Zusammenfassung der Vorlesung Berechenbarkeit und Komplexität

Von Simon Faust (simon.faust@post.rwth-aachen.de)

KAPITEL I

Definition I.1.1:
Eine endliche, nicht-leere Menge Σ heißt Alphabet.
Die Elemente eines Alphabets werden Buchstaben (Zeichen, Symbole) genannt.

Definition I.1.2:
Sei Σ ein Alphabet. Ein Wort über Σ ist eine endliche Folge von Buchstaben aus Σ.
Das leere Wort λ (ε) ist die leere Buchstabenfolge.
Die Länge $|\omega|$ eines Wortes ω ist die Länge des Wortes als Folge.
Σ^* ist die Menge aller Wörter über .
$\Sigma^+ = \Sigma^* - \{\lambda\}$

Definition I.1.3:
Die Verkettung (Konkatenation) für ein Alphabet Σ ist eine Abbildung $K: \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$, so daß $K(x,y) = x \bullet y = xy$

Definition I.1.4:
Sei Σ ein Alphabet. $\forall x \in \Sigma^* : x^\circ = \lambda$, $x^1 = x$, $x^i = x \bullet x^{(i-1)}$ für $\forall i \in \{1,.....\}$
Seien $v, w \in \Sigma^*$:
v heißt Teilwort von $w \Leftrightarrow \exists x,y \in \Sigma^* : w = xvy$
v heißt Suffix von $w \Leftrightarrow \exists x \in \Sigma^* : w = xv$
v heißt Präfix von $w \Leftrightarrow \exists y \in \Sigma^* : w = vy$
v heißt echtes Teilwort von $w \Leftrightarrow v \neq w$ und v ist ein Teilwort von w.

Definition I.1.5:
Eine Sprache L über einem Alphabet Σ ist ein Teilmenge von Σ^*.
$L_\varnothing = \varnothing$ ist die leeree Sprache und $L_\lambda = \{\lambda\}$ ist die einelementige Sprache, die nur aus dem leeren Wort besteht.
Sind L_1 und L_2 Sprachen über Σ, so ist

$$L_1 \bullet L_2 = L_1L_2 = \{ vw \mid v \in L_1 \text{ und } w \in L_2 \}$$

Ist L eine Sprache über Σ, so definieren wir

$$L^\circ := L_\lambda , L^{(i+1)} = L^i \bullet L \text{ für alle } i \in \mathbb{N}$$

$$L^* = \bigcup L^i , i \in \mathbb{N} \text{ und } L^+ = \bigcup L^i , i \in \mathbb{N} = L \bullet L^*$$

Definition I.1.6:
Sei X eine Menge. Dann ist $|X|$ die Anzahl der Elemente in X.
Sei $x \in \Sigma^*$ und $a \in \Sigma$. Dann ist $\#_a(x)$ die Anzahl des Vorkommens von a in x.

Definition I.1.7:
Seien Σ_1 und Σ_2 zwei Alphabete.
Eine Substitution von Σ_1^* nach Σ_2^* ist eine Abbildung $\sigma: \Sigma_1^* \rightarrow P(\Sigma_2^*)$ mit folgenden Eigenschaften:
1. $\sigma(\lambda) = \{\lambda\}$
2. Für alle $a \in \Sigma_1$ ist $\sigma(a)$ eine Sprache über Σ_2
3. Für alle $x,y \in \Sigma_1^*$ gilt: $\sigma(xy) = \sigma(x)\sigma(y)$

Definition: Homomorphismus
Eine Substitution σ mit der Eigenschaft, daß $\sigma(a)$ aus einem einzigen Wort σ_a besteht (d.h. $|\sigma(a)| \leq 1$), heißt Homomorphismus.

Definition I.1.8: s (L), s(L)
Sei σ ein Homomorphismus von Σ_1^* nach Σ_2^* , sei L eine Sprache über Σ_1 ($L \subseteq \Sigma_1^*$).
Definiere: $\sigma(L) := \{\sigma(x) \mid x \in L\}$
Sei s eine Substitution von Σ_1^* nach $P(\Sigma_2^*)$.
Definiere: $s(L) := \cup s(x)$, $x \in L$.

Definition I.1.9: Kanonische Ordnung
Sei $\Sigma = \{s_1, s_2 , ... , s_m\}$, $m \geq 1$ ein Alphhabet und sei $s_1 < s_2 < ... < s_m$ eine Ordnung auf Σ.
Wir definieren die "kanonische Ordnung" in Σ^* wie folgt:

$u < v \quad \Leftrightarrow \quad |u| < |v|$ oder
$|u| = |v|$, $u = xs_iu'$, $v = xs_jv'$ für $x,u',v' \in \Sigma^*$ und $i < j$.

Dabei seien $u,v \in \Sigma^*$.

Definition I.2.1: Entscheidungsproblem:
Ein Entscheidungsproblem für ein gegebenes Alphabet Σ und eine gegebenne Sprache L ($L \subseteq \Sigma^*$) ist es, für jedes $x \in \Sigma^*$ zu entscheiden, ob $x \in L$ oder $x \notin L$.
Ein Algorithmus A "löst" das Entscheidungsproblem (L, Σ),
falls für alle $x \in \Sigma^*$ gilt:

$$A(x) = \begin{cases} 1 \text{ falls } x \in L. \\ 0 \text{ falls } x \notin L. \end{cases}$$

Wir sagen auch, daß A L "erkennt".

Definition I.2.2:
Seien Σ und Γ zwei Alphabete. Wir sagen, daß ein Algorithmus A eine Funktion (Transformation)

$f: \Sigma^* \rightarrow \Gamma^*$

berechnet (realisiert), falls $A(x) = f(x)$ für alle $x \in \Sigma^*$.

Definition I.2.3:
Seien Σ und Γ zwei Alphabete, und sei $R \subseteq \Sigma^* \times \Gamma^*$ eine Relation in Σ^* und Γ^*. Ein Algorithmus A berechnet R, falls für jedes $x \in \Sigma^*$, $(x, A(x)) \in R$.

Definition I.2.4:
Ein Optimierungsproblem ist ein 6-Tupel U = (Σ_I, Σ_O, L, M, cost, goal), wobei
(i) Σ_I ist ein Alphabet, genannt Eingabealphabet
(ii) Σ_O ist ein Alphabet, genannt Ausgabealphabet
(iii) $L \subseteq \Sigma_I^*$ ist die Sprache der zulässigen Eingaben
Ein $x \in L$ ist ein Problemfall von U.
(iv) M ist eine Funktion von L nach Σ_O^*, und für jedes $x \in L$, ist M(x) die Menge der zulässigen Lösungen für x.
(v) cost ist eine Funktion, cost : $\cup M(x) \times L$, $x \in L \rightarrow |R+$, genannt Preisfunktion
(vi) goal $\in$ {Minimum, Maximum}

Ein Algorithmus A löst U, falls für jedes $x \in L$,
a) $A(x) \in M(x)$
b) cost (A(x)) = goal{cost(z,x) | $z \in M(x)$}

Jede zulässige Lösung $y \in M(x)$ heißt optimal, falls cost(y) = goal{cost(z,x) | $z \in M(x)$}

Definition I.2.5:
Sei Σ ein Alphabet und sei $x \in \Sigma^*$. Wir sagen, daß ein Algorithmus A das Wort x generiert, falls A für eine Eingabe λ die Ausgabe x liefert.

Definition I.2.6:
Sei Σ ein Alphabet und sei $L \subseteq \Sigma^*$.
A ist ein Aufzählungsalgorithmus für L, falls A für jede Eingabe $n \in |N - \{0\}$, die Wortfolge $x_1,x_2,...,x_n$ ausgibt, wobei $x_1,x_2,...,x_n$ die kanonisch ersten Wörter in L sind.

Definition I.3.1:
Sei x ein Wort über Σ_{BOOL}. Die Kolmogorov-Komplexität des Wortes x, K(x), ist die binäre Länge des kürzesten Pascal-Programmes, das x generiert.

Lemma I.3.1:
Es existiert eine Konstante d, so daß für jedes $x \in \Sigma_{BOOL}^*$ gilt:
$K(x) \leq |x| + d$

Lemma I.3.2:
Zu jeder Zahl $n \in |N - \{0\}$ existiert ein Wort $x_n \in \Sigma_{BOOL}\hat{}n$, so daß
$K(x_n) \geq |x_n| = n$
(Bedeutung: Zu jeder Eingabelänge existiert ein binäres Wort, das man nicht komprimieren kann)

Definition: Zufällig
Ein Wort $x \in \Sigma_{BOOL}^*$ heißt zufällig, falls : $K(x) \geq |x|$.
D.h. x ist nicht komprimierbar.

KAPITEL II

Definition II.1.1: Endlicher Automat

1. Ein endlicher Automat (EA) ist ein Quintupel $M = (Q, \Sigma, \delta, q_o, F)$,
   wobei: - Q eine endliche Menge von Zuständen ist
   - Σ ein Alphabet - genannt Eingabealphabet - ist
   - $q_o \in Q$ der Anfangszustand ist
   - $F \subseteq Q$ die Menge der Endzustände
   - $\delta: Q \times \Sigma \to Q$ die Übergangsfunktion
2. Eine Konfiguration C von M ist ein Element aus $Q \times \Sigma^*$.
   Ein Schritt von M ist eine Relation $\to_M \subseteq (Q \times \Sigma^*) \times (Q \times \Sigma^*)$ definiert durch:
   $(q,w) \to_M (p,x) :\Leftrightarrow w = ax, a \in \Sigma, x \in \Sigma^*$ und $\delta(q,a) = p$.
   $\to_M^*$ ist die reflexive, transitive Hülle von $\to_M$.
   Jede Konfiguration aus $\{q_o\} \times \Sigma^*$ wird Startkonfiguration genannt.
   Jede Konfiguration aus $Q \times \{\lambda\}$ ist eine Endkonfiguration.
3. Eine Berechnung von M ist eine endliche Folge $C_o, C_1, ..., C_n$ von Konfigurationen, so daß
   $C_i \to_M C_{i+1}$ für alle $0 \le i \le n - 1$ gilt,
   wobei C_o eine Start- und $C_n \in Q \times \{\lambda\}$ eine Endkonfiguration ist.
4. Die Sprache L(M) ist definiert als:
   $L(M) := \{ w \in \Sigma^* \mid (q_o, w) \to_M^* (p,\lambda) \text{ mit } p \in F\}$
   und heißt die von M akzeptierte Sprache.
5. Definiere $\delta^\wedge : Q \times \Sigma^* \to Q$ durch:
   $\delta^\wedge(q,\lambda) := q$
   $\delta^\wedge(q,wa) := \delta(\delta^\wedge(q,w)a)$ mit $a \in \Sigma, w \in \Sigma^*$
6. Definiere: L(EA) := {L(M) | M ist ein endlicher Automat}

Satz II.2.1:
Sei Σ ein Alphabet und seien $M_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$ und $M_2 = (Q_2, \Sigma, \delta_1, q_{02}, F_2)$ zwei EAen
Es existiert ein EA M, so daß $L(M) = L(M_1) \otimes L(M_2)$ wobei $\otimes \in \{\cup, \cap, \neg\}$.

Lemma II.3.1:
$L = \{a^\wedge n b^\wedge n \mid n \in |N\} \notin L(EA)$

Satz II.3.2:
Sei L = L(M) für einen endlichen Automaten $M = (Q, \Sigma, \delta, q_0, F)$.
Sei $L_x := \{ xy \in \Sigma^* \mid xy \in L\}$ für jedes $x \in \Sigma^*$.
Sei xy das n-te Wort in L_x bezüglcih kanonischer Ordnung.
Dann ist: $K(y) \le \lceil \log_{2(}n) \rceil + const_M$

Definition II.4.1: Nichtdeterministischer endlicher Automat:

1. Ein nichtdeterministischer endlicher Automat (NEA) ist ein Quintupel $M = (Q, \Sigma, \delta, q_0, F)$
   wobei : - Q, Σ, q_0, F die gleiche Bedeutung haben wie bei einem endlichen Automaten
   - $\delta : Q \times \Sigma \to P(Q)$ ist
2. Eine Konfiguration C von M ist ein Element aus $Q \times \Sigma^*$.
   Ein Schritt von M ist eine Relation $\to_M \subseteq (Q \times \Sigma^*) \times (Q \times \Sigma^*)$ definiert durch:
   $(q,w) \to_M (p,x) :\Leftrightarrow w = ax, a \in \Sigma, x \in \Sigma^*$ und $p \in \delta(q,a)$.
   Eine Berechnung von M auf einem Wort $x \in \Sigma^*$ ist eine Folge von Konfigurationen
   $B = C_o \to_M C_1 \to_M \ldots \to_M C_m$
   wobei: - $C_o = (q_0, x)$ die Startkonfiguration für x ist und
   - $C_m = (p,\lambda)$ für ein $p \in Q$ oder
   $C_m = (q, ax)$ für ein $a \in \Sigma$, wobei $\delta(q,a) = \varnothing$
   $\to_M^*$ die transitive Hülle von $\to_M$.
   B ist eine akzeptierende Berechnung, falls $C_m = (p,\lambda)$ mit $p \in F$ ist.
3. Die Sprache: $L(M) := \{ w \in \Sigma^* \mid$ es existiert eine akzeptierende Berechnung mit $(q_0, w) \to_M^* (p,\lambda), p \in F \}$ ist die von M akzeptierte Sprache.
4. $\delta^\wedge : Q \times \Sigma^* \to P(Q)$ ist definiert durch:
   $\delta^\wedge(q,\lambda) := \{q\}$
   $\delta^\wedge(q,wa) := \{ p \mid$ es existiert ein $r \in \delta^\wedge(q,w)$, so daß: $p \in \delta(r,a)\}$ mit $a \in \Sigma, w \in \Sigma^*$

Satz II.4.1:
Zu jedem nichtdeterminnistischen Automaten M existiert ein endlicher Automat M' so daß $L(M) = L(M')$.

KAPITEL III

Definition II.1.1:

(i) Eine Turingmaschine (TM) ist ein 7-Tupel $M = (Q, \Sigma, \Gamma, \delta, q_o, q_{accept}, q_{reject})$, wobei
- (1) Q ist eine endliche Menge, genannt die Menge der Zustände von M
- (2) Σ ist das Eingabealphabet, das Blank-Symbol $\_ \notin \Sigma$.
- (3) Γ ist das Ausgabealphabet, $\Sigma \subseteq \Gamma$, $\&, \_ \in \Gamma$.
- (4) $\delta: (Q - \{ q_{accept}, q_{reject} \}) \times \Gamma \to Q \times \Gamma \times \{L,R,O\}$ ist eine Abbildung mit der Eigenschaft $\delta(q,\&) \in Q \times \{\&\} \times \{R,O\}$ $\forall q \in Q$
  δ nennt man die Übergangsfunktion von M.
- (5) $q_o \in Q$ ist der Anfangszustand.
- (6) $q_{accept} \in Q$ ist der akzeptierende Zustand.
- (7) $q_{reject} \in Q - \{ q_{accept} \}$ ist der verwerfende Zustand.

(ii) Eine Konfiguration C von M ist ein Element aus
$Konf(M) = \{\&\}\Gamma^* \times Q \times \Gamma^* \cup Q \times \Gamma^*$ w_1qw_2,
wobei $w_1 \in \{\&\}\Gamma^*$, $q \in Q$, $w_2 \in \Gamma^*$ repräsentieren die folgende Situation:
- M ist in Zustand q
- w_1w_2 ist der Inhalt des Bandes
- Der Lese/Schreibkopf liegt auf dem am weitesten links stehenden Symbol von w_2

Eine Startkonfiguration für ein Eingabewort x ist $q_o\&x$.

(iii) Ein Schritt von M ist eiine Relation $\to_M$ auf Konfigurationen definiert wie folgt:
- (1) $x_1x_2...x_{i-1}qx_i...x_n \to_M x_1x_2...x_{i-1}py x_{i+1}...x_n$ falls $\delta(q, x_i) = (p,y,O)$
- (2) $x_1x_2...x_{i-1}qx_i...x_n \to_M x_1x_2...x_{i-2}px_{i-1}yx_{i+1}...x_n$ falls $\delta(q, x_i) = (p,y,L)$
- (3) $x_1x_2...x_{i-1}qx_i...x_n \to_M x_1x_2...x_{i-1}ypx_{i+2}...x_n$ falls $\delta(q, x_i) = (p,y,R)$

(iv) Eine Berechnung von M ist eine (potentiell unendliche) Folge von Konfigurationen $C_0,C_1,..., C_i,C_{i+1},...$ so daß $C_i \to_M C_{i+1}$ für alle $i \in \{1,2,....\}$
Wenn $C_0 \to_M C_1 \to_M ... \to_M C_i$ für ein $i \in \{1,2,....\}$, dann $C_0 \to_{M^*} C_i$
Eine Berechnung von M auf einer Eingaabe x ist eine Berechnung, die mit $C_0 = (q_o\&x)$ anfängt und ist entweder unendlich oder endet in einer Konfiguration w_1qw_2, wobei $q \in \{q_{reject}, q_{accept}\}$. Eine Berechnung heißt akzeptierend, falls sie in einer akzeptierenden $w_1q_{accept}w_2$ Konfiguration endet.

(v) Die von der Turingmaschine M akzeptierte Sprache ist L(M)
$L(M) = \{ w \in \Sigma^* \mid q_o\$w \to_{M^*} yq_{accept}z, y, z \in \Sigma^* \}$
Wir sagen, daß M eine Funktion $F: \Sigma^* \to \Gamma^*$ berechnet, falls für alle $x \in \Sigma^*$:
$q_o\$x \to_{M^*} q_{accept}\$F(x)$

(vi) Eine Sprache L heißt rekursiv aufzählbar, falls eine TM M existiert, so daß L = L(M)
$L_{RE} = \{ L(M) \mid M$ ist eine TM $\}$ ist dei Menge aller rekursiv aufzählbaren Sprachen.
Eine Sprache $L \subseteq \Sigma^*$ heißt rekursiv (oder endscheidbar), falls L = L(M) für eine TM M, die für alle $x \in \Sigma^*$

$q_o\$x \to_{M^*} yq_{accept}z$, $y,z \in \Gamma^*$ falls $x \in L$ und
$q_o\$x \to_{M^*} uq_{reject}v$, $u,v \in \Gamma^*$ falls $x \notin L$

Wir sagen auch, daß M auf jeder Eingabe hält in einem der Zustände q_{accept}, q_{reject}. Eine TM, die immer hält, ist ein formales Modell des Begriffs Algorithmus.
$L_R = \{ L(M) \mid M$ ist eine TM, die immer hält $\}$ ist die Menge der rekursiven (algorithmisch erkennbaren) Sprachen.

Lemma III.2.1:
Zu jeder 2TM(k) A existiert eine TM B, so daß L(A) = L(B)

Definition II.3.1: Nichtdeterministische Turingmaschine

(1) Eine nichtdeterministische Turingmaschine (NTM) ist ein 7-Tupel
$M = (Q, \Sigma, \Gamma, \delta, q_o, q_{accept}, q_{reject})$, wobei $Q, \Sigma, \Gamma, q_o, q_{accept}, q_{reject}$ die gleiche Bedeutung haben wie bei einer TM,
$\delta: Q \times \Gamma \rightarrow P(Q \times \Gamma \times \{L,R,O\})$ die Übergangsfunktion von M ist.

(2) Eine Konfiguration wird wie bei einer TM definiert.

(3) Ein Schritt von M ist eine Relation $\rightarrow_M$ auf Konfigurationen definiert wie folgt:
$x_1x_2...x_{i-1}qx_i...x_n \rightarrow_M x_1x_2...x_{i-1}py x_{i+1}...x_n$ falls $(p,y,O) \in \delta(q, x_i)$
$x_1x_2...x_{i-1}qx_i...x_n \rightarrow_M x_1x_2...x_{i-2}px_{i-1}yx_{i+1}...x_n$ falls $(p,y,L) \in \delta(q, x_i)$
$x_1x_2...x_{i-1}qx_i...x_n \rightarrow_M x_1x_2...x_{i-}1ypx_{i+2}...x_n$ falls $(p,y,R) \in \delta(q, x_i)$

(4) Eine Berechnung von M ist eine (potentiell unendliche) Folge von Konfigurationen $C_0,C_1,..., C_i,C_{i+1},...$ so daß $C_i \rightarrow_M C_{i+1}$ für alle $i \in \{1,2,....\}$
Eine Berechnung auf einer Eingabe x ist eine Berechnung, die mit $C_0 = (q_o\&x\$)$ anfängt und ist entweder unendlich oder in einer Konfiguration w_1qw_2, wobei $q \in \{q_{reject}, q_{accept}\}$. Eine Berechnung heißt akzeptierend, falls sie in einer $w_1q_{accept}w_2$ Konfiguration endet.

(5) Die von der NTM akzeptierte Sprache ist definiert durch:
$L(M) = \{ w \in \Sigma^* \mid q_o\&w\$ \rightarrow_{M^*} yq_{accept}z, y, z \in \Sigma^* \}$

Definition: Berechnungsbaum

Sei M eine NTM und sie x ein Wort über dem Eingabealphabet von M.
Ein Berechnungsbaum TM,x von M auf x ist ein (potentiell unendlicher) gerichteter Baum, der wie folgt definiert ist:

- Jeder Knoten ist mit einer Konfiguration bezeichnet
- Die Wurzel hat den Ingrad 0 und ist mit $q_o\&x\$$ beschriftet
- Jeder Knoten des Baums (sei C die Beschriftung) hat genau so viele Söhne, wie die Anzahl der Nachfolgekonfigurationen von C) und diese Söhne sind mit Nachfolgekonfigurationen von C beschriftet

Satz III.2.1:

Sei M eine NTM. Dann existiert eine TM A, so daß L(M) = L(A)

KAPITEL IV

Definition IV.1.1: Aufzählbar
Eine Menge A heißt aufzählbar, falls: A endlich ist oder |A| = | |N|

Satz IV.1.1:
|R ist nicht aufzählbar.

Lemma IV.1.1:
| 2^(Σ_{BOOL}*) | = | [0,1) |

Satz IV.2.1:
Es gibt aufzählbar viele TM (Algorithmen).

Satz IV.2.2:
Es existiert eine Sprache über Σ_{BOOL}, die nicht rekursiv aufzählbar ist.

Satz IV.2.3:
L_d ist nicht rekursiv aufzählbar.
L_d = { w ∈ {0,1}* | w = w_i für ein i ∈ |N\{0} und M_i akzeptiert w_i nicht }

Definition IV.3.1:
Die universelle Sprache ist L_U = { <M>,w | w ∈ {0,1}* und M akzeptiert w }

Satz IV.3.1:
Es gibt eine TM UTM (universelle TM), so daß L(UTM) = L_U.

Lemma IV.3.1:
Sei Σ ein Alphabet und sie L eine Sprache über Σ.
Falls L ∈ L_R, dann L^c = Σ* - L ∈ L_R.

Satz IV.3.2:
L_U ∈ L_R.

Satz IV.3.3:
L_R ⊂ L_{RE}.

Satz IV.3.4:
L_H ∈ L_{RE} - L_R.

Church'sche These:
TM ist die Formalisierung des Begriffs Algorithmus.

- L_R ist die Menge der algorithmisch entscheidbaren Sprachen
- Die Funktionen, die TM-berechenbar sind, sind genau die Funktionen, die algorithmsich berechenbar sind.

Definition IV.4.1:
Seine $L_1 \subseteq \Sigma_1^*$ und $L_2 \subseteq \Sigma_2^*$ zwei Sprachen.
Wir sagen, daß L_1 ist auf L_2 rekursiv reduzierbar, $L_1 \leq_R L_2$,
falls $L_1 \in L_R \Rightarrow L_2 \in L_R$.

Lemma IV.4.1:
Definiere $L_{empty} := \{<M> \mid L(M) = \emptyset\}$
Dann gilt: $(L_{empty})^c \in L_{RE}$.

Lemma IV.4.2:
$L_{empty} \in L_R$.

Korollar IV3.1:
Die folgenden Probleme sind nicht entscheidbar
1. $L_{EQ} = \{ <M_1>, <M_2> \mid L(M_1) = L(M_2) \}$
2. $L_{\subseteq} = \{ <M_1>, <M_2> \mid L(M_1) \subseteq L(M_2) \}$

Lemma IV.3.2:
Definiere $L_{H,\lambda} = \{ <M> \mid M$ hält nicht auf $\lambda \}$
Es gilt: $L_{H,\lambda}$ ist nicht rekursiv.

Definition IV.4.2: Semantisch nichttriviales Entscheidungsproblem:
Eine Sprache $L = \{ <M> \mid M$ ist eine TM $\}$ heißt semantisch nichttriviales Entscheidungsproblem über Turingmaschinen falls:
(1) $L(M_1) = L(M_2)$ für zwei Turingmaschinen M_1 und M_2 impliziert:
$<M_1> \in L \Leftrightarrow <M_2> \in L$
(2) Es existieren zwei Turingmaschinen M_3 und M_4 so daß
$<M_3> \in L \Leftrightarrow <M_4> \notin L$.

Satz IV.4.1: Satz von Rici:
Jedes semantisch nichttriviale Entscheidungsproblem über Turingmaschinen ist unentscheidbar.

Definition IV.4.3: Post-Korrespondenzproblem (PKP)
Eingabe: $A = w_1,w_2,...,w_k$, $B = x_1,x_2,...,x_k$, $k \in |N\backslash\{0\}$,
$w_i,x_i \in \Sigma^*$
Ausgabe: "accept" falls $\exists\, i_1,i_2,...,i_m \in \{1,..,k\}$, $m \geq 1$, so daß $w_{i1}w_{i2}...w_{im} = x_{i1}x_{i2}...x_{im}$
"reject" sonst.
Die Folge $i_1,i_2,...,i_m$ nennt man eine Lösung des Problemfalls PKP(A,B)

Definition IV.4.2:Das modifizierte PKP (MPKP)
Eingabe: $A = w_1,w_2,...,w_k$, $B = x_1,x_2,...,x_k$, $k \in |N\backslash\{0\}$,
$w_i,x_i \in \Sigma^*$
Ausgabe: "accept" falls $\exists\, i_1,i_2,...,i_m \in \{1,..,k\}$, $m \geq 1$, so daß $\mathbf{w_1}w_{i1}w_{i2}...w_{im} = \mathbf{x_1}x_{i1}x_{i2}...x_{im}$
"reject" sonst.

Lemma IV.4.4:
Falls PKP entscheidbar ist, dann auch MPKP. $MPKP \leq_R PKP$.

Satz IV.4.2:
Das MPKP ist nicht entscheidbar.

KAPITEL V

Definition V.5.1: Zeitkomplexität:
Sei M eine MTM (übliche TM) und sei $x \in \Sigma^*$.
Sei $D = C_1C_2...C_k$ eine Berechnung von M auf x.
Dann ist die Komplexität der Berechnung von M auf x definiert durch:

$T_M(x) := k - 1$

Die Zeitkomplexität (im schlechtesten Fall) der TM M, die immer hält, ist die Funktion
$T_M: |N \rightarrow |N$, definiert durch: $T_M(n) := \max\{ T_M(x) \mid x \in \Sigma\text{^}n\}$.

Definition V.5.2: Speicherplatzkomplexität:
Sei M eine MTM (2-TM(k), $k \in |N\backslash\{0\}$, die immer hält.
Sei $C = (q,(x,i),(a_1,i_1),...,(a_k,i_k)$, wobei $|a_j| \geq i_j$ eine Konfiguration.
Die Speicherplazkomplexität von C ist definiert durch:
$S(C) := \max \{|a_i| , i = 1,..,k\}$
Sei $C_1C_2...C_k$ die Berechnung von M auf x.
Die Speicherplazkomplexität von M auf x ist definiert durch:
$S_M(x) := \max \{ S(C_i) , i = 1,..,k\}$
Die Speicherplazkomplexität ist eine Funktion $S_M : |N \rightarrow |N$ definiert durch:
$S_M(n) := \max \{ S_M(x) , x \in \Sigma\text{^}n \}$

Definition V.1.3:Asymptotik:
Für jede Funktion $f: |N \rightarrow |R+$ definiere:
$O(f(n)) := \{r: |N \rightarrow |R+ \mid \exists\, n_o \in |N, \exists\, c \in |N: r(n) \leq c*f(n)\ \forall\, n \geq n_o \}$
Für jedes $r \in O(f(n))$ sagen wir, daß r asymptotisch nicht schneller als f wächst.
Für jede Funktion $g: |N \rightarrow |R+$ definiere:
$\Omega(g(n)) := \{q: |N \rightarrow |R+ \mid \exists\, n_o \in |N, \exists\, d \in |N: q(n) \geq (1/d)*g(n)\ \forall\, n \geq n_o \}$
Für jedes $q \in \Omega(g(n))$ sagen wir, daß q asymptotisch mindestens so schnell wie g wächst.
Definiere:
$\theta(h(n)) := \{z:|N \rightarrow |R+ \mid \exists\, n_o,c,d \in |N: \ (1/d)*h(n) \leq z(n) \leq c*h(n)\ \forall\, n \geq n_o \} =$
$O(h(n)) \cap \Omega(h(n))$
Falls $z \in \theta(h(n))$, so sagen wir daß z und h asymptotisch äquivalent sind.

Definition V.2.1: Reduzierbar:
Seien A und B zwei Maschinenmodelle. Wir sagen, daß A auf B polynomzeitreduzierbar ist ($A \leq_{pol} B$), falls ein Polynom p existiert, so daß für jedes $A \in A$ ein Algorithmus $B \in B$ existiert, so daß:

$L(A) = L(B)$ und $T_B(n) \in O(p(T_A(n)))$

Zwei Maschinenmodelle A und B heißen polynomialzeit- äquivalent ($A \equiv_{pol} B$),falls:

$A \leq_{pol} B$ und $B \leq_{pol} A$

Falls $p(n) = n$, so sagen wir, daß A auf B linearzeit-reduzierbar ist.

Lemma V.2.1:
Für jede TM M existiert eine äquivalente 2TM(1) M', so daß $T_{M'}(n) \leq T_M(n) + 2n + 2$

Lemma V.2.2:
Für jede 2TM(k) A, $k \in |N\backslash\{0\}$, mit daß $T_A(n) \geq n$, existiert eine äquivalente TM M, so daß
$T_M(n) \in O((T_A(n))^2)$.

Theorem V.2.1:
$\forall k \in |N-\{0\}$: TM $\equiv_{pol}$ 2TM(k).

Lemma V.2.3:
Sei k eine positive ganze Zahl. Für jede 2TM(k) A existiert eine äquivalente 2TM(1) M, so daß $S_M(n) \leq S_A(n)$.

Definition V.2.2: Fundamentale deterministische Komplexitätsklassen:
Für alle Funktionen f,g : $|N \rightarrow |R+$
TIME(f) = {L(M) | M ist eine MTM mit $T_A(n) \in O(f(n))$}
SPACE(g) = { L(A) | A ist eine MTM mit $S_A(n) \in O(g(n))$}
TIME-SPACE(f,g) = { L(B) | B ist eine MTM mit $T_B(n) \in O(f(n))$ und $S_B(n) \in O(g(n))$}
DLOG = SPACE($\log_2(n)$)
P = $\cup$ TIME(n^c) , c $\in |N$
EXPTIME = $\cup$ TIME(2^(n^d)) , d $\in |N$
PSPACE = $\cup$ SPACE(n^c) , c $\in |N$

Lemma V.2.4:
Für jede Funktion t: $|N \rightarrow |R+$ TIME(t(n)) $\leq$ SPACE(t(n)).

Korollar V.2.1:
P $\subseteq$ PSPACE

Definition V.2.3:
Seien s, S Funktionen von $|N \rightarrow |R+$. s heißt <u>S-konstruierbar</u>, falls eine 2TM(1) M existiert, so daß
(1) $S_M(n) \leq S(n)$ für $\forall n \in |N$ und
(2) Für die Eingabe 0^n auf dem Eingabeband schreibt M 0^(s(n)) auf das Arbeitsband und hält.
Eine Funktion s: $|N \rightarrow |N$ heißt <u>platzkonstruierbar</u>, falls s <u>s-platzkonstruierbar</u> ist.
Seien t, T Funktionen aus |N nach |N. t heißt <u>T-zeitkonstruierbar</u>, falls eine 2TM(2) M' existiert, so daß:
(1) $T_{M'}(n) \leq T(n)$ für $\forall n \in |N$ und
(2) Für die Eingabe 0^n schreibt M' 0^(t(n)) auf das erste Arbeitsband und hält.
Eine Funktion t: $|N \rightarrow |N$ heißt <u>zeitkonstruierbar</u>, falls t T-zeitkonstruierbar ist, wobei
$T(n) \in O(t(n))$.

Lemma V.2.5:
Sei s: $|N \rightarrow |N$ eine platzkonstruierbare Funktion. Für jede MTM M mit $S_M(x) \leq s(|x|)$ für alle $x \in L(M)$, existiert eine äquivalente MTM A , so daß $S_A(n) \leq s(n)$.

Lemma V.2.6:
Sei t: $|N \rightarrow |N$ eine zeitkonstruierbare Funktion. Für jede MTM M mit $T_M(x) \leq t(|x|)$ für alle $x \in L(M)$, existiert eine äquivalente MTM A , so daß $T_A(n) \in O(t(n))$.

Satz V.2.1:
SPACE(s(n)) $\subseteq \cup$ TIME(c^(s(n))), c $\in |N$
für alle platzkonstruierbaren Funktionen s mit $s(n) \geq \log_2(n)$.

Definition V.3.1:
Sei M eine NTM. Sei $x \in L(M)$. Die Zeikomplexität von M auf x $T_M(x)$, ist die Länge der kürzesten akzeptierenden Berechnung von M auf x. Die Zeitkomplexität von M ist die Funktion $T_M: \mathbb{N} \to \mathbb{N}$ definiert wie folgt:
$T_M(n) = \max \{ T_M(x) \mid x \in L(M) \cap \Sigma^n \}$
Die Speicherplatzkomplexität von M auf x ist
$S_M(x) = \min \{ S(D) \mid D$ ist eine akzeptierende Berechnung von M auf x$\}$
$S_M(n) = \max \{ S_M(x) \mid x \in L(M) \cap \Sigma^n \}$.

Definition V.3.2: Nichtdeterministische Komplexitätsklassen
Für alle $f,g : \mathbb{N} \to \mathbb{R}+$
$NTIME(f) = \{ L(M) \mid M$ ist eine NMTM mit $T_M(n) \in O(f(n))\}$
$SPACE(g) = \{ L(M) \mid M$ ist eine NMTM mit $S_M(n) \in O(g(n))\}$
$NLOG = NSPACE(\log_2(n))$
$NP = \cup NTIME(n^c)$, $c \in \mathbb{N}$
$NPSPACE = \cup NSPACE(n^c)$, $c \in \mathbb{N}$

Lemma V.3.1:
Für alle platzkonstruierbaren Funktionen s,t:
$NTIME(t) \subseteq NSPACE(t)$
$NSPACE(s) \subseteq \cup NTIME(c^{s(n)})$, $c \in \mathbb{N}$

Satz V.3.1:
Für jede platzkonstruierbare Funktion $t: \mathbb{N} \to \mathbb{R}+$
(i) $TIME(t(n)) \subseteq NTIME(t(n))$
(ii) $SPACE(t(n)) \subseteq NSAPCE(t(n))$
(iii) $NTIME(t(n)) \subseteq SPACE(t(n)) \subseteq \cup TIME(c^{t(n)})$, $c \in \mathbb{N}$

Definition V.4.1: p-Verifizierer:
Sei $L \subseteq \Sigma^*$ eine Sprache, sei $p: \mathbb{N} \to \mathbb{N}$ eine Funktion.
Wir sagen, daß ein Algorithmus (eine TM) A ein p-Verifizierer ist ($V(A) = p$), falls A mit folgenden Eigenschaften auf Eingaben aus $\Sigma^* \times \Sigma_{BOOL}^*$ arbeitet:
(1) $T_A(w,x) \leq p(|w|)$ für jede Eingabe $(w,x) \in \Sigma^* \times \Sigma_{BOOL}^*$
(2) Für jedes $w \in L$ existiert ein $x \in \Sigma_{BOOL}^*$, $|x| \leq p(|w|)$, so daß $(w,x) \in L(A)$
(3) Für jedes $y \notin L$ gilt $(y,z) \notin L(A)$ für alle $z \in \Sigma_{BOOL}^*$
Falls $p(n) \in O(n^k)$ für ein $k \in \mathbb{N}$, so ist A ein polynomialzeit-Verifizierer.
Definiere: $VP := \{ V(A) \mid A$ ist ein polynomialzeit-Verifizierer$\}$

Satz V.4.1:
NP = VP

Definition V.5.1: Polynomielle Reduzierbarkeit von Sprachen
Seien $L_1 \subseteq \Sigma_1^*$ und $L_2 \subseteq \Sigma_2^*$ zwei Sprachen.
L_1 ist polynomiell auf L_2 reduzierbar ($L_1 \leq_p L_2$), falls ein polynomieller Algorithmus (TM) A existiert, der für jedes Wort $x \in \Sigma_1^*$ ein Wort $A(x) \in \Sigma_2^*$ berechnet, so daß:
$$x \in L_1 \quad \Leftrightarrow \quad A(x) \in L_2$$
A wird eine polynomielle Reduktion von L_1 auf L_2 genannt.

Definition V.5.2: NP schwer / NP vollständig
Eine Sprache L ist NP schwer, falls für alle $U \in NP$: $U \leq_p L$
Eine Sprache heißt NP vollständig falls:
i) $L \in NP$
ii) L ist NP schwer

Lemma V.5.3:
Falls $L \in P$ und L ist NP schwer, dann gilt. P = NP.

Satz V.5.4: Satz von Cook
SAT ist NP vollständig.

Lemma V.5.5:
Seien L_1 und L_2 zwei Sprachen. Falls $L_1 \leq_p L_2$ und L_1 ist NP schwer, dann ist L_2 NP schwer.

Lemma V.5.6:
$SAT \leq_p CLIQUE$

Lemma V.5.7:
$CLIQUE \leq_p VC$

Lemma V.5.8:
$SAT \leq_p 3SAT$

Definition V.5.9:
NPO ist die Klasse der Optimierungsprobleme wobei $U = (\Sigma_I, \Sigma_O, L, M, cost, goal) \in NPO$ falls folgenden Bedingungen erfüllt sind:
(i) $L \in P$ (es kann effizient verifiziert werden ob ein Eingabe zulässig ist)
(ii) $\exists$ ein Polynom P_U, so daß
  (ii.1) für jedes $x \in L$ und jedes $y \in M(x)$, $|y| \leq P_U(|x|)$
  (ii.2) ein polynomialzeit Algorithmus A existiert, der für jedes $y \in \Sigma_O^*$ und jedes $x \in L$ mit $|y| \leq P_U(|x|)$ entscheidet, ob $y \in M(x)$ oder nicht.
(iii) die Funktion cost in polynomieller Zeit berechenbar ist.

Definition V.5.10:
PO ist die Klasse von Optimierungsproblemen $U = (\Sigma_I, \Sigma_O, L, M, cost, goal)$, so daß
(i) $U \in NPO$
(ii) ein polynomialzeit Algorithmus A existiert, so daß für jedes $x \in L$ A(x) eine optimale Lösung für x ist.

Definition V.5.11:
Sei $U = (\Sigma_I, \Sigma_O, L, M, cost, goal)$ ein Optimierungsproblem aus NPO.
Eine Treshold-Sprache für U ist
$Lang_U = \{ (x,a) \in L \times \Sigma_{BOOL}^* \mid Opt_U(x) \leq Nummer(a)\}$ falls goal = Minimum und
$Lang_U = \{ (x,a) \in L \times \Sigma_{BOOL}^* \mid Opt_U(x) \geq Nummer(a)\}$ falls goal = Maximum.
Wir sagen, daß U NP schwer ist, falls $Lang_U$ NP schwer ist.

Lemma V.5.12:
Falls ein Optimierungsproblem $U \in$ PO, dann $Lang_U \in$ P.

Satz V.5.13:
Sei $U \in$ NPO. Falls $Lang_U$ NP schwer ist und P $\neq$ NP dann $U \notin$ PO.
Lemma V.5.14:
MAX-SAT ist NP schwer.

Lemma V.5.15:
MAX-CL ist NP schwer.