

# ***Einführung in die Informatik II*** ***Automaten***

Prof. Bernd Brügge, Ph.D  
Institut für Informatik  
Technische Universität München

Sommersemester 2001

2. - 4. Juli 2001

## ***Ziele dieser Vorlesung***

- ❖ Sie können ein Objekt mit interessantem dynamischen Verhalten als endlichen Automaten modellieren.
- ❖ Sie verstehen, dass UML-Zustandsdiagramme eine Notation für endliche Automaten sind.
- ❖ Sie verstehen, dass endliche Automaten und reguläre Grammatiken äquivalente Interpretationen desselben Konzeptes sind.
- ❖ Sie können aus einem endlichen Automaten eine reguläre Grammatik erstellen, die dieselbe Sprache akzeptiert wie der Automat.
- ❖ Sie können für eine gegebene reguläre Grammatik den entsprechenden endlichen Automaten konstruieren.

## ***Modellierung mit endlichen Automaten***

- ❖ Ein Automat ist ein Modell, das zur Modellierung von vielen Problemen verwendet werden kann. Die Ursprünge für endliche Automaten kamen aus folgenden Anwendungsdomänen:
- ❖ **Biologie (McCullough & Pitts, "A logical calculus of the ideas immanent in nervous activity", 1943)**
  - ◆ Modellierung des menschlichen Gehirns: Ein Gehirn besteht aus etwa  $2^{35}$  Neuronen (ca. 35 Milliarden), und es sollte möglich sein, den Zustand jedes Neurons durch einige Bits zu beschreiben.
- ❖ **Elektrotechnik (Mealy, "A method for synthesizing sequential circuits", 1955)**
  - ◆ Entwurf von Schaltkreisen aus einer endlichen Zahl von Gattern, die nur einen von 2 Werten annehmen können.
- ❖ **Linguistik (Chomsky "Three models for the description of language", 1956)**
  - ◆ Beschreibung von Grammatiken für die natürliche Sprache

## ***Endliche Automaten in der Informatik***

- ❖ **Interaktive Systeme:**
  - ◆ Spezifikation der Eingaben, die in einem interaktiven System akzeptierbar sein sollen.
- ❖ **Compilerbau:**
  - ◆ Zur lexikalischen Analyse von Programmen, d.h zur Erkennung von Schlüsselwörtern und Bezeichnern
- ❖ **Softwaretechnik:**
  - ◆ Zur Beschreibung von Systemen, die durch die dynamischen Eigenschaften eines einzigen Objektes bestimmt werden.

## Endliche Automaten

- ❖ Ein endlicher Automat ist ein gutes Werkzeug zur Modellierung von Problemen aus der Wirklichkeit, wenn folgende Eigenschaften vorliegen.
  - ◆ **Diskrete Ein-/Ausgabe:**
    - ◆ Das System reagiert nur auf diskrete Eingabe-Werte und gibt nur diskrete Werte aus.
  - ◆ **Endliche Anzahl von Zuständen:**
    - ◆ Das System kann nur in einer endlichen Zahl von internen Konfigurationen sein, die wir Zustände nennen.
  - ◆ **Gedächtnislosigkeit:**
    - ◆ Der Zustand, in dem das System gerade ist, umfasst die gesamte Information über die Vergangenheit des Systems
    - ◆ Nur dieser Zustand entscheidet bei der nächsten Eingabe, was der nächste Zustand ist.

## Beispiel für einen endlichen Automaten

- ❖ Die **Steuerung eines Fahrstuhls** kann man mit einem endlichen Automaten modellieren.
  - ◆ **Diskrete Ein-/Ausgabe:**
    - ◆ Eingabe: Welche Knöpfe wurden in welchen Etagen gedrückt, aber noch nicht bedient?
    - ◆ Ausgabe: In welche Richtung (hoch, runter) und wie weit muss der Fahrstuhl als nächstes fahren?
  - ◆ **Endliche Anzahl von Zuständen:**
    - ◆ Der Fahrstuhl bedient nur eine endliche Anzahl von Etagen.
  - ◆ **Gedächtnislosigkeit:**
    - ◆ Die Steuerung braucht nicht zu wissen, welche Etagen der Fahrstuhl bis jetzt schon besucht hat.
    - ◆ Wichtig ist der derzeitige Zustand: Wo steht der Fahrstuhl?

## Ein altes Problem: Wolf, Kohl und Ziege

- ❖ Eine Hirte steht mit einem Wolf, einer Ziege und einem Kohlkopf am linken Ufer eines Flusses. Sie wollen den Fluss überqueren.
- ❖ Es gibt eine Fähre, die hin und her fährt. Die Fähre ist allerdings so klein, dass sie höchstens den Hirten und einen weiteren Passagier gleichzeitig transportieren kann.
- ❖ Es gibt folgende Probleme:
  - ◆ Allein gelassen, frisst der Wolf die Ziege
  - ◆ Allein gelassen, frisst die Ziege den Kohlkopf
- ❖ Wie können alle den Fluss erfolgreich überqueren, ohne dass irgend jemand gefressen wird?



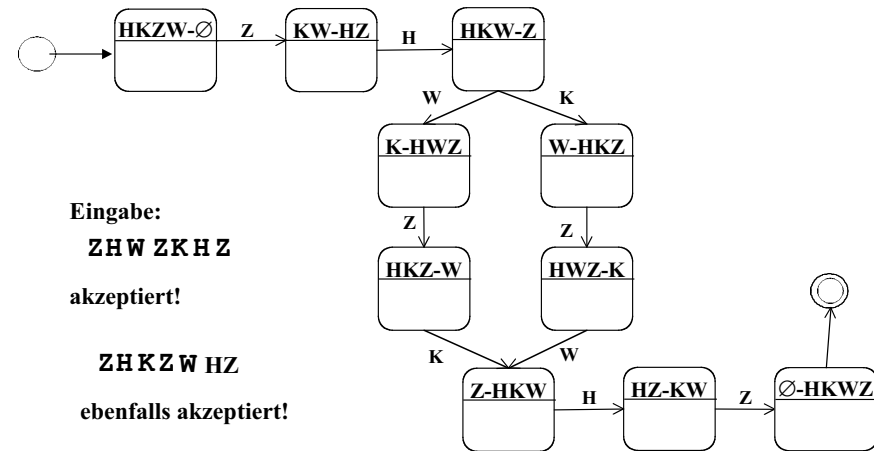
## Wie modellieren wir das Problem?

- ❖ Sollen wir **Anwendungsfälle** mit Ereignisfolgen probieren?
  - ◆ Akteure: Wolf, Hirte, Ziege, ...
  - ◆ Ereignisfolge:
    - ◆ Alle sind am linken Ufer
    - ◆ Der Hirte fährt mit der Ziege zum rechten Ufer
    - ◆ Dort lässt er die Ziege,
    - ◆ Dann fährt er allein zum linken Ufer
    - ◆ ...
- ❖ Sollen wir ein **Klassendiagramm** erstellen? Das Problem hat viele Objekte. Abbott's Analyse ergibt: Hirte, Wolf, Ziege, Kohlkopf, Fluss, Ufer, überqueren, fressen, transportieren, Fähre, Kapazität.
- ❖ Brauchen wir dafür ein **Sequenzdiagramm**?
- ❖ Oder können wir das ganze Problem allein mit einem **Zustandsdiagramm** modellieren?

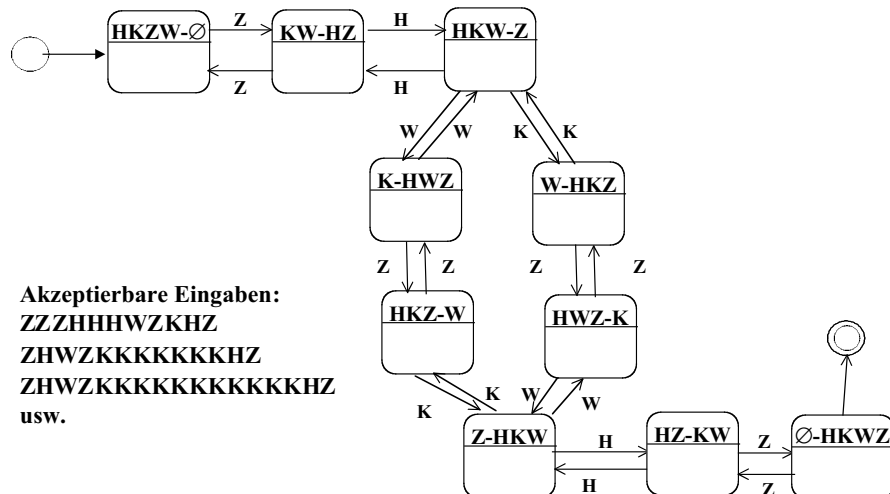
### Analyse des Hirte-Wolf-Ziege-Kohl-Problems

- Wir führen folgende Bezeichnungen ein: H ist der Hirte, W der Wolf, Z die Ziege und K der Kohl.  $\emptyset$  bezeichnet die leere Menge.
- Wir identifizieren eine Klasse **Steuerung** mit einem Attribut **Uferverteilung** vom Typ Zeichenkette mit Werten wie "HWZK- $\emptyset$ ", "WK-HZ", usw. Die Attributwerte modellieren die Besetzung der beiden Uferseiten folgendermaßen:
  - "HWZK- $\emptyset$ ": Alle Beteiligten sind am linken Ufer.
  - "WK-HZ": Wolf und Kohl am linken Ufer, Hirte und Ziege am rechten Ufer.
  - ....
  - "K-HWZ": Kohl am linken Ufer, Hirte, Wolf und Ziege am rechten Ufer.
  - " $\emptyset$ -HWZK": Alle Beteiligten sind am rechten Ufer.
- Insgesamt kann **Uferverteilung** höchstens  $2^4 = 16$  verschiedene Werte annehmen, wir haben also 16 Zustände (davon sind nicht alle erstrebenswert). Als Namen für die Zustände nehmen wir die Werte: HWZK- $\emptyset$  ist unser Anfangszustand, den Endzustand  $\emptyset$ -HWZK wollen wir erreichen.

### Modellierung des Problems mit UML-Zustandsdiagramm



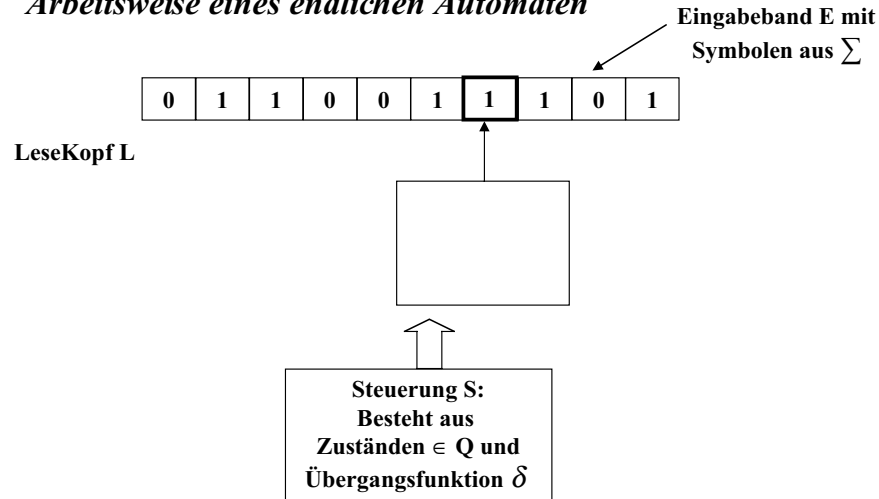
### Ebenfalls mögliche Modellierung des Problems



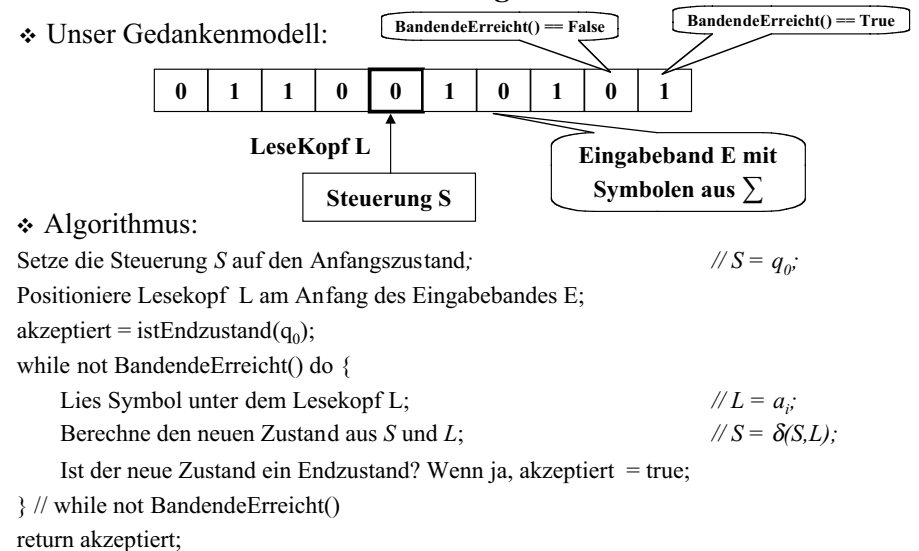
### Endlicher Automat

- Definition:** Ein endlicher Automat ist ein 5-Tupel  $A = (\Sigma, Q, \delta, q_0, F)$ , wobei gilt:
  - $\Sigma$  ist ein endlicher Zeichenvorrat (also ein Alphabet),
  - $Q$  ist eine nichtleere endliche Menge von Zuständen,
  - $\delta : Q \times \Sigma \rightarrow Q$  ist eine Übergangsfunktion (*engl. transition function*), wobei gilt:
    - $\delta(q, a)$  ist ein definierter Zustand für jeden Zustand  $q$  und Eingabe  $a \in \Sigma$ .
  - $q_0 \in Q$  ist ein Anfangszustand
  - $F$  ist eine Menge von Endzuständen

## Arbeitsweise eines endlichen Automaten



## Ein endlicher Automat ist ein Algorithmus



## Darstellungsmöglichkeiten für die Übergangsfunktion

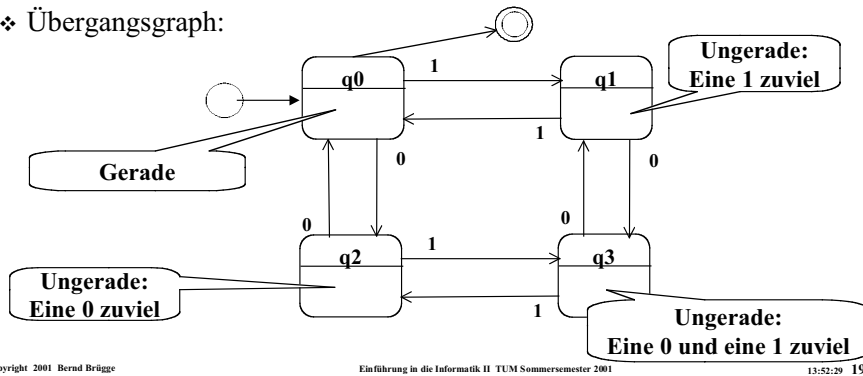
- ❖ Die Übergangsfunktion  $\delta$  eines endlichen Automaten lässt sich auf verschiedene Art und Weise darstellen:
- ♦ als Graph  $\Rightarrow$  Übergangsgraph
  - ♦ als Tabelle  $\Rightarrow$  Übergangstabelle
  - ♦ als Menge  $\Rightarrow$  Menge von Produktionen

## Übergangsfunktion $\delta$ als Graph

- ❖ Ein endlicher Automat kann als **gerichteter Graph G** dargestellt werden, wenn wir folgende Zuordnungen festlegen:
- ♦ die Menge der **Zustände Q** ist die Menge der **Knoten** des Graphen mit  $q_0$  als Anfangszustand
  - ♦ Zwei Knoten  $q_1$  und  $q_2$  im Graphen sind durch eine mit der Eingabe  $a$  beschriftete gerichtete **Kante** verbunden, wenn es einen **Übergang** vom Zustand  $q_1$  nach  $q_2$  gibt, d.h. wenn  $\delta(q_1, a) = q_2$  gilt.
- ❖ Den gerichteten Graphen nennen wir **Übergangsgraph**
- ♦ UML-Zustandsdiagramme sind also Übergangsgraphen.

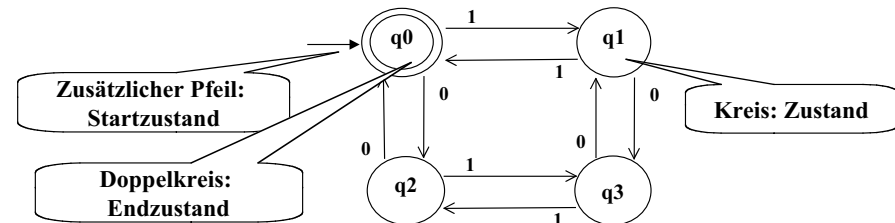
### Beispiel für einen Übergangsgraph

- Wir wollen einen Prüfer bauen, der eine vorgegebene Zeichenkette aus '0' und '1' nur dann akzeptiert, wenn die Kette eine gerade Anzahl von 0'en und eine gerade Anzahl 1'en hat.
- Beispiel: 110101 wird akzeptiert, 1101011 wird nicht akzeptiert
- Übergangsgraph:



### Alternative Notation für Übergangsgraphen

- In der Softwaretechnik benutzen wir UML-Zustandsdiagramme für die Notation von endlichen Automaten.
- In der Automatentheorie hat sich vorher schon eine einfachere Notation etabliert:
  - Zustände werden als Kreise gezeichnet
  - Der Anfangszustand hat einen kleinen gerichteten Pfeil
  - Der Endzustand besteht aus 2 konzentrischen Kreisen



- Bei endlichen Automaten ohne größere Komplexität werden wir diese Notation öfters benutzen.

### Übergangsfunktion $\delta$ als Tabelle

- Definition Übergangstabelle:** Tabellenförmige Darstellung der Übergangsfunktion eines endlichen Automaten.

Zustände	Eingabe aus $\Sigma$					
	a	b	...	j	...	z
$q_0$	$\delta(q_0, a)$	$\delta(q_0, b)$	...	$\delta(q_0, j)$	...	$\delta(q_0, z)$
$q_1$	$\delta(q_1, a)$	$\delta(q_1, b)$	...	$\delta(q_1, j)$	...	$\delta(q_1, z)$
...	...	...	...	...	...	...
$q_i$	$\delta(q_i, a)$	$\delta(q_i, b)$	...	$\delta(q_i, j)$	...	$\delta(q_i, z)$
...	...	...	...	...	...	...
$q_n$	$\delta(q_n, a)$	$\delta(q_n, b)$	...	$\delta(q_n, j)$	...	$\delta(q_n, z)$

### Übergangstabelle für den Gleichheitsprüfer

Zustand \ Eingabe	0	1
	$q_0$	$q_2$
$q_1$	$q_3$	$q_0$
$q_2$	$q_0$	$q_3$
$q_3$	$q_1$	$q_2$

**Erweiterung der Übergangsfunktion  $\delta$  von einzelnen Zeichen auf Zeichenketten:**  
 $\delta(q, aw) = \delta(\delta(q, a), w)$   
 $a \in \Sigma, w \in \Sigma^*$

- Beispiel: Eingabe 110101  
 $\delta(q_0, 1) = q_1 \Rightarrow \delta(q_1, 1) = q_0 \Rightarrow \delta(q_0, 0) = q_2 \Rightarrow \delta(q_2, 1) = q_3 \Rightarrow \delta(q_3, 0) = q_1 \Rightarrow \delta(q_1, 1) = q_0$
- Für diese Folge von Berechnungen schreiben wir auch verkürzt:  
 $\delta(q_0, 110101) \xrightarrow{*} q_0$

## Sprache eines Automaten

- ❖ Die von einem endlichen Automaten A **akzeptierte Sprache**  $L(A)$  ist definiert als

$$L(A) = \{ x \mid x \in \Sigma^*, q_0 x \xrightarrow{*} q_f \text{ mit } q_f \in F \}$$

- ❖ Alle Eingabeworte, die den Automaten A vom Anfangszustand in einen Endzustand überführen, sind also Teil der Sprache  $L(A)$ 
  - ♦ Wolf-Kohl-Ziege Automat A:  
 $ZHWZKKKKKKKHZ \in L(A)$ ,  $KZHW \notin L(A)$
  - ♦ Gleichheits-Checker P:
    - ♦  $110101 \in L(P)$ ,  $1101011 \notin L(P)$
- ❖ Im Übergangsgraphen entspricht ein  $x \in L(A)$  einem Weg vom Anfangszustand  $q_0$  zu einem Endzustand.

## Endliche Automaten mit Ausgabe

- ❖ Endliche Automaten, wie wir sie bis jetzt definiert haben, besitzen nur eine Eingabe, aber keine richtige Ausgabe:

Die Ausgabe ist ein binäres Signal:

"Die Eingabe ist akzeptiert" oder "Die Eingabe ist nicht akzeptiert"

- ❖ Es gibt Probleme, die mit endlichen Automaten modellierbar sind, bei denen die Ausgabe wichtiger ist als der erreichte Endzustand.
- ❖ Beispiel: MVV-Fahrkartenautomat.
  - ♦ Ich wähle einen Fahrschein aus, z.B. eine Tageskarte.
  - ♦ Ich werfe den korrekten Geldbetrag in den Automaten.
  - ♦ Jetzt erwarte ich, dass der Fahrschein ausgegeben wird.
  - ♦ Ob der Automat dabei in einem Endzustand ist, interessiert mich absolut nicht.
- ❖ Es gibt viele Arten von Automaten, die Ausgabe ermöglichen.
  - ♦ Für uns wichtige Varianten sind die Mealy-, Moore- und Harel-Automaten.

## Moore-Automat

- ❖ Ein **Moore-Automat** ist ein 6-Tupel  $M = (Q, \Sigma, T, \delta, \lambda, q_0)$ , wobei  $Q, \Sigma, \delta$  und  $q_0$  wie beim endlichen Automaten definiert sind.  $T$  ist das Ausgabealphabet, und  $\lambda$  ist eine Abbildung von  $Q$  nach  $T$ .
  - ♦ In einem **Zustand** wird also ein Zeichen  $t \in T$  ausgegeben.
- ❖ Jeder MVV-Kartenautomat enthält zwei Moore-Automaten, wobei das Eingabealphabet immer aus erlaubten Geldstücken besteht:
  - ♦ **Fahrscheinausgabe F**: Wenn der korrekte Geldbetrag eingegeben worden ist, wird ein Zustand erreicht, in dem der Fahrschein ausgegeben wird.
  - ♦ **Wechselgeldausgabe W**: Wenn mehr als der korrekte Geldbetrag eingegeben wurde, muss ein Zustand erreicht werden, in dem das zuviel eingezahlte Geld ausgegeben wird.

## Mealy-Automat

- ❖ Ein **Mealy-Automat** ist ein 6-Tupel  $M = (Q, \Sigma, T, \delta, \lambda, q_0)$  wobei  $Q, \Sigma, T, \delta$  und  $q_0$  wie beim Moore-Automaten definiert sind.  $\lambda$  ist allerdings eine Abbildung von  $Q \times \Sigma$  nach  $T$ .
  - ♦ Beim Übergang von einem Zustand zu einem anderen Zustand wird also bei Eingabe  $a \in \Sigma$  ein Zeichen  $t \in T$  ausgegeben.
- ❖ Jeder MVV-Kartenautomat enthält einen Mealy-Automaten:
  - ♦ Eingabealphabet sind wieder die erlaubten Geldstücke.
  - ♦ **Anzeige des fehlenden Betrages F**: Nach dem Einwurf jedes Geldstückes wird der noch fehlende Betrag auf dem Bildschirm angezeigt.

## Harel-Automat

- ❖ Die durch UML-Zustandsdiagramme beschreibbaren Automaten nennen wir **Harel-Automaten**.
- ❖ Ein Harel-Automat hat also
  - ◆ eine Zustandsmenge  $Q$ , die hierarchisch organisiert sein kann (durch Superzustände)
  - ◆ eine Eingabemenge  $\Sigma$  von Ereignissen
  - ◆ eine Menge von Prädikaten  $\Pi$  für Wächter
  - ◆ eine Ausgabemenge  $T$
  - ◆ einen Anfangszustand  $q_0$
  - ◆ Zwei Funktionen mit Ausgabe:
    - ◆ Übergangsfunktion  $\delta: Q \times \Sigma \times \Pi \rightarrow T \times Q$
    - ◆ Zustandsausgabe  $\lambda: Q \rightarrow T$

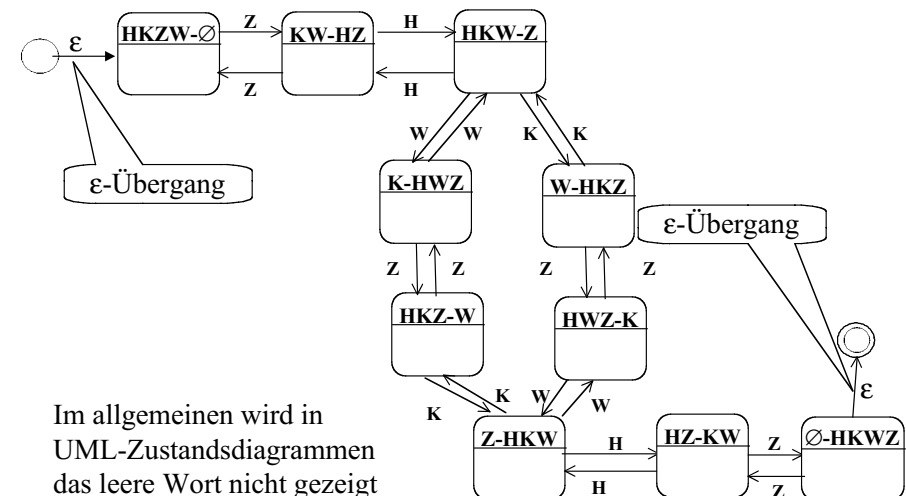
## Endliche Automaten vs Harel-Automaten

- ❖ Die **Eingabe** in einem endlichen Automaten entspricht den **Ereignissen** in einem Harel-Automaten. Die Ereignisse kommen dabei aus der Modellierung der Anwendungsdomäne.
- ❖ Die **Ausgabe** in einem endlichen Automaten ist äquivalent zu einer **Aktion** oder einer **Aktivität** im entsprechenden Harel-Automaten.
  - ◆ Ein UML-Zustandsdiagramm beschreibt einen Mealy-Automaten, wenn es keine Wächter und keine Aktivitäten enthält, sondern **nur Aktionen** der Form  $a/t$ .
  - ◆ Ein UML-Zustandsdiagramm beschreibt einen Moore-Automaten, wenn es keine Wächter und keine Aktionen enthält, sondern **nur Aktivitäten** der Form  $do/print(t)$

## Automaten mit $\epsilon$ -Übergängen

- ❖ Ein endlicher Automat kann auch Transitionen ohne spezifische Eingabe ausführen. In diesem Fall verallgemeinern wir die Übergangsfunktion  $\delta$  folgendermaßen:
  - ❖  $\delta: Q \times (\Sigma \cup \{\epsilon\}) \rightarrow Q$ , wobei gilt:
    - ◆  $\delta(q, a)$  ist ein definierter Zustand für jeden Zustand  $q$  und jede Eingabe  $a \in \Sigma \cup \{\epsilon\}$ .
  - ❖ **Erstaunlich:** Die Sprachen von Automaten mit  $\epsilon$ -Übergängen sind nicht mächtiger als die von endlichen Automaten.
    - ◆ **Satz:** Jeder Automat mit  $\epsilon$ -Übergängen kann in einen äquivalenten endlichen Automaten (ohne  $\epsilon$ -Übergänge) transformiert werden.
- Beweis  $\Rightarrow$  Hauptstudium (Automatentheorie)

## Beispiel für einen Automaten mit $\epsilon$ -Übergängen



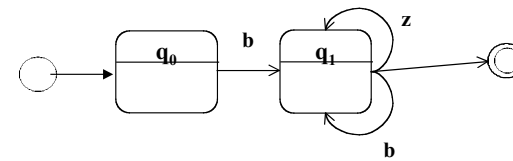
Im allgemeinen wird in UML-Zustandsdiagrammen das leere Wort nicht gezeigt

### Nichtdeterministischer endlicher Automat

- ❖ Bisher haben wir angenommen, dass  $\delta : Q \times \Sigma \rightarrow Q$  für die Berechnung von Zustandsübergängen **eine Funktion** ist.
  - ♦ Für jedes Paar Zustand/Eingabe gibt es nur einen neuen Zustand.
  - ♦ Solche Automaten nennen wir **deterministisch**.
- ❖ Im allgemeinen Fall kann man von einem gegebenen Zustand  $q$  bei Eingabe eines Zeichens mehr als einen Zustand bekommen.
  - ♦ Beispiel:  $\delta(q_1, a) = q_3$  **und**  $\delta(q_1, a) = q_4$
- ❖  $\delta$  ist dann keine Funktion, sondern eine **Relation**. Ein endlicher Automat, bei dem  $\delta$  eine Relation ist, heißt **nichtdeterministisch** oder auch **indeterministisch**.
- ❖ **Erstaunlich**: Die Sprachen, die von nichtdeterministischen endlichen Automaten akzeptiert werden, sind nicht mächtiger als die von deterministischen endlichen Automaten.
  - ♦ **Satz**: Jeder nichtdeterministische endliche Automat kann in einen äquivalenten deterministischen endlichen Automaten transformiert werden.

### Ein weiteres Beispiel: Erkennung von Zahlen

- ❖ Wir wollen einen endlichen Automaten konstruieren, der Bezeichner akzeptiert. Ein **Bezeichner** muss mit einem Buchstaben anfangen, gefolgt von beliebig vielen Buchstaben oder Ziffern.
- ❖ Endlicher Automat A:
  - ♦ Eingabealphabet  $\Sigma = \{ b, z \}$ 
    - ♦ wobei b ein Buchstabe ist und z eine Ziffer
  - ♦ Übergangsdiagramm:



### Grammatik für Bezeichner

- ❖ Für die eben definierten **Bezeichner** können wir auch eine Grammatik entwickeln, und zwar mit den Produktionen P:
  - { **Bezeichner** → 'b' **ZahlBuchstabe** | 'b'
  - ZahlBuchstabe** → 'b' **ZahlBuchstabe** | 'b'
  - ZahlBuchstabe** → 'z' **ZahlBuchstabe** | 'z'
  - }
- ❖ Diese Grammatik akzeptiert dieselbe Sprache wie der eben entwickelte Automat !
- ❖ Dies ist kein Zufall, sondern liegt an einem tieferen Zusammenhang zwischen Automaten und Grammatiken, den wir jetzt genauer herausarbeiten.

### Einschub: Reguläre Grammatik (Chomsky-3-Grammatik)

- ❖ Gegeben sei eine Chomsky-Grammatik  $G = (T, N, P, Z)$ . Seien  $A$  und  $B$  Nichtterminale aus  $N$  und  $a$  ein Terminal aus  $T$ .
- ❖ Wir definieren eine **Rechtslineare Produktion** als eine Produktion der Form  $A \rightarrow aB$
- ❖ Wir definieren eine **Linkslineare Produktion** als eine Produktion der Form  $A \rightarrow Ba$
- ❖ **Definition**: Eine Grammatik ist eine **reguläre Grammatik** oder auch **Chomsky-3-Grammatik**, wenn sie neben terminierenden Produktionen entweder nur links- oder nur rechtslineare Produktionen enthält.
- ❖ Beispiele:
  - ♦  $P = \{A \rightarrow aB, A \rightarrow a, B \rightarrow aB\} \Rightarrow G$  ist regulär
  - ♦  $P = \{A \rightarrow Ba, B \rightarrow a, B \rightarrow Ba\} \Rightarrow G$  ist regulär
  - ♦  $P = \{A \rightarrow Ba, B \rightarrow a, B \rightarrow aB\} \Rightarrow G$  ist nicht regulär
  - ♦  $P = \{A \rightarrow BA, B \rightarrow a, B \rightarrow aB\} \Rightarrow G$  ist nicht regulär

## Regulär oder nicht regulär?

$P = \{A \rightarrow aB, A \rightarrow a, B \rightarrow aB\}$

Grammatik ist regulär  
(Produktionen sind terminal  
oder rechtslinear)

$P = \{A \rightarrow Ba, B \rightarrow a, A \rightarrow \varepsilon\}$

Grammatik ist **nicht** regulär  
(hat eine sog.  $\varepsilon$ -Produktion, die  
weder rechts- noch linkslinear ist)

$P = \{A \rightarrow Ba, B \rightarrow a, B \rightarrow aB\}$

Grammatik ist **nicht** regulär  
(hat sowohl rechts- als auch  
linkslineare Produktionen)

$P = \{A \rightarrow BA, B \rightarrow a, B \rightarrow aB\}$

Grammatik ist **nicht** regulär  
(hat eine Produktion, die weder  
rechts- noch linkslinear ist)

## Automaten und Grammatiken

- ❖ Wir hatten  $L(A)$  als die Sprache definiert, die von einem Automaten  $A$  akzeptiert wird.
- ❖ Für eine gegebene Chomsky-Grammatik  $G = (\Sigma, N, P, Z)$  hatten wir  $L(G)$  als die Sprache definiert, die von  $G$  erzeugt wird (siehe Info I - Vorlesung 4).
- ❖ **Satz über Äquivalenz zwischen endlichen Automaten und regulären Grammatiken**
  - ♦ Eine Menge  $L \subseteq \Sigma^*$  ist genau dann die Sprache  $L = L(A)$  eines endlichen Automaten  $A$ , wenn es eine reguläre Grammatik  $G = (\Sigma, N, P, Z)$  gibt, so dass  $L = L(G)$ .
  - ♦ Die Äquivalenz ist beschränkt auf die nichtleeren Worte aus  $L$ : Eine reguläre Grammatik kann  $\varepsilon$  nicht im Sprachschatz haben, es gibt aber endliche Automaten, die  $\varepsilon$  erkennen (Welche?)

## Endliche Automaten und reguläre Grammatiken

- ❖ Zurück zu unserem Satz:
  - ♦ Immer wenn wir einen endlichen Automaten haben, der eine Sprache  $L(A)$  akzeptiert, können wir eine reguläre Grammatik konstruieren, die denselben Sprachschatz akzeptiert.
  - ♦ Dasselbe gilt auch umgekehrt.
- ❖ Wir führen den Beweis in beiden Richtungen.
  1. Gegeben eine reguläre Grammatik  $\Rightarrow$  Endlicher Automat
  2. Gegeben ein endlicher Automat  $\Rightarrow$  reguläre Grammatik

## Reguläre Grammatik $\Rightarrow$ Endlicher Automat

- ❖ Der Beweis ist konstruktiv, d.h. wir zeigen, wie für eine beliebige vorgegebene reguläre Grammatik der entsprechende Automat konstruiert werden kann.
- ❖ Sei eine reguläre Grammatik  $G = (T, N, P, Z)$  gegeben. Die Produktionen in  $P$  haben also alle die Form  $A \rightarrow aB, A \rightarrow a$  für rechtslineare Produktionen wird der Beweis analog geführt

### Reguläre Grammatik $\Rightarrow$ Endlicher Automat (3)

❖ Aus der Grammatik  $G = (T, N, P, Z)$  konstruieren wir den Automaten  $A = (\Sigma, Q, \delta, q_0, F)$  folgendermaßen:

$\Sigma = T$	Eingabealphabet $\Sigma$ : Alle Terminale $T$
$Q = N \cup \{q_f\}$	Zustandsmenge $Q$ : Alle Nichtterminale $N$ plus ein neuer Endzustand $q_f$
$q_0 = Z$	Anfangszustand $q_0$ : Das Axiom $Z$
$F = \{q_f\}$	Endzustandsmenge $F$ : Neu eingeführter Zustand $q_f$
$\delta(A, a) = B$ wenn $A \rightarrow aB \in P$	Für jede Produktion aus $P$ berechnen wir einen Wert für die Übergangsfunktion $\delta$ . Der Rest von $\delta$ ist undefiniert
$\delta(A, a) = q_f$ wenn $A \rightarrow a \in P$	

Das heißt: Für jede reguläre Grammatik können wir einen endlichen Automaten konstruieren, der dieselbe Sprache akzeptiert.

### Endlicher Automat $\Rightarrow$ Reguläre Grammatik

❖ Aus dem Automaten  $A = (\Sigma, Q, \delta, q_0, F)$  konstruieren wir die Grammatik  $G = (T, N, P, Z)$  folgendermaßen:

$T = \Sigma$	Terminale $T$ : Alle Elemente aus dem Eingabealphabet $\Sigma$
$N = Q$	Nichtterminale $N$ sind alle Zustände von $Q$
$P$ :	
♦ Für jedes $\delta(A, a) = B$ definieren wir eine Produktion $A \rightarrow aB$	
♦ Für jedes Eingabesymbol $a$ , das aus Zustand $A$ zu einem Endzustand $q_f \in F$ führt, d.h. $\delta(A, a) = q_f$ definieren wir eine Produktion $A \rightarrow a$	
$Z = q_0$	Als Axiom $Z$ nehmen wir den Anfangszustand $q_0$

Das heißt: Für jeden endlichen Automaten können wir eine reguläre Grammatik konstruieren, die dieselbe Sprache (bis auf  $\epsilon$ ) akzeptiert.

**q.e.d.**

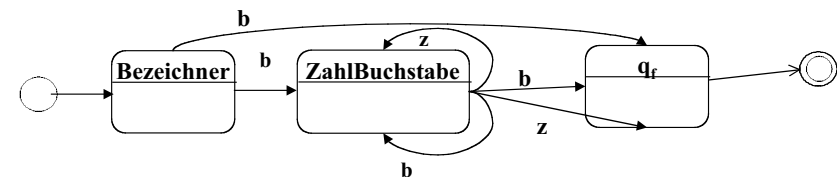
### Beispiel: Konvertierung einer Grammatik in endlichen Automaten

❖  $G = (T, N, P, Z)$ , wobei gilt:

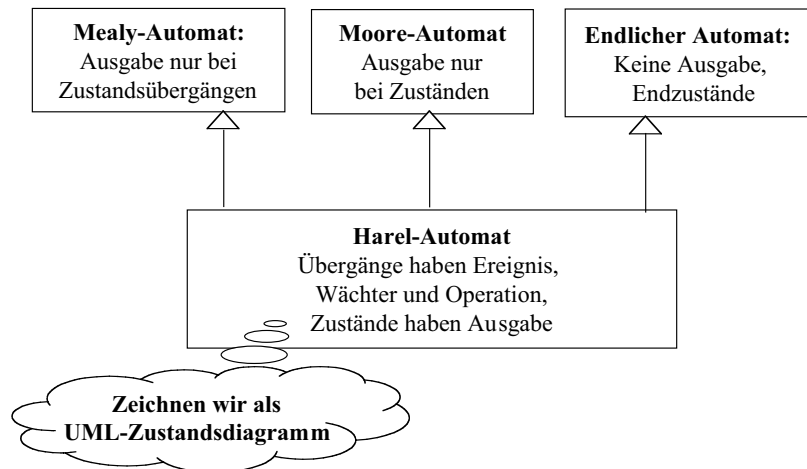
- $T = \{ 'b', 'z' \}$
- $N = \{ \text{Bezeichner, ZahlBuchstabe} \}$
- $P = \{$ 
  - Bezeichner  $\rightarrow$  'b' ZahlBuchstabe | 'b'
  - ZahlBuchstabe  $\rightarrow$  'b' ZahlBuchstabe | 'b'
  - ZahlBuchstabe  $\rightarrow$  'z' ZahlBuchstabe | 'z'
- $\}$
- $Z = \text{Bezeichner}$

### Konstruktion des Automaten

$\Sigma = T = \{b, z\}$
$Q = N \cup q_f = \{ \text{Bezeichner, ZahlBuchstabe, } q_f \}$
$q_0 = Z = \text{Bezeichner}$
$F = q_f$
$\delta(A, a) = B$ wenn $A \rightarrow aB \in P$
$\delta(\text{Bezeichner}, b) = \{ \text{ZahlBuchstabe, } q_f \}$ // Bezeichner $\rightarrow$ 'b' ZahlBuchstabe
$\delta(\text{ZahlBuchstabe}, b) = \{ \text{ZahlBuchstabe, } q_f \}$ // ZahlBuchstabe $\rightarrow$ 'b' ZahlBuchstabe
$\delta(\text{ZahlBuchstabe}, z) = \{ \text{ZahlBuchstabe, } q_f \}$ // ZahlBuchstabe $\rightarrow$ 'z' ZahlBuchstabe



## Zusammenfassung (1)



## Zusammenfassung (2)

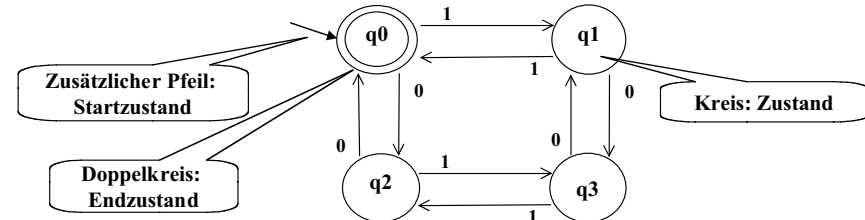
- ❖ Endlicher Automat zur Modellierung von *dynamischen* Aspekten in Systemen
- ❖ Verschiedene Darstellungen für die Übergangsfunktion:
  - ♦ *Übergangsgraph, Übergangstabelle, Produktionsmenge*
- ❖ Wichtige Automaten mit Ausgabe: *Mealy, Moore, und Harel*
- ❖ *UML-Zustandsdiagramme* sind *Übergangsgraphen* für Harel-Automaten.
- ❖ *Endliche Automaten* und *reguläre Grammatiken* sind verschiedene Interpretationen desselben Sachverhaltes
  - ♦ Viele Einsichten über Automaten erhalten wir, weil wir zwischen diesen Interpretationen hin und her wechseln können.
  - ♦ Da ein Übergangsgraph ein Graph ist, können wir auch Kenntnisse aus der Graphentheorie zum Verständnis der Eigenschaften von endlichen Automaten verwenden.

## Ziele dieses Vorlesungsabschnitts

- ❖ Sie können einen nichtdeterministischen endlichen Automaten in einen endlichen Automaten transformieren.
- ❖ Sie können mit dem Pumping-Lemma zeigen, dass eine gegebene Sprache nicht regulär ist.
- ❖ Sie verstehen, dass es Sprachen gibt, die man mit endlichen Automaten nicht erkennen kann.

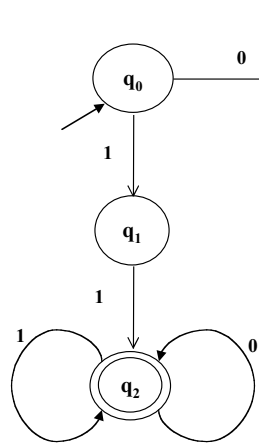
## Wiederholung: Alternative Notation für Übergangsgraphen

- ❖ In der Softwaretechnik benutzen wir UML-Zustandsdiagramme für die Notation von endlichen Automaten.
- ❖ In der Automatentheorie hatte sich schon vorher eine einfachere Notation etabliert:
  - ♦ Zustände werden als Kreise gezeichnet
  - ♦ Der Anfangszustand hat einen kleinen gerichteten Pfeil
  - ♦ Der Endzustand besteht aus 2 konzentrischen Kreisen



- ❖ Bei endlichen Automaten ohne größere Komplexität werden wir diese Notation öfters benutzen.

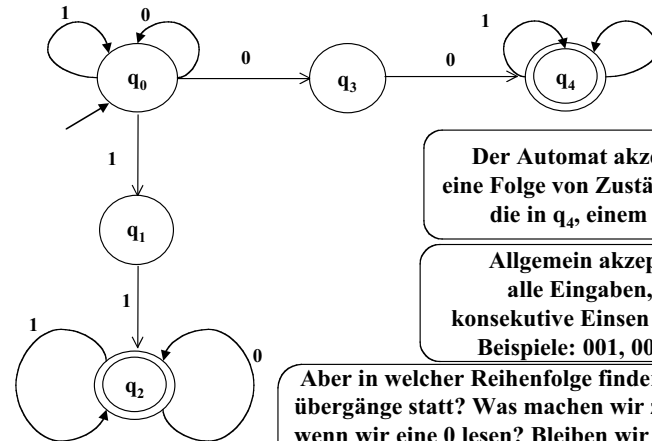
### Übung: Welche Sprache akzeptiert dieser Automat?



Der Automat akzeptiert 0011, denn die Folge von Zuständen  $q_0, q_3, q_4, q_4$  endet in  $q_4$ , einem Endzustand

Allgemein akzeptiert der Automat alle Eingaben, die entweder mit 00 oder mit 11 anfangen.  
Beispiele: 001, 0001, 11, 110, 1101

### Und welche Sprache akzeptiert dieser Automat?



Eingabebeispiel: 01001

Der Automat akzeptiert 01001: Es gibt eine Folge von Zuständen  $q_0, q_0, q_0, q_3, q_4, q_4$  die in  $q_4$ , einem Endzustand, endet

Allgemein akzeptiert der Automat alle Eingaben, die mindestens 2 konsekutive Einsen oder Nullen enthalten.  
Beispiele: 001, 0001, 11, 110, 1101, ...

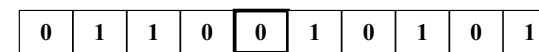
Aber in welcher Reihenfolge finden die Zustandsübergänge statt? Was machen wir z.B in Zustand  $q_0$ , wenn wir eine 0 lesen? Bleiben wir in  $q_0$  oder gehen wir nach  $q_3$ ? Wenn wir immer in  $q_0$  blieben, würde 01001 nicht akzeptiert werden. Wann tun wir was?

### Nicht-Deterministische endliche Automaten

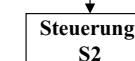
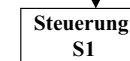
- ❖ Endliche Automaten mit Zuständen, die für dieselbe Eingabe mehr als einen Übergang erlauben, hatten wir nicht-deterministisch genannt.
  - ❖ Das Konzept der nicht-deterministischen endlichen Automaten spielt eine große Rolle in mehreren Gebieten der Informatik, insbesondere im Übersetzerbau und beim Beweisen von Theoremen.
  - ❖ Wir hatten schon erwähnt, dass nicht-deterministische endliche Automaten keine mächtigeren Sprachen akzeptieren, als deterministische endliche Automaten.
    - ♦ Das werden wir nun im folgenden beweisen.
    - ♦ Eine solche Äquivalenz gilt nicht für alle Arten von Automaten (unendliche Automaten, Kellerautomaten).
- ⇒ Hauptstudium

### Gedankenmodell: Ein nicht-deterministischer endlicher Automat hat mehrere Steuerungen

Eingabeband E mit Symbolen aus  $\Sigma$



LeseKopf L



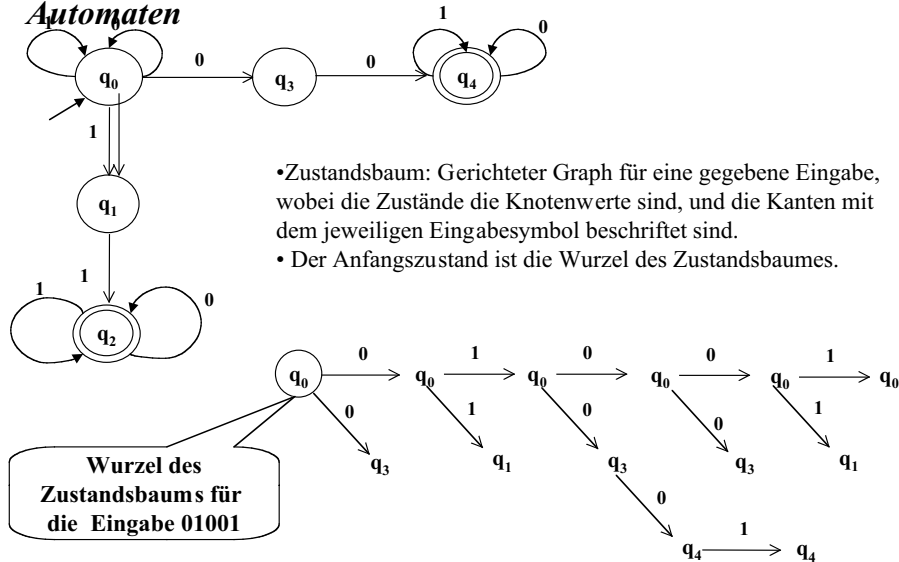
$P = \{S1, S2\}$

#### ❖ Algorithmus:

```

Setze die Steuerung S auf den Anfangszustand; // S = q0;
Akzeptiert = istEndzustand(S);
While not BandendeErreicht() do {
  Lies Symbol unter dem Lesekopf // L = a_i;
  Berechne die Menge P aller Zustände aus S und L; // P = {S1, S2, ...Sn} = delta(S,L);
  // P enthält eine Menge von Steuerungen S_i. Alle Steuerungen zeigen auf dasselbe Eingabesymbol,
  // können aber in verschiedenen Zuständen sein
  // In einer Java-Implementation würden die Steuerungen als Threads implementiert
  Ist eine der Steuerungen in P in einem Endzustand? Wenn ja, Akzeptiert = true;
} // while not Akzeptiert
    
```

## Zustandsbaum eines nicht-deterministischen endlichen Automaten

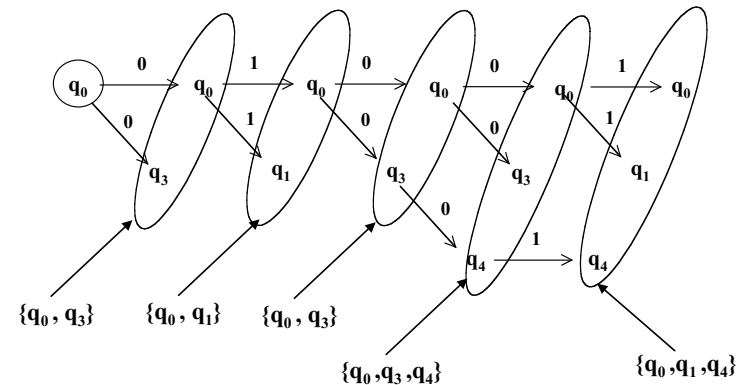


Copyright 2001 Bernd Brügge

Einführung in die Informatik II TUM Sommersemester 2001

13:52:29 53

## Ein nicht-deterministischer endlicher Automat kann also immer in mehr als einem Zustand sein



- ❖ Wir verallgemeinern die Übergangsfunktion  $\delta : Q \times \Sigma \rightarrow Q$  deshalb zu  $\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$ 
  - ♦ wobei  $\mathcal{P}(Q)$  die Potenzmenge von  $Q$  bezeichnet.

Copyright 2001 Bernd Brügge

Einführung in die Informatik II TUM Sommersemester 2001

13:52:29 55

## Nicht-deterministischer Endlicher Automat: Definition

- ❖ **Definition:** Ein nichtdeterministischer endlicher Automat ist ein 5-Tupel  $A = (\Sigma, Q, \delta_{nea}, q_0, F)$ , wobei gilt:
  - ♦  $\Sigma, Q, q_0$  und  $F$  (Eingabe, Zustände, Anfangszustand und Endzustände) sind wie beim deterministischen Automaten definiert.
  - ♦ Die Übergangsfunktion  $\delta_{nea} : Q \times \Sigma \rightarrow \mathcal{P}(Q)$  bildet dabei Tupel (Zustand, Eingabe) auf eine Teilmenge der Potenzmenge von  $Q$  ab.
  - ♦  $\delta_{nea}$  ist eine Erweiterung von  $\delta$  wie folgt:
    - ♦  $\delta_{nea}(q, \varepsilon) = \{q\}$
    - ♦  $\delta_{nea}(q, wa) = \{p \mid \exists r \in \delta_{nea}(q, w) \Rightarrow \delta(r, a) = p\}$ 
      - d.h. ausgehend von einem Zustand  $q$  lesen wir die Zeichenkette  $w$ . Dann gibt es einen Zustand  $r$ , von dem wir aus mit dem Eingabezeichen  $a$  nach  $p$  gehen können.

Copyright 2001 Bernd Brügge

Einführung in die Informatik II TUM Sommersemester 2001

13:52:29 56

## Die Äquivalenz von nicht-deterministischen und deterministischen endlichen Automaten

- ❖ **Satz:** Sei  $L$  eine von einem nicht-deterministischen endlichen Automaten akzeptierte Sprache. Dann existiert ein deterministischer endlicher Automat, der  $L$  auch akzeptiert.
- ❖ **Beweisidee:** Der Beweis baut darauf auf, dass wir für einen gegebenen nicht-deterministischen endlichen Automaten NEA einen deterministischen endlichen Automaten DEA bauen können, der NEA simuliert.
  - ♦ Jeder Zustand im DEA wird dabei rein syntaktisch aus den Zustandsteilmengen (die "Ellipsen" auf Folie 10) von NEA konstruiert.

Copyright 2001 Bernd Brügge

Einführung in die Informatik II TUM Sommersemester 2001

13:52:29 57

## Äquivalenz-Beweis

- Wir beginnen mit einem nicht-deterministischen endlichen Automaten  $NEA = (\Sigma, Q, \delta_{nea}, q_0, F)$ . Aus diesem konstruieren wir einen deterministischen endlichen Automaten  $DEA = (\Sigma', Q', \delta', q_0', F')$  wie folgt:
  - $\Sigma' = \Sigma$
  - $Q' = \mathcal{P}(Q)$ . Die Zustände in DEA sind also Teilmengen der Potenzmenge der Zustände von NEA. Elemente von  $Q'$  bezeichnen wir als **DEA-Zustand**  $[q_1, q_2, \dots, q_n]$ .
    - Ein Zustand ist nur dann ein DEA-Zustand  $[q_1, q_2, \dots, q_n]$ , wenn der NEA in einem der **korrespondierenden Zustände**  $q_1, q_2, \dots$  oder  $q_n$  ist.
  - $q_0' = [q_0]$
  - $F'$  ist die Menge aller DEA-Zustände  $[q_1, \dots, q_i, \dots, q_n]$  in  $Q'$ , bei denen mindestens ein Element  $q_i$  aus  $F$  ist.
- Jetzt müssen wir "nur noch" die Übergangsfunktion des deterministischen Automaten konstruieren.

## Äquivalenz-Beweis (2)

- Wir definieren die Übergangsfunktion des deterministischen Automaten DEA wie folgt:
 
$$\delta'([q_1, q_2, \dots, q_n], a) = [p_1 p_2, \dots, p_n]$$
 genau dann wenn
 
$$\delta_{nea}(\{q_1, q_2, \dots, q_n\}, a) = \{p_1 p_2, \dots, p_n\}$$
- $\delta'$  für DEA-Zustand  $[q_1, q_2, \dots, q_n]$  bei Eingabe  $a$  wird so berechnet:
  - Wir wenden die  $\delta_{nea}$  vom NEA auf jeden der NEA-Zustände  $q_1, q_2, \dots, q_n$  für  $a$  an, was mehrere Zustandsmengen ergibt.
  - Diese Zustandsmengen vereinigen wir dann zur Menge  $\{p_1 p_2, \dots, p_n\}$ , woraus wir den DEA-Zustand  $[p_1 p_2, \dots, p_n]$  erhalten.
- Illustratives Beispiel:**
  - NEA:**  $\delta_{nea}(q_1, a) = \{p_1\}$ ,  $\delta_{nea}(q_2, a) = \{p_2, p_7\}$ ,  $\delta_{nea}(q_3, a) = \{p_2, p_3, p_{12}\}$
  - $\{p_1 p_2, p_3, p_7, p_{12}\} \Rightarrow [p_1 p_2, p_3, p_7, p_{12}]$
  - DEA:**  $\delta'([q_1, q_2, q_3], a) = [p_1 p_2, p_3, p_7, p_{12}]$

## Äquivalenz-Beweis (3)

- Jetzt sind wir soweit, den Beweis der Äquivalenz zu machen. Wir benutzen dabei vollständige Induktion über die Länge der Eingabe-Zeichenkette  $x$ . D.h. für alle  $x$  beliebiger Länge gilt:
 
$$\delta'(q_0', x) = [p_1 p_2, \dots, p_n] \text{ genau dann, wenn } \delta_{nea}(q_0, x) = \{p_1 p_2, \dots, p_n\}$$
- Induktionsanfang:**  $|x| = 0$ . Als Eingabe ist also nur das leere Wort  $\epsilon$  möglich. Beide Automaten akzeptieren  $\epsilon$  genau dann, wenn  $q_0$  ein Endzustand ist ( $\delta'(q_0', \epsilon) = \delta_{nea}(q_0, \epsilon)$ ) aufgrund der Konstruktion von DEA (siehe Folie 13), wo  $q_0' = [q_0]$ . Also  $L(DEA) = L(NEA)$ .
- Induktionsschritt:** Annahme: Für alle Eingaben der Länge  $|x| \leq m$  akzeptieren die Automaten dieselbe Sprache  $L(DEA) = L(NEA)$ .
 
$$\delta'(q_0', x) = [p_1 p_2, \dots, p_n] \text{ genau dann, wenn } \delta_{nea}(q_0, x) = \{p_1 p_2, \dots, p_n\}$$
- Nun betrachten wir Eingaben der Länge  $m+1$ , d.h. alle  $xa$ , wobei  $a$  ein beliebiges Element aus  $\Sigma$  ist. Wir müssen jetzt zeigen, dass genau dann, wenn  $xa$  vom nicht-deterministischen endlichen Automaten erkannt wird, es auch vom deterministischen endlichen Automaten erkannt wird.

## Äquivalenz-Beweis (4)

- Aufgrund der Induktionshypothese gilt für  $|x| \leq m$ :
 
$$\delta'(q_0', x) = [p_1 p_2, \dots, p_n] \text{ genau dann, wenn } \delta_{nea}(q_0, x) = \{p_1 p_2, \dots, p_n\}$$
  - d.h. die Eingabe von  $x$  wird von DEA genau dann erkannt, wenn sie auch von NEA erkannt wird.
- Zur Erinnerung:** Die Übergangsfunktion  $\delta'$  von DEA ist:
 
$$\delta'([p_1, p_2, \dots, p_n], a) = [r_1, r_2, \dots, r_n]$$

$$\delta_{nea}(\{p_1, p_2, \dots, p_n\}, a) = \{r_1, r_2, \dots, r_n\}$$
- Außerdem gilt:**  $\delta'(q_0', xa) = \delta'(\delta'(q_0', x), a)$ 
  - Aus diesen 3 Gleichungen erhalten wir:
 
$$\delta'(q_0', xa) = [r_1, r_2, \dots, r_n] \text{ genau dann, wenn } \delta_{nea}(q_0, xa) = \{r_1, r_2, \dots, r_n\}$$
    - d.h. die Eingabe von  $xa$  führt auf einen DEA-Zustand genau dann, wenn die Eingabe von  $xa$  einen der korrespondierenden Zustände im NEA ergibt.

### Äquivalenz-Beweis (4, ohne Kreise)

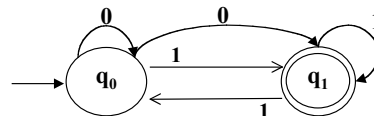
- ❖ Aufgrund der Induktionshypothese gilt für  $|x| \leq m$  :  
 $\delta'(q_0', x) = [p_1 p_2 \dots, p_n]$  genau dann, wenn  $\delta_{nea}(q_0, x) = \{p_1 p_2 \dots, p_n\}$ 
  - ♦ d.h. die Eingabe von  $x$  wird von DEA genau dann erkannt, wenn sie auch von NEA erkannt wird.
- ❖ **Zur Erinnerung:** Die Übergangsfunktion  $\delta'$  von DA ist:  
 $\delta'([p_1, p_2, \dots, p_n], a) = [r_1, r_2, \dots, r_n]$  genau dann, wenn  
 $\delta_{nea}(\{p_1, p_2, \dots, p_n\}, a) = \{r_1, r_2, \dots, r_n\}$
- ❖ **Außerdem gilt:**  $\delta'(q_0', xa) = \delta'(\delta'(q_0', x), a)$
- ❖ Das heißt, es gilt :  
 $\delta'(q_0', xa) = [r_1, r_2, \dots, r_n]$  genau dann, wenn  
 $\delta_{nea}(q_0, xa) = \{r_1, r_2, \dots, r_n\}$ 
  - ♦ d.h. die Eingabe von  $xa$  führt auf einen DEA-Zustand genau dann, wenn die Eingabe von  $xa$  im NEA auf einen korrespondierenden Zustand führt.

### Äquivalenz-Beweis (5)

- ❖ Um den Beweis zu beenden, müssen wir nur noch sicherstellen, dass  $\delta'(q_0', x)$  genau dann in  $F'$  ist, wenn beim nicht-deterministischen endlichen Automaten die Berechnung  $\delta(q_0, x)$  einen Zustand ergibt, der in  $F$  ist.
- ❖ Damit gilt also:  $L(\text{NEA}) = L(\text{DEA})$
- ❖ q.e.d
- ❖ **Bemerkung:** Für den vollständigen Beweis der Äquivalenz von NEA und DEA muss strenggenommen noch gezeigt werden, wie aus einem NEA ein DEA konstruiert wird:
  - ♦ nichts zu tun, da jeder DEA bereits ein NEA ist

### Beispiel: Konstruktion eines DEA aus einem NEA

- ❖ Gegeben sei ein NEA =  $(\Sigma, Q, \delta_{nea}, q_0, F)$  mit
  - ♦ Eingabealphabet  $\Sigma = \{0, 1\}$
  - ♦ Zustandsmenge  $Q = \{q_0, q_1\}$
  - ♦ Übergangsfunktion  $\delta_{nea}$ :  
 $\delta_{nea}(q_0, 0) = \{q_0, q_1\}$   
 $\delta_{nea}(q_0, 1) = \{q_1\}$   
 $\delta_{nea}(q_1, 0) = \emptyset$   
 $\delta_{nea}(q_1, 1) = \{q_0, q_1\}$
  - ♦ Anfangszustand  $q_0$
  - ♦ Endzustandsmenge  $F = \{q_1\}$



### Beispiel: Konstruktion eines DEA aus einem NEA (2)

- ❖ Wir konstruieren nun den DEA =  $(\Sigma', Q', \delta', [q_0], F')$  wie folgt:

- ♦ **Eingabealphabet:**  $\Sigma' = \Sigma = \{0, 1\}$

- ♦ **Zustandsmenge:**  $Q' = \{[q_0], [q_1], [q_0, q_1]\}$

denn die Potenzmenge von  $Q = \{q_0, q_1\}$  ist  $\mathcal{P}(Q) = \{\emptyset, \{q_0\}, \{q_1\}, \{q_0, q_1\}\}$

- ♦ **Übergangsfunktion  $\delta'$ :**

$\delta'([q_0], 0) = [q_0, q_1]$  denn  $\delta_{nea}(q_0, 0) = \{q_0, q_1\}$

$\delta'([q_0], 1) = [q_1]$  denn  $\delta_{nea}(q_0, 1) = \{q_1\}$

$\delta'([q_1], 0) = \emptyset$  denn  $\delta_{nea}(q_1, 0) = \emptyset$

$\delta'([q_1], 1) = [q_0, q_1]$  denn  $\delta_{nea}(q_1, 1) = \{q_0, q_1\}$

$\delta'([q_0, q_1], 0) = [q_0, q_1]$  denn  $\delta_{nea}(\{q_0, q_1\}, 0) = \delta_{nea}(q_0, 0) \cup \delta_{nea}(q_1, 0) = \{q_0, q_1\} \cup \emptyset = \{q_0, q_1\}$

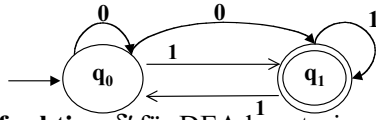
$\delta'([q_0, q_1], 1) = [q_0, q_1]$  denn  $\delta_{nea}(\{q_0, q_1\}, 1) = \delta_{nea}(q_0, 1) \cup \delta_{nea}(q_1, 1) = \{q_1\} \cup \{q_0, q_1\} = \{q_0, q_1\}$

- ♦ **Anfangszustand:**  $[q_0]$

- ♦ **Endzustände:**  $F' = \{[q_1], [q_0, q_1]\}$

### Einfachere Konstruktion der Übergangsfunktion $\delta'$

❖ Wenn der NEA als Übergangsgraph vorliegt, lässt sich der korrespondierende DEA relativ leicht konstruieren.



❖ Die Übergangsfunktion  $\delta'$  für DEA konstruieren wir, indem wir die Spalte **Zustände** der Übergangstabelle *schrittweise* erstellen.

❖ Wir fangen mit dem Anfangszustand  $[q_0]$  an.

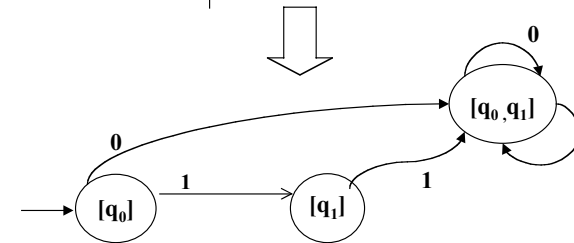
		Eingabe	
		0	1
Zustände	$[q_0]$	$[q_0, q_1]$	$[q_1]$
	$[q_0, q_1]$	$[q_0, q_1]$	$[q_0, q_1]$
	$[q_1]$	$\emptyset$	$[q_0, q_1]$

Annotations:  
 2:  $[q_0, q_1]$  wird hinzugefügt  
 4:  $[q_1]$  wird hinzugefügt  
 3:  $[q_1]$  ist Resultat einer Berechnung  
 1:  $[q_0, q_1]$  ist Resultat einer Berechnung

❖ Neue Zustände werden also nur dann in die Spalte **Zustände** hinzugefügt, wenn sie das Resultat einer Zustandsberechnung sind.

### Aus der Übergangstabelle erzeugen wir dann den Übergangsgraph von DEA

		Eingabe aus $\Sigma'$	
		0	1
Zustände	$[q_0]$	$[q_0, q_1]$	$[q_1]$
	$[q_0, q_1]$	$[q_0, q_1]$	$[q_0, q_1]$
	$[q_1]$	$\emptyset$	$[q_0, q_1]$



### Sprachen erzeugen interessante Probleme

❖ Wenn wir Sprachen durch eine Grammatik oder durch einen Automaten spezifizieren, gibt es viele interessante Fragen.

- Sind 2 Sprachen, die wir mit verschiedenen Grammatiken spezifiziert haben, äquivalent?
- Sind 2 Sprachen, die von verschiedenen Automaten akzeptiert werden, äquivalent?
- Ist die Sprache, die von einem endlichen Automaten akzeptiert wird, endlich oder unendlich?
- Ist die Sprache regulär oder nicht?

❖ Wir nennen eine Sprache **regulär**, wenn sie von einem endlichen Automaten erkannt wird, oder von einer regulären Grammatik erzeugt wird.

- Im Rest dieses Vorlesungsabschnitts beschäftigen wir uns mit einem wichtigen Werkzeug, mit dem wir zeigen können, dass bestimmte Sprachen nicht regulär sind.

### Pumping-Lemma: Vorüberlegungen

❖ Gegeben sei ein endlicher Automat  $A = (\Sigma, Q, \delta, q_0, F)$  mit  $n$  Zuständen.

❖ Als Eingabe nehmen wir eine Zeichenkette  $a_1 a_2 \dots a_m$ , wobei  $m \geq n$  gilt. Die Zeichenkette hat also mindestens die Länge  $n$ .

❖ Die Übergangsfunktion  $\delta$  ist definiert als:  $\delta(q_0, a_1 a_2 \dots a_m) = q_i$  für  $i = 1, 2, \dots, m$ , d.h. sie ist so konstruiert, dass folgendes gilt:

$$\begin{aligned} \delta(q_0, a_1) &= q_1 \\ \delta(q_0, a_1 a_2) &= q_2 \\ \delta(q_0, a_1 a_2 a_3) &= q_3 \end{aligned}$$

...

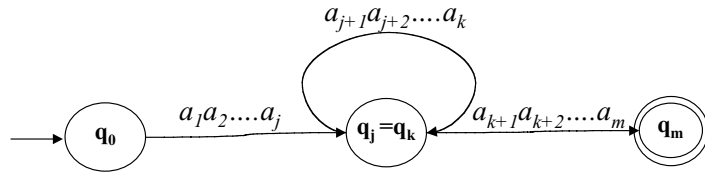
$$\delta(q_0, a_1 a_2 \dots a_m) = q_m$$

❖ Da der Automat nur  $n$  Zustände hat und  $m \geq n$  ist, ist es nicht möglich, dass die Zustände  $q_0, q_1, q_2, q_3, \dots, q_m$  in der Übergangsfunktion alle verschieden sind!

❖ Diese Erkenntnis ist das Kernstück des **Pumping-Lemmas**.

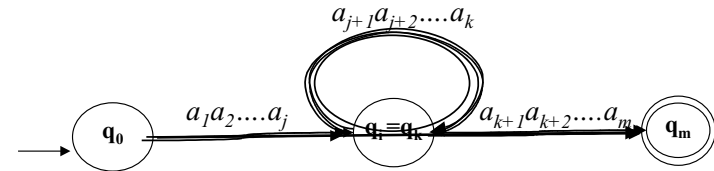
### Pumping-Lemma: Vorüberlegungen (2)

- ❖ Schauen wir uns unsere Übergangsfunktion einmal im Übergangsgraphen für die Eingabe  $a_1a_2\dots a_m$  der Länge  $m$  an:



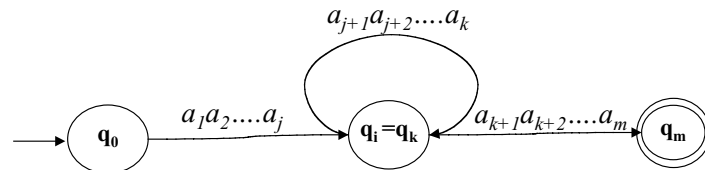
- ❖ Da die Eingabe  $m \geq n$  Zeichen enthält und wir daher mindestens  $n+1$  Zustände  $q_0, \dots, q_m$  durchlaufen, wir aber nur  $n$  verschiedene Zustände haben, muss es zwei Indizes  $j$  und  $k$  geben,  $0 \leq j < k \leq m$ , für die gilt:  
 $q_j = q_k$
- ❖ Das Teilwort  $a_{j+1}a_{j+2}\dots a_k$  hat dabei mindestens die Länge 1.
- ❖ Im Weiteren gehen wir davon aus, dass wir die kleinsten Indizes  $j, k$  finden (d.h. die Länge des Teilworts  $a_1a_2\dots a_k$  ist kleiner gleich  $n$ ).

### Pumping-Lemma: Vorüberlegungen (3)



- ❖ Da  $q_m$  ein Endzustand ist, ist  $a_1a_2\dots a_m$  in  $L(A)$ , d.h. in der von  $A$  akzeptierten Sprache.
- ❖ Es ist aber auch  $a_1a_2\dots a_j a_{k+1}a_{k+2}\dots a_m$  in  $L(A)$ , also die Eingabe, die die Schleife  $a_{j+1}a_{j+2}\dots a_k$  beim Zustand  $q_i = q_k$  vermeidet.
- ❖ Und es ist  $a_1a_2\dots a_j a_{j+1}a_{j+2}\dots a_k a_{j+1}a_{j+2}\dots a_k a_{k+1}a_{k+2}\dots a_m$  in  $L(A)$ , d.h. die Eingabe, die zweimal über die Schleife beim Zustand  $q_i = q_k$  geht.
- ❖ Dies können wir natürlich verallgemeinern.

### Pumping-Lemma: Vorüberlegungen (4)



- ❖ Es gilt für ein beliebiges  $i \geq 0$ :
  - ♦ Wenn  $q_m$  ein Endzustand ist und  $a_1a_2\dots a_m$  in  $L(A)$ , dann ist auch  $a_1a_2\dots a_j (a_{j+1}a_{j+2}\dots a_k)^i a_{k+1}a_{k+2}\dots a_m$  in  $L(A)$ , d.h. eine Eingabe, die  $i$ -mal über die Schleife bei  $q_i = q_k$  geht.
- ❖ Wir haben also folgendes:
  - ♦ Gegeben sei eine lange Eingabezeichenkette  $E$ , die von einem endlichen Automaten  $A$  akzeptiert wird
  - ♦ Dann können wir immer eine Teilzeichenkette in  $E$  finden, die wir "aufpumpen" können.
  - ♦ Die daraus resultierende Zeichenkette wird auch von  $A$  akzeptiert.

### Pumping-Lemma für reguläre Sprachen

- ❖ **Lemma:** Sei  $L(A)$  eine von einem endlichen Automaten mit  $n$  Zuständen akzeptierte Sprache, also eine reguläre Sprache.
  - ♦ Sei  $z$  ein beliebiges Wort mit der Länge  $|z| \geq n$ . Dann ist es möglich,  $z$  als Konkatination von Teilwörtern  $u, v$  und  $w$  zu schreiben:  $z = uvw$ , wobei  $|uv| \leq n$  und  $|v| \geq 1$ .
  - ♦ Dann ist auch  $uv^i w$  in  $L(A)$  für ein beliebiges  $i \geq 0$ .

## Pumping-Lemma für reguläre Sprachen: Beweis

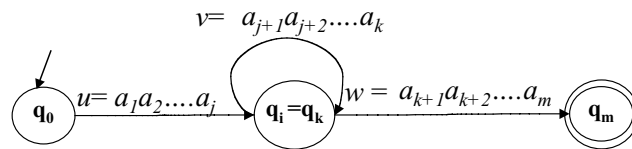
- ❖ **Beweis:** Ausgangspunkt ist der Übergangsgraph aus unseren Vorüberlegungen.
- ❖ Wir belegen  $z$ ,  $u$ ,  $v$  und  $w$  einfach wie folgt:

$$z = a_1 a_2 \dots a_m$$

$$u = a_1 a_2 \dots a_j$$

$$v = a_{j+1} a_{j+2} \dots a_k$$

$$w = a_{k+1} a_{k+2} \dots a_m$$



## Anwendungen des Pumping-Lemmas

- ❖ Wir benutzen das Pumping-Lemma mit Vorliebe dann, wenn wir zeigen wollen, dass eine bestimmte Sprache **nicht** von einem endlichen Automaten erkannt werden kann.
- ❖ Wir benutzen dabei die Beweismethode "Beweis durch Widerspruch":
  - ♦ Wir nehmen an, die Sprache ist regulär, d.h. von einem endlichen Automaten erkennbar.
  - ♦ Dann wählen wir eine beliebige, hinreichend lange Zeichenkette
  - ♦ Wir pumpen sie nach dem Pumping-Lemma auf
  - ♦ Dann zeigen wir, dass die so aufgepumpte Zeichenkette nicht in der Sprache liegt.
  - ♦ Das ist ein Widerspruch zur Annahme, denn nach dem Pumping-Lemma müssen alle aufgepumpten Zeichenketten in der Sprache liegen.

## "Algorithmus" für das Ausführen von Beweisen mit dem Pumping-Lemma

**2-Personenspiel: Kontrahent** behauptet, die Sprache  $L$  ist regulär. **Sie** behaupten, dass die Sprache  $L$  nicht regulär ist.

- 1) Der **Kontrahent** behauptet, dass  $L$  regulär ist, also muss das Pumping-Lemma gelten.
- 2) **Sie** bitten Ihren **Kontrahenten**, eine beliebige endliche Zahl  $n$  auszuwählen.  $n$  ist beliebig, aber einmal ausgewählt ist  $n$  fest.
- 3) **Sie** wählen nun eine Zeichenkette  $z$  aus  $L$  mit  $|z| \geq n$ . Die Wahl von  $z$  ist im allgemeinen von  $n$  abhängig.
- 4) **Sie** bitten Ihren **Kontrahenten**,  $z$  in  $u$ ,  $v$  und  $w$  aufzubrechen, wobei die Einschränkungen  $|uv| \leq n$  und  $|v| \geq 1$  einzuhalten sind.
- 5) Nun pumpen **Sie** die Zeichenkette  $i$ -mal nach dem Pumping-Lemma auf.
- 6) Dann zeigen **Sie**, dass die Zeichenkette  $uv^i w$  nicht in  $L$  ist. Das ist ein Widerspruch zur Annahme 1), denn es würde bedeuten, dass das Pumping-Lemma nicht gilt.
- 7) Wenn der **Kontrahent** alle möglichen Zerlegungen ausprobiert hat, haben **Sie** gewonnen:  $L$  kann nicht regulär sein.

## Beispiel für eine nicht-reguläre Sprache

- ❖ Gegeben sei die Sprache  $L = \{ a^n b^n \mid n > 0 \}$
- ❖ Beispiel für  $L$ :  $a = ($  und  $b = )$ . Dann gilt unter anderem
  - ♦ In  $L$ :  $()$ ,  $(( ))$ ,  $(( ( )))$ ,  $(( ( ( )))$
  - ♦ Nicht in  $L$ :  $(, )$ ,  $((, ( )))$ ,  $(( (, )) ( ($
- ❖ Kontrahent behauptet,  $L$  ist regulär. **Sie** behaupten,  $L$  ist nicht regulär. Kontrahent wählt  $n = 4$ . Wir wählen die Zeichenkette  $z = ((( ( ))) \in L$
- ❖ Kontrahent bricht  $z$  auf in  $u = (((, v = ($  und  $w = ))))$
- ❖ Wenn  $L$  regulär ist, gilt das Pumping-Lemma.
- ❖ Also können **Sie**  $v$  aufpumpen, z.B. auf  $v^2$ , eine Konkenation von 2 öffnenden Klammern:  $v^2 = (($
- ❖  $uv^2w$  ist nicht in  $L$ , da die öffnenden und schließenden Klammern eindeutig nicht balanziert sind.
- ❖ Wegen des Pumping-Lemmas muss  $uv^2w$  aber in  $L$  liegen. Widerspruch für diese Zerlegung!
- ❖ Widerspruch für **jede** Zerlegung  $\Rightarrow L$  kann nicht regulär sein. q.e.d.

### ***Es gibt eine Grammatik für $L = \{ a^n b^n \mid n > 0 \}$***

❖ Die Chomsky-Grammatik  $G = (T, N, P, Z)$  mit

$$N = \{Z\},$$

$$T = \{a, b\} \text{ und}$$

$$P = \{ Z \rightarrow aZb,$$

$$Z \rightarrow ab \}$$

erkennt die Sprache  $L = \{ a^n b^n \mid n > 0 \}$

❖ Beispiel einer Ableitung:

$$Z \Rightarrow aZb \Rightarrow aaZbb \Rightarrow a^3Zb^3 \Rightarrow \dots \Rightarrow a^{n-1}Zb^{n-1} \Rightarrow a^n b^n$$

❖  $G$  ist nicht regulär, denn die Produktion  $Z \rightarrow aZb$  ist weder linkslinear noch rechtslinear.

❖ Man bezeichnet solche Produktion als ***kontextfrei***. Eine Grammatik, die kontextfreie Produktionen enthält, bezeichnet man als ***Chomsky-2 oder kontextfreie Grammatik***

*mehr über Grammatiken in Info IV*

### ***Zusammenfassung***

❖ Für jede Sprache, die von einem ***nicht-deterministischen endlichen Automaten*** akzeptiert wird, gibt es einen ***äquivalenten deterministischen endlichen Automaten***.

❖ Wir brauchen daher zwischen deterministischen und nicht-deterministischen endlichen Automaten nicht zu unterscheiden und sprechen deshalb nur von ***endlichen Automaten***.

❖ Das ***Pumping-Lemma*** ist ein wichtiges Werkzeug, wenn man zeigen möchte, dass eine bestimmte Sprache nicht regulär ist, also von einem endlichen Automaten nicht erkannt werden kann.