# Grundlagen Theoretischer Informatik 2

## WiSe 2011/12 in Trier

Henning Fernau
Universität Trier
fernau@uni-trier.de

## Grundlagen Theoretischer Informatik 2 Gesamtübersicht

- Organisatorisches; Einführung
- Ersetzungsverfahren: Grammatiken und Automaten
- Rekursionstheorie-Einführung
- Komplexitätstheorie-Einführung

## Zentrale Frage der Rekursionstheorie

- Was ist prinzipiell mit Rechnern möglich?

Hierbei Grundlage: Church-Turing-These
Nur so kann man beweisen, dass gewisse Fragen durch Rechner nicht beantwortet werden können.
Rekursionstheorie beschäftigt sich also mit den prinzipiellen Grenzen der Informatik als Computer Science.

# Wieviele berechenbare Funktionen gibt es?

Satz: Die Menge aller von Turingmaschinen berechneten Funktionen über einem gegebenen Alphabet $\Sigma$ ist abzählbar, es gibt zu jedem $\Sigma$ eine totale Funktion $h$ mit Definitionsbereich $\{0,1\}^*$, deren Bild alle(!) berechenbaren Funktionen $g : \Sigma^* \multimap \Sigma^*$ umfasst.

Konstruiere dazu Liste aller Turing-berechenbaren Funktionen, beispielsweise wie folgt:

- Beschränkung auf deterministische Turingmaschinen $TM = (S, \Sigma, A, \delta, s_0, \square, F)$
- Festes Alphabet $\Sigma$ mit $\{0,1,\#\} \subseteq \Sigma$
- $\delta(s,a)$ undefiniert für alle $a \in A$ und Endzustände $s \in F$
- $\delta(s,a)$ definiert für alle $a \in A$ und Nicht-Endzustände $s \notin F$
- Falls bei Eingabe $w$ mit Konfiguration $us_fv$ hält:

$$f_{TM}(w) := \text{ das längste Präfix } \in \Sigma^* \text{ von } v$$

## **Fortsetzung der Konstruktion** O.B.d.A.:

- $S = \{s_0, s_1, \ldots, s_n\}$ mit Startzustand $s_0$
- Endzustände am Listenende: $F = \{s_{n-e+1}, \ldots, s_n\}$ mit $e = |F|$
- $A = \{a_0, a_1, \ldots, a_k\}$
- $E \subset A$ mit $E = \{a_0, a_1, \ldots, a_{l-1}\}$ für $l = |E|$
- $\square = a_d$ für ein $d$

Übergänge der Form '$\delta(s_i, a_j)$ enthält $(s_t, a_m, B)$' beschreibbar durch

$$\Delta_{i,j} := \#\#bin(i)\#bin(j)\#bin(t)\#bin(m)\#bin(b)$$

mit $b = 0$, wenn $B = L$, $b = 1$, wenn $B = N$ und $b = 2$, wenn $B = R$.

$\Rightarrow$ $TM$ beschreibbar durch $n$, $e$, $k$, $l$, $d$ und Angabe der $\delta(s_i, a_j)$,
d.h. als Wort über $\{0, 1, \#\}$ in folgender Form:

$$bin(n)\#bin(e)\#bin(k)\#bin(l)\#bin(d)\#\Delta_{0,0}...\Delta_{n-e,k}$$

Codiere z.B. $0 \mapsto 00$, $1 \mapsto 01$ und $\# \mapsto 11$
$\Rightarrow$ *jede* Turingmaschine durch ein $w \in \{0, 1\}^*$ beschreibbar.

## **Ziel: Aufzählung aller berechenbaren Wortfunktionen**, dazu

- Zu jedem $w \in \{0,1\}^*$ definiere $M_w$ durch

$$M_w := \begin{cases} M, & \text{falls } w \text{ eine Beschreibung der Maschine } M \text{ ist .} \\ M^{ud}, & \text{sonst} \end{cases}$$

- Dabei sei $M^{ud}$ beliebig gewählt, z.B.

$$M^{ud} := (\{s_0, s_1\}, E, E \cup \{\Box\}, \delta, \Box, \{s_1\})$$

  mit $\delta(s_0, a) = (s_0, a, N)$ für $a \in A$ (d.h., $M^{ud}$ hält nie an...)

- $h_w$ sei die von $M_w$ berechnete Wortfunktion.

$\Rightarrow w$ ist 'Programm' für die Funktion $h_w : E^* \multimap E^*$

$\Longrightarrow$ Abbildung $w \mapsto h_w$ ist 'Programmiersprache'

## Wichtige Schlussfolgerungen

1. Rechner können nicht alle Aufgaben lösen.
2. Formalisierung A: Es gibt Wortfunktionen, die von keiner Turingmaschine berechet werden können.

   Eigentlich zeigt der Beweis noch stärker: “Fast alle” Wortfunktionen sind nicht berechenbar.

   Denn: Es gibt $|\mathbb{N}^{\mathbb{N}}|$ viele Wortfunktionen, aber nur $|\mathbb{N}|$ viele berechenbare.
3. Formalisierung B: Es gibt formale Sprachen, die nicht vom Typ 0 sind.

   Jedenfalls ergibt sich das durch einen analogen Beweis.
4. Natürliche Zahlen, Wörter und berechenbare Wortfunktionen sind in gewissem Sinne “dasselbe”. Diese wichtige Erkenntnis werden wir gleich formalisieren.

## Notationen

Sei $\Sigma$ ein Alphabet. Eine *Notation* (Programmiersprachenineterpretation) für die berechenbaren Funktionen $g : \Sigma^* \multimap \Sigma^*$ ist eine surjektive Funktion

$$h' : \{0,1\}^* \to \{ \text{ berechenbare Wortfunktionen über } \Sigma^* \}$$

- Jedem $w \in \{0,1\}^*$ (also jedem Programmtext) wird eine berechenbare Wortfunktion zuordnet.
- Zu jeder berechenbaren Wortfunktion $g$ gibt es ein Wort $w \in \{0,1\}^*$ derart, dass $h'(w)$ gerade die Funktion $g$ ist. Jedes solche $g$ lässt sich also programmieren.

Wir schreiben oft auch $h'_w$ für die Funktion $h'(w)$.
Ein konkretes Beispiel liefert die Funktion $w \mapsto h_w$ von oben.

## **Zwei Arten charakteristischer Funktionen** (Indikatorfunktionen)

Betrachte Teilmenge $A \subseteq \Sigma^*$:

- Die *charakteristische Funktion*

  $$\chi_A : \Sigma^* \to \{0, 1\}$$

  ist definiert durch

  $$\chi_A(w) := \begin{cases} 1, & w \in A \\ 0, & w \notin A \end{cases}$$

- Die *eingeschränkte charakteristische Funktion*

  $$\chi'_A : \Sigma^* \multimap \{0, 1\}$$

  ist definiert durch

  $$\chi'_A(w) := \begin{cases} 1, & w \in A \\ \text{undefiniert}, & w \notin A \end{cases}$$

Wichtig: $\chi_A$ ist total; $\chi'_A$ ist partiell mit Definitionsbereich $A$

## Entscheidbarkeit und rekursive Aufzählbarkeit

1. Eine Teilmenge $A \subseteq \Sigma^*$ heißt ***entscheidbar***, wenn ihre charakteristische Funktion $\chi_A : \Sigma^* \to \{0, 1\}$ **berechenbar** ist.

2. Eine Teilmenge $A \subseteq \Sigma^*$ heißt ***semi-entscheidbar***, wenn ihre eingeschränkte charakteristische Funktion $\chi'_A : \Sigma^* \multimap \{0, 1\}$ **berechenbar** ist.

Für Mengen $A \subseteq \mathbb{N}^k$ werden die Begriffe analog definiert.

## Anschluss an Formale Sprachen

Satz: Jede konstextsensitive Sprache ist entscheidbar.

Beweisidee: Ist $L$ kontextsensitiv, so gibt es monotone Grammatik $G$ mit $L = L(G)$. Will man feststellen, ob $w \in L$, also ob $S \overset{*}{\Rightarrow} w$ gilt, mit $w \neq \lambda$, so muss man daher lediglich Satzformen der Länge höchstens $|w|$ betrachten. In Abhängigkeit vom Gesamtalphabet $V = N \cup \Sigma$ gibt es höchstens $\sum_{i=1}^{|w|} |V|^i \leq |V|^{|w|+1}$ viele "interessante" Satzformen. Die Frage $S \overset{*}{\Rightarrow} w$ lässt sich also auf das Erreichbarkeitsproblem in einem endlichen Graphen zurückführen.

## Der Satz von Post

Satz: Eine Sprache $A$ ist genau dann entscheidbar, wenn sowohl $A$ als auch ihr Komplement $\Sigma^* \setminus A$ semi-entscheidbar sind.

Beweis '$\Rightarrow$': Sei $A$ entscheidbar, d.h., $\chi_A$ ist berechenbar (mit einer Maschine $M$).
Konstruiere $M'$ wie folgt:

- Bei Eingabe $w$ berechnet $M'$ zunächst $\chi_A(w)$ (mittels $M$).
- Ist das Resultat $1$, so hält $M'$ und akzeptiert $w$.
- Ist das Resultat $0$, so startet $M'$ eine Endlosschleife.

$\Rightarrow$ $M'$ berechnet $\chi'_A$, d.h., $A$ ist semi-entscheidbar.

Vertausche Rollen von $1$ und $0$ bei $M'$:
$\Rightarrow$ $M'$ berechnet jetzt $\chi'_{\Sigma^* \setminus A}$, d.h., $\Sigma^* \setminus A$ ist semi-entscheidbar.

## Der Satz von Post

Satz: Eine Sprache $A$ ist genau dann entscheidbar, wenn sowohl $A$ als auch ihr Komplement $\Sigma^* \setminus A$ semi-entscheidbar sind.

Beweis '$\Leftarrow$': Gegeben Maschinen $M'$, $M''$:
$M'$ berechne $\chi'_A$, $M''$ berechne $\chi'_{\Sigma^* \setminus A}$.
Konstruiere neue Turingmaschine $M$ wie folgt:

- $w$ sei Eingabe für $M$.
- $M$ simuliert abwechselnd einen Rechenschritt von $M'$ auf $w$ und einen Schritt von $M''$ auf $w$.
- Hält $M'$ an, so hält auch $M$ und akzeptiert $w$.
- Hält $M''$ an, so hält auch $M$ und verwirft $w$.
- Bei Eingabe von $w$ hält genau eine der Maschinen $M'$ und $M''$, also $M$ hält in jedem Fall an und berechnet $\chi_A$.

## Semi-Entscheidbarkeit und rekursive Aufzählbarkeit

Eine Sprache $A \subseteq \Sigma^*$ heißt *rekursiv aufzählbar*, falls $A$ leer ist oder Bildbereich einer totalen berechenbaren Funktion $f : \mathbb{N} \to \Sigma^*$ ist:

$$A := \{f(0), f(1), ...\}$$

Für Teilmengen $A \subseteq \mathbb{N}^k$ wird der Begriff analog definiert.

Satz: Eine Sprache $A$ aus $\Sigma^*$ ist genau dann semi-entscheidbar, wenn sie rekursiv aufzählbar ist.

**Semi-Entscheidbarkeit und rekursive Aufzählbarkeit**; Beweis: $\Leftarrow$

Sei $A$ rekursiv aufzählbar.

Fall (1), $A = \emptyset$: Dann $\chi_A(x) = 0$ für alle $x$, also $A$ entscheidbar.

Fall (2), $A = \{f(0), f(1), ...\}$ mit totalem berechenbaren $f : \mathbb{N} \to E^*$:

Betrachte (berechenbares!) $g$ definiert über

```
INPUT(w);
FOR n = 0,1,2,3,... DO
    IF f(n) = w THEN OUTPUT(1) END;
END.
```

$w \in A \Rightarrow g(w) = 1$
$w \notin A \Rightarrow g(w)$ undefiniert
$\Longrightarrow A$ semi-entscheidbar

## **Semi-Entscheidbarkeit und rekursive Aufzählbarkeit**; Beweis: $\Rightarrow$

Sei nun $A$ semi-entscheidbar, $A \neq \emptyset$.
$M$ sei eine Turingmaschine, die $\chi'_A$ berechnet.
$a$ sei beliebig gewähltes Element von $A$.
Betrachte (berechenbares!) $g$ definiert über

```
INPUT(n);
k := p1(n); l := p2(n); w := ν(k);
IF   Angesetzt auf w stoppt M
                nach höchstens l Schritten mit Ausgabe von 1
THEN OUTPUT(w)  ELSE OUTPUT(a)
END.
```

- $p_1$, $p_2$: LOOP-berechenbare Umkehrfunktionen der Cantorschen Bijektion $\langle \cdot, \cdot \rangle$ zwischen $\mathbb{N}^2$ und $\mathbb{N}$
- $n$ codiert (bijektiv!) zwei Zahlen $k$ und $l$
- $k$ codiert (bijektiv!) Worte $w$

*Schwalbenschwanzkonstruktion*

$\Longrightarrow$ Algorithmus testet

- für alle möglichen Wörter $w \in E^*$ und
- für alle möglichen Laufzeiten $l$,
- ob $M$ bei Eingabe von $w$ nach $l$ Schritten anhält.

1. Die berechnete Funktion $g$ ist total.
2. Stets gilt $g(n) \in A$.
3. Für jedes $w \in A$ hält $M$ nach $l_w$ Schritten für ein $l_w \in \mathbb{N}$,
   d.h. $g(n_w) = w$ für $n_w$ mit $p_2(n_w) = l_w$ und $\nu p_1(n_w) = w$.

Damit $A = \{g(n) \mid n \in \mathbb{N}\}$.

## Äquivalenzen auf einen Blick

Für $A \subseteq \Sigma^*$ sind folgende Aussagen äquivalent:

- $A$ ist Sprache vom Typ 0.
- $A$ wird von einer nichtdeterministischen Turingmaschine erkannt.
- $A$ wird von einer deterministischen Turingmaschine erkannt.
- Es gibt eine deterministische Turingmaschine, die bei Eingabe eines Wortes $w \in \Sigma^*$ genau dann nach endlich vielen Schritten anhält, wenn $w \in A$ ist.
- $A$ ist semi-entscheidbar, d.h., die eingeschränkte charakteristische Funktion $\chi'_A$ ist berechenbar.
- $A$ ist Definitionsbereich einer berechenbaren Funktion.
- $A$ ist rekursiv aufzählbar.
- $A$ ist leer oder Wertebereich einer totalen berechenbaren Funktion.
- $A$ ist Wertebereich einer (partiellen) berechenbaren Funktion.

Die blauen Behauptungen wurden nicht in der VL bewiesen.

## Das spezielle Halteproblem

Sei $\mathfrak{h}$ wieder die Notation der berechenbaren Wortfunktionen.
$M_w$ sei die durch $w \in \{0,1\}^*$ bezeichnete Turingmaschine, die $\mathfrak{h}_w$ berechnet.

Das *spezielle Halteproblem* oder *Selbstanwendbarkeitsproblem* ist die Menge

$$\begin{aligned} K &:= \{w \in \{0,1\}^* \mid M_w \text{ angesetzt auf } w \\ &\qquad\qquad \text{hält nach endlich vielen Schritten an}\} \\ &= \{w \in \{0,1\}^* \mid \mathfrak{h}_w(w) \text{ ist definiert }\} \end{aligned}$$

Satz: Das spezielle Halteproblem $K$ ist rekursiv aufzählbar, aber nicht entscheidbar.

**Der Beweis** dieses wichtigen Satzes benötigt weitere Begriffe, die wir im Folgenden zunächst studieren.

Was bedeutet aber dieser Satz?

Er zeigt ein konkretes Beispiel für eine nicht berechenbare Funktion.

Diese Funktion ist zudem von großer praktischer Bedeutung:
Man kann nicht algorithmisch feststellen, ob ein Programm hält!

**Zwei wichtige Eigenschaften von Notationen** $h'$ (wie z.B. $h$)

*utm-Eigenschaft* (universelle Turingmaschine)

Die Wortfunktion, die Wörter $w\#x$ für $w \in \{0,1\}^*$ und $x \in E^*$ auf $h'_w(x)$ abbildet, ist berechenbar.

*smn-Eigenschaft* (Projektion)

Ist $g : E^* \multimap E^*$ eine berechenbare Wortfunktion, so gibt es eine totale berechenbare Wortfunktion $r$ derart, dass für alle $x \in \{0,1\}^*$ und alle $y \in E^*$ gilt:

$$g(x\#y) = h'_{r(x)}(y)$$

Beachte: Paare von Wörtern $x\#y$ lassen sich bijektiv mit der Cantorschen Paarfunktion und $\nu$ bzw. $\nu^{-1}$ auf Wörter bzw. Zahlen abbilden.

Satz: Die von uns oben definierte Funktion $\mathfrak{h}$, die jedes Wort $w$ aus $\{0,1\}^*$ auf eine berechenbare Funktion $\mathfrak{h}_w : \Sigma^* \text{—}\circ\!\!\rightarrow \Sigma^*$ abbildet, ist eine Notation der berechenbaren Wortfunktionen.

$\mathfrak{h}$ hat zudem die utm-Eigenschaft und die smn-Eigenschaft.

Beweis: Nach Konstruktion ist $\mathfrak{h}$ Notation der berechenbaren Wortfunktionen:

- $\mathfrak{h}_w$ ist berechenbare Funktion für jedes $w$.
- Jede Turingmaschine wird durch ein $w$ kodiert.

Zur utm-Eigenschaft: Bilde *universelle Turingmaschine* $\mathfrak{U}$ wie folgt:

- Bei Eingabe $w\#x$ bestimmt $\mathfrak{U}$ die von $w$ codierte Maschine $M$ (d.h. direkt $M = M_w$ oder aber $M = M^{ud}$)
- Danach simuliert $\mathfrak{U}$ die Maschine $M$ auf der Eingabe $x$ (wie ein Interpreter).

$f_{\mathfrak{U}}$ ist dann die für die utm-Eigenschaft notwendige Funktion!

## Zur smn-Eigenschaft

Zur berechenbaren Wortfunktion $g : E^* \multimap E^*$ sei $M_g$ fixierte TM, die $g$ berechnet.
Betrachte eine Turingmaschine $R$, die wie folgt arbeitet:

- $R$ liest ihre Eingabe $x$ und modifiziert dann (eine Codierung von) $M_g$ zu (der Codierung einer TM) $M_g^x$ mit folgender Arbeitsweise:

  Bei einer Eingabe $y$ schreibt $M_g^x$ zunächst das Wort $x\#$ links vor das Wort $y$ und arbeitet dann weiter wie $M_g$.

- $x$ wird dabei in der Zustandsmenge von $M_g^x$ codiert.

- $R$ erzeugt dann als Ausgabe eine Codierung von $M_g^x$.

- $M_g$ wird in der Zustandsmenge von $R$ codiert.

$R$ erzeugt also aus einer Eingabe $x$ (die Codierung von) $M_g^x$ mit

$$h_{f_R(x)}(y) = f_{M_g^x}(y) = f_{M_g}(x\#y) = g(x\#y)$$

$f_R$ ist also das in der smn-Eigenschaft geforderte (totale!) $r$.

## Erinnerung

Berechnung der Summe zweier natürlicher Zahlen $m$ und $n$ als Funktion $f : \mathbb{N}^2 \to \mathbb{N}$ mit $f(m, n) = m + n$:

```
import java.util.Vector;
import java.math.BigInteger;
class Addition {
  public static void main(String args[]) {
    BigInteger m=new BigInteger(args[0]);
    BigInteger n=new BigInteger(args[1]);
    while (n.compareTo(BigInteger.ZERO) > 0) {
      n = n.subtract(BigInteger.ONE);
      m = m.add(BigInteger.ONE);
    }
    System.out.println(m.toString());
  }
}
```

## Anwendung der smn-Eigenschaft in JAVA

Betrachte Funktion zu Addition zweier Zahlen in JAVA (s.o.),
wende smn-Eigenschaft für ersten Parameter $m$ an:

```
class SMN {
public static void main(String args[]) {
System.out.println(¨import java.util.Vector;¨);
System.out.println(¨import java.math.BigInteger;¨);
System.out.println(¨class Addition {¨);
System.out.println(¨public static void main(String args[]){¨);
System.out.println(¨BigInteger m=new BigInteger(\”¨
                + args[0] + ¨\”);¨ );
System.out.println(¨BigInteger n=new BigInteger(args[0]);¨);
System.out.println(¨while(n.compareTo(BigInteger.ZERO)>0){¨);
System.out.println(¨n = n.subtract(BigInteger.ONE);¨);
System.out.println(¨m = m.add(BigInteger.ONE);¨);
System.out.println(¨}¨);
System.out.println(¨System.out.println(m.toString());¨);
System.out.println(¨}¨);
System.out.println(¨}¨);
} }
```

## Veranschaulichungen der Eigenschaften

Universelle Turingmaschinen sind Interpreter, die bei Eingabe einer Notation einer berechenbaren Funktion $f$ (also z.B. einem JAVA- oder Turingmaschinenprogrammtext für $f$) und dessen Argument $x$ den Wert von $f(x)$ ausrechnet.

Die smn-Eigenschaft hat eher was mit Compilern zu tun:
Bei Eingabe der Notation einer berechenbaren Funktion $f$ wird ein JAVA-Programm erzeugt, das bei Eingabe von $x$ den Wert $f(x)$ ausrechnet.
Das vorige "Anwendungsbeispiel" funktioniert sogar etwas wörtlicher!
Dies entspricht dem "Currying" in funktionalen Programmiersprachen wie ML oder Haskell.

## Eine typische Aufgabe

Zu zeigen ist: Es existiert eine totale und berechenbare Funktion $g\colon \mathbb{N}^2 \to \mathbb{N}$, so dass für $i, j, x \in \mathbb{N}$ gilt: $h_{g(i,j)}(x) = h_i(x) + h_j(x)$.

Erinnnere Cantorsche Paarfunktion $\langle \cdot, \cdot \rangle$.
Definiere $\psi(\langle i, j \rangle, x) = h_i(x) + h_j(x)$.
$\psi$ ist berechenbar wegen utm-Eigenschaft.
Nach dem smn-Theorem gilt: $h_{\hat{g}(\langle i,j \rangle)}(x) = \psi(\langle i, j \rangle, x)$ für totale und berechenbare Funktion $\hat{g}\colon \mathbb{N} \to \mathbb{N}$,
aus der das gesuchte $g$ sich mit der Inversen der Cantorschen Bijektion ergibt.

## **Das spezielle Halteproblem**; der Beweis

$K$ ist rekursiv aufzählbar:

- $g(w\#v) := h_w(v)$ ist berechenbar (utm-Eigenschaft)
- also auch $f$ mit $f(w) := g(w\#w) = h_w(w)$
- $w \in K \Leftrightarrow f(w)$ definiert

Annahme: $K$ sei entscheidbar.

- Dann insbesondere $E^* \setminus K$ semi-entscheidbar
- Sei $TM$ eine Turingmaschine, die genau dann auf $w$ hält, wenn $w \in E^* \setminus K$
- Sei $x$ ein Codewort für $TM$, d.h.

$$h_x(w) \text{ ist definiert } \Leftrightarrow w \in E^* \setminus K$$

Betrachte $h_x(x)$:

$$\begin{aligned} x \in K &\Leftrightarrow TM \text{ hält nicht auf } x \\ &\Leftrightarrow h_x(x) \text{ nicht definiert} \\ &\Leftrightarrow x \notin K \text{ (Widerspruch...)} \end{aligned}$$

Also war die Annahme falsch, und $K$ ist *nicht* entscheidbar.

## Das spezielle Halteproblem

Der Beweis als *Diagonalisierung* jetzt für Zahlfunktionen:

$K$ jetzt Zahlmenge; $i \in K$ gdw. $h_i(i)$ ist definiert.

Übliche Schreibweise: $\uparrow$ für undef. und $\downarrow$ für def..

Betrachte unendliche Tabelle für $h_i(j)$, die z.B. wie folgt aussieht:

| $h_i(j)$ | $j=0$ | $j=1$ | $j=2$ | $j=3$ | $\ldots$ |
|---|---|---|---|---|---|
| $i=0$ | $\uparrow$ | $\downarrow$ | $\downarrow$ | $\uparrow$ | $\ldots$ |
| $i=1$ | $\downarrow$ | $\downarrow$ | $\uparrow$ | $\uparrow$ | $\ldots$ |
| $i=2$ | $\uparrow$ | $\downarrow$ | $\uparrow$ | $\uparrow$ | $\ldots$ |
| $i=3$ | $\uparrow$ | $\uparrow$ | $\downarrow$ | $\uparrow$ | $\ldots$ |

Invertiert man alle Pfeile der Diagonalen, so beschreibt dann der $j$-te (invertierte) Diagonaleintrag $\chi'_{\mathbb{N}\setminus K}$ (lies $\uparrow = 1$), diese partielle Zahlfunktion hat keine Notation, da sie sich von jedem $h_i$ bei Eingabe von $i$ unterscheidet.