CoNe
Freiburg

Informatik III

3.3 Orakel, Selbstreferenz und Beschreibungskomplexität

Christian Schindelhauer

Albert-Ludwigs-Universität Freiburg
Institut für Informatik
Rechnernetze und Telematik

Wintersemester 2007/08

Berechenbarkeitstheorie

Der Satz von Rice

2

Der Satz von Rice

- **Jede Menge von Turing-Maschinen, die über eine funktionale Eigenschaft definiert werden, ist nicht entscheidbar.**
- **Theorem**
  - Sei $K \subseteq P(\Sigma^*)$ eine nicht triviale Klasse von rekursiv aufzählbaren Sprachen, d.h.
    - K ist nicht leer und
    - K beinhaltet nicht alle rekursiv aufzählbare Sprachen
  - Dann ist die folgende Sprache nicht entscheidbar

$$L_K = \{\langle M \rangle \mid M \text{ ist eine TM und } L(M) \in K\}$$

- **Beispiele**

$$L_1 = \{\langle M \rangle \mid M \text{ ist eine TM, die eine reguläre Sprache akzeptiert}\}$$

$$L_2 = \{\langle M \rangle \mid M \text{ ist eine TM, welche keine Eingabe akzeptiert}\}$$

$$L_3 = \{\langle M \rangle \mid M \text{ ist eine TM, so dass } M(10) = 1\}$$

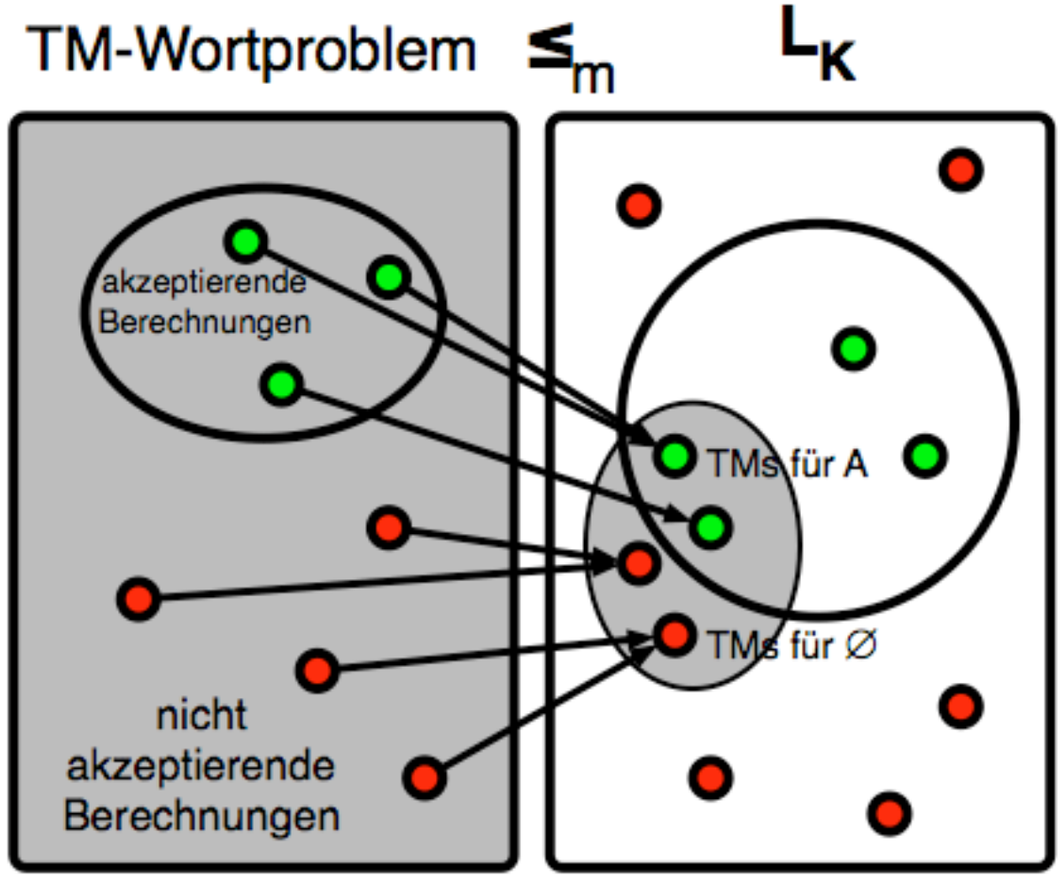
Beweis des Satzes von Rice - 1. Teil

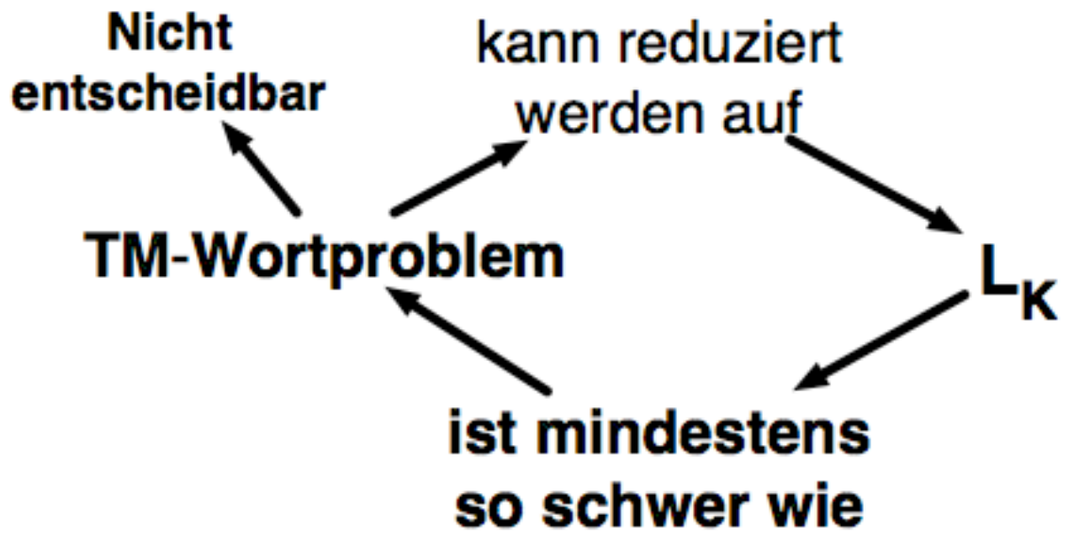
- **Theorem**
  - Sei K ⊆ P(Σ*) eine nicht triviale Klasse von rekursiv aufzählbaren Sprachen, d.h.
    - K ist nicht leer und
    - K beinhaltet nicht alle rekursiv aufzählbare Sprachen
  - Dann ist die folgende Sprache nicht entscheidbar

$L_K = \{\langle M \rangle \mid M \text{ ist eine TM und } L(M) \in K\}$

- **Beweis: 1. Fall: $\emptyset \notin K$**
  - Reduktion: $A_{TM} \leq_m L_K$:
  - Da die Klasse K nicht trivial ist,
  - existiert eine Sprache $A \in K$
  - Sei M_A eine TM mit $L(M_A) = A$
- **Betrachte Reduktionsfunktion F:**
  - F = "Auf Eingabe <M, w>:
    - Konstruiere TM M':
      M' = "Für Eingabe x:
      - Führe M auf Eingabe w aus
      - Falls M das Wort w akzeptiert,
        - führe TM M_A auf x aus,
        - gib Ergebnis $M_A(x)$ aus
      - Ansonsten verwerfe"
    - F gibt <M'> aus
- **Korrektheit der Reduktion:**
  - Falls <M, w> $\in A_{TM}$
    - F(<M, w>) = TM, die A akzeptiert
    - daraus folgt: F(<M, w>) $\in L_K$
  - Falls <M, w> $\notin A_{TM}$
    - F(<M, w>) = TM, die nichts akzeptiert
    - daraus folgt: F(<M, w>) $\notin L_K$

Beweis des Satzes von Rice - 1. Teil





Beweis des Satzes von Rice - 2. Teil

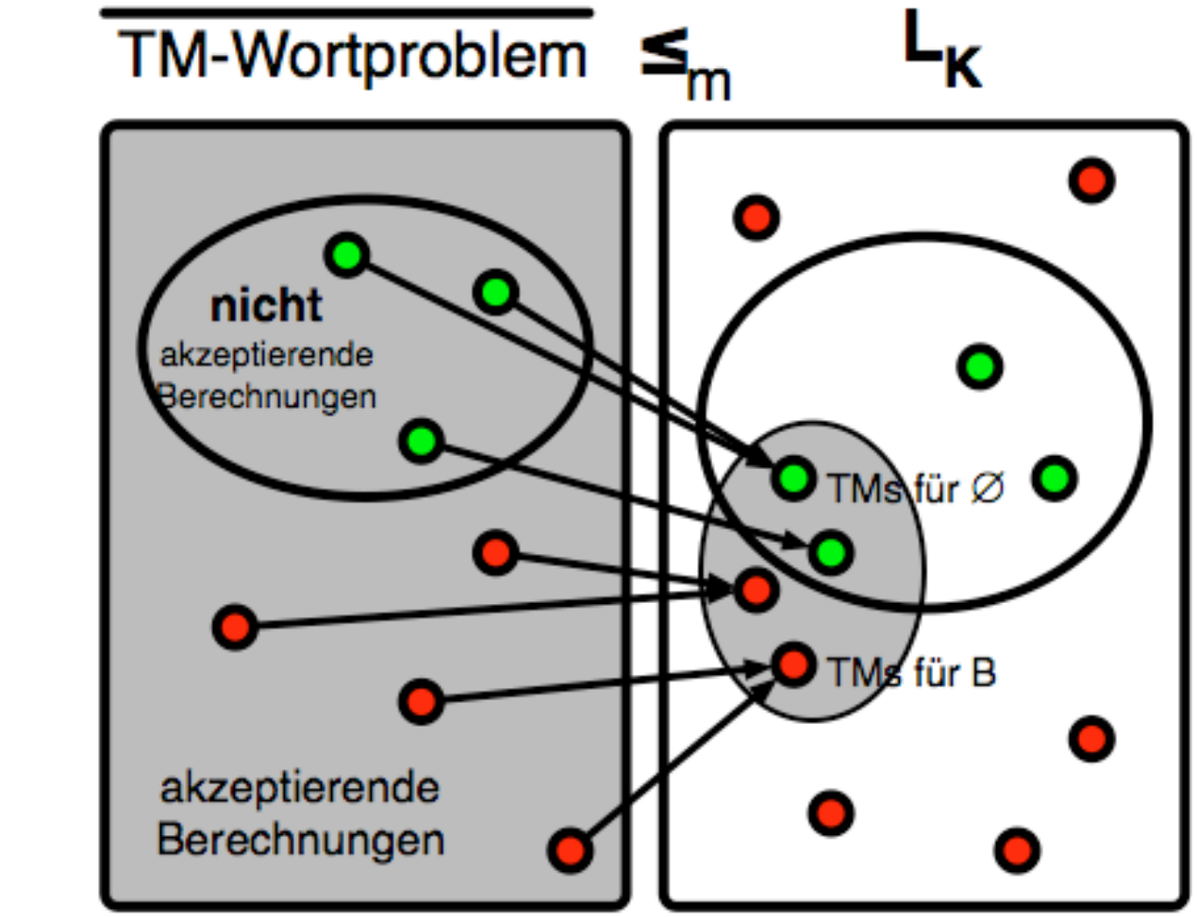
- **Theorem**
  - Sei K ⊆ P(Σ*) eine nicht triviale Klasse von rekursiv aufzählbaren Sprachen, d.h.
    - K ist nicht leer und
    - K beinhaltet nicht alle rekursiv aufzählbare Sprachen
  - Dann ist die folgende Sprache nicht entscheidbar

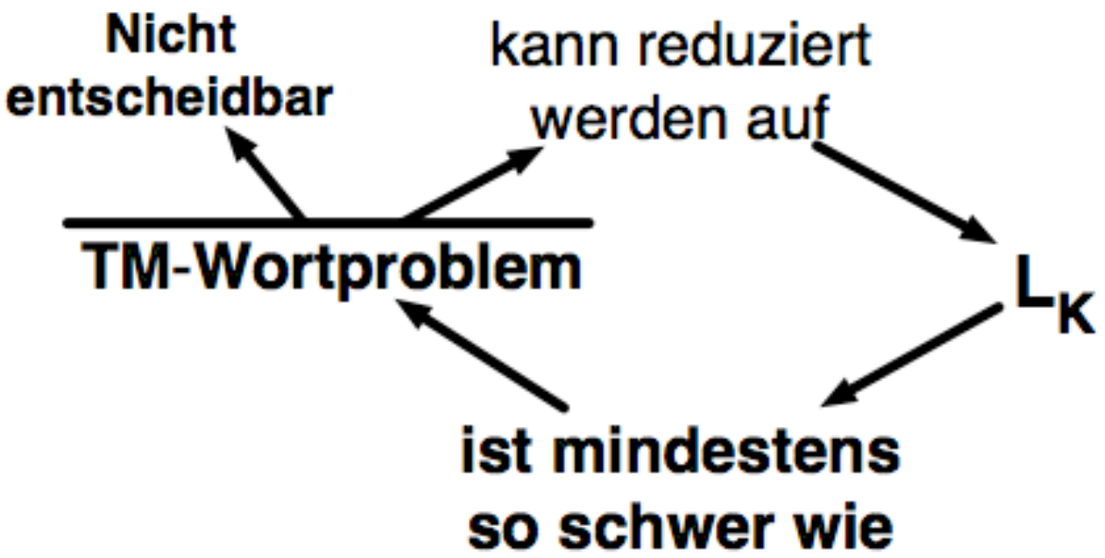
$$L_K = \{\langle M \rangle \mid M \text{ ist eine TM und } L(M) \in K\}$$

- **Beweis: 2. Fall: $\emptyset \in \overline{K}$**
  - Reduktion: $A_{TM} \leq_m L_K$:
  - Da die Klasse K nicht trivial ist,
  - existiert eine Sprache B ∉ K
  - Sei M_B die TM mit $L(M_B) = B$

- **Betrachte Reduktionsfunktion F:**
  - F = "Auf Eingabe <M, w>:
    - Konstruiere TM M':
      M' = "Für jede Eingabe x:
      - Führe M auf Eingabe w aus
      - Falls M das Wort w akzeptiert,
        - führe TM M_B auf x aus,
        - gib Ergebnis $M_B(x)$ aus
      - Ansonsten verwerfe "
    - F gibt <M'> aus"
- **Korrektheit der Reduktion:**
  - Falls <M, w> ∈ A_{TM}
    - F(<M, w>) = TM, die B akzeptiert
    - daraus folgt: F(<M, w>) ∉ L_K
  - Falls <M, w> ∉ A_{TM}
    - F(<M, w>) = TM, die nichts akzeptiert
  - daraus folgt: F(<M, w>) ∈ L_K

Beweis des Satzes von Rice - 2. Teil





Berechenbarkeitstheorie

Turing-Reduktion

8

Turing-Reduktionen: Vorüberlegung

- **Die Abbildungsreduktion ist ein "schwacher" Reduktionsbegriff**

- **Gedankenexperiment:**
  - Sei B eine entscheidbare Sprache
  - Dann gibt es ein haltendes Programm M, das B entscheidet.
  - Wir benutzen M nun als Unterprogramm
  - Wenn ein Programm M' jetzt ein anderes Problem A entscheiden will,
    - kann es M beliebig häufig als Unterprogramm verwenden
      - weil M immer hält
  - Wenn das Programm M' auf jeder Ausgabe von M immer hält und eine Entscheidung trifft,
    - dann löst es A mit Hilfe von B.
  - A ist entscheidbar,
    - weil man M und M' zu einem Programm verschmelzen kann.

- **Diese Art Reduktion ist nicht durch den Begriff der Abbildungsreduktion abgedeckt.**

Orakel-Turing-Maschinen

- **Definition**
  - Ein Orakel für eine Sprache B
    - ist eine externe Einheit, welche für ein gegebenes Wort w entscheidet, ob w ein Element von B ist.
  - Eine **Orakel-Turing-Maschine (OTM)**, ist eine modifizierte Turing-Maschine,
    - welche beliebig häufig ein Orakel befragen kann.
      - Hierzu schreibt die OTM die Anfrage auf ein separates Orakelband
      - und findet nach dem Schreiben des Endsymbols
        - sofort die Antwort auf dem selben Band.
- **Beobachtung:**
  - Orakel müssen nicht notwendigerweise berechenbar sein.
  - Z.B. mit dem Halteproblem als Orakel lässt sich das TM-Wortproblem lösen:
    - Frage Halteproblem-Orakel, ob gegebene TM M auf gegebener Eingabe w hält
    - Falls nein, gib nein aus.
    - Falls ja, führe M auf Eingabe w aus
    - Gib Ergebnis von M(w) aus

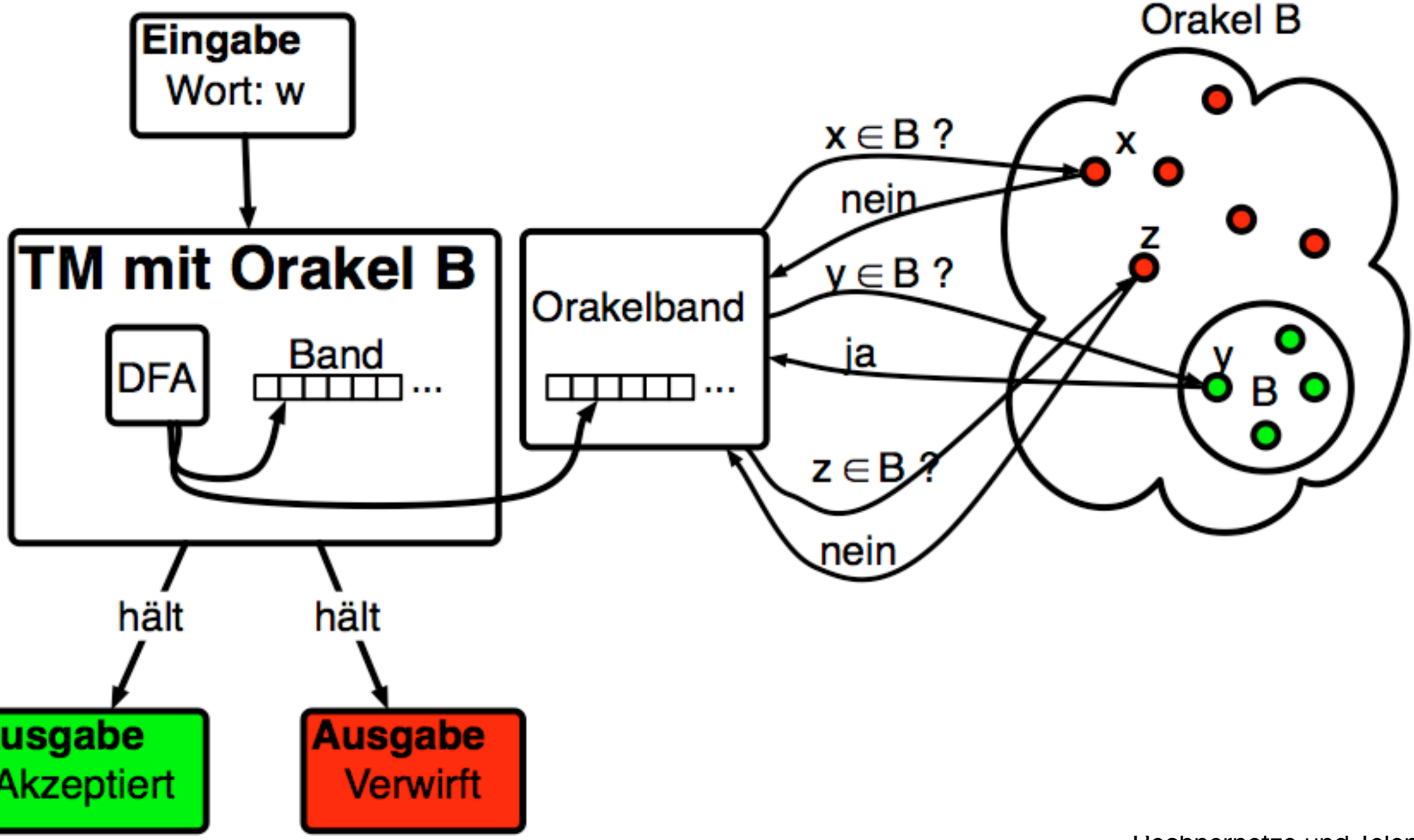
Der Begriff der Turing-Reduktion

- **Definition**
  - Eine Sprache A ist entscheidbar bezüglich Orakel B,
    - falls eine OTM M existiert mit Orakel B,
    - die auf allen Ausgaben von Orakel B hält und
    - die Sprache A entscheidet.
- **Definition**
  - Eine Sprache A ist **Turing-reduzierbar** auf eine Sprache B,

$$\mathbf{A \leq_T B}$$

  - falls A entscheidbar ist bezüglich des Orakels B.
- **Korollar**
  - Aus $A \leq_m B$ folgt $A \leq_T B$.

Die Orakel-Turing-Maschine und ihr Orakel



Der Nutzen der Turing-Reduktion

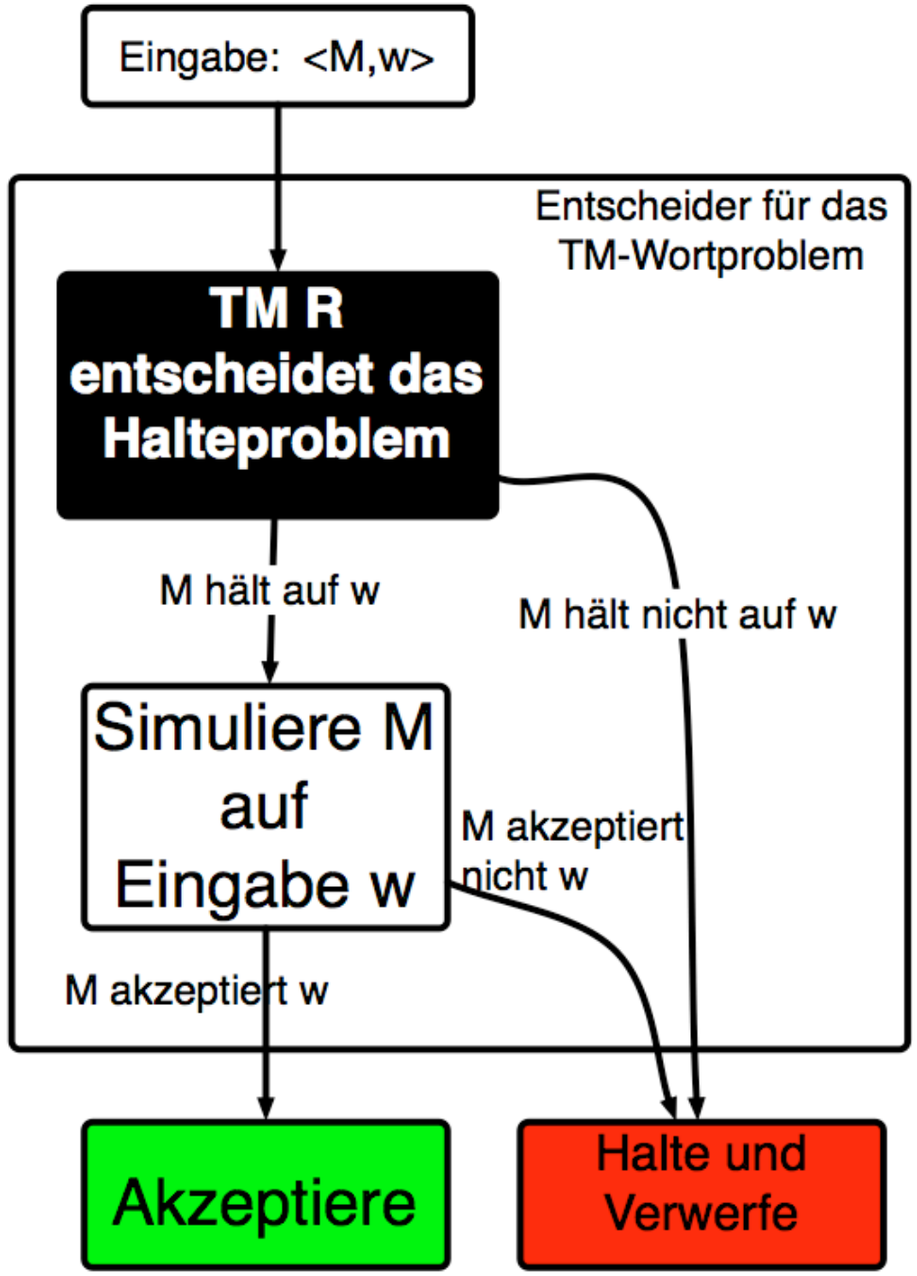
- **Definition**
  - Eine Sprache A ist **Turing-reduzierbar** auf eine Sprache B,

$$\mathbf{A} \leq_T \mathbf{B}$$

  - falls A entscheidbar ist bezüglich des Orakels B.
- **Theorem**
  - Falls $A \leq_T B$ und B ist entscheidbar, dann ist A entscheidbar.
- **Korollar**
  - Falls $A \leq_T B$ und A ist nicht entscheidbar, dann ist B nicht entscheidbar.
- **Achtung:**
  - Solche Sätze gelten nicht für die rekursive Aufzählbarkeit!

Ein dritter Beweis für das Halteproblem

- **Theorem**
  - $A_{TM} \leq_T HALT_{TM}$.
- **Beweis**
  - Konstruiere nun OTM S:
  - S = “Auf Eingabe <M,w>, Kodierung einer TM und Zeichenkette w
    - Frage Halteproblem-Orakel auf Eingabe <M,w>
    - Falls Orakel verwirft, verwirft S
    - Falls Orakel akzeptiert:
      - Simuliere M auf Eingabe w bis M hält
      - Falls M akzeptiert, dann akzeptiert S
      - Falls M verwirft, dann verwirft S”
  - S Turing-reduziert A_{TM} auf $HALT_{TM}$
- **Theorem**
  - Aus $A_{TM} \leq_T HALT_{TM}$ und der Nichtentscheidbarkeit von ATM folgt, dass $HALT_{TM}$ nicht entscheidbar ist.



Berechenbarkeitstheorie

Selbstreferenz

15

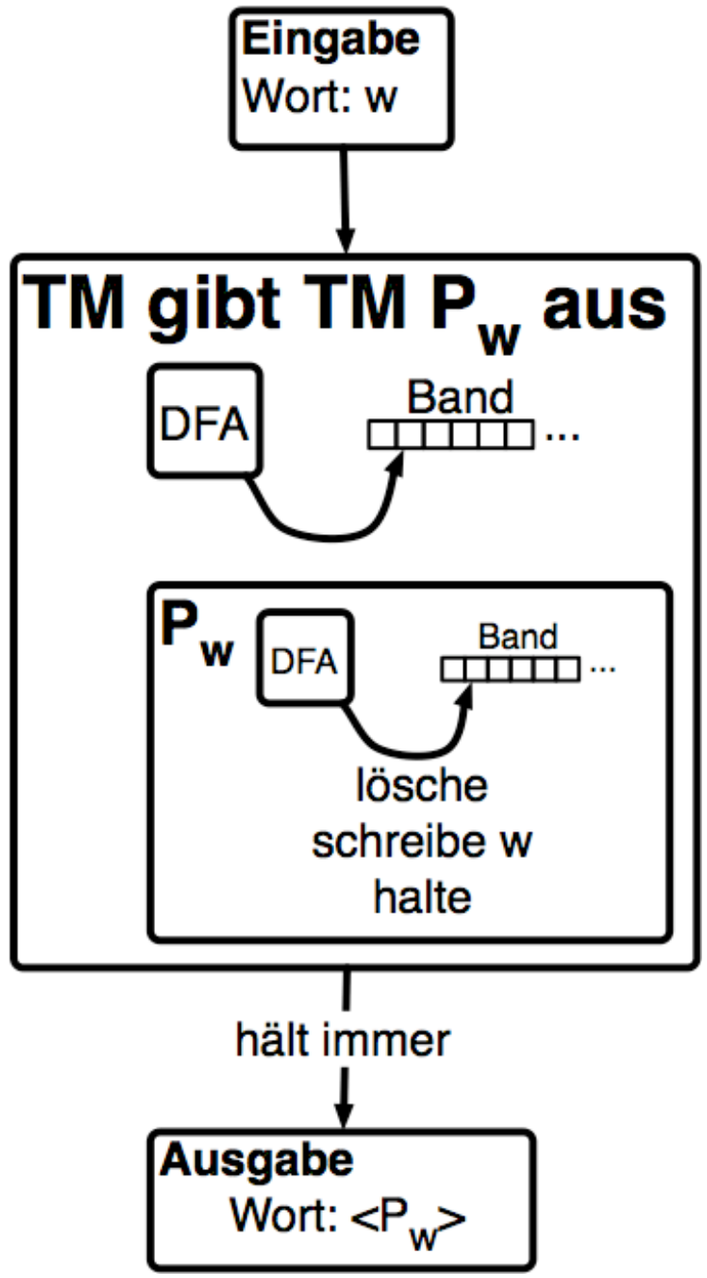
Thesen

1. **Lebewesen sind Maschinen.**
2. **Lebewesen können sich reproduzieren.**
3. **Maschinen können sich nicht reproduzieren.**

- **Wer werden zeigen: These 3 ist falsch!**
- **Wir zeigen:**
  - Maschinen können sich vollständig reproduzieren
  - Maschinen können ihre eigene Beschreibung verwenden
  - Es gibt Maschinen, wenn man ihren Code quadriert, erhält man eine funktionsgleiche Maschine.
  - Es gibt Maschinen, wenn man ihre Beschreibung mit 217 multipliziert, erhält man eine funktionsgleiche Maschinen.
  - Es gibt Maschinen, die in Spiegelschrift notiert die gleiche Funktionalität besitzen.

Selbstreferenz - Vorbereitung

- **Lemma**
  - Es gibt eine berechenbare Funktion q: $\Sigma^* \rightarrow \Sigma^*$,
    - wobei für jedes Wort w
    - q(w) eine Turing-Maschine Pw beschreibt,
    - die w ausgibt und hält.
- **Beweis:**
  - Konstruiere TM Q:
  - Q = “Auf Eingabe w:
    - Konstruiere Pw= “Für jede Eingabe:
      - Lösche die Eingabe
      - Schreibe w auf das Band
      - Halte.”
    - Gib <Pw> aus.”



Die SELBST-Maschine

- **Die SELBST-Maschine gibt eine Beschreibung von sich selbst aus.**
- **Wir konstruieren zwei TM-Teile A und B**
  - A = $P_{<B>}$,
  - B = “Auf Eingabe <M>, wobei M ein Anfang eines TM-Programms ist
    - Berechne q(<M>)
    - Kombiniere das Ergebnis mit <M>, um die TM zu vervollständigen
    - Berechne die Beschreibung der TM und halte”
- **Kombiniere Teile A und B**
- **Resultierende Maschine AB:**
- **A:**
  - Produziert Maschinenbeschreibung von <B> auf das Band
- **B:**
  - Findet Eingabe <B>
  - B berechnet q(<B>) = $P_{<B>}$ = A
  - Aus A wird AB
  - Aus AB wird <AB>
- **Also ist das Ergebnis:**
  - <AB>

Selbst-Reproduktion in anderen Sprachen

http://www.nyx.net/~gthompso/quine.htm

- **C:**

```c
char*f="char*f=%c%s%c;main(){printf
  (f,34,f,34,10);}%c"; main(){printf
  (f,34,f,34,10);}
```

- **Java:**

```java
class S{public static void main
(String[]a){String s="class S{public
static void main(String[]a){String
s=;char c=34;System.out.println
(s.substring(0,52)+c+s+c+s.substring
(52));}}";char
c=34;System.out.println(s.substring
(0,52)+c+s+c+s.substring(52));}}
```

- **Pascal:**

```pascal
const a='const a=';b='begin write
(a,#39,a,#39#59#98#61#39,b,#39#59#10
,b) end.';
begin write
(a,#39,a,#39#59#98#61#39,b,#39#59#10
,b) end.
```

- **Lisp:**

```lisp
((lambda (x)
     (list x (list (quote quote) x)))
 (quote
     (lambda (x)
          (list x (list (quote quote) x)))))
```

- **Brainfuck**

```
>>+++++++>>++>>++++>>+++++++>>+>>++++>>+>>++
+>>+>>+++++>>+>>++>>+>>++++++>>++>>++++>>++++
+++>>+>>+++++>>++>>+>>+>>++++>>+++++++>>+>>++
+++>>+>>+>>+>>++++>>+++++++>>+>>+++++>>++++++
++++++++>>+>>+>>++++>>+++++++>>+>>+++++>>++>>
+>>+>>++++>>+++++++>>+>>+++++>>++++++++++++++
+++++++++++++++>>+>>+>>++++>>+++++++>>+>>++++
+>>++>>+>>+>>+++++>>+>>++++++>>+>>++>>+>>++++
++>>+>>++>>+>>++++++>>+>>++>>+>>++++++>>+>>+
+>>+>>++++++>>+>>++>>+>>++++++>>+>>++>>+>>+++
+++>>++>>++++>>+++++++>>+>>+++++>>+++++++>>
+>>+++++>>+>>+>>+>>++++>>+>>++>>+>>++++++>>
+>>+++++>>+++++++>>+>>++++>>+>>+>>++>>+++++>>
+>>+++>>+>>++++>>+>>++>>+>>++++++>>+>>+++++>>
+++++++++++++++++++>>++>>++>>+++>>++>>+>>++>>
++++>>+++++++>>++>>+++++>>++++++++++>>+>>++>>
++++>>+>>++>>+>>++++++>>++++++>>+>>+>>+++++>>
+>>++++++>>++>>+++++>>+++++++>>++>>++++>>+>>+
+++++[<<]>>[>++++++[-<<++++++++++>>]<<+
+..------------------>[-<.>>+<]>[-<+>]>]<<[-[-[-[-[-[-[>++>]<++
+++++++++++++++++++++++++++>]<++>]<++++++++++++
+++>]<+>]<++>]<<[->.<]<<]
```

Die Programmiersprache Brainfuck

- **Erfunden von Urban Müller 1993 als Erweiterung von P”**
  - P” von Corrado Böhm 1964 ist Brainfuck ohne Ein-/Ausgabe-Operationen “,” und “.”.
- **Daten:**
  - 1 Zeiger
  - 30.000 Zellen mit je einem Byte
- **Programm besteht aus den Befehlen:**
  - > < + - . , [ ]
- **8 Befehle:**
  - > : Erhöhe Zeigerwert um 1. Der Zeiger zeigt im nächsten Schritt auf die nächste rechte Zelle
  - < : Verringere Zeigerwert um 1. Der Zeiger zeigt im nächsten Schritt auf die nächste linke Zelle.
  - + : Erhöhe das Byte an der Zeigerstelle um 1.
  - - : Verringere das Byte an der Zeigerstelle um 1.
  - . : Ausgabe des Byte-Werts an der Zeigerstelle.
  - , : Einlesen eines Byte-Werts mit Speicherung an der Zeigerstelle.
  - [: Überspringe das Programmstück bis direkt nach der passenden Klammer], falls das Byte an der Stelle 0 ist.
  -] : Springe zurück zur passenden Klammerstelle, falls das Byte an der Zeigerstelle nicht 0 ist.
- **Brainfuck ist Turing-vollständig,**
  - wenn man beliebig viele Zellen verwendet (statt 30.000), weil jede TM sich mit Brainfuck darstellen lässt

Selbst-Reproduktion in DEUTSCH

- **Hier nochmal die Konstruktion in der Programmiersprache “DEUTSCH”**

- **P_w entspricht:**
  - “Schreiben Sie “w””.
- **AB entspricht:**
  - Schreiben Sie den folgenden Satz zweimal: Das erste Mal so; das zweite Mal in Anführungszeichen: “Schreiben Sie den folgenden Satz zweimal: Das erste Mal so; das zweite Mal in Anführungszeichen:”

- **A =**
  - Schreiben Sie den folgenden Satz zweimal: Das erste Mal so; das zweite Mal in Anführungszeichen:
- **B =**
  - “Schreiben Sie den folgenden Satz zweimal: Das erste Mal so; das zweite Mal in Anführungszeichen:”

Berechenbarkeitstheorie

Das Rekursions-theorem

Das Rekursionstheorem

- **Rekursionstheorem**
  - Sei T eine TM, welche die Funktion t: $\Sigma^* \rightarrow \Sigma^*$ berechnet.
  - Dann gibt es eine TM R,
    - welche die Funktion r: $\Sigma^* \rightarrow \Sigma^*$ für alle w berechnet:
    - r(w) = t(<R>,w).
- **Interpretation:**
  - Für die Berechnung einer Funktion kann man eine TM finden, welche ihre eigene Kodierung mitverwenden kann.
- **Achtung:**
  - R “weiß” nicht (unbedingt), wie seine eigene Kodierung aussieht
  - R wird dahingehend konstruiert, dass die Kodierung zur Verfügung steht.

Beweis des Rekursionstheorems

- **Rekursionstheorem**
  - Sei T eine TM, welche die Funktion t: $\Sigma^* \rightarrow \Sigma^*$ berechnet.
  - Dann gibt es eine TM R,
    - welche die Funktion r: $\Sigma^* \rightarrow \Sigma^*$ für alle w berechnet:
    - r(w) = t(<R>,w).
- **Beweis**
  - Konstruiere eine TM R aus drei Programmteilen A,B und T (gegeben)
  - $A=P_{<BT>}$ mit der Funktionalität von q',
    - wobei q' eine TM baut,
    - welche die Ausgabe <BT> an eine existierende Bandanschrift w anhängt
  - B = "Lies das Band und wende q' auf den mit einem Sondersymbol getrennten rechten Bandinhalt an
    - ✳ Als Ergebnis wird $<P_{<BT>}>$ = <A> rechts angehängt
    - Kombiniere A, B, und T zu einer einzigen TM: <ABT> = <R>
    - Kodiere diese Beschreibung zusammen mit w zu <R,w> auf das Band
    - Übergebe Kontrolle an T"

Anwendungen des Rekursionstheorems

- **Computer-Viren**
  - sind reproduzierende Programme
  - welche sich ohne Einfluss eines Benutzers selbst kopieren und ausbreiten.

- **Computer-Viren “leben” das Rekursionstheorem**
  - sie kennen ihre eigene Beschreibung
  - sie sind in der Lage Kopien von sich selbst zu schreiben

Berechenbarkeitstheorie

Beschreibungs-komplexität

26

Information und Zufall

- **Welche Sequenzen sind zufällig?**
- **Welche haben viel oder wenig Information?**
  - 00000000000000000000000000000000000000000000000000
  - 01010101010101010101010101010101010101010101010101
  - 10101100111101010101101010110101011010101011110011010 1
  - 01101010010110001011110001110101011100001011101100010
  - 10001101001101011010010101010110101000000000000000000

Information und Zufall

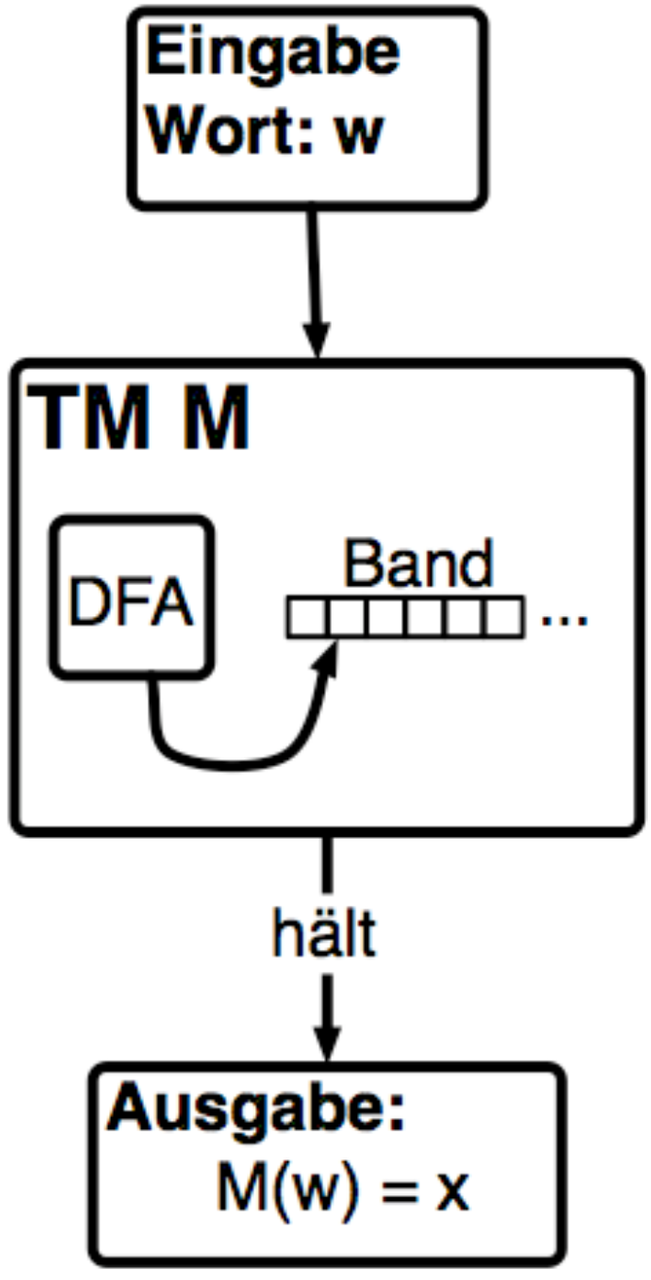
- **Bessere Beschreibungen:**
  - 000000000000000000000000000000000000000000000000
    = 0^{54}
  - 010101010101010101010101010101010101010101010101
    = $(01)^{27}$
  - 101011001111010101011010101101010101111001101 01
    = wwrev, für 10101100111101010101101010
  - 100011010011010110100101010101101010000000000000000000
    = 1000w0ww$(01)^5$10101 0^{12}, für w=11010
  - 011010100101100010111100011101010111000010111101100010
    = ????
- **Gibt es nichtkomprimierbare Binärfolgen?**

Wie wir <M,w> kodieren

- **Mit Hilfe eines größeren Alphabets kann man die Wortlänge weiter komprimieren.**
  - z.B. 137913 = 100001101010111001
- **Um ein festes Maß der Beschreibungskomplexität zu finden, kodieren wir Turing-Maschinen und Eingaben binär wie folgt:**
  - Sei 10110110...10 die Binärkodierung der TM M
  - Sei 01101011...010 das Wort w ∈ {0,1}*
- **Dann ist**
  - <M,w> =
    11 00 11 11 00 11 11 00 ... 11 00
    01
    01101011...010
- **Jedes Bit von <M> wird verdoppelt**
  - Aus 1 wird 11
  - Aus 0 wird 00
- **Dann das Trennsymbol 01 eingefügt**
- **Dann wird w angehängt**

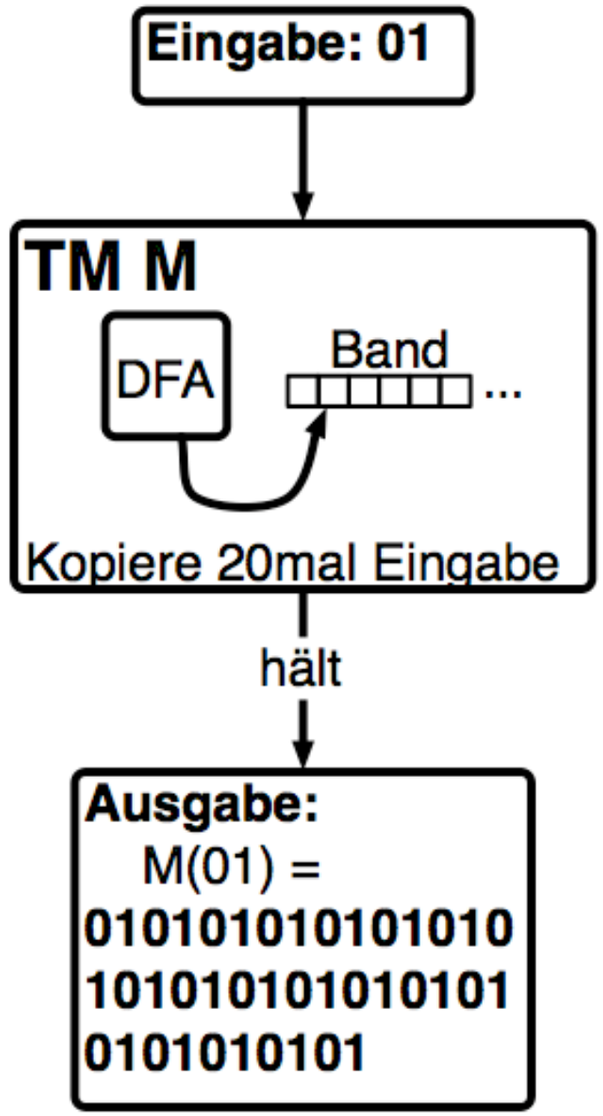
Beschreibungskomplexität
Definition

- **Definition**
  - Sei x eine binäre Zeichenkette
  - Die minimale Beschreibung d(x) von x ist
  - die kürzeste binäre Zeichenkette <M,w>,
    - wobei M eine TM ist und
    - M auf Eingabe w hält und x auf das Band schreibt.
  - Die **Beschreibungskomplexität** (Kolmogorov-Komplexität oder Kolmogorov-Chaitin-Komplexität) ist definiert als
    - $K(x) = |d(x)|$



Beschreibungskomplexität
Beispiel

- **Definition**
  - Sei x eine binäre Zeichenkette
  - Die minimale Beschreibung d(x) von x ist
  - die kürzeste binäre Zeichenkette <M,w>,
    - wobei M eine TM ist und
    - M auf Eingabe w hält und x auf das Band schreibt.
- Die **Beschreibungskomplexität** (Kolmogorov-Komplexität oder Kolmogorov-Chaitin-Komplexität) ist definiert als
  - K(x) = |d(x)|



w = 01

<M> = 1100101010110

<M,w> =
111100001100110
011001111000101

x =
0101010101010101010101
0101010101010101010101

Aufwärmübungen

- **Theorem**
  - Die Beschreibungskomplexität einer Zeichenkette ist höchstens um eine Konstante größer als die Länge der Zeichenkette, d.h.
  - $\exists c \in \mathbf{N}: \forall x \in \{0,1\}^*: K(x) \leq |x| + c$
- **Beweis**
  - Betrachte TM M =
    - “Auf Eingabe w:
      - kopiere w auf das Ausgabeband
      - halte”
  - Dann ist <M,x> eine Beschreibung von x.
    - |<M,x>| = 2k + 2 + |x|
    - Nun ist 2k+2 eine Konstante
      - die wir c nennen.
  - Die kürzeste Beschreibung von x hat also höchstens die Länge |x|+c.

Die Beschreibungs-Komplexität ist nicht berechenbar

- **Theorem**
  - Die Beschreibungs-Komplexität ist nicht berechenbar.
- **Zum Beweis benötigen wir folgendes Lemma:**
- **Lemma**
  - Angenommen, die Beschreibungs-Komplexität ist berechenbar. Dann gibt es eine TM M, die auf Eingabe n ein Wort x mit Beschreibungs-komplexität n ausgibt, wenn so ein Wort existiert.
- **Beweis**
  - Betrachte folgende TM B
  - B = “Auf Eingabe n:
    - Für t=1,2,3,...:
      - Für jede TM M und jedes Wort w mit |<M,w>| = n:
        - Simuliere M auf Eingabe w für t Schritte.
        - Falls M hält innerhalb von t Schritten hält,
        - Sei x die Ausgabe
        - Berechne K(x)
      - Falls K(x) = n, halte und gib x aus”

Die Beschreibungs-Komplexität ist nicht berechenbar

- **Theorem**
  - Die Beschreibungs-Komplexität ist nicht berechenbar.
- **Beweis**
  - Angenommen, K(x) ist berechenbar.
  - Lemma (der letzten Seite) sagt:
    - Dann gibt es eine TM M, die auf Eingabe n ein Wort x mit Beschreibungskomplexität n ausgibt.
  - Betrachte folgende TM R,
    - R = “Auf Eingabe w,
      - Schreibe 2|w| auf das Band.
      - Simuliere M auf n= 2|w|
      - Gib x aus.”
  - Da M x ausgibt, ist die Beschreibungskomplexität von K(x)=2|w|
  - Mit Hilfe von R kann man aber x auch so beschreiben:
    - |<R,w>| = |w| + 2 + 2<R>
  - Falls |w| > 2+2|<R>| ist aber |<R,w>| < K(x)
    - Es gibt dann eine kürzere Beschreibung als die kürzeste Beschreibung
    - Ein Widerspruch.

Eigenschaften der Beschreibungskomplexität

- **Theorem**
  - $\exists c \in \mathbf{N}: \forall x,y \in \{0,1\}^*$: $K(xy) \leq 2K(x) + K(y) + c$
- **Beweis:**
  - Betrachte minimale Beschreibung für x: <M,w>
  - und minimale Beschreibung für y: <M',w'>
  - Ersetze in <M,w> jede 0 durch 00 und jede 1 durch 11
    - Sei <<M,w>> das Ergebnis
  - Betrachte z = <<M,w,>> 01 <M',w'> und die TM R:
  - R = "Auf Eingabe z = <<M,w,>>01<M',w'>
    - Simuliere M auf Eingabe w und
    - Simuliere M' auf Eingabe w'
    - Gibt konkateniertes Ergebnis aus"

- Die Komplexität dieser Beschreibung <R,z> ist:

  $2|\langle R\rangle| + 2 + 2\,K(x) + 2 + K(y)$

  $\leq 2K(x)+K(y) + c$

  für $c = 2|\langle R\rangle| + 4$

Eigenschaften der Beschreibungskomplexität

- **Theorem**
  - $\exists c \in \mathbf{N}: \forall x,y \in \{0,1\}^*$:
    - $K(xy) \leq 2K(x) + K(y) + c$
- **kann verbessert werden zu:**
- **Theorem**
  - $\exists c \in \mathbf{N}: \forall x,y \in \{0,1\}^*$:
    - $K(xy) \leq K(x) + K(y) + 2 \log |x| + c$
- Es gilt aber **nicht:**
  - $K(xy) \leq K(x) + K(y) + c$
- **Warum?**

Universalität der Beschreibungskomplexität

- **Definition**
  - Sei $K_p(x)$ die Beschreibungskomplexität bezüglich eines Maschinenmodells p, welches jede Berechnung einer Turing-Maschine berechnen kann.
- **Theorem**
  - Für jedes Maschinenmodell p gilt:
    $\exists c \in \mathbf{N}: \forall x \in \{0,1\}^*: K_p(x) \leq K(x) + c$
- **Beweis**
  - Betrachte TM U, die auf Eingabe von TM M und Wort w kodiert als <M,w> das Ergebnis M(w) berechnet
  - Sei V die Maschine im Maschinenmodell p, die U berechnet.
  - Sei <M,w> die minimale Beschreibung von x
  - Dann ist <V,<M,w>> auch eine Beschreibung von x aber in p.
  - Es gilt:
    - $|\langle V,\langle M,w\rangle\rangle|$

      $= 2|\langle V\rangle| + 2 + K(x)$

      $\leq K(x) + c$

    - für $c = 2|\langle V\rangle| + 2$

Komprimierbarkeit

- **Definition**
  - Sei x ein Wort.
  - x ist **c-komprimierbar**, falls
    - $K(x) \leq |x| - c$
  - Falls x nicht c-komprimierbar ist, wird x als **c-unkomprimierbar** bezeichnet
  - Falls x nicht 1-komprimierbar ist, wird x als **unkomprimierbar** bezeichnet.
- **Theorem**
  - Es gibt unkomprimierbare Wörter.
- **Beweis**
  - Die Anzahl der Wörter der Länge = n ist 2^n
  - Die Anzahl der Wörter mit Beschreibungskomplexität $\leq$ n-1 ist
    - $1 + 2 + 4 + 8 + \ldots + 2^{n-1} = 2^n - 1$
  - Damit muss es mindestens ein Wort geben, das unkomprimierbar ist.

Die Anzahl unkomprimierbarer Wörter

- **Theorem**
  - Es gibt mindestens 2^n-2^{n-c+1}+1 viele **c-unkomprimierbare Wörter**.
- **Beweis**
  - Die Anzahl der Wörter der Länge = n ist 2^n
  - Die Anzahl der Wörter mit Beschreibungskomplexität $\leq$ n-c ist
    - $1 + 2 + 4 + 8 + ... + 2^{n-c} = 2^{n-c+1}-1$
  - Damit muss es mindestens 2^n-2^{n-c+1} +1 Wörter geben, für die kein Code mit Beschreibungskomplexität $\leq$ n-c zur Verfügung steht.
- **Damit gilt:**
  - Die Wahrscheinlichkeit, dass ein Wort c-komprimierbar ist,
    - ist höchstens $1/2^{c-1}$.
  - Ein zufälliges Wort ist also mit großer Wahrscheinlichkeit c-unkomprimier-bar
- **Zufall und Beschreibbarkeit:**
  - Die Unkomprimierbarkeit ist typisch für zufällige Worte.
  - Während alle maschinell erzeugten Wort kleine Beschreibungs-komplexität haben.

Ende