert durch leeren Keller oder Endzustand
 - ⊕ äquivalent für nichtdeterministische Kellerautomaten (PDA)
 - ⊕ Endzustand ist mächtiger für deterministische Kellerautomaten (DPDA)
- akzeptierte Sprache
 - ⊕ PDA: kontextfreie Sprachen
 - ⊕ DPDA: deterministisch kontextfreie Sprachen

Quellen

Die Folien entstanden auf Basis folgender Literatur

- H. Ernst, J. Schmidt und G. Beneken: *Grundkurs Informatik*. Springer Vieweg, 6. Aufl., 2016.
- Schöning, U.: *Theoretische Informatik – kurz gefasst*. Spektrum Akad. Verlag (2008)
- Sander P., Stucky W., Herschel, R.: *Automaten, Sprachen, Berechenbarkeit*, B.G. Teubner, 1992