**ETI Notizen**

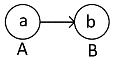
**Begriffe erklären**

**Alphabet**: Endliche Menge von Zeichen.

**Buchstabe**: Element eines Alphabets.  
**Wort**: Eine endliche Folge von Buchstaben.

**Sprache** über einem Alphabet Teilmenge von \*

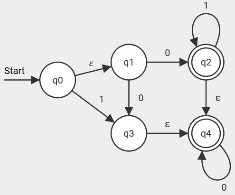
**Grammatik**: Hilfsmittel um Sprache zu beschr.

**Funktion**   
**total**: Abbildung in B für jedes A  
**funktional**: Nur eine Abb pro A  
**injektiv**: Zwei A nicht dieselbe Abb in B  
**surjektiv**: Zu jedem B führt eine Abb  
**bijektiv**: 1zu1 surjektiv&injektiv

**Klasse der Sprachen**Grammatik L(G): beste Regel gewinnt 🡪0  
Sprache: schlechteste Regel gewinnt  
Klasse der Grammatik: schlechteste Regel gewinnt 🡪3

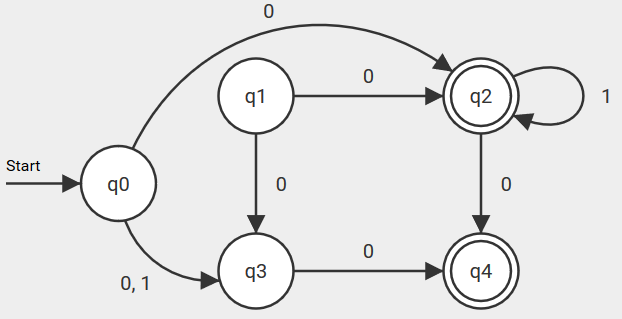
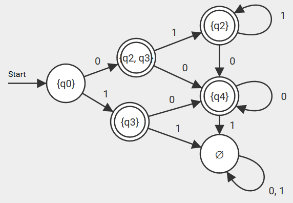
|  |  |  |
| --- | --- | --- |
| Kl. | Bezeichnung | Bedingung |
| 0 | allgemein | keine |
| 1 | kontextsensitiv | aCb |
| 2 | kontextfrei | A+  Immer ein Nichtterminalsymbol  gefolgt von einem Terminalsymbol |
| 3 | regulär | B nicht länger als 2 |

Spezialfall: S regulär  
 C allgemein

**-Hülle ohne -Übergang**Ausgangslage  
NFA

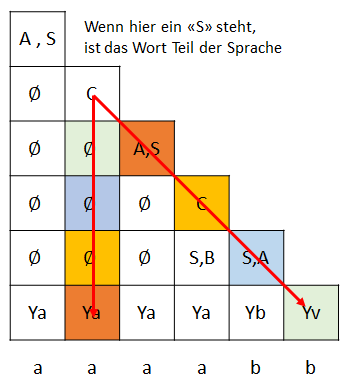
-Hülle eines NFA ohne -Übergang  
 = {q0, q1}  
 = {q1}  
 = {q2, q4}  
 = {q3, q4}  
 = {q4}(q[ ],[ ]) : = =

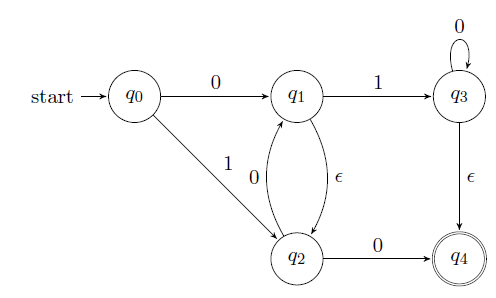
|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
| {q0} | {q2, q3} | {q3} |
| {q2, q3} | {q4} | {q2} |
| {q3} | {q4} |  |
| {q4} | {q4} |  |
| {q2} | {q4} | {q2} |
|  |  |  |

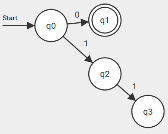
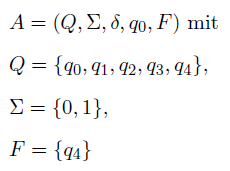
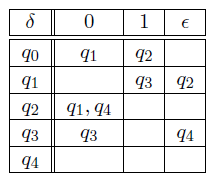
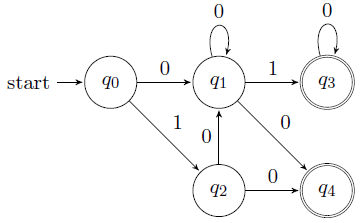
  
DFA daraus generieren

**Coche-Younger Kasami Algorithmus**

Vorbedingungen:  
- L ist kontextfrei  
- L =L(a) mit Grammatik in CNF



**Satz 2.3 NFA mit - Übergang zu ohne   
**

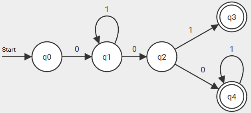
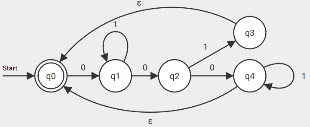
****  
- Hüllen der Zustände  
**Umwandlung NFA zu DFA**

B

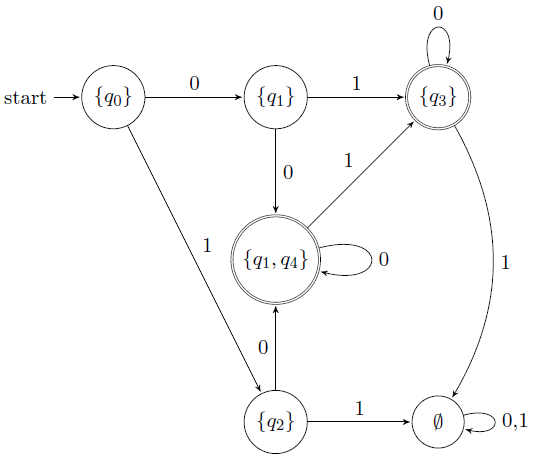
A

B

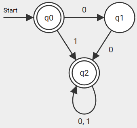
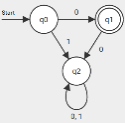
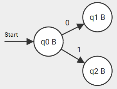
C

Was ist ein ?  
Def:   
Weg in Graphen, der bei q0 beginnt in einem akzeptierenden Zustand endet}  
**Satz 2.9 Rabin Scott**  
Voraussetzung: A  
Beziehung:   
Beweis durch Beispiel

C

**Endliche Automaten (DFA)**  
DFA ist ein 5-Tupel  
Wir haben hier immer eine Leere Menge  
Benötigt immer die Abzweigung 1 und 0  
  
Wenn danach eine Grammatik verlangt wird, können wir den Elias Algorithmus verwenden.

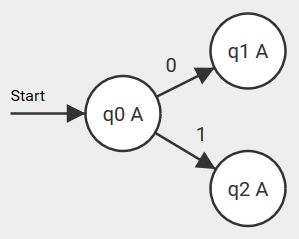
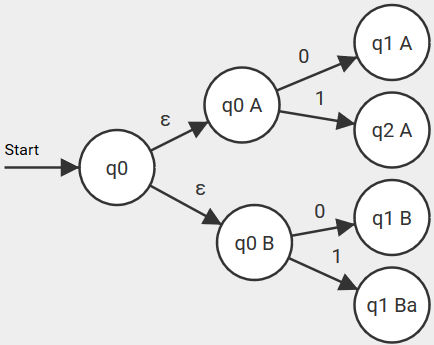
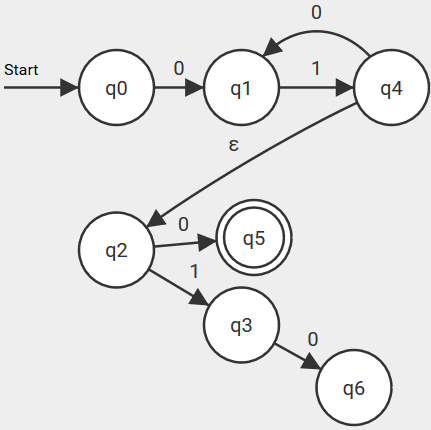
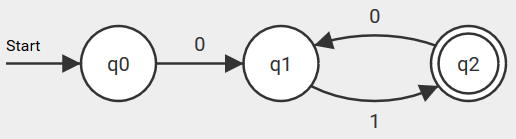
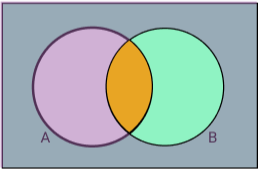
**2.2 Nicht deterministische endliche Automaten (NFA)**  
Hier kann man mit mehreren 0en oder 1er weg oder auch ohne wegführende Pfeile.  
Hat keine Leere Menge

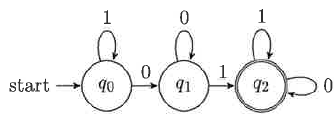
****Satz Variable L1, L2 regulär**Notation   
Behauptung:   
Beweisskizze:   
Idee: Wir konstruieren mit Hilfe von A und B einen NFA, , der die entsprechende Sprache akzeptiert

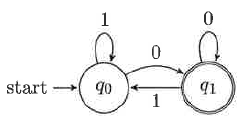
C

A

B

2.   
   
3. 
4. L={000}
5. 

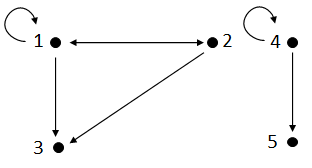
A

B

1

0

|  |  |  |  |
| --- | --- | --- | --- |
| C | B | | |
|  | 0 | | 1 |
| A | 0  1  2 |  |  |
| 1 | 0  1  0  0  0 |
|  | 1  0 |

**Mengenlehre Repetition**  
1) reflexiv (loops)  
2) symmetrisch (doppelpfeile)  
3) transitiv (abkürzungen)

**Nerode Relaltion**Die Nerode-Relation ist die Vergröberung des Automaten  
RL ist eine Äquivalenzrelation von der Partition von RA ist eine Äquivalenzrelation auf (Verfeinerung)  
RA ist die Teilmenge von RL  
Wir wollen die Äquivalenzklassen (Blöcke) von RL bestimmen.

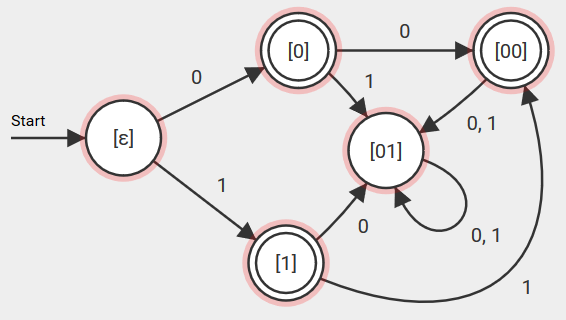
Beispiel:

Gegeben   
Äquivalenzklassen von L zu RL: und mit Begründung!

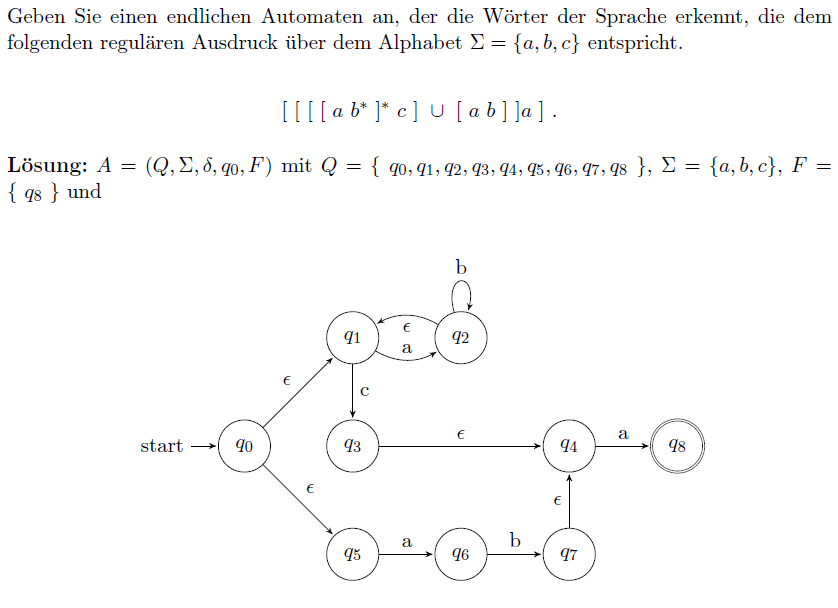
= { }  
[0] = {0}  
[1] = {1}  
[00] = {00,11}  
[01] = {w}

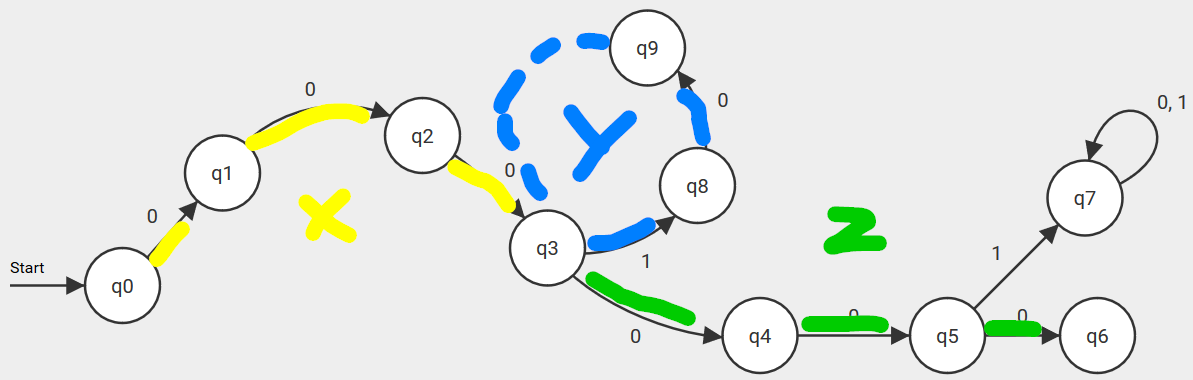
Tabelle und Graph zur Sprache L gehörigen Nerode Automaten

|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
|  | [0] | [1] |
| [0] | [00] | [01] |
| [1] | [01] | [00] |
| [00} | [01] | [01] |
| [01] | [01] | [01] |



**Kleene-Myhill**



**2.5 Das Pumping Lemma und Myhill-Nerode**Sei L eine reguläre Sprache   
Sei **Pumping Lemma ( als Beweis benutzen)**Voraussetzung:

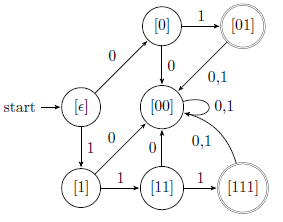
Das Pumping Lemma hilf, einer Sprache mit kontextfreier Grammatik zu entscheiden, ob es vielleicht eine reguläre Grammatik gibt oder nicht.

X&z können leer sein

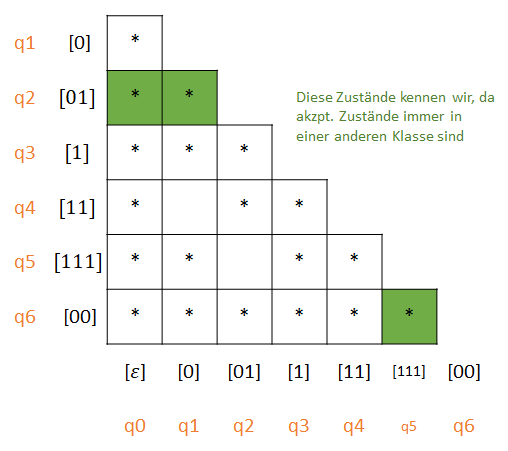
mit wenn Regel 4 wiederlegt wird, ist die Sprache nicht reg.

**Minimale Automaten**Die Sprache L3 = {01,111} besteht wie folgt  
mit

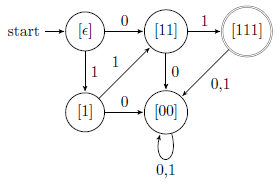
|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
|  | [0] | [1] |
| [0] | [00] | [01] |
| [01] | [00] | [00] |
| [1} | [00] | [11] |
| [11] | [00] | [111] |
| [111] | [00] | [00] |
| [00] | [00] | [00] |

Konstruiere den Graphen  
  
Zustände trennen

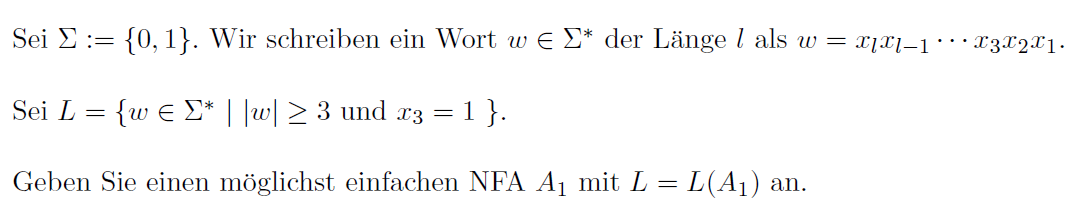
* Ein „Paar“ aus akzeptierend und nicht akzeptierend = \*
* Wenn ich aus zwei Zustände mit dem gleichen Wort an das gleiche Ort komme -> kein \*
* Wenn ich mit zwei Zustände z.B. mit null weitergehe und mit einem Zustand in einen akzept komme und mit dem anderen nicht, dann ist er nicht im selben Block =\*.
* ODER den Automat minimieren und die neu kreirten Zustände leer lassen und den Rest mit Stern versehen.

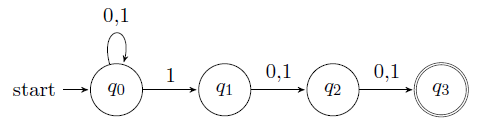
  
Es folgt [0] = [11] und [01] = [111]

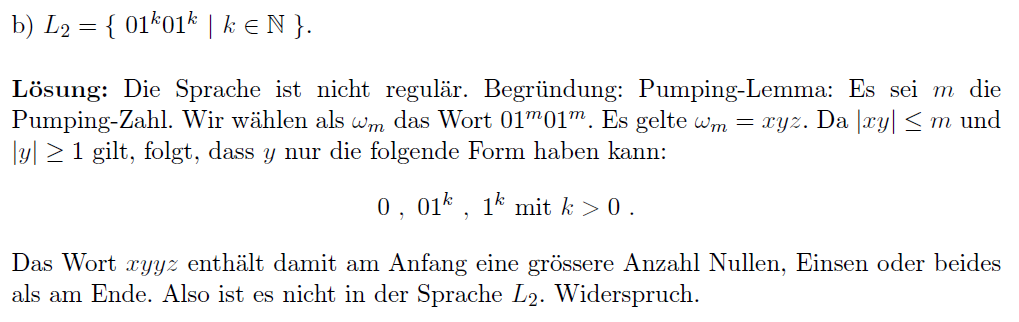
Der minimale Automat besitzt also nur 5 Zustände:  
[],[1].[11].[111] und [00] und einen akzeptierenden Zustand [111]



Beispiel aus w einen Automaten zeichnen



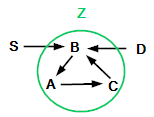


**Beispiel für Pumping Lemma**  


**Chomsky Normalform Lernkonrolle 6**

**0. Schritt**

**1. Schritt Elimination von -Regeln**  
   
 ASA | SA | AS

**2. Schritt Elimination von Kettenregeln**   
a) Kette zeichnen

b) Kette löschen

1. Alle NT in Kette in Z umwandeln

Alle Z auf eine Regel stellen, alleinstehende Z eliminieren, gleiche Regeln nicht aufschreiben

1. Z in NT-Regel einfüllen, welche nicht in Kette vorkommen  
   Da Z nicht mehr existiert, alleinstehende Z eliminieren

**3. Nutzlose Variablen eliminieren**

(die die nicht terminieren und die, die man nicht erreicht)  
alle Nichtterminalsymbole in w1 sind nicht terminieren. Tada! Nun wird D gestrichen.

Wenn es eine Regel gibt welche nicht terminiert wird die ganze Regel gelöscht  
**4. Abändern von Regeln der Form A**  
**w mit**  und Terminalsymbole in ihre Variablen ersetzen (Terminalsymbole durch entspr. NT ersetzen)

Alle Terminalsymbole mit ersetzen

**Komplexität**

Eine Komplexitätsklasse ist durch vier Parameter spezifiziert:

1. Das Berechnungsmodell: Die Art der Maschine, mit welcher   
 die Berechnung ausgeführt wird, z.B. 1-Band-TN, k-Band-TN… 2. Der Berechnungsmodus: Wie die Maschine zu ihrem Schluss   
 kommt, z.B. deterministisch oder nicht-deterministisch.

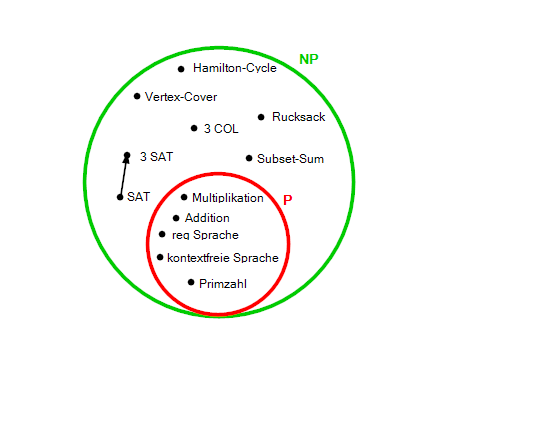
3. Die betrachtete Ressource, z.B. Zeit, Platz, Energie etc.  
 Time = Schritte, Space = Platz  
 DSPACE ist eine Teilmenge von DTIME

4. Eine Schranke für die betrachtete Ressource:   
 Wie viele Schritte, wie viel Platz darf die Maschine verwenden?   
 nk, 2n, nlogn etc.

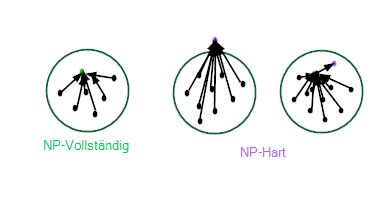
Klasse P  
Die Klasse P ist die Menge alles Sprachen L (Probleme), für die eine **deterministische** Turing-Maschine existiert, deren Zeitkomplexitätsfunktion polynomial ist, d.h. es existiert ein Polynom p mit

Klasse NP  
Die Klasse NP ist die Menge aller Sprachen L, für die es eine **nichtdeterministische** Turing Maschine gibt, deren Zeitkomplexitätsfunktion polynomial beschränkt ist.

Alle Sprachen in NP sind entscheidbar



NP-Vollständig und NP-Hart



- NP-Vollständig auch == NP-Hart!

-Wenn sich jedes Problem auf jedes Problem reduzieren  
 lässt, dann sind die Probleme alle gleich schwer

Schwierigkeit abschätzen, wenn man sich nicht sicher ist:

Problem NP-hart, d.h. es gibt wahrscheinlich keinen schnellen Algorithmus, um für eine beliebige natürliche Zahl n den Wert l(n) zu bestimmen.

Subset-Sum-Problem (NPC)

Satz 1.27 Die Klasse der NP-Entscheidungsprobleme ist genau die Klasse der in polynomialer Zeit mit einer nicht-deterministischen Turing-Maschine entscheidbaren Sprachen, d.h.

Vorhanden , nicht vorhanden , negiert

Erfüllbare Bedingung:

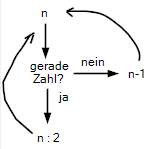
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | Variablen | | | | Klauseln | | |
|  |  |  |  |  |  | **C1** | **C2** | **C3** |
|  |  | **1** | 0 | 0 | 0 | 1 | 0 | 0 |
|  | **1** | 0 | 0 | 0 | 0 | 0 | 1 |
|  |  |  | **1** | 0 | 0 | 1 | 0 | 1 |
|  |  | **1** | 0 | 0 | 0 | 1 | 0 |
|  |  |  |  | **1** | 0 | 1 | 1 | 0 |
|  |  |  | **1** | 0 | 0 | 0 | 0 |
|  |  |  |  |  | **1** | 0 | 0 | 1 |
|  |  |  |  | **1** | 0 | 1 | 0 |
|  |  |  |  |  |  | **2** | **2** | **1** |
|  |  |  |  |  |  | 1 |  |  |
|  |  |  |  |  |  | 1 |  |  |
|  |  |  |  |  |  |  | 1 |  |
|  |  |  |  |  |  |  | 1 |  |
|  |  |  |  |  |  |  |  | 1 |
|  |  |  |  |  |  |  |  | 1 |
|  | **T** | 1 | 1 | 1 | 1 | 3 | 3 | 3 |

**3** wähle nichts, **2** wähle eines von beiden, **1** beides hinzufügen,**Straight-Line Programs (aSLP)**

Sei F ein Körper oder Ring oder eine Gruppe

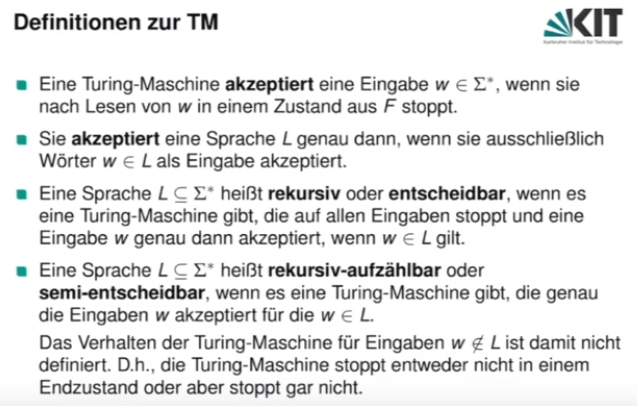
Def Algebrahische straight-line programs (aSLP)  
Ein aSLP der Länge r mit Input I={x1,…xn}  
- Programm ohne Loop und Verzweigung  
- Ist ein Monoid sieht aus wie eine Gruppe ist aber keine.

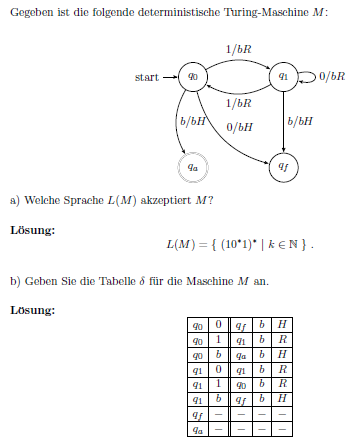
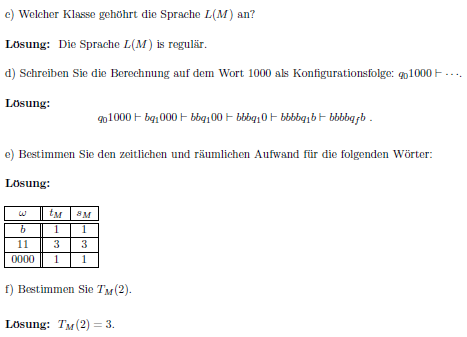
Additionskette  
Eine Folge von positiven natürlichen Zahlen heisst Additionskette (AK).   
Schritte Zählen = Berechnungsaufwand r = Anzahl Addition  
Minimaler Berechnungsaufwand ist l(n)  
binäre Algorithmus (bA)



**Funktionen**  
Sind die Funktionen gleich:  
x = 3 , y= 4 und z =5  
 Operator geht nicht unter 0  
 3 + 4 - 5 =

Gegeben sei die Funktion

Berechnen Sie die Funktionswerte  
**Turing Maschine**

**  
**

**Universelle Turing Maschine**

Gödelnummer <M>:  
Die Gödelnummer beginnt und endet mit **111**. Die einzelnen Codezeilen werden mit **11** unterteilt und der Code mit **1** in seine Bestandteile zerlegt.

Zeile 1:

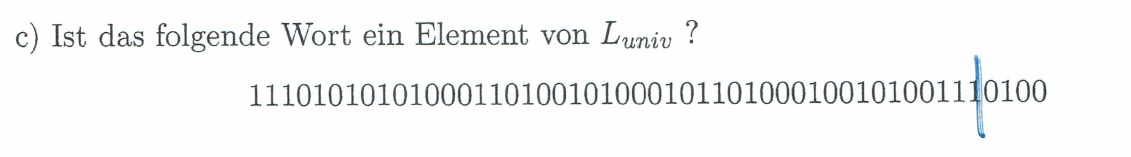
Zeile 2:

Zeile 3:

**111|**01010101000**|11|**010010101000**|11|**0100010010001000**|111**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | i+1 | j | k+1 | l | m |
|  |  |  |  |  |  |
|  | q0 | 0 | q0 | 0 | H |
|  | q0 | 1 | q0 | 0 | R |
|  | q0 | b | q1 | b | R |
|  | Zustand | Lesen | Neuer Zustand | Schreiben | Aktion |

Input rauslesen:  
Beispiel Input 0bbb  
  
   
Oftmals wird die Gödelnummer gezeigt und danach das Inputwort:



Mit der Aktion H oder mit der Tatsache, dass keine Operation in der Tabelle ist, ist man fertig. «F» ist der akzept. Zustand. Ist der Endzustand gleich F, dann ist es Element von

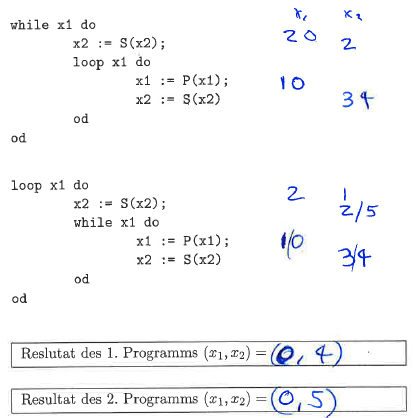
Gibt es ein Wort , das kein Element von ist?   
- nach einem Zyklus suchen. Am Besten mit «b» arbeiten.

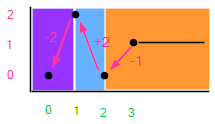
**Loop**Semantik:  
goto := goto m2 Springe zur Marke m2  
beendet dann, Sprung auf etwas was nicht existiert oder Sprung mehrmals vorkommt.

Aufgabe 3  
Stellen Sie die Funktion in einem loop dar:

x =0  
while f(x,y) 0 do  
 x := S(x);  
od

x =0  
while f(x,y,z) 0 do  
 x := S(x)  
od

****

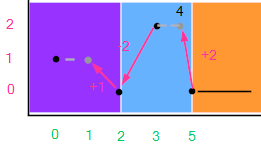
**Primitiv Rekursiv**Allgemein zu wissen:Beispiele

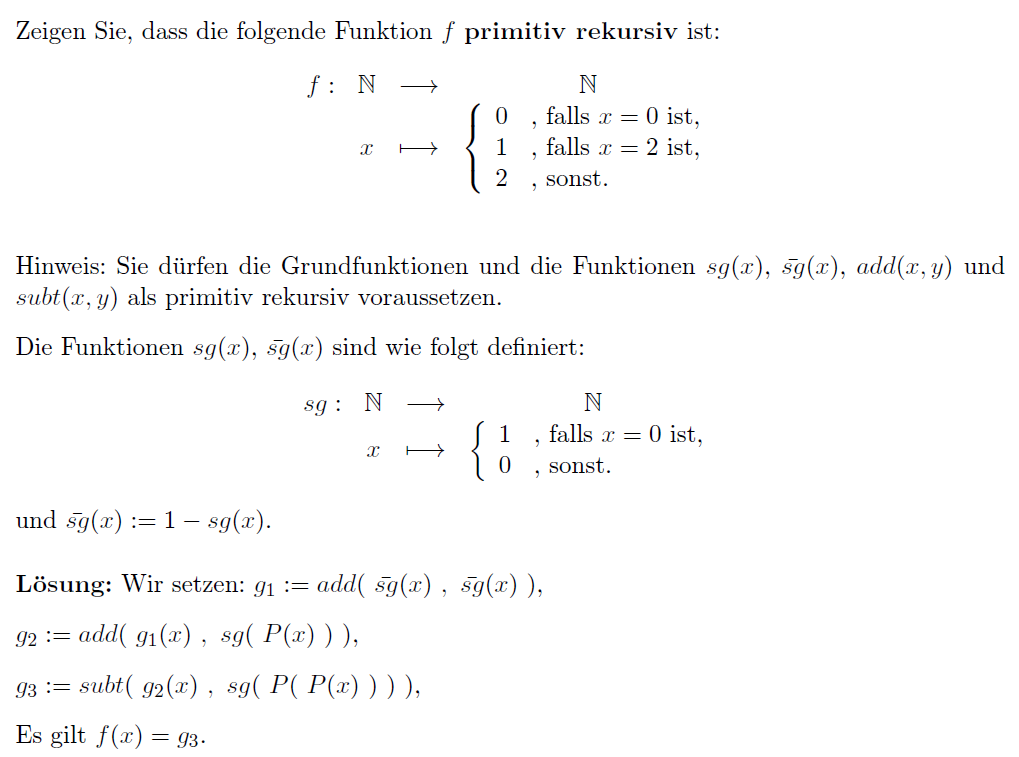
P(x)

subt(x)

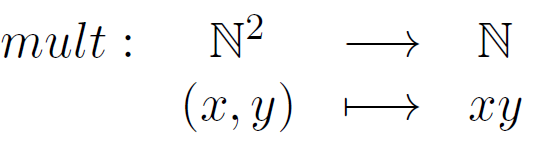
add(x)

sg(x)

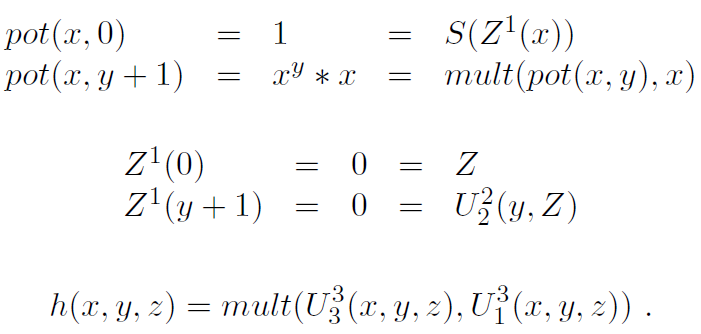


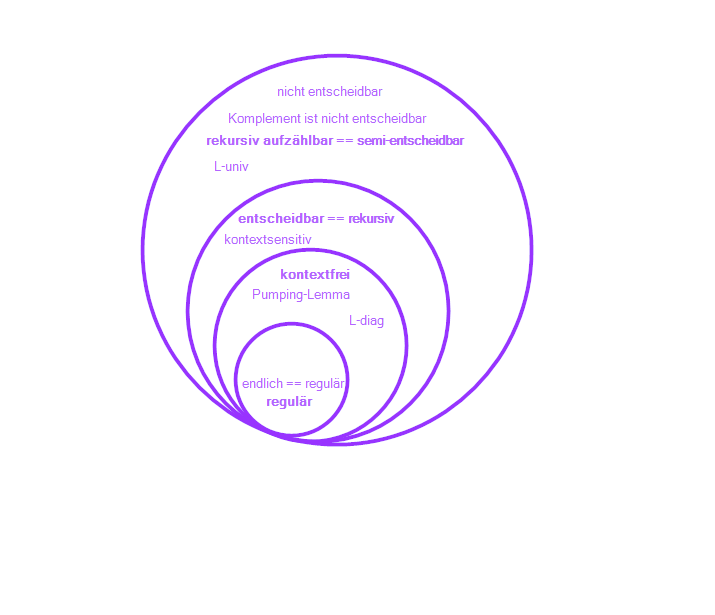


Subtraktionsbeispiel  
eine primitiv-rekursive Funktion ist, wobei gelten soll:  
x-y =

Multiplikationsbeispiel  
  
Faktorisierungsbeispiel  
fac(0) = 1   
fac(y) = (y) fac(y-1) = mult(y,fac(S(y)) )

Additionsbeispiel

Potenzbeispiel  


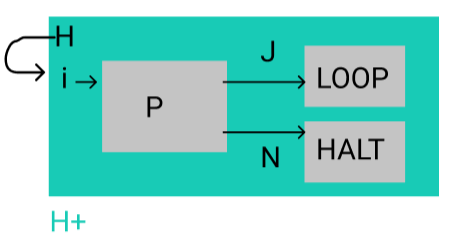
**Church Turing These**

These 1.21 (Church) Die Menge der intuitiv berechenbaren Funktionen ist die Menge der

-rekursiven Funktionen R.

Die These besagt, dass alles, was intuitiv berechenbar ist mit einer 1-Band Turing Maschine berechnet werden kann.  
Alles deutet darauf hin, dass die These richtig ist, da sich verschiedene Modelle alle als äquivalent herausgestellt haben.  
Beispiel: Turin Maschine -rekursiven Funktion, WHILE Programme. Lamda Kalkül, mit all diesen Modellen lässt sich dasselbe berechnen.

Paradox:  
a) eine Turing Maschine kann alles rechnen was ein digitaler Computer rechnen kann  
b) Eine Turing Maschine kann alles berechnen was berechnet werden kann



**These von Cobham und Edmonds**

Wenn zwei universelle Berechnungsmodelle (mit logarithmischem Kostenmass) gegeben sind, dann existiert ein Polynom p(n), so dass t Schritte im ersten Modell durch p(t) Schritte im zweiten Modell simuliert werden können.

**Reduktion**

steht für eine **p**olynomiale Reduktion.

steht für eine Turing-berechenbare totale Funktion f.

Beispiel

Gilt oder , keines oder beides?  
: Ja, da ich entscheiden kann ob ein Wort in liegt mittels einer t-berechenbare Funktion da sie rekursiv ist (weil regulär, weil endlich).  
Dann:

: Ja da ich eine Turing Maschine bauen könnte, die mir entscheidet ob ein Wort in liegt oder nicht. Dies weil kontextfrei ist und somit rekursiv.   
Beweis durch Grammatik mit kontextfreier Regel:

Lemma:  
1. Wenn rekursiv, dann ist auch rekursiv  
2. Wenn nicht rekursiv, dann ist auch nicht rekursiv  
Dieses Lemma wird mit dem Satz von Rice bewiesen.  
  
**Satz von Rice**  
1. R die Menge aller partiell, F berechenbaren Funktionen   
2. 3. L(S) ist die Menge der Gödelnummern aller TM’s, die eine Funktion aus S berechnen  
🡪 dann ist die Sprache L(s) nicht rekursiv  
🡪 bedeutet: man kann kein Programm schreiben, das testet ob die Programme (<M>) aus L(S) eine bestimmte Eigenschaft haben  
🡪 bedeutet: Aussagen über die von einer TM berechneten Funktion sind nicht entscheidbar

Die Diagonalsprache   
 wird für die Gödelisierung verwendet. Diese Sprache ist nicht entscheidbar! Auch nicht semi-entscheidbar und nicht rekursiv

enthält alle für die an der Diagonalen an der Stelle (i,i) eine Null steht.

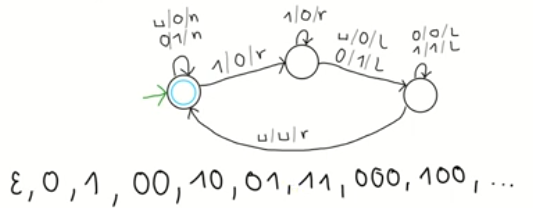
Die Universelle Sprache   
 ist die Menge aller Wörter, die die Turingmaschine bei der Eingabe der Gödelnummer akzeptiert. Auch diese Sprache ist nicht entscheidbar! Sie ist semi-entscheidbar

Das Post’sche Korrespondenzproblem.  
Das ist auch nicht entscheidbar. Gegeben sind Wortpaare ohne das Leere Wort. Mit Halteproblem beweisbar.

Regulär  
Jede reguläre Sprache ist eine kontextfreie Sprache

Jede kontextfreie Sprache ist eine entscheidbare Sprache  
 - folgt aus der Entscheidbarkeit des Wortproblems des kontextfreien Sprache.

Rekursiv Aufzählbar / Semi entscheidbar

- Eine Sprache ist rekursiv aufzählbar wenn das   
 Komplement auch aufzählbar ist.  
- Es existiert eine Turing Maschine M die die Eingaben   
 genau dann akzeptiert, wenn sie in der Sprache L liegen,   
 d.h. wenn L = L(M) gilt.  
- Wenn alle Zustände des Automaten semi-entscheidbar   
 sind, dann ist L Turing-erkennbar  
  
 Beispiel für eine Turing Maschine dieser Form ist diese :  
  
 = Diese Sprache ist rek aufzählbar.

Allgemeine Aussagen

- Ist bei einer kontextfreien Grammatik die Sprache regulär? -   
 Unentscheidbar eher nein

- Es gibt Sprachen, für die es keine erz. Grammatik gibt. 🗸

- Jede reguläre Sprache ist endlich 🗴  
- Jede endliche Sprache ist regulär 🗸

- Es gibt Sprachen für die es keine akzept. TM’s gibt 🗸

- ist nicht rekursiv, aber ist eine offene Frage ob   
 rekursiv aufzählbar ist 🗴

- ist nicht rekursiv, aber ist eine offene Frage ob   
 rekursiv aufzählbar ist 🗴

- es gibt Sprachen, die rekursiv aufzählbar aber nicht rekursiv   
 sind. 🗸

- Die Sprache ist nicht rekursiv. Komplement auch nicht 🗸

oder Komplement, eines von beiden darf nicht rek   
 aufzählbar sein 🗸

ist nicht rekursiv aber Rek aufzählbar. 🗸

und gehören in die gleiche Klasse 🗸

- Das Komplement einer endlichen Sprache ist rekursiv aufz, aber   
 nicht unbedingt rekursiv. 🗴  
- Es ist eine offene Frage ob es rekursiv aufz. Sprachen gibt die nicht rekursiv sind. 🗴

- - Schnitt kontextfrei und regulär ist nicht regulär

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Vereinigung | Schnitt | Komplement | Konkatenation |
|  |  |  |  |  |
| Kontextfrei | 🗸 | 🗴 | 🗴 | 🗸 |
| Regulär | 🗸 | 🗸 | 🗸 | 🗸 |
|  |  |  |  |  |
| Rek. Aufz. | 🗸 | 🗸 | 🗴 |  |

Gewichtung Theorie  
Berechnungstheorie: alle gleichwertig Komplexitätstheorie: 1-Band Maschine   
2-Band Maschine