Prüfungszusammenfassung Mathe 2

Begriffe	
DEA (Deterministischer endlicher Automat; DFA)	Ohne Mehrdeutigkeiten: Darf nicht vom gleichen Zustand aus mehr als ein Übergang mit demselben Buchstaben enthalten.
NEA (Nichtdeterministischer endlicher Automat; NFA)	Mit Mehrdeutigkeiten: Ein Automat der mehrere Übergänge mit demselben Buchstaben vom gleichen Zustand aus enthält.
regulär	Eine Sprache ist *regulär*, wenn sie durch einen regulären Ausdruck, eine reguläre Grammatik oder einen endlichen Automaten erkannt werden kann.

Chomsky Hierarchie		
regulär	endlicher Automat	Typ-3
kontextfrei	nicht det. PDA (Stackautomat)	Typ-2
kontextsensitiv	TM (…)	Typ-1
rekursiv aufzählbar	Turing Maschine	Typ-0

$T3 \subset T2 \subset T1 \subset T0$ (die Teilmengen sind echt)

Abgeschlossenheitseigenschaften					
Klasse	**Durchschnitt**	**Vereinigung**	**Komplement**	**Konkatenation**	**Kleene-Stern**
Typ-3	ja	ja	ja	ja	ja
DKF	nein	nein	ja	nein	nein
Typ-2	nein	ja	nein	ja	ja
Typ-1	ja	ja	ja	ja	ja
Typ-0	ja	ja	nein	ja	ja

Grammatiken

Grammatik: (V, Σ, P, S)

V = Variable („Satzteil“)	Σ = Terminalsymbol (Buchstaben, reservierte Wörter)
S = Startvariable	P = Produktionsregel ($A \to w, A \in V, w \in (\Sigma \cup \mathrm{V})*$

Kontextfreie Grammatik: Wenn alle Regeln (Produktionen) die Form haben: $l \to r, l \in V$
Die Bezeichnung kontextfrei erklärt sich daraus, dass bei der Anwendung einer Regel der Form $X \to r$ in der Ableitung eines Wortes der Kontext, in dem die Variable X steht, also das links und rechts von X stehende Teilwort, keine Rolle spielt. Eine Regel der Form $aXb \to r$ dagegen ist nur anwendbar, wenn die Variable X im richtigen Kontext steht.

Grammatik konstruieren, Beispiel
Ausdrücke der Aussagenlogik in zwei Prädikaten p und q und den Verknüpfungsoperationen ∧,∨, ¬.
Produktionsregeln: $S \to S \wedge \mathrm{S} \mid \mathrm{S} \vee \mathrm{S} \mid \mathrm{S} \neg \mathrm{S} \mid (\mathrm{S}) \mid \mathrm{p} \mid \mathrm{q}$

Grammatik für Palindrome über Σ = {a,b}
$S \to aSa \mid bSb \mid a \mid b \mid \varepsilon$

Elimination von ε-Regeln
Wird ein ε entfernt, muss bei jeder Regel diese Regeln hinzugefügt werden, wo die Variable vorkommt, bei dessen Regel das ε entfernt wurde. zB.: $D \to BD \mid \varepsilon$, wird das ε entfernt, gibt es die neue Regel $D \to B$, also gibt es dann: $D \to BD \mid B$

$S \to B0C \mid CB$ $B \to 1B \mid \varepsilon$ $C \to 0C \mid DB$ $D \to BD \mid \varepsilon$	$S \to B0C \mid CB$ $B \to 1B \mid \varepsilon$ $C \to 0C \mid DB \mid B$ $D \to BD \mid B$	$S \to B0C \mid CB \mid 0C \mid C$ $B \to 1B \mid 1$ $C \to 0C \mid DB \mid B \mid D \mid \varepsilon$ $D \to BD \mid B \mid D \mid \varepsilon$	$S \to B0C \mid CB \mid 0C \mid C$ $B \to 1B \mid 1$ $C \to 0C \mid DB \mid B \mid D \mid \varepsilon$ $D \to BD \mid B \mid D$

Lösung:
$S \to B0C \mid CB \mid 0C \mid C \mid B0 \mid B \mid 0 \mid \varepsilon$
$B \to 1B \mid 1$
$C \to 0C \mid DB \mid B \mid D \mid 0$
$D \to BD \mid B$

Wenn eine Regel übrig bleibt wie zB.: $A \to B \mid BY$ aber Y wurde entfernt, weil es eine (reine) ε-Regel war, dann kann BY weggelassen werden, weil Y nie auf der linken Seite einer Regel vorkommt.

Elimination von Kettenregeln
Alle Regeln der Form $X \to Y$ müssen ersetzt werden. Dazu ersetzt man einfach ein Symbol der Regel, durch die Symbole der entsprechenden Variable. zB.: $S \to X$ und $X \to K \mid 12$, dann kann man überall dort wo das X rechts vorkommt, dieses ersetzen durch $K \mid 12$, also gibt es: $S \to K \mid 12$

$S \to ASB \mid C \mid BS$ $A \to 1A \mid B$ $B \to C \mid A$ $C \to D \mid AC$ $D \to DD \mid 0$	$S \to ASB \mid D \mid AC \mid BS$ $A \to 1A \mid B$ $B \to C \mid A$ $C \to D \mid AC$ $D \to DD \mid 0$	$S \to ASB \mid D \mid AC \mid BS$ $A \to 1A \mid B$ $B \to C \mid A$ $C \to DD \mid 0 \mid AC$ $D \to DD \mid 0$	$S \to ASB \mid DD \mid 0 \mid AC \mid BS$ $A \to 1A \mid AC \mid DD \mid 0$ $B \to 1A \mid AC \mid DD \mid 0$ $C \to DD \mid 0 \mid AC$ $D \to DD \mid 0$

(Zellen 1-3 enhalten mögliche erste drei Schritte, die letzte Zelle das Endergebnis)

CNF – Chomsky-Normalform
Eine kontextfreie Grammatik G mit $\varepsilon \notin \mathcal{L}(G)$ heisst in *Chomksy-Normalform*, wenn alle ihre Regeln von einer der beiden folgenden Formen sind: $\boldsymbol{A \to BC}$ oder $\boldsymbol{A \to a}$
Jede kontextfreie Grammatik G mit $\varepsilon \notin \mathcal{L}(G)$ lässt sich in Chomsky-Normalform transformieren. Die Transformation beginnt mit folgenden Schritten:
1. Elimination von ε-Regeln, **2.** Elimination von Kettenregeln, **3.** Elimination nutzloser Variablen

Jetzt müssen noch die Regeln der Form $A \to w$ in die Form $A \to BC$ gebracht werden. Dazu wird für jedes Terminalsymbol s, das in w vorkommt, eine neue Variable X_s und eine Regel $X_s \to s$ eingeführt. Dann s in w durch X_s ersetzen.

Nun müssen noch Regeln, die rechts mehr als eine Variable haben, so geändert werden, dass rechts genau zwei Variablen sind.
D.h.: $U \to V_1 \dots V_k$ (wenn k > 2) wird zu: $U \to V_1 W_1, W_1 \to V_2 W_2, \dots, W_{k-2} \to V_{k-1} V_k$

Automat zu einer Grammatik

$S \to 0S \mid 0A \mid 1S$
$A \to 1B$
$B \to 0C$
$C \to \varepsilon$

	0	1
Z_S	Z_S, Z_A	Z_S
Z_A	-	Z_B
Z_B	Z_C	-
Z_C / E	-	-

<u>Konstruktion des Minimalautomaten</u>
1. Tabelle mit allen Zuständen erstellen (horizontal und vertikal)
2. Diagonale als äquivalent kennzeichnen (mit Ξ), weil da jeder Zustand mit sich selber verglichenwird

3. Alle Endzustände als nicht-äquivalent markieren (mit x) (die ganze Horizontale und Vertikale, ausser dort wo bereits ein Ξ ist)
4. Mit allen Zuständen Paare (z, z‘) Bilden und im Zustandsdiagramm des Automaten die sich ergebenden Zustände auf eine Markierung (mit einem x) prüfen. Falls sie markiert sind, kann (z, z‘) auch markiert werden.
5. Schritt 4 wiederholen, bis keine neuen Markierungen mehr gemacht werden können.
zB.: man bildet das Paar (z_0, z_1) und sieht, dass in der Tabelle die zugehörigen Zustände (z_2, z_3) bereits markiert sind, also kann nun das neue Paar (z_0, z_1) auch markiert werden.

	0	1
z_0	z_1	z_2
z_1	z_2	z_3

Hat man das gemacht, weiss man, welche Zustände äquivalent sind. Nun kann man den Automaten neu zeichnen; Einfach je einer der äquivalenten Zustände weglassen. Sind zB. zustand A und B äquivalent und B geht nach C und D nach B, dann kann man B weglassen und muss bei A folgendes zusätzlich machen: A nach D und D nach A (die waren ja von B).

Stackautomat / Kellerautomat / PDA
$a, \varepsilon \rightarrow b$: Wenn ein ‘a‘ kommt, nimm nichts (ε) vom Stack und lege ein ‘b‘ darauf.

Übergangsfunktion
$\delta_1: (z, A) \rightarrow_a (z', W)$

Befindet sich Automat K im Zustand z und liest das Eingabezeichen a und das oberste Kellerzeichen A, so wechselt er in Zustand z‘ und ersetzt das oberste Kellerzeichen A durch das Wort W.

PDA aus CNF
1. Zuerst normal $ und ein S (für das Startsymbol der Grammatik) auf den Stack legen
2. Regeln der Form $A \rightarrow BC$: Pfeil von $Z_n \; nach \; Z_{n+1}$ $(\varepsilon, A \rightarrow C)$ und von Z_{n+1} zurück nach Z_n $(\varepsilon, \varepsilon \rightarrow B)$
3. Regeln der Form $A \rightarrow a$: Pfeil von $Z_n \; nach \; Z_n$ $(\varepsilon, A \rightarrow a)$
4. Für jedes Zeichen des Alphabets: Pfeil von $Z_n \; nach \; Z_n$ $(a, a \rightarrow \varepsilon)$ $(a \in \Sigma)$
5. Pfeil zum Endzustand: $\varepsilon, \$ \rightarrow \varepsilon$

Myhill-Nerode
Sei L eine Sprache über Σ. Ist $w \in \Sigma*$, so sei $L(w) = \{v \in \Sigma * | \text{wv} \in \text{L}\}$
Dann gilt: **a)** Ist L regulär, dann ist die Menge L(w) endlich. **b)** Ist L(w) endlich, so ist L regulär. **c)** In Fall b) gibt |L(w)| die Anzahl Zustände eines minimalen DEA an, der L akzeptiert.

Beispiel
Zeige, ob die folgende Sprache regulär ist: $L = \{0^n 1^m | m \geq 0, n > m\}$

Nun macht man ein $Wn = 0^n 1$ (das 1 am Schluss wird gebraucht, damit man das Wort nicht mit beliebig vielen 0en fortsetzen kann, so wird das fortsetzen mit 1en erzwungen). Jetzt muss die Menge $L(Wn)$ bestummen werden: $L(Wn) = \{1^k \big| k \leq n - 2\}$, da nun k beliebig gross gemacht werden kann, wird auch die Menge $L(Wn)$ beliebig gross, daher ist die Sprache nicht regulär.

Pumping Lemma

Mit dem Pumping Lemma kann man ebenfalls zeigen, dass eine gegebene Sprache nicht regulär ist. Wenn die Häufigkeit eines Zeichens von der eines anderen abhängt, so ist diese Sprache im Allgemeinen nicht regulär: $L = \{a^m b^k c^{2m}\}$: c hängt von der Häufigkeit von a ab.

Sei L eine reguläre Sprache. Dann existiert eine Zahl $N \geq 0$, sodass jedes Wort $x \in L$ mit $|x| \geq N$ in der folgenden Form schreiben lässt.

$\boldsymbol{x = uvw}$ mit $\boldsymbol{|v| \geq 1}$ und $\boldsymbol{|uv| \leq N}$

und für alle $i \geq 0$ gilt, dass $uv^i w \in L$ ist.

Beispiel

Ist die Sprache $L = \{a^m b^m | m \geq 1\}$ regulär?
Angenommen L ist regulär, dann gibt es gemäss Pumping Lemma eine Zahl N sodass sich alle Wörter $x \in L$ mit $|x| \geq N$ wie beschrieben zerlegen lassen.
Nun wählt man speziell das Wort $\boldsymbol{x = uvw = a^N b^N}$. Da $\boldsymbol{|uv|}$ **kleiner oder gleich N** und **v nicht leer** ist (laut Definition), besteht uv und somit auch v also nur aus a's.
Nun kann man mit $uv^i w$ aufpumpen, wodurch es plötzlich mehr a's als b's hat (weil v ja nur aus a's besteht). L ist also nicht regulär.

Beispiel 2

Man soll zeigen, dass die Sprache der Palindrome nicht regulär ist.
w ist ein Palindrom, d.h.: $w = a^N b a^N$
Dann weiter wie oben; uv kommt komplett im ersten a vor ($|uv| \leq N$) da und somit wird beim Aufpumpen die Irregularität gezeigt.

Beispiel 3

Zeige, dass korrekt geschachtelte Klammerausdrücke nicht regulär sind.
w = (…()…) (man wählt das Wort w so, dass beide markierten Teile jeweils die Länge N haben. Dadurch lässt sich wieder der linke Teil aufpumpen.)

Pumping Lemma für kontextfreie Sprachen

Sei L eine kontextfreie Sprache. Dann gibt es eine Konstante N, sodass sich alle Wörter $z \in L$ mit $\boldsymbol{|z| \geq N}$ in $\boldsymbol{z = uvwxy}$ zerlegen lassen, so dass gilt:

1. $|vx| \geq 1$
2. $|vwx| \leq N$
3. $\forall i \geq 0: uv^i wx^i y \in L$

Man weiss jeweils nicht, wo das Teilwort vwx liegt, also gibt es mehrere Möglichkeiten!

Um zu zeigen, dass eine gegebene Sprache L nicht kontextfrei ist, weist man nach, dass L die im Pumping-Lemma konstatierte Eigenschaft nicht besitzt.

Nur weil eine Sprache die Eigenschaft des Pumping-Lemma erfüllt, heisst das nicht zwingend, dass sie kontextfrei ist. Es kann sein, muss aber nicht.

Beispiel

Sprache mit $\Sigma = \{a, b, c, d\}$. Man soll zeigen, dass die Sprache $L = \{w \in \Sigma^* | \, |w|_a = |w|_b \wedge |w|_c = 3 * |w|_d \}$ nicht kontextfrei ist. Dazu wählen wir das Wort: $a^N c^{3N} b^N d^N$.n Wo genau der Teil vxy liegt wissen wir nicht, aber wegen $|vxy| \leq N$ können höchstens zwei der Buchstaben darin vorkommen, und zwar nur die Paare (a, c), (c, b) oder (b, d). Beim Pumpen ändert sich die Anzahl der Buchstaben und kein Paar enthält Buchstaben der Beziehung: $|w|_a = |w|_b$ oder $|w|_c = 3 * |w|_d$. somit wird sich beim Aufpumpen jeweils eine Seite der Gleichungen ändern, die andere jedoch nicht. Dies führt zu einem Widerspruch und zeigt, dass die Sprache nicht kontextfrei ist.

Regex

?	Der voranstehende Ausdruck ist optional, er kann einmal vorkommen, muss es aber nicht, d. h. der Ausdruck kommt null- oder einmal vor. (Dies entspricht `{0,1}`)
+	Der voranstehende Ausdruck muss mindestens einmal vorkommen, darf aber auch mehrfach vorkommen. (Dies entspricht `{1,}`)
*	Der voranstehende Ausdruck darf beliebig oft (auch keinmal) vorkommen. (Dies entspricht `{0,}`)
{*n*}	Der voranstehende Ausdruck muss exakt *n*-mal vorkommen.
{*min*,}	Der voranstehende Ausdruck muss mindestens *min*-mal vorkommen.
{,*max*}	Der voranstehende Ausdruck darf maximal *max*-mal vorkommen.
{*min*,*max*}	Der voranstehende Ausdruck muss mindestens *min*-mal und darf maximal *max*-mal vorkommen.
[A-Za-z0-9]	Ein beliebiger lateinischer Buchstabe oder eine beliebige Ziffer
[^a]	Ein beliebiges Zeichen außer „a“ („^“ am Anfang einer Zeichenklasse negiert selbige)
^	Steht für den Zeilenanfang (nicht zu verwechseln mit zB.: [^a])
$	Steht je nach Kontext für das Zeilen oder Stringende.

Turing Maschine

Definition: M = (Z, Σ, Γ, δ, z_0, ц, E)

Z = Zustände	z_0 = Startzustand	Σ = Eingabealphabet	Γ = Bandalphabet
ц ∈ Γ\Σ = Blank	E = Endzustände	$\delta: Z \times \Gamma \to Z \times \Gamma \times \{L, R\}$	

DEA

$0 \to$ ц, R: Wenn eine 0 kommt, überschreibe sie mit Blank und gehe eins nach rechts

$a \to R$: Wenn ein a kommt, gehe eins nach rechts

Konfiguration

(v, z, w): v = Wort links vom Zeiger, z = aktueller Zustand, w: Wort rechts vom Zeiger

Nicht deterministische TM

Es gibt mehrere Wahlmöglichkeiten zur Laufzeit um aus einem Zustand zum nächsten zu gelangen. Also ist ein Orakel notwendig, welches sagt, welche Wahl günstig ist. 3 Bänder: 1. Band mit Input (unverändert), 2. Zählerband für Orakel, 3. Arbeitsband

Algorithmus: Band 3 mit Band 1 überschreiben, TM auf Band 3 laufen lassen, wenn ursprüngliche TM akzeptiert: akzeptieren, sonst: Zähler auf Band 2 erhöhen, von vorne beginnen.

Beispielaufgabe
Seien L_1 und L_2 Sprachen über dem Alphabet Σ, die von Turing-Maschinen M_1 und M_2 erkannt werden. Konstruieren Sie eine Turing-Maschine, die folgende Sprachen erkennt: „
1. $L_1 \cup L_2$: TM mit 2 Bändern. Kopiere Wort auf beide Bänder und lasse M1 auf den ersten und M2 auf dem zweiten Band laufen. Wenn eine von beiden akzeptiert, akzeptiere ebenfalls.
2. $L_1 \cap L_2$: Wie 1., aber nur akzeptieren, wenn beide TM akzeptieren.
3. $\Sigma^* \backslash L_1$: Nicht möglich: Halteproblem

Gödel Nummer
Jede TM kann in eine Binärzahl codiert werden ($n \in \mathbb{N}$): ist die Gödelnummer der TM
Damit kann eine TM auf einer anderen TM simuliert werden.

Entscheidungsprobleme				
Klasse	**Wortproblem**	**Leerheitsproblem**	**Endlichkeitsproblem**	**Äquivalenzproblem**
Typ-3 (reg.)	ja	ja	ja	ja
DKF	ja	ja	ja	?
Typ-2 (kf.)	ja	ja	ja	nein
Typ-1 (ks.)	ja	nein	nein	nein
Typ-0	nein	nein	nein	nein

DKF = deterministisch kontextfrei (akzeptiert von det. Kellerautomaten)

NP = Nichtdeterministisch Polynomiell
NP bezeichnet die Klasse der Entscheidungsprobleme, die von einer **nichtdeterministischen TM** bezüglich der Eingabelänge in Polynomialzeit entschieden werden können.
P bezeichnet die Klasse der Entscheidungsprobleme, die von einer deterministischen TM in Polynomialzeit lösbar sind.

Satz von Rice
Sei E irgendeine nichttriviale Eigenschaft berechenbarer Funktionen. Dann ist das folgende Problem unentscheidbar: Gegeben: Eine Turing-Maschine M, Gefragt: Hat f_M die Eigenschaft E?

Triviale Aussagen sind solche, die immer wahr oder falsch sind.
Beispiel
Die Eigenschaft einer TM unendlich viele Wörter zu akzeptieren, ist nichttrivial. Denn es gibt TM die keine Wörter akzeptieren und solche, die alle Wörter akzeptieren. Also ist die Aussage nichttrivial, weil sie manchmal wahr und manchmal falsch ist.
Beispiel 2
Man soll beweisen, dass nicht entscheidbar ist, dass eine TM M mit Σ={0,1} das Wort 1011 akzeptiert.
Lösung: Dass M das Wort 1011 akzeptiert ist gleichbedeutend mit der Eigenschaft von f_M, dass 1011 im Definitionsbereich von f_M enthalten ist. Diese Eigenschaft ist nichttrivial (denn es gibt TM die 1011 akzeptieren und solche die 1011 nicht akzeptieren) und deshalb nach dem Satz von Rice nicht entscheidbar.

Leerheitsproblem / Markierungsalgorithmus
Man hat einen Automaten. Nun markiert man alle Endzustände. Dann markiert man alle Ursprungszustände, der bereits markierten Zustände. Dies wiederholt man, bis keine neuen Zustände mehr markiert werden können.
Ist der Startzustand markiert, ist die vom Automaten akzeptierte Sprache NICHT leer.
Ist der Startzustand **NICHT** markiert, ist die vom Automaten akzeptierte Sprache leer.

Es geht auch für CFG. Dann gibt es nämlich einen Parse-Tree:
1. markiere Terminalsymbole, **2.** markiere A in $A \rightarrow [xy\ das\ bereits\ markiert\ ist]$, **3.** markiere A in $A \rightarrow XY$ (wobei X und Y beide markiert sind), **4.** Schritte 2/3 wiederholen, bis keine neuen Markierungen mehr gemacht werden. **Entscheid: S markiert**: die akzeptierte Sprache ist **nicht leer**, **S nicht markiert**: die akzeptierte Sprache ist **leer**.

Halteproblem
Das allgemeine Halteproblem ist für Turingmaschinen unentscheidbar. Man stelle sich die Tabelle vor, die ALLE TM beinhaltet. Nun gibt es eine TM H, die einen Eintrag (j,j) aus der Tabelle liefert und das negierte Ergebnis zurückgibt. Da diese Tabelle aber ALLE TM enthält, müsste diese Tabelle auch die TM H beinhalten. Steht in der Tabelle, dass H das Wort w akzeptiert, würde H also laut der Tabelle zurückliefern, dass H das Wort w NICHT akzeptiert. Dies führt zum Widerspruch und zeigt, dass das Problem unentscheidbar ist.

Man kann nun praktisch alle Unentscheidbarkeitsresultate auf das grundlegende allgemeine Halteproblem zurückführen.

Endlichkeitsproblem
Ist das folgende Problem für einen DEA A entscheidbar: $|L(A)| < \infty$
Ein endlicher Automat akzeptiert genau dann unendlich viele Wörter, wenn sein Graph eine Schleife enthält die vom Ausgangszustand erreichbar ist und von der man in einen Endzustand gelangen kann. Dies erkennt man daran, dass im regulären Ausdruck ein * vorkommt, der nicht nur auf das leere Wort wirkt.

GRAPH
Polynomialer Algorithmus: 1. Start markieren, 2. die nächsten möglichen Ecken markieren (den Pfeilen folgend), 3. Schritt 2 wiederholen, bis keine neuen Markierungen mehr gemacht werden können. 4. Prüfe, ob Zielpunkt (t) markiert ist.

<u>Komplexität</u>

Gross-O	**Klein-o**
$f(n) = O\big(g(n)\big) \Leftrightarrow$ für n gross genug, ist g im Wesentlichen grösser. $\Leftrightarrow \exists n_0, c \quad f(n) \leq c * g(n)$ für $n \geq n_0$	$f(n) = o\big(g(n)\big) \Leftrightarrow \lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) = 0$ g wächst schneller als f

Für klein-o einfach immer $\frac{f(n)}{g(n)}$ aufschreiben, kürzen und wenns 0 gibt; ja, sonst nein

Addition von 2 Zahlen: O(n)	Multiplikation von 2 Zahlen: O(n*log(n))
Division von 2 Zahlen: $O(n^2)$	ggT: $O(n^3)$

Ein **deterministischer Algorithmus** ist ein Algorithmus, bei dem nur definierte und reproduzierbare Zustände auftreten. Anders gesprochen heißt das, auf eine Anweisung im Algorithmus folgt unter den gleichen Voraussetzungen immer die gleiche Anweisung. Zu jedem Zeitpunkt ist der nachfolgende Abarbeitungsschritt des Algorithmus eindeutig festgelegt. Das bedeutet auch, dass alle Zwischenergebnisse innerhalb des Algorithmus immer gleich sind. Ein deterministischer Algorithmus ist immer determiniert, d.h. er liefert bei gleicher Eingabe immer die gleiche Ausgabe.

Im Umkehrschluss können bei einem **nichtdeterministischen Algorithmus** nicht reproduzierbare und undefinierte Zustände auftreten. Zum Beispiel hat ein Algorithmus, der eine (theoretische) Zufallszahl liefert, ein nichtdeterministisches Verhalten.

Nichtdeterministische Turingmaschinen: Sie ermöglichen es einem Algorithmus quasi zu „raten". Damit werden viele Probleme mit sehr viel weniger Aufwand lösbar. Solche Turingmaschinen definieren in der Komplexitätstheorie eine eigene Komplexitätsklasse.

CLIQUE
Ein Teilgraph, in dem jede Ecke mit jeder anderen Ecke des Teilgraphen verbunden ist heisst clique. k-clique entspricht einer clique mit k Ecken.

Es ist gefragt, ob zu einem einfachen Graph G und einer Zahl k in G eine Clique der Größe mindestens k existiert. Das heißt, ob G mindestens k Knoten enthält, die untereinander paarweise verbunden sind.

langsamer Algorithmus: - h Teilgraphen bilden $O\left(\binom{n}{k}\right)$ - testen	**nicht-deterministischer Algorithmus:** - errate Lösung (TM fragt Orakel) - teste
polynomialer Verifizierer: V, Input: Graph G, k Punkte aus dem Graphen Lösung c= {n_1, …, x_n} ⊂ G 1. Nachzählen, dass c genau k-Punkte Enthält: O(n) 2. Alle Paare (x,y) ∈ c testen, ob die Kante xy in G ist: O(n^2) => CLIEQUE ∈ NP	
Algorithmus für die Zahlvariante sei cl1(G,k) ein Algorithmus, der berechnet, ob der Graph eine k-Clique enthält. Algorithmus cl2(G) for i := |V| to 1 if cl1(G,i) then return i return „unlösbar"	

Seien A1 und A2 Sprachen, die von einer NTM in polynomialer Zeit entscheidbar sind. Man soll zeigen dass:
a) $A1 \cap A2$ ist in NP, b) A1A2 ist in NP
Man muss nur nachweisen, dass es einen Verifizierer gibt, der polynomiale Laufzeit hat. Dabei kann man davon ausgehen, dass V_1 in polynomialer Zeit p1(n) $w_1 \in A_1$ verifiziert, und analog für V_2.

a) Um $w \in A1 \cap A2$ zu verifizieren, muss man nur beide Verifizierer V1 und V2 anwenden, wozu Laufzeit p1(n) + p2(n) notwendig ist.
b) Aus V1 und V2 konstruiert man einen Verifizierer V, der w = w1w2, $w1 \in A1$ und $w2 \in A2$ in Zeit n + p1(n) + p(n), also in polynomialer Zeit verifiziert.

Man betrachte das Problem, von einem endlichen Automaten zu entscheiden, ob er alle Wörter akzeptiert.
a) Ist dieses Problem entscheidbar? Ja, wenn man den minimalen Automaten bildet und dieser nur einen Zustand hat, welcher Endzustand ist, akzeptiert er alle Wörter.
b) Ist dieses Problem in polynomialer Zeit entscheidbar? Die Reduktion eines Automaten auf den minimalen Automaten ist in höchstens |Z| Iterationen möglich, die jeweils eine Analyse einer Tabelle erforderlich, deren Grösse durch die Anzahl der Zustände und die Grösse des Alphabetes

limitiert ist, mithin durch ein Polynom in n. Damit ist der Aufwand f¨ur die Ermittlung des minimalen Automaten polynomial in n.

SAT

Gegeben ist eine aussagenlogische Formel φ in Klauselnormalform. Gefragt ist ob φ erfüllbar ist.

Man soll zeigen dass DOUBLE-SAT in polynomial reduzierbar ist auf SAT:
$\mathrm{DOUBLE-SAT} = \{\varphi | \varphi$ kann mit mindestens zwei Verschiedenen Zuordnungen von Wahrheitswerten erfüllt werden$\}$

Man braucht eine Formel, die genau dann erfüllbar ist, wenn φ auf zwei Arten erfüllbar ist:

$$\psi = \varphi(x_1, \ldots, x_n) \wedge \varphi(x'_1, \ldots, x'_n) \wedge \bigwedge_i ((x_i \wedge \overline{x'_i}) \vee (\overline{x_i} \wedge x'_i))$$

LOOP

x := 1; x := y + 1;

LOOP x DO //x ist unveränderlich
  P
END
	IF x=0 THEN
	  P
	END

x = y * z:
x := 0
LOOP y DO
  LOOP z DO
  x := x + 1
  END
END
	WHILE x>0 DO
	  P
	END

l Studenten $S_1 \ldots S_l$ sollen Prüfungen $P_1 \ldots P_k$ ablegen, jeder Student legt nur eine Teilmenge von Prüfungen ab. Die Prüfungen sollen so in h Slots angeordnet werden, dass jeder Student alle seine Prüfungen ablegen kann. Formulieren Sie dieses Problem als Wortproblem für eine Sprache und zeigen Sie, dass es NP-vollständig ist.
Hinweis: Führen Sie dieses Problem auf das Problem zurück, einen Graphen mit einer Anzahl Farben so einzufärben, dass keine zwei mit einer Kante verbundene Ecken die gleiche Farbe haben. Es ist bekannt, dass dieses Problem NP-vollständig ist.
Lösung: Das Prüfungsplanproblem kann wie folgt in ein Vertex-Coloring-Problem abgebildet werden. Die Knoten entsprechen den Prüfungen, die Farben den Zeit- Slots. Die Studenten haben sich für Prüfungen angemeldet, zwei Prüfungen dürfen nicht in gleichzeitige Slots eingeteilt werden, wenn ein Student beide Prüfungen ablegen will. Dies kann dadurch verhindert werden, dass eine Kante zwischen zwei Prüfungen gebildet wird, wann immer ein Student die beiden Prüfungen belegt hat. Eine Lösung des so konstruierten Vertex-Coloring Problems ist auch eine Lösung des Prüfungsplan-Problems. Damit ist noch nicht gezeigt, dass das Prüfungsplan-Problem NP-vollständig ist, denn dazu muss umgekehrt gezeigt werden, dass das NP-vollständige Problem VERTEX-COLORING in polynomialer Zeit auf das Prüfungsplan-Problem reduziert werden kann. Gegeben ein Graph G und eine Menge von Farben C, konstruieren wir ein Prüfungsplan-Problem wie folgt. Die Ecken betrachten wir als Prüfungen, die Farben als Zeitslots. Für jede Kante erfinden wir einen Studenten, der die beiden von der Kante verbundenen Prüfungen ablegen will. Dieses Prüfungsplanproblem ist genau dann lösbar, wenn das ursprüngliche Vertex- Coloring-Problem lösbar ist.