

Grundlagen	
Alphabet	$\Sigma = \{0, 1\}$
Wort w	$w \in \Sigma^n$ $\varepsilon$ = leere Wort
Menge aller Wörter	$\Sigma^* = \{\varepsilon, 0, 1, 01, 10, \dots\}$ $\Sigma^0 = \{\varepsilon\}$
Sprache L	Teilmenge von $\Sigma^*$ , $L \subset \Sigma^*$ $L = \{0^n 1^n \mid n \geq 0\}$ $\emptyset$ = die leere Sprache
$\varepsilon$	Leeres Wort
$\emptyset$	Leere Sprache (Es gibt zwei Sprachen mit leerem Alphabet: $L_0 = \emptyset \wedge L_1 = \Sigma^0 \{\varepsilon\}$ )
Logische Formeln	<p>Beispiele:</p> <p>1. Übersetzen Sie die folgenden Aussagen soweit möglich in logische Formeln.</p> <p>a) Eine Aussage (nennen Sie diese P) ist entweder wahr oder falsch. <math>P \vee \neg P</math></p> <p>b) Jede reelle Zahl ist entweder positiv, negativ oder 0. <math>\forall x \in \mathbb{R} (x &gt; 0 \vee x &lt; 0 \vee x = 0)</math></p> <p>c) Negative (reelle) Zahlen haben keine reellen Wurzeln. <math>x \in \mathbb{R} \wedge x &lt; 0 \Rightarrow \neg \exists w \in \mathbb{R} (x = w^2)</math></p> <p>2. Übersetzen Sie die folgenden logischen Formeln in deutsche Sätze:</p> <p>a) <math>\exists m \in \mathbb{N} (n=2m)</math> n ist eine gerade Zahl</p> <p>b) <math>\exists m \in \mathbb{N} (n=2m) \Rightarrow (-1)^n &gt; 0</math> Eine gerade Potenz von -1 ist positiv</p> <p>c) <math>\neg x \in \mathbb{R} &gt; 0 \vee \exists w \in \mathbb{R} (x=w^2)</math> wenn x eine positive reelle Zahl ist, dann hat x eine Quadratwurzel</p> <p>3. Was ist falsch an folgendem Beweis für <math>2 = 1</math>?</p> <div style="display: flex; justify-content: space-between;"> <div> <math>a = b</math>  <math>a^2 = ab</math>  <math>a^2 - b^2 = ab - b^2</math>  <math>(a+b)(a-b) = b(a-b)</math>  <math>a+b = b</math>  <math>1+1 = 1</math> </div> <div> <math>  \cdot a</math>  <math>  -b^2</math>  <math>  : (a-b)</math>  <math>a = 1, b = 1</math> </div> </div> <p>Lösung: Die Teilung durch (a-b) ist nicht erlaubt, wenn <math>a = 1</math> und <math>b = 1</math>, da in dies in diesem Fall einer Division durch 0 entspricht!</p>
Konjunktion	$P \wedge Q$
Disjunktion	$P \vee Q$
Negation	$\neg P$
Implikation	$P \Rightarrow Q \quad = \neg P \vee Q$
Übersicht	<p>Es gibt überabzählbar viele Sprachen Es gibt aber nur abzählbar viele Turing Maschinen. Folglich gibt es Sprachen die nicht Turing erkennbar sind.</p>

<b>Lösungs- übersicht</b>	<p>Sprache regulär <math>\rightarrow</math> Pumping Lemma oder Myhill-Nerode</p> <p>Sprache kontextfrei / gibt es eine Grammatik? <math>\rightarrow</math> Pumping Lemma</p> <p>Automaten vergleichen <math>\rightarrow</math> Minimal Automat / Kreuzchen Algorithmus</p> <p>Problem in polynomieller Zeit von NTM <u>entscheidbar</u> <math>\rightarrow</math> Zertifikat, Verifikation</p> <p>Ob ein Wort aus einer Variablen der Grammatik abgeleitet werden kann <math>\rightarrow</math> CYK: Cocke-Younger-Kasami Algorithmus</p> <p>Ist eine Sprache nicht entscheidbar <math>\rightarrow</math> Satz von Rice</p>
Primzahl	Alle Primzahlen sind ungerade
Eigenschaften	Die erste Primzahl ist 2 $\rightarrow P = \{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43\}$
Palindrom	Die Länge des Palindrom kann gerade und ungerade sein
Eigenschaften	Es muss vorwärts und rückwärts gelesen auf das selbe Ergebnis führen Bsp. ANNA, BOB, ROTOR

Reguläre Sprachen	
Definition	Eine Sprache L(A) heisst regulär, wenn es einen DEA A gibt, der sie akzeptiert.
DEA: Deterministische Endlicher Automaten / DFA: Deterministic Finit Automat	
Ein endlicher Automat hat keinen Speicher, nur Zustände	
Definition	<p>Quintupel <math>(Q, \Sigma, \delta, q_0, F)</math></p> <p><b>Q:</b> Menge an Zuständen, <b><math>\Sigma</math>:</b> Alphabet, <b><math>\delta</math>:</b> Übergangsfunktion, <b><math>q_0</math>:</b> Startzustand, <b>F:</b> Menge aller Akzeptierzustände</p>
Ist eine Zahl durch 3 teilbar	<p><math>\Sigma = \{0, 1\} \quad L = \{w \in \Sigma^* \mid w \text{ ist durch 3 teilbar}\}</math></p>
Wort enthält maximal ein b (links)	
Gerade Anzahl von a und mindestens zwei b (rechts)	
An jeder ungeraden Stelle eine 1 (links)	
w kürzer wie 2 Zeichen (rechts)	

Pumping LEMMA für DEA's	
Mit dem Pumping Lemma kann erkannt werden ob eine Sprache nicht regulär ist. Eine Sprache ist nicht regulär, wenn sie mindestens <u>ein Wort</u> enthält, welches nicht aufgepumpt werden kann.	
Definition	<p>Ist L eine reguläre Sprache, dann gibt es eine Zahl N, die Pumping Length, so dass jedes Wort <math>w \in L</math> mit <math> w  \geq N</math> in drei Teile <math>w = xyz</math> zerlegt werden kann, so dass:</p> <p><math> y  &gt; 0 \quad  xy  \leq N \quad xy^kz \in L \quad \forall k \geq 0</math></p> <p>ACHTUNG: Die genaue Position von <math> xy </math> ist nicht bekannt!</p>
Beispiel: Palindrom	<p><math>L = \{w \in \Sigma^* \mid w \text{ ist ein Palindrom}\}</math> ist nicht regulär.</p> <ol style="list-style-type: none"> <li>Annahme: L ist regulär</li> <li>PL: <math>\exists N</math></li> <li>konstruiere w: <math>w = 0^N 1^N 0^N</math></li> <li>Zerlegung:</li> </ol> <ol style="list-style-type: none"> <li>Pumpen: <math>k &gt; 1 \Rightarrow xy^kz</math> hat mehr Nullen links als rechts <math>\Rightarrow \notin L</math></li> <li>Widerspruch: L ist nicht regulär</li> </ol>
Beispiel: $0^n 1^n$	<p><math>L = \{0^n 1^n \mid n \geq 0\}</math></p> <ol style="list-style-type: none"> <li>Annahme: L ist regulär</li> <li>Pumping Lemma: <math>\exists n</math></li> <li>Konstruiere <math>w = 0^n 1^n</math>, <math> w  = 2n \geq n</math>, <math>w \in L</math></li> <li>Zerlegung:</li> </ol> <ol style="list-style-type: none"> <li>Pumpen <math>k &gt; 1 \Rightarrow xy^kz</math> hat mehr 0 als 1 <math>\Rightarrow xy^kz \notin L</math></li> <li>Widerspruch: L ist nicht regulär!</li> </ol>
Beispiel: Sind zwei binäre Zahlen gleich	<p><math>L = \{w = w \mid w \in \{0,1\}^*\}</math></p> <ol style="list-style-type: none"> <li>Annahme: L ist regulär</li> <li>Pumping Lemma: <math>\exists n</math></li> <li>Konstruiere w: <math>10^n = 10^n</math></li> <li>Zerlegung:</li> </ol> <ol style="list-style-type: none"> <li>Pumpen: Auf der linken Seite des Gleichheitszeichens wird eine grössere Binärzahl wie auf der rechten Seite stehen.</li> <li>Widerspruch: L ist nicht regulär!</li> </ol>
Mathematische r Beweis anstatt Zerlegung	<p>Sei <math>\Sigma = \{a, b, c, \dots, z\}</math> und <math>L = \{w \in \Sigma^* \mid \text{für alle } a, b \in \Sigma \text{ gilt }  w _a &lt;  w _b, \text{ wenn } b \text{ im Alphabet auf } a \text{ folgt}\}</math></p> <p><math>w = a^{N+1} b^{N+2}</math>  <math>w_k = x y^k z = a^{N+1+(k-1) y } b^{N+2}</math></p> <p>Es gilt für <math>k &gt; 1</math>  <math>N + 1 + (k - 1) y  \geq N + 1 + (k - 1) = N + k \geq N + 2</math>  Das neue Wort <math>w_k</math> erfüllt nicht mehr die Spezifikation der Sprache</p>

### Myhill-Nerode

Mit dem Satz von Myhill-Nerode kann erkannt werden, ob eine Sprache regulär ist. Diese relative aufwändige Methode erlaubt es einen DEA zu erstellen.

### Rekonstruktion eines DEA

Beispiel:

$\Sigma = \{a, b\}$      $L = \{w \in \Sigma^* \mid w = b^*ab^*ab^*\}$

// erstelle eine Tabelle  
links kommen Wörter des Alphabets  
rechts kommt „was muss ich anhängen, damit das Wort Teil der Sprache L wird?“

w	L(W)
$\epsilon, b, bb, \dots = b^*$	L
$a, ba, ab, \dots = b^*ab^*$	$\{a, ba, ab, \dots\} = \{b^nab^n \mid n \in \mathbb{N}\}$
$aa, aab, \dots = b^*ab^*ab^*$	$\{\epsilon, b, bb, \dots\} = \{b^n \mid n \in \mathbb{N}\}$
$aaa, abaa, \dots$	$\emptyset$

//Jede Menge der Tabelle entspricht einem Zustand im DEA.  
(es entsteht ein minimal-Automat)  
endliche Menge an Zuständen → es handelt sich also um eine reguläre Sprache!

### Minimal Automat / Kreuzchen Algorithmus

Mit dem minimal Automaten kann bewiesen werden ob zwei endliche Automaten A und B, die gleiche Sprache akzeptieren  $L(A) = L(B)$ . Zu jedem DEA gibt es einen Minimalautomaten.

### Algorithmus

1. Tabelle aller Zustände erstellen
2. alle Zustände zu sich selber äquivalent markieren ( $\equiv$ )
3. alle Zustände zu allen Akzeptierzuständen als nicht äquivalent markieren(x)
4. Der Reihe nach alle leeren Felder kontrollieren:  
Von einem Paar  $(q_i, q_j)$  mit einem bestimmten Zeichen-Übergang zu einem Paar kommt, welches bereits mit x markiert ist, sind auch  $(q_i, q_j)$  x.
5. Schritt 4 wiederholen, bis keine neuen x mehr gesetzt werden können
6. alle restlichen Paare sind  $\equiv$ .

### Beispiel

Ausgangs DEA:

Tabelle:

X	q0	q1	q2	q3
q0	$\equiv$			
q1	x	$\equiv$		
q2	x		$\equiv$	
q3	x	x	x	$\equiv$

Minimal-Automat:

### REGEX

.	Jedes Zeichen
\d	Beliebige Zahl
\s	Whitespace
\w	Alphanumerisches Zeichen [a-zA-Z_0-9]
^	Start einer Zeile / Beginn des Textes
\$	Zeilenende / Ende des Textes
()	Gruppe
?	Kommt 0 oder 1 Mal vor
+	Kommt 1 oder mehrere Male vor
*	Kommt 0 oder mehrere Male vor
{n}	Kommt n Mal vor
{n,m}	Kommt n bis m Mal vor
X Y	X oder Y

Beispiele

IP-Adressen:  $[0-9]+\.[0-9]+\.[0-9]+\.[0-9]+$   
MAC-Adresse:  $([0-9A-Fa-f]{2}[:-]){5}([0-9A-Fa-f]{2})$   
Ganzzahlen ohne führende Nullen:  $0|((\s|-)[1-9][0-9]*)$

### Beispiel Winkelangaben

Format	Beispiel
dezimale Grade	65.4321
Grad, dezimale Minuten	65 25.926
Grad, Minuten, dezimale Sekunden	65 25 55.56

Optionaler Nachkommateil:  $(\.[0-9]*)?$

Maximal drei Gruppen von Ziffern inkl Leerzeichen:  $[0-9]+(\ *[0-9]+){0,2}$

Optionales Vorzeichen:  $(|-|+)$

Ergibt:  $(|-|+)[0-9]+(\ *[0-9]+){0,2}(\.[0-9]*)?$

### REGEX → NEA

Vorgehen

1. Den Regex von innen nach aussen übersetzen (Klammern zuerst)
2. Die einzelnen Teile mittels  $\epsilon$ -Übergänge verbinden

### Bausteine

Ausdruck r	Bedeutung
a	steht für das Zeichen a $a \in \Sigma$
.	steht für ein beliebiges Zeichen aus $\Sigma$
[aeiou]	steht für ein Zeichen aus {a, e, i, o, u} $\subset \Sigma$
[1-9]	steht für die positiven Ziffern
$\epsilon$	steht für das leere Wort
$\emptyset$	steht für die leere Sprache

a		
ab		
a+		Gleich wie a* aber neuer Zustand kein Akzeptierzustand
a*		Zusätzlicher Akzeptierzustand und Rückwärts $\epsilon$ -Übergang
.		Beliebiges Wort
$\epsilon$		Leeres Wort
$\emptyset$		Leere Sprache
$\Sigma^*$		Komplettes Alphabet
a b		Alternative (Siehe auch reguläre Operationen)

### Beispiel

$(ab)^*c$

### Beispiel

$a^+|(ab)^+$

### DEA → REGEX

Vorgehen

1. Neuer Akzeptierzustand und Startzustand erstellen und mit  $\epsilon$  Übergängen verbinden
2. Von Aussen nach innen alle «Wege» zum Akzeptierzustand notieren

### Beispiel

Ausgangslage sei der DEA der testet, ob eine Zahl durch 3 teilbar ist

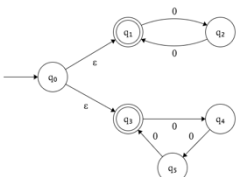
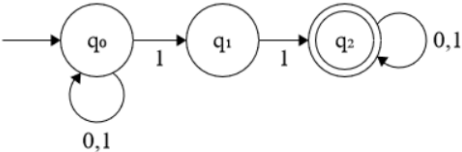
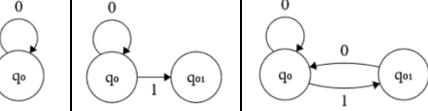
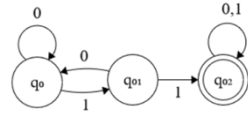
### Ausgangs DEA:

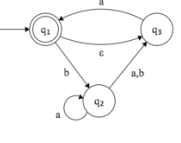
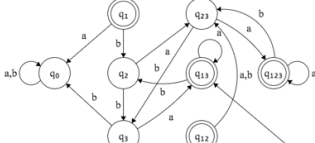
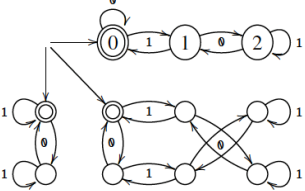
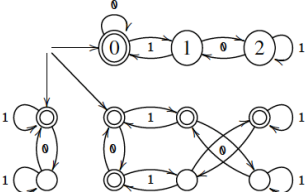
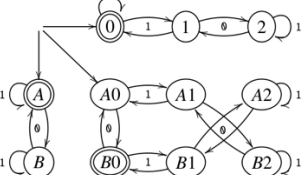
### Neuer Start und Akzept.Zustand

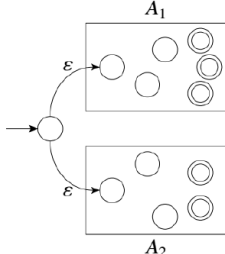
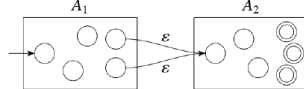
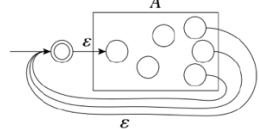
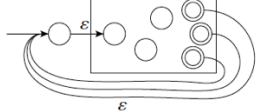
### zuletzt werden die „inneren“ Zustände entfernt

### Beispiel

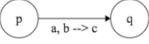
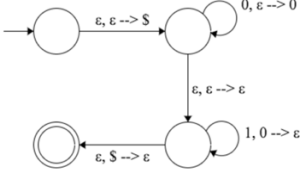
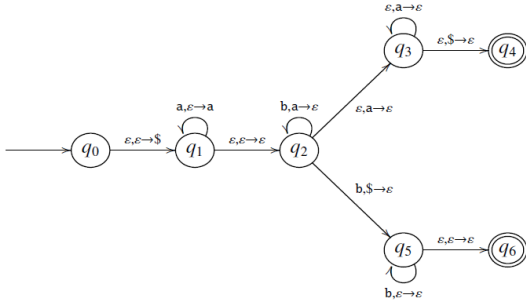
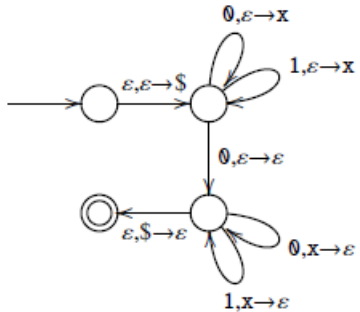
$(a|b)^*$

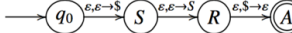
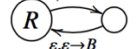


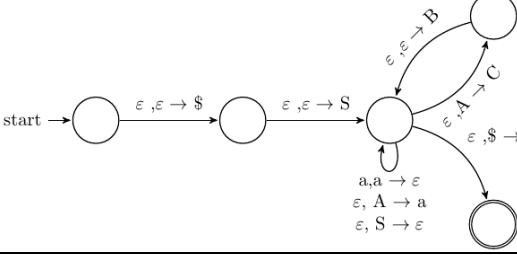
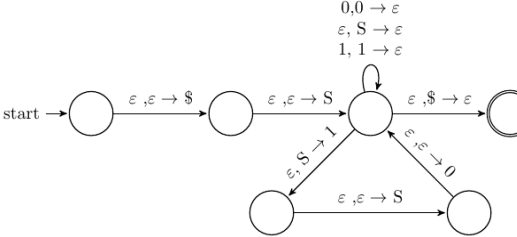
NEA: Nichtdeterministische endliche Automaten	
Man kann jeden NEA in in einen DEA umwandeln.	
Definition	Quintupel $(Q, \Sigma, \delta, q_0, F)$ <b>Q:</b> Menge an Zuständen, <b><math>\Sigma</math>:</b> Alphabet, <b><math>\delta</math>:</b> Übergangsfunktion, <b><math>q_0</math>:</b> Startzustand, <b>F:</b> Menge aller Akzeptierzustände
erlauben zusätzlich	1. Pfeile weglassen 2. mehrere Pfeile mit dem selben Zeichen pro Zustand 3. Pfeile ohne Input = $\epsilon$ -Übergänge = Sprünge
Akzeptiert der NEA?	1. Für das aktuelle Zeichen sind mehrere Übergänge möglich 2. Der Automat teilt sich in mehrere Kopien <ol style="list-style-type: none"> <li>Falls für ein Zeichen kein Übergang vorhanden ist, stirbt die Kopie</li> <li><math>\epsilon</math>-Übergänge sind gratis</li> </ol> 3. Jede Kopie folgt einer Möglichkeit 4. Der NEA akzeptiert das Wort falls er in einer oder mehreren Möglichkeiten einen Akzeptierzustand erreicht.
Beispiel	 $L = \{w \in \Sigma^* \mid  w  \text{ ist durch 2 oder 3 teilbar}\}$ $\Sigma = \{0\}$
NEA $\rightarrow$ DEA Transformation (rote Punkte)	
Ziel:	Es gibt einen Algorithmus, mit dem entschieden werden kann, ob zwei Automaten die selbe Sprache akzeptieren. $\Rightarrow L(A) = L(B)$ ? <ol style="list-style-type: none"> <li>Wandle A und B in DEA's um</li> <li>Reduziere A und B zu Minimalautomaten</li> <li><math>L(A) = L(B) \Leftrightarrow A</math> und B identisch sind</li> </ol>
Vorgehen	1. Startzustand aufzeichnen 2. Für alle Elemente des Alphabets einen Übergang einzeichnen 3. Allenfalls neuer kombinierter Zustand erstellen, falls man nach dem Übergang auf unterschiedlichen Zuständen landet 4. Für alle neuen Zustände wieder alle Übergänge einzeichnen 5. Zu guter Letzt muss überprüft werden, ob von jedem DEA Zustand nur ein Pfeil eintrifft und weggeht. Alle anderen können weggelassen werden.
Beispiel: Ausgangs NEA	
Schritt für Schritt	
Akzeptierzustände sind alle Zustände die auch im NEA ein Akzeptierzustand war.	Fertiger DEA 

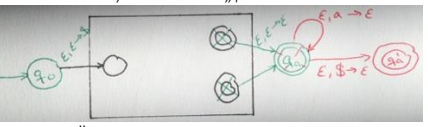
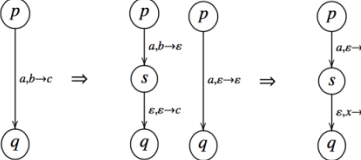
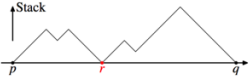
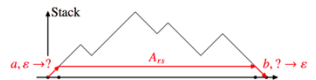
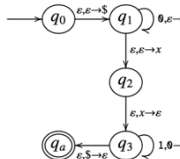
Bei kompliziertere NEAs muss wie folgt vorgegangen werden:	
1. Potenzmenge der NEA Zustände bilden ( $\{\text{leerer Zustand } q_0, q_1, q_2, q_3, (q_1, q_2), (q_1, q_3), (q_2, q_3), (q_1, q_2, q_3)\}$ ) 2. Übergänge zeichnen oben erwähntem «rote Punkte» Algorithmus 3. Akzeptierzustände des NEA sind auch im NEA Akzeptierzustände: Alle Zustände mit „1“ sind DEA-Akzeptierzustände $\rightarrow q_1, q_{13}, q_{12}, q_{123}$ etc. 4. Zur Wahl des neuen Startzustands auch $\epsilon$ -Übergänge beachten! Da man von $q_1$ «gratis» nach $q_3$ kommen kann, muss der Kombinierte Status $q_{13}$ als neuer Startzustand angenommen werden.	
Beispiel	NEA:  DEA: 
Produktautomat	
Vorgehen	1. Je ein Automat vertikal und horizontal schreiben 2. Bei jedem Schnittpunkt einen neuen Zustand zeichnen 3. Pro ursprüngliches Zustands-Paar die neuen Zustände verbinden und Übergänge anschreiben 4. Akzeptierzustände gemäss folgenden Mengen Operationen definieren
Vereinigung U	$L_1 \cup L_2$ 
Schnittmenge $\cap$	$L_1 \cap L_2$ 
Differenz $L(A) \setminus L(B) \Leftrightarrow L(A) \cap \overline{L(B)}$	


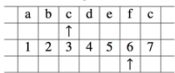
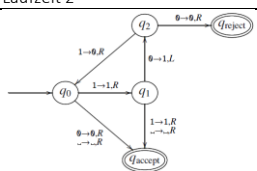
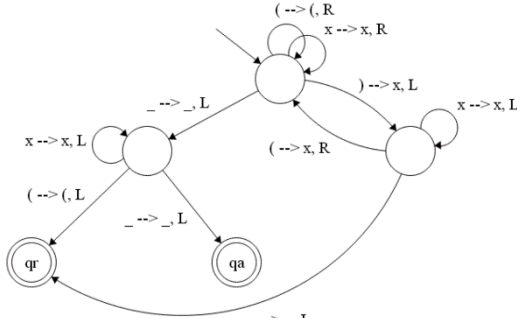
Reguläre Operationen	
Mit den regulären Operationen lassen sich aus regulären Sprachen neue reguläre Sprachen erzeugen.	
Vereinigung / Alternative $(L_1, L_2) \mapsto L_1$	Neuen Startzustand erstellen und mittels $\epsilon$ -Übergang mit den ursprünglichen Startzuständen verbinden 
Verkettung $(L_1, L_2) \mapsto L_1 L_2$	Akzeptierzustände A_1 mit Startzustand A_2 mit $\epsilon$ -Übergang verbinden 
Stern Operation $L \mapsto L^*$	Akzeptierzustände von A_1 mit einem neu erstellten Akzeptierzustand verbinden. Dieser neue Akzeptierzustand mit e-Übergang mit dem Startzustand des Automaten verbinden. 
$L^+$	

Pumping Lemma für kontextfreie Sprachen	
Mit dem Pumping Lemma für CFG's kann erkannt werden, ob eine Sprache nicht kontextfrei ist	
Definition	<p>Ist L eine kontextfreie Sprache, dann gibt es eine Zahl N, die Pumping Length, so dass jedes Wort <math>w \in L</math> mit <math> w  \geq N</math> in fünf Teile <math>w = uvxyz</math> zerlegt werden kann, so dass:</p> <p><math> vy  &gt; 0 \quad  vxy  \leq N \quad uv^kxy^kz \in L \quad \forall k \geq 0</math></p> <p>ACHTUNG: Die genaue Position von <math>vxy</math> ist nicht bekannt!</p>
Beispiel	<p><math>L = \{a^n b^n c^n \mid n \in \mathbb{N}\}</math></p> <ol style="list-style-type: none"> <li>Annahme: L ist kontextfrei</li> <li>Pumping Lemma: <math>\exists N</math></li> <li><math>w = a^N b^N c^N \quad  w  = 3N</math></li> <li>Unterteilung: <div style="display: flex; align-items: center;"> <div style="margin-right: 10px;"> <math>w =</math>  <math>w =</math> </div> <div style="display: flex; align-items: center;"> <div style="margin-left: 10px;"> <math>\in L</math>  <math>\in L</math> </div> </div> </div> <div style="margin-top: 10px;"> <math>uv^2xy^2z \notin L</math>  <math>uv^3xy^3z \notin L</math> </div> </li> <li>Pumpen: egal wie vxy liegt, können höchstens zwei der drei Buchstaben gepumpt werden.</li> <li>Widerspruch: <math>\notin L</math></li> </ol>
Grammatik für Programmierblöcke in C und Pascal	<div style="display: flex; justify-content: space-between;"> <div style="width: 45%;"> <p>for-Schleife in C:</p> <pre>for (i = 1; i &lt; 47; i++) {     powers[i] = 2 * powers[i - 1];     sum = sum + c[i] * powers[i]; }</pre> </div> <div style="width: 45%;"> <p>for-Schleife in Pascal:</p> <pre>for i := 1 to 46 do begin     powers[i] := 2 * powers[i - 1];     sum = sum + c[i] * powers[i] end</pre> </div> </div> <p>block <math>\rightarrow</math> '{ anweisungsfolge '}</p> <p><math>\rightarrow</math> B anweisungsfolge E</p> <p><math>\rightarrow</math> B E</p> <p>B <math>\rightarrow</math> 'b' 'e' 'g' 'i' 'n'</p> <p>E <math>\rightarrow</math> 'e' 'n' 'd'</p> <p>anweisungsfolge <math>\rightarrow</math> anweisungsfolge folgeelement</p> <p><math>\rightarrow</math> anweisungsfolge ';' anweisung</p> <p><math>\rightarrow</math> anweisung</p> <p><math>\rightarrow</math> <math>\epsilon</math></p> <p>folgeelement <math>\rightarrow</math> anweisung ';' </p>

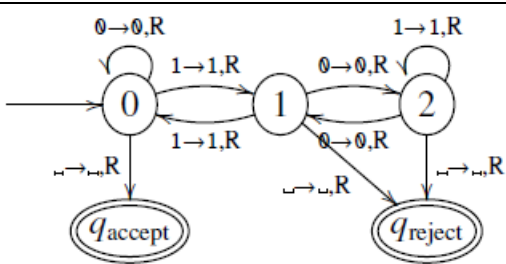
Stackautomaten / PDA: Push Down Automat	
Mit einem Stackautomaten kann herausgefunden werden, ob eine Sprache kontextfrei ist. Ein Stackautomat verfügt im Gegensatz zu einem DEA einen Speicher. Ein Stack Automat ist immer nicht deterministisch.	
Definition	6-Tupel $(Q, \Sigma, \Gamma, \delta, q_0, F)$ $Q$ : Menge an Zuständen, $\Sigma$ : Eingabe-Alphabet, $\Gamma$ : Stack Alphabet, $\delta$ : Übergangsfunktion, $q_0$ : Startzustand, $F$ : Menge an Akzeptierzuständen
Gerichteter, beschrifteter Graph	a: Input b: vorher auf Stack c: nachher auf Stack  $\$$ = Zuunterst auf dem Stack
Operationen mit dem leeren Wort $\epsilon$	1. $a, \epsilon \rightarrow c$ = c wird auf den Stack gelegt 2. $a, b \rightarrow \epsilon$ = b wird vom Stack entfernt 3. $a, \epsilon \rightarrow \epsilon$ = Stack bleibt unverändert
	Der Stackautomat akzeptiert, wenn der Stack beim Akzeptierzustand leer ist
Beispiel	$L = \{0^n 1^n \mid n \in \mathbb{N}\}, \Gamma = \{\$, 0, 1\}$ 
	$L = \{a^i b^j \mid i \neq j\}$ 
	$L = \{w \in \Sigma^* \mid  w  \text{ ist ungerade und das mittlere Symbol in } w \text{ ist } 0\}$ Zusätzlich zum Stackalphabet zwei Zeichen $\$$ und $x$ . Für jedes gelesene Input Zeichen legen wir ein $x$ auf den Stack. Nach der zentralen 0 entfernen wir für jedes gelesene Zeichen wieder ein $x$ . Akzeptiert wird nur wenn der Stack leer ist.
	

Grammatik $\rightarrow$ Stackautomat Transformation / Blümchenalgorithmus	
Bedingungen	1. Ist L eine kontextfreie Sprache mit Grammatik G, gibt es einen PDA der L akzeptiert 2. Die Grammatik muss in der Chomsky Normalform sein
Vorgehen	1. Grundgerüst für Stack und Startvariable erstellen  2. Gemäss den Regeln der Grammatik folgende Schritte durchführen: a. Für jede Regel der Art: $A \rightarrow BC$  b. Für jede Regel der Art: $A \rightarrow a$  c. Für jedes Terminalsymbol:  d. Falls die Regel $S \rightarrow \epsilon$ besteht wird ein zusätzlicher loop „ $\epsilon, S \rightarrow \epsilon$ “ erstellt
Beispiel	$L = \{0^n 1^n \mid n \geq 0\}$ Grammatik: $S \rightarrow 0S1$ $\rightarrow \epsilon$ Konstruktion des Automaten: 
	

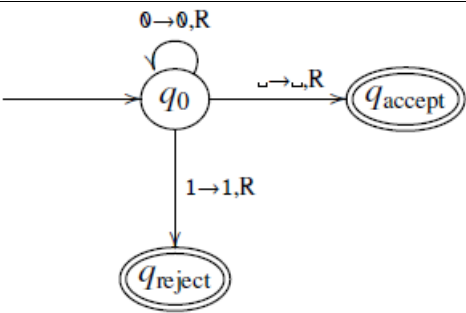
Stackautomat $\rightarrow$ Grammatik Transformation	
Bedingungen	1. Ordnung in Stackautomat bringen a. Nur ein Akzeptierzustand b. Stack leer beim Akzeptierzustand, dh zuerst alle $a \in \Gamma$ vom Stack entfernen, danach noch „ $\$$ “ entfernen  c. Bei jedem Übergang muss GENAU EIN Zeichen auf den Stack oder vom Stack 
	2. Grammatik erstellen a. Variablen der Grammatik sind Übergänge von Zustand p in Zustand q $\rightarrow A_{pq}$ Beginnend mit Start und Endzustand wird geschaut, was für Zeichen auf den Stack gelegt und vom Stack entfernt werden. Dabei werden zwei Fälle unterschieden. c. Fall 1: Die Zeichen unterscheiden sich. Dh. Das Zeichen muss früher bereits vom Stack entfernt worden sein.  Grammatik-Regel: $A_{pq} \rightarrow A_{pr} A_{rq}$ Fall 2: Die Zeichen sind gleich  Grammatik-Regel: $A_{pq} \rightarrow a A_{rs} b$ wobei a und b die jeweiligen Inputzeichen sind.
Beispiel	Ausgangslage: $L = \{0^i 1^n \mid n \geq 0\}$  Startvariable ist $A_{q_0 q_4}$ //Bei $q_0$ wird $\$$ auf den Stack gelegt und bei $q_4$ wird $\$$ vom Stack entfernt $\rightarrow$ Fall 2 Inputzeichen ist jeweils $\epsilon$ $A_{q_0 q_4} \rightarrow \epsilon A_{q_1 q_3} \epsilon$ $A_{q_1 q_3} \rightarrow \epsilon A_{q_2 q_2} \epsilon$ $0 A_{q_1 q_3} 1$ $A_{q_2 q_2} \rightarrow \epsilon$

Turing Maschinen																																																																									
Eine Turing Maschine kann alles was ein moderner Computer auch kann, ist dabei aber grundsätzlich langsamer. Die TM verfügt über ein <u>unendlich langes Band</u> , welche in einzelne Speicherzellen aufgeteilt ist. In einer Speicherzelle hat genau ein Zeichen platz. Der Schreib-, Lesekopf kann nur nach links oder rechts bewegt werden. Das Resultat einer Turing Maschine steht auf dem Band, sobald die Maschine den Akzeptierzustand erreicht. Wenn das ganze Band verarbeitet wurde, muss man entweder zum Zustand $q_{\text{accept}}$ oder $q_{\text{reject}}$ kommen.																																																																									
Definition	7-Tupel( $Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}}$ ) $Q$ : Menge an Zuständen, $\Sigma$ : Eingabe-Alphabet (ohne Blank Zeichen), $\Gamma$ : Band Alphabet, $\delta$ : Übergangsfunktion, $q_0$ : Startzustand, $q_{\text{accept}}$ : Akzeptierzustand, $q_{\text{reject}}$ : Ablehnungszustand																																																																								
Übergänge	Ersetze a mit b und verschiebe nach rechts oder links $(q_1) \xrightarrow{a \rightarrow b, R} (q_2)$																																																																								
Jede Sprache, die von einer mehrspurigen/mehrere Bänder Turingmaschine erkannt werden kann, kann auch von einer einspurigen/einem Band Turingmaschine erkannt werden.																																																																									
Mehrspurige Turing Maschine	Aus einer Turingmaschine mit n Spuren, kann immer eine einspurige Turingmaschine erstellt werden, welche mit einem n-Tupel als Bandsymbol arbeitet (i.e. Bandalphabet $\Gamma^n$ ) 																																																																								
Mehrband Turing Maschine	aus jeder Turingmaschine mit n Bändern, kann eine Turingmaschine mit einem Band und 2n Spuren erstellt werden. Wobei jede zweite Spur der Markierung des Lesekopfes dient.  Mehrband TM mit Laufzeit $O(t(n)) \rightarrow$ Standard TM mit Laufzeit $O(t(n)^2)$																																																																								
Nicht deterministische Turing Maschinen	Jede nichtdeterministische Turingmaschine ist äquivalent zu einer deterministischen Turingmaschine Nicht deterministische TM mit Laufzeit $O(t(n)) \rightarrow$ Standard TM mit Laufzeit $2^{O(t(n))}$																																																																								
Beispiel	<div><p>Inputwort = 10010 Rote Stelle = Kopf</p></div> <table><tr><td><math>q_0</math></td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>⋮</td><td>⋮</td></tr><tr><td><math>q_1</math></td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>⋮</td><td>⋮</td></tr><tr><td><math>q_2</math></td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>⋮</td><td>⋮</td></tr><tr><td><math>q_0</math></td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>⋮</td><td>⋮</td></tr><tr><td><math>q_1</math></td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>⋮</td><td>⋮</td></tr><tr><td><math>q_2</math></td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>⋮</td><td>⋮</td></tr><tr><td><math>q_0</math></td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>⋮</td><td>⋮</td></tr><tr><td><math>q_1</math></td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>⋮</td><td>⋮</td></tr><tr><td><math>q_{\text{accept}}</math></td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>⋮</td><td>⋮</td></tr></table>	$q_0$	1	0	0	1	0	⋮	⋮	$q_1$	1	0	0	1	0	⋮	⋮	$q_2$	1	1	0	1	0	⋮	⋮	$q_0$	0	1	0	1	0	⋮	⋮	$q_1$	0	1	0	1	0	⋮	⋮	$q_2$	0	1	1	1	0	⋮	⋮	$q_0$	0	0	1	1	0	⋮	⋮	$q_1$	0	0	1	1	0	⋮	⋮	$q_{\text{accept}}$	0	0	1	1	0	⋮	⋮
$q_0$	1	0	0	1	0	⋮	⋮																																																																		
$q_1$	1	0	0	1	0	⋮	⋮																																																																		
$q_2$	1	1	0	1	0	⋮	⋮																																																																		
$q_0$	0	1	0	1	0	⋮	⋮																																																																		
$q_1$	0	1	0	1	0	⋮	⋮																																																																		
$q_2$	0	1	1	1	0	⋮	⋮																																																																		
$q_0$	0	0	1	1	0	⋮	⋮																																																																		
$q_1$	0	0	1	1	0	⋮	⋮																																																																		
$q_{\text{accept}}$	0	0	1	1	0	⋮	⋮																																																																		
$\Sigma = \{ (, ) \}$ $L = \{ w \in \Sigma^* \mid w \text{ korrekte Klammerausdruck} \}$ z.B. $(( ( ) ) ) , ( ( ) ) ( )$ 																																																																									

Ist eine binärzahl durch drei Teilbar



Turing Maschine mit drei Zuständen die  $L = \{0^n \mid n \geq 0\}$



Berechenbarkeit	
Unendlich	Eine unendlich grosse Menge ist so gross, dass man darin noch Platz für eine Kopie der ganzen Menge finden kann
Abzählbar unendlich	<ul style="list-style-type: none"> <li>- Eine abzählbare Vereinigung von endlichen Mengen ist abzählbar</li> <li>- <math>\mathbb{N}</math>: natürlichen Zahlen</li> <li>- <math>\mathbb{Z}</math>: die ganzen Zahlen, da sie die Vereinigung der natürlichen Zahlen und deren negativen Abbildung sind. <math>\mathbb{Z} = \mathbb{N} \cup \{-n \mid n \in \mathbb{N}\}</math></li> <li>- <math>\mathbb{Q}</math>, die rationalen Zahlen, da sie durch Brüche (Paare aus natürlichen Zahlen) dargestellt werden können <math>\mathbb{Q} = \left\{ \frac{p}{q} \mid p \in \mathbb{Z}, q \in \mathbb{N} \setminus \{0\} \right\}</math></li> <li>- Die Menge aller Wörter <math>\Sigma^*</math></li> <li>- Die Menge aller DEA, NEA, reguläre Sprachen, CFG, Stackautomaten und Turing Maschinen</li> </ul>
Überabzählbar unendlich	<ul style="list-style-type: none"> <li>- <math>\mathbb{R}</math>: Die reellen Zahlen</li> <li>- Ist A eine abzählbar unendliche Menge, dann ist <math>P(A)</math> überabzählbar</li> <li>- Menge aller Sprachen über <math>\Sigma</math> ist eine Potenzmenge aller Wörter (abzählbar unendlich) <math>\rightarrow</math> ist also überabzählbar</li> </ul>
Entscheidbarkeit / Erkennbarkeit	
Turing entscheidbar	TM terminiert in endlicher Zeit. Ein Entscheider ist ein Programm das in endlicher Zeit terminiert.
Turing erkennbar	TM kann unendlich lang dauern. (man weiss nicht ob es nie ein Resultat geben wird)
Entscheider	Ein Entscheider ist eine Turing Maschine die garantiert anhält



Sprachprobleme	
Ein Hamiltonscher Pfad in einem gerichteten Graphen ist ein Pfad, der jeden Vertex des Graphen genau einmal besucht. Als Sprachproblem formuliert HAMPATH = $\{\langle G \rangle \mid G \text{ ist ein Graph mit einem hamiltonschen Pfad} \}$	
Formulieren Sie das Entscheidungsproblem “Gehört die Zahl n zu einem Primzahlwillings-paar?” als Sprachproblem. $L = \left\{ w \in \Sigma^* \mid w \text{ ist die Binärcodierung einer Zahl } n, \text{ die eine Primzahl ist,} \right. \\ \left. \text{so dass } n - 2 \text{ oder } n + 2 \text{ ebenfalls eine Primzahl ist} \right\}$	
<b>Entscheidungsalgorithmus:</b> <ol style="list-style-type: none"><li>1. Prüfe ob n ungerade, ansonsten <math>q_{\text{reject}}</math></li><li>2. Prüfe ob n eine Primzahl ist, ansonsten <math>q_{\text{reject}}</math></li><li>3. Prüfe ob n-2 oder n +2 Primzahlen sind. Falls beide keine Primzahlen sind <math>q_{\text{reject}}</math> andernfalls <math>q_{\text{accept}}</math></li></ol>	
Welche natürlichen Zahlen sind Quadrate einer natürlichen Zahl? $L = \{w \in \Sigma^* \mid w \text{ ist die Binärdarstellung einer Quadratzahl} \}$	
Falls $n \in \mathbb{N}$ eine Quadratzahl ist, finde man die Wurzel. $\Sigma = \{0,1,:\} \quad L = \left\{ w \in \Sigma^* \mid w \text{ ist von der Form } w_1:w_2, \text{ wobei } w_i \text{ Binärdarstellungen} \right. \\ \left. \text{von Zahlen } n_i \text{ sind mit } n_1 = n_2^2 \right\}$	
Man finde die Primfaktoren einer Zahl n. $\Sigma = \{0,1,:\} \quad L = \left\{ w \in \Sigma^* \mid w \text{ ist von der Form } n:p_1:n_2:p_2:\dots:p_k:n_k \text{ und es gilt} \right. \\ \left. n = p_1^{n_1} p_2^{n_2} \dots p_k^{n_k}, \text{ wenn man die } p_i \text{ und } n_i \text{ als Binärzahlen interpretiert} \right\}$	
Reduktionen	
Mittels Reduktion kann bewiesen werden, dass eine TM nicht entscheidbar ist. Man reduziert dabei eine TM auf eine nicht entscheidbares TM (z.B Empty <sub>TM</sub> ).	
Entscheidbarkeitsprobleme für reguläre Sprachen	
Akzeptanz-probleme	Kann mit einem Programm herausgefunden werden, ob ein DEA B ein Wort akzeptiert $A_{\text{DEA}} = \{ \langle B, w \rangle \mid B \text{ ist ein DEA und } B \text{ akzeptiert } w. \}$ <ol style="list-style-type: none"><li>1. Simuliere B auf einer TM mit Input w</li><li>2. Falls B akzeptiert <math>\rightarrow q_{\text{accept}}</math></li><li>3. Falls B nicht akzeptiert <math>\rightarrow q_{\text{reject}}</math></li></ol>
Leerheits-problem	Kann man mit einem Programm herausfinden ob ein DEA nichts akzeptiert? $L(B) = \emptyset$ $E_{\text{DEA}} = \{ \langle A \rangle \mid A \text{ ist ein DEA und } L(A) = \emptyset \}$ <ol style="list-style-type: none"><li>1. Markiere den Startzustand</li><li>2. Solange sich noch neue Zustände markieren lassen. Markiere alle Zustände, die sich von den bereits markierten aus erreichen lassen</li><li>3. Falls ein Akzeptierzustand markiert wurde: <math>q_{\text{reject}}</math> ansonsten <math>q_{\text{accept}}</math></li></ol>
Gleichheits-problem	Kann man mit einem Programm herausfinden ob zwei DEA's gleich sind? $EQ_{\text{DEA}} = \{ \langle A, B \rangle \mid A \text{ und } B \text{ sind DEAs und } L(A) = L(B) \}$ <ol style="list-style-type: none"><li>1. Erzeugen den minimalen Automaten A' zu A</li><li>2. Erzeuge den minimalen Automaten B' zu B</li><li>3. Falls A' = B': <math>q_{\text{accept}}</math> andernfalls <math>q_{\text{reject}}</math></li></ol>

Entscheidbarkeitsprobleme für kontextfreie Sprachen	
Akzeptanz-probleme	$A_{\text{CFG}} = \{ \langle G, w \rangle \mid \text{die kontextfreie Grammatik } G \text{ erzeugt } w \}$ <ol style="list-style-type: none"><li>1. Siehe CYK: Cocke-Younger-Kasami</li></ol>
entscheidbar	
Leerheits-problem	$E_{\text{CFG}} = \{ \langle G \rangle \mid G \text{ ist eine kontextfreie Grammatik und } L(G) = \emptyset \}$ <ol style="list-style-type: none"><li>1. Wandle G in Chomsky Normalform um</li><li>2. Markiere alle Terminalsymbole</li><li>3. Solange sich neue Variablen markieren lassen: markiere alle Variablen A, zu denen es eine Regel gibt, so dass alle Symbole auf der rechten Seite der Regel bereits markiert sind.</li><li>4. Falls S markiert wurde: <math>q_{\text{reject}}</math> andernfalls <math>q_{\text{accept}}</math></li></ol>
entscheidbar	
Gleichheits-problem	$EQ_{\text{CFG}} = \{ \langle G, H \rangle \mid G \text{ und } H \text{ sind kontextfreie Grammatiken und } L(G) = L(H) \}$
nicht entscheidbar	
Entscheidbarkeitsprobleme für Turing Maschinen	
Akzeptanzproblem	Hält die TM in $q_{\text{accept}}$ ? $A_{\text{TM}} = \{ \langle M, w \rangle \mid M \text{ ist eine TM und } M \text{ erkennt } w \}$
nicht entscheidbar	
Halteproblem	Hält die TM an? Entweder in $q_{\text{accept}}$ oder $q_{\text{reject}}$ ?
nicht entscheidbar	$HALT_{\text{TM}} = \{ \langle M, w \rangle \mid M \text{ ist eine TM und } M \text{ hält auf Input } w \}$
Leerheitsproblem	$E_{\text{TM}} = \{ \langle M \rangle \mid M \text{ ist eine TM und } L(M) = \emptyset \}$
nicht entscheidbar	
Gleichheitsproblem	$EQ_{\text{TM}} = \{ \langle M_1, M_2 \rangle \mid M_i \text{ sind Turingmaschinen und } L(M_1) = L(M_2) \}$
nicht entscheidbar	
Halting Theorem	
Es ist nicht möglich ein Programm zu schreiben, welches entscheiden kann ob ein anderes Programm jemals zu einem Ende kommt mit gegebenen Eingabe	
Beispiel	Man findet im Internet Websites, die anbieten, Word-Dokumente in PDF umzuwandeln. Ein Jungunternehmer möchte einen ähnlichen Dienst anbieten, um XML Files mit Hilfe von XSLT Stylesheets in PDF Files umzuwandeln. Die Kunden sollen einen Satz von XML-Files und XSLT Stylesheets zum Beispiel als ZIP-File hochladen können, und als Antwort ein fertig formatiertes PDF erhalten. Im Apache-Projekt FOP findet er geeignete freie Software dafür. Sie sind in diesem Projekt als Sicherheitsverantwortliche(r) dafür zuständig, dass alle XML-/XSLT-File-Kombinationen zurückgewiesen werden, die dazu führen könnten, dass der Server beliebig lange mit diesem Auftrag beschäftigt ist. Wie stehen Ihre Aussichten, dies zu realisieren?
	Die Sprache XSLT ist Turing-vollständig, man kann in ihr also alles programmieren, was mit einer Turing-Maschine gemacht werden kann. Die gestellte Aufgabe besteht also darin, herauszufinden, ob ein gegebenes Programm (XML-Files und XSLT Stylesheets) je anhalten wird. Dies ist das Halteproblem, welches nicht entscheidbar ist. Es gibt also keine Möglichkeit, einem Auftrag anzusehen, ob er dazu führen wird, dass der PDF-Renderer nicht terminieren wird.

Satz von Rice	
Der Satz von Rice besagt ob eine Sprache nicht entscheidbar ist. P sei eine Eigenschaft der Turing erkennbaren Sprache $L_1$ , wobei es eine Sprache $L_1$ gibt, welche die Eigenschaft P besitzt und eine Sprache $L_0$ , welche die Eigenschaft nicht hat.	
Vorgehen	<ol style="list-style-type: none"><li>1. Nichttriviale Eigenschaft P aufschreiben</li><li>2. Die beiden Sprachen <math>L_0</math> und <math>L_1</math> bilden</li><li>3. Gibt es ein Programm, welches beide Sprachen erkennen kann? Sind beide Sprachen Turing erkennbar?</li><li>4. Dann besagt der Satz von Rice, dass die Sprache nicht entscheidbar ist.</li></ol>
Beispiel	$P_{\text{PRIMES}} = \left. \begin{matrix} \text{„Sprache besteht genau aus den Primzahlen“} \\ L_0 = \{42\} \\ L_1 = \{\text{Primzahlen}\} = \mathbb{P} \end{matrix} \right\} \Rightarrow \text{ Turing erkennbar}$  $\text{Rice} \Rightarrow P_{\text{PRIMES}} \text{ nicht entscheidbar}$
	Können Sie einen Algorithmus angeben, der bei zu einer beliebigen turing-erkennbaren Sprache herausfinden kann, ob alle in der Sprache enthaltenen Wörter “wachsend” sind. Die Eigenschaft, dass eine Sprache nur wachsende Wörter enthält, ist nicht trivial: Die Sprache der wachsenden Wörter, hat sie, die Sprache $\Sigma^*$ nicht. Nach dem Satz von Rice ist diese Eigenschaft also nicht entscheidbar, einen Algorithmus der verlangten Art kann es also nicht geben.
	Können Sie einen Algorithmus angeben, der zu einer rekursiven Sprache herausfindet, ob sie ausschliesslich periodische Wörter Nein. Sei P die Eigenschaft der Sprache L “Sprache L enthält nur periodische Wörter”. Die leere Sprache $\emptyset$ hat diese Eigenschaft, die Sprache $\{01\}$ jedoch nicht, 01 ist nicht periodisch. Damit sind die Voraussetzungen des Satzes von Rice erfüllt, also ist die Eigenschaft P nicht entscheidbar.

Laufzeitkomplexität	
Klasse P	Zu der Klasse P gehören Sprachen die mit einem Entscheider in polynomieller Laufzeit ( $n^k$ ) entschieden werden können (Entscheider = TM die garantiert anhält) Bsp: Kontextfreie Sprachen
Klasse NP	NP ist die Klasse der Sprachen, für die ein deterministischer polynomieller Verifizierer existiert. Zu der Klasse NP gehören Sprachen die in polynomieller Zeit von einer nicht deterministischen Turing Maschine entschieden werden können.
NP-vollständig	die schwersten Probleme in NP → siehe Katalog von Karp
Verifizierer	Ein Verifizierer für die Sprache A ist eine Turingmaschine V für die gilt: $A = \{w \mid \exists V \text{ akzeptiert}(w,c)\}, c: \text{Lösungszertifikat}$
Polynomielle Reduktion	Allgemein: $A \leq_p B$ "A leichter als B" Konsequenz: $B$ entscheidbar $\Rightarrow A$ entscheidbar $A$ nicht entscheidbar $\Rightarrow B$ nicht entscheidbar $B \in P \Rightarrow A \in P$ $B \in NP \Rightarrow A \in NP$
	Stundenplan $\leq_p$ k-VERTEX-COLORING $S \rightarrow k\text{-VERTEX-COLORING}$ Fach $\mapsto$ Vertex Zeitfenster $\mapsto$ Farbe Anzahl Zeitfenster $\mapsto k$ Student/Anmeldung $\mapsto$ Kante
Vorgehen: Ist ein Problem NP? → Finde einen polynomiellen Verifizierer für das Problem	<ol style="list-style-type: none"><li>Ist ein Problem entscheidbar? Kann es mit reinem durchprobieren gelöst werden? → Es gibt einen polynomiellen Verifizieren für das Problem</li><li>Lösungszertifikat für den Verifizierer definieren. Was ist das Ziel des Rätsels? Wie kann es gelöst werden? Dies ist unser Lösungszertifikat.</li><li>Verifikationsalgorithmus: Wie kann das Zertifikat verifiziert werden. Was muss unternommen werden, damit das vorliegende Rätsel mit dem Lösungszertifikat verglichen werden kann. Man notiert sich in O-Notation den Rechenaufwand für die Prüfungen. Laufzeitkomplexität für die Verifikation bestimmen. (Grösste O-Notation) Ist der Algorithmus polynomiell, ist der Verifizieren NP</li><li></li></ol>
	SAT = { $\phi \mid \phi$ ist eine erfüllbare logische Formel } Behauptung: SAT $\in$ NP Beweis: <ol style="list-style-type: none"><li>entscheidbar? Ja: alle Wertekombinationen durchprobieren</li><li>Lösungszertifikat: Wahrheitswerte</li><li>Verifikation: Zertifikat einsetzen &amp; Wahrheitswerte berechnen</li><li>Laufzeit <math>\leq O(n)</math></li></ol>

Rätsel (NP Vollständig)	Das Spiel Hitori wird auf einem $n \times n$ -Feld gespielt, in jeder Zelle des Spielfeldes ist eine Zahl zwischen 1 und $n$ eingetragen. Der Spieler muss nun Zellen schwärzen, so dass zwei Regeln eingehalten werden:																				
	<ul style="list-style-type: none"><li>In einer Zeile darf keine (nicht geschwärzte) Zahl mehr als einmal vorkommen.</li><li>Benachbarte Zellen dürfen nicht geschwärzt werden.</li></ul>																				
	Beweis:																				
	1. entscheidbar Ja: alle Kombinationen probieren																				
	2. Lösungszertifikat: geschwärzte Felder																				
	3. Verifikation: Für jede geschwärzte Zelle kontrolliere, ob die maximal vier Nachbarzellen nicht geschwärzt sind. Für jede nicht geschwärzte Zelle kontrolliere, ob die anderen Zellen in der gleichen Zeile eine andere Zahl enthalten. Für jede nicht geschwärzte Zelle kontrolliere, ob die anderen Zellen in der gleichen Spalte eine andere Zahl enthalten.																				
	<table><tr><td>1.</td><td>Nachbarzellen nicht geschwärzt</td><td><math>O(n^2)</math></td></tr><tr><td>2.</td><td>Keine gleichen Werte in einer Zeile</td><td><math>O(n^3)</math></td></tr><tr><td>3.</td><td>Keine gleichen Werte in einer Spalte</td><td><math>O(n^3)</math></td></tr></table>	1.	Nachbarzellen nicht geschwärzt	$O(n^2)$	2.	Keine gleichen Werte in einer Zeile	$O(n^3)$	3.	Keine gleichen Werte in einer Spalte	$O(n^3)$											
	1.	Nachbarzellen nicht geschwärzt	$O(n^2)$																		
	2.	Keine gleichen Werte in einer Zeile	$O(n^3)$																		
	3.	Keine gleichen Werte in einer Spalte	$O(n^3)$																		
<table><tr><td>4.</td><td>Lösung für Hitori</td><td><math>O(n^3)</math></td></tr></table>	4.	Lösung für Hitori	$O(n^3)$																		
4.	Lösung für Hitori	$O(n^3)$																			
Das Spiel Light Up wird auf einem $n \times m$ Feld gespielt. Einzelne Felder sind schwarz gefäbt, manche davon sind zusätzlich mit einer Zahl versehen. Auf den weissen Quadraten müssen Glühbirnen plaziert werden, so dass die folgenden Regeln eingehalten werden:																					
<ul style="list-style-type: none"><li>Die Zahlen auf den schwarzen Quadraten geben an, wieviele Glühbirnen auf weissen Feldern sind, die über eine Kante an dieses schwarze Feld grenzen.</li><li>Jedes weisse Feld wird von mindestens einer Glühbirne beleuchtet. Eine Glühbirne leuchtet waagrecht und senkrecht bis zu einem schwarzen Feld oder zum Rand des Spielfeldes.</li><li>Glühbirnen dürfen sich nicht gegenseitig beleuchten.</li></ul>																					
Lösung: Schritt 1 und 2 siehe andere Bsp 3+4																					
<table><tr><th>Verifikation</th><th>Aufwand</th></tr><tr><td>1. Für jedes schwarze Feld mit einer Zahl überprüfe, ob die Anzahl der Glühbirnen auf den vier Nachbarfeldern der Zahl entspricht.</td><td><math>O(4nm)</math></td></tr><tr><td>2. Für jedes weisse Feld, welches eine Glühbirne enthält kontrolliere, ob in der gleichen Zeile oder Spalte eine weitere Glühbirne vorkommt.</td><td><math>O(mn(m+n))</math></td></tr><tr><td>3. Für jedes weisse Feld überprüfe, ob in der gleichen Zeile oder Spalte eine Glühbirnen vorhanden ist.</td><td><math>O(mn(m+n))</math></td></tr><tr><td>Gesamtaufwand</td><td><math>O(mn(m+n))</math></td></tr></table>	Verifikation	Aufwand	1. Für jedes schwarze Feld mit einer Zahl überprüfe, ob die Anzahl der Glühbirnen auf den vier Nachbarfeldern der Zahl entspricht.	$O(4nm)$	2. Für jedes weisse Feld, welches eine Glühbirne enthält kontrolliere, ob in der gleichen Zeile oder Spalte eine weitere Glühbirne vorkommt.	$O(mn(m+n))$	3. Für jedes weisse Feld überprüfe, ob in der gleichen Zeile oder Spalte eine Glühbirnen vorhanden ist.	$O(mn(m+n))$	Gesamtaufwand	$O(mn(m+n))$											
Verifikation	Aufwand																				
1. Für jedes schwarze Feld mit einer Zahl überprüfe, ob die Anzahl der Glühbirnen auf den vier Nachbarfeldern der Zahl entspricht.	$O(4nm)$																				
2. Für jedes weisse Feld, welches eine Glühbirne enthält kontrolliere, ob in der gleichen Zeile oder Spalte eine weitere Glühbirne vorkommt.	$O(mn(m+n))$																				
3. Für jedes weisse Feld überprüfe, ob in der gleichen Zeile oder Spalte eine Glühbirnen vorhanden ist.	$O(mn(m+n))$																				
Gesamtaufwand	$O(mn(m+n))$																				
Heyawake ist ein japanisches Rätsel, welches auf einem $n \times m$ -Feld gespielt wird. Das Spielfeld ist durch dickere Linien unterteilt in grössere rechteckige Gebiete, die Zimmer genannt werden. Ziel des Spieles ist, einzelne Felder schwarz einzufärben, so dass die folgenden Regeln erfüllt sind:																					
<ul style="list-style-type: none"><li>Schwarze Quadrate grenzen niemals über eine Kante aneinander.</li><li>Alle weissen Quadrate hängen über Kanten zusammen.</li><li>Die Zahlen geben an, wie viele schwarze Quadrate in einem Zimmer vorkommen.</li><li>Ein Zimmer ohne Zahl kann beliebig viele schwarze Quadrate enthalten (solange die anderen Regeln erfüllt sind).</li><li>Eine gerade (horizontale oder vertikale) Linie aus zusammenhängenden weissen Quadraten kann sich höchstens über zwei Zimmer erstrecken.</li></ul>																					
<table><tr><th>Regel</th><th>Arbeit</th><th>Aufwand</th></tr><tr><td>1.</td><td>Für jedes schwarze Feld 4 Nachbarn überprüfen</td><td><math>O(nm)</math></td></tr><tr><td>2.</td><td>Zusammenhang überprüfen</td><td><math>O(n^2m^2)</math></td></tr><tr><td>3.</td><td>Für jede Zahl die Anzahl schwarzer Felder im Zimmer prüfen</td><td><math>O(nm)</math></td></tr><tr><td>4.</td><td>automatisch erfüllt nach 3.</td><td>0</td></tr><tr><td>5.</td><td>Für jedes weisse Feld, überprüfe Zeilenlänge</td><td><math>O(nm(n+m))</math></td></tr><tr><td colspan="2">c ist Lösung des Heyawake-Rätsels</td><td><math>O(n^3m^2)</math></td></tr></table>	Regel	Arbeit	Aufwand	1.	Für jedes schwarze Feld 4 Nachbarn überprüfen	$O(nm)$	2.	Zusammenhang überprüfen	$O(n^2m^2)$	3.	Für jede Zahl die Anzahl schwarzer Felder im Zimmer prüfen	$O(nm)$	4.	automatisch erfüllt nach 3.	0	5.	Für jedes weisse Feld, überprüfe Zeilenlänge	$O(nm(n+m))$	c ist Lösung des Heyawake-Rätsels		$O(n^3m^2)$
Regel	Arbeit	Aufwand																			
1.	Für jedes schwarze Feld 4 Nachbarn überprüfen	$O(nm)$																			
2.	Zusammenhang überprüfen	$O(n^2m^2)$																			
3.	Für jede Zahl die Anzahl schwarzer Felder im Zimmer prüfen	$O(nm)$																			
4.	automatisch erfüllt nach 3.	0																			
5.	Für jedes weisse Feld, überprüfe Zeilenlänge	$O(nm(n+m))$																			
c ist Lösung des Heyawake-Rätsels		$O(n^3m^2)$																			

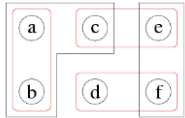
Turing vollständig		
Definition	Eine Sprache A ist Turing-vollständig, genau dann wenn es einen Turing Maschinen-Simulator in A gibt. (Eine Turing-vollständige Sprache muss Endlosschleifen machen können)	
Turing vollständige Sprachen	C, C++, Java, Java Script, XSLT, Latex, WHILE, GOTO, Brainfuck, Ook, etc.	
LOOP (nicht turing vollständig)		
LOOP terminiert immer und besitzt daher kein Halte-Problem → nicht Turing vollständig		
Syntax	Variablen $x_0, x_1, x_2, \dots$ Konstanten: 0, 1, 2, ... Zuweisung: $:=$ Trennung von Anweisungen: ; Operatoren: + und - Schlüsselwörter: LOOP, DO, END	
Beispiel:	Summe zweier Variablen	Produkt zweier Variablen
	$\begin{array}{l} x_0 := x_1 \\ \text{LOOP } x_2 \text{ DO} \\ \qquad x_0 := x_0 + 1 \\ \text{END} \end{array}$	$\begin{array}{l} x_0 := 0 \\ \text{LOOP } x_1 \text{ DO} \\ \qquad \text{LOOP } x_2 \text{ DO} \\ \qquad \qquad x_0 := x_0 + 1 \\ \qquad \text{END} \\ \text{END} \end{array}$
IF Erweiterung	$\text{IF } x = 0 \text{ THEN } \underline{P \text{ END}}$ $\begin{array}{l} y := 1; \\ \text{LOOP } x \text{ DO } y := 0 \text{ END}; \\ \text{LOOP } y \text{ DO } P \text{ END}; \end{array}$	
WHILE und GOTO (turing vollständig)		
WHILE	$\text{WHILE } x_i > 0 \text{ DO } P \text{ END}$	
GOTO	$M_j: \quad \text{IF } x_i = c \text{ THEN GOTO } M_j$	

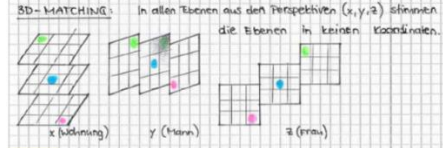

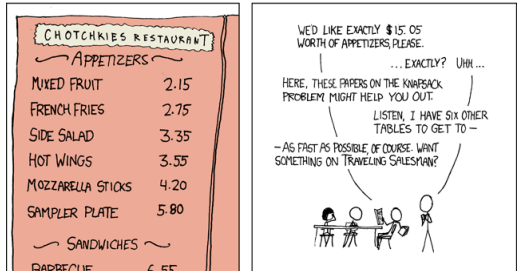


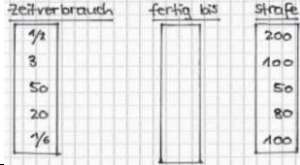
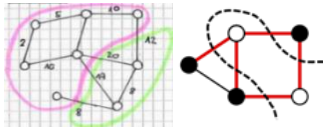
Reduktionen	
Mittels Reduktion kann bewiesen werden, dass ein Problem nicht entscheidbar ist. Man reduziert dabei ein Problem auf eine bekanntes, nicht entscheidbares Problem	
Vorgehen	<ol style="list-style-type: none"> <li>1. Gewähltes Problem aus dem Katalog von Karp notieren</li> <li>2. Reduktion des gegebenen Problems, auf das bekannte aus Karp</li> <li>3. Beschreibung der Reduktion</li> <li>4. Schlussfolgerung: Da das Problem aus dem Katalog von Karp NP-vollständig ist, gibt es auch keinen effizienten (polynomiellen) Algorithmus, der das gegebene Problem lösen könnte.</li> </ol>
Graphen	Vertices $v$ = Ecken (singular: Vertex) = Knoten Edges $e$ = Kanten = Verbindungen $e1=\{v1,v2\}$ , $e2=\{v2,v3\}$ , $e3=\{v1, v3\}$ $G1=\{e1, e2, e3\}$
Katalog von Karp	
SAT	SAT = $\{ \phi \mid \phi \text{ ist eine erfüllbare logische Formel} \}$
3SAT	Im Gegensatz zu SAT enthält 3SAT nur Formeln in konjunktiver Normalform, und jede Klausel enthält <b>genau drei Terme</b> . Eine typische Formel in 3SAT ist $\phi = (x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee x_3 \vee x_5)$ . Die Sprache 3SAT ist: 3SAT = $\{ \phi \mid \phi \text{ ist eine erfüllbare 3cnf-Formel} \}$ .
VERTEX-COLORING	Die Vertices eines Graphen $G$ können mit $k$ Farben eingefärbt werden, so dass <b>n</b> ie zwei durch eine Kante <b>verbundene Vertices die gleiche Farbe</b> bekommen 
	Aneinandergrenzende Länder dürfen nie gleich angemalt werden. Wie viele Farben sind dazu minimal nötig um das Problem zu lösen.
	Stundenplanprobleme: Knoten des Graphen $\leftrightarrow$ Zu platzierende Veranstaltungen Kante $\leftrightarrow$ Zwei Veranstaltungen die nicht gleichzeitig stattfinden können (z.B gleicher Dozent) Die möglichen Farben $\leftrightarrow$ Zuteilbare Zeitfenster
BIP (Ganzzahl Optimierung)	BIP ist "binary integer programming", zu einer ganzzahligen Matrix $C$ und einem ganzzahligen Vektor $d$ ist ein binärer Vektor $x$ zu finden mit $Cx = d$
CLIQUE	Eine $k$ -Clique in einem Graph $G$ ist eine Menge von $k$ Ecken des Graphen so, dass in $G$ jede Ecke der Teilmenge mit jeder anderen Ecke verbunden ist. Im umgangssprachlichen Gebrauch ist eine Clique eine Gruppe von Leuten, in der jeder jeden kennt. 
	Job-Parallelisierbarkeit: Gegeben sei eine Menge von $n$ Jobs $J_1, \dots, J_n$ , die jeweils exklusiv auf $m$ Ressourcen $r_1, \dots, r_m$ zugreifen, diese Jobs dürfen nicht gleichzeitig laufen. Zeigen Sie, dass das Problem zu entscheiden, ob mit diesen Jobs zu irgend einem Zeitpunkt mehr als $k$ Prozessoren ausgelastet werden können, NP-vollständig ist. Ecken $\leftrightarrow$ Jobs Kante $\leftrightarrow$ Jobs die gleichzeitig laufen, daher kein Ressourcenkonflikt haben

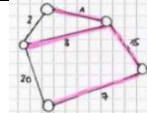
VERTEX-COVER	Gegeben ein Graph $G$ und eine Zahl $k$ , gibt es eine Teilmenge von $k$ Vertices so, dass jede Kante des Graphen ein Ende in dieser Teilmenge hat?  Führt jede Kante des Graphen $g$ auf einen Knoten aus $U$ , welcher eine Teilmenge aus maximal $k$ Knoten ist.  Ein Verkehrsnetz soll regelmässig durch Mitarbeiter kontrolliert werden, die ihre Basis an einzelnen Knotenpunkten des Netzes haben. Kann man auf effiziente Art und Weise herausfinden, an welchen Knotenpunkten man Kontrolleure stationieren muss, damit jede Strecke in einem Knoten mit Kontrolleur endet? Knotenpunkte $\leftrightarrow$ Knoten Strecke $\leftrightarrow$ Kante Anzahl Kontrolleure $\leftrightarrow k$ Knoten mit Kontrolleur $\leftrightarrow$ Knoten aus dem Vertex Cover
SET-PACKING	$k$ Teilmengen, welche sich nicht überlappen. Gegeben eine Familie $(S_i)_{i \in I}$ von Mengen und eine Zahl $k \in \mathbb{N}$ . Gibt es eine $k$ -elementige Teilfamilie $(S_i)_{i \in J}$ mit $I \subseteq J$ , $ J  = k$ , derart, dass die Mengen der Teilfamilie paarweise disjunkt sind, also $S_i \cap S_j = \emptyset \quad \forall i, j \in J$ Ausgedeutcht: Man hat eine Menge die aus $k$ Teilmengen besteht, welche untereinander disjunkt (keine Schnittmenge) sind.
	In einer Küche hat man eine Menge an Zutaten (Eier, Salz, Mehl, Zucker, Bananen,...) und ein Rezeptbuch voller Rezepte. Nun möchte man möglichst viele der Rezepte kochen, ohne dabei eine Zutat mehrmals zu verwenden (z.B. Brot backen, Spaghetti Bolognese, Fruchtsalat)
FEEDBACK-NODE-SET	Gegeben ein gerichteter Graph $G$ und eine Zahl $k$ , gibt es eine endliche Teilmenge von $k$ Vertices von $G$ so, dass jeder Zyklus in $G$ einen <b>Vertex</b> in der Teilmenge enthält? 
	Es gibt mehrere Buslinien. Wo muss das Putzpersonal platziert werden damit alle Linien geputzt werden können. Man möchte dabei möglichst wenig Buspersonal einsetzen.
FEEDBACK-ARC-SET	Gegeben ein gerichteter Graph $G$ und eine Zahl $k$ , gibt es eine Teilmenge von $k$ Kanten so, dass jeder Zyklus in $G$ eine <b>Kante</b> aus der Teilmenge enthält? 
	Das FEEDBACK-ARC-SET bezeichnet eine Menge von Kanten, durch deren Entfernung aus einem Graphen, dieser azyklisch wird.
HAMPATH	$G$ ist ein Graph mit einem hamiltonschen Pfad. also ein geschlossener Pfad in einem Graphen, der jeden Knoten genau einmal enthält. 
	Bereisen der ganzen Schweiz ohne eine Stadt mehr als einmal zu besuchen.
	Der neue CEO einer grossen Fluggesellschaft möchte das Personal in allen Flughäfen persönlich besuchen, die die Fluggesellschaft anfliegt. Er bittet seine Sekretärin, einen optimalen Besuchsplan zusammenzustellen, bei der er jede Destination nur genau einmal besuchen muss. Lösung: Dieses Problem entspricht dem HAMPATH Problem. Dieses Problem ist NP-vollständig, es gibt also keine effizienten Algorithmen für grosse solche Probleme.  HAMPATH $\leq_p$ BESUCH Knoten $\leftrightarrow$ Destination Kante $\leftrightarrow$ Flug hamiltonischer Pfad $\leftrightarrow$ Besuchsplan

U HAMPATH	UHAMPATH ist das Problem in einem ungerichteten Graphen einen hamiltonschen Pfad zu finden. 
	Facebook hat einige hundert Millionen aktiver Mitglieder. Jeder Teilnehmer kann mit jedem anderen Teilnehmer befreundet sein oder auch nicht. Wie aufwendig ist es, die Liste aller Freunde eines Teilnehmers so zu sortieren, dass zwei aufeinanderfolgende Freunde in der Liste untereinander ebenfalls befreundet sind? U HAMPATH $\leq_p$ FACEBOOK Vertices $\leftrightarrow$ Teilnehmer Kanten $\leftrightarrow$ Befreundete Teilnehmer
SET-COVERING	Gegeben eine endliche Familie endlicher Mengen $(S_j)_{j \in J}$ $1 \leq j \leq n$ und eine Zahl $k$ , gibt es eine Unterfamilie bestehend aus $k$ Mengen, die die gleiche Vereinigung hat? Menge $M$ $(a, b), (b, c), (a, c, d, e), (c, d)$ ↓ <b>Vereinigung</b> $(a, b, c, d, e)$ Die Teilmengen 2 und 4 können weggelassen werden, da diese bereits in den Teilmengen 1 und 3 vorkommen. Die Vereinigung bleibt dabei immer noch gleich.
	Mitglieder des sozialen Netzwerkes Twitter folgen einander. Ein Twitter Mitglied kann ein Follower eines anderen Mitglieds sein, das Umgekehrte muss aber nicht zutreffen. Die Twitter-Mitglieder bilden also einen sehr grossen gerichteten Graphen. Mitteilungen einzelner Mitglieder heissen Tweets, sie werden allen Followern angezeigt. Die Follower können die Tweets retweeten, so dass ihre eigenen Follower diese Tweets auch sehen können. Ein Tweet kann also potentiell eine sehr grosse Menge von Twitterern erreichen. Gibt es einen effizienten Algorithmus, mit dem man entscheiden kann, ob es $k$ Twitterer gibt, die zusammen potentiell alle Twitter-Mitglieder erreichen könnten? Lösung: Sei $S_i$ die Menge der Twitterer, die Twitterer $i$ erreichen kann, wenn alle seine Tweets retweetet werden. Es ist klar das $\bigcup_{i \in I} S_i = \{ \text{Twitter Mitglieder} \}$ Alle Twitterer umfasst. Gefragt ist eine $k$ -elementige Teilmenge $I' \subset I$ sodass die Vereinigung der $S_i \in I'$ ebenfalls alle Twitterer umfasst $\bigcup_{i \in I'} S_i = \bigcup_{i \in I} S_i$ Elemente $\leftrightarrow$ Twitterer $i$ Teilmengen $\leftrightarrow$ von $i$ erreichbaren Twitterer $S_i$
CLIQUE-COVER	Gegeben ein Graph $G$ und eine positive Zahl $k$ , gibt es $k$ Cliques so, dass jede Ecke in genau einer der Cliques ist? 
	Für eine Gruppenarbeit sollen $k$ Gruppen gebildet werden. Um die Zeit für das gegenseitige Kennenlernen möglichst kurz zu halten, sollen sich die Leute einer Gruppe bereits gegenseitig kennen. Alle Leute sollen beschäftigt sein. Können Sie einen effizienten Algorithmus formulieren, mit dem eine solche Gruppeneinteilung auch bei einer grossen Teilnehmerzahl gefunden werden kann? Teilnehmer $\leftrightarrow$ Knoten Kennen sich $\leftrightarrow$ Kante Anzahl Gruppen $\leftrightarrow k$ Gruppe $\leftrightarrow$ Clique

EXACT-COVER	<p>Gegeben eine Familie <math>(S_j)_{j \in J}</math> von Teilmengen einer Menge <math>U</math> gibt es eine Unterfamilie von Mengen, die disjunkt sind, aber die gleiche Vereinigung haben? Die Unterfamilie <math>(S_{j_i})_{i \in I}</math> muss also <math>S_{j_i} \cap S_{j_k} = \emptyset</math> und</p> $\bigcup_{j=1}^n S_j = \bigcup_{i=1}^m S_{j_i}$ <p>erfüllen.</p>  <p> <math>X = (a,b,c,d,e,f)</math>  <math>S = ((a,b), (a,b,c), (c,e), (d,f), (e,f))</math>  Die Menge  <math>U = ((a,b), (c,e), (d,f))</math> </p>
	<p>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?</p> <p>Eigenschaft <math>\leftrightarrow</math> Menge <math>S_j</math>  Teilmenge von Eigenschaften <math>\leftrightarrow</math> Unterfamilie <math>S_{j_i}</math>  Genau eine der Eigenschaften <math>\leftrightarrow S_{j_i} \cap S_{j_k} = \emptyset \forall i \neq k</math>  Alle Kunden erfasst <math>\leftrightarrow \bigcup_{j=1}^n S_j = \bigcup_{i=1}^m S_{j_i}</math></p>

3D-MATCHING	<p>Sei <math>T</math> eine endliche Menge und <math>U</math> eine Menge von Tripeln aus <math>T : U \subset T \times T \times T</math>. Gibt es eine Teilmenge <math>W \subset U</math> so, dass <math> W  =  T </math> und keine zwei Elemente von <math>W</math> stimmen in irgendeiner Koordinate überein?</p>  <p>Gegeben: Menge <math>M</math>, 3 Teilmengen von <math>M</math> (z.B. Mann, Frau, Wohnung)  Gesucht: 1 Element aus jeder Teilmenge, die nicht gleich sind.</p>
SUBSET-SUM	<p>Gegeben ist eine Menge <math>S</math> von ganzen Zahlen, kann man darin eine Teilmenge finden, die als Summe einen bestimmten Wert <math>t</math> hat?</p>
	<p>Aus einer Menge von Objekten, die jeweils ein Gewicht und einen Nutzwert haben, soll eine Teilmenge ausgewählt werden, deren Gesamtgewicht eine vorgegebene Gewichtsschranke nicht überschreitet. Unter dieser Bedingung soll der Nutzwert der ausgewählten Objekte maximal werden.</p> 
	<p>My HOBBY:  EMBEDDING NP-COMPLETE PROBLEMS IN RESTAURANT ORDERS</p> 

SEQUENCING	<p>Gegeben sei ein Vektor <math>(t_1, \dots, t_p) \in \mathbb{Z}^p</math> von Laufzeiten von <math>p</math> Jobs, ein Vektor von spätesten Ausführungszeiten <math>(d_1, \dots, d_p) \in \mathbb{Z}^p</math>, einem Strafenvektor <math>(s_1, \dots, s_p) \in \mathbb{Z}^p</math> und eine positive ganze Zahl <math>k</math>. Gibt es eine Permutation <math>\pi</math> der Zahlen <math>1, \dots, p</math>, so dass die Gesamtstrafe für verspätete Ausführung bei der Ausführung der Jobs in der Reihenfolge <math>\pi(1), \dots, \pi(p)</math> nicht grösser ist als <math>k</math>? Formal lautet die Bedingung</p> $\sum_{j=1}^p \theta(t_{\pi(1)} + \dots + t_{\pi(j)} - d_{\pi(j)}) s_{\pi(j)} \leq k,$ <p>darin ist <math>\theta</math> die Stufenfunktion definiert durch</p> $\theta(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0. \end{cases}$  <p>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.</p>
PARTITION	<p>Gegeben ein Folge von <math>s</math> ganzen Zahlen <math>c_1, c_2, \dots, c_s</math>, kann man die Indizes <math>1, 2, \dots, s</math> in zwei Teilmengen <math>I</math> und <math>\bar{I}</math> teilen, so dass die Summe der zugehörigen <math>c_i</math> identisch ist:</p> $\sum_{i \in I} c_i = \sum_{i \in \bar{I}} c_i$ <p>Ein aufstrebendes Film-Festival ist derart gewachsen, dass der Vorführsaal nicht mehr reicht. Daher müssen jetzt zwei gleich grosse Säle verwendet werden, und zwar in einem Masse, dass überhaupt nur Stars und Prominente samt ihrer Entourage eingelassen werden können, für einzelne Besucher gibt es keine Plätze. Doch die Stars stören sich daran, dass sie möglicherweise nicht ihre ganze Entourage im gleichen Saal haben können. Daher muss kurzfristig eine Aufteilung der Festival-Gäste gefunden werden, so dass die beiden Säle so gefüllt werden können, dass jede Entourage vollständig in einem der Säle Platz nimmt.</p> <p>Der Festival-Direktor ist jedoch sehr überrascht, dass die Bestimmung einer solchen Aufteilung so lange dauert. Warum sind Sie nicht überrascht?</p>
MAX-CUT	<p>Der maximale Schnitt eines Graphen ist eine Zerlegung seiner Knotenmenge <math>V</math> in zwei Teilmengen <math>(S, T)</math>, so dass das Gesamtgewicht der zwischen den beiden Teilen verlaufenden Kanten maximal wird. einfacher:  Gesucht ist also ein Subset von Vertizes, so dass die Kanten zwischen dem Subset und dem Komplementär-Set (= das Gewicht) möglichst gross wird.</p>  <p>Feindliche Übernahme einer Firma, mit resultierender Aufteilung der Abteilung, dass diese möglich ineffizient miteinander kommunizieren können.</p> <p>Abteilung <math>\leftrightarrow</math> Vertex  Kommunikationsbeziehung <math>\leftrightarrow</math> Kante  Kommunikationsvolumen <math>\leftrightarrow</math> Gewicht einer Kante</p>

HITTING-SET	<p>Gegeben eine Menge von Teilmengen <math>S_i \subset S</math>, gibt es eine Menge <math>H</math>, die jede Menge in genau einem Punkt trifft, also <math> H \cap S_i  = 1 \forall i</math>?  Einfacher:  Gegeben ist eine Menge von Teilmengen <math>S</math> eines „Universums“ <math>T</math>, gesucht ist eine Teilmenge <math>H</math> von <math>T</math> so, dass jede Menge in <math>S</math> mindestens ein Element aus <math>H</math> enthält. Zusätzlich ist gefordert, dass die Anzahl der Elemente von <math>H</math> einen gegebenen Wert <math>k</math> nicht überschreitet.</p> <p>Aus einer Menge von Fachleuten, die zum Teil in mehreren Gebieten <math>i = 1, \dots, n</math> tätig sind, soll eine Expertenkommission gebildet werden. Da zwei Experten für das gleiche Fachgebiet sich erfahrungsgemäss immer streiten, will man in der Expertenkommission jedes Fachgebiet durch genau einen Experten vertreten haben. Können Sie einen effizienten Algorithmus zur Auswahl der Mitglieder der Kommission angeben?</p> <p>Experte <math>\leftrightarrow</math> Punkt in <math>S</math>  Fachgebiet <math>i \leftrightarrow</math> Teilmenge <math>U_i</math> aller Experten für dieses Gebiet  Expertenkommission <math>\leftrightarrow</math> Hitting Set <math>H</math></p>
STEINER-TREE	<p>Gegeben ein Graph <math>G</math>, eine Teilmenge <math>R</math> von Vertices, und eine Gewichts-funktion <math>w : E \rightarrow \mathbb{Z}</math> und eine positive Zahl <math>k</math>, gibt es einen Baum mit Gewicht <math>\leq k</math>, dessen Knoten in <math>R</math> enthalten sind?  Das Gewicht des Baumes ist die Summe der Gewichte <math>w(\{u, v\})</math> über alle Kanten <math>\{u, v\}</math> im Baum.</p>  <p>Der STEINER-TREE ist auch eine Verallgemeinerung des minimalen SPANNING-TREE. Es wird der kürzeste Graph gesucht, der endlich viele gegebene Punkte miteinander verbindet und auf diese Weise das kürzeste Wegetz zwischen diesen Punkten bildet.</p> <p>In einem Entwicklungsland sollen die aus dem Ausland erhaltenen Unterstützungsmittel dazu verwendet werden, endlich alle Ortschaften mit mindestens 100 Einwohnern ans Stromnetz anzuschliessen. Der Bau von Leitungen zwischen einzelnen Ortschaften ist je nach Gelände unterschiedlich teuer, zum Teil auch schlicht unmöglich. Es wird entschieden, dass man in einer ersten Phase auf Redundanz des neu zu erstellenden Netzes verzichten will. Der Minister möchte endlich wissen, ob das vorhandene Geld für das Projekt ausreicht, und ist sehr ungehalten darüber, dass die Verwaltung so lange braucht, diese Frage zu beantworten. Kann man dies erklären?</p> <p>Steiner Tree <math>\leq_p</math> Stromnetz  Knoten <math>\leftrightarrow</math> Ortschaften  Knoten in <math>R \leftrightarrow</math> zu erschliessende Ortschaften  Gewicht <math>w</math> einer Kante <math>\leftrightarrow</math> Baukosten einer Verbindungsleitung  Maximales Gewicht <math>k \leftrightarrow</math> Budget</p>