# 1 Organisatorisch

Wöchentlich Übung, bewertet Abgeben
Ausgabe Freitags, Abgabe bis Montag 10:00 (Woche danach) Kästen 4. Stock
50% Übungen für Zulassung
Gruppenarbeit bis zu 3 Personen

# 2 Berechenbarkeitstheorie

Frage: Welche Probleme kann man mit einem Computer lösen?

## 2.1 Begriff der Berechenbarkeit

Intuitiver Begriff der Berechenbarkeit:
Es existiert ein Algorithmus/Maschine, mit der man es ausrechnen kann.
Formale Definition:
Einschränkung: Funktionen auf den natürlichen Zahlen
Beispiel: Fig.1 $f : \mathbb{N}^k \to \mathbb{N}$
berechenbar, falls es einen Algorithmus gibt (d.h. ein Programm), der $f(n_1, ..., n_k)$ berechnet, d.h. nach endlich vielen Schritten stoppt und das korrekte Ergebnis ausgibt.
Anmerkung: Eine partielle Funktion $f : \mathbb{N}^k \to \mathbb{N}$ darf das Programm für Werte $(n_1, ..., n_k) \notin Def_f$ auch in einer Endlosschleife enden.
Beispiele: Fig.2
Behauptung: Es gibt reele Zahlen r, sodass $f_r(n)$ nicht berechenbar ist.
Beweis: Es gibt überabzählbar viele reele Zahlen, aber nur abzählbar viele Algorithmen da Programmtext endlich.

## 2.2 Die Churchsche These

Es gibt verschiedene konkrete Vorschläge für die Berechenbarkeitsmodelle// (oben einfacher, alle äquivalent)
- Turing Maschine (TM)
- While-Programme
- Rekursive Funktionen
- Random Access Maschine
- Java Programm

Churchsche These: Jedes dieser Modelle definiert den intuitiven Berechenbarkeitsbegriff

## 2.3 Die Turing-Berechenbarkeit

Allan Turing hat eine Maschine vorgeschlagen
Fig.3 In Abhängigkeit des aktuellen Zustands und dem jeweiligen Zeichen führt die TM eine Aktion aus.
Aktion: Schreibe ein Symbol, bewege den Kopf eine Position nach links/rechst, oder bleibe stehen.
Anfangszustand: Das Band ist bis auf die Eingabe leer
Formale Definition:
Eine Turing Maschine ist ein Siebentupel Fig.4 $M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, E)$
Q: endliche Menge von Zuständen
$\Sigma$: endliche Menge von Eingabesymbolen (Eingabealphabet)
$\Gamma$ : endliche Menge von Bandsymbolen
$\delta$: $Qx\Gamma \to Qx\Gamma x\{L, R, N\}$
nicht deterministisch: $Qx\Gamma \to 2^{Qx\Gamma x\{L,R,N\}}$
$q_0$: Anfangszustand
$\Box$: Leeres Bandsymbol
$\delta(q, a) = (q', b, m)$ bedeutet: Befindet sich M im Zustand q und liest das Symbol a ein, dann wechselt M in den

Zustand q', schreibt das Symbol b und bewegt den Kopf wie durch m beschrieben.
Definition: Die Konfiguration einer TM M ist ein Wort $k \in \Gamma^* Q \Gamma^*$ (beschreibt Momentaufnahme)
$k = \alpha q \beta$ bedeutet: Auf dem Band steht $\alpha$ gefolgt von $\beta$, der Kopf ist am ersten Zeichen von $\beta$, M ist im Zustand q.
Startkonfiguration bei Eingabe von $w \in \Sigma^*$: $q_0 w$
$\delta$ Übergangsfunktion beschreibt einen Schritt $\delta : Q \times \Gamma \rightarrow Q \times L, R, N$
Übergang zwischen Konfigurationen beschrieben durch Relation $\vdash$
Formal:

$$\vdash \begin{cases} a_1, ..., 1_m, q', c, b_2, ..., b_n, \delta(q, b_1) = (q', c, N) \\ a_1, ..., 1_m, c, q', b_2, ..., b_n, \delta(q, b_1) = (q', c, R) \\ a_1, ..., 1_{m-1}, q', a_m, c, b_2, ..., b_n, \delta(q, b_1) = (q', c, R) \end{cases} \tag{1}$$

Nichtdeterministisch: .... falls $(q', c, N) \in \delta(q, b_1)$
Transitiver Abschluss $\vdash^*$: $\alpha q \beta \vdash^* \alpha' q' \beta'$: es existiert eine Folge von Übergängen von 1 nach 2.
Definition: Eine Funktion $f : \mathbb{N}^k \rightarrow \mathbb{N}$ heißt Turing-berechenbar, falls es eine deterministische Touringmaschine M gibt, sodass für alle $n_1, ..., n_k \in \mathbb{N}$ gilt $f(n_1, ..., n_k) = m \Leftrightarrow q_0 bin(n_1) \# bin(n_2), ..., \# bin(n_k) \vdash^* q_e bin(m)$
Wobei bin(n) die Binärdarstellung und # ein Trennzeichen ist.
Mehrband Touringmaschine:
$f : Q \times \Gamma^* \times \{L, R, N\}^*$
Leichere Programmierung für mehrere Argumente (z.B. Addition etc.).
Satz: Zu jeder Mehrband-TM M' gibt es eine Einband-TM M die dieselbe Funktion beschreibt wie M'.
Beweisidee: Sei k=#Bänder, $\Gamma$ Arbeitsalphabet von M. Fig. 5. Jede Spalte sei zusammengefasst in ein neues Bandsymbol.
Simulation von M durch M' $\gamma' \rightarrow \gamma$, k is konstante Anzahl von Bändern. M' simuliert einen Schritt vom M:
1. Positioniere den Kopf links von allen Köpfen von M, setze Markierungen bei den Köpfen.
2. Laufe nach rechts, bis alle Markierungen durchlaufen sind. Jetzt weiß M' worauf $\delta_M$ anzuwenden ist.
3. M' schreibt die entsprechenden Symbole, bewegt den Kopf durch die Markierungen und geht in einen neuen Zustand.
Technik: Hintereinanderschalten von TM ($Start \rightarrow M_1 \rightarrow M_2 \rightarrow Stop$

## 2.4 Loop-,While- und Goto-Berechenbarkeit

Kombination von TM und abstrakter Ablauf:
Beispiele:

1. $Start \rightarrow Band(Band + 1) \rightarrow Band(2 * Band) \rightarrow stop$
2. Fig. 6
3. Mehrband TM: while Band≠0 do M od

### 2.4.1 Loop-Programme

Variablen: $x_0, x_1$
Konstanten: 0,1,2,3...
Operatoren: +,-,:=
Schlüsselwörter: LOOP, DO, END,;
Induktive Definition von Loop-Programmen:

i $x_i := c, x_i := x_j + c, x_i := x_j - c$ sind Loop-Programme

ii Falls $P_1, P_2$ Loop-Programme, dann ist auch $P_1; P_2$ (hintereinander augeführt)

iii Falls P Loop-Programm und $x_i$ eine Variable, dann ist auch: LOOP $x_i$ DO P END; ein Loop-Programm.

Semantik: Klar, bis auf $x_i := x_j - c \rightarrow$ Wert von $x_i = max(0, x_j - c) \rightarrow \mathbb{N}$
Startsituation: Variable $x_i$ enthält i-tes Argument (für i=1...k), $_i = 0$, falls i¿k oder i=0
Resultat: steht am Ende in $x_0$

Bemerkung: alle Funktionen sind total (keine Endlos-Schleife) $\rightarrow$ Wir erhalten immer ein Ergebnis.
Definition: Eine Funktion $f : \mathbb{N}^k \rightarrow \mathbb{N}$ ist loop-berechenbar, falls es ein Loop-Programm P gibt, das gestartet, mit $n_2, ..., n_k$ in den Vriablen $x_1, ..., x_k$ mit $x_0 = f(n_2, ..., n_k)$ stoppt.
Weitere Operationen (simulation):
if x=0 then A fi $\rightarrow$ LOOP x DO y:=0 END; LOOP y DO A END;
Addition: $x_0 := x_1 + x_2 \rightarrow x_0 = x_1$; LOOP $x_2$ DO $x_0 := x_0 + 1$ END;

### 2.4.2 While-Programme

i Jedes Loop-Programm ist ein While-Programm

ii Ist P ein While-Programm und $x_i$ eine Variable, dann ist: WHILE $x_i \neq 0$ DO P END; ein While-Programm

Definition: While-Berechenbarkeit analog zu Loop-Berechenbarkeit
Achtung: Nicht alle Funktionen total!
Lemma1: Jede While-Berechenbare Funktion ist auch Turing-Berechenbar.
Beweis: Jedes While-Programm kann durch eine Mehrband-TM simuliert werden: Band i enthält Wert von $x_i$ (binär):
while Band $i \neq 0$ etc.

### 2.4.3 Goto-Programme

Grunsyntax: Variable, :=, +, -, ... Wie loop nur ohne loop
Definition: Folge von markierten Anweisungen $M_1 : A_1; M_2 : A_2; ...; M_l : A_l$ mit $M_i$ Label/Markierungen
Anweisung: $A_i \qquad x_i := x_j \pm c$
Es gibt unbedingten Sprung: GOTO $M_i$, sowie bedingter Sprung: IF $x_i = c$ THEN GOTO $M_i$
Außerdem gibt es eine HALT Anweisung.
Definition: Goto-berechenbar analog zu While/Loop. Auch hier sind nicht totale Funktionen möglich.
Lemma2: Jede While-berechenbare Funktion ist Goto-berechenbar.
Beweis: Jedes While-Programm kann durch ein Goto-Programm simuliert werden:
While $x_i \neq 0$ DO P END; kann simuliert werden durch:
$M_i$ : IF $x_i = 0$ THEN Goto $M_j$
P
Goto $M_i$
$M_j$ HALT
Lemma3 (Gegenrichtung): Jede Goto-berechenbare Funktion ist auch While-berechenbar.
Beweis: Behaupung: Jedes Goto-Programm kann durch ein While-Programm mit genau einer While-Schleife simuliert werden. (Kleenesche Normalform-Satz. Jede berechenbare Funktion kann durch ein Programm mit genau einer While-Schleife berechnet werden.)
Satz: Jedes Goto-Programm kann durch While-Programm simuliert werden, sogar mit nur einer einzigen While-Schleife.
Beweis: Gegeben ein Goto-Programm:
$M_1 : A_1; M_2 : A_2; ....M_k : A_k$
Wird simuliert durch folgendes While-Programm:
$count \leftarrow 1$,//aktuelle Position im Goto-Programm($M_{count}$)
WHILE $count \neq 0$ DO
    IF $count = 1$ THEN $A'_1$ END
    IF $count = 2$ THEN $A'_2$ END
    .
    .
    IF $count = k$ THEN $A'_k$ END
END

$$A'_i = \begin{cases} x_j := x_i \pm c \text{ falls } A_i : x_j := x_i \pm c \\ count := j \text{ falls } A_i : Goto - M_j \\ \text{IF } x_j = c \text{ THEN count:=j ELSE count=count+1 falls IF } x_j = c \text{ THEN Goto } M_j \\ \text{count:=0 falls } A_i := HALT \end{cases}$$

Folgerung (Kleen'sche Normalformsatz): Jede While-Berechenbare Funktion kann durch ein While-Programm mit nur einer Schleife berechnet werden (While - Goto - While)

Bis jetzt haben wir gezeigt: $GOTO \leftrightarrows WHILE \rightarrow TM$

$\uparrow$

LOOP

Wir zeigen nun: $TM \dashrightarrow GOTO$

Satz: Turing-Maschinen können urch GOTO-Programme simuliert werden (TM-Berechenbar → GOTO-Berechenbar)

Beweis: Sei $M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ Eine TM. Das simulierende Goto-Programm hat folgende Struktur:

$M_1 : P_1; M_2 : P_2; M_3 : P_3$ HALT

Dabei bedeuten

$P_1$: (Transformiert alle Anfangswerte in Binärdarstellung) und erzeugt eine Darstellung der Startkonfiguration von M in den Variablen x, y, z ($\mathbb{N}$).

$P_2$: Simuliert die Arbeitswege von M Schritt für Schritt durch Veränderung der Werte x,y,z.

$P_3$: Erzeugt aus der Darstellung x,y,z der Endkonfiguration von M das eigentliche Ergebnis in $x_0$.

Bemerkung: $P_1, P_3$ hängen nicht von M ab.

Kodierung der Konfiguration in die Variablen x,y,z:

Sei $Q = \{q_1, ..., q_k\}$ Zustand $q_i$ →Zahl i.

$\Gamma = \{a_1, ..., a_k\}$ sei $b > m$ (konstant)

Konfiguration von M: $a_{i1}...a_{ip}q_l a_{j1}...a_{jq} \rightarrow i_1...i_p l j_1...j_q$, wobei $i_1....i_p$ als Zahl zur Basis b in x, sowie $j_1...j_q$ als Zahl zur Basis b in y gespeichert wird. l wird in z gespeichert.

$x = (i_1...i_p)_b \in \mathbb{N}$, $y = (j_q...j_1)_b \in \mathbb{N}$.

y invertiert da letzte Stelle durch Modulo leichter zu erreichen.

Goto-Programmstück $M_2 : P_2$:

$M_2 : a :=$ y mod b

IF z=1 AND a=1 THEN Goto $M_{11}$

IF z=1 AND a=2 THEN Goto $M_{12}$

...

IF z=1 AND a=m THEN Goto $M_{1m}$

...

IF z=k AND a=m THEM Goto $M_{km}$

$M_{11} : Q_{11}; GOTO M_2$ ... $M_{ij} : Q_{ij} GOTO M_2$ ... $M_{km} : Q_{km} GOTO M_2$ für alle Paare $Q \times \Gamma$

Programmstück $Q_{ij}$:

simuliert den Übergang $\delta(q_i, a_j) = (q_{i'}, a_{j'}, \{L, R, N\})$

Sei Kopfbewegung L (Rest analog) für Konfiguration wie oben:

$z := i'$ // Wechesl in Zustand $q_i$

y:= y div b; $y := b * y + j'$ //$a_j$ durch $a_{j'}$ ersetzen.

y:= b*y+x mod b; x:=x div b

Falls $q_i \in F$: $M_{ij}$ : GOTO $M_3$

Folgerung: GOTO, WHILE und TM-Berechenbarkeit sind äquivalent.

Die Ackermann-Funktion:

Beispiel für Funktion, die berechenbar (z.B. durch Java-Programm), aber nicht LOOP-Berechenbar ist.

$A : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$

$A(0, y) = y + 1, A(x, 0) = A(x - 1, 1), A(x, y) = A(x - 1, A(x, y - 1))$

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 2 | 3 | 5 | 7 | 9 | 11 | 13 | 15 |
| 3 | 5 | 13 | . | . | . | . | . |

Wir betrachten abgewandelte Ackerman-Funktion:

$a : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$

$a(0, y) = a(x, 0) = 1;\ a(1, y) = 3y + 1;\ a(x, y) = a(x - 1, a(x - 1, a(y - 1, a(x - 1, ...y)))), y - mal$

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 4 | 7 | 10 | 13 |

Berechenbarkeit in Pseudocode:

```
int a(int x, int y){
        if(x==0 v y==0) return 1;
        if(x==1) return 3*y+1;
        int s=y;
        for(int i=0;i<y;i++){
                s=a(x-1,s);
        }
        return s;
```

a ist WHILE-berechenbar:
Schritt 1: Nicht rekursive Version von a
Idee: Stack für Zwischenergebnisse.

```
s.push(x);
s.push(y);
while(s.size()>1)
        y=s.pop();
        x=s.pop();
        if(x==0 v y==0) s.push(1);
        else if(x==1) s.push(3*y+1);
        else
                for(int i=0;i<y;i++)
                        s.push(x-1);
                s.push(y);
return s.pop();
```

Schritt 2: Stack mit WHILE-Programm realisieren:
Idee: Stack in einer Zahl codieren
1. Codierung von Zahlenpaaren $x, y \in \mathbb{N}$
Funktion c $\mathbb{N} \times \mathbb{N} \to \mathbb{N}$ mit $c(x,y) = 2^{x+y} + x$ da $x < 2^{x+y}$
Sei n=c(x,y). Sei k maximal mit $2^k < n$, dann ist k=x+y, wobei $n - 2^k = x$
Definiere $c_1(n) = x, c_2(n) = y$
2. Codiren des Stacks:
$s.push(x) : n_s \leftarrow c(x, n_s)$
$s.pop() : x \leftarrow c_1(n_s)$
$n_s \leftarrow c_2(n_s)$
$s.size \neq 1 : c_2(n_s) \neq 0$
Lemma: a-Funktion ist WHILE-Berechenbar.
a-Funktion ist nicht LOOP-Berechenbar:
Idee: a wächst schneller als jede Loop-berechenbare Funktion.
Sei P ein LOOP-Programm, ordne P die Funktion $f_P : \mathbb{N} \to \mathbb{N}$ zu, mit: Seien $x_0, ..., x_k$ alle in P vorkommenden Variablen, $n_i$ der Startwert voon $x_i, i = 1, .., n$ und $n'_i$ der Endwert von $x_i$
$f_P(n) := max\{\sum_{i=0}^k n'_i | \sum_{i=0}^k n_i < n\}$
Lemma: Für jedes LOOP-Programm P existiert eine Konstante mit: $f_P(n) < a(k, n)$
Induktion: Starte P: $x_i := x_j \pm c$
offensichtlich $f_P(n) \leq n + n + c = 2n + c < 3n + 1 = a(1, n) \leq a(k, n)$ für alle $k \leq 1, n \leq c$
Wähle $k = c$
P hat die Form $P_1, P_2$ (Hintereinanderausführung) (Induktionsschritt)
Induktionsannahme: Behauptung gilt für $P_1$ und $P_2$
$f_P(n) = f_{P_2}(f_{P_1}(n))$ nach Definition von $f_P$
$< f_{P_2}(a(k, n))$ nach IA $< a(k_2, a(k_1, n))$. Sei nun $k_3 = max(k_1, k_2)$: $\leq a(k_3, a(k_3, n))$
$\leq a(k_3, a(k_3, a(k_3, ......)))$.. n-mal $= a(k_3 + 1, n)$

Offensichtlich gilt Behauptung für $k = k_3 + 1$.
P hat die Form: LOOP $x_i$ DO P' END
I.A.: $\exists$ Konstante k' für P', sodass $f_{P'}(n) < a(k', n)$ für alle $n \geq k'$
Es gilt: $f_P(n) \leq f_{P'}(f_{P'}(....f_{P'}(n)..),\ n\ mal$
I.A. einsetzen: $< a(k', a(k', ....a(k', n)..),\ n\ mal = a(k'+1, n)$
Satz: Die Funktion a ist nicht LOOP-berechenbar.
Beweis (indirekt): Annahme: a ist LOOP-Berechenbar.
Die Funktion $g(n) := a(n,n)$ ist Loop-berechenbar. Sei P das entsprechende LOOP-Programm für g.
$g(n) \leq f_P(n)$ am Anfang: $x_0$ enthält n, am Ende: $x_0$ enthält g(n)
Lemma: für P existiert Konstante k mit $f_P(n) < a(k,n)$
Für Eingabe n=k für P: $g(k) \leq f_P(k) < a(k,k) = g(k)$ Widerspruch.

## 2.5 Entscheidbarkeit

Berechenbarkeit betrifft Funktionen.Wir führen einen entsprechenden Begriff ein für (formale) Sprachen.
Definition: Sei $\Sigma$ ein Endliches Alphabet, dann heißt eine Teilmenge $L \subseteq \Sigma*$ Sprache über $\Sigma$
Definition: Eine Sprache $A \subseteq \Sigma*$ heißt entscheidbar, falls die charakteristische Funktion $\varphi_A : \Sigma* \to \{0,1\}$ mit $\varphi_A(w) = (1, falls\ w \in A;\ 0\ sonst)$ berechenbar ist.
Beispiel Kodierung von Problemen durch Sprachen: $\Sigma = \{0,1,\#\};\ A = \{bin(x)\#bin(x^2)|x \in \mathbb{N}\}$
$\varphi_A(w) : 1) w \in \{0,1\}^*\#\{0,1\}^*(Syntax)$; 2) Test ob $w_1 = \sqrt{w_2}|w_2 = w_1^2$
$A \subseteq \Sigma*$ heißt semi-entscheidbar, falls die partielle charakteristische Funktion $\varphi'_A : \Sigma* \to \{1, undef\}$ mit $\varphi'_A(w) = (1,\ falls\ w \in A;\ undef\ sonst\ (terminiertnicht))$
Satz: A ist entscheidbar $\iff$ A und $\overline{A}(= \Sigma^* \backslash A)$ sind semi-entscheidbar
Beweis: " $\Rightarrow$ A entscheidbar $\Rightarrow$ A semi-entscheidbar.
$\Rightarrow \overline{A}$ entscheidbar. $(\varphi_{\overline{A}}(w) = 1 - \varphi_{\overline{A}}(w)$
" $\Leftarrow$ A und $\overline{A}$ semi-entscheidbar $\Rightarrow$ $\varphi'_A, \varphi'_{\overline{A}} \to M_a, M_{\overline{A}}$ entsprechende TM; $\to$ modifizierte $M'_A, M'_{\overline{A}} \to M'_A(n) und M'_{\overline{A}}(n)$ stoppen nach n schritten.
Berechnung von $\varphi_A(w)$:

```
for i=1,2,3,.... do
        if M'_A(I)=1 then
                return 1
        fi
        if M'_Ā(i)=1 then
                return 0
        fi
od
```

For-Schleife stoppt nach endlich vielen Schritten da w in A oder $\overline{A}$ liegen muss.

## 2.6 Das Halteproblem

Problem: Frage: Kann man entscheiden, ob eine TM mit Eingabe w hält oder nicht?
Modellierung als Sprache $H = \{w_1\#w_2|w_1$ Kodierung TM M, $w_2 \in \Sigma^*$, M hält auf $w_2\}$
Kodierung einer TM $M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ als binärer String:
Sei $Q = \{q_0, ..., q_n\}$, $\Gamma = \{a_0, ..., a_k\}$
1. Schreibe $\delta(q_i, a_j) = (q_{i'}, a_{j'}, y)$ mit $y \in \{0,1,2\}$ als Wort $w_{i,j,i',j',y} = \#\#bin(i)\#bin(j)\#bin(i')\#bin(j')\#bin(y)$
kodiert durch Konkatenation aller $w_{i,j}\ 0 \leq i \leq n,\ 0 \leq j \leq k; \to W$
Maschine: $bin(n)\#bin(k)\#W$ (eigentlich redundant)
2. Kodierung über {0,1}: Ersetze in $w_M\ 0 \to 00; 1 \to 01; \# \to 11$

$$M(w) = \begin{cases} \text{TM, die durch w kodiert wird} \\ \text{null, falls w keine Kodierung einer TM ist} \end{cases}$$

Universelle TM $M_u$ als Interpreter: $M_u$ erhält als Eingabe einen String $w\#w'$, wobei w eine Kodierte TM und w' eine Eingabe für diese ist. $M_u$ simuliert die Arbeitsweise von $M(w)$ auf der Eingabe w'.
Definition: Die Sprache $K = \{w \in \{0,1\}^*|M(w) \neq null$ und die TM M(w) mit Eingabe w hält$\}$ heißt das spezielle Halteproblem.

Erläuterung: Lasse TM auf ihre eingene Kodierung laufen.
Satz: Das spezielle Halteproblem K ist nicht entscheidbar.
Beweis: (indirekt) Annahme: K ist entscheidbar $\leftrightarrow \varphi_K$ ist berechenbar $\leftrightarrow \exists$ TM die $\varphi_K$ berechnet.
Konstruiere die Maschine M' wie folgt:
$\downarrow$ w
M 0 $\rightarrow$ w nicht in K
$\downarrow$ 1 w in K, laufe hier in Endlosschleife.
Damit gilt, dass M(w) auf w nicht terminiert.(berechnet $\varphi_K$)
Sei $w'$ = die Kodierung von M' (d.h. M(w')=M'
Nun lasse M' auf w' laufen: M' hält mit w' als Eingabe $\Leftrightarrow$ M hält auf w' nicht, d.h. M gibt 0 aus.

$\Leftrightarrow \varphi_K(w') = 0$
$\Leftrightarrow w' \notin K$
$\Leftrightarrow M(w') = M'$ hält auf w' nicht. Widerspruch.

### 2.6.1 Reduzierbarkeit

Definition: Sei $A, B \subseteq \Sigma^*$, A heißt reduzierbar auf B ($A \leqslant B$), falls es eine totale und berechenbare Funktion $f : \Sigma^* \rightarrow \Sigma^*$ gibt mit $x \in A \Leftrightarrow f(x) \in B \forall x \in \Sigma^*$
Lemma: Sei $A \leqslant B$

1. Falls B entscheidbar, dann ist auch A entscheidbar.
2. Falls A nicht entscheidbar, dann ist auch B nicht entscheidbar

Beweis: $\exists$ totale, berechenbare Funktion mit $x \in A \Leftrightarrow f(x) \in B$ für $x \in \Sigma^*$

1. B ist entscheidbar $\Rightarrow \varphi_B$ ist berechenbar. $\varphi_A : \varphi_A(x) = \varphi_B(f(x))$, f berechenbar. Da das berechenbar ist, ist A berechenbar.
2. Kontraposition von 1.

Entscheidbarkeit von $A \subseteq \Sigma^*$: Zeige $A \leqslant B$ für eine bekannte entscheidbare Menge B.
Unentscheidbarkeit von $A \subseteq \Sigma^*$: Zeige $B \leqslant A$ für eine bekannte unentscheidbare Menge B.

### 2.6.2 Das allgemeine Halteproblem H

Definition: $H = \{w\#x | M(w)$ mit Eingabe x hält$\}$
Satz: Das allgemeine Halteproblem H ist nicht entscheidbar.
Beweis: Zeige, dass $K \leqslant H$
Finde eine berechenbare Funktion f mit $w \in K \Leftrightarrow f(w) \in H$. Wähle $f : \Sigma^* \rightarrow \Sigma^*$ mit $f(w) = w\#w$
Analog : $H_0$ Halteproblem auf leerem Band ist nicht entscheidbar. ($H \leqslant H_0$) (Maschine, die ohne Eingabe startet, und sich im ersten Schritt selbst auf das Band codiert.)

## 2.7 Das Post'sche Korrespondenzprinzip (PCP)

Definition:
Gegeben: Endliche Folge von Paaren aus Strings $(x_1, y_1)...(x_k, y_k)$ $x_i, y_i \in \Sigma^+$
Problem: Gibt es eine Folge von Indizes $i_1, ..., i_n \in \{1, ..., k\}, n \geq 1$ mit $x_{i_1} x_{i_2} ... x_{i_n} = y_{i_1} y_{i_2} ... y_{i_n}$
Beispiele:

1. PCP: $\{(1, 101), (10, 00), (011, 11)\}$. Eine Lösung (1,3,2,3), da x: 101110011, y: 101110011.

Satz 2: PCP ist semi-entscheidbar. Beweis: Probiere alle Kombinationen durch bis Lösung gefunden.
Satz: PCP ist nicht entscheidbar. Beweis:

1. Betrachte eine spezielle Variante MPCP (modified PCP)
2. MPCP $\leqslant$ PCP
3. H $\leqslant$ MPCP (Reduzierbarkeit transitiv)

Definition MPCP: $K = \{(x_1, y_1), ..., (x_k, y_k)\} x_i, y_i \in \Sigma^+$
Frage: Gibt es eine Folge von Indizes $i_2, ..., i_n \in \{1, ..., k\}, n \geq 1$ mit $x_1 x_{i_2} ... x_{i_n} = y_{i_1} y_{i_2} ... y_{i_n}$, beginnend mit $x_1$
Lemma: MPCP $\leqslant$ PCP. Beweis: Finde totale, berechenbare Funktion
$f : \Sigma^* \rightarrow \Sigma^*$ mit $x \in MPCP \Leftrightarrow f(x) \in PCP$

Seien #,$ 2 Symbole in $\Sigma$, $\Sigma' \leftarrow \Sigma \cup \{\#, \$\}$
Wir bilden $K = ((x_1, y_1), .., (x_k, y_k)$ ab auf $f(K) = (x_0', y_0'), (x_1', y_1'), ..., (x_k', y_k'), (x_{k+1}', y_{k+1}'))$
wobei $x_i'$ entsteht aus $x_i$, füge hinter jedem Zeichen ein # ein ($x_i = a_1a_2a_3, x_i' = a_1\#a_2\#...$), $1 \leq i \leq k$
$y_i'$ entsteht aus $y_i$, füge vor jedem Zeichen ein # ein.
$x_0' = \#x_1', x_{k+1}' = \$$
$y_0' = y_1', y_{k+1}' = \#\$$
Beobachtung: Nur $x_0' und y_0'$ beginnen mit dem gleichen Zeichen # . Nur $x_{k+1}' und y_{k+1}' enden mit dem gleichen Zeichen \$$.
f ist offensichtlich berechenbar.
Beispiel: Für MPCP $K = ((ab, a), (c, abc), (a, b))$ (Bildlich als Dominosteine)
Hat Lösung: $(ab, a)(a, b)(ab, a)(c, abc)$
Lösung für PCP (f(K)): $(\#a\#b\#, \#a)(a\#, \#b)(a\#b\#, \#a)(c\#, \#a\#b\#c)(\$, \#\$)$
$f(K) = (\#a\#b\#, \#a)(a\#b\#, \#a)(c\#, \#a\#b\#c)(a\#, \#b)(\$, \#\$)$
Zu zeigen: 1. $k \in MPCP \Rightarrow f(k) \in PCP$
2. Rückrichtung
1.: $K \in MPCP$ d.h. $\exists$ Lösung $(1, i_2, ..., i_n)$ mit $x_1x_{i_2}...x_{i_n} = y_1y_{i_2}...y_{i_n} = a_1...a_s$
$i \geq 1$, für geeignete Symbole $a_i, 1 \leq i \leq s$
Dann ist $(0, i_2, ..., i_n, k+1)$ eine Lösung für PCP f(K).
Begründung: $x_0'x_{i_2}'...x_{i_n}'\$ = \#a_1\#a_2\#...\#a_s\#\$ = y_0'y_{i_2}'...y_{i_n}'\#\$$
2.: $f(K) \in PCP \Rightarrow K \in MPCP$ $\exists$ Lösung für f(K)
Sei nun $(i_1, i_2, .., i_n)$ eine Lösung minimaler Länge für f(K).
Beobachtungen:
1. $i_1 = 0$, weil nur $x_0'$ und $y_0'$ mit dem gleichen Zeichen beginnen.
2. $i_j \neq 0$ für $2 \leq i \leq n$, weil sonst im oberen Wort 2 aufeinanderfolgende # auftauchen.
3. $i_j \neq k+1$ für $1 \leq j < n$ d.h. Stein k+1 kann nur am Schluss stehen ($i_n = k+1$),
denn würe $ schon vorher in den beiden Folgen vorkommen, könnten wir die Lösung verkürzen (durch abschneiden an dieser Stelle).
Lösung für PCP f(K) hat folgende Struktur: $x_0'x_{i_2}'...x_{i_n}' = \#a_1\#a_2\#...\#a_s\#\$ = y_0'y_{i_2}'...y_{i_n}'$
Lösung für MPCP K: $x_1x_{i_1}...x_{i_{n-1}} = a_1...a_s = y_1y_{i_1}...y_{i_{n-1}}$
d.h. $f(K) \in PCP \Rightarrow K \in MPCP$
Lemma 2: $H \leqslant MPCP$ Idee: Simuliere die Arbeitsweise einer TM M durch ein Dominostein Spiel (MPCP)
M hält $\Rightarrow$ Spiel hat eine Lösung
Beispiel: Betrachte folgende TM M $\Sigma = \{0, 1\}, \Gamma = \{0, 1, \Box\}, Q = \{q_0, q_1, q_2, \overline{q}\}, F = \{\overline{q}\}$

| $\rho$ | 0 | 1 | $\Box$ |
|---|---|---|---|
| $q_0$ | $(q_0, 0, R)$ | $(q_1, 1R)$ | $(\overline{q}, 1, N)$ |
| $q_1$ | $(q_2, 0, R)$ | $(q_1, 1, R)$ | $(\overline{q}, 1, N)$ |
| $q_2$ | $(q_2, 0, R)$ | $(q_2, 1, R)$ | $(q_2, \Box, R)$ |

M erkennt, ob Eingabe die Form $\{0\}^*\{1\}^*$.Terminiert nur, wenn Eingabe die Form hat.
Bsp.: Konfigurationsfolge bei Eingabe 0011:
$q_00011 \mapsto 0q_0011 \mapsto 00q_011 \mapsto 001q_11 \mapsto 0011q_0 \mapsto 0011\overline{q}1$
Wir simulieren den Abluaf durch ein MPCP-Spiel (Dominos):
1. Startdomino: $(\#, \#\#q_00011\#)$
2. kopier-Dominos: $(0, 0), (1, 1), (\Box, \Box), (\#, \#)$ für jedes Zeichen aus $\Gamma \cup \{\#\}$
3. Überführungsdominos: Für jeden Eintrag in $\phi$-Tabelle:
$(q_00, 0q_0), (q_01, 1q_1), (q_0\Box, \overline{q}1), (q_10, 0q_2), (q_11, 1q_1), (q_1\Box, \overline{q}1), (q_20, 0q_2), (q_21, 1q_2), (q_2\Box, \Box q_2)$
4. Spezielle Überführung: $(q_\#, \overline{q}1\#), (q_1\#, \overline{q}1\#)$ (implizite blanks)
Wie können wir den Rückstand des oberen Strings aufholen?
5. Lösch-Dominos (nur für akzeptierenden Zustand):
$(\overline{q}0, \overline{q}), (\overline{q}1, \overline{q}), (\overline{q}\Box, \overline{q}), (0\overline{q}, \overline{q}), (1\overline{q}, \overline{q}), (\Box\overline{q}, \overline{q})$
6. Abschluss-Dominos:
$(\#\overline{q}\#\#, \#)$
Allgemein: Für beliebige TM M (Wort w)
Konstruktion des entsprechenden MPCP:
1. Start-Domino: $(\#, \#\#q_0\#)$
2. Kopieren: $(a, a)$ für alle $a \in \Gamma \cup \{\#\}$
3. Überführung: $(qa, q'c)$ falls $\varphi(q, a) = (q', c, N)$ für $q \in Q \backslash \{\overline{q}\}, a \in \Gamma$
$(qa, cq')$ falls $\varphi(q, a) = (q', c, R)$

$(bqa, q'bc)$ falls $\varphi(q, a) = (q', c, L)$
4. Spezielle Überführung (implizierte blanks): $(\#qa, \#q'\Box c)$ falls $\varphi(q, a) = (q', c, L)$
$(q\#, q'c\#)$ falls $\varphi(q, \Box) = (q', c, N)$
$(q\#, cq'\#)$ falls $\varphi(q, \Box) = (q', c, R)$
$(bq\#, q'bc)$ falls $\varphi(q, \Box) = (q', c, L)$
$(\#q\#, \#q'\Box c\#)$ falls $\varphi(q, \Box) = (q', c, L)$
5. Löschen: $(a\overline{q}, \overline{q}), (\overline{q}a, \overline{q})$ für alle $a \in \Gamma$
6. Abschluss $(\#\overline{q}\#\#, \#)$ Dies definiert die Funktion f in der Reduktion $H \leqslant MPCP$
$f : (M, w) \mapsto$Paare für MPCP

Lemma: M hält mit Eingabe w $\Leftrightarrow$ MPCP f(M,w) hat eine Lösung.
Beweis: Formale Variante eines Beispiels
Satz 1: $H \leqslant MPCP$
Satz 2: MPCP ist nicht entscheidbar
Satz 3: PCP nicht entscheidbar. MCPC Lösung:

1. beginnt mit erstem Stein.
2. der untere String rekonstruiert die Konfigurationsfolge.
3. der obere String folgt mit Rückstand.

# 3 Komplexitätstheorie

Untersucht den Ressourcen-Bedarf von Algorithmen bzw. Problemen
$\rightarrow$ Obere und untere Schranke (untere Schranke ist schwieriger zu finden)
Ressourcen: Laufzeit (Rechenschritte einer TM), Platzbedarf (Felder auf Bändern)
optimaler Algorithmus: optimal, falls er (asymptotisch) die Laufzeit der unteren Schranke erreicht.

## 3.1 Komplexitätsmaße und -klassen

Beschreibe ein Problem als formale Sprache $L \subseteq \Sigma^*$
$x \in L \Leftrightarrow x$ löst das Problem
Beispiele:

Erfüllbarkeitsproblem der Aussagenlogik (SAT)
Gegeben eine Boolsche Formel mit n Variablen
Frage: Gibt es die Belegung $x_1, ..., x_n$ sodass $f(x_1, ..., x_n) = 1$
Als Formale Sprache: $L_{SAT} = \{$alle Formeln mit n Variablen, die erfüllbar sind$\}$
Konkrete Fromel f erfüllbar $\Leftrightarrow f \in L_{SAT}$
Eine TM M, die $L_{SAT}$ akzeptiert (L(M)=$L_{SAT}$) löst das SAT-Problem

Optimierungsprobleme:

TSP (Travelling Salesman Problem):
n Orte in der Ebene, konstruiere die kürzeste Rundereise durch alle Orte.
Entsprechendes Entscheidungsproblem TSP(x) $\rightarrow$ gibt es eine Rundreise der Länge $\leq$ x?
$L_{TSP(x)} = \{...\}$
Lösung des TSP: Probiere alle möglichen x durch.
Oft nur triviale Algorithmen:
SAT: probiere alle Belegungen aus (Laufzeit $2^n$).
TSP: probiere alle Rundwege aus (Laufzeit $2^n$).

Sei nun M ein (Mehrband) TM die für alle Eingaben x hält.
Definition: $time_M : \Sigma^* \rightarrow \mathbb{N}, time_M(x) = \#$ Rechenschritte von M bei Eingabe x
Analog $space_M$ (Von M benötigte Felder auf Band)
Sei $L \subseteq \Sigma^*$ eine Sprache, $time_L(x) = min\{time_M(x) | L(M) = A\}$
M akzeptiert A
Komplexitätsklassen:

Sei $f : \mathbb{N} \times \mathbb{N}$ (Laufzeitfunktion)
TIME(f)=$\{A \subseteq \Sigma^* | time_A(x) \leq f(|x|)\}$
D.h. alle Probleme (Sprachen), die in f(n) Schritten gelöst werden können.

SPACE(f(n))=$\{A \subseteq \Sigma^* | space_A(x) \leq f(|x|)\}$
Bsp: TIME(3n+7)=TIME(O(n)) (Polynomiell)
TIME($2^n$) (Exponentiell)
TIME($2n^2 + 7n$)=TIME(O($n^2$)) (Polynomiell)

## 3.2 Die Klassen P und NP

Polynomiell $P : \mathbb{N} \to \mathbb{N}$ (vom Grad d)
$P(x) = a_d * x^d + ... + a_1 * x + a_0$, d Konstant
Komplexitätsklasse $P = \cup_{p\ Polynom} TIME(p(n)) = \cup_{k \geq 1} TIME(O(n^k))$
Alle Probleme, die in polynomieller Laufzeit durch eine (deterministische) TM gelöst werden.
Laufzeiten $nlog(n) = O(n^2) \to P$
Nicht in P $2^n, n^{log(n)}$...
Allgemein akzeptiert: 1. P effizient lösbares Problem, 2. exponentielle Algorithmen nicht effizient.
Klasse NP: Durch eine nicht-deterministische TM in Polynomzeit lösbar.
$NP = \cup_{P\ Polynom} NTIME(P(n)) = \cup_{k \geq 1} NTIME(O(n^k))$ NTIME: Nicht-deterministische TM (NTM)
Definition:
Eine nicht-deterministische TM hat in jedem Rechenschritt eine Menge möglicher Folgekonfigurationen.
Eine akzeptierende Berechnung besteht aus einer möglichen Folge von Konfigurationen startend in $s_0$ und endend in einem akzeptierenden Zustand.
Offensichtlich gilt $P \subseteq NP$, da jede det. TM auch eine NTM ist.
$P \neq NP$ oder P=NP ist noch offen (P-NP Problem)

## 3.3 NP-Vollständigkeit

Wir versuchen in NP schwierige Probleme zu definieren.
**Definition:** Seien $A, B \subseteq \overline{Z^*}$, A ist polynomiell reduzierbar auf B: $A \leqslant_P B$
Falls es eine totale und in Polynomzeit deterministische berechenbare Funktoin $f : \overline{Z^*} \to \overline{Z^*}$ gibt, sodass für alle $w \in \overline{Z^*}$ gilt: $w \in A \Leftrightarrow f(w) \in B$
**Lemma1:** $\leqslant_P$ ist transitiv
$A \leqslant_P B$ und $B \leqslant_P C \Rightarrow A \leqslant_P C$
**Lemma2:**
1. $A \leqslant_P B$ und $B \in P$
2. $A \leqslant_P B$ und $B \in NP \Rightarrow A \in NP$

Beweis: Da $A \leqslant_P B$ mittels der Funktion f in Polynomzeit P(n).
Sei $B \in P$ mittels einer TM M in Polynomzeit q(n)
Dann akzeptiert $(M_f; M)$ (nacheinander Ausführung) die Sprache A in Zeit $p(|x|) + q(|f(x)|)$ bei Eingabe x.
$\leqslant p(|x|) + q(p(|x|))$ Polynomiel
2. Analog

**Definition**
1. Eine Sprache A heißt NP-hart, falls für alle Sprachen $L \in NP$ gilt: $L \leqslant_P A$
2. A heißt NP-vollständig, falls A ist NP-hart und $A \in NP$ (schwierige Sprache in NP)

**Satz:**Sei A NP-Vollständig, dann gilt $A \in P \Leftrightarrow P = NP$
**Beweis:** "$\Rightarrow$"$A \in P$ und $L \in NP$ beliebig.
Da A NP-hart $\Rightarrow L \leqslant_P A$
Lemma2 $\Rightarrow L \in P$
"$\Leftarrow$"$P = NP \Rightarrow A \in P$
**NP-vollständiges Problem A**
Falls es einen polynomiellen deterministischen Algorithmus gibt für A, dann ist P=NP
Falls man zeigen kann, dass es keinen solchen Algorithmus gibt, dann gilt $P \neq NP$
Wie zeigt man, dass $A \subseteq \Sigma^*$ NP-Vollständig.
1. NP-hart: a) Annahme: Wir kennen eine NP-harte Sprache B. Dann zeige $B \leqslant_P A$ .
Daraus folgt A NP-hart wegen transitivität.

b) Zeige (falls noch kein NP-hartes Problem bekannt), dass für alle $L \in NP$ gilt: $L \leqslant_P A$
2. Zeige, dass $A \in NP$(oft sehr leicht / Guess and check)
**Ein erstes NP-Vollständiges Problem:** SAT (Erfüllbarkeitsproblem der Aussagenlogik)
Gegeben: Formel F der Aussagenlogik mit n Variablen.
Frage: Ist F erfüllbar, existiert eine Belegung, sodass F wahr ist.
Formal: Sprache L= Menge der Formeln, die erfüllbar sind. F erfüllbar $\Leftrightarrow F \in L$
Eine TM M, die L akzeptiert (L=L(M) löst das SAT-Problem
**Satz von Cook**
SAT ist NP-Vollständig
Beweis: 1. $SAT \in NP$ 2. SAT ist NP-hart
1.: Angabe einer NTM M für SAT
M stellt fest welche Variablen in F vorkommen
Seien diese $x_1...x_n$. M rät die Belegung jeder Variable, d.h. 0 oder 1 für jede Variable in einem Schritt.
berechne den Wert.(check)
Falls SAT erfüllbar, dann gibt es eine aktzeptierende Berechnung in Zeit O(p(n))
2.: SAT ist NP-Hart, d.h. $\forall L \in NP : L \leqslant_p SAT$
Betreachte beliebige Sprache L aus NP.
Sei dann eine NTM mit L(M)=L mit polynomieller Laufzeit, d.h. # Rechenschritte=p(n)
Obda $\delta(z_e, a) = \delta(z_e, a, N)$, d.h. man kann die Mashine immer genau p(n) Schritte laufen lassen.
Sei $x \in \Sigma^*$ die Eingabe von M, genauer: $x = x_1 x_2 x_3 ... x_n$
Konstruktion einer Formel (Aussagenlogik) F mit $x \in L(M) \Leftrightarrow$ F ist erfüllbar $(F \in SAT)$
Sei weiter: $\Gamma = \{a_1, ..., a_l\}$ Bandalphabet
$Z = \{z_1, ..., z_k\}$ Zustandsmenge
Parameter: n: Eingabegröße, p(n): # Anzahl Schritte, l: Größe des Bandalphabets, k: # Zustände.
Da M p(n) Schritte ausführt, befindet sich der Kopf immer an einer Position $i \in \{-p(n), ..., p(n)\}$
F enthält folgende Variablen:

| | | |
|---|---|---|
| $zust_{t,z}$ | $t = 0, ..., p(n), z = 1, ..., k$ | $zust_{t,z} = 1 \Leftrightarrow$ M befindet sich nach t Schritten im Zustand z |
| $pos_{t,i}$ | $t = 0, ..., p(n), i = -p(n), ..., p(n)$ | $pos_{t,i} = 1 \Leftrightarrow$ Der Kopf von M befindet sich nach t Schritten an Stelle i |
| $band_{t,i,a}$ | $t = 0, ..., p(n), i = -p(n), .., p(n), a \in \{1, ..., l\}$ | $band_{t,i,a} = 1 \Leftrightarrow$ Zum Zeitpunkt t steht auf Position i das Zeichen a |

Beobachtung: # Variablen =O(q(n), q Polynom, $q \leq c * (p(n))^2$
Aufbau von F: Hilfsformel $G(x_1, .., x_m) = 1 \Leftrightarrow$ für genau ein $i \in \{1, ..., m\}$ gilt $x_i = 1$
Behauptung: G existiert und $|G| = O(m^2)$
Beweis: $G(x_1, .., x_m) = \vee_{i=1}^{m}(x_i) \wedge (\wedge_{j=1}^{m-1} \wedge_{k=j+1}^{m} (\overline{x_j \wedge x_k}))$
Dann Definiere $F = R \wedge A \wedge U_1 \wedge U_2 \wedge E$
R: Randbedingungen, A: Anfangsbedingung, U: Übergangsbedingungen, E: Endbedingung
R: zu jedem Zeitpunkt $t \in \{0, ..., p(n)\}$
$zust_{t,z} = 1$ für genau einen Zustand z.
$pos_{t,i} = 1$ für genau eine Position i
für jede Position $i \in \{-p(n), ..., p(n)\}$ $band_{t,i,a} = 1$ für genau ein Symbol a
$R = \wedge_t (G(zust_{t,i}, ..., zust_{t,k}) \wedge G(pos_{t,-p(n)}, ..., pos_{t,p(n)}) \wedge (\wedge_i G(band_{t,i,1}, ..., band_{t,i,l})))$
Es gilt: $|R| = O((p(n))^2)$
**Anfangsbedingung**, Wert der Variablen zum Zeitpunkt t=0:
$A = zust_{0,0} \wedge (\wedge_{i=0}^{n-1} band_{0,i,x_{i+1}}) \wedge (\wedge_{i=-p(n)}^{-1} band_{0,i,\square}) \wedge (\wedge_{i=n+1}^{p(n)} band_{0,i,\square}))$
$|A| = O(p(n))$
**Übergang 1**
Beschreibt den Übergang von Zeitpunkt t auf t+1 für $y \in \{-1, 0, 1\}$ (Kopfbewegungen)
$U_1 = \wedge_{t,z,i,a}[zust_{t,z} \wedge pos_{t,i} \wedge band_{t,i,a} \rightarrow \vee_{(z',a',y) \in \delta(z,a)}(zust_{t+1,z'} \wedge pos_{t+1,i+y} \wedge band_{t+1,i,a'}]$
$|U_1| = O((p(n))^2)$
**Übergang 2**
An allen übrigen Bandstellen (wo der Kopf nicht steht) passiert nichts.
$U_2 = \wedge_{t,i,a}[(\overline{pos_{t,i}} \wedge band_{t,i,a}) \rightarrow band_{t+1,i,a}]$

$|U_2| = O((p(n))^2) = O(q(n))$
**Endbedingung**
Es wird überprüft, ob der Endzustand erreicht ist (dies geschieht immer auch zum Zeitpunkt p(n))
$E = zust_{p(n),z_e} \wedge_{z \in E} zust_{p(n),z}$

Weitere NP-vollständige Probleme:

Definition: 3SAT:

Gegeben: Boolsche Formel F in KNF mit höchstens (genau) 3 Literalen pro Klausel.

Variablen: $x_1, ..., x_n$ Literale: $z \in \{x_1, .., x_n, \overline{x_1}, ..., \overline{x_n}\}$

Satz1: 3SAT ist NP-Vollständig Bemerkung: 2SAT in P

Beweis:

i. 3SAT$\in$ NP guess&check

ii. 3SAT ist NP-hart

Reduktion: SAT $\leq_P$ 3SAT

Angabe eines poylnomiellen Verfahrens, dass eine beliebige Formel F in eine KNF F' mit höchstens 3 Literalen pro Klausuel umformt mit

F erfüllbar $\Leftrightarrow$ F' erfüllbar (Erfüllbarkeitsequivalenz)

Annahme: Formel F auch in KNF. K-Klausel in F: Klausel mit k Literalen.

K=1,2,3 übernehmen

K=4: $(z_1 \vee z_2 \vee z_3 \vee z_4)$ ersetze durch $(z_1 \vee z_2 \vee h) \wedge (\overline{h} \vee z_3 \vee z_4)$

K>4: $(z_1 \vee z_2 \vee ... \vee z_k) \Leftrightarrow_{erf} (z_1 \vee z_2 \vee ... \vee z_{k-2} \vee 1) \wedge (\overline{h_1} \vee z_{k-1} \vee z_k)$.

Wiederhole bis alle Klauseln 3 Literale.

Pro Iteration singt Länge um 2 $\rightarrow$ höchstens $k_{max}$ Iterationen $\rightarrow$ Polynomiell

$\Rightarrow$ SAT $\leq_P$ 3SAT, d.h. 3SAT ist NP-hart.

Das Clique-Problem:

Definition: Gegeben: Ein ungerichteter Graph G=(V,E) und eine Zahl $k \in \mathbb{N}$

Frage: Enthält G einen vollständigen Teilgraphen mit k Knoten

Satz: Clique ist NP-vollständig

Beweis: 1. CLIQUE $\in$ NP (guess and check)

2. CLIQUE ist NP-hart:

Zeige 3SAT $\leq_P$ CLIQUE

Sei F KNF mit genau 3 Literalen pro Klausel

$F = (z_{1,1} \vee z_{1,2} \vee z_{1,3}) \wedge ... \wedge (z_{m,1} \vee z_{m,2} \vee z_{m,3})$

wobei $z_{i,j} \in \{x_1, ..., x_n, \overline{x_1}, ..., \overline{x_n}$

Konstruktion eines äquivalenten CLIQUE-Problems:

1. Graph G=(V,E) mit $V = \{v_{1,1}, v_{1,2}, ..., v_{m,1}, v_{m,2}, v_{m,3}\}$

$E = \{(v_{i,j}, v_{p,q}) | i \neq p \ und \ z_{i,j} \neq \overline{z_{p,q}}\}$

2. Die Zahl k:=m (Anzahl Klauseln)

Es gilt: F ist erfüllbar durch eine Belegung B $\Leftrightarrow$ Jede Klausel enthält ein Literal das unter B den Wert 1 annimmt z.B. $z_{1,j_1}, ..., z_{m,j_m}$

$\Leftrightarrow \exists$ Literale $z_{1,j_1}, ..., z_{m,j_m}$ die paarweise nicht komplementär sind.

$\Leftrightarrow \exists$ Knoten in G: $v_{1,j_1}, ..., v_{m,j_m}$, die paarweise durch Kanten verbunden sind

$\Leftrightarrow$ G hat eine Clique der Größe K