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.

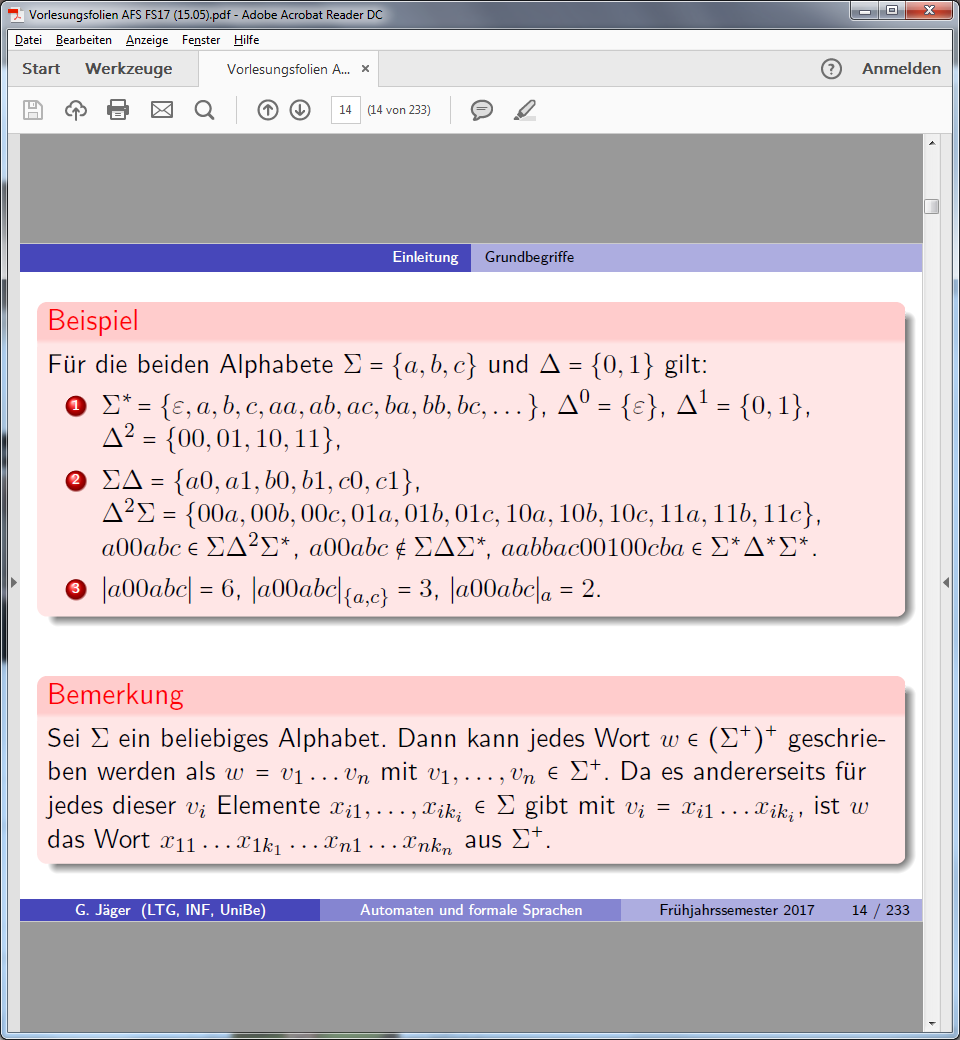
#### Bemerkung



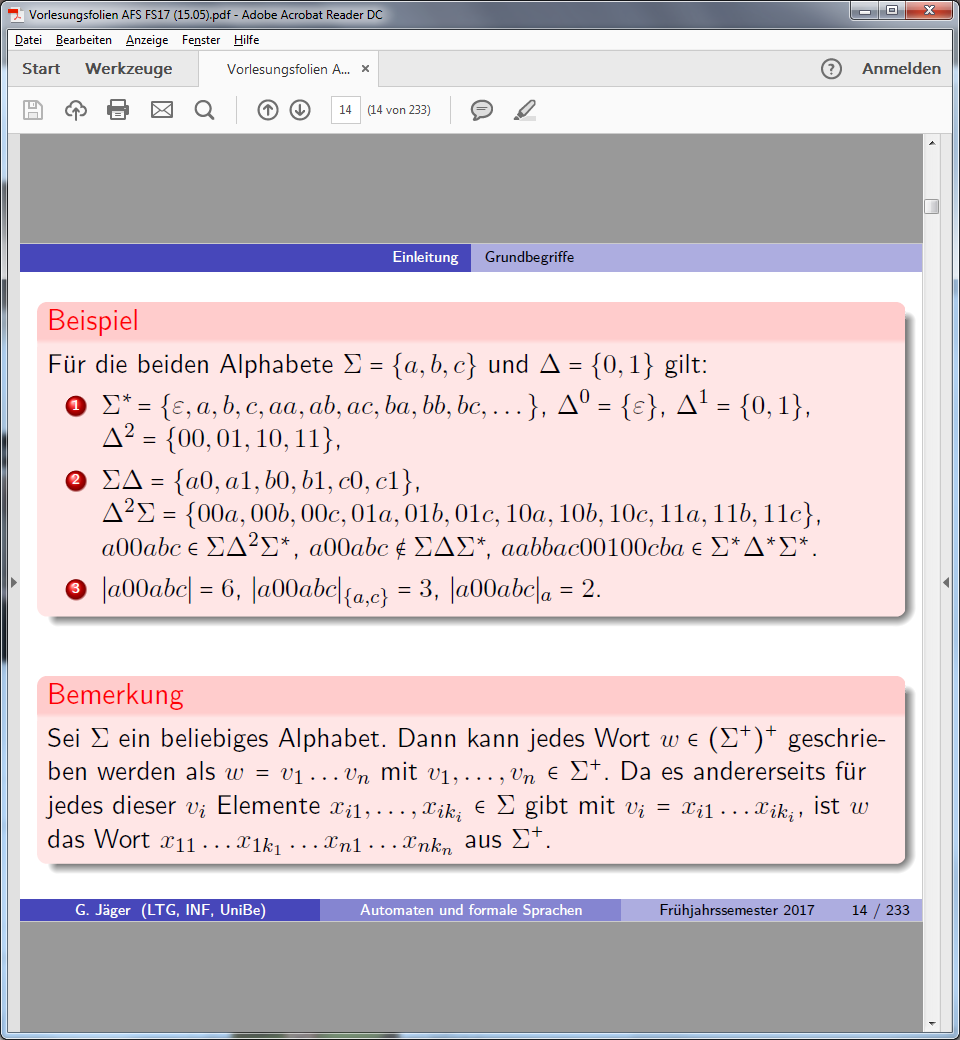
### Definition 5

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.

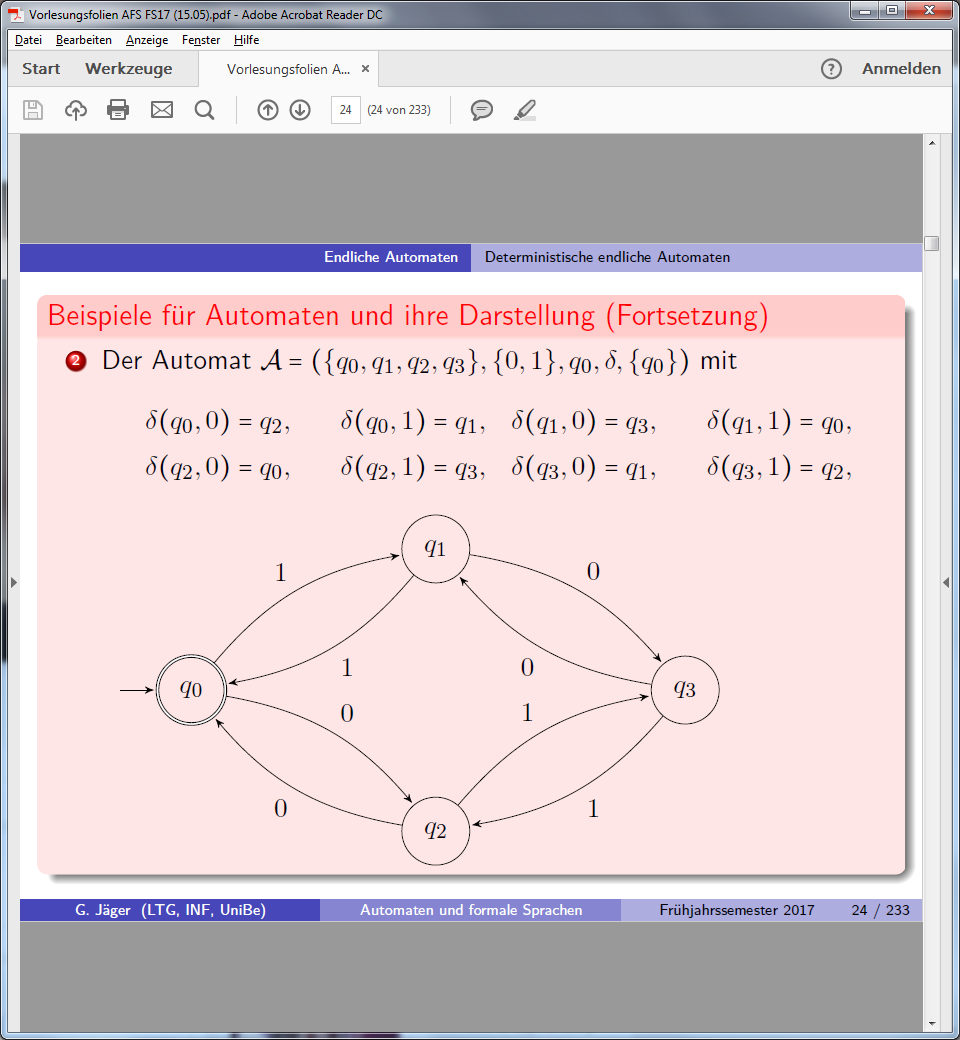
## Deterministische endliche Automaten

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

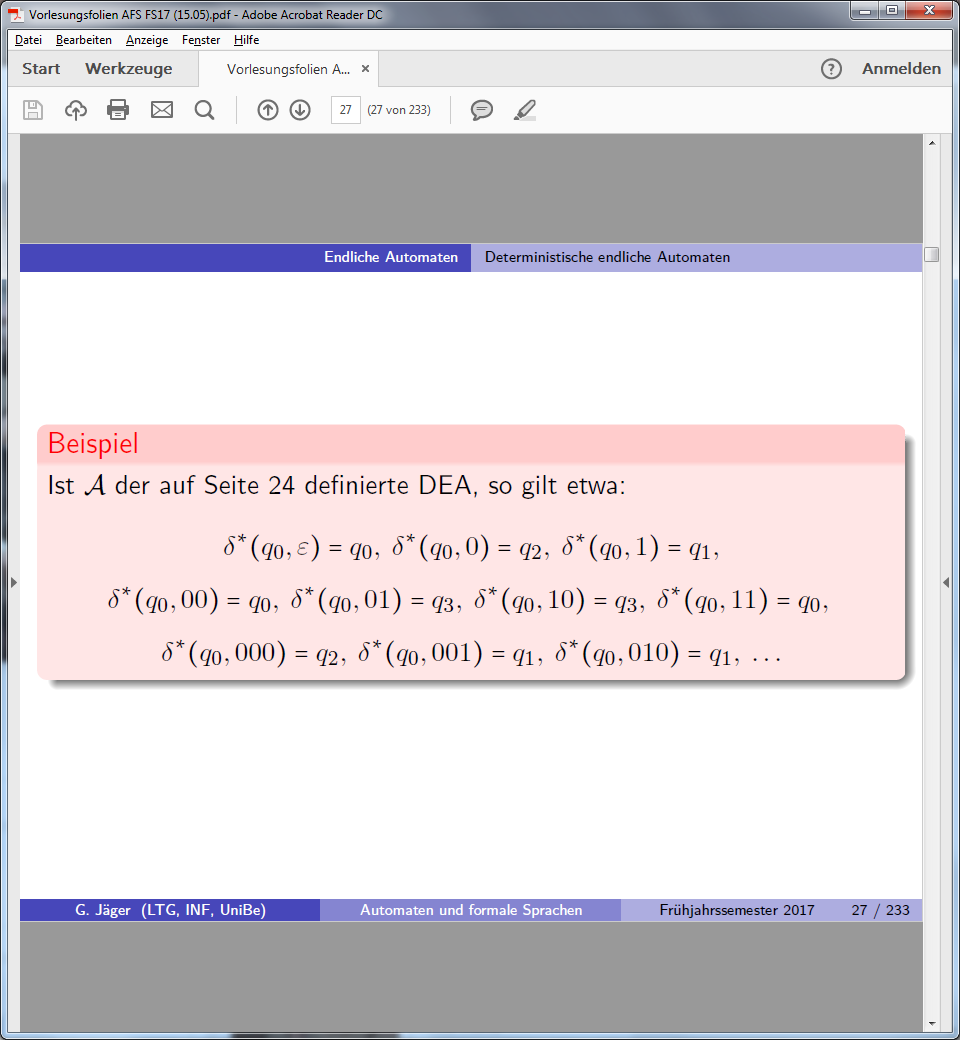
Sei A= (Q, , q0, ,F) ein DEA.

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



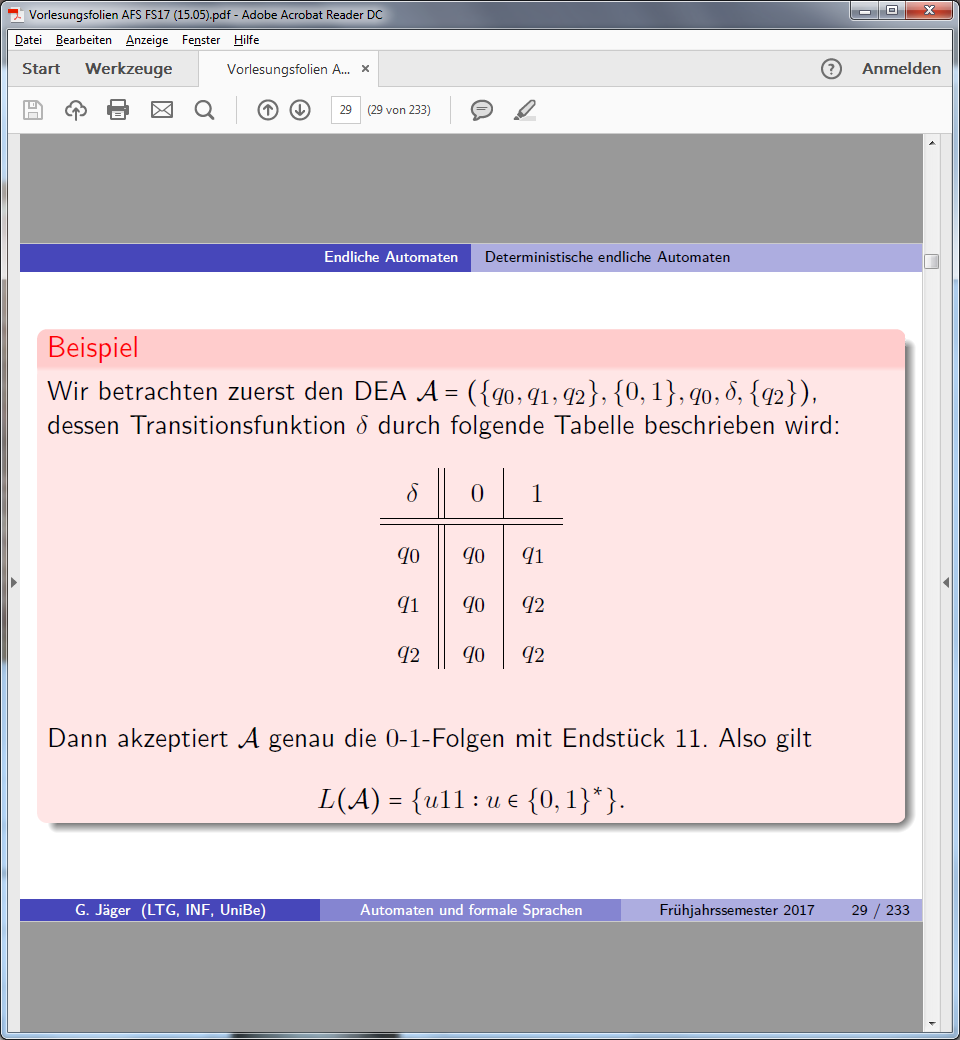
### Definition 11

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.

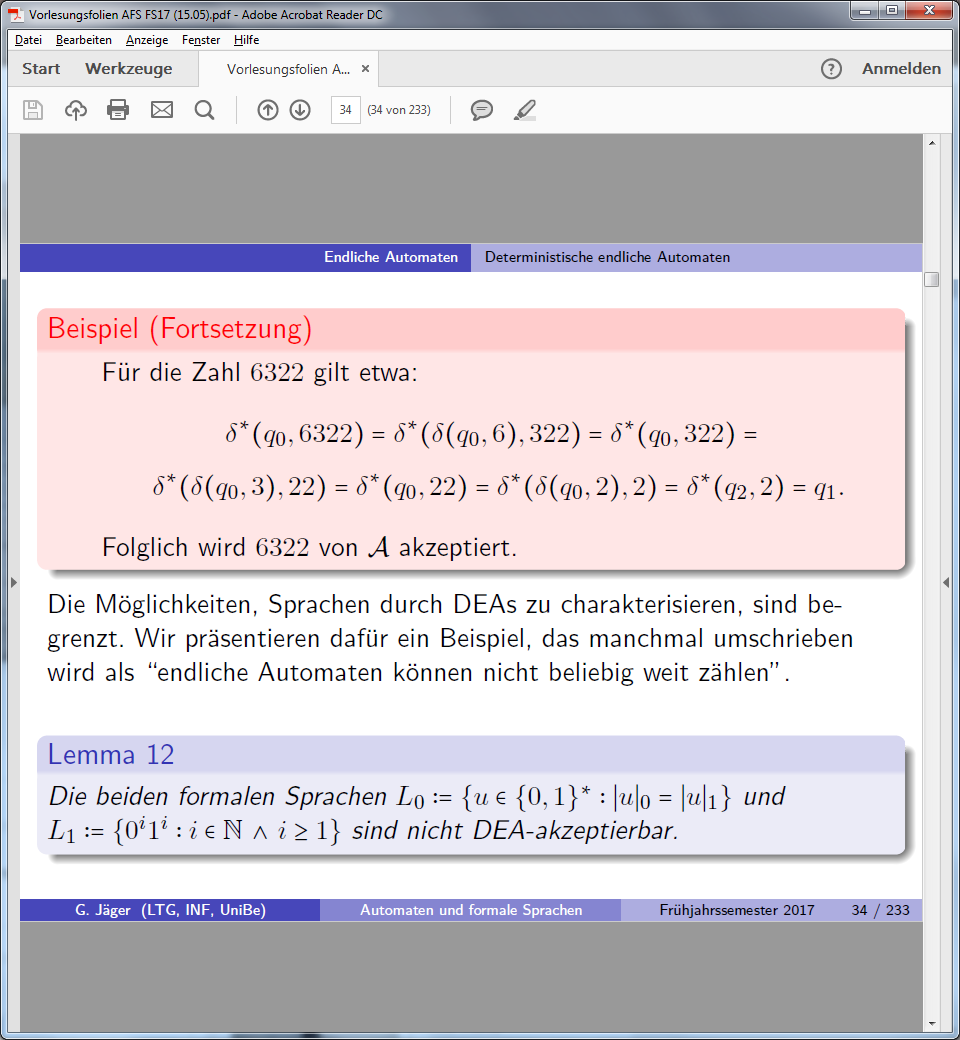
Die von einem DEA A **akzeptierte Sprache L(A)** ist definiert als alle Wörter, die der DEA akzeptiert.

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

#### Beispiel



### Lemma 12



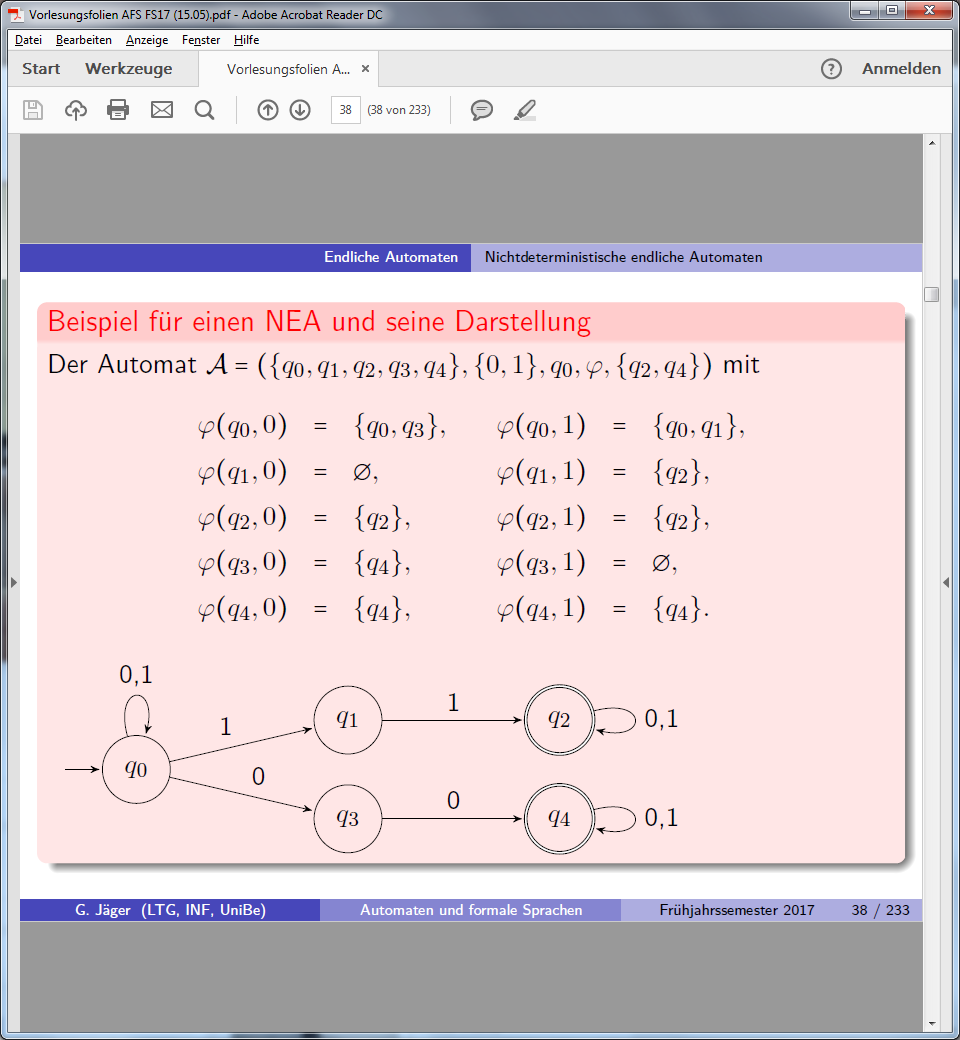
## Nichtdeterministische endliche Automaten

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

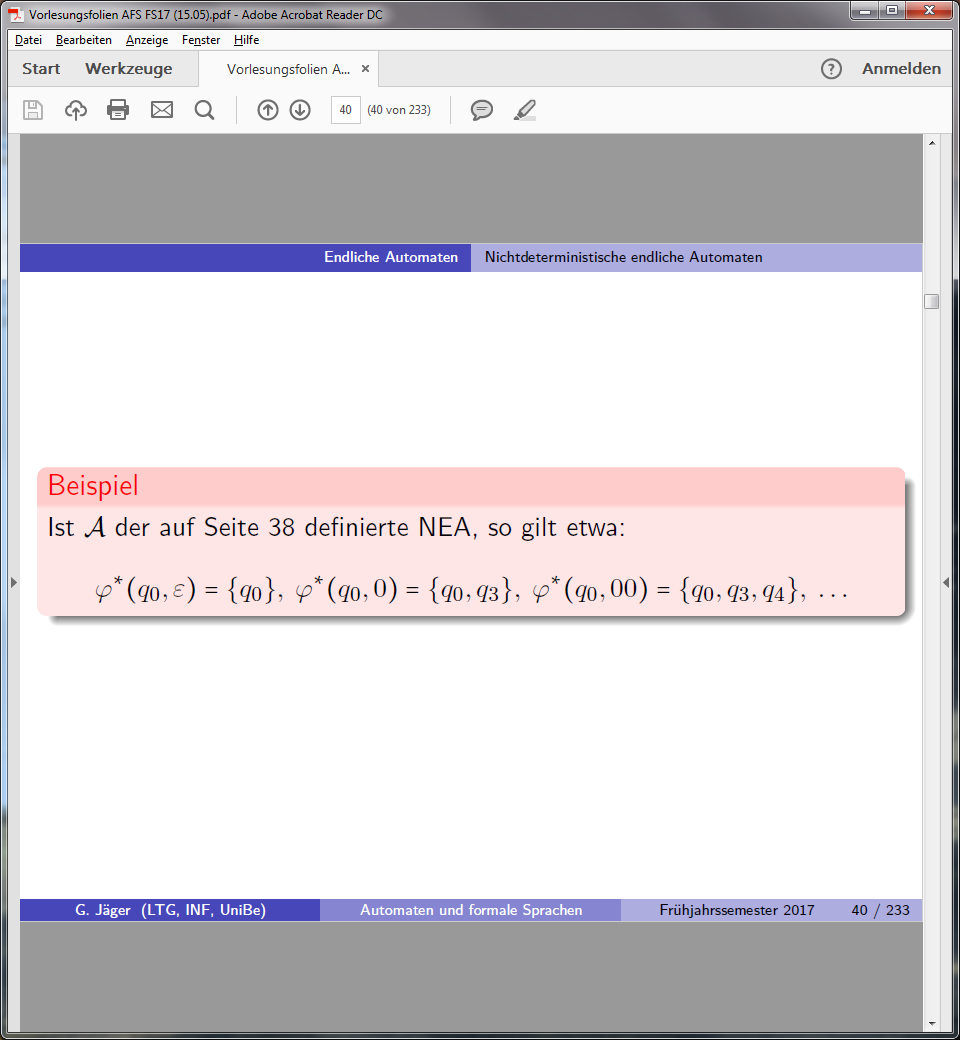
Wir erweitern nun die Transitionsfunktion induktiv auf Wörter, ähnlich wie beim DEA.

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

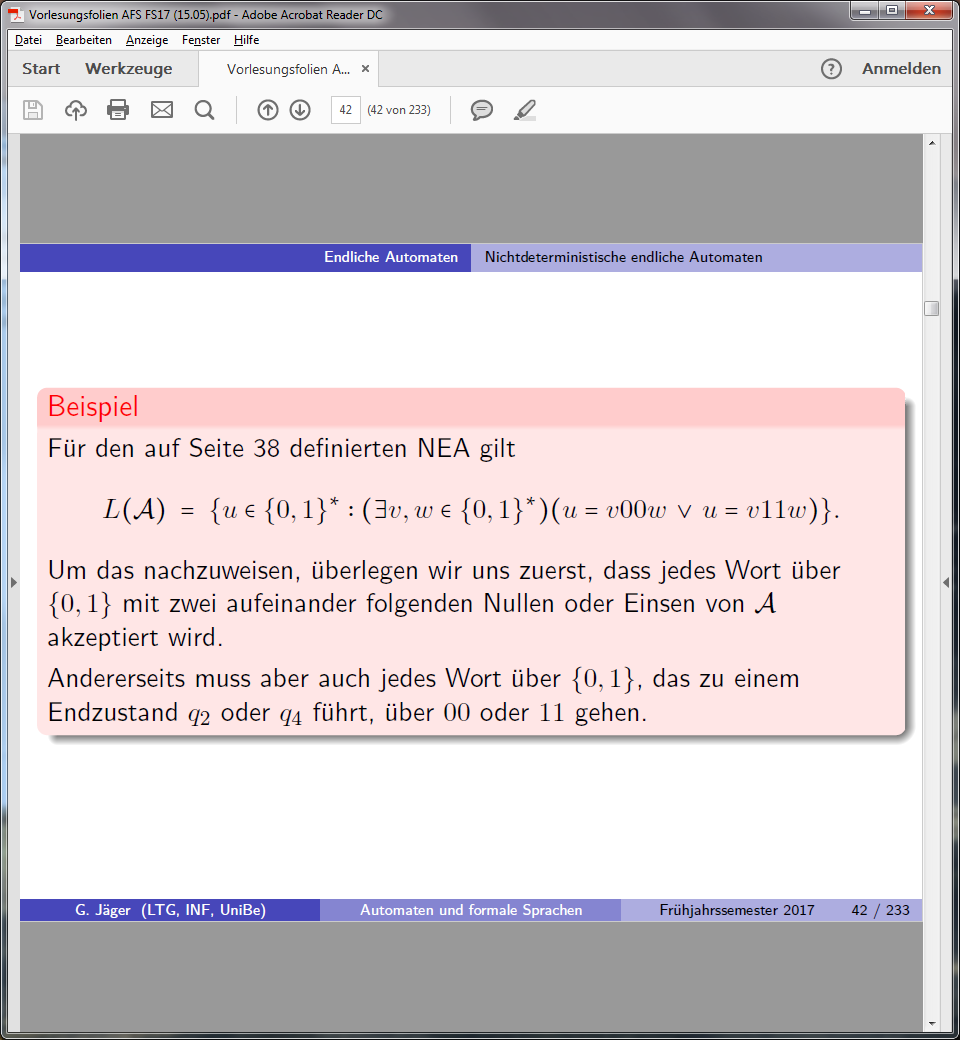
Sei A = (Q, , q0, ,F) ein NEA. Dann sagen wir, dass A das Wort u aus \* genau dann **akzeptiert**, wenn \*(q0, u) F . Also wenn einer der Zustände aus der Zustandsmenge, die nach u erreichbar ist ein Endzustand ist.

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.

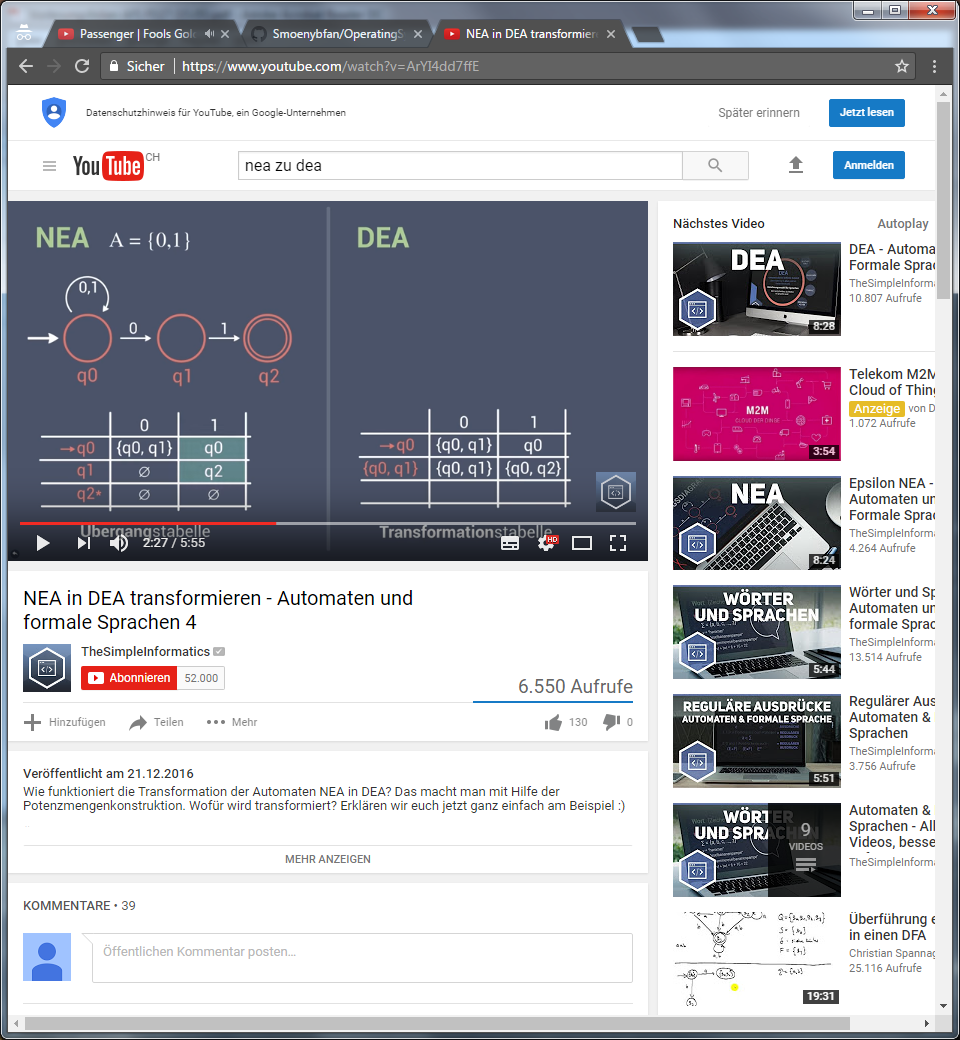
Sei A := (Q, , q0, ,F) ein beliebiger DEA. Dann definieren wir : Q x -> p(Q), (q, a) :={}, so ist B := (Q, , q0, ,F) ein NEA, für den gilt \*(q, u) = {\*(q, u)}. Weiter gilt L(A) = L(B).

### 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.

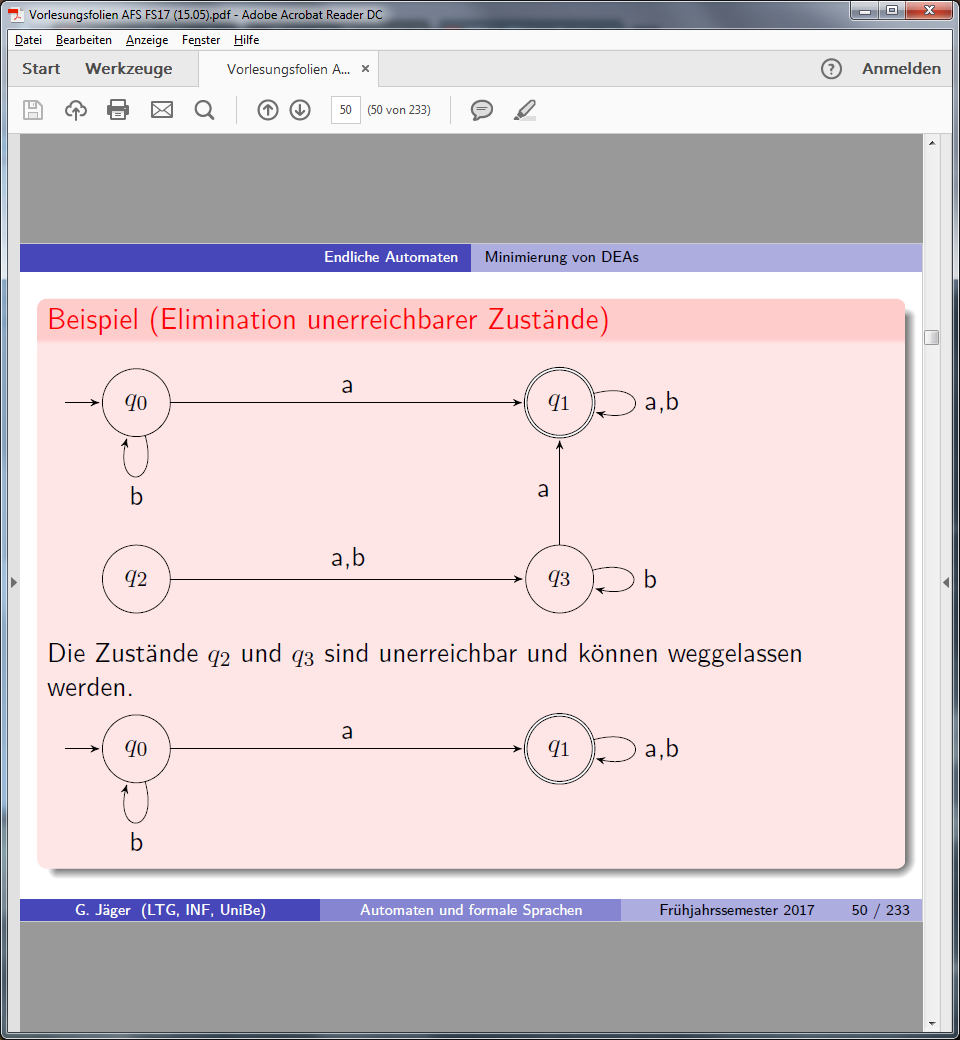
## Minimierung von DEAs

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

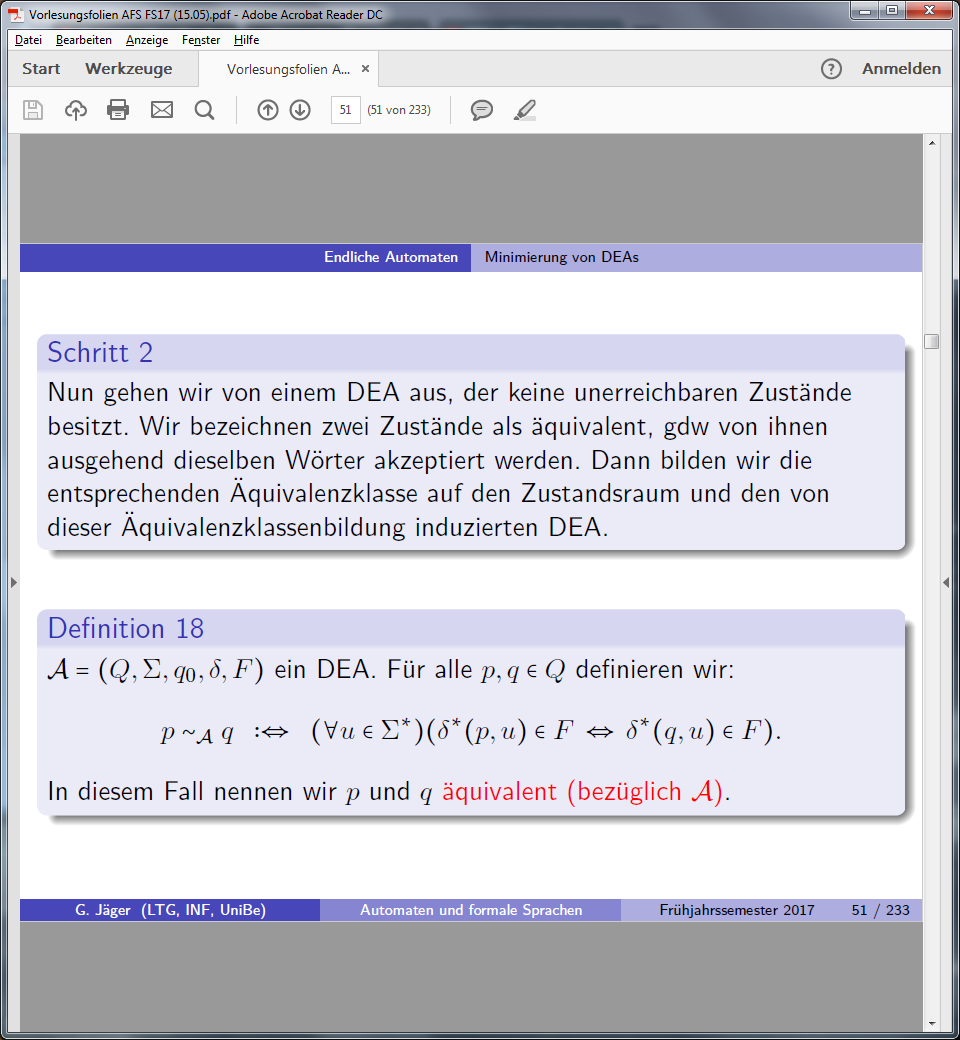


### Schritt 2

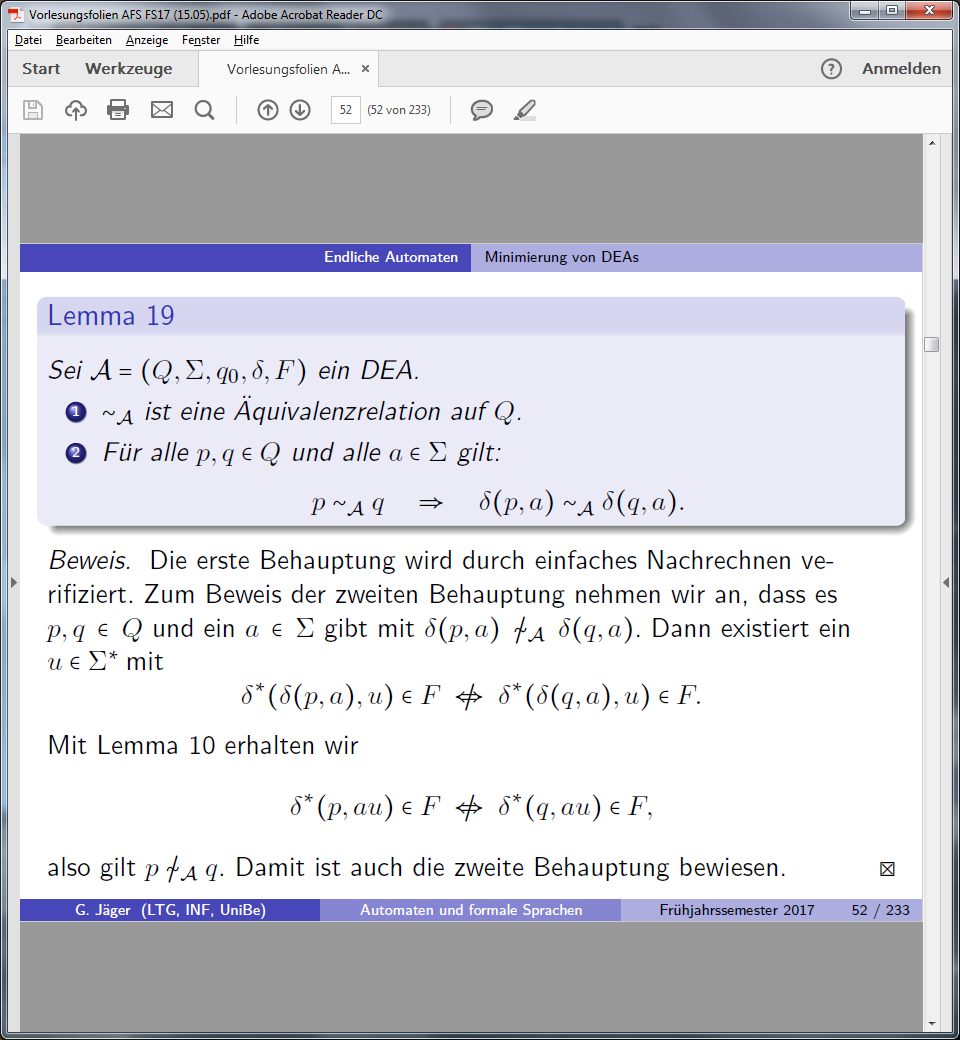
Wir gehen nun von einem DEA aus, der keine unerreichbaren Zustände mehr besitzt. Wir bezeichnen zwei Zustände genau dann als äquivalent, wenn von ihnen ausgehend dieselben Wörter akzeptiert werden. Wir können dann die Äquivalenzklassen zu jeweils einem Zustand zusammenfassen.

### Definition 18

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

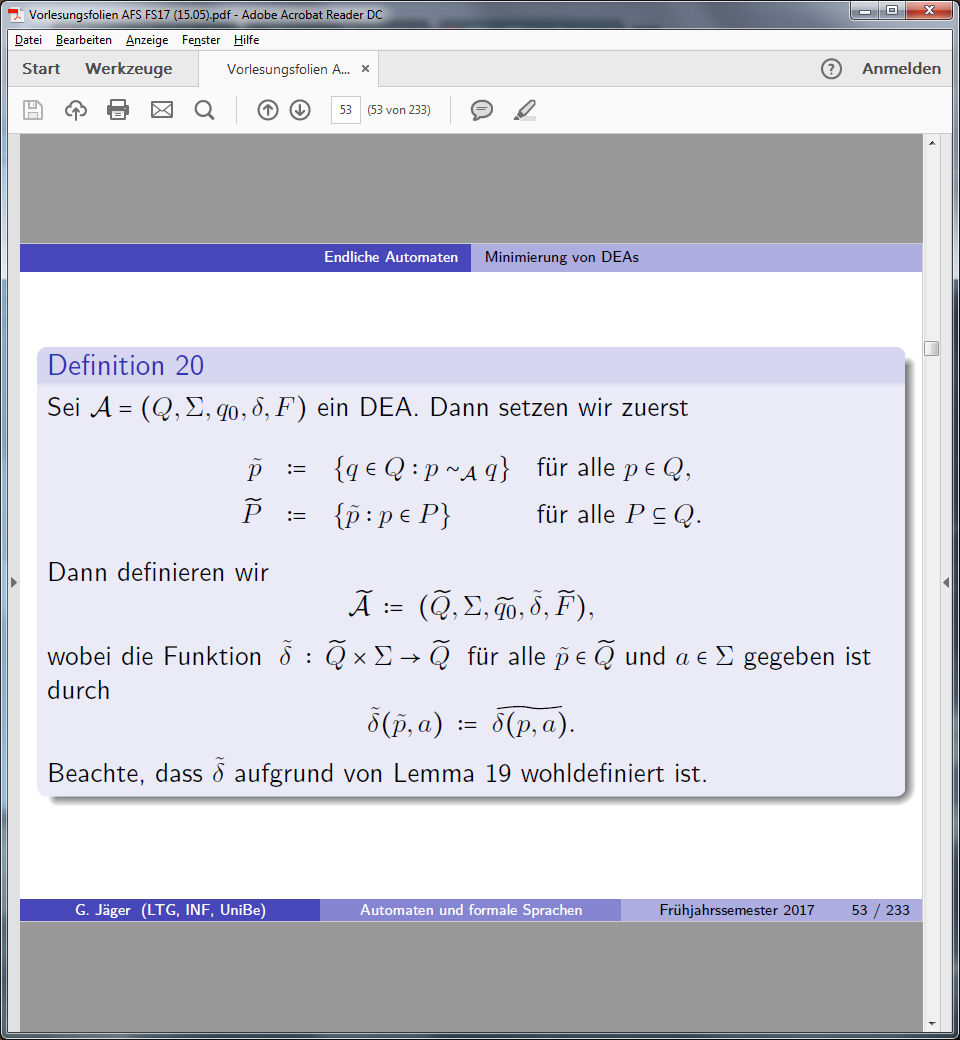


### Lemma 19



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.

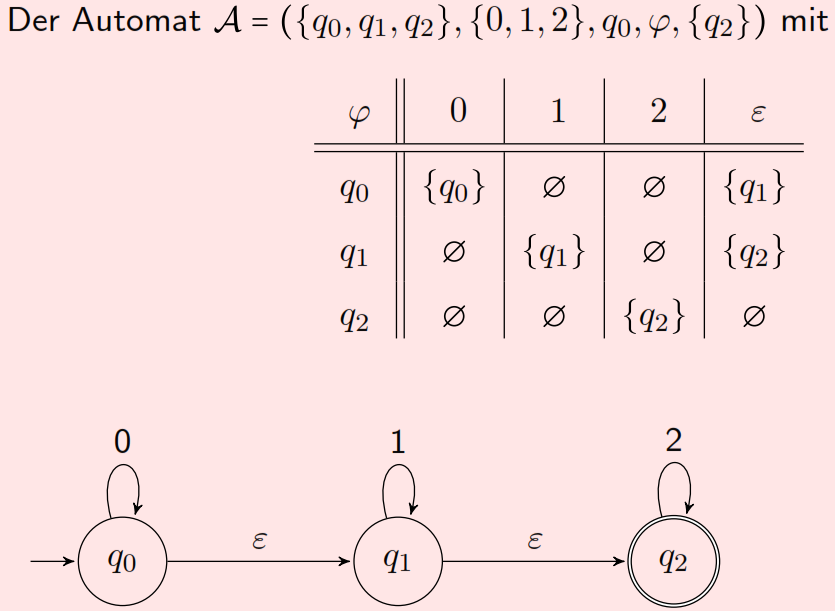
## -Automaten

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

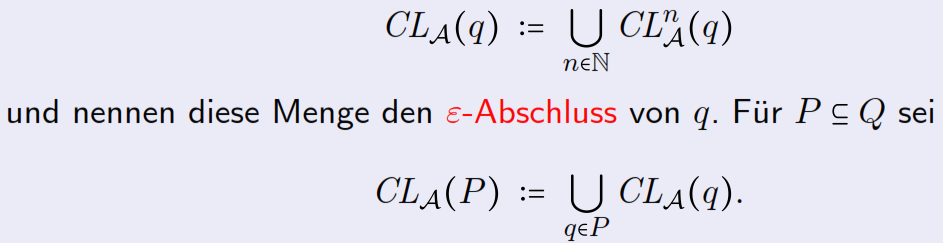
Ein **ε-Automat** ist ein 5-Tupel(Q, , q0, ,F), bei dem Q, Σ, q0 wie in der Definition der NEA definiert sind. Der Unterschied liegt hier bei der Transitionsfunktion, für die auch das leere Wort ε eine mögliche Eingabe ist: ϕ : Q x (Σ {ε}) -> p(Q).

#### Beispiel



### Definition 25

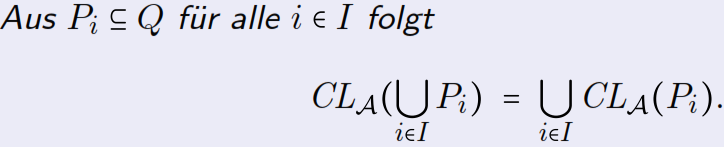
Wir führen nun für jeden ε-Automaten A und jeden seiner Zustände q den **ε-Abschluss CLA(q)** ein. Dabei handelt es sich um die Menge aller Zustände, die von q aus durch einen Pfad erreicht werden können, der nur mit ε markiert ist/nur ε als Eingabe hat.



### 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

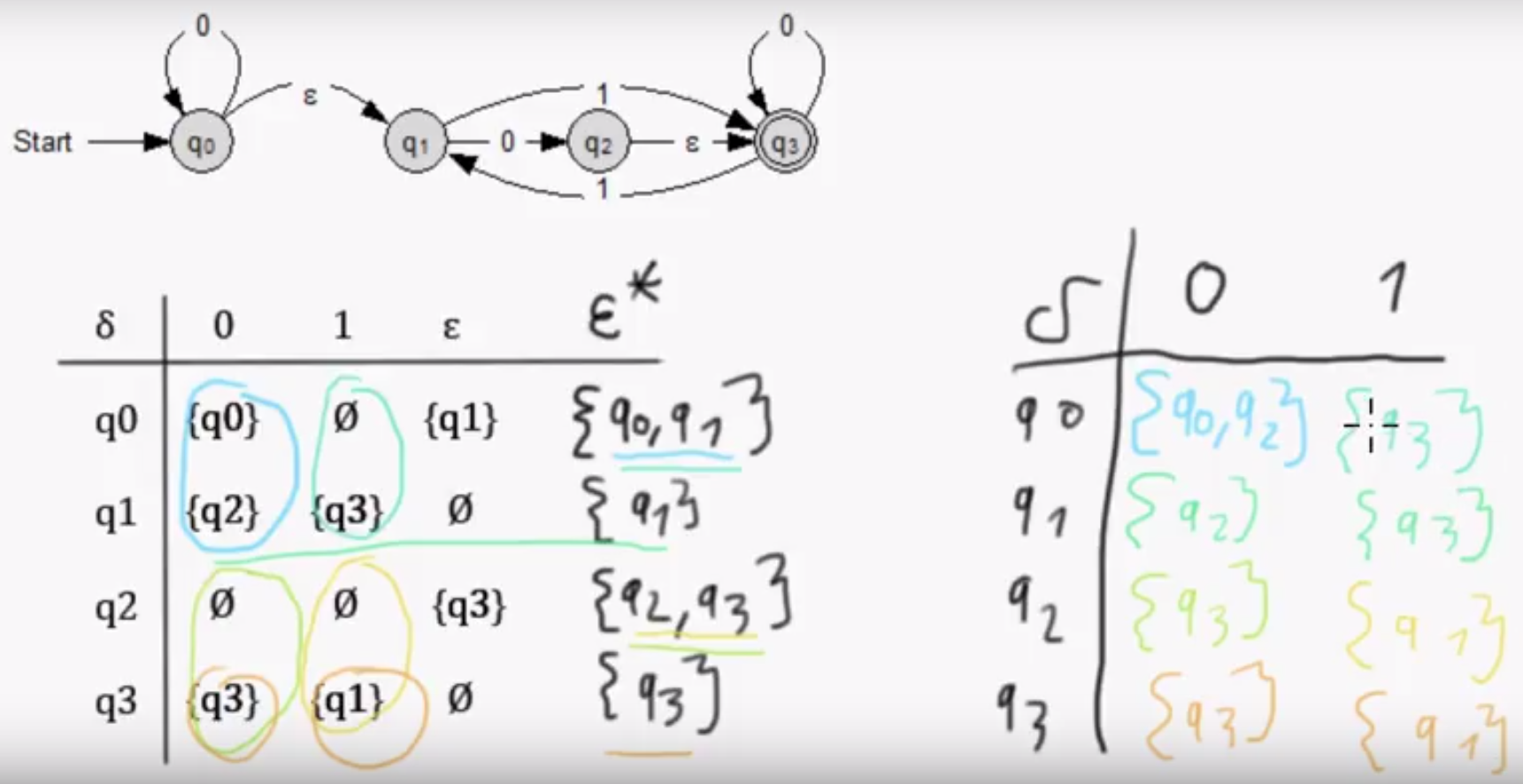
Ist A =(Q, , q0, ,F) ein ε-Automat, so gilt für alle q aus Q, u aus Σ\* und a aus Σ, dass

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).

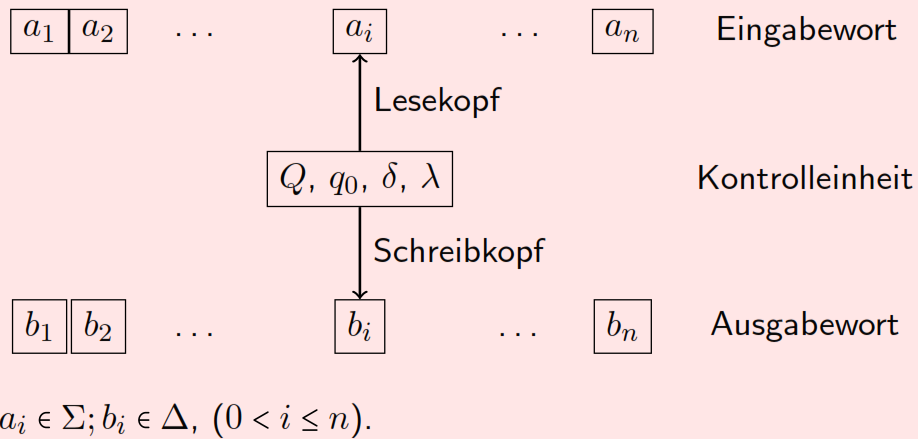
Um diesen zu konstruieren, betrachten wir wohin wir von einem bestimmten Zustand mit ε gelangen. Wenn wir zB von q0 mit ε nach q1 gelangen ist unsere Menge {q0, q1}. Nun verknüpfen wir für unseren ursprünglichen die Zustandsmengen dieser Mengen für ein Eingabesymbol. Wenn wir also beim ε-Automaten mit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

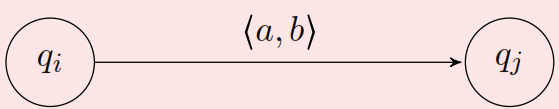
### Definition 31 (Mealy-Maschine)

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)

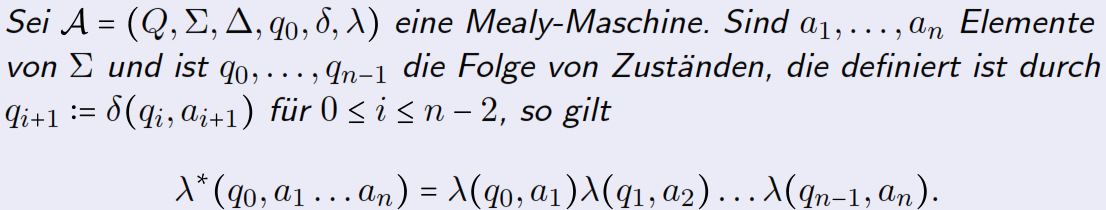
Gegeben die Mealy-Maschine A = (Q, Σ, Δ, q0, δ, λ). Wir definieren nun λ\* : Q x Σ\* -> Δ\*:

λ\*(q, ε) := ε

λ\*(q, ua) := λ\*(q,u)λ(δ\*(q,u),a)

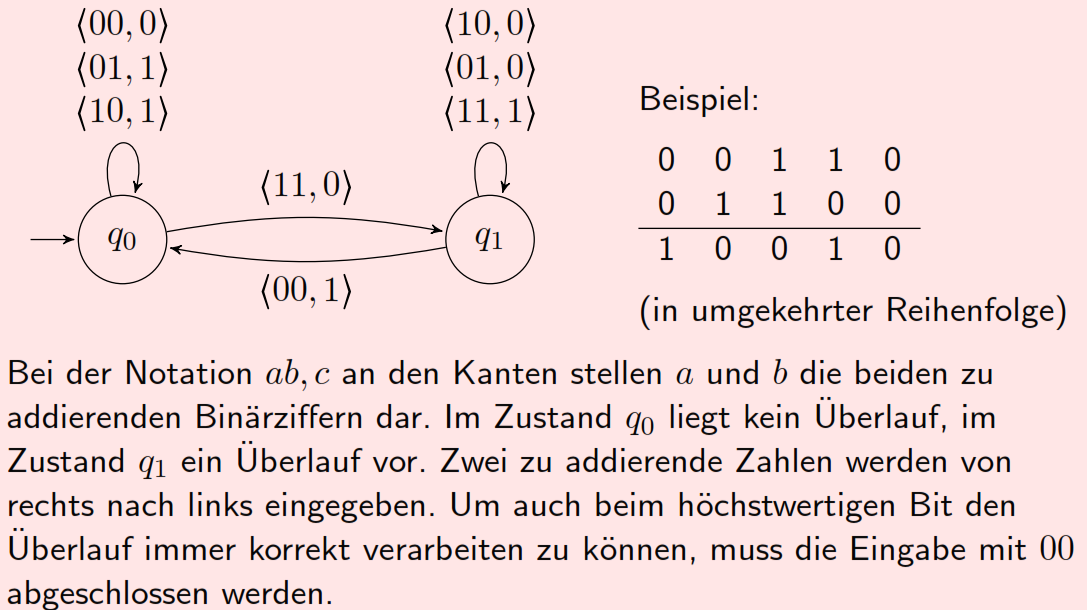
Wir arbeiten also wieder von links nach rechts Symbol für Symbol ab.

### 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

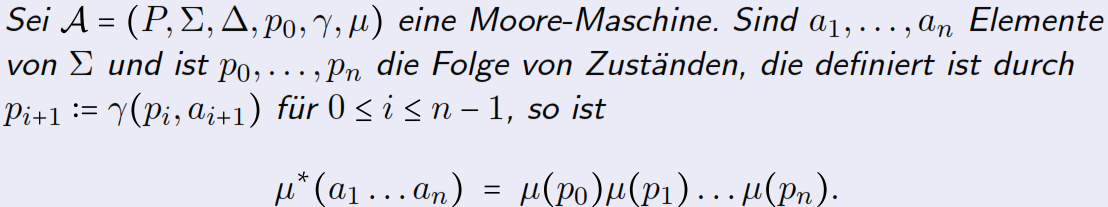
Gegeben die Moore-Maschine A = (P, Σ, Δ, p0, γ, μ). Wir definieren μ\* : Σ\* -> Δ\*:

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

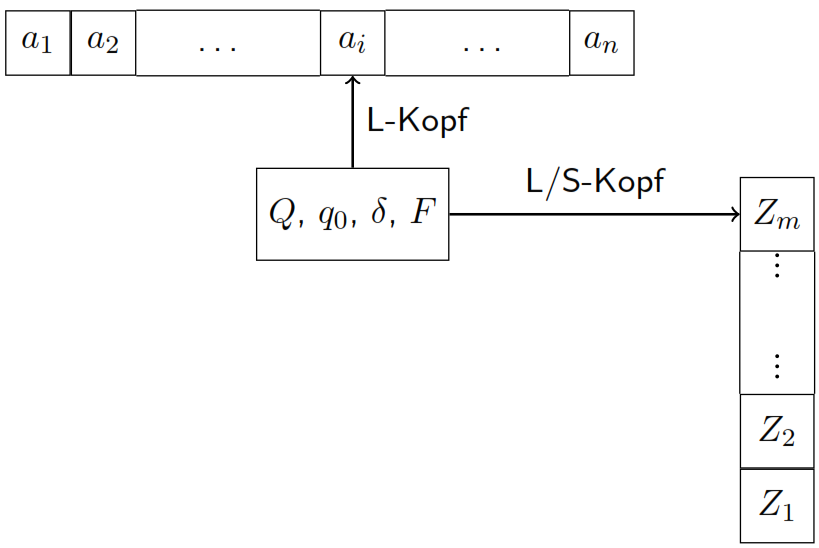
Bei DEA-Problemen wie 0i1i nicht DEA-akzeptierbar ist das Problem, dass ein DEA kein unendliches Gedächtnis hat. Bei einem endlichen Automaten können ja nur Informationen via Zustände gespeichert werden und da diese Anzahl endlich ist können wir auch nur endlich viele Informationen speichern.

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.

## Formale Einführung

### Definition 40

Ein **Kellerautomat KA** ist ein 7-Tupel A = (Q, Σ, Γ, q0, Z0, δ, F), wobei gilt:  
Q ist die Zustandsmenge,  
Σ ist das Eingabealphabet,  
Γ ist eine nicht-leere endliche Menge, das **Kelleralphabet**,  
q0 ist der Anfangszustand,  
Z0 ist das **Startsymbol** für den Keller,  
δ : Q x (Σ {ε}) x Γ -> pf(Q x Γ\*) ist die **Transitionsfunktion**, wobei pf(Q x Γ\*) die Menge der endlichen Teilmengen von Q x Γ\* ist.   
F ist die Menge der Endzustände



Der Keller verfügt also über unbegrenzte Speicherkapazität. Die Kontrolleinheit kann verschiedene Zustände annehmen und hat einen Lese-Kopf für die Eingabe, sowie einen Lese- und Schreibkopf für den Keller. Dabei kann die Kontrolleinheit nur nach rechts bewegt werden und nur auf das oberste Kellersymbol zugreifen.

Zu Beginn steht der L-Kopf auf a1, im Keller befindet sich nur Z0, der L/S-Kopf zeigt auf Z0 und der KA ist im Zustand q0.

Nun gibt es zwei mögliche Verarbeitungsschritte. Im ersten liest der KA das Zeichen unter dem L-Kopf. Abhängig von diesem Zeichen, dem Zeichen unter dem L/S-Kopf und dem aktuellen Zustand geht der KA in einen neuen Zustand über und ersetzt das Zeichen unter dem L/S-Kopf durch ein Wort bestehend aus Symbolen aus dem Kelleralphabet. Dafür kann es mehrere, aber auch keine Möglichkeiten geben. Danach bewegt sich der L-Kopf einen Schritt nach rechts und der L/S-Kopf zeigt auf das oberste Kellersymbol.

Im zweiten Fall liest der L-Kopf kein Zeichen (ε) und der KA geht in Abhängigkeit von ε, dem Zeichen unter dem L/S-Kopf und dem aktuellen Zustand in einen neuen Zustand über. Wieder kann das Zeichen unter dem L/S-Kopf durch ein Wort über dem Kelleralphabet ersetzt werden. Danach bewegt sich der L/S-Kopf auf das oberste Symbol des Kellers, der L-Kopf wird nicht bewegt.

Im Folgenden wird diese informelle Beschreibung formalisiert.

### Definition 41 (Konfiguration)

Sei A = (Q, Σ, Γ, q0, Z0, δ, F) ein KA. Ein Tripel (q, u, γ) (Q x Σ\* x Γ\*) nennen wir eine **Konfiguration** von A.

Die aktuelle Konfiguration (q, u, γ) eines KA gibt den gegenwärtigen Zustand q, das noch nicht gelesene Endstück u des Eingabewortes sowie den aktuellen Kellerinhalt an. Dabei ist bei γ = Z1…Zn Z1 das oberste Kellerzeichen, wo auch der L/S-Kopf steht.

### Definition 42

Sei A = (Q, Σ, Γ, q0, Z0, δ, F) ein KA.

Ist k = (q, au, Zα) eine Konfiguration, so heisst die Konfiguration k’ = (p, u, βα) genau dann **direkter Nachfolger** von k bzgl. A, wenn (p, β) δ(q, a, Z), dafür schreiben wir k A k’.

Die Relation k \*A ist der reflexive und transitive Abschluss von A. Für Konfigurationen k und k’ gilt   
k \*A k’ genau dann, wenn es eine Folge von Konfigurationen gibt mit k0 = k, kn= k’ und ki  ki+1. Also eine Kette von direkten Nachfolgern von k nach k’.

A **akzeptiert** genau dann das Wort u, wenn (q0, u, Z0) \*A (p, ε, γ) gilt, wobei p F. Also wenn es eine Kette direkter Nachfolger vom Startzustand in den Endzustand gibt und das ganze Wort abgearbeitet wurde. Ob danach der Endzustand mit ε-Übergängen wieder verlassen wird spielt keine Rolle, genauso was im Keller steht.

Die von A **akezptierte Sprache L(A)** ist die Menge der akzeptierten Wörter.

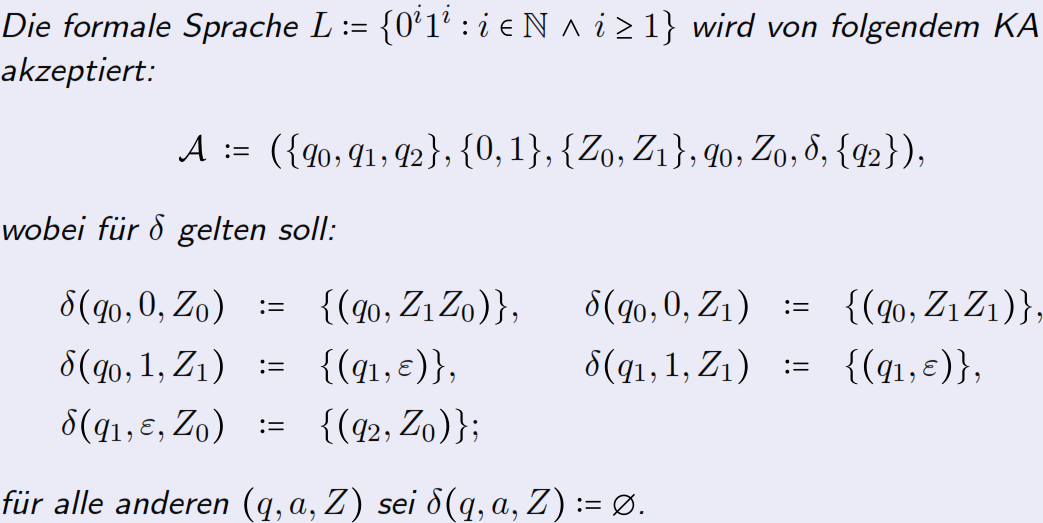
Die von A **durch leeren Keller akzeptierten Sprache Lε(A)** ist die Menge der Wörter, bei denen am Ende der Keller leer ist, also der Abschluss in (p, ε, ε) endet. Der Zustand p spielt hier keine Rolle.

### Theorem 43



Dies besagt, dass das Akzeptieren durch Endzustände und das Akzeptieren durch leeren Keller äquivalent sind.

### Lemma 44



### Lemma 45

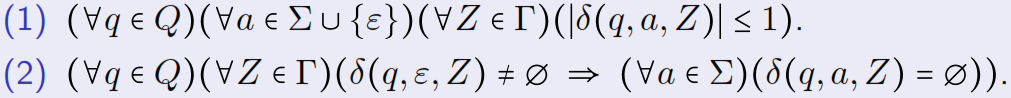
Ist die Sprache L DEA-Akzeptierbar, so gibt es einen KA B für den L = L(B) gilt.

## Deterministische Kellerautomaten

Sei A = (Q, Σ, Γ, q0, Z0, δ, F) ein beliebiger KA. Dann hat in der Regel eine Konfiguration (q, au, Zα) mehrere direkte Nachfolger. Daher sind KA in der Regel nichtdeterministisch. Wir führen nun aber die Teilklasse der KAs ein, bei denen jede Konfiguration höchstens einen direkten Nachfolger hat und sprechen dann von deterministischen Kellerautomaten.

### Definition 46 (Deterministischer Kellerautomat)

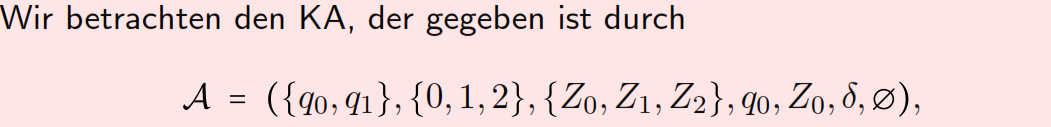
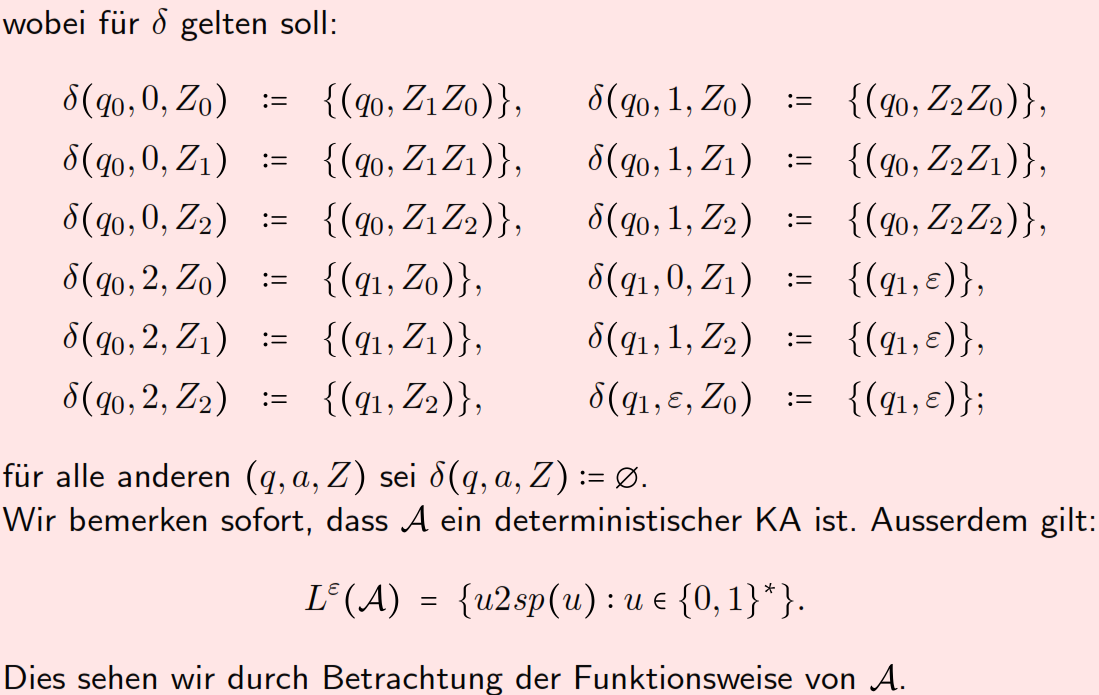
Ein KA A = (Q, Σ, Γ, q0, Z0, δ, F) heisst genau dann **deterministisch**, wenn er maximal einen direkten Nachfolger hat und wenn gilt, dass ein Zustand entweder einen ε-Übergang oder einen Zeichen-Übergang hat.



#### Bemerkung

Jede DEA-akzeptierte Sprache wird auch von einem deterministischen KA akzeptiert.

#### Beispiel

# Turingmaschinen

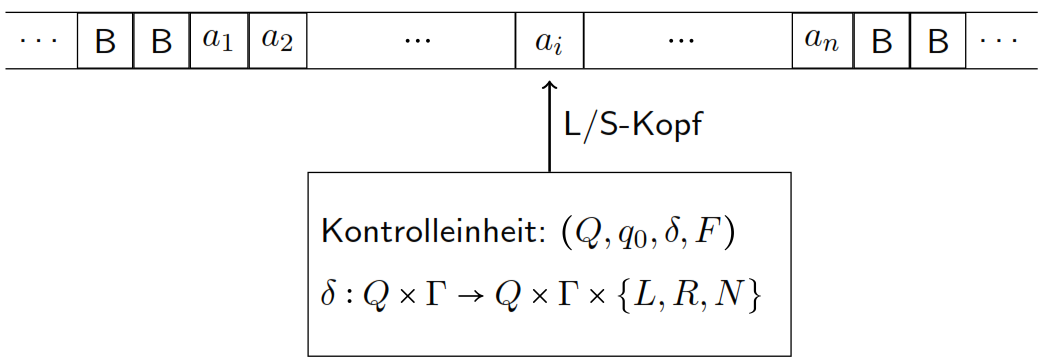
## Definition und Beschreibung

Der wesentliche neue Aspekt, der bei Turingmaschinen zum Tragen kommt, ist die Verwendung eines unendlichen Bandes, das in Felder eingeteilt ist. Jedes Feld kann ein einzelnes Zeichen des Arbeitsalphabets der Maschine enthalten. Auf dem Band bewegt sich ein Lese- und Schreibkopf. Eine Turingmaschine ist ein sehr allgemeiner Typ eines Automaten.

### Definition 47

Eine **Turingmaschine TM** ist ein 7-Tupel M = (Q, Σ, Γ, q0, B, δ, F), wobei gilt:  
Q ist die Zustandsmenge,  
Σ ist das Eingabealphabet,  
Γ Σ ist eine nicht leere Menge, das **Arbeitsalphabet**,  
q0 ist der Anfangszustand,  
B aus Γ \ Σ ist das **Blank**,  
δ: Q x Γ -> Q x Γ x {L, R, N} ist die **Transitionsfunktion**,  
F ist die Menge der Endzustände

Wir nehmen dabei an, dass Q und Γ disjunkte Mengen sind.



Eine TM umfasst ein beidseitig unendliches Band, das in Felder eingeteilt ist. Jedes Feld enthält genau ein Zeichen des Arbeitsalphabets. Die TM verfügt über einen Schreib- und Lesekopf, der sich immer über genau einem Feld befindet.

Die TM arbeitet in diskreten Zeitschritten. Zu jedem Zeitpunkt liest der L/S-Kopf das Zeichen des Feldes ein, über dem er steht und druck ein Zeichen des Arbeitsalphabets. Danach bewegt er sich um höchsten eine Stelle nach links oder rechts. Das ist durch die Transitionsfunktion δ bestimmt.

δ(q, a) = (p, b, x) wird dabei wie folgt umgesetzt: Wenn sich die TM im Zustand q befindet, und unter dem L/S-Kopf das Zeichen a steht, so geht die TM im nächsten Schritt in den Zustand p über, schreibt an die Stelle von a das Zeichen b auf das Band und bewegt danach den L/S-Kopf in Richtung x {L, R, N}, wobei L für links, R für Rechts und N für neutral (stehenbleiben) steht.

Ist δ(q, a) nicht definiert, hält die TM im Zustand q.

Eine TM wird im Startzustand q0 gestartet, die Eingabe u Σ\* steht auf dem Band und der L/S-Kopf befindet sich über dem ersten Zeichen von u, falls u ≠ ε und über B, falls u = ε. Alle anderen Felder des Bands sind mit B beschrieben.

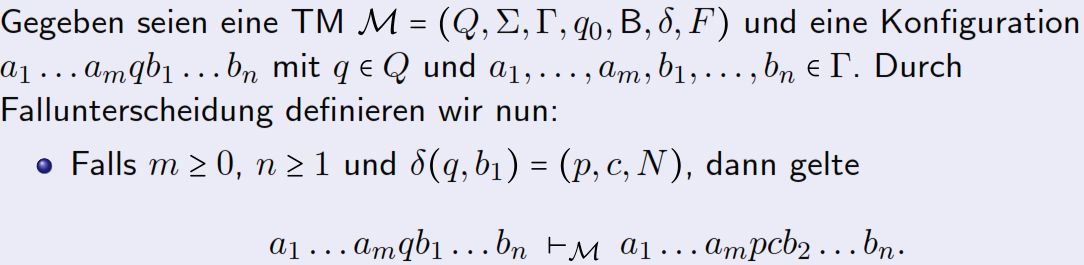
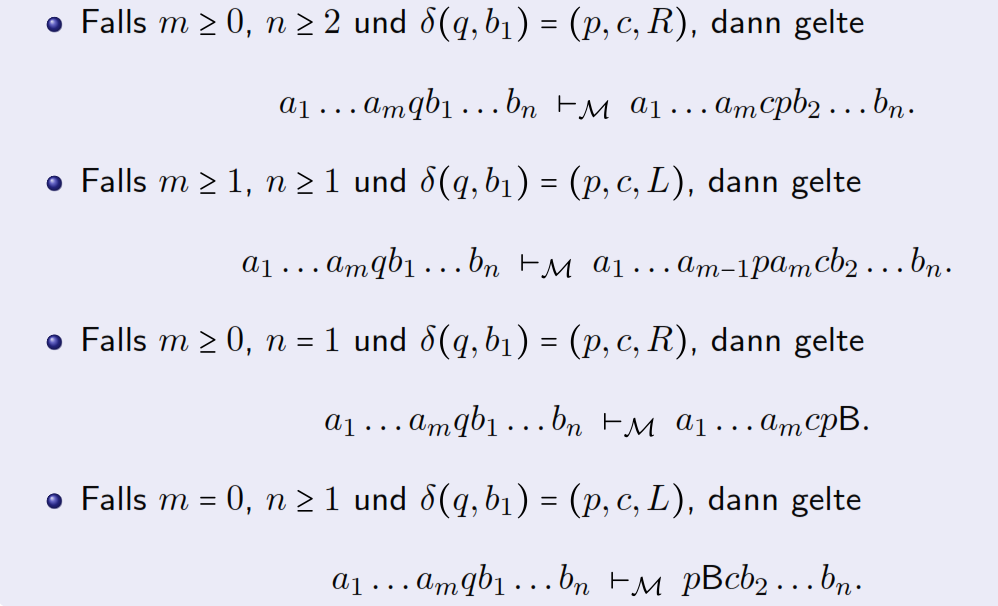
Wir werden diese informelle Beschreibung nun durch eine formelle Definition abrunden.

### Definition 48 (Konfiguration)

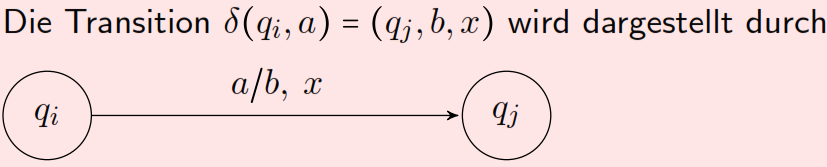
Sei M = (Q, Σ, Γ, q0, B, δ, F) eine TM. Ein Wort uqv Γ\*QΓ+ nennen wir eine **Konfiguration** von M.

Das bedeutet, dass sich die TM im Zustand q befindet. Das Wort uv Γ+ steht auf (|u| + |v|) aufeinander folgenden Feldern des Bands und entspricht dem schon besuchten Teil des Bands. Der L/S-Kopf befindet sich über dem ersten Symbol von v.

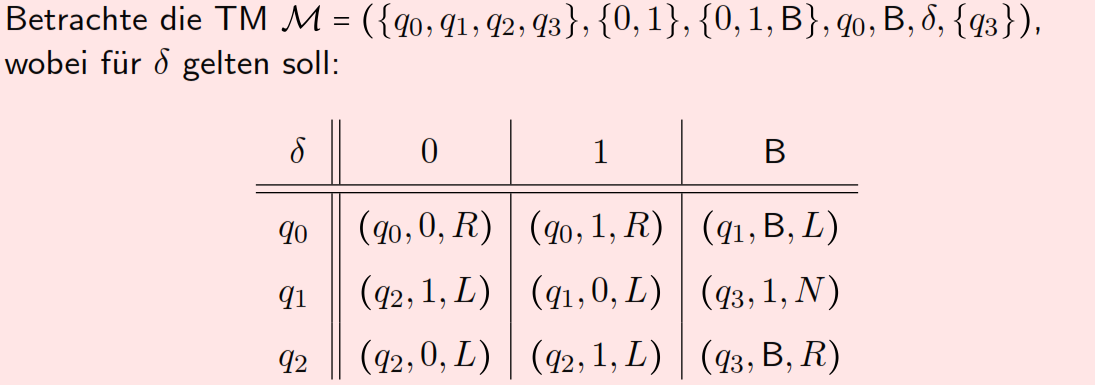
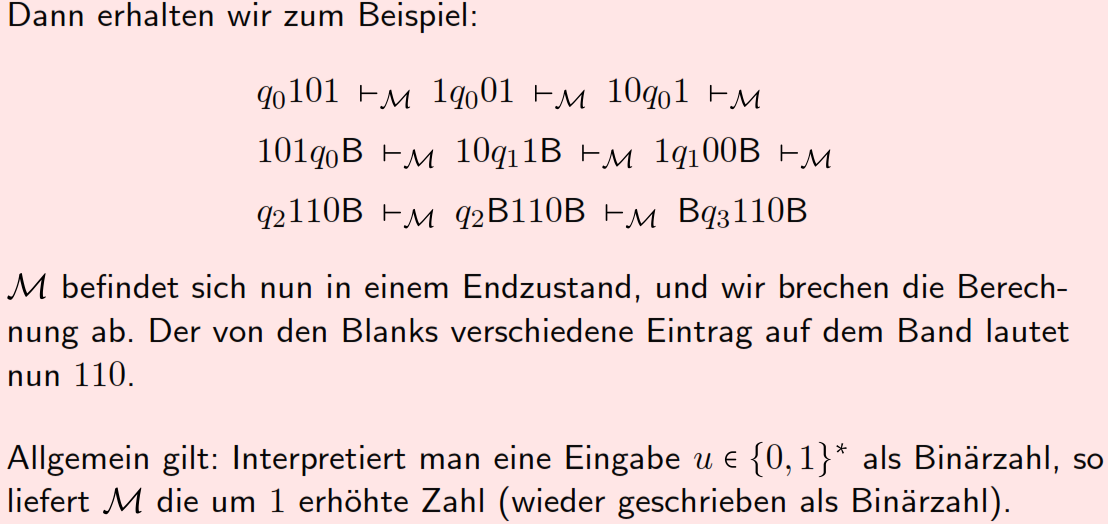
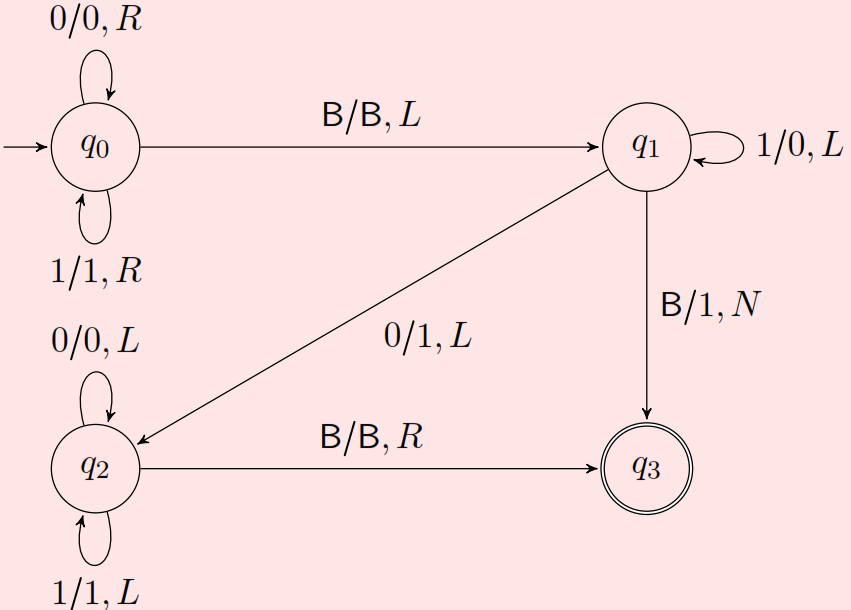
### Definition 49

### Graphische Darstellung



#### Beispiel

## TM-Akzeptierbarkeit und TM-Entscheidbarkeit

Wir gehen nun anlog zum Fall der KAs vor. Wir führen wieder für jede TM M den reflexiven und transitiven Abschluss ein. Dann definieren wir, dass ein Wort u über dem Eingabealphabet genau dann von M akzeptiert wird, falls man von der Eingabekonfiguration q0ub im Sinne von \*M zu einem Endzustand gelangen kann.

### Definition 50

Die Relation \*M ist der reflexive und transitive Abschluss von M. Für Konfigurationen k und k’ bzgl. M gilt also k \*M k’ genau dann, wenn es eine Folge von Konfiguration für die k0=k, kn=k’ und   
ki M ki+1 gilt. Also wenn es eine Kette von direkten Nachfolgern von k nach k’ gibt.

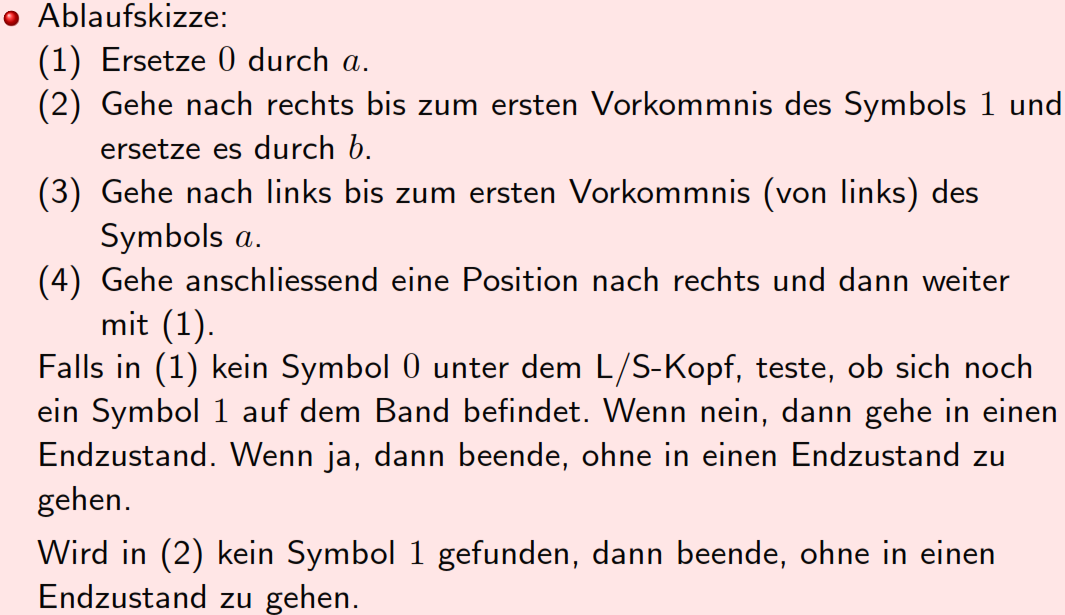
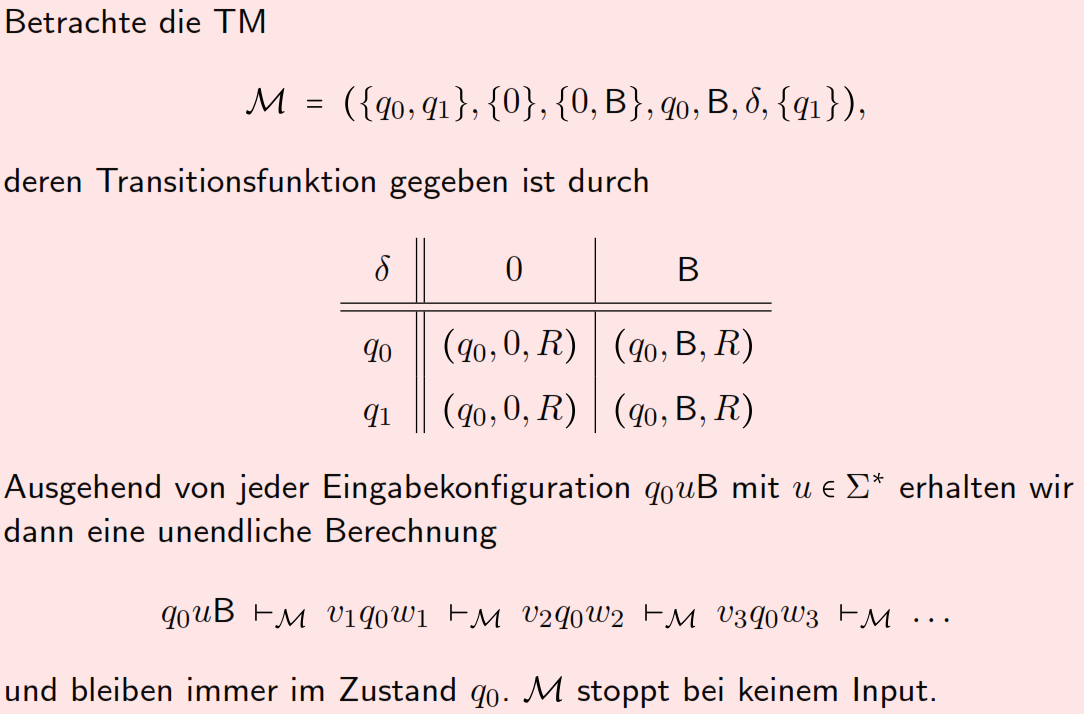
M **akzeptiert** das Wort u genau dann, wenn es ein q0uB \*M vpw (v Γ\*, w Γ+, p F) gibt. Also wenn es eine Kette von direkten Nachfolgern gibt, so dass die TM nach der Abarbeitung der Konfiguration q0uB in einem Endzustand landet.

Die von M **akzeptierte Sprache L(M)** ist also die Menge der akzeptierten Wörter.

Eine Sprache L heisst genau dann **Turing-akzeptierbar**, wenn es eine TM M gibt mit L = L(M).

Ein Wort kann nicht akzeptiert werden, weil die TM auf dem Weg zu einem Übergang δ(q, a) gelangt, der nicht definiert ist oder wenn sie in einer Endlosschlaufe von Konfigurationen landet, wobei kein Zustand ein Endzustand ist.

#### Beispiel

### Definition 51

Sei M = (Q, Σ, Γ, q0, B, δ, F) eine TM. Eine Konfiguration k Γ\*QΓ+ heisst genau dann **Stoppkonfiguration** von M, wenn es keine Konfiguration k’ von M mit k M k’ gibt. Also wenn es keinen direkten Nachfolger gibt.

Eine Sprache L heisst genau dann **Turing-entscheidbar**, wenn es eine TM M gibt, so dass für jedes Wort u eine Stoppkonfiguration k von M mit q0Ub \*M k existiert und L = L(M) gilt. Also wenn die Turingmaschine für jedes Wort stoppt.

#### Bemerkung

Es gibt Turing-akzeptierbare Sprachen, die nicht Turing-entscheidbar sind.

# Reguläre Sprachen

## Reguläre Ausdrücke

Bisher haben wir formale Sprachen umgangssprachlich beschrieben. Nun werden wir diese noch mathematisch präzise beschreiben.

### Konvention

Im Folgenden sind φ, ◊, +, ● und □ (in den Folien wird hierfür ein Stern verwendet) beliebige, aber fest gewählte Symbole, die zu keinem der betrachteten Alphabete Σ und keiner der entsprechenden Wortmengen Σ\* gehören. Ausserdem nehmen wir an, dass die Klammern () weder zu Σ, noch zu Σ\* gehören.

### Definition 52

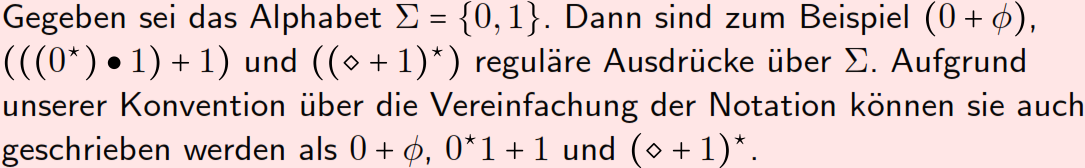
Geben sei ein Alphabet Σ. Die Menge **Reg(Σ) der regulären Ausrücken** über Σ wird induktiv definiert durch:

φ, ◊ und alle Elemente a aus Σ gehören zu Reg(Σ).

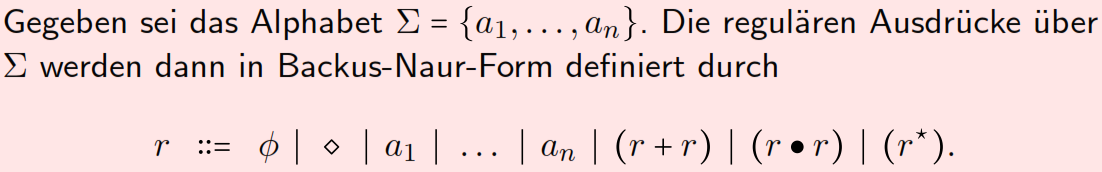
Sind r und s Elemente von Reg(Σ), so gehören auch (r + s), (r ● s) und (r□) zu Reg(Σ).

Um die Notation zu vereinfachen gehen wir davon aus, dass die Operation □ stärker bindet als ●, aber ● stärker bindet als +. Ausserdem schreiben wir häufig nur (rs) statt (r ● s) und lassen die Klammern fort, solange keine Unklarheiten entstehen.

#### Beispiel



#### Bemerkung (Backus-Naur-Form)



### Definition 53

Gegeben ein Alphabet Σ.

Für jeden regulären Ausdruck r Reg(Σ) wird die Sprache L(r) induktiv nach dem Aufbau von r definiert durch:

L():=, L(◊):={ε}, L(a)={a} und

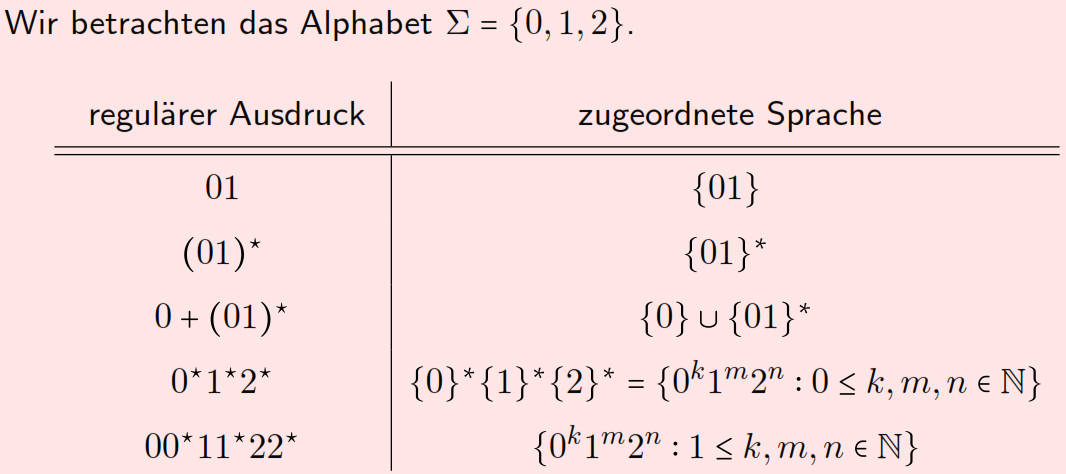
L(s1 + s2) := L(s1) L(s2),

L(s1 ● s2) := L(s1) L(s2),

L(s□) := L(s)\*.

Eine Sprache L über Σ heisst genau dann **regulär**, wenn es einen regulären Ausdruck r Reg(Σ) mit   
L = L(r) gibt.

#### Beispiel



## Der Äquivalenzsatz von Kleene

### Lemma 54

Gegeben sei eine Sprache L über dem Alphabet Σ. Ist L DEA-akzeptierbar, so ist L eine reguläre Sprache über Σ.

### Lemma 55 (schöner ε-Automat)

Ein ε-Automat A = (Q, Σ, q0, ϕ, F) heisst genau dann **schön**, wenn F eine einelementige Menge {f} ist und ϕ(f, x) = für alle x Σ {ε} gilt. Also wenn es nur einen Endzustand gibt und dieser nicht verlassen wird.

### Lemma 56

Für jeden regulären Ausdruck r über einem Alphabet Σ gibt es einen schönen ε-Automaten A   
mit L(r) = L(A).

### Theorem 57

Für jede formale Sprache L über einem Alphabet Σ sind folgende Aussagen äquivalent:

L ist regulär  
L ist DEA-akzeptierbar  
L ist NEA-akzeptierbar  
L ist ε-akzeptierbar

# Grammatiken

## Allgemeine Grundlagen

Im folgenden Abschnitt beschäftigen wir uns mit sogenannten generischen Systemen, bei denen die Elemente einer Sprache aus Grundausdrücken in endlich vielen Schritten durch vorgegebene Regeln erzeugt werden. Derartige Systeme werden durch Grammatiken beschrieben.

### Definition 58

Eine **formale Grammatik (Chomsky-Grammatik)** ist ein 4-Tupel G = (N,T,P,S), wobei gilt:  
N ist das Alphabet der **nichtterminalen Zeichen (Nonterminals),**T ist das Alphabet der **terminalen Zeichen (Terminals)**, wobei gelten muss, dass N und T disjunkt sind  
P ist eine endliche Teilmenge von (N T)+ x (N T)\*, die **Menge der Produktionen**,  
S N ist das **Startsymbol.**

Die Menge N T nennen wir das **Gesamtalphabet** von G

Eine Produktion p = (u, v) P einer Grammatik G wir nun folgendermassen zur Transformation von Wörtern verwendet: Innerhalb eines Wortes wird das Teilwort u durch das Teilwort v ersetzt. Dadurch erhält man ein neues Wort, auf das wieder dieselbe oder eine andere Produktion von G angewendet werden darf. Die rechte Seite der Produktion darf leer sein, während die linke Seite immer mindestens ein Zeichen enthalten muss. Mithilfe von Produktionen lässt sich auf diese Weise ausgehend vom Startsymbol eine Menge von Wörtern erzeugen.

Das Gesamtalphabet ist dabei aufgeteilt in die disjunkten Teilmengen N und T. Die Terminals sind diejenigen Zeichen, die in den Wörtern der Sprache, die durch eine Grammatik auftreten soll, auftreten dürfen. Die Nonterminals sind Hilfssymbole, die typischerweise in den Produktionen vorkommen und für den Aufbau der Sprache verwendet werden.

### Definition 59

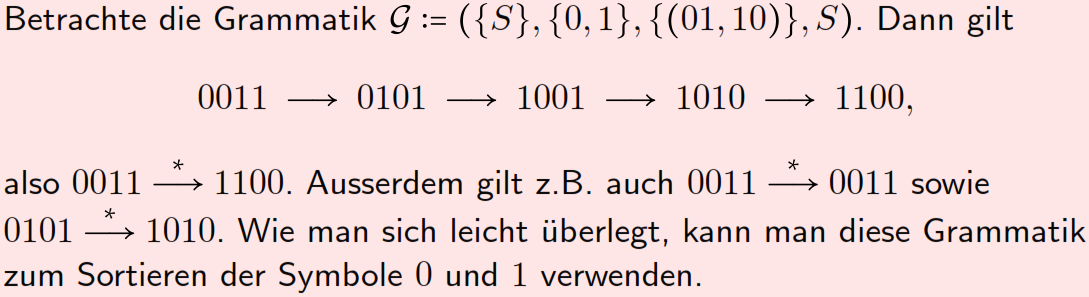
Gegeben sei eine Grammatik G = (N, T, P, S), eine Produktion p = (u, v) sowie Wörter w, w’ (N T)\*.

Das Wort w’ heisst genau dann **unmittelbar ableitbar (in G)** aus w mittels p, wenn es x,y (N T)\* gibt mit w = wux und w’=wvy. Dann schreiben wir G: w w’ oder nur w w’. Also dann, wenn ein Teil (u) von w (durch v) ersetzt werden kann und dadurch w’ entsteht.

Das Wort w’ heisst genau dann **ableitbar (in G)**, wenn es eine Folge von Wörtern w0,…,wn und eine Folge von Produktionen p0,…,pn-1 gibt mit w0=w, wn=w’ und wi wi+1. Dann schreiben wir w w’. Also wenn nach mehreren Ersetzungen durch Produktionen w in w’ umgewandelt werden kann.

Ist bei einer unmittelbaren Ableitung w w’ die Produktion p irrelevant oder klar aus dem Kontext schreiben wir oft nur w w’.

#### Beispiel



### Definition 60 (von Grammatik erzeugte Sprache)

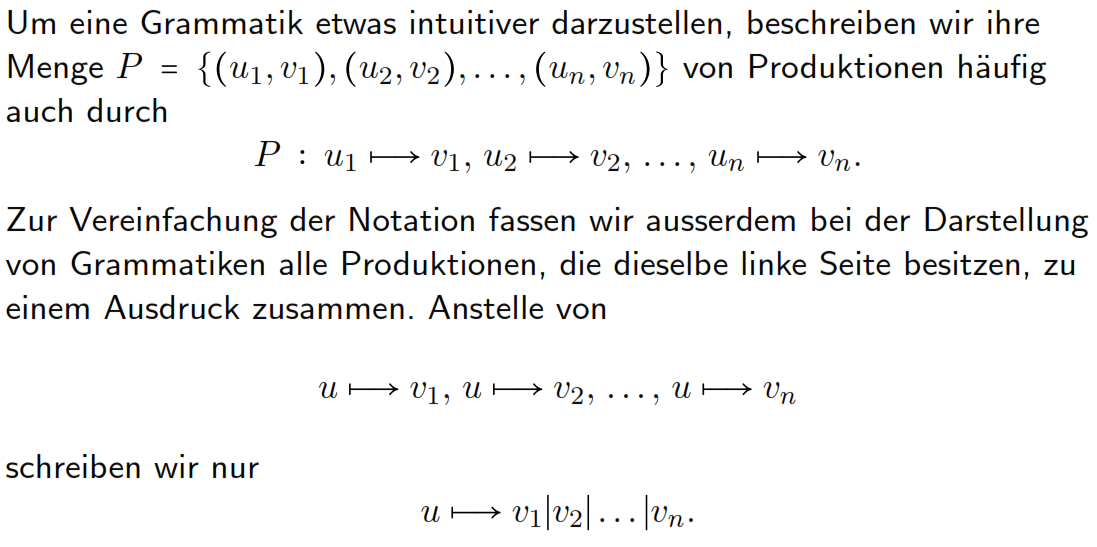
Sei G = (N, T, P, S) eine Grammatik. Die von G **erzeugte Sprache** ist definiert durch die Wörter bestehend ausschliesslich aus Terminalen, die aus dem Startsymbol S ableitbar sind:

L(G) := { w T\* : S w}

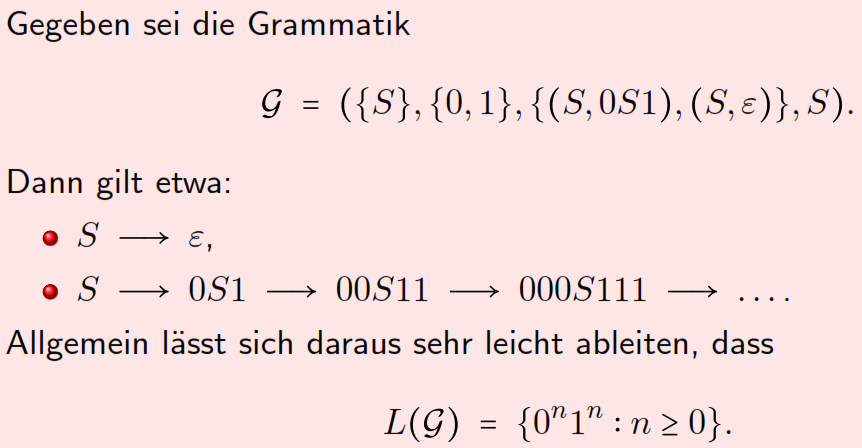
### Lemma 61

Ist G = (N, T, P, S) eine Grammatik, so gibt es endliche Teilmengen N0 N und T0 T, so dass G0 := (N0, T0, P, S) ebenfalls eine Grammatik ist und L(G) = L(G0) gilt.

#### Bemerkung



#### Beispiel (G2)

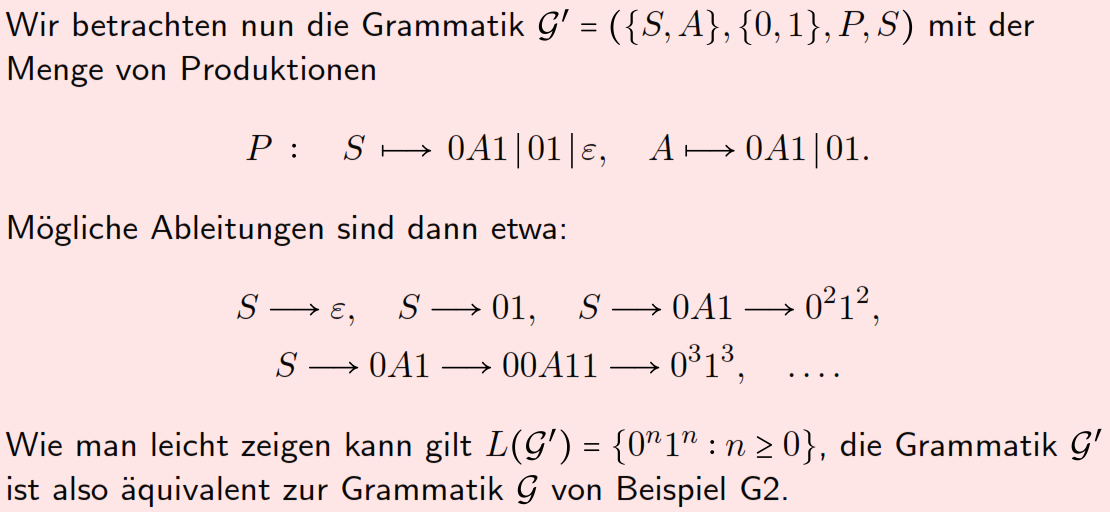


Wie gut zu sehen ist, kann diese Produktion als rekursiv verstanden werden.

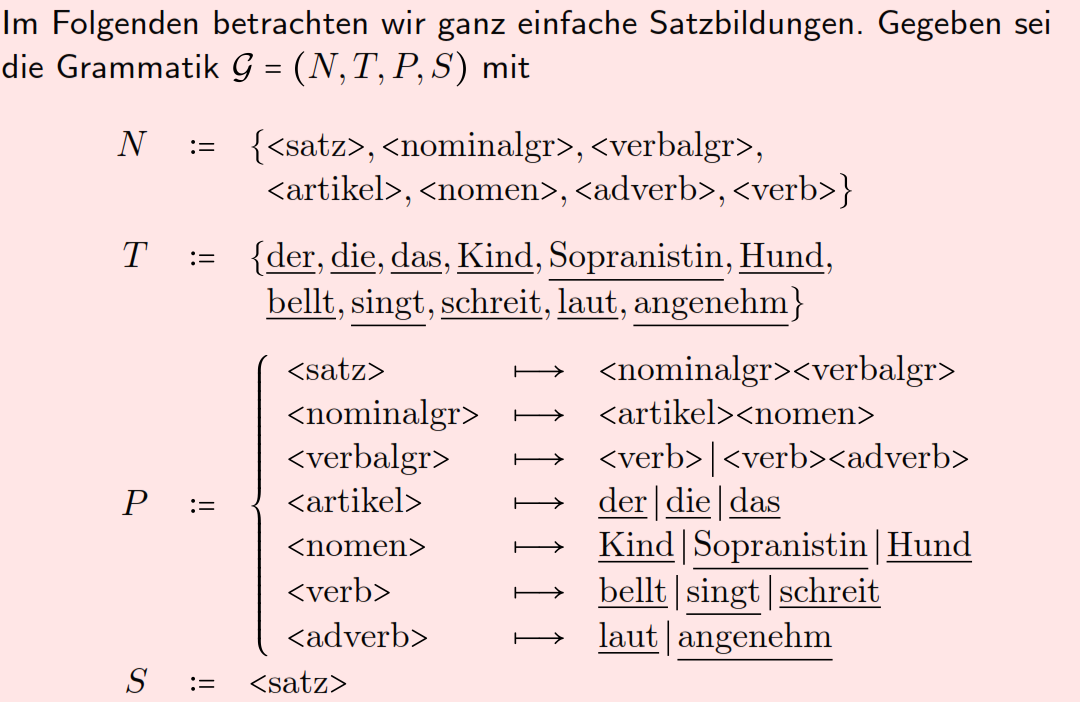
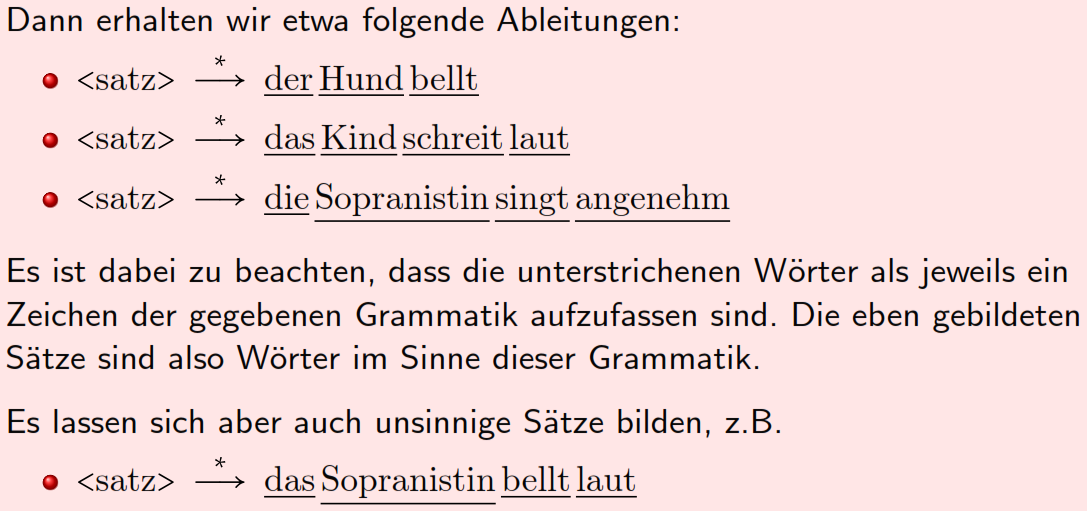
### Definition 62

Zwei Grammatiken G1 und G2 heissen genau dann **äquivalent**, wenn L(G1) = L(G2).

#### Beispiel (G3)



#### Beispiel (G5)

### Lemma 63 (Ableitbarkeitsrelationen)

Sei G = (N, T, P, S) eine Grammatik und seien und die eingeführten zu G gehörenden Ableitbarkeitsrelationen. Dann gilt für alle u,v,w,u’,v’ (N T)\*:

(u, v ) P => u v

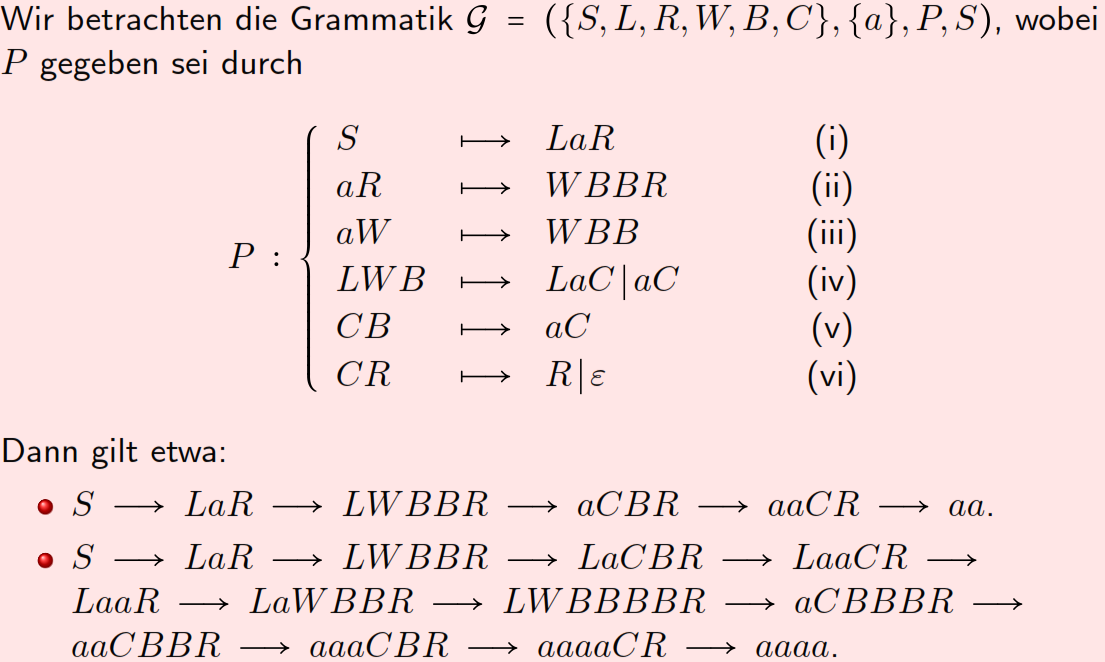
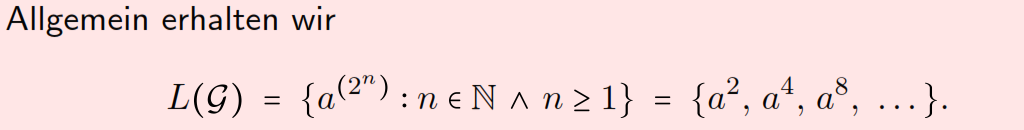
u u

u v v w => u w

u u’ v v’ => uv u’v’

#### Beispiel (G6)

Wir betrachten nun eine Grammatik, bei der auf der linken Seite mehr als ein nichtterminales Symbol auftreten kann.

Dies zeigt, dass die Komplexität der Produktion einer Grammatik Einfluss auf die Komplexität der dadurch generierten Sprache hat.

Wir führen nun eine Klassifizierung von Grammatik aufgrund der Form ihrer Produktionsregeln ein.

### Definition 64 (Klassifizierungen von Grammatiken)

Sei G = (N, T, P, S) eine Grammatik. Dann bezeichnen wir G als eine Grammatik vom

**Typ-0 (allgemein)**, genau dann, wenn es sich um eine Grammatik im Sinne von Def. 58 handelt.

**Typ-1 (kontextsensitiv)**, genau dann, wenn alle Produktionen die Form

uAv uwv

haben, wobei A N, u,v (N T)\* und w (N T)+ haben. Wenn S nicht auf der rechten Seite einer Produktion auftritt, darf zusätzlich S ε hinzugenommen werden. Also müssen all Produktion genau ein nichtterminales Symbol durch ein nichtleeres Wort ersetzen und sind abhängig von den Zeichen vor und nach dem zu ersetzenden nichtterminalen Symbol (der Kontext).

**Typ-2 (kontextfrei)**, genau dann, wenn alle Produktionen die Form

A u

mit A N und u (N T)\* haben. Also wenn genau ein nichtterminales Symbol durch ein Wort ersetzt wird, unabhängig davon, was vor oder nach dem Symbol steht.

**Typ-3 (rechtslinear)**, genau dann, wenn alle Produktionen die Form

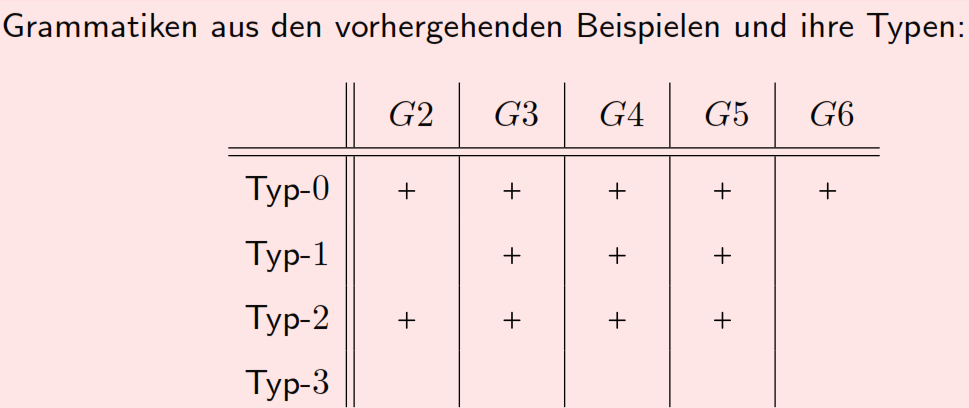
A ε oder A aB

mit A, B N und a T haben. Ein nichtterminales Symbol wird also durch das leere Symbol oder durch ein terminales, gefolgt von einem nichtterminalen Symbol ersetzt.

### Definition 65

Eine formale Sprache L heisst genau dann Sprache vom **(Chomsky- Typ-i)**, wenn eine Grammatik vom Typ-i mit L = L(G) existiert. Wir schreiben **[Typ-i]** für die Menge aller formalen Sprachen vom Typ-i. Sprachen von Typ-2 werden auch als **kontextfreie** Sprachen bezeichnet.

#### Bemerkung



### Lemma 66



#### Bemerkung

Grammatiken vom Typ-2 und Typ-3 lassen Produktionen der Form A ε (ε-Regeln) ohne Einschränkungen zu, bei Grammatiken vom Typ-1 ist dies nur unter Einschränkungen der Fall. Deshalb ist nicht jede Grammatik vom Typ-2 auch eine Typ-1 Grammatik (vgl. G2 in der Grafik der vorigen Bemerkung).

### Lemma 67

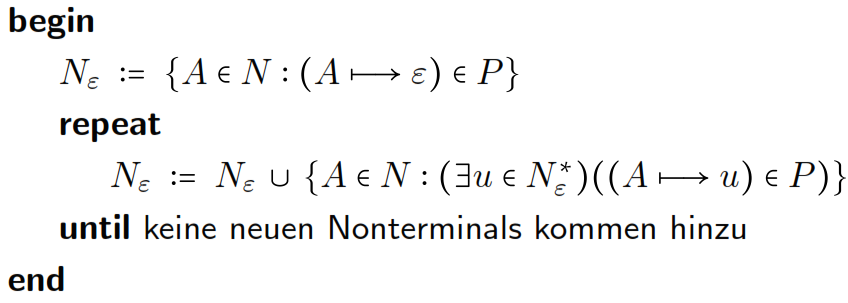
Sei G = (N, T, P, S) eine kontextfreie Grammatik. Dann existiert eine kontextfreie Grammatik G’ = (N, T, P’, S) ohne ε-Regeln, für die L(G’) = L(G) \ {ε} gilt.

#### Konstruktion Grammatik ohne ε-Regeln

Da G kontextfrei ist, gilt für alle u (N T)\*

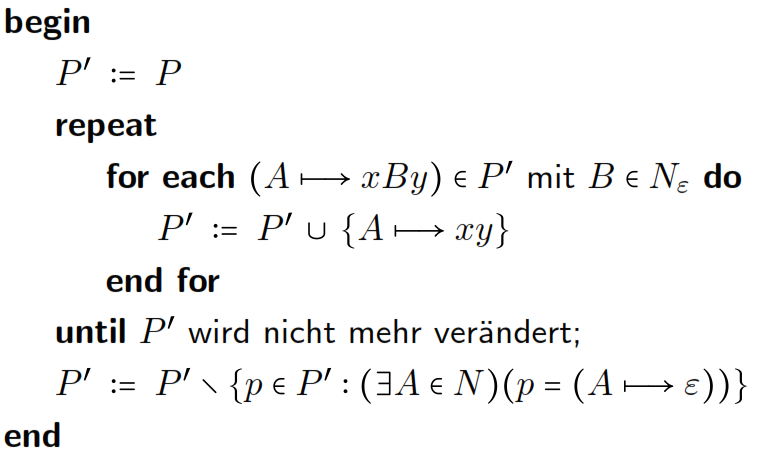
1. u ε => u N\*

kann das leere Wort nur von Nonterminals aus erreicht werden. Wir bestimmen nun zuerst die Menge Nε aller Nonterminals, aus denen das leere Wort ableitbar ist. Wegen (1) lässt sich Nε durch folgenden Algorithmus berechnen:



Wir fügen also erst alle einzelnen Noterminals hinzu, aus denen das leere Wort ableitbar ist. Dann testen wir, ob von einem einzelnen Nonterminal ein Wort bestehend aus Nε-Symbolen ableitbar ist. Falls ja, fügen wir dieses hinzu. Das machen wir so lange, bis keine neuen Symbole hinzukommen.

Als nächstes betrachten wir die Produktionen, die Zeichen aus Nε enthalten und konstruieren analoge Produktionen, bei denen diese Zeichen weggelassen sind. Anstatt also die ε-Regeln auszuführen, simuliert man ihre Anwendung, indem die Zeichen, die dann später weggelassen würden gar nicht erst generiert:



Wir nehmen also alle Produktionen, die ein Nonterminal enthalten. Dann fügen wir für diese je die Produktion hinzu, bei denen dieses Nonterminal einfach weggelassen wird. Dies wird so lange gemacht, bis sich die Menge nicht mehr verändert. Schliesslich entfernt man alle ε-Regeln.

#### Beispiel

