Zusammenfassung Automaten und formale Sprachen FS 17

# Einleitung

## Grundbegriffe

### Definition 1

Ein **Alphabet** ist eine Menge von Zeichen (Symbolen, Buchstaben). Eine endliche Folge x1x2…xn (wobei die xi aus stammen) wird ein **Wort** der Länge n genannt. Die Länge eines Wortes u wird mit |u| bezeichnet.

Für jedes Alphabet existiert ausserdem das leere Wort der Länge 0, welches mit bezeichnet wird.

Für jedes Alphabet schreiben wir \* für die Menge aller möglichen Wörter über . Ausserdem sei + die Menge aller möglichen Wörter ohne das leere Wort.

### Definition 2

Zwei Wörter sind genau dann **gleich** wenn sie gleich lang sind und das i-te Zeichen des einen Wortes immer gleich dem i-ten Zeichen des anderen Wortes ist.

Für Wörter u= x1x2…xn und v = y1y2…ym ist uv := x1x2…xny1y2…ym als die **Konkatenation** definiert. Die Konkatenation von u und dem leeren Wort gibt wieder u.

Die n-te Potenz un eines Wortes u ist definiert als u0 := und un+1 = unu

#### Bemerkung

Für jedes Alphabet gilt u(vw)=(uv)w (Kommutativität).

### Definition 3

Ein Wort u heisst **Teilwort** eines Wortes v, genau dann wenn es Wörter w1 und w2 gibt, so dass v=w1uw2 gilt. Ist w1=, nennen wir u ein Anfangsstück von v, ist w2= ist u ein Endstück von v.

Das **Spiegelbild** sp(u) eines Wortes u ist definiert durch sp() := und sp(x1x2…xn) := xn…x2x1 . Wörter mit der Eigenschaft sp(u)=u werden **Palindrome** genannt. ^

### Definition 4

Gegeben das Alphabet

Sind M und N Teilmengen von \*, so ist das **Produkt** MN definiert durch MN := {uv: u aus M und v aus N}. Also alle möglichen Kombinationen von Wörtern aus M und N. Häufig schreibt man für dieses Produkt nur MN. Für mehrere Teilmengen Mi von \* gilt ausserdem: M1M2…Mn := (…(M1M2)…Mn)

Für jede Teilmenge M von \* und jede natürliche Zahl n ist die n-te Potenz induktiv definiert als   
M0 :={} und Mn+1M. Ausserdem setzen wir M\* := (i eine natürliche Zahl) und M+:=M\* ohne das leere Wort.

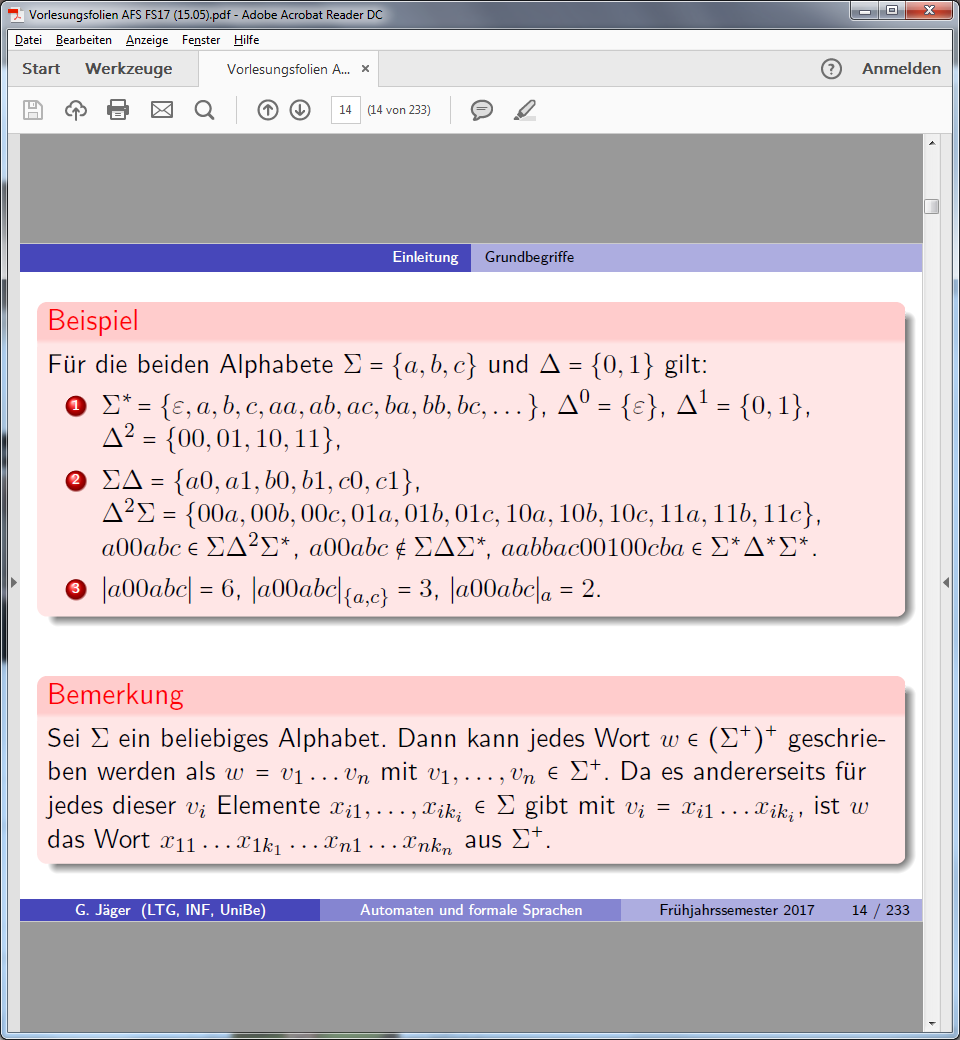
#### Bemerkung



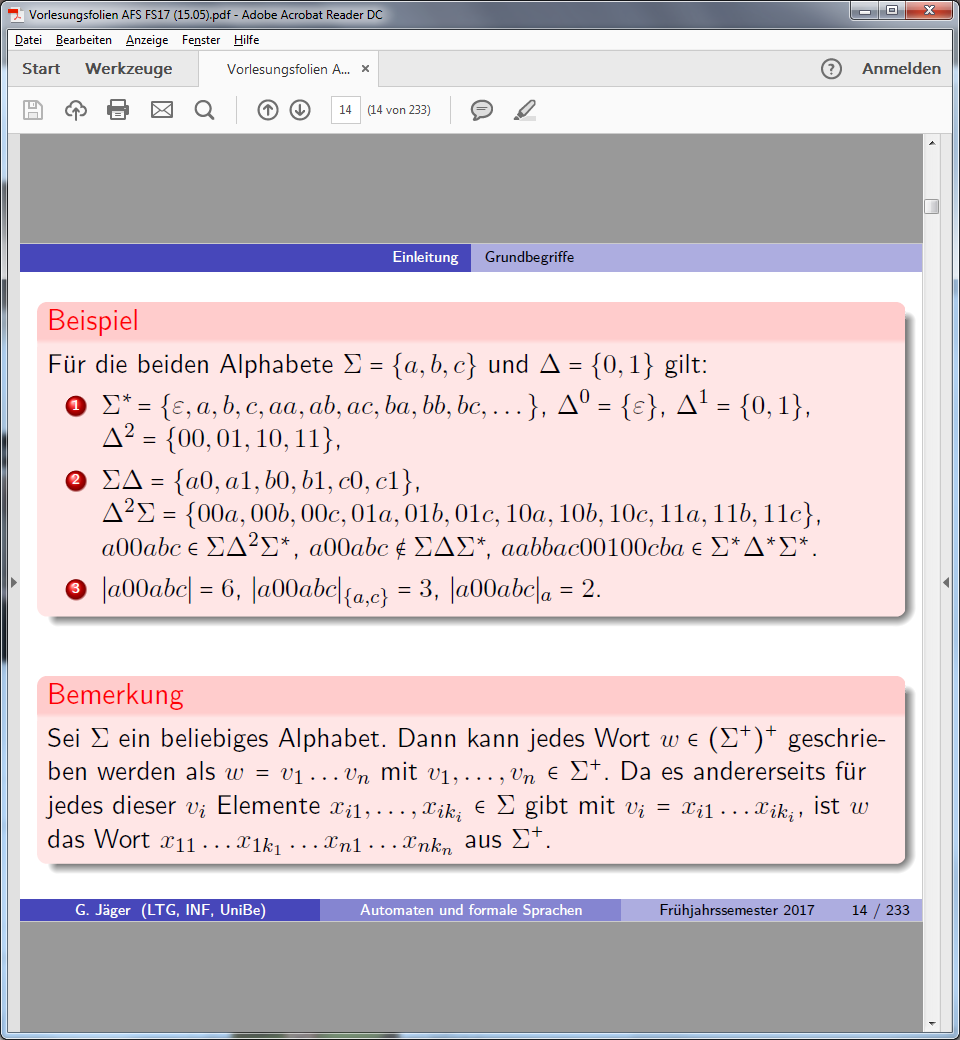
### Definition 5

Gegeben seien ein Alphabet , eine Teilmenge M davon, sowie ein Wort u aus \* und ein Symbol x aus . Dann setzen wir |u|M := Anzahl der Vorkommnisse von Elementen (Symbolen) aus M in u und |u|x := Anzahl der Vorkommnisse von x in u.

#### Beispiel



#### Bemerkung



### Lemma 6

Für jedes Alphabet und alle M,N,N1,N2 Teilmengen von \* gilt

und \*=(M\*N)\*M\*.

### Definition 7

Gegeben sei ein Alphabet . Dann bezeichnet man jede Teilmenge L von \* als **formale Sprache** über .

# Endliche Automaten

## Allgemeines

Ein endlicher Automat ist ein Automat, der sich nur in endlich vielen verschiedenen Zuständen befinden kann. Sehr schematisch kann man einen Automaten in drei Komponeneten untergliedern: Die **Eingabe**, womit dem Automaten von aussen Informationen geliefert werden (z.B. Geldeinwurf). Den **Zustandsraum**: Ein Automat befindet sich zu jedem Zeitpunkt in einem bestimmten Zustand. Als Reaktion auf eine Eingabe kann er dann etwa eine bestimmte Sequenz von Zuständen durchlaufen. Die **Ausgabe**, welche ein Text oder Waren sein kann. Eine Eingabe muss aber nicht zwingend zu einer Ausgabe führen.

Deterministisch heisst, dass der Automat von einem Zustand mit einem Input nur in einem weiteren Zustand landen kann.

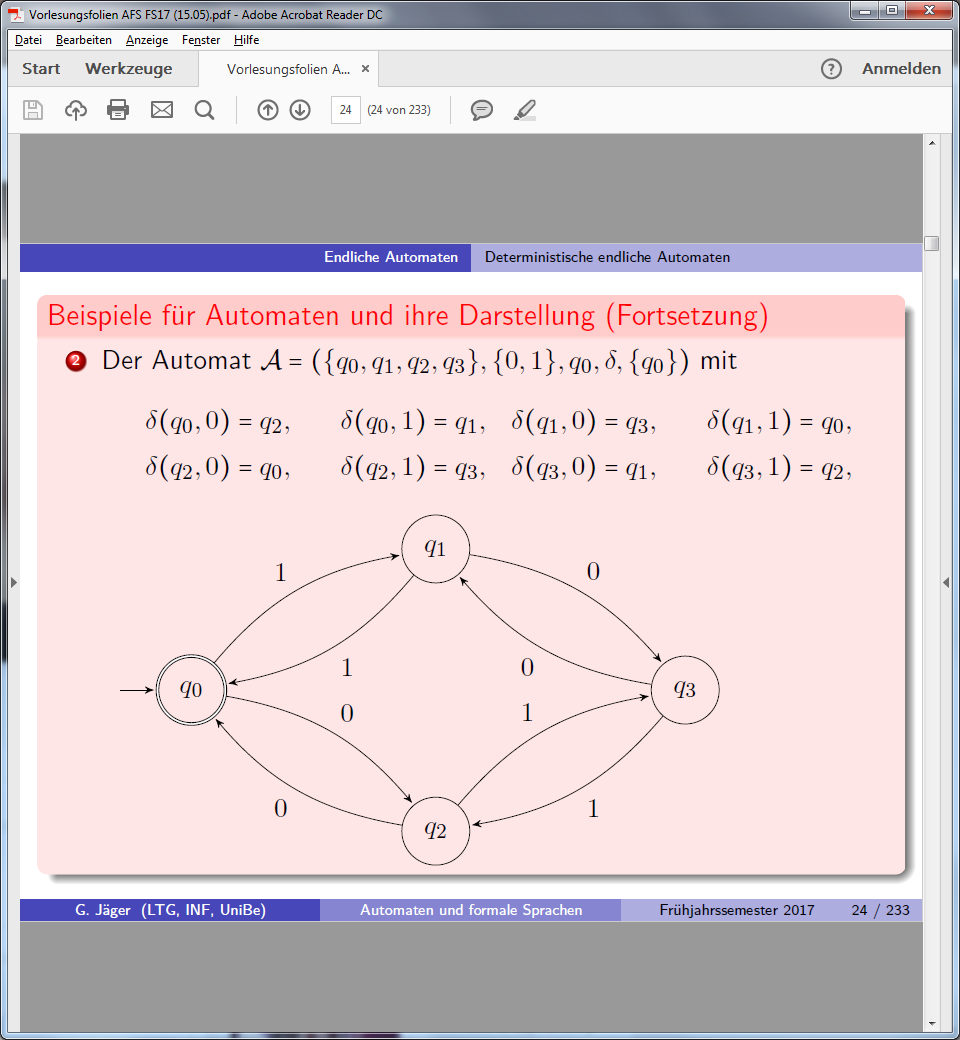
## Deterministische endliche Automaten

Ein **deterministischer endlicher Automat (DEA)** ist ein 5-Tupel A = (Q, , q0, ,F).   
Q ist eine nicht-leere endliche Menge, die **Zustandsmenge**  
 ist eine nicht-leere endliche Menge, das **Eingabealphabet**  
q0 aus Q ist der **Anfangszustand**  
 : Q x -> ist die **Transitionsfunktion**, die von einem Zustand mit einem Eingabeelement in einen Zustand überführt.   
F Q ist die Menge der **Endzustände**.

Der Zustand q‘ = (q,a) nennt man den **Nachfolgezustand von q unter a**. Befindet sich ein Automat im Zustand q und liest das Eingabezeichen a, geht er in den Zustand q‘. Vor dem ersten Einlesen befindet sich der Automat im Startzustand q0.

### Graphische Darstellung von DEAs durch Transitionsgraphen

Knoten repräsentieren Zustände, beschriftete Kanten Übergänge und der Anfangszustand ist der Knoten mit eingehender Knoten ohne Quelle, die Endzustände sind doppelt umkreist.



### Definition 9

Gegeben sei der DEA A= (Q, , q0, ,F). Durch Induktion nach dem Aufbau der Wörter über dem Alphabet definieren wir nun die Erweiterung \*: Q x\* -> Q

Das Wort wird also von links nach rechts, Schritt für Schritt, abgearbeitet und jeweils der allfällige Zustandswechsel vollzogen.

Für ein DEA gilt offensichtlich \*(q,a) = (q,a).

### Lemma 10

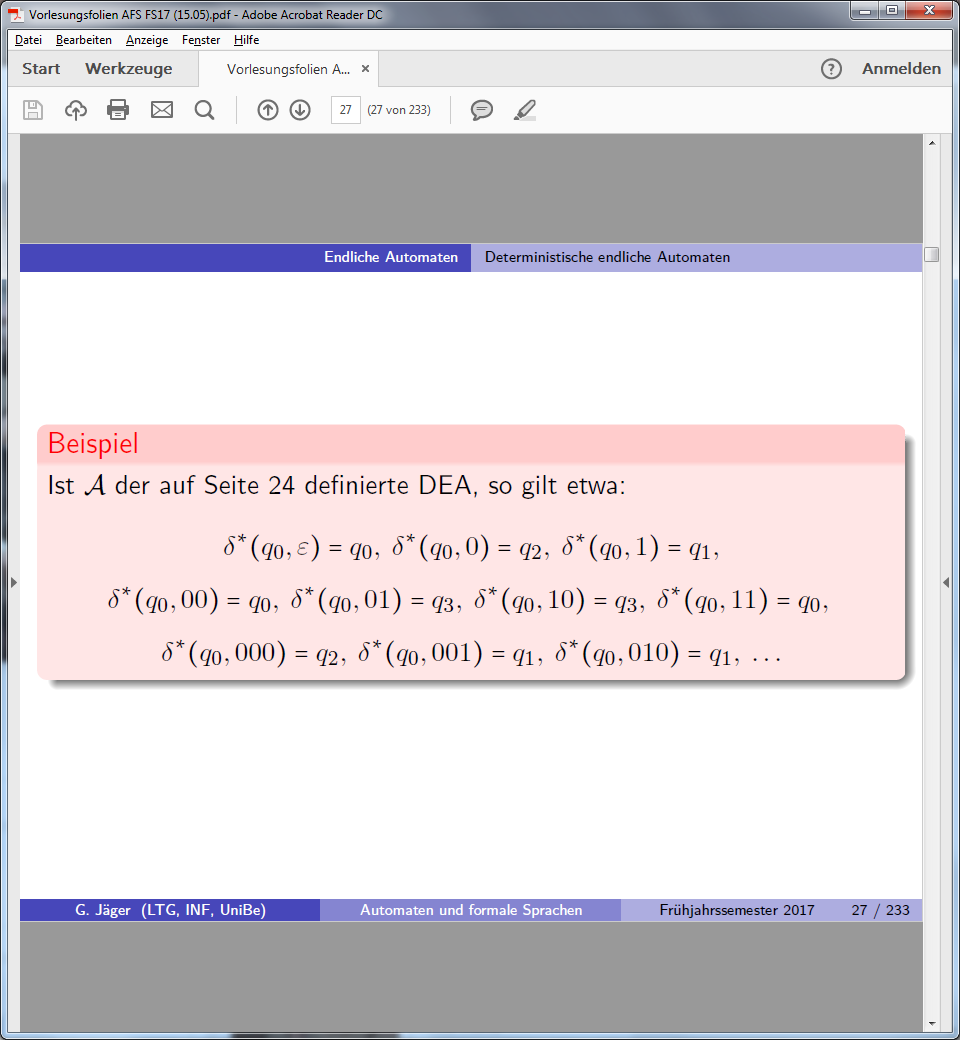
Sei A= (Q, , q0, ,F) ein DEA.

Ist a1…an ein Wort aus \* und ist q1,…qn die Folge von Zuständen, die definiert ist durch   
qi+1:=(qi, ai+1), dann gilt: \*(q0, a1…an) = qn

Für alle q aus Q und alle u,v aus \* gilt

Also wird zuerst das Wort u, dann das Wort v abgearbeitet.

#### Beispiel



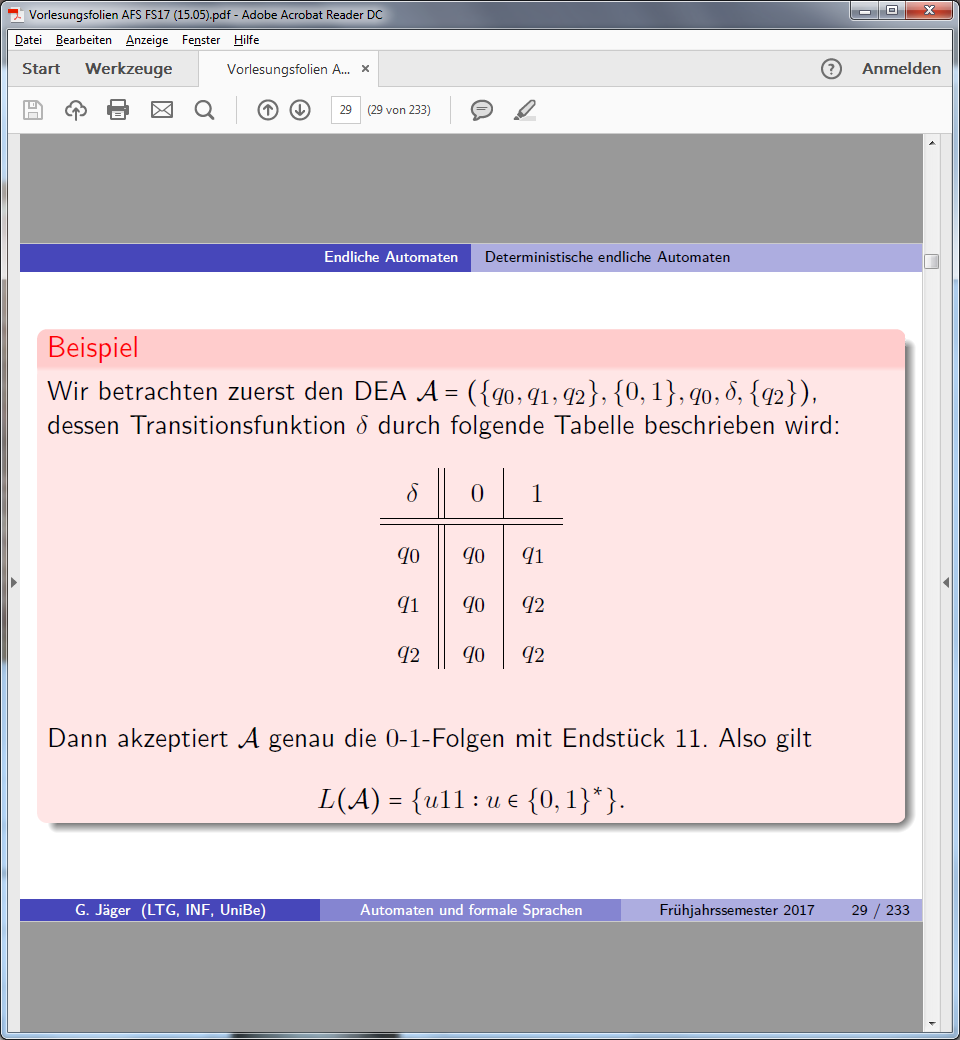
### Definition 11

Es sei A= (Q, , q0, ,F) ein DEA. Dann sagen wir, dass A das Wort u aus \* genau dann **akzeptiert**, wenn \*(q0, u) ein Endzustand ergibt. Also wenn der Automat nach dem Abarbeiten des Wortes sich in einem Endzustand befindet.

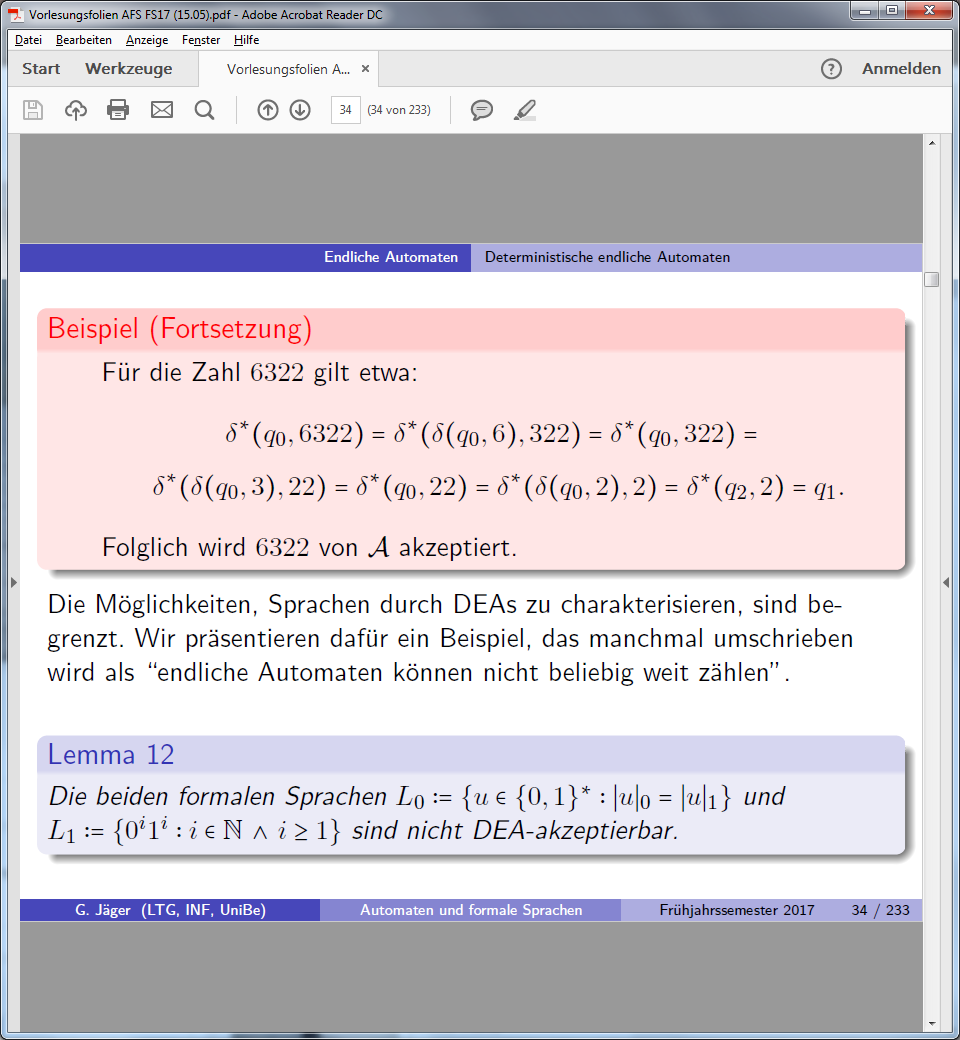
Die von einem DEA A **akzeptierte Sprache L(A)** ist definiert als alle Wörter, die der DEA akzeptiert.

Eine Sprache L heisst genau dann **DEA-akzeptierbar**, wenn es einen DEA A gibt mit L=L(A).

#### Beispiel



### Lemma 12



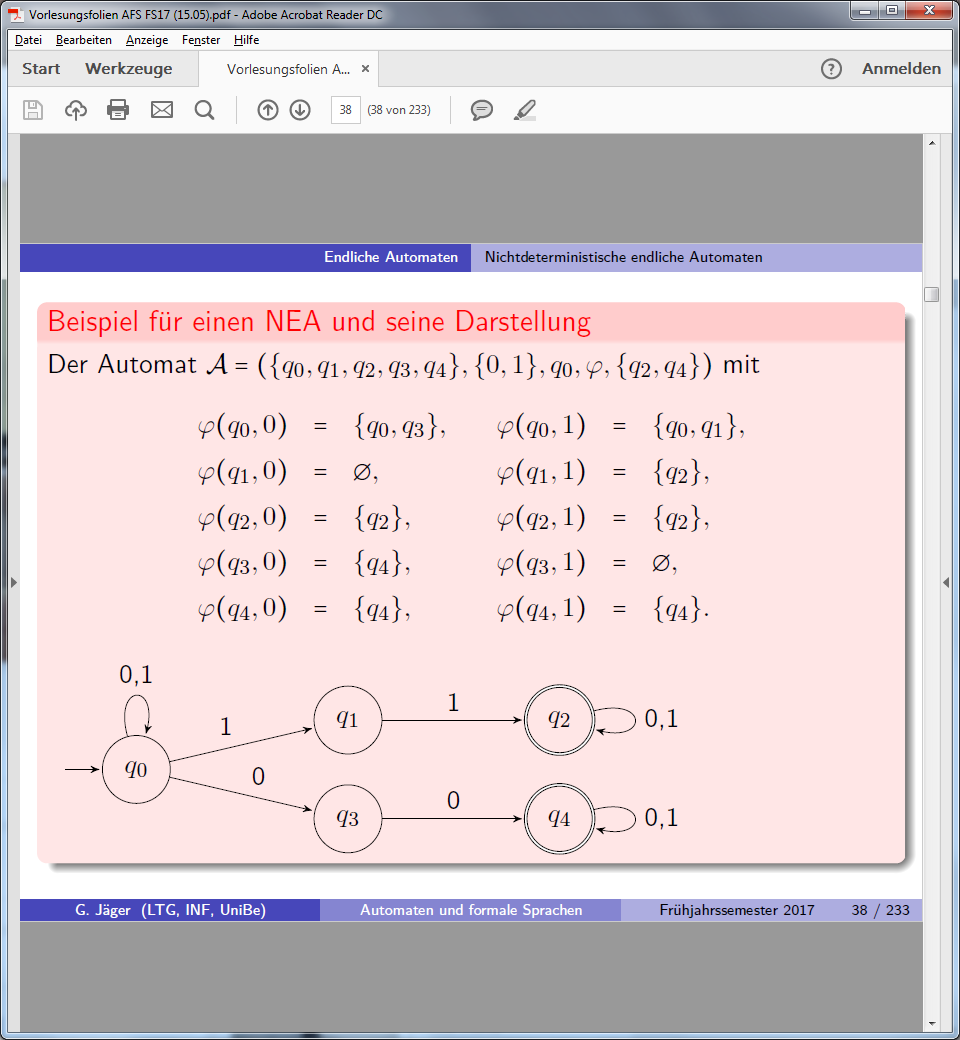
## Nichtdeterministische endliche Automaten

Im Gegensatz zum DEA kann ein Automat hier bei beliebigen Zuständen für einen Input in mehrere Zustände wechseln.

### Definition 13

Ein **nichtdeterministischer endlicher Automat (NEA)** ist ein 5-Tupel A = (Q, , q0, ,F), wobei der Unterschied zum DEA bei der Transitionsfunktion liegt:

: Q x -> p(Q), wobei p(Q) die Potenzmenge (alle möglichen Mengen formbar aus Q) von Q ist. (q, a) = ist dabei zugelassen.



### Definition 14

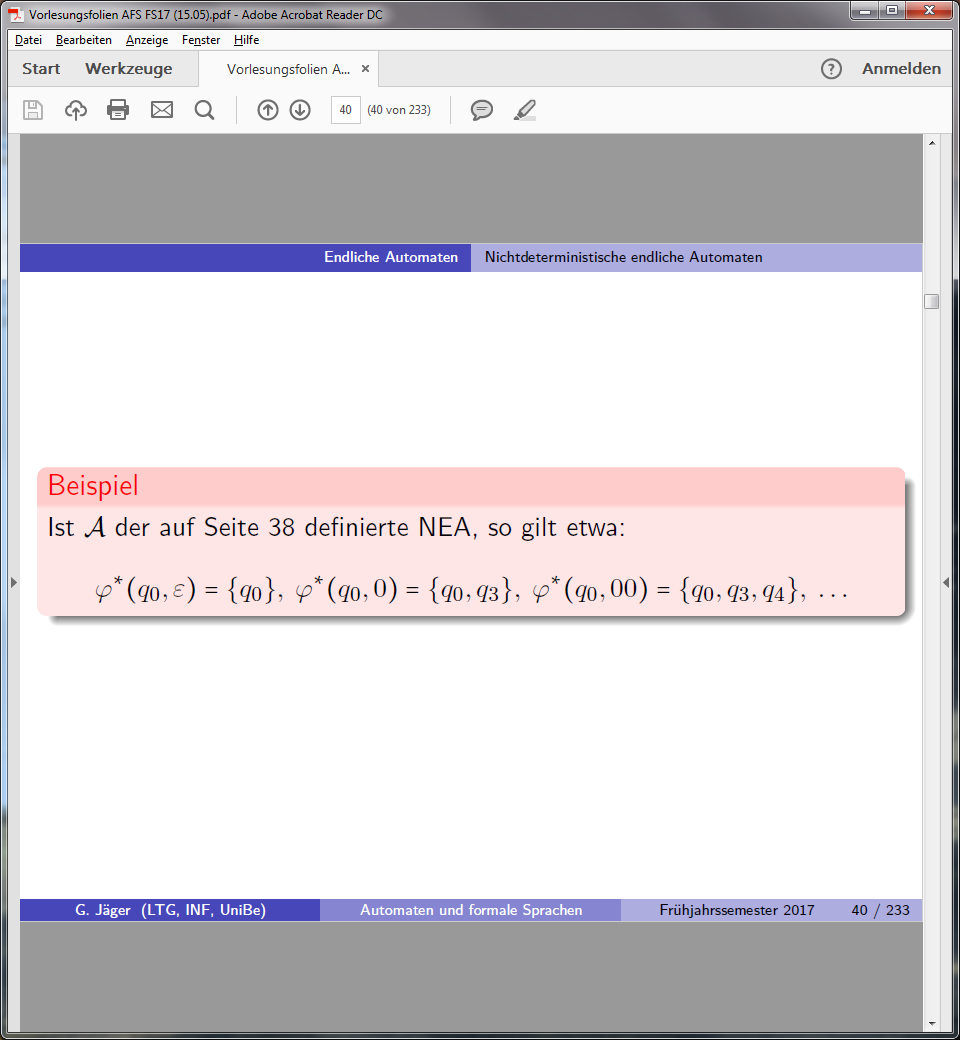
Wir erweitern nun die Transitionsfunktion induktiv auf Wörter, ähnlich wie beim DEA.

Dabei ist \*(q,) := {q} und \*(q,ua) := wobei q aus Q, u aus \* und a aus ist. Ausgeschrieben wird für jeden Zustand aus der Zustandsmenge, die nach u erreicht/erreichbar ist geschaut, in welchen Zuständen der Automat nach a landet und dann diese Menge vereinigt.

Intuitiv beschreibt diese Menge dann alle Zustände, in der der Automat nach dem Abarbeiten von links nach rechts landen kann.

#### Beispiel

Mit dem oben definierten NEA würde dann gelten



### Definition 15

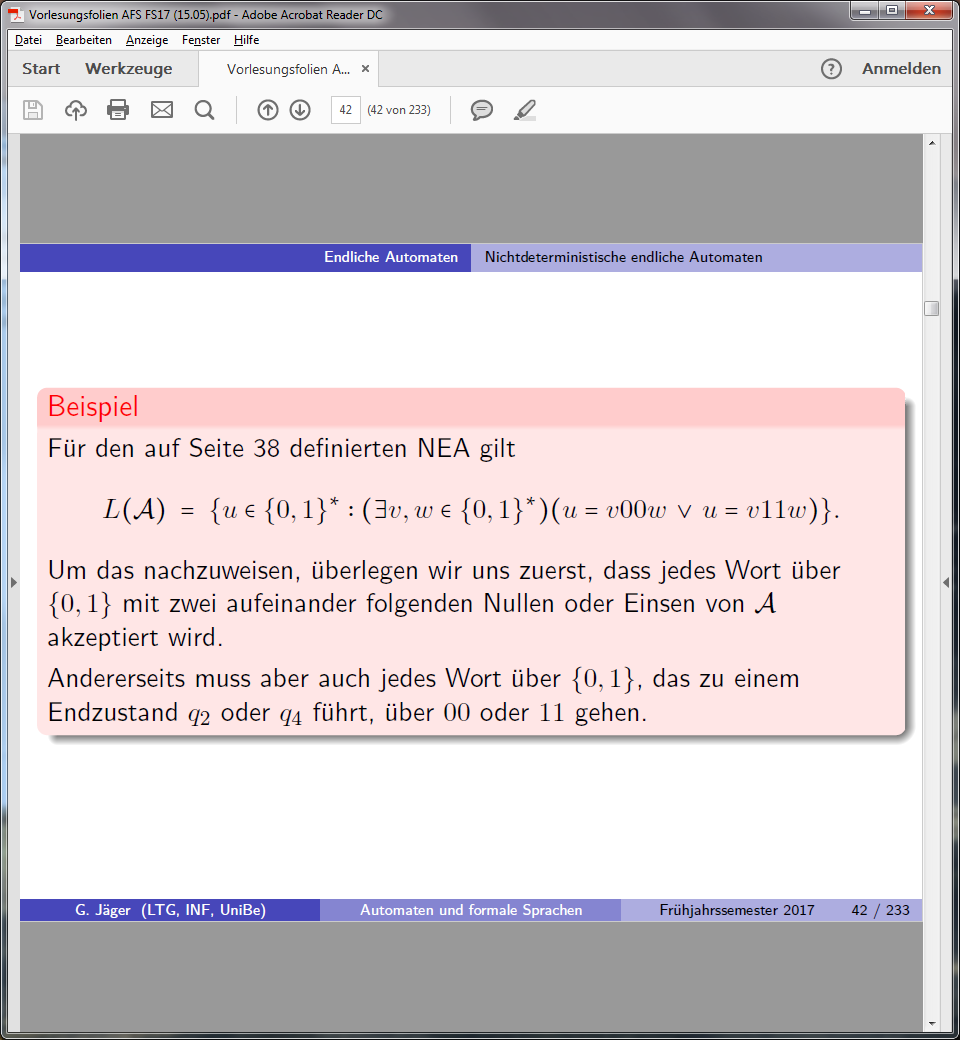
Sei A = (Q, , q0, ,F) ein NEA. Dann sagen wir, dass A das Wort u aus \* genau dann **akzeptiert**, wenn \*(q0, u) F . Also wenn einer der Zustände aus der Zustandsmenge, die nach u erreichbar ist ein Endzustand ist.

Die vom NEA A **akzeptierte Sprache L(A)** ist dann die Menge der akzeptierten Wörter.

Eine Sprache heisst genau dann **NEA-akzeptierbar**, wenn es einen NEA A mit L = L(A) gibt.

#### Beispiel

Mit dem oben definierten NEA würde dann gelten



### Lemma 16

Jeder DEA kann auch als NEA verstanden werden und für jeden NEA gibt es einen DEA, der dieselbe formale Sprache akzeptiert.

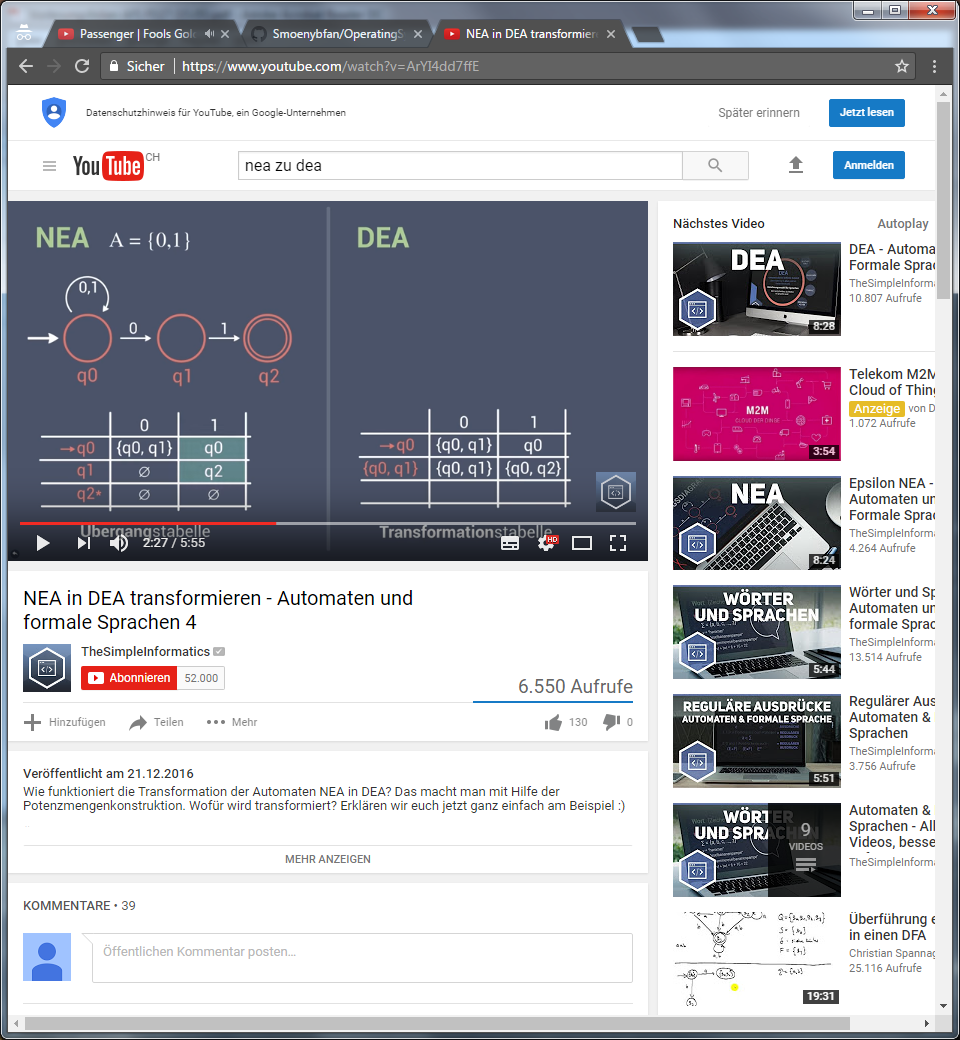
Sei A := (Q, , q0, ,F) ein beliebiger DEA. Dann definieren wir : Q x -> p(Q), (q, a) :={}, so ist B := (Q, , q0, ,F) ein NEA, für den gilt \*(q, u) = {\*(q, u)}. Weiter gilt L(A) = L(B).

### Theorem 17

Für jeden NEA A gibt es einen DEA B, mit L(A) = L(B), also einen DEA, der dieselbe Sprache akzeptiert.

Diesen DEA finden wir über die Potenzmengenkonstruktion. Wir übernehmen den Startzustand des NEA. Dann schauen wir, in welchen Zuständen, beziehungsweise Mengen von Zuständen der NEA vom Startzustand aus landet. Die Einträge, die so entstanden sind werden zu Zuständen und für diese wird dann wieder geprüft, in welchen Zuständen der NEA (für jeweils alle Zustände der Menge) landet, usw.

Im Grunde genommen wird für jede Teilmenge der Potenzmenge betrachtet, in welchen Mengen von Zuständen sie landet. Der oben beschriebene Weg lässt bereits einige unnötige Zustände weg (betrachte Bemerkung).

#### Bemerkung

Hat ein NEA n Zustände, hat der konstruierte äquivalente Automat 2n Zustände. Viele dieser Zustände sind aber nicht notwendig, da sie nicht erreichbar sind und können weggelassen werden. Das ist im obigen Beispiel schon geschehen, z.B. {q0,q1,q2} ist kein erreichbarer Zustand des NEA.

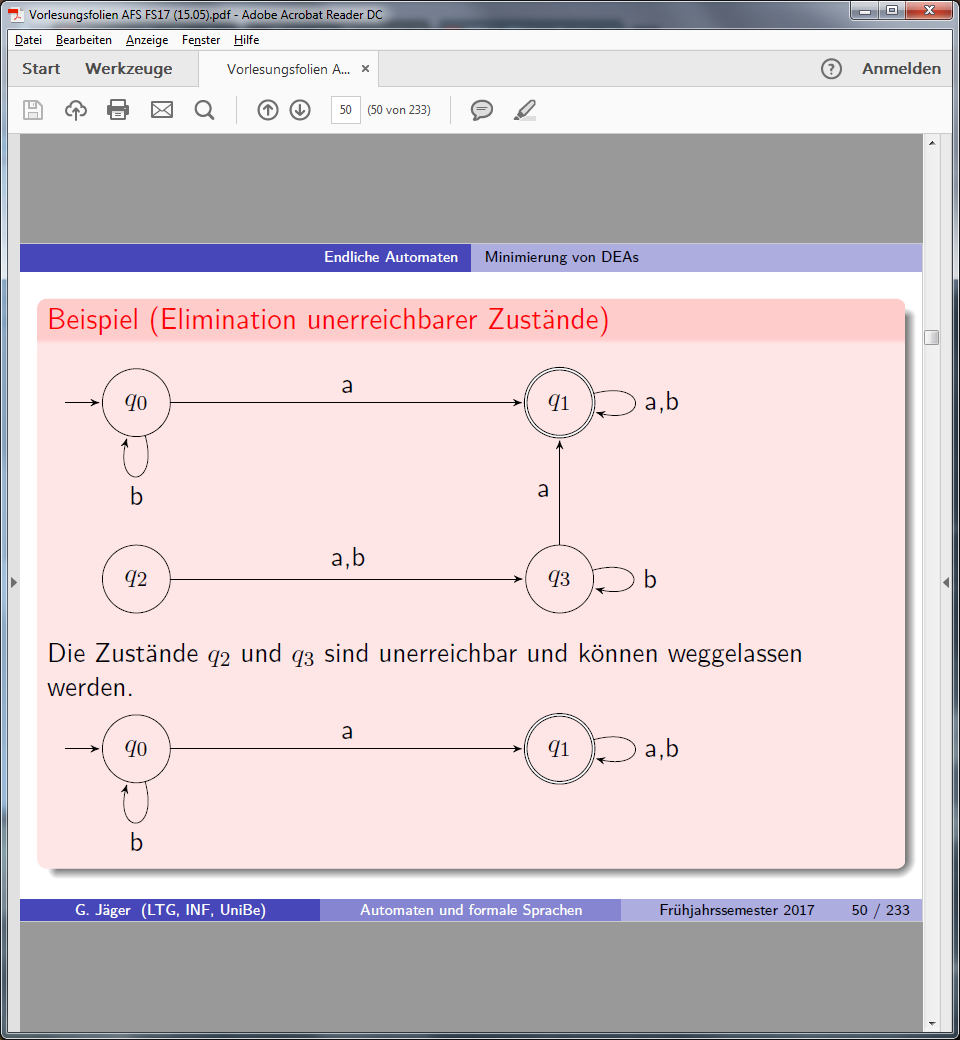
## Minimierung von DEAs

Wir wollen nun einen DEA in einen DEA transformieren, der mit möglichst wenigen Zuständen auskommt. Dafür werden unerreichbare Zustände eliminiert und eine Identifizierung von Zuständen durchgeführt.

### Schritt 1

Wir bestimmen im gegebenen DEA alle Zustände, die vom DEA erreichbar sind. Alle unerreichbaren Zustände können dann gestrichen und die Transitionsfunktion entsprechend angepasst werden.

#### Beispiel

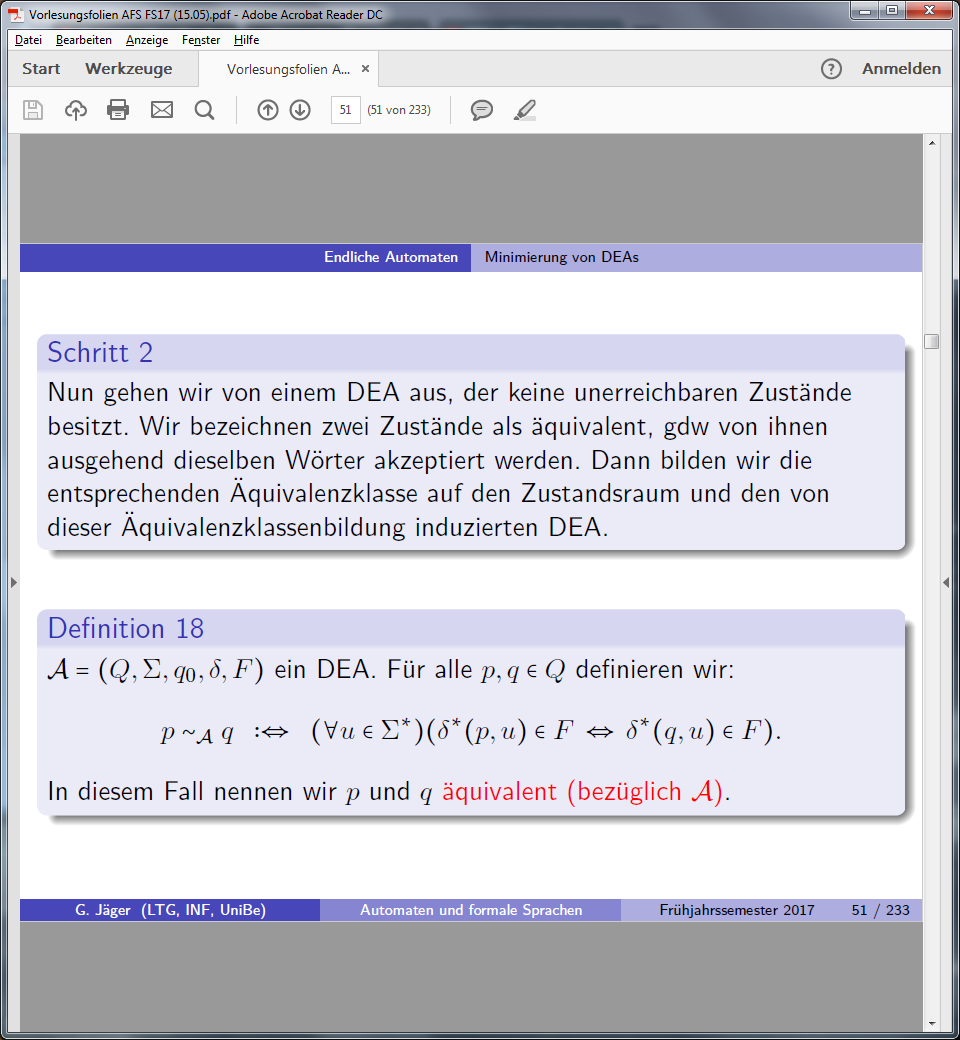


### Schritt 2

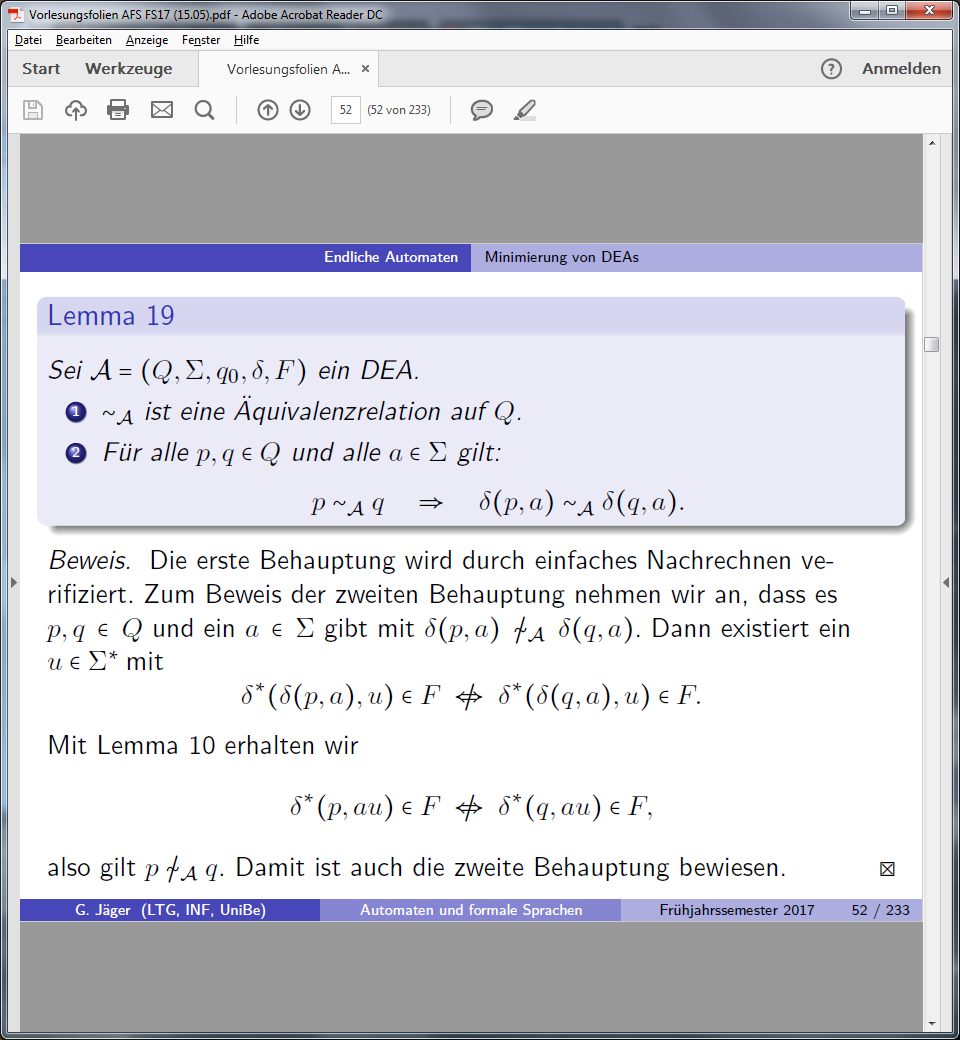
Wir gehen nun von einem DEA aus, der keine unerreichbaren Zustände mehr besitzt. Wir bezeichnen zwei Zustände genau dann als äquivalent, wenn von ihnen ausgehend dieselben Wörter akzeptiert werden. Wir können dann die Äquivalenzklassen zu jeweils einem Zustand zusammenfassen.

### Definition 18

Sei A := (Q, , q0, ,F) ein DEA. Zwei Zustände p,q aus Q sind genau dann **äquivalent (bez. A)**, wenn gilt:

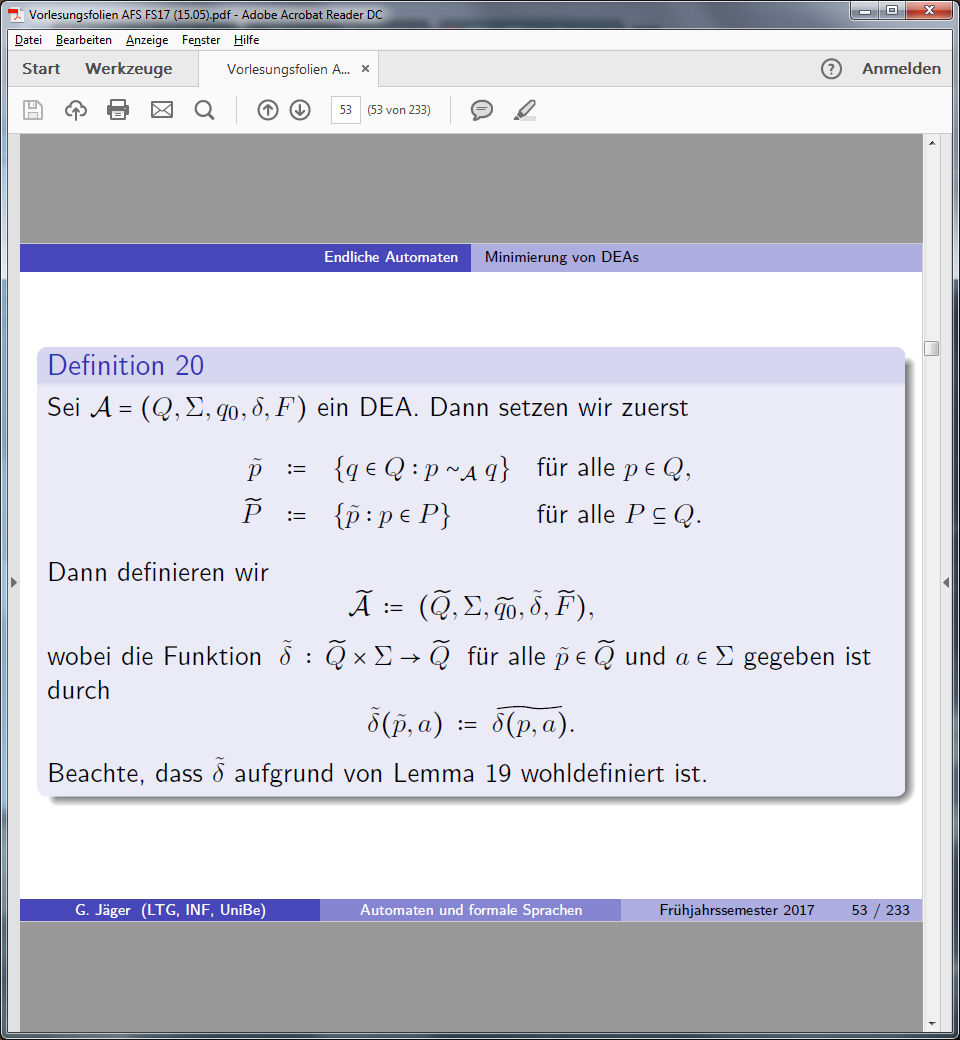


### Lemma 19



2 besagt, dass wenn zwei Zustände äquivalent sind, es auch der Zustand ist, der von beiden mit dem gleichen Eingabesymbol erreichbar ist.

### Definition 20



Wir sehen, dass die Menge der zu p äquivalenten Zustände sind. ist dann die Menge der Zustände, die zu den Elementen von P äquivalent sind.

Die Funktion ist dann gegeben durch . Es wird also zuerst die Transitionsfunktion ausgeführt und dann vom resultierenden Zustand aus die Menge der äquivalenten Zustände gebildet.

### Theorem 21

Wenn A ein DEA ist, so ist der nach Definition 20 gebildete DEA ein DEA und es gilt L(A) = L()

Im Grunde genommen bestimmen wir die Äquivalenzklassen und ersetzen dann alle Zustände dieser Äquivalenzklasse durch jeweils einen Vertreter. Die Übergangsfunktion bildet dabei von einer Äquivalenzklasse, beziehungsweise deren Vertreter auf eine Äquivalenzklasse ab.

### Definition 22

Sei A := (Q, , q0, ,F) ein DEA. Die Zustände p,q Q heissen genau dann **k-trennbar** bez. A wenn es ein Wort u \* der Länge |u| k gibt, für das gilt:



Zwei Zustände sind also dann k-trennbar wenn für die Eingabe u nur genau eines der beiden in einem Endzustand ankommt.

#### Bemerkung

Zwei Zustände sind genau dann äquivalent, wenn sie für kein k trennbar sind.

### Markierungsalgorithmus

Wir starten mit der Eingabe eines DEA A := (Q, , q0, ,F)

Nun markieren wir alle Paare {p,q} von Zuständen, bei denen jeweils genau ein Zustand ein Endzustand und der andere kein Endzustand ist.

Nun sind wir in der Schleife. Wir markieren hier jedes Paar {p,q} von Zuständen, für das ein Symbol   
a existiert, so dass {(p,a), (q,a)} bereits markiert ist. Das machen wir so lange, bis sich keine neuen Markierungen ergeben.

### Theorem 23

Sei A := (Q, , q0, ,F) ein DEA.

Der Markierungsalgorithmus terminiert bei Input A.

Die dadurch berechnete (markierte) Menge ist die Menge aller Paare, die nicht äquivalent sind.

Der DEA kann also nur mit dieser Menge an Zuständen konstruiert werden. Die nicht markierten Zustände sind alle in einer Äquivalenzklasse einer dieser Zustände und können weggelassen werden.

#### Bemerkung

Diese Minimierung ist bis auf Isomorphie eindeutig.

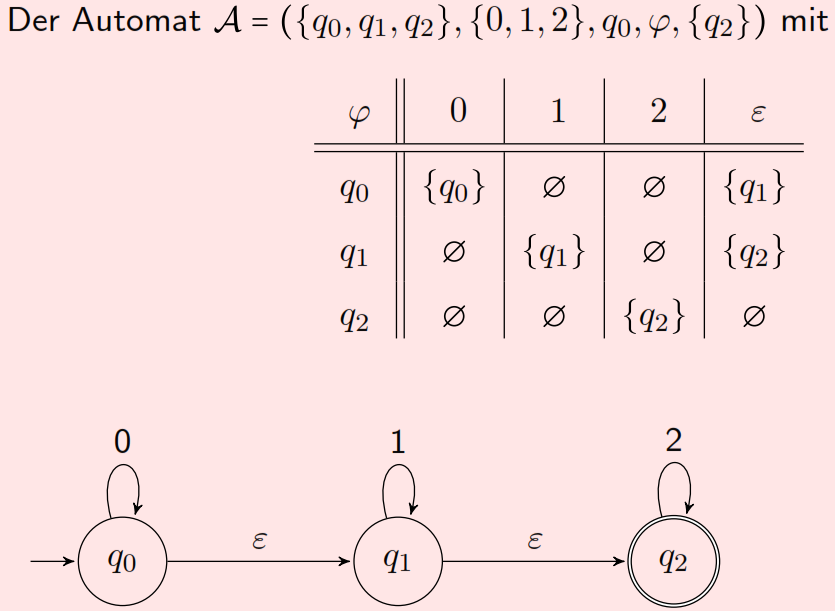
## -Automaten

Nun erweitern wir die NEA zu Automaten, bei denen auch ein Zustandswechsel erfolgen kann, ohne dass ein Zeichen eingelesen wird. Formal geschieht das dadurch, dass auch Paaren (q, ε) eine Zustandsmenge zugeordnet wird.

### Definition 24

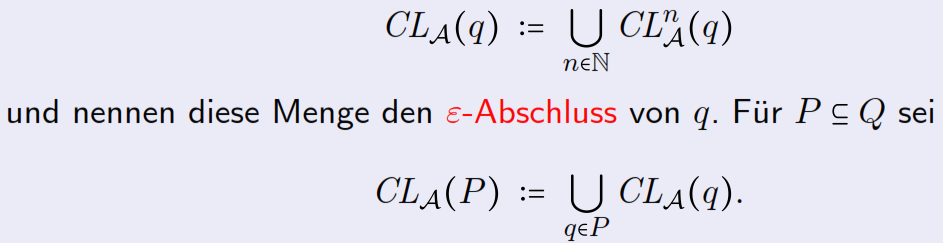
Ein **ε-Automat** ist ein 5-Tupel(Q, , q0, ,F), bei dem Q, Σ, q0 wie in der Definition der NEA definiert sind. Der Unterschied liegt hier bei der Transitionsfunktion, für die auch das leere Wort ε eine mögliche Eingabe ist: ϕ : Q x (Σ {ε}) -> p(Q).

#### Beispiel



### Definition 25

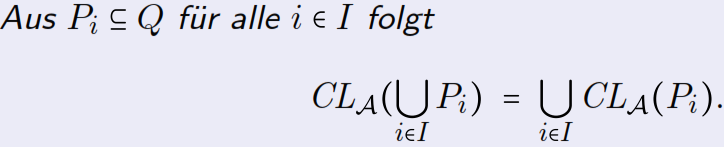
Wir führen nun für jeden ε-Automaten A und jeden seiner Zustände q den **ε-Abschluss CLA(q)** ein. Dabei handelt es sich um die Menge aller Zustände, die von q aus durch einen Pfad erreicht werden können, der nur mit ε markiert ist/nur ε als Eingabe hat.



### Lemma 26

Sein A = (Q, , q0, ,F) ein ε-Automat.

Für q aus Q und p aus CLA(q) folgt, dass CLA(p) eine Teilmenge von CLA(q) ist (logisch, da p auf diesem ε-Pfad liegen muss).



### Definition 27 (Erweiterung der Transitionsfunktion auf Wörter)

Gegeben sei der ε-Automat A= (Q, , q0, ,F). Durch Induktion definieren wir nun die Erweiterung

ϕ# : Q x Σ\* -> p(Q), indem wir setzen:

ϕ#(q, ε) := CLA(q)

ϕ#(q, ua) := CLA(

Wir lesen also weiterhin von links nach rechts, erweitern nun aber die Eingabe um ε.

### Definition 28 (Von ε-Automaten akzeptierte Sprachen)

Es sei A = (Q, , q0, ,F) ein ε-Automat. A **akzeptiert** genau dann ein Wort u, wenn es nach dem Abarbeiten dessen (inklusive möglicher ε-Übergänge) in einem Endzustand landet. Also wenn gilt ϕ#(q0, u) ∩ F ≠ .

Die von einem ε-Automaten A **akzeptierte Sprache L(A)** ist definiert als die Menge der akzeptierten Wörter.

Eine Sprache L heisst genau dann **ε-akzeptierbar**, wenn es einen ε-Automaten gibt mit L = L(A)

### Lemma 29

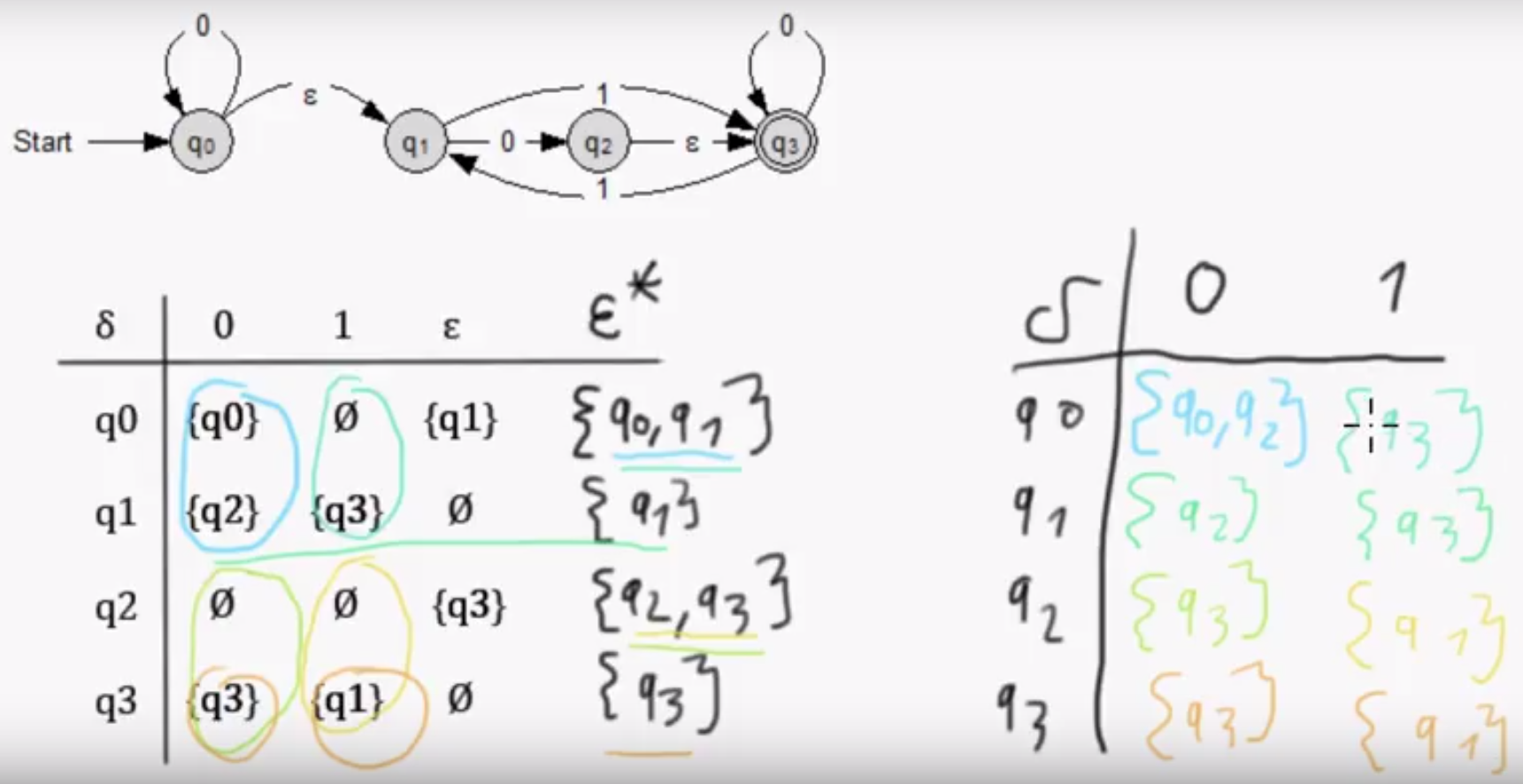
Ist A =(Q, , q0, ,F) ein ε-Automat, so gilt für alle q aus Q, u aus Σ\* und a aus Σ, dass

Wir sind also in allen möglichen Zuständen nach Abarbeitung des Wortes u und bearbeiten dann noch das Zeichen a.

### Theorem 30 (Äquivalenz von ε-Automaten und NEAs)

Für jeden ε-Automaten A gibt es einen NEA B mit L(A) = L(B).

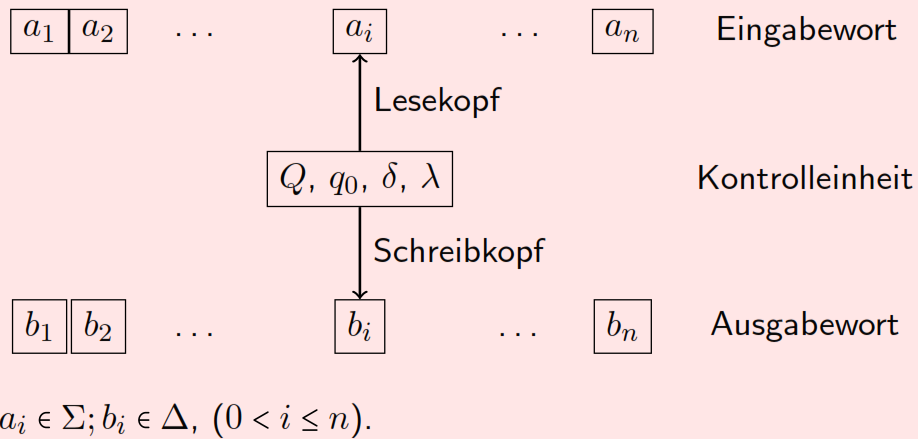
Um diesen zu konstruieren, betrachten wir wohin wir von einem bestimmten Zustand mit ε gelangen. Wenn wir zB von q0 mit ε nach q1 gelangen ist unsere Menge {q0, q1}. Nun verknüpfen wir für unseren ursprünglichen die Zustandsmengen dieser Mengen für ein Eingabesymbol. Wenn wir also beim ε-Automaten mit 0 von q0 nach q0 und von q1 nach q2 gelangen, gelangen wir bei unserem neuen NEA von q0 nach {q0, q2}.



## Endliche Automaten mit Ausgabe

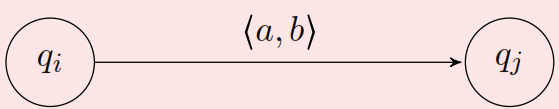
### Definition 31 (Mealy-Maschine)

Eine **Mealy-Maschine** ist ein 6-Tupel A = (Q, Σ, Δ, q0, δ, λ), wofür gilt:  
Q ist die Zustandsmenge,  
Σ ist das Eingabealphabet,  
Δ ist eine nicht-leere endliche Menge, das **Ausgabealphabet**,  
q0 ist der Anfangszustand,  
δ ist die Transitionsfunktion,  
λ : Q x Σ -> Δ ist die **Ausgabefunktion.**



(Dies ist nicht die Darstellung einer Mealy-Maschine sondern dient der Veranschaulichung)

Die Mealy-Maschine geht wie ein DEA vom aktuellen Zustand q und dem Eingabezeichen a in den Nachfolgezustand δ(q, a) über und gibt das Zeichen λ(q, a) aus. Sie hat keine Endzustände. Es geht also nicht darum, Wörter abzuarbeiten, sondern eine Ausgabe zu erzeugen.

Darstellung:  
  
Wobei a das Eingabe- und b das Ausgabesymbol ist.

### Definition 32 (Erweiterung zu Wörtern)

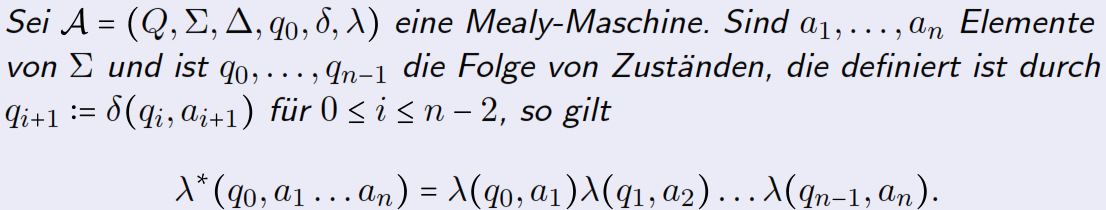
Gegeben die Mealy-Maschine A = (Q, Σ, Δ, q0, δ, λ). Wir definieren nun λ\* : Q x Σ\* -> Δ\*:

λ\*(q, ε) := ε

λ\*(q, ua) := λ\*(q,u)λ(δ\*(q,u),a)

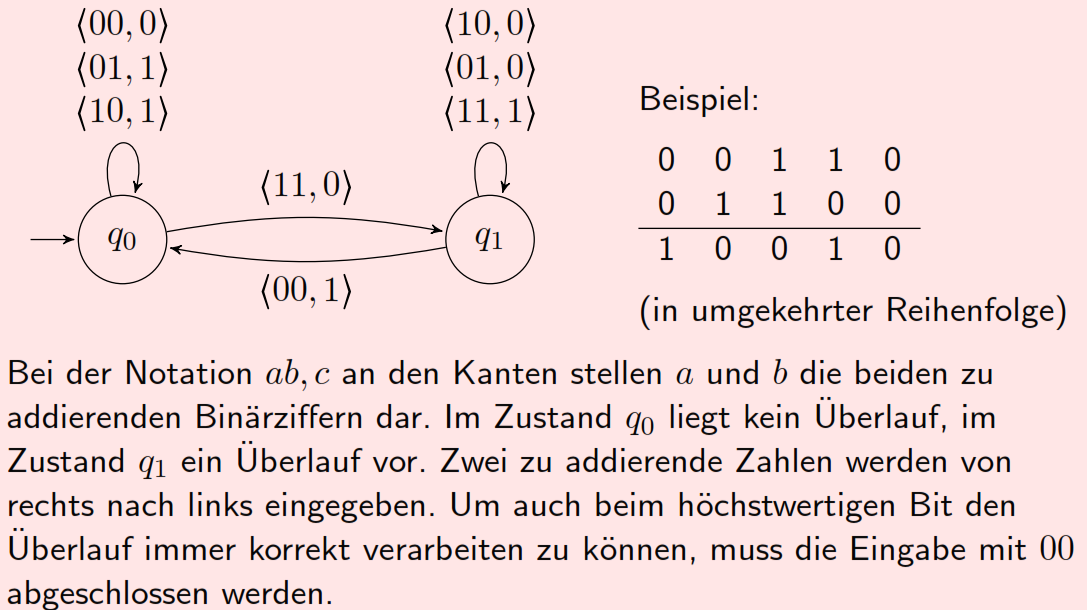
Wir arbeiten also wieder von links nach rechts Symbol für Symbol ab.

### Lemma 33



Geht aus obiger Definition hervor.

### Beispiel Mealy-Maschine zur Addition zweier Binärzahlen



### Definition 34

Eine **Moore-Maschine** ist ein 6-Tupel A = (P, Σ, Δ, p0, γ, μ), wobei gilt:  
P ist die Zustandsmenge,  
Σ ist das Eingabealphabet,  
Δ ist das Ausgabealphabet,   
p0 ist der Anfangszustand,  
γ : P x Σ -> P ist die Transitionsfunktion,  
μ : P -> Δ ist die Ausgabefunktion

Die Moore-Maschine unterscheidet sich also von der Mealy-Maschine dadurch, dass ihre Ausgabe nicht von der Eingabe abhängt. Jeder DEA kann als Spezialfall einer Moore-Maschine aufgefasst werden. Es ist zu beachten, dass eine Moore-Maschine immer (auch bei leerem Input) das Zeichen μ(p0) schreibt.

### Definition 35

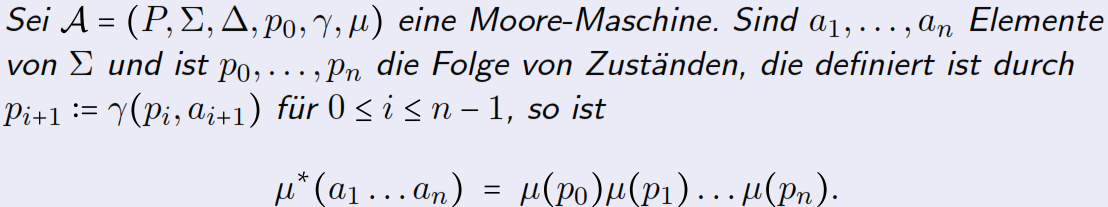
Gegeben die Moore-Maschine A = (P, Σ, Δ, p0, γ, μ). Wir definieren μ\* : Σ\* -> Δ\*:

μ\*(ε) := μ(pp)

μ\*(ua) := μ\*(u)μ(γ\*(p0, ua))

Der erste Teil bestimmt also induktiv das Ausgabewort von u und über das Gamma wird der aktuelle Zustand bestimmt, zu dem dann die letzte Ausgabe angehängt wird. Wir lesen/schreiben also wie üblich von links nach rechts.

### Lemma 36



Geht aus vorheriger Definition hervor.

### Definition 37 (Äquivalenz von Mealy- und Moore-Maschinen)

Gegeben eine Mealy-Maschine A = (Q, Σ, Δ, q0, δ, λ) und eine Moore-Maschine B = (P, Σ, Δ, p0, γ, μ). Die Maschinen A und B heissen genau dann **äquivalent**, wenn für alle Eingabewörter die (bis auf das Startsymbol der Moore-Maschine) jeweils gleiche Ausgabe produziert wird:

μ(p0)λ\*(q0, u) = μ\*(u)

Beachte hier, dass μ(p0) vor die Mealy-Maschine angefügt werden muss, da die Moore-Maschine immer das erste Zeichen für den ersten Zustand ausgibt, unabhängig von der Eingabe.

### Theorem 38

Für jede Mealy-Maschine A gibt es eine dazu äquivalente Moore-Maschine B.

Für diese Konstruktion weisen wir dem Startzustand ein beliebiges Zeichen aus dem Ausgabealphabet zu. Dann definieren wir γ((q, b), a) := (δ(q, a), λ(q, a)). Die so entstandene Transitionsfunktion bildet also einen Zustand der Mealy-Maschine und eine Eingabe a auf das Paar ab, das aus dem Folgezustand der Mealy-Maschine und dem Zeichen, das auf dem Weg der Mealy-Maschine ausgegeben worden wäre ab. Dann definieren wir μ((q, b)) := b. Ist ein Zustand (q, b) erreicht, wird durch μ das Zeichen b ausgegeben.

Kurz gefasst: Wir gehen von einem Zustand (q, b) mit der Eingabe a in den Zustand über, den die Mealy-Maschine mit q und a erreicht und ordnen diesem Zustand noch das Symbol zu, dass die Mealy-Maschine bei diesem Übergang ausgeben würde (δ(q, a), λ(q, a))

### Theorem 39

Für jede Moore-Maschine A gibt es eine dazu äquivalente Mealy-Maschine B.

Wir haben die Moore-Maschine A = (P, Σ, Δ, p0, γ, μ). Dann setzen wir B = (P, Σ, Δ, p0, γ, λ), wobei wir λ definieren durch: λ(p, a) := μ(γ(p, a)). Also ausgegeben wird das Symbol, das sonst erst beim Zustand ausgegeben würde («auf dem Weg statt beim Zustand»).

Die Äquivalenz lässt sich leicht über die Induktion nach dem Aufbau der Wörter zeigen.

# Kellerautomaten

## Informelle Beschreibung

Bei DEA-Problemen wie 0i1i nicht DEA-akzeptierbar ist das Problem, dass ein DEA kein unendliches Gedächtnis hat. Bei einem endlichen Automaten können ja nur Informationen via Zustände gespeichert werden und da diese Anzahl endlich ist können wir auch nur endlich viele Informationen speichern.

Deshalb wird nun ein **Kellerspeicher** hinzugefügt. Dieser kann einen Stapel (Stack) beliebiger Höhe aufnehmen. Dabei wird nach dem LIFO-Prinzip (Last-In-First-Out) gehandelt. Informationen werden also oben auf den Stapel gelegt und auch von oben vom Stapel genommen.