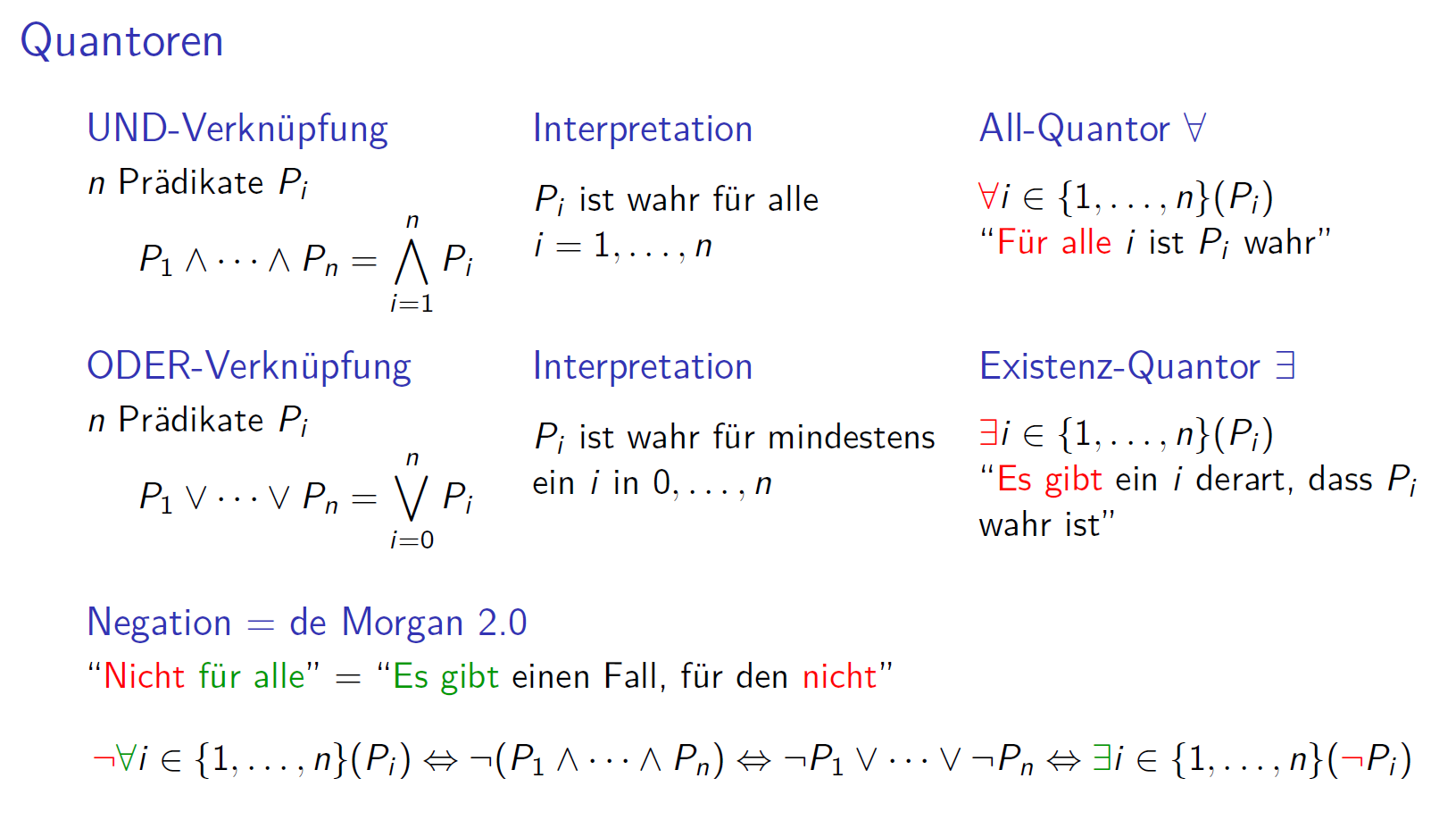
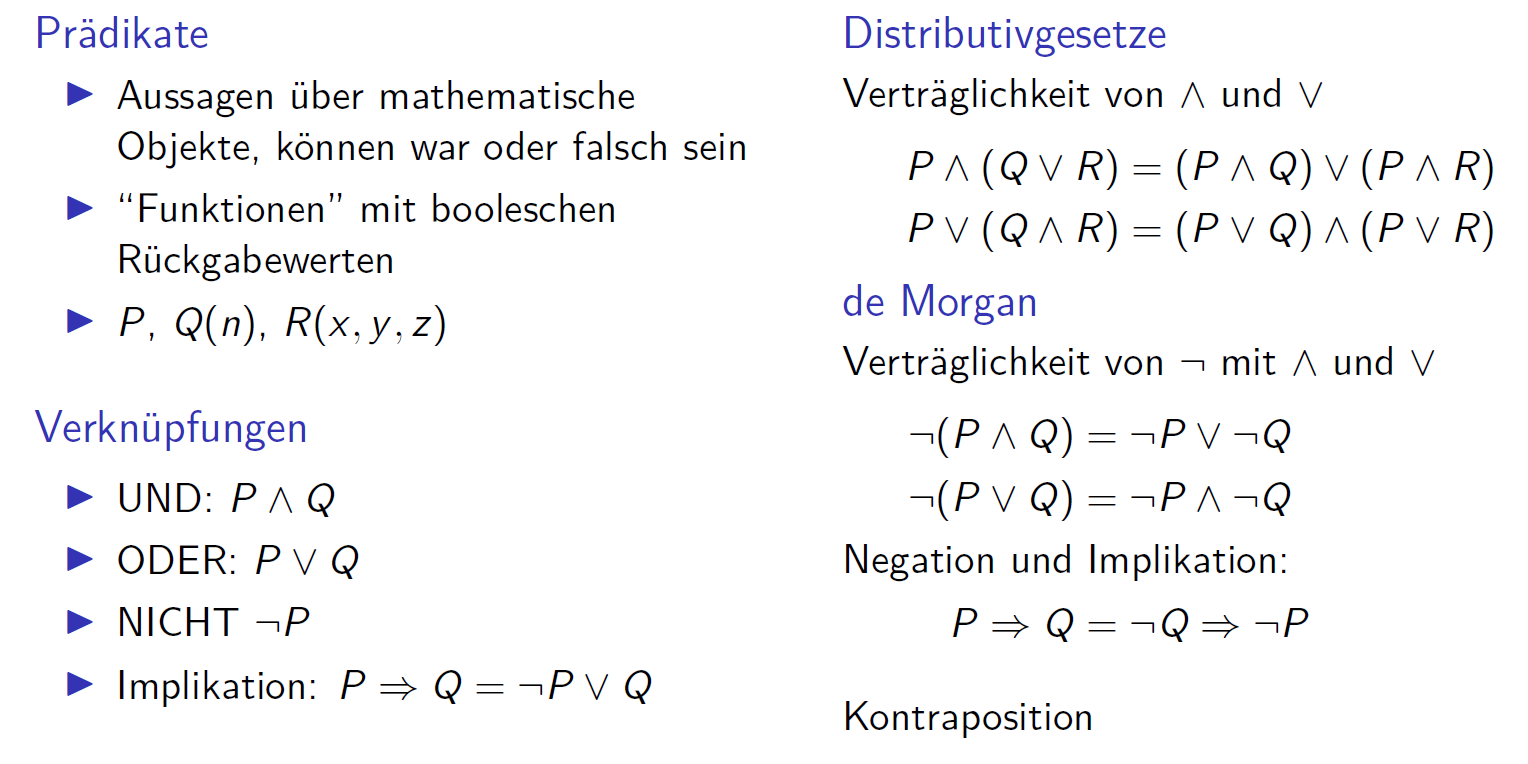
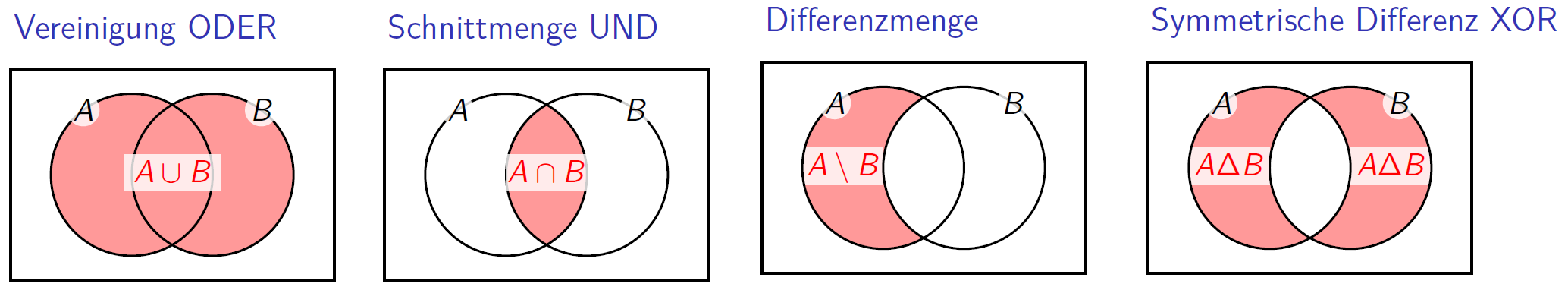
# Grundlagen

## Logik & Quantoren

  
Normalformen: Konjunktiv: UND von ODER-Thermen, Disjunktiv: ODER von UND-Thermen

## Mengenlehre

Mengen = «Primitive Datentypen der Mathematik», Zusammenfassung bestimmter Objekte.

**Konstruktion**: Grundmenge *G*, Prädikat *P*(*x*) *A* = {*x* ∈ *G* | *P*(*x*)} 

## Paare und Tupel, Graphen

## 

## Beweise

* **Konstruktiver Beweis**: Liefert explizite Lösung, Algorithmus
* **Widerspruchsbeweis**, besonders geeignet für Unmöglichkeitsaussagen
* **Vollständige Induktion**: Für Aussagen die für alle n ∈ N gelten
  + 1. Verankerung: n = 1
  + 2. Annahme: Formel korrekt für n
  + 3. Induktionsschritt: Beweis für n + 1

### Unendlichkeit

Eine Unendliche Menge A heisst abzählbar Unendlich, wenn sie gelichmächtig ist wie die natürlichen Zahlen. A heisst überabzählbar unendlich, wenn es keine Bijektion wischen N und A gibt.

# Sprachen

**Alphabet**: Eine nichtleere endliche Menge **Σ** = {…}  
**Zeichen**: Elemente der Menge **Σ  
Wort:** Zeichenkette (Tupel aus Elementen des Alphabets der Länge n 🡪 w ∈ Σn**Leeres Wort:** Zeichenkette ε ∈ Σ0 = { ε } der Länge 0  
**Menge aller Wörter:** Σ\* = {𝜀} ∪ 𝛴1 ∪ 𝛴2 ∪ 𝛴3 ∪ … =   
**Wortlänge:** w ∈ Σn 🡪 |w| = n

w ∈ Σn & a ∈ Σ 🡪 |w|a = Anzahl Zeichen a im Wort w  
 Bsp: |01010|1 = 2, |a5b3|a = 5 (b3 = bbb)  
**Sprache:** Teilmenge L ⊂ 𝛴∗ 🡪 Zu jeder Maschine M gibt es eine Sprache L(M)

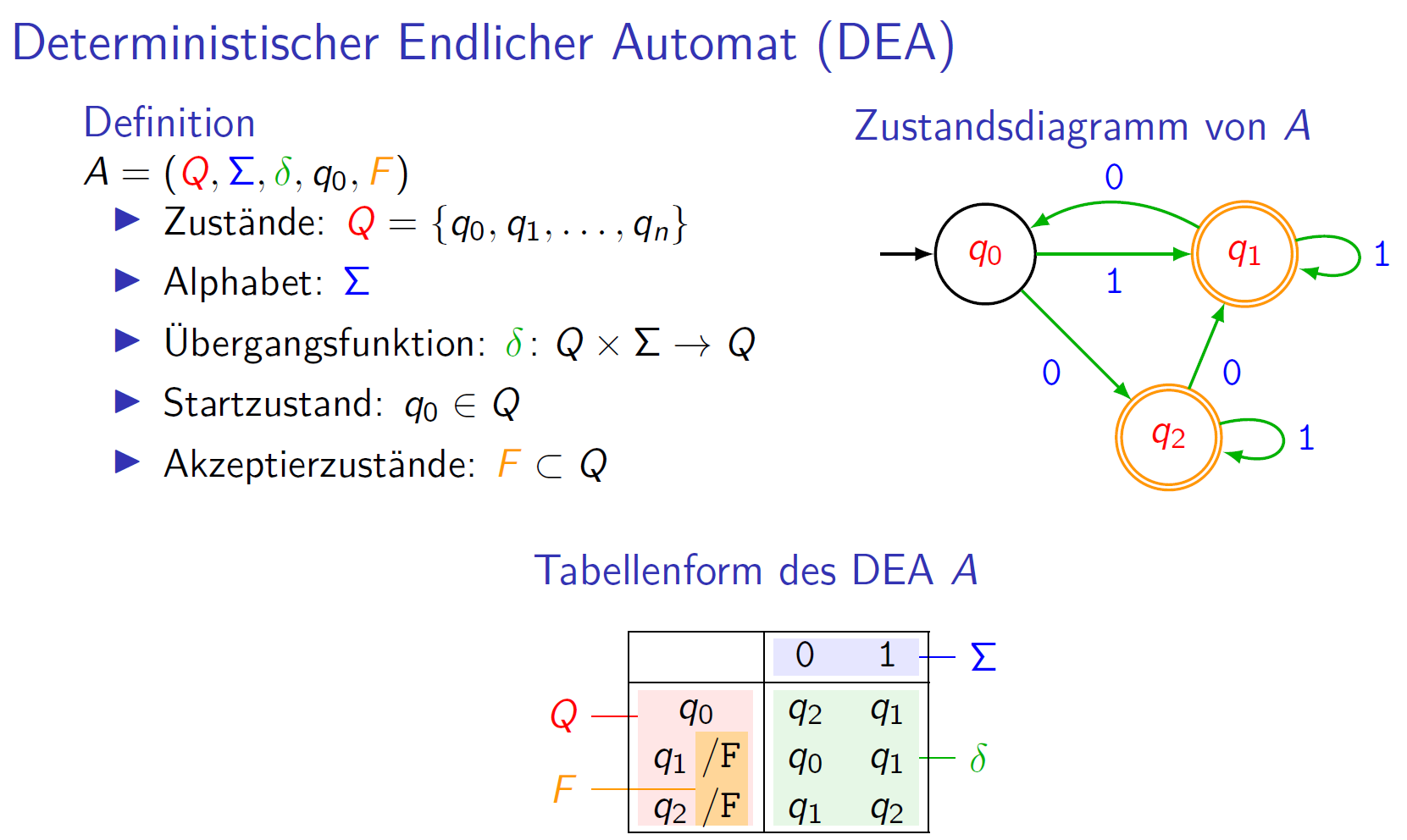
Bsp: 𝛴 = {0, 1}, Sprache aller Binärstrings: L = 𝛴\*

# Reguläre Sprachen

Eine Sprache L ⊂ 𝛴∗ heisst regulär, wenn es ein DEA A gibt mit L(A) = L.  
Die regulären Sprachen sind kontextfrei.

## Deterministischer Endlicher Automat (DEA)

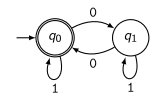
Ein endlicher Automat hat keine Speicher, nur Zustände



## Myrell-Nerode Automat (Rekonstruiere DEA):

Mittels dem Satz von Myrell-Nerode kann erkannt werden, ob eine Sprache regulär ist.

**Beispiel: Σ** = {0, 1}, L = { w ∈ Σ\* | |w|0 gerade }

* *Es gibt zwei Zustände.*  
   

|  |  |  |
| --- | --- | --- |
| **w** | **L(w)** | **Q** |
| Σ | L(Σ) = L | q0 |
| 0 | L(0) = { w ∈ Σ\* | |w|0 ungerade } | q1 |
| 1 | L(1) = { w ∈ Σ\* | |w|0 gerade } = L | q0 |
| 10 | L(0) = { w ∈ Σ\* | |w|0 ungerade } | q1 |
| … | … | … |

Ergibt dieser Automat keine endliche Menge von Zuständen, dann gibt es kein DEA.

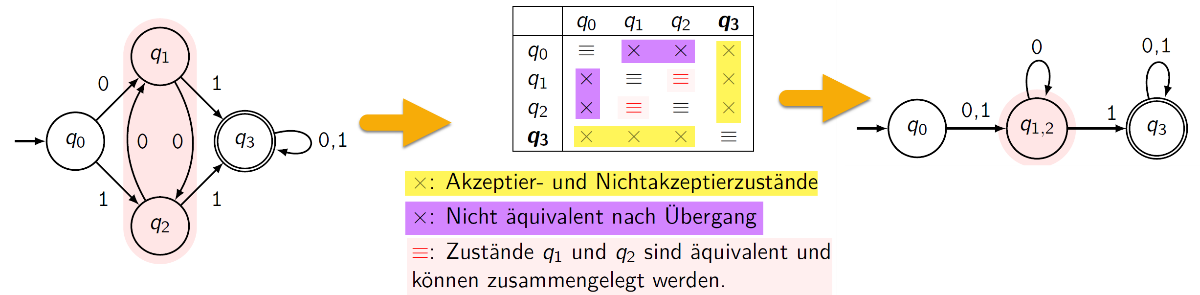
## Minimaler Automat

Dient als Beweis ob zwei Automaten die gleiche Sprache akzeptieren: L(A) = L(B)  
Jeder DEA hat ein Minimalautomat und kann so mit einem andern verglichen werden.

### Ablauf:

1. Tabelle aller Zustände erstellen
2. Alle Zustände zu sich selber äquivalent markieren 🡪 ≡
3. Paare aus einem Akzeptier- und einem Nichtakzeptierzustand 🡪 X d
4. Zweifingertechnik:
   1. Von einem Paar alle Übergänge testen, führt zu einem Paar mit X? Ja 🡪 X
   2. Wiederhole a) für alle Paare ohne Zuordnung
5. Wiederhole 4. Bis kein neues X gesetzt werden kann.
6. Restliche Zustände sind Äquivalent (🡪 ≡) und können minimiert werden.

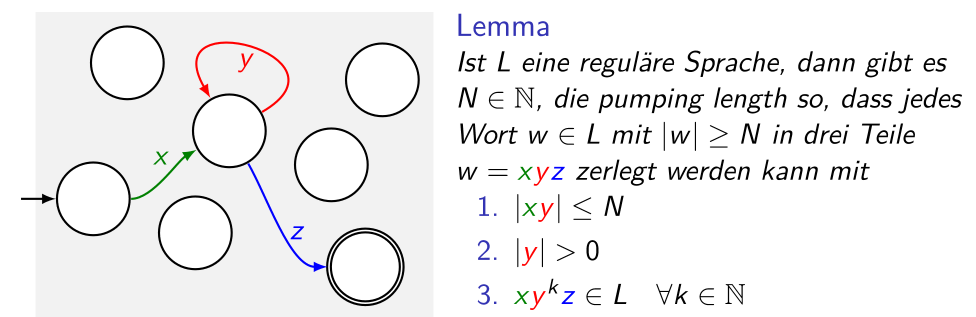
**Beispiel:**



## Pumping Lemma

Idee: Wenn ein Wort länger sein kann als die Anzahl zustände, dann muss mindestens ein Zustand mehrmals vorkommen im endlichen Automat. Das heisst ein Wort ist aufpumpbar.

Man kann damit aber nur nachweisen, dass eine Sprache **nicht regulär** ist, wenn sie mindestens ein Wort enthält, das nicht aufpumpbar ist. 🡺 Widerspruchsbeweis.



### Beweis/Ablauf

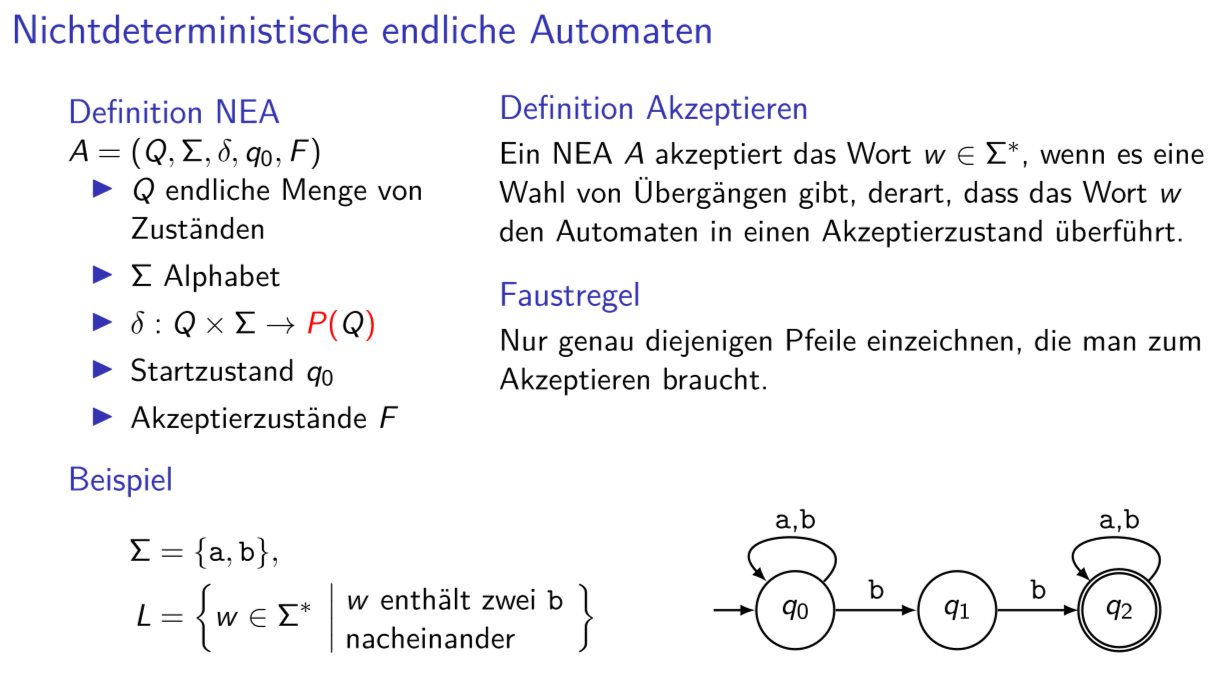
1. Annahme: L ist regulär
2. Es gibt die Pumping Length N
3. Wähle ein Wort w ∈ L mit | w | ≥ N, Definition mit N schreiben
4. Aufteilung des Wortes gemäss Pumping Lemma w = xyz, |xy| ≤ N, |y| > 0
5. Auswirkung des Pumpens mit Begründung
6. Widerspruch und Schlussfolgerung

### Beispiele

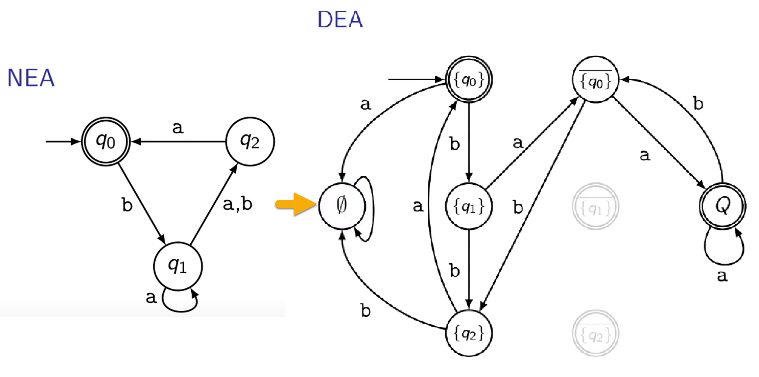
|  |  |
| --- | --- |
| L = {0n1n | n ≥ 0}   1. Annahme: L ist regulär 2. Pumping Length: ∃𝑁 ∈ ℕ 3. W = 0N1N 4. Aufteilung: 5. Pumpen erhöht nur die Anzahl 0, jedoch nicht Anzahl 1 6. Widerspruch: 𝑥𝑦𝑘𝑧 ∉ 𝐿 für 𝑘 ≠ 1, im Widerspruch zum Pumping Lemma Wort ist nicht mehr Element der Sprache für k!=1, daher ist L nicht regulär | 𝐿 = {𝑤 = 𝑤|𝑤 ∈ {0,1}∗ }   1. Annahme: L ist regulär 2. Pumping Length: ∃𝑁 ∈ ℕ 3. W = 10N 4. Aufteilung: 5. Pumpen: Auf der linken Seite des Gleichheitszeichen wird eine grössere Binärzahl, wie auf der rechten Seite, stehen. 6. Widerspruch: L ist nicht regulär |

## Nichtdeterministische endliche Automaten (NEA):

Automat kann sich aufhängen. Jede Verzweigung muss einzeln geprüft werden.

  
***Ɛ-Übergänge***: Übergang ist Gratis, jeder NEAƐ kann immer in NEA umgewandelt werden

### Transformation NEA 🡪 DEA

1. Alle Kombinationszustände einzeichnen
2. Alle Zustände, mit einem enthaltenen Akzeptierzustand 🡪 Akzeptierzustände
3. Vom Startzustand ausgehend alle Übergänge einzeichnen. 

## Mengenoperationen

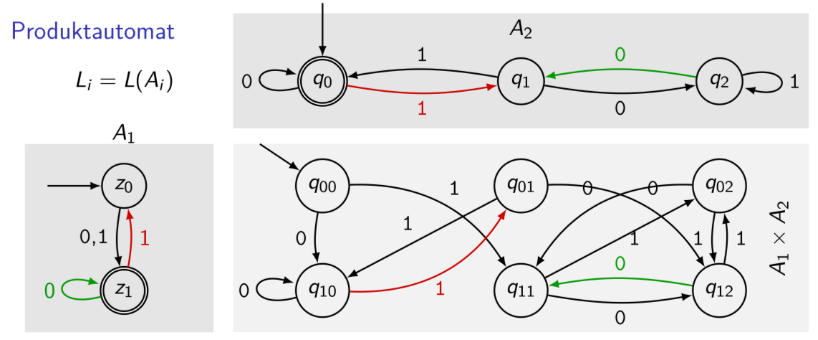
Wenn A1 und A2 DEA’s sind so kann man Mengenoperationen darauf anwenden und erhält so immer wieder einen DEA, also wieder eine reguläre Sprache.

**Operationen:**

* Schnittmenge 𝐿1⋂𝐿2: 𝐹 = 𝐹1𝑥𝐹2
* Vereinigung 𝐿1⋃𝐿2: 𝐹 = 𝐹1𝑥𝑄2 ⋃ 𝑄1𝑥𝐹2
* Differenz 𝐿1\𝐿2: 𝐹 = 𝐹1𝑥(𝑄2\𝐹2)

### Produktautomat

1. Beide Automaten, wenn möglich, auf eine Line strecken/dehnen
2. Beide Automaten als Tabelle darstellen, dass die Startzustände einander zugeneigt sind
3. Alle Übergänge einzeln übertragen. Wobei ein Zustandswechsel des einten Automaten sich auf die Horizontale auswirkt und bei der anderen auf die Vertikale.
4. Zustände des neuen Automaten einzeichnen
   1. Jedes Paar aus A1 und A2 Zustände ergeben ein neuen
   2. Je nach Operation Akzeptierzustände einzeichnen
      1. **Schnittmenge (𝐿1 ∩ 𝐿2):** Alle Zustände, welche bei beiden Automaten Akzeptierzustände sind.
      2. **Vereinigung (𝐿1 ∪ 𝐿2):** Alle Zustände, welche bei min einem Automaten ein Akzeptierzustand ist.
      3. **Differenz (L1\ L2):** Alle Zustände, welche nur beim ersten Automaten ein Akzeptierzustand sind.

**Akzeptierzustände:  
𝐿1∩𝐿2**: {q10}

**𝐿1∪𝐿2:** {q00, q10, q11, q12}

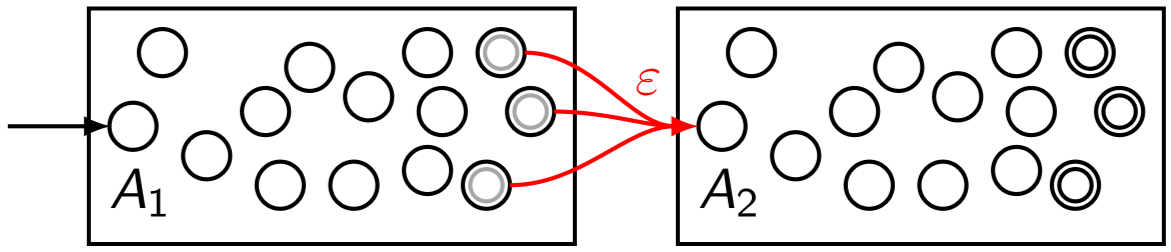
**L1\L2 :** {q11, q12}

## Reguläre Operationen

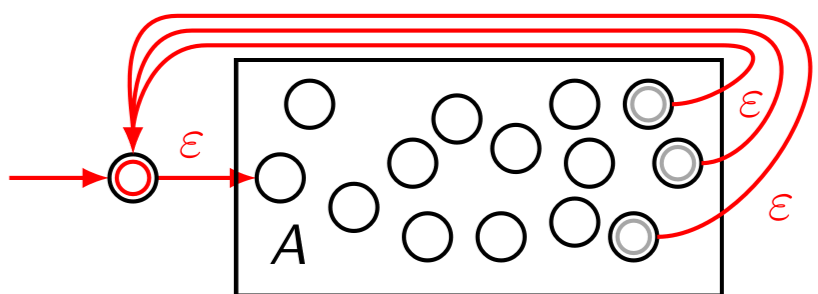
1. Vereinigung/Alternative (Oder) : L = 𝐿1 ⋃ 𝐿2 = 𝐿(A1)⋃ 𝐿(A2)



Neuer Startzustand mit Ɛ-Übergängen zu den einzelnen Automaten

1. Verkettung: L = 𝐿1 𝐿2 = 𝐿(A1)𝐿(A2) = { w1w2 | wi ∈ Li}

A1 Akzeptierzuständen mit Ɛ-Übergängen zu Startzugang A2,   
A1 hat keine Akzeptierzustände mehr

1. \*-Operation: L\* = { Ɛ } ∪ L ∪ L2 ∪ … = 

Neuer Startzugang mit Ɛ-Übergang zum alten Startzustand, A-Akzeptierzustände werden mit Ɛ-Übergang verknüpft mit neuem Startzugang.   
Leeres Wort ist ebenso Akzeptiert

## Reguläre Ausdrücke

|  |  |  |
| --- | --- | --- |
| Ausdruck r | L(r) | Bedeutung |
| a | {a} | *Steht für das Zeichen a ∈ Σ* |
| . | Σ | *Steht für ein beliebiges Zeichen aus Σ* |
| [abc] | {a, b, c, d} | *Steht für ein Zeichen aus {…} ∈ Σ* |
| [1-9] | {1, 2, …, 9} | *Steht für eine positive Ziffer* |
|  | { Ɛ } | *Steht fürs leere Wort* |
|  | {} | *Steht für die leere Sprache* |
| [^xyz] |  | *Nicht x, y oder z* |
| a\* | {a\*} | *0 oder beliebig viele a* |
| a+ | {aa\*} | *Mindestens 1 a* |
| a? oder |a |  | *0 oder 1 mal a* |
| a{5} | {a5) | *a genau 5 Mal* |
| a{2,} | {aa\*} | *a mindestens 2 mal* |
| a{1,3} |  | *a zwischen 1 oder 3 mal.* |
| a{,4} |  | *a maximal 4 mal.* |
| ab|cd |  | *ab oder cd* |

### Umwandlung Regex -> NEA

1. Von innen nach Aussen übersetzen (Klammern zuerst)
2. Einzelne Teile mittels Ɛ-Übergängeverbinden

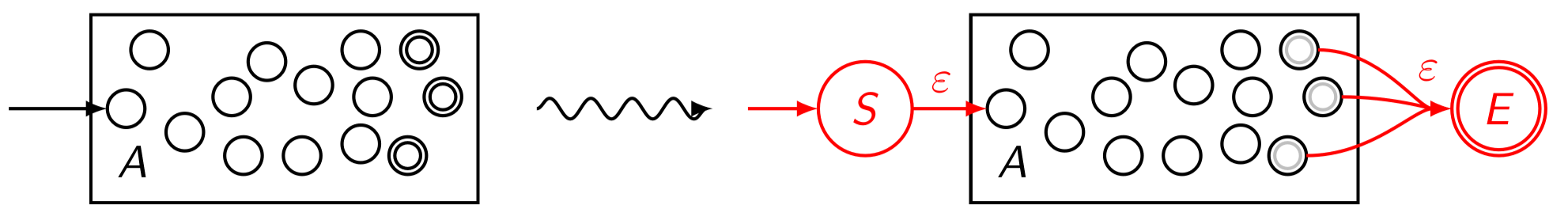
**Beispiele:**

|  |  |  |  |
| --- | --- | --- | --- |
| *ab|cd* | | *(ab)\*c* | |
| *a* | *Ɛ* | *∅* | *ab* |

### Umwandlung DEA -> VNEA mit einem Übergang

VNEA = NEA mit rexex-Übergängen

### Vorgehen

1. Startzugang darf keine Übergänge haben und Akzeptierzustand muss minimiert werden:  
    
2. Reduktion: Alle Zwischenzustände entfernen. Übrig bleibt ein regulärer Ausdruck:



### Beispiel

# Kontextfreie Sprachen

Kontextfreie Sprachen können nur von nichtdetermistischen Stackautomaten erkannt werden. Existiert ein Stackautomat, Regex oder eine Grammatik, ist die Sprache kontextfrei.

Der Name «kontextfrei» basiert darauf, dass die Regeln nicht auf den Kontext ankommt in dem die Variable auf der linken Seite sich befindet.

## Kontextfreie Grammatik

Eine kontextfreie Grammatik ist ein Quadrupel G = (V, Σ, R, S). Dabei gelten folgende Rahmenbedingungen:

1. V ist eine endliche Menge von Variablen
2. Σ ist eine endliche Menge von Zeichen, disjunkt zu V, auch Terminalsymbole genannt
3. R ist eine Menge von Regeln
4. S ∈ V ist die Startvariable

***Regel***: *Besteht aus einer Variable und einer Kette von Variablen und Terminalsymbolen, geschrieben als: A 🡪 BCx (Rechts eine beliebige Folge von Variablen und Terminalsymbole)*

### Beispiele

|  |  |
| --- | --- |
| *L = { w ∈ {(, )} | w gültige Klammerung}*  V = {K} Σ = {(, )} S = K R = {K 🡪 Ɛ, K 🡪 KK, K 🡪 (K)} | *L = {0n1n | n ≥ 0}*  V = {Q},  Σ = {0, 1} S = Q R = {Q 🡪 Ɛ, Q 🡪 0Q1} |

### Grammatik für reguläre Operationen

Neue Startvariable S0 notwendig. Variablen V = V1⋃ V2 ⋃ {S0}, Regeln müssen geeignet erweitert werden:

* **Alternative / OR**: Regeln für 𝐿1𝑈𝐿2 => 𝑅 = 𝑅1 𝑈 𝑅2 𝑈 {𝑆0🡪𝑆1,𝑆0🡪𝑆2}
* **Verkettung / AND**: Regeln für 𝐿1𝐿2 => 𝑅 = 𝑅1 𝑈 𝑅2 𝑈 {𝑆0🡪𝑆1𝑆2}
* **\*-Operation**: Regeln für 𝐿1∗ => 𝑅 = 𝑅1 𝑈 {𝑆0🡪𝑆0𝑆1,𝑆0🡪 Ɛ}

## Parse-Tree

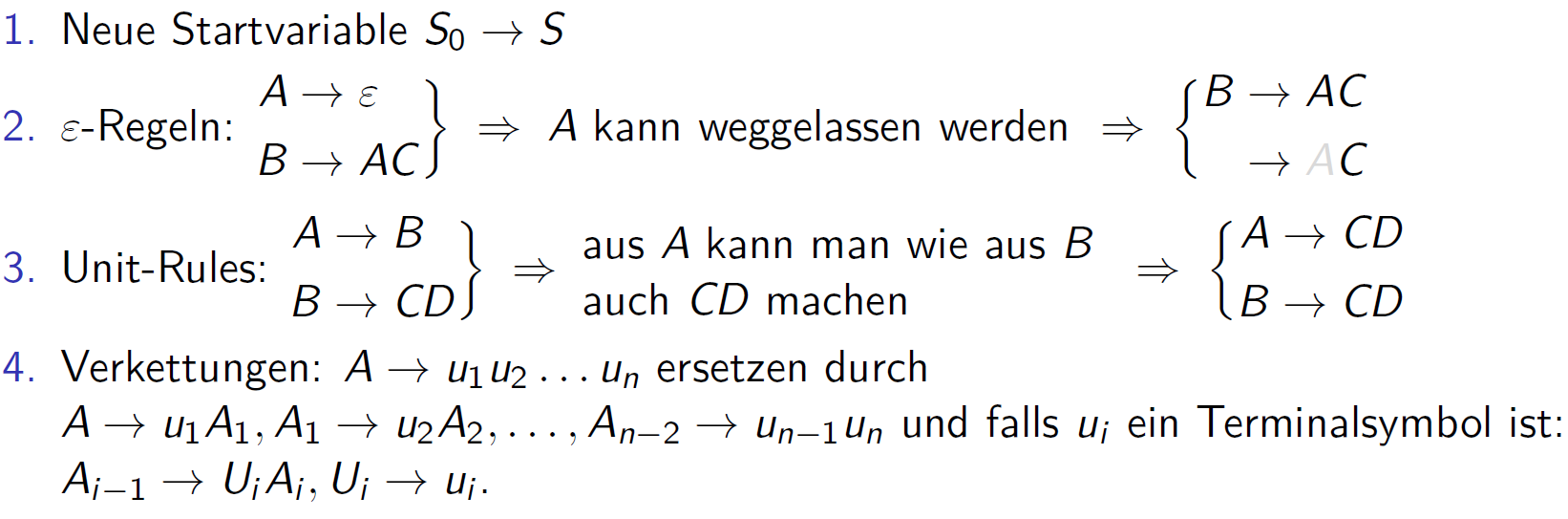
Zwei Ableitungen eines Wortes w einer kontextfreien Sprache L heissen äquivalent, wenn sie den gleichen Ableitungsbaum (Parse-Tree haben. Hat eine Sprache, Wörter mit verschiedenen Parse-Trees heisst sie mehrdeutig.

**Beispiel (Mehrdeutiger Parse Tree):** 

## Chomsky-Normalform

Eine Regel ist in der Chomsky-Normalform, wenn S auf der rechten Seite nicht vorkommt und jede Regel von der Form **A 🡪 BC** oder **A 🡪 a** ist. Zusätzlich ist noch die Regel **S 🡪 Ɛ** erlaubt.

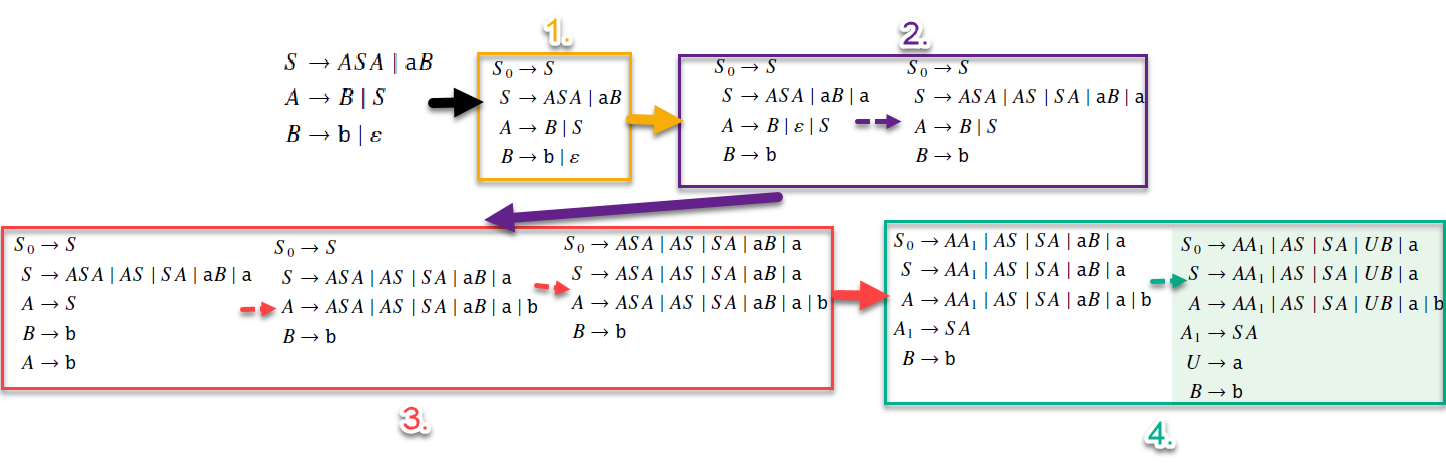
### Umwandlung



Damit kann die Länge der Ableitung eines Wortes bestimmt werden.   
Jedes Wort kann in 2\*|w|-1 Regelanwendungen gebildert werden.

* Höchstens |w|-1 Anwendungen von A🡪BC, |w| Anwendungen von A🡪a

### Beispiel



## CYK-Algorithmus (Cocke-Younger-Kasami)

Mit dem CYK-Algorithmus kann ermittelt werden ob Wort zur kontextfreien Sprache gehört.

Dauer: |w|3 🡪 O(n3)

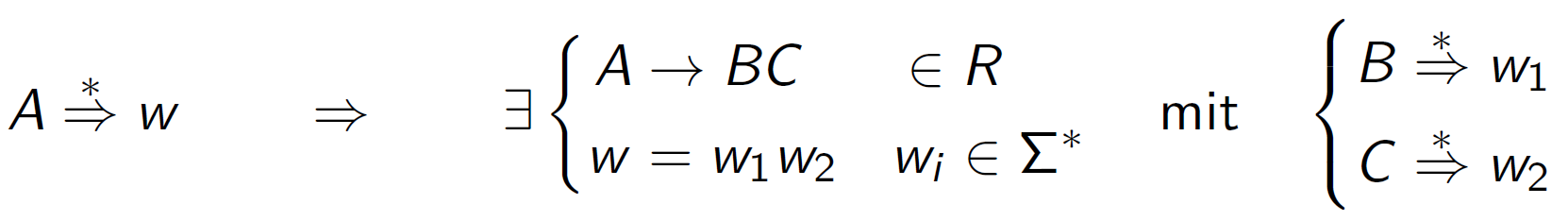
**Gegeben:**

* Grammatik G = (V, Σ, R, S) in Chomsky-Normalform
* Variablen A ∈ V
* Wort W ∈ Σ\*

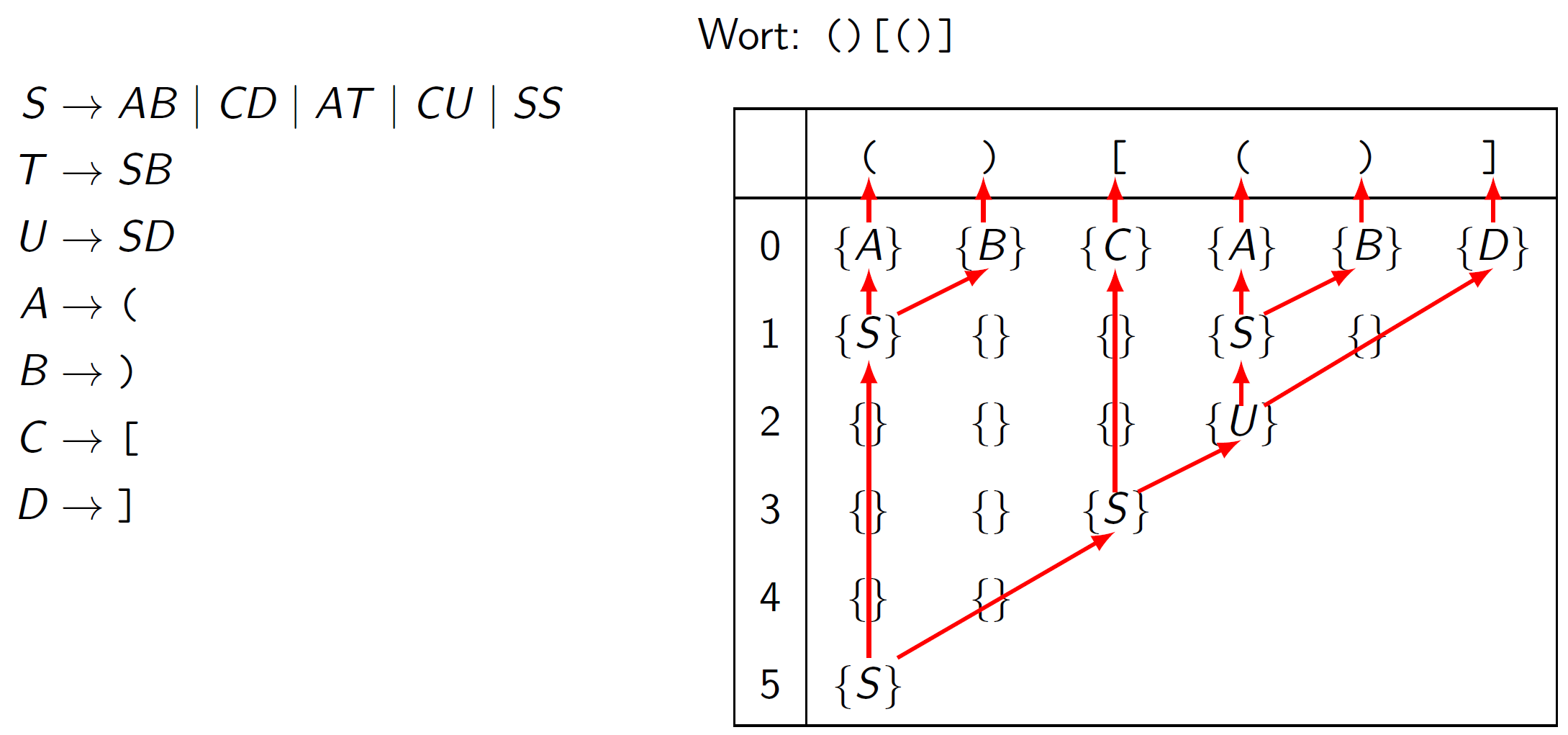
**Frage**: Ist w aus A ableitbar?

**Spezialfälle**: w = Ɛ und |w| = 1 🡪 w ist ein Terminalsymbol

Falls|w| > 1:

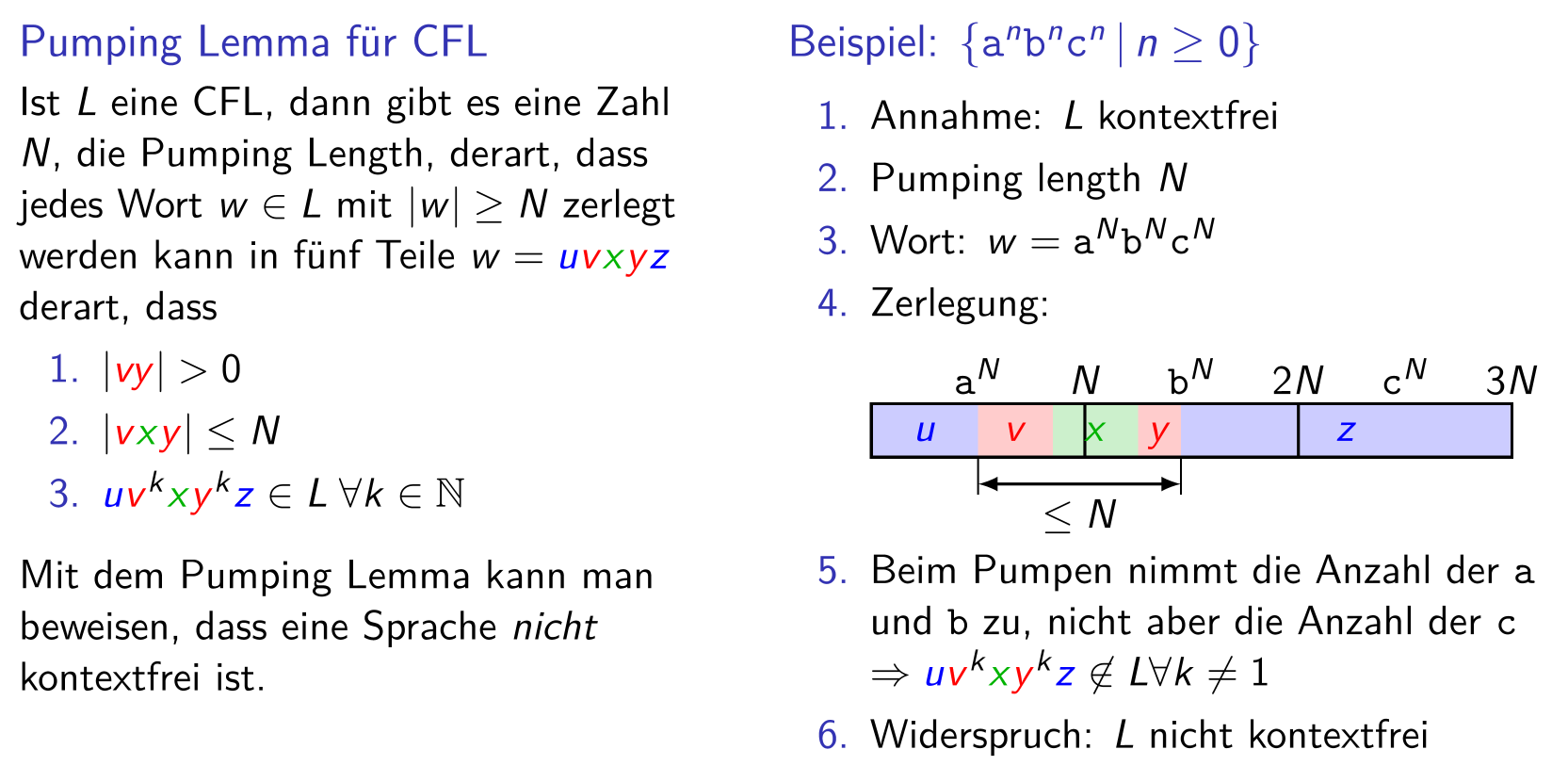


### Beispiel

  
*Hinweis: Ergibt auch den Parse-Tree des Wortes.*

## Pumping Lemma für kontextfreie Sprachen

Es gibt auch sprachen die nicht kontextfrei sind. Z.B die Sprache: L = {anbn cn| n ≥ 0}

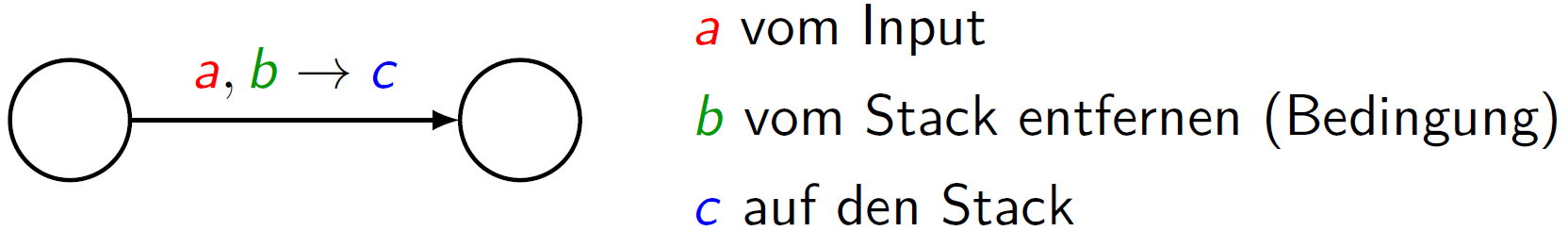
Auch dafür kann man Pumping Lemma verwenden als Beweis, muss aber die Grammatik dazu nehmen. 

# Stackautomat / Push Down Automat (PDF)

Stackautomat kann nur kontextfreie Grammatik erkennen. Er ist immer NICHT deterministisch.

Stackautomat P = (Q, Σ, Г, δ, S, F)

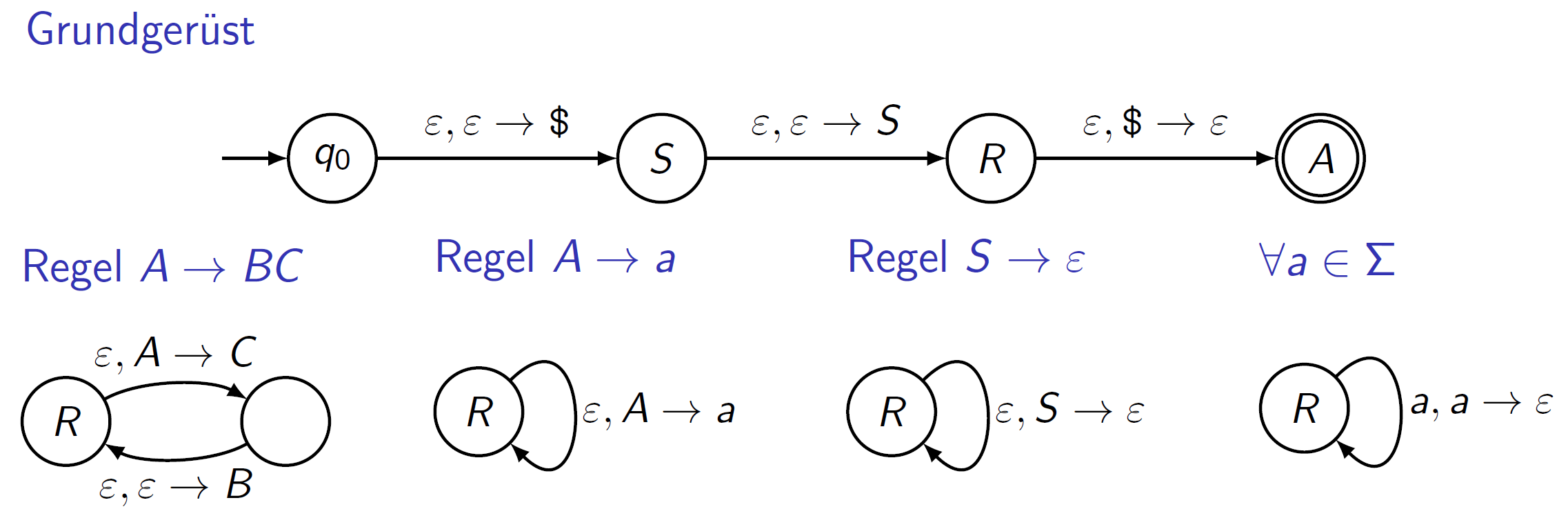
|  |  |  |
| --- | --- | --- |
| **Zeichen** | **Bezeichnung** | **Beispiel** |
| Q | Zustände | q0, q1, ... |
| Σ | Eingabe-Alphabet | {0,1} |
| Г | Stack-Alphabet | {0,1,$} |
| δ | Q x ΣƐ x ГƐ 🡪 P(Q x ГƐ) | Ɛ 🡪 $ |
| S | Startzustand | q0 |
| F | Akzeptierzustände | q3 |

**Übergänge:**   
  
*Verarbeite Input a und ersetze b Auf dem Stack durch c*

## Beispiel

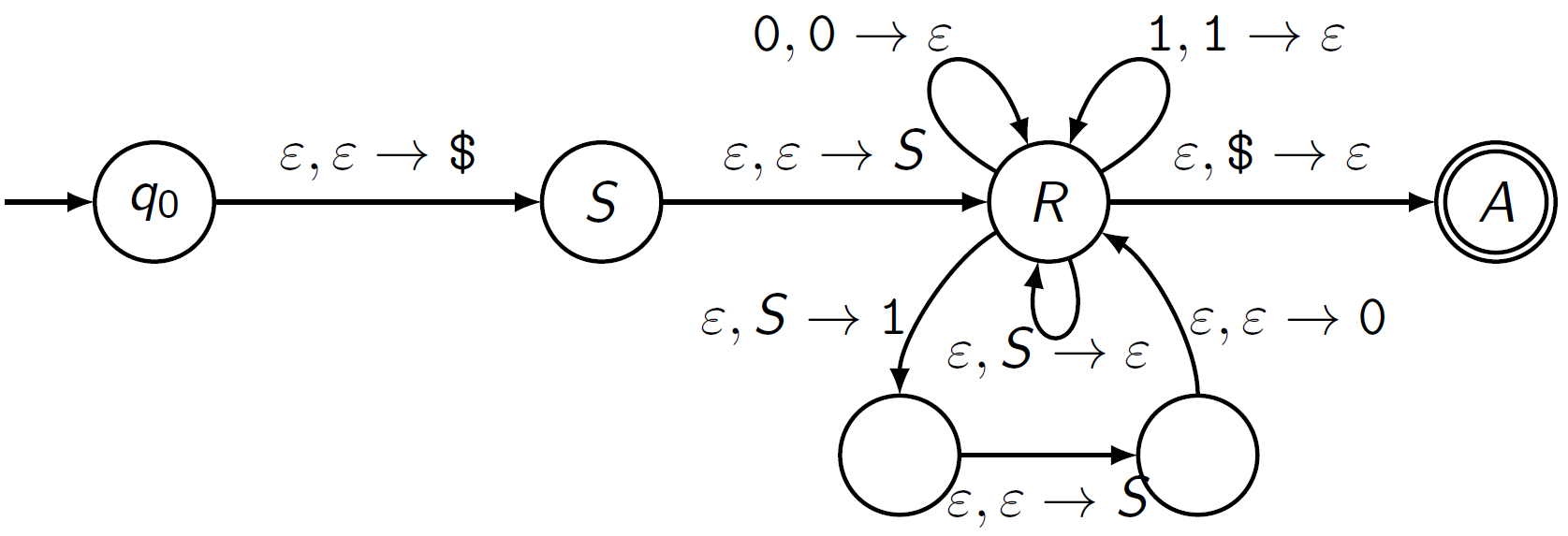
L = {0n1n | n ≥ 0}

## Umwandlung Grammatik zu Stackautomat



### Beispiel

L = {0n1n | n ≥ 0} R = {S 🡪 Ɛ, S 🡪 0S1}



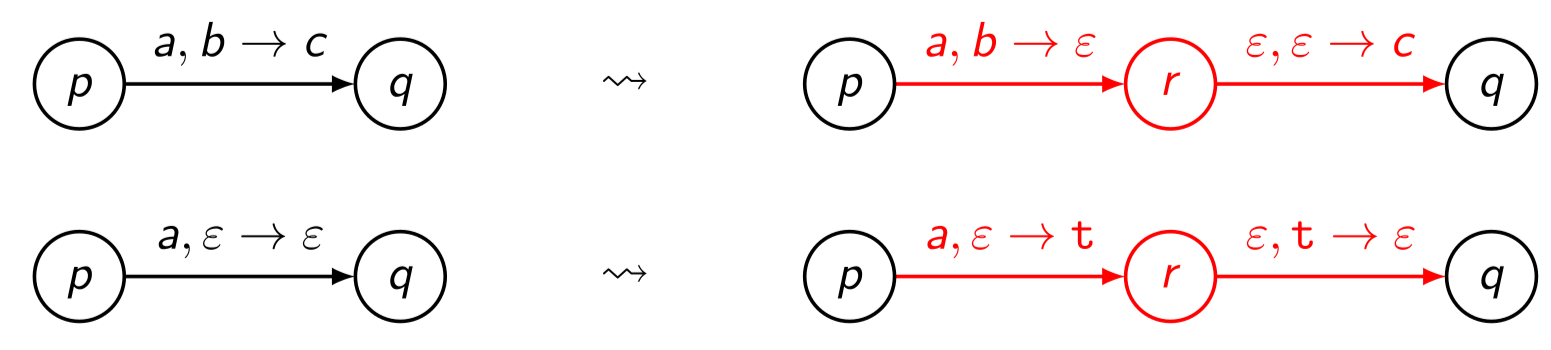
## Umwandlung Stackautomat zu Grammatik

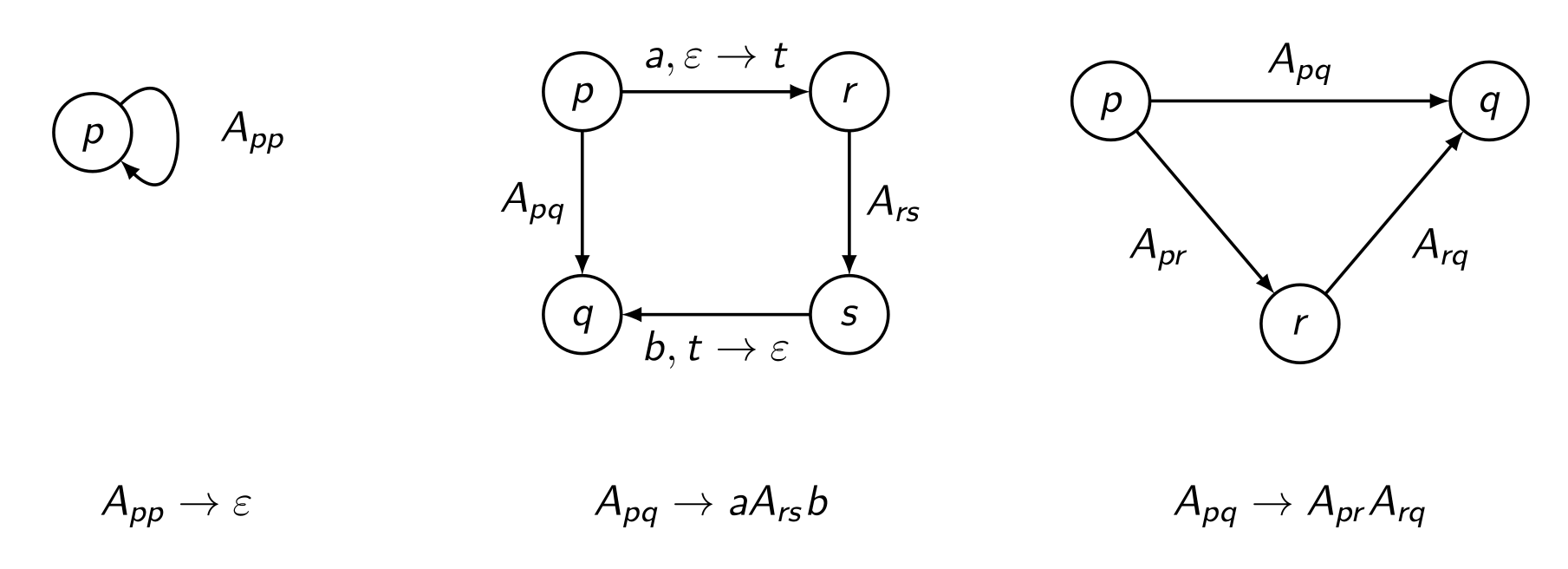
Ist P ein Stackautomat, dann gibt es eine Grammatik G, die die gleiche Sprache produziert.

Der Stackautomat muss zuerst normalisiert werden:

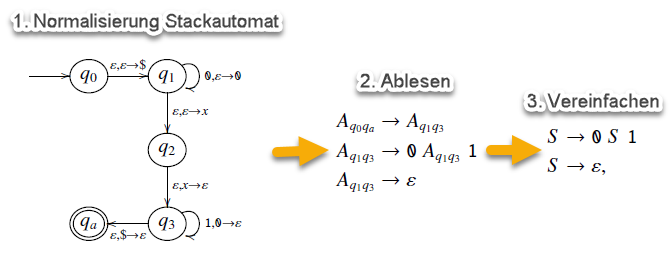
1. Stackautomat darf nur ein Akzeptierzustand haben   
   
2. Vor akzeptieren muss Stack geleert werden. Wird mit zusätzlichem Zeichen gelöst (Neuer Startzustand und nochmals neuer Akzeptierzustand)



1. Jeder Übergang legt ein Zeichen auf dem Stack oder entfernt ein Zeichen, nicht gleichzeitig. Ɛ-Übergänge benötigen weitern Zwischenzustand mit beliebigen Zeichen. 

Anschliessend können die Regeln abgelesen werden: 

**Beispiel:**L = {0n1n | n ≥ 0}

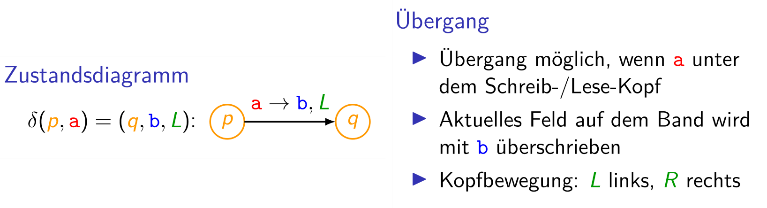


# Turing Maschine

Speicher ist ein unendliches Band mit mehreren Speicherzellen. Pro Zelle, ein Zeichen. Lesekopf liest/schreib Zeichen. Er muss sich immer entweder Links oder Rechts bewegen. Eine Sprache L heisst «**Turing-Erkennbar**», wenn es eine Turing-Maschine M gibt, die das Wort akzeptiert.

Turing Maschine M = (Q, Σ, Г, δ, S, qaccept, qreject)

|  |  |  |  |
| --- | --- | --- | --- |
| **Zeichen** | **Bezeichnung** | **Beispiel** |  |
| Q | Zustände | q0, q1, ... |
| Σ | Eingabe-Alphabet | {0,1} |
| Г | Stack-Alphabet | {0,1,⊔ } |
| δ | Q x Г 🡪 Q x Г x {L,R} | a 🡪 b, R |
| S | Startzustand | q0 |
| qaccept  qreject | Akzeptierzustand  Ablehnungszustand |  |



**Ablauf Programm:**

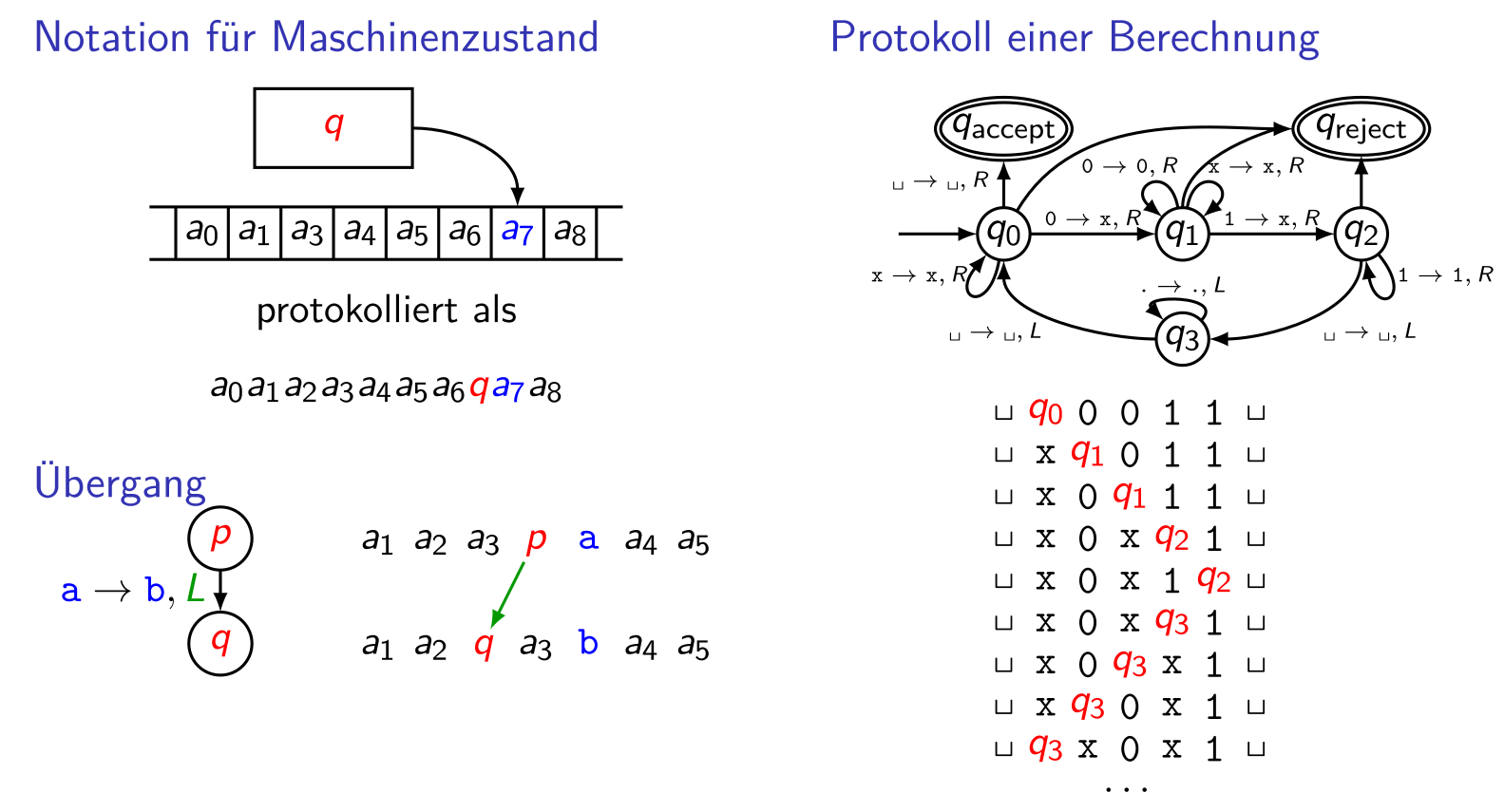
1. Inputwort w auf Band schreiben
2. Schreib-/Lesekopf auf erstes Zeichen positionieren (Meist ⊔)
3. Maschine starten, t(w) Einzelschritte ausführen
4. Maschine hält in qaccept oder qreject an und akzeptiert oder verwirft Wort entsprechend

**Beispiele**

|  |  |
| --- | --- |
| L = {anbncn | n ≥ 0} | L = {02n | n ∈ ℕ} |
| Binärzahlen um 1 erhöhen | Binärzahl durch 3 Teilbar (DEA in Turing umwandeln) |

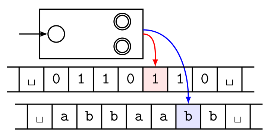
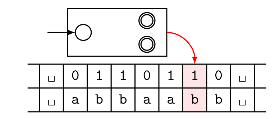
## Berechnungsgeschichte

(Kann auch auf ein Ausfüllrätsel reduziert werden und somit beweisen das SAT NP-Vollständig ist.)



## Varianten von Turing-Maschinen

Jede Sprache, die von einer alternativen Turingmaschine erkennt werden kann, kann auch von einer einspurigen Touring-Maschine erkennt werden. Dauert einfach länger.

* **Anderes Bandalphabet**
  + Simuliert z.B. durch binäre Codierung
  + O(t(n))
* **Mehrere Bänder (Ein Pointer pro Band)** 
  + Simulierbar mit 2n Mehrspurigem Band, jedes Band bekommt eine zusätzliche Spur für Position. Die Position wird über ein Zeichen gelöst, welches verschoben wird, und die Position anzeigt
  + ****O(t(n)2)
* **Mehrspurige TM (ein Pointer)** 
  + Simuliert, indem man die Spuren alle seriell hinterlegt
  + O(t(n))
* **Nicht deterministische TM**
  + Bei jedem Übergang maximal N verschiedene Möglichkeiten
  + Mehrere Berechnungswege möglich
  + Akzeptiert sobald ein Weg zu qaccept führt
  + Simulation: Maximal Nt(n)
    - 3 Bänder: Arbeitspand, Kopie von w und Liste aller Folgen von Wahlmöglichkeiten
* **Aufzähler**
  + Eine TM mit einem Drucker bei dem gefundene Wörter «ausgedruckt» werden können. Kann endlos laufen.
  + Wort kann mit «Ausdruck» verglichen werden um so akzeptiert zu werden
  + Aufzählbare Sprache ⬄ Turing-erkennbare Sprache

# Entscheidbarkeit

* Eine deterministische TM M heisst Entscheider, wenn sie auf jedem Input w anhält.
* Eine nichtdeterministische TM M heisst Entscheider, wenn jede Berechnungsgeschichte terminiert
* Entscheider = immer einen Definitiven Endzustand

**Turing-erkennbare Sprache**

* Die Sprache L heisst Turing-erkennbar, wenn es eine TM M gibt
* Kann evtl. auch unendlich lange laufen

**Turing-entscheidbare Sprache**

* Die Sprache L heisst Turing-entscheidbare Sprache, wenn es ein Entscheider M gibt

**Ist 𝑤 ∈𝐿(𝑀)?**

* Turing-erkennbare Sprache:
  + Wann weiss man, ob M nicht anhält oder einfach noch mehr Zeit benötigt?
* Turing-entscheidbare Sprache:
  + M wird auf Input w garantiert irgendwann anhalten

**Entscheidbare Sprache vs. erkennbare Sprachen**

Entscheidbare Sprache hält für jeden beliebigen Input an.

Erkennbare Sprache hält nur für akzeptierten Input an.

## Entscheidbare Probleme

E = Leerheitsproblem, EQ = Gleicheitsproblem, A = Akzeptanzproblem

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Problem | Wort | Bedingung | Entscheidbar | Entscheidungsalgorithmus |
| EDEA | <A> | L(A)=∅ | ja | Minimalautomat hat keinen Akzeptierzustand |
| ECFG | <G> | L(G)= ∅ | ja | Chomsky-Normalform |
| ETM | <M> | L(M)= ∅ | nein |  |
| EQDEA | <A1,A2> | L(A1)=L(A2) | ja | Vergleich der minimalen Automaten |
| EQCFG | <G1,G2> | L(G1)=L(G2) | nein |  |
| EQTM | <M1,M2> | L(M1)=L(M2) | nein |  |
| ADEA | <A,w> | 𝑤∈𝐿(𝐴) | ja | Regex-Engines simulieren beliebige DEAs auf beliebigen Input-Wörtern w |
| ACFG | <G,w> | 𝑤∈𝐿(𝐴) | ja | Cocke-Younger-Kasami deterministischer Parse-Algorithmus |
| ATM | <M,w> | 𝑤∈𝐿(𝐴) | nein | Halteproblem |

## Satz von Rice

**Grobe Aussage:**

Ist P eine nichtriviale Eigenschaft Turing-erkennbarer Sprachen, dann ist   
*PTM= {<M> | L(M) hat Eigenschaft P}* nicht entscheidbar

**Voraussetzung**:

keine Einschränkungen machen, wie Zahlengrösse Reduzieren

**Definition nichtrivial:**

Eine Eigenschaaft P Turing-erkennbarer Sprachen heisst nichtrivial, wenn es zwei Turing-Maschinen M1 und M2 gibt mit: L(M1) hat die Eigenschaft, L(M2) nicht.

**Folgerung:**

Es ist nicht möglich, einem Programm anzusehen, ob die akzeptierte Sprache eine nichttriviale Eigenschaft hat.

### Lösunganleitung einer Prüfungsfrage:

1. Nichttriviale Eigenschaft P aufschreiben
2. Die beiden Sprachen L(M1) und L(M2) bilden
3. Gibt es ein Programm, welches beide Sprachen erkennen kann? Sind beide Sprachen Turing erkennbar?
4. Dann besagt der Satz von Rice, dass die Sprache nicht entscheidbar ist

**Beispiel**

PPRIMES = "Sprache besteht nur aus den Primzahlen"

𝐿0={42} 𝐿1={𝑃𝑟𝑖𝑚𝑧𝑎ℎ𝑙𝑒𝑛} =>𝑇𝑢𝑟𝑖𝑛𝑔 𝑒𝑟𝑘𝑒𝑛𝑛𝑏𝑎𝑟

Satz von Rice greift da, so PPRIMES nicht entscheidbar

*Alternativ:*

1. Reduktion auf Problem mit Erklärung
   1. Halteproblem
   2. Leerheitsproblem
   3. Regularitätsproblem
2. Problem ist nicht entscheidbar
3. Satz von Rice als zusätzliche Begründung

## Reduktion

Mittels Reduktion kann bewiesen werden, dass eine TM nicht entscheidbar ist. Dazu reduziert man eine TM auf eine nicht entscheidbares TM (ETM).

Idee: Neues Problem schaffen, welches allenfalls einfacher zu lösen ist.

Berechenbare Abbildung:

𝑤 ∈𝐴< = >𝑓(𝑤)∈𝐵

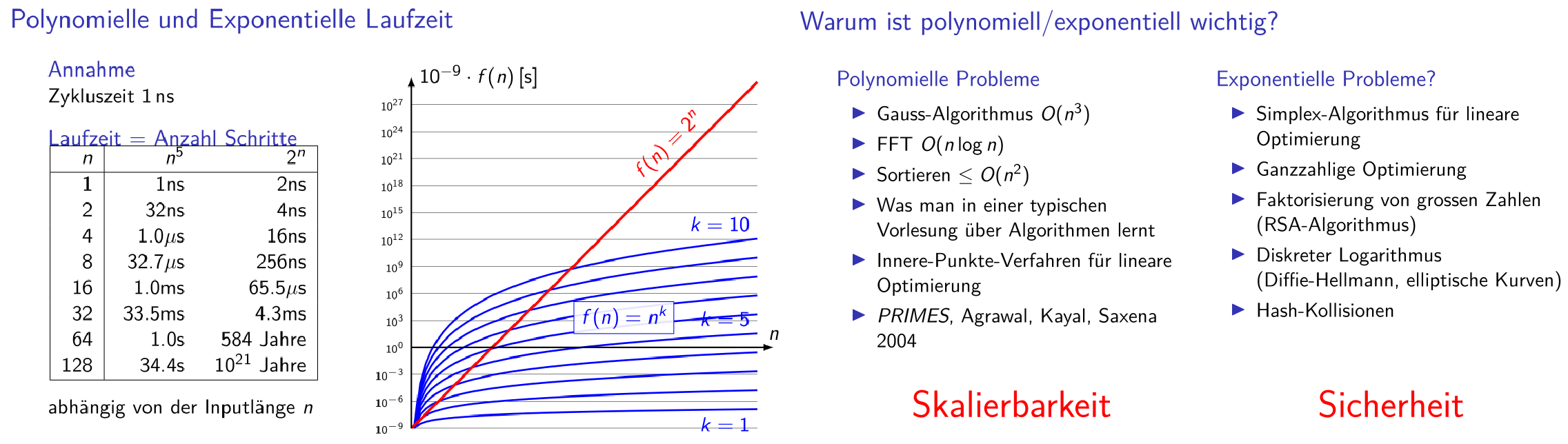
Ist B entscheidbar, 𝑓:𝐴≤𝐵 🡺 A entscheidbar

Ist B nicht entscheidbar, 𝑓:𝐴≤𝐵 🡺 A nicht entscheidbar

# Beispiel Komplexität

Eine Turingmaschine mit mehreren Bändern und Laufzeit t(n) kann in Laufzeit O(t(n)2) auf einer Standard-Touring-Maschine simuliert werden.

Sei N aber eine Nichtdeterministische Touring-Maschine, dann kann diese in Laufzeit 2O(t(n)) simuliert werden. (Gilt für NTM die auch ein Entscheider sind.

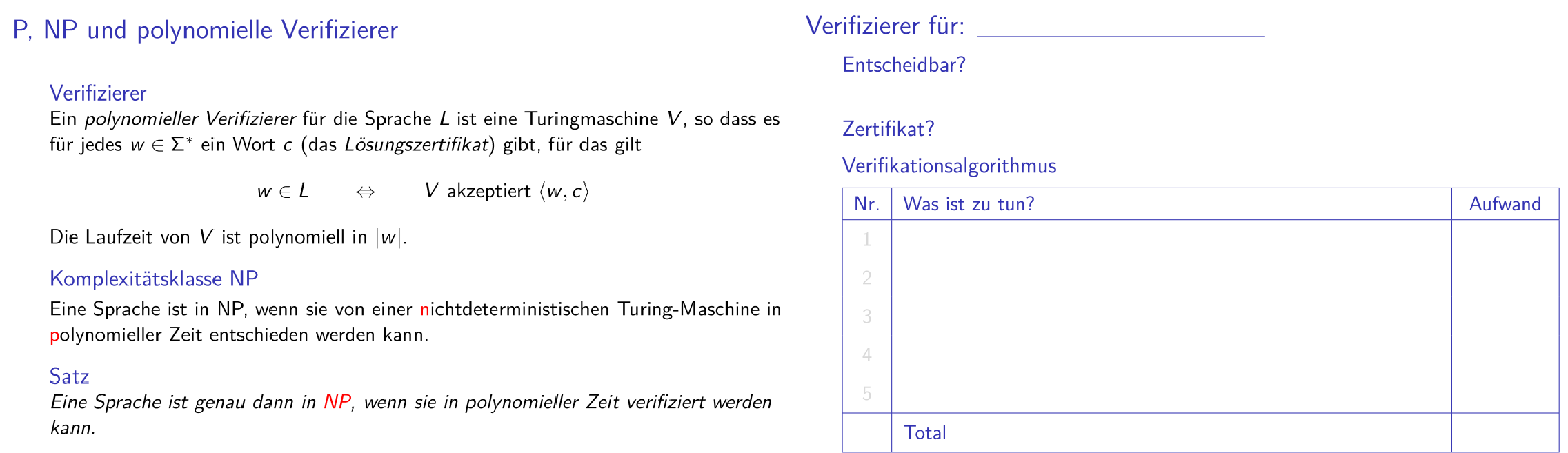


## Klassen P und NP

Die *Klasse P* besteht aus den Sprachen, die mit einem Entscheider (DTM) in polynomieller Laufzeit entschieden werden können. Ist eine Teilmenge von NP.

Die Klasse NP besteht aus Sprachen, die mit einer nichtdeterministischen Turingmaschine (NTM) in polynomieller Laufzeit entschieden werden können. Insbesondere enthält NP alle Sprachen mit einem polynominellen Verifizierter.

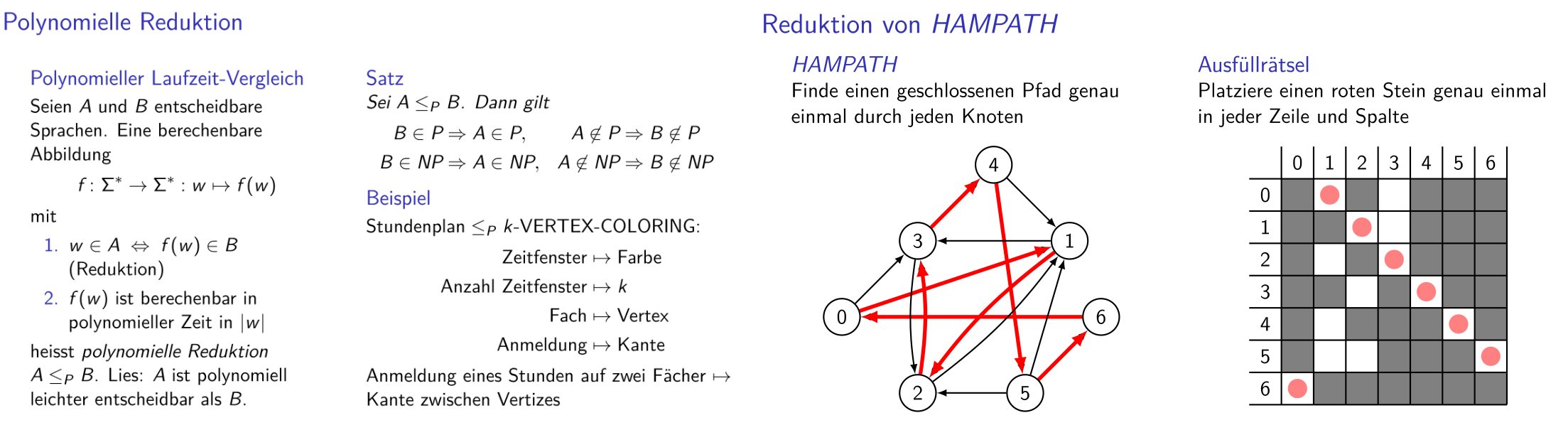
## Verifizierer ( Gleichbedeutend mit NP)



|  |  |
| --- | --- |
| **Beispiel Sudoku**  Regeln:  Jede zahl von 1-9 darf nur 1x vorkommen in jeder:   * Spalte * Zeile * Kleinem Quadrat | **Beispiel NURIKABE** |

## Reduktion

Dient dazu zwei Probleme zu vergleichen.



# Mathematik: Was es mit P ≠ NP auf sich hatNP-Vollständig

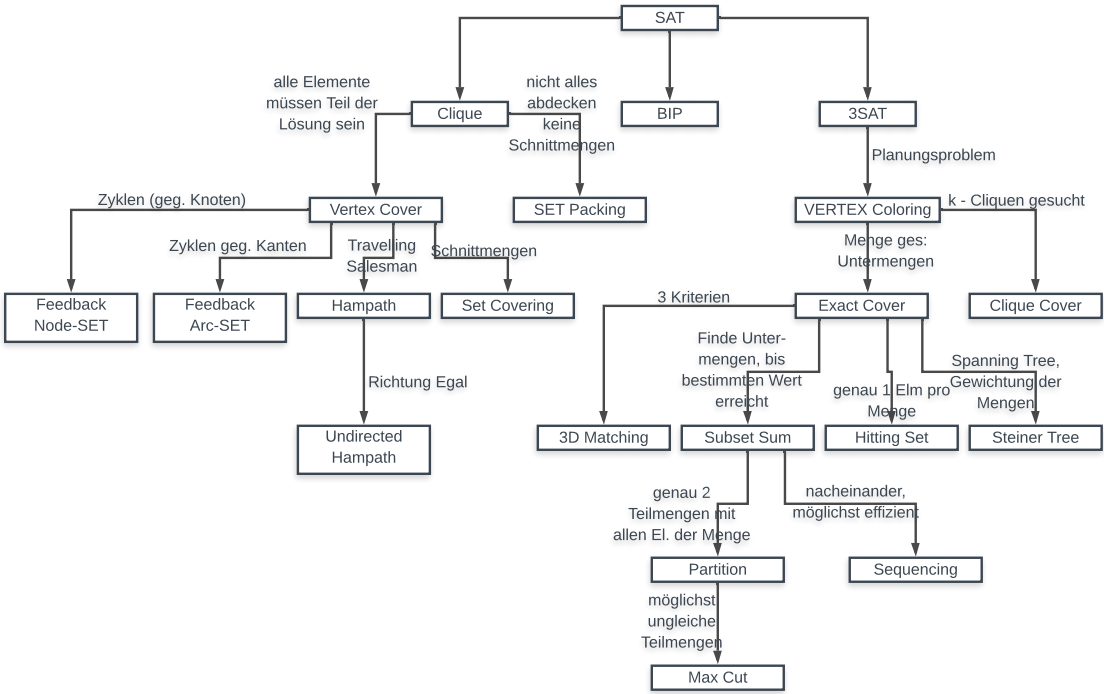
Eine Entscheidbare Sprache B heisst NP-vollständig, wenn sich jede Sprache A in NP polynomiell auf B reduzieren lässt. Sind die schwersten Probleme in NP (Katalog von Karp)

# Katalog von Karp

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***Probleme mit K Zahlen:***   * k-CLIQUE * VERTEX-COLORING * VERTEX-COVER * FEEDBACK-\*-SET * SET-COVERING * STEINER-TREE * (MAX-CUT) * SET-PACKING * SEQUENCING | ***Aufteilung zwei Teilmengen:***   * PARTITION * MAX-CUT   ***Probleme mit Zahl 3:***   * 3D-MATCHING * 3SAT   ***Unterschiede HITTING-SET, EXACT-COVER:***   * Hitting-Set: Menge von Punkte * Exact-Cover: Menge von Teilmengen | | | | ***Unterschiede FEEDBACK-\*-SET:***   * NODE 🡪 Vertex entfernen * ARC 🡪 Kanten entfernen   ***Unterschiede SET-\****   * COVERING 🡪 endliche Familie endlicher Mengen * PACKING 🡪 eine Familie von Menge   ***Unterschiede \*-COVER***  Graph oder Mengen mit einer Zahl k |
| **SAT (LOGIK)** | | | **3SAT (LOGIK)** | | |
| Gibt es eine mögliche Lösung der Booleschen Gleichung, damit sie TRUE ergibt.  3SAT: Klausel besteht höchstens aus 3 Literalen pro Klausel und ist in KNF  Klausel = Klammerausdruck, Literale = Variable | | | | | |
| Bsp: Elektriker:   * n-Schalter 🡪 n-Variablen * m-Räume 🡪 m-Klauseln * Überall Licht 🡪 Aussage = true * Stromkreise 🡪 Wertebereich (True, False) | | | | Jedes Ausfüllrätsel ist mit SAT beschreibbar, Man kann Wahrheitstabellen bilden. NP-Vollständig ist jedes Problem, dass sich auf SAT reduzieren lässt. | |
| **k-CLIQUE** | | | | | |
| Gibt es k Knoten, die alle miteinander verbunden sind? | | | | | |
| **SET-PACKING** | | | | | |
| Geg : eine Familie (𝑆𝑖)𝑖∈𝐼 von Mengen und eine Zahl 𝑘∈ℕ  Gibt es eine k-Elementige Teilfamilie (𝑆𝑖)𝑖∈𝐽 𝑚𝑖𝑡 𝐽⊂𝐼, d. |J|=k derart, dass die Menge der Teilfamilie paarweise disjunkt sind, also 𝑆𝑖∩𝑆𝑗=∅ ∀𝑖,𝑗∈𝐽 𝑚𝑖𝑡 𝑖≠𝑗  Gleich wie Exact-Cover nur Ausgangslage ist anders.  Einfach: Menge S und Menge T von Teilmengen. Gibt es k Teilmengen in T, welche disjunkt sind und die Menge S abdecken? | | | | | |
| The Algorithms Design Manual, Set and String Problems - nitishpuri.github.ioBeispiel 1 Küche: In einer Küche hat man eine Menge an Zutaten und ein Rezeptbuch voller Rezepte. Nun möchte man möglichst viele der Rezepte kochen, ohne eine Zutat mehrmals zu verwenden.  Beispiel 2 Medizinstudie:Für eine medizinische Studie ist eine grosse Zahl von Probanden rekrutiert worden. Sie sind bereits auf Allergien getestet worden, man weiss also von jedem Probanden, auf welche Allergene (Pollen, Katzenhaare, Hausstaub, Lactose,. . . ) er allergisch reagiert. Die Untersuchung soll sich auf eine Teilmenge von k = 17 oder noch mehr ausgewählten Allergenen beschränken, die so beschaffen ist, dass kein Proband auf mehr als eines der ausgewählten Allergene reagiert. Es stellt sich als schwierig heraus, eine solche Teilmenge zu finden. Warum?  *Allergene 🡨🡪 I*  *auf Allergen i allergische Probanden 🡨🡪 Si*  *ausgewählte Allergene 🡨🡪 J*  *Ausschlussbedingung zwischen Allergenen i und j <-> Si ∩ Sj = ∅*  *Es wird verlangt, k Allergene auszuwählen, also eine Teilmenge J ⊂ I mit |J| = k zu finden.* | | | | | |
| **VERTEX-COVER** | | | | | |
| Geg: Ein Graph G und eine Zahl k  Gibt es eine Teilmenge von k Knoten so, dass jede Kante des Graphen ein Ende in dieser Teilmenge hat? | | | | | |
| Bsp: Gibt es 4 Knoten, so dass jeder andere Knoten eine Kante zu diesem hat?  Bsp: Ein Verkehrsnetz soll regelmässig durch Mitarbeiter kontrolliert werden, die ihre Basis an einzelnen Knotenpunkten des Netzes haben. Kann man auf effiziente Art herausfinden, an welchen Knotenpunkten man Kontrolleure stationieren muss, damit jede Strecke in einem Konten mit Kontrolleur endet?  *Knotenpunkte 🡨🡪 Knoten*  *Strecke 🡨🡪 Kante*  *Anzahl Kontrolleure 🡨🡪 k* | | | | | |
| **FEEDBACK-NODE-SET** | | | | | |
| Geg: Ein gerichteter Graph G und eine Zahl k  Gibt es eine endliche Teilmenge von k Vertizes/Knoten von G, so dass jeder Zyklus in G einen Vertex in der Teilmenge enthält.  Es gibt mehrere Buslinien. Wo muss das Putzpersonal platziert werden, damit alle Linien geputzt werden können. Man möchte möglichst wenig Personal einsetzen. | | | | | |
| **FEEDBACK-ARC-SET** | | | | | |
| Geg: Ein gerichteter Graph G und eine Zahl k  Gibt es eine endliche Teilmenge von k Kanten von G, so dass jeder Zyklus in G eine Kante aus der Teilmenge enthält. | | | | | |
| **HAMPATH & UHAMPATH** | | | | | |
| Ein Hamilton-Pfad in einem gerichteten Graphen ist ein Pfad, der jeden Knoten genau einmal enthält. 🡪 Haus des Nikolaus.   * Komme zu jedem Knoden, jedoch nur einmal auf jeden Knoten.   **Ungericht ist es der UHAMPATH**. Bsp.: Ganze Schweiz bereisen ohne eine Stadt mehrmals besuchen | | | | | |
| **SET-COVERING** | | | | | |
| Geg: eine endliche Familie endlicher Mengen (𝑆𝑗)1≤𝑗≤𝑛 und eine Zahl k  Gibt es eine Unterfamilie bestehend aus k Mengen, die die gleiche Vereinigung hat?  Einfach: Kann man k Teilmengen bilden, welche die Menge S komplett abdecken.  Bsp.:k=3 => D &E & A oder B oder C  A={1,3,5,11,19}  B={1,3,7,9,17}  C={1,3,21,29,27}  D={1,5,21,27,29}  E={1,9,11,21,31}  Beispiel Ältestenrat: Von einem uralten Planetensystem soll ein Ältestenrat gebildet werden. Jedoch kann nicht von jedem Stamm (wegen der Entfernung) ein Mitglied bestimmt werden. Darum wurde beschlossen das Mitglieder nur einen einzigen Tropfen Blut eines Stammes in sich haben muss um Mitglied zu werden (weit weit entfernter Vorfahre muss Mitglied eines Stammes gewesen sein).  *Zu jedem Bürger i ist die Menge Si aller Stämme bekannt, von denen er Blut in sich trägt. Die Vereinigung aller Si ist die Familie "U" aller Stämme. Gefragt wird jetzt nach k Bürgern sodass alle Stämme abgedeckt werden.* | | | | | |
| **VERTEX-COLORING (Planungsprobleme)** | | | | | |
| Graph coloring - WikipediaKann man die Knoten so mit k Farben einfärben, dass benachbarte Knoten verschiedene Farben haben?  Bsp: Job-Planung  - n Job 🡪 Knoten,  - m gleiche Ressourcen 🡪 Verbindungen  - parallele Jobs (bzw. Interval) 🡪 gleiche Farbe  - N Intervalle 🡪 n Farbe | | | | | |
| **EXACT-COVER** | | | | | |
| Geg: eine Familie (𝑆𝑗)1≤𝑗≤𝑛 von Teilmengen einer Menge U  Gibt es eine Unterfamilie von Mengen, die disjunkt sind, aber die gleiche Vereinigung haben?  Die Unterfamilie (𝑆𝑗𝑖)1≤𝑖≤𝑚 muss also  Einfach: Jedes Element in U, soll genau in einer der Teilmengen in der Familie S vorkommen. Die gesuchte Menge bildetn eine exakte Überdeckung  U={a,b,c,d,e,f}, S={(a,b), (a,b,c), (c,e), (d,f), (e,f)}   Student Xaver Tecco soll im Rahmen einer Big-Data-Studienarbeit die Kunden einer grossen Shop-Website untersuchen und klassifizieren. Es steht eine grosse Zahl von binären Eigenschaften zur Verfügung, zum Beispiel ob Kunden ein bestimmtes Produkt gekauft haben, oder ob ein Kunde nur im Dezember einkauft. Herr Tecco soll herausfinden, ob es eine Teilmenge von Kriterien derart gibt, dass jeder Kunde genau eine der Eigenschaften hat. Die Abgabe der Arbeit steht in zwei Tagen bevor, und er hat noch keinen funktionierenden Algorithmus. Muss er sich Sorgen machen?  *Eigenschaft 🡨🡪 Menge Sj*  *Teilmenge von Eigenschaften 🡨🡪Unterfamilie Sji*  *Genau eine der Eigenschaften 🡨🡪*  *Alle Kunden erfasst 🡨🡪* | | | | | |
| **3D-MATCHING** | | | | | |
| Geg: Endliche Menge T und eine Menge U von Tripeln TxTxT  Gibt es eine Teilmenge W von U so, dass |W| = |T| und keine zwei Elemente von w stimmen in irgendeiner Koordinate überein?  Ein Koch hat je n Rezepte für Vorspeisen, Hauptspeisen und Desserts. Nicht alle Vorspeisen lassen sich mit jeder Hauptspeise kombinieren, dasselbe gilt auch für Desserts. Damit jedes seiner Rezepte regelmässig zum Einsatz kommt, möchte der Koch eine Folge von n Menus zusammenstellen, so dass jedes Rezept in genau einem der Menus vorkommt. Nach längerem tüfteln gibt er jedoch frustriert auf. Können Sie erklären, warum ihm die Menugestaltung so schwer gefallen ist.  *Nummern der Rezepte 🡨🡪Menge T*  *Menuzusammenstellungen 🡨🡪Tripel aus TxTxT*  *Menge der Möglichen Menüzusammenstellung 🡨🡪 Menge U*  *Gesuchte Menü's n 🡨🡪 Teilmenge W, so n=|T|=|W|* | | | | | |
| **SUBSET-SUM (Rucksack-Problem)** | | | | | |
| Geg: Menge S von ganzen Zahlen  Kann man eine Teilmenge finden, die als Summe einen bestimmten Wert t hat?  Bsp. Rucksack-Problem  Grösse der Gegenstände 🡪 Zahl  Menge der Gegenstände 🡪 Menge S der Zahlen  Grösse des Rucksacks 🡪 Wert t (Summe)  Gegenstände in Rucksack 🡪 Teilmenge | | | | | |
| **SEQUENCING** | | | | | |
| Geg: ein Vektor ∈ ℤ𝑝 von Laufzeiten von p Jobs, ein Vektor ∈ ℤ𝑝 von spätesten Ausführzeiten, ein Strafenvektor ∈ ℤ𝑝 und eine positive Zahl k  Gibt es eine Permutation der Zahlen 1,…,p, so dass die Gesamtstrafe für verspätete Ausführung bei der Ausführung der Jobs nicht grösser ist als k?  Einfach: Geg.: eine Menge an Jobs, pro Job eine Ausführungszeit, Deadline und eine Strafe. Sowie eine maximale Strafe von k. Die Jobs müssen sequenziell abgearbeitet werden. Wird ein Job zu spät fertig, muss eine Strafe gezahlt werden.  Ges.: Eine Reihenfolge von Jobs so, dass die Strafe kleiner gleich k ist.  Eine Firma hat eine bestimmte Anzahl laufende Verträge. Der Firma ist es nicht möglich alle Verträge in einer bestimmten Zeit abzuarbeiten. Sie versucht also Schadensbegrenzung zu machen, indem sie möglichst viele Verträge in der verbleibenden Zeit abarbeitet die eine hohe Strafe zur Folge haben. | | | | | |
| **PARTITION** | | | | | |
| Geg: Eine Folge von S ganzen Zahlen c1, c2, …, cs  Kann man die Indizes 1,2,…,S in zwei Teilmengen 𝐼 und teilen, so dass die Summe der zugehörigen Zahlen 𝑐𝑖 identisch ist:  Einfach: Gibt es zwei disjunktive Teilmengen mit der gleichen Summe?  Bsp: eine Reihe von Wassergläsern unterschiedlich gefüllt. Es sollen 2 Behälter gleich viel mit den Gläsern gefüllt werden. Welche Gläser müssen in welche Behälter geleert werden. | | | | | |
| **MAX-CUT** | |  | | | |
| Geg.: ein Graph G mit einer Gewichtsfunktion 𝑤:𝐸→ℤ und eine Zahl W  Gibt es eine Teilmenge S der Vertizes, so dass das Gesamtgewicht der Kanten, die S mit seinem Komplement verbinden, mindestens so gross ist wie W.  Einfach: Der Max-Cut eines Graphen ist eine Zerlegung seiner Knotenmenge in zwei Teilmengen, so dass das Gesamtgewicht der zwischen den beiden Teilen verlaufenden Kanten mindestens W wird.  Feindliche Übernahme einer Firma, mit resultierender Aufteilung der Abteilung, dass diese möglich ineffizient miteinander kommunizieren können.  *Abteilung 🡨🡪 Vertex*  *Kommunikationsbeziehung 🡨🡪Kante*  *Kommunikationsvolumen 🡨🡪 Gewicht einer Kante* | | | | | |
| **HITTING-SET** | | | | | |
| Geg: eine Menge von Teilmengen 𝑆𝑖⊂𝑆  Gibt es eine Menge H, die jede Menge in genau einem Punkt trifft, also |𝐻∩𝑆𝑖|=1  Geg.: A={1,2,3}, B={1,2,4}, C={1,2,5}  Ges.: H={3,4,5} | | | | | |
| **STEINER-TREE** | | | | | |
| Geg: ein Graph G, eine Teilmenge R von Knoten und eine Gewichtsfunktion und eine positive Zahl k.  Gibt es einen Baum mit Gewicht <= k, dessen Knoten in R enthalten sind? Das Gewicht des Baumes ist die Summe der Gewichte über alle Kanten im Baum.  Beispiel: Bau einer Zugstrecke oder Stromnetzes:  Stromnetz/Zugstrecke 🡨🡪 STEINER-TREE  *Ortschaften 🡨🡪 Knoten*  *zu erschliessende Ortschaften 🡨🡪Knoten in R*  *Baukosten 🡨🡪 Gewicht w einer Kante*  *Budget 🡨🡪 maximales Gewicht k* | | | | | |
| **CLIQUE-COVER (EXOR mit bestimmter Zahl k)** | | | | | |
| Geg: ein Graph G und eine Positive Zahl k  Gibt es k Cliquen so, dass jede Ecke in genau einer der Cliquen ist?  Bsp.: Für eine Gruppenarbeit sollen k Gruppen gebildet werden. Um die Zeit für das Kennenlernen möglichst kurz zu halten, sollen sich die Leute einer Gruppe bereits kennen. Alle Leute sollen beschäftigt sein.  *Teilnehmer 🡨🡪 Knoten*  *Kennen sich 🡨🡪 Kante*  *Anzahl Gruppe 🡨🡪 k*  *Gruppe 🡨🡪 Clique* | | | | | |
| **BIP (binary integer programming)** | | | | | |
| Zu einer ganzzahligen Matrix C und einem ganzzahligen Vektor d, ist ein binärer Vektor x zu finden mit C\*x = d | | | | | |

***Vorgehen bei Prüfungen:***

1. Problem aus Liste suchen (Dabei Punkte beachten, wie: Ist es ein Mengenproblem, Planungsproblem, kommen k-Zahlen vor, Teilmengen?)
2. Reduktion des Problems auf Karp mit Pfeil-Liste darstellen (Eigenschaften)
3. Zusammenfassung Schreiben: «Problem x ist äquivalent mit Karp-Katalog-Problem x und deshalb gehört es zu den NP-Vollständigen Problemen und daher ist es nicht einfach entscheidbar (Es gibt keinen polynominellen Algorithmus derzeit).»



# Turing-Vollständigkeit

Turing-Maschine und moderne Computer haben einige Gemeinsamkeiten. Er erfüllt alle Eigenschaften einer Turing-Maschine (Zustände, Band, Schreib-/Lesekopf/Anhalten). Einzig kann er mehrere Programme ausführen und eine Turing-Maschine ist Problemspezifisch.

Ist sie das wirklich? NEIN, es gibt die Universelle Turing-Maschine mit..

* …eigenes Band für Codierung der Übergansfunktionen
* …eigenes Band für aktuellen Zustand
* …Arbeitsband

Eine solche Mehrbandmaschine kann auch auf einer Standard-Turingmaschine simuliert werden.

Es gibt aber Komponenten die ein PC hat, aber eine Touring-Maschine nicht. Doch sind diese wesentlich (Verändern Fähigkeiten einer TM nicht-polynomiell)?NEIN sind sie nicht:  
**Persistenter Speicher:** Files nicht unterscheidbar von Daten, für TM egal  
**Interaktion:**  Lösung eines spezifizierten Problem braucht keine Interaktion

**Input/Output:** Output ist nicht wesentlich. Input «existiert nicht». (Wenn TM  
angehalten wird, kann Kernel Band «manipulieren» und TM sieht nun geänderter Input, weiss aber nicht ob es nun selbst gemacht hat oder von aussen kommen. Also nicht wesentlich.

## Vergleich von Maschinen

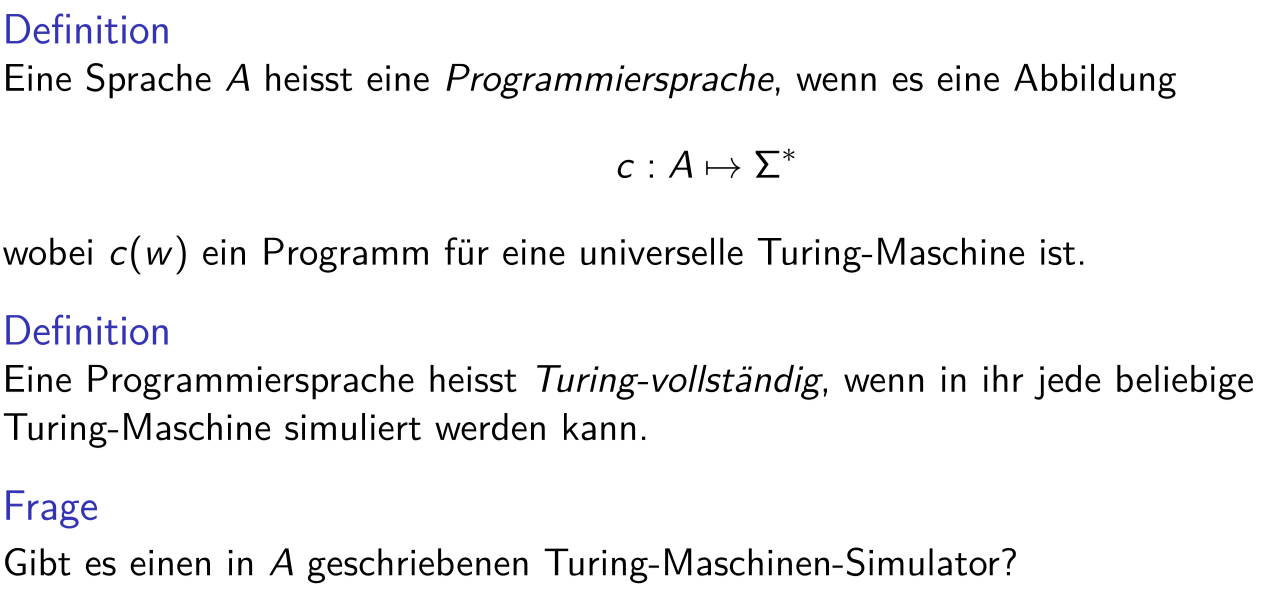
Eine TM *M1* ist «leistungsfähiger als eine TM *M2*, wenn *M1* die Maschine *M2* simulieren kann.

(M2 ist simulierbar auf M1)

## Programmiersprachen

Frage: Ist eine Programmiersprache Turing-Vollständig?

* Turing-Vollständig bedeutet das Programme ebenso mit Halteproblem, Rice und NP konfrontiert sind. So kann man nicht sagen ob ein Programm anhält, akzeptabler Input eine bestimmte Eigenschaft hat, ob Spezifikationen erfüllt sind und NP-Vollständige Probleme haben exponentielle Laufzeit.

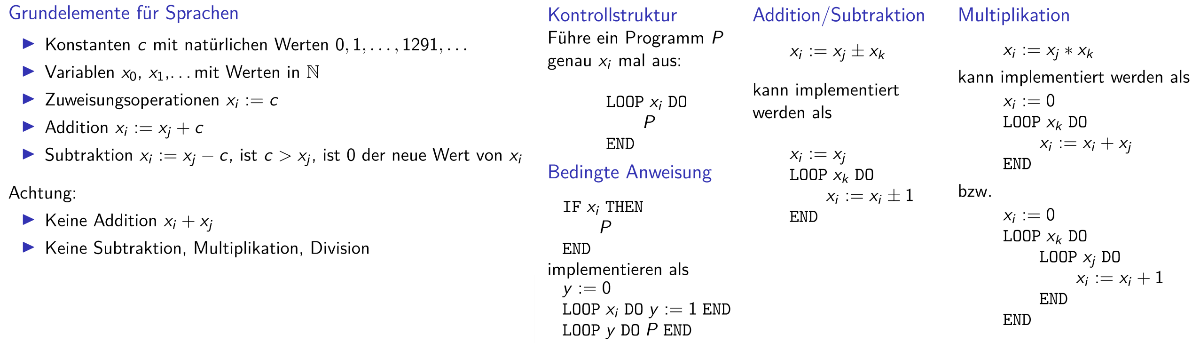


Beispiele für Turing-Vollständige Sprachen:

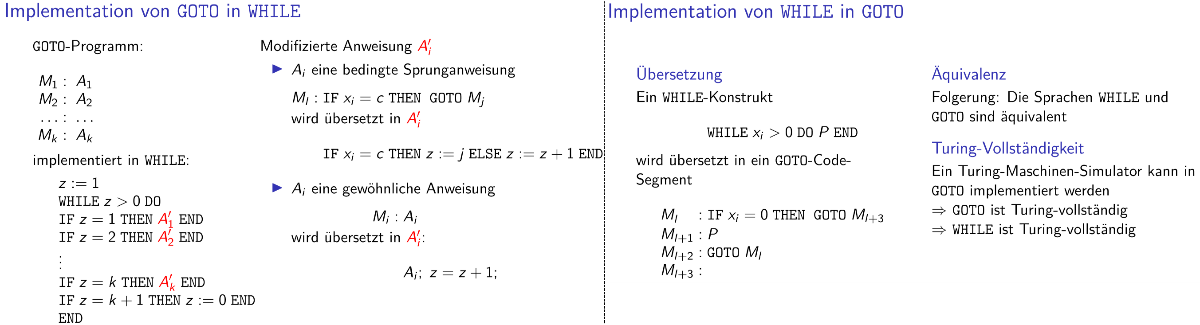
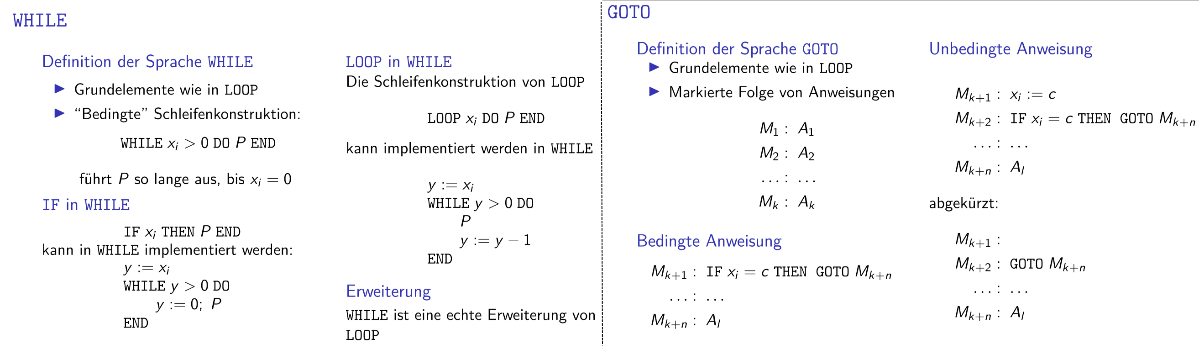
* Maschinennahe Programmiersprache *C*
* *Javascript* ist ebenso Turing-Vollständig weil es ein Programm gibt in JS, dass Linux und auch den C-Compiler simulieren kann und dann ausführen.
* *LATEX* und *XSLT* ebenso Turing-Vollständige Sprachen.

### LOOP

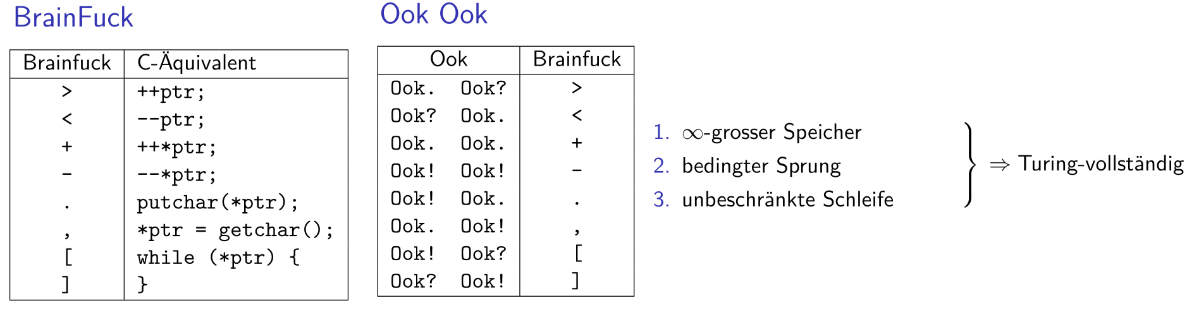
Ist nicht Turing-Vollständig, da Programme immer terminieren. (Lässt sich beweisen mit vollständiger Induktion)



## WHILE und GOTO

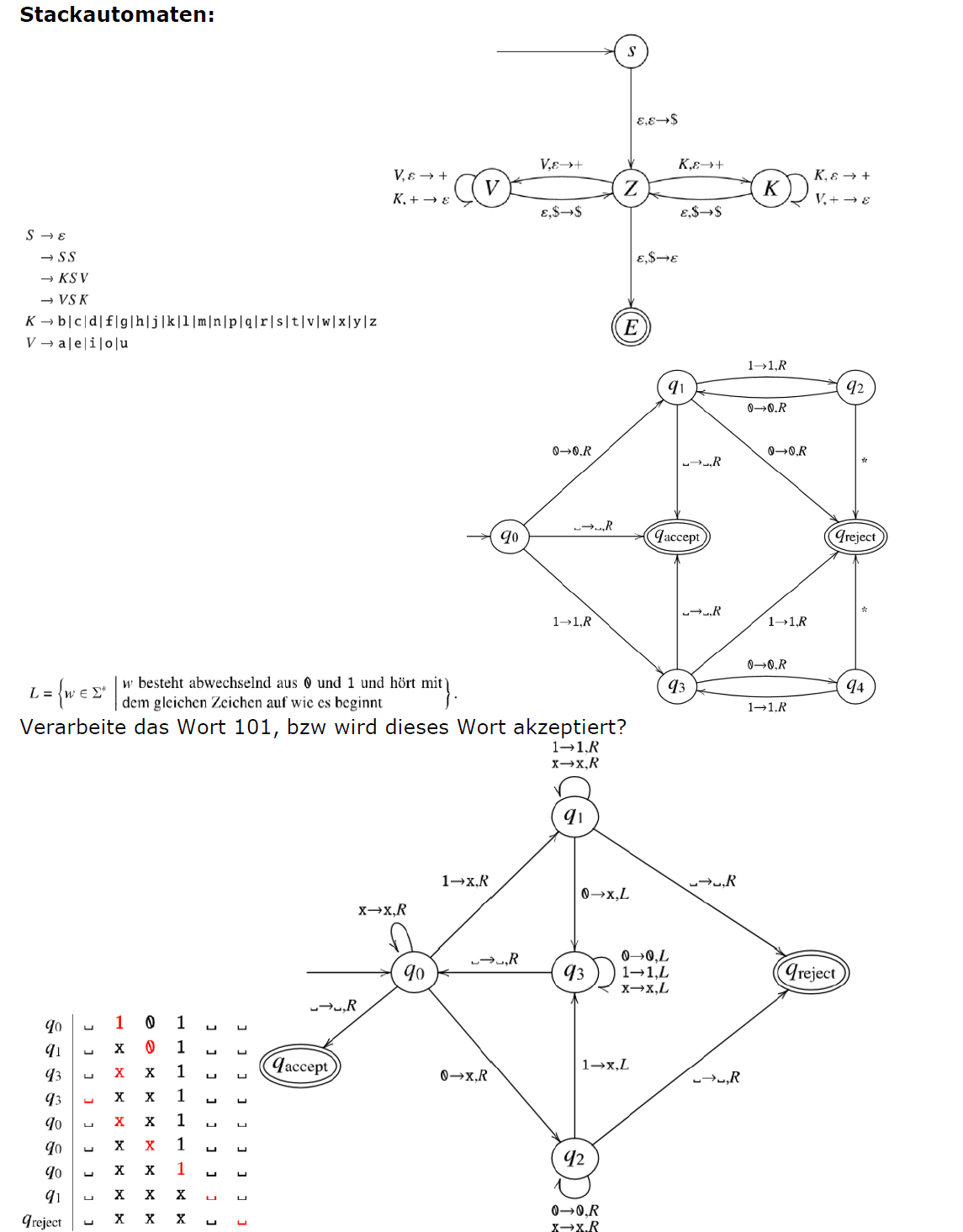
Turing-Vollständig und Äquivalenz 

## Brainfuck und Ook Ook

Eine sehr primitive Sprache reicht aus für Turing-Vollständigkeit.(Auch PowerPoint ist Turing-Vollständig) 

# Diverse alte Aufgaben

## Stackautomaten und Turing-Maschine



## Pumping Lemna Kontextfrei K

## Turing-Maschine für binäres Additionsprogramm Ein Bild, das Text enthält. Automatisch generierte Beschreibung

## Logik-Rätsel Fillomino

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Kreuzworträtsel enthält.

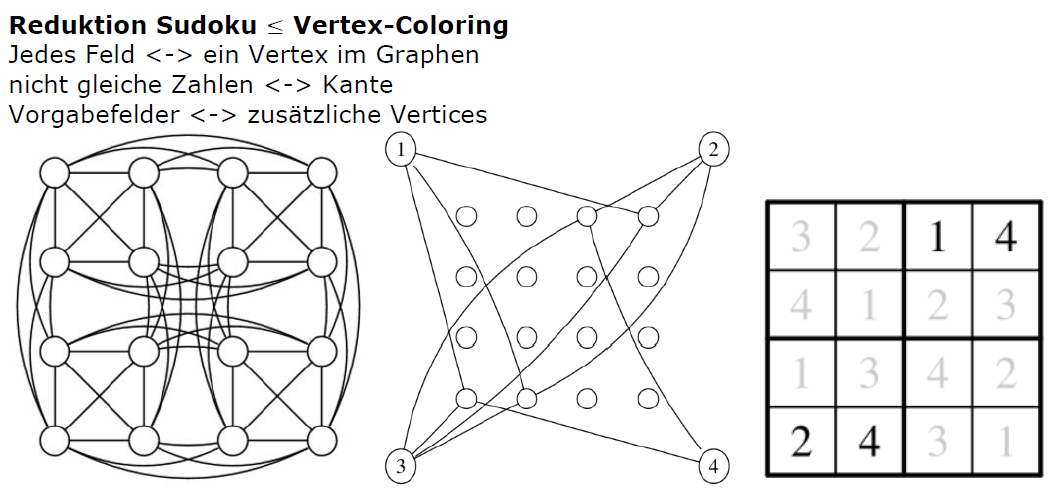
Automatisch generierte Beschreibung

### Verifizierer

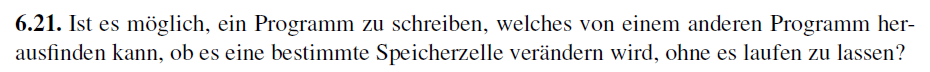
Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

## Reduktion Sudoku-Vertex-Coloring



## Entscheidbarkeit (Reduktion Halteproblem, RICE)

 Ein Bild, das Text, Zeitung, Dokument enthält.

Automatisch generierte Beschreibung Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

## Sprachproblem

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

## Abzählbar oder nicht?

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung Ein Bild, das Text enthält.

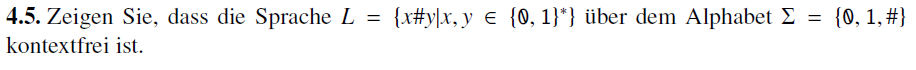
Automatisch generierte BeschreibungEin Bild, das Text enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

## Kontextfreie Grammatik

 Ein Bild, das Text enthält.

Automatisch generierte Beschreibung Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

## Grammatik in Chmosky-Normalform

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung