

Grundbegriffe der Informatik

Einheit 16: Turingmaschinen

Thomas Worsch

Universität Karlsruhe, Fakultät für Informatik

Wintersemester 2008/2009

Eine technische Vorbemerkung

Turingmaschinen

Berechnungen

Eingaben für Turingmaschinen

Ergebnisse von Turingmaschinen

Berechnungskomplexität

Komplexitätsmaße

Komplexitätsklassen

Schwere Probleme

Unentscheidbare Probleme

Codierungen von Turingmaschinen

Das Halteproblem

Die Busy-Beaver-Funktion

Eine technische Vorbemerkung

Turingmaschinen

- Berechnungen

- Eingaben für Turingmaschinen

- Ergebnisse von Turingmaschinen

Berechnungskomplexität

- Komplexitätsmaße

- Komplexitätsklassen

Schwere Probleme

Unentscheidbare Probleme

- Codierungen von Turingmaschinen

- Das Halteproblem

- Die Busy-Beaver-Funktion

- ▶ partielle Funktion von A nach B
- ▶ rechtseindeutige Relation $f \subseteq A \times B$
- ▶ d. h.: für jedes a gibt es **höchstens ein** $b \in B$ mit $(a, b) \in f$
- ▶ ggf. wieder $b = f(a)$ geschrieben
- ▶ andernfalls ist „ $f(a)$ undefiniert“
- ▶ Notation $f : A \dashrightarrow B$ um anzudeuten, dass f partiell ist
- ▶ beachte: totale Funktionen sind spezielle partielle Funktionen

Eine technische Vorbemerkung

Turingmaschinen

Berechnungen

Eingaben für Turingmaschinen

Ergebnisse von Turingmaschinen

Berechnungskomplexität

Komplexitätsmaße

Komplexitätsklassen

Schwere Probleme

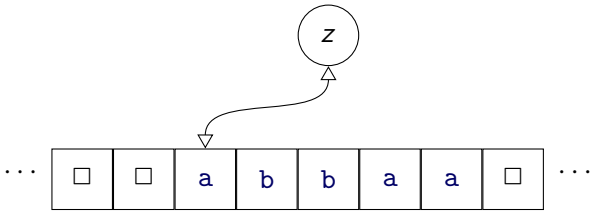
Unentscheidbare Probleme

Codierungen von Turingmaschinen

Das Halteproblem

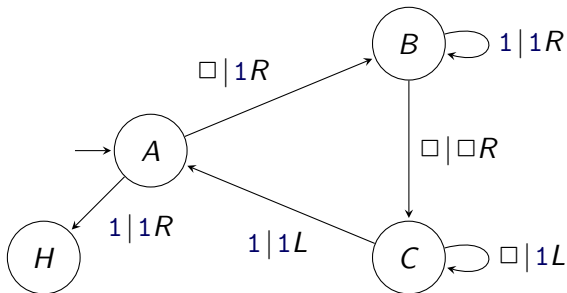
Die Busy-Beaver-Funktion

- ▶ eingeführt von Alan Turing (1912 – 1954)
- ▶ bahnbrechende Arbeit:
„*On computable numbers, with an application to the Entscheidungsproblem*“
Proceedings of the London Mathematical Society **42**, 1936,
S. 230–265.
- ▶ Motivation: *extrem* starke Vereinfachung dessen, was ein Mensch macht, wenn er rechnet
- ▶ sehr weit verbreitetes Modell zur Untersuchung von Fragestellungen, die mit „Berechnungen“ zu tun haben
- ▶ viele Varianten
- ▶ wir betrachten nur den einfachsten Fall

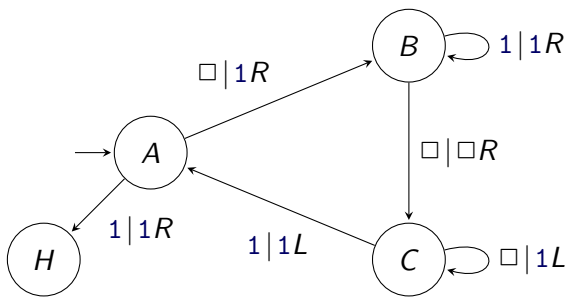


$T = (Z, z_0, X, f, g, m)$ festgelegt vorstellen durch

- ▶ eine **Zustandsmenge** Z
- ▶ einen Anfangszustand $z_0 \in Z$
- ▶ ein **Bandalphabet** X
- ▶ eine partielle **Zustandsüberföhrungsfunktion**
 $f : Z \times X \dashrightarrow Z$
- ▶ eine partielle **Ausgabefunktion**
 $g : Z \times X \dashrightarrow X$ und
- ▶ eine partielle **Bewegungsfunktion**
 $m : Z \times X \dashrightarrow \{-1, 0, 1\}$
- ▶ f, g, m für die gleichen Paare
 $(z, x) \in Z \times X$ definiert bzw. nicht definiert



	A	B	C	H
□	1, R, B	□, R, C	1, L, C	
1	1, R, H	1, R, B	1, L, A	



	A	B	C	H
□	B, 1, R	C, □, R	C, 1, L	
1	H, 1, R	B, 1, R	A, 1, L	

- ▶ „Gesamtzustand“ einer Turingmaschine
- ▶ $c = (z, b, p) \in Z \times X^{\mathbb{Z}} \times \mathbb{Z}$
 - ▶ aktueller Zustand $z \in Z$ der Steuereinheit,
 - ▶ aktuelle Beschriftung des gesamten Bandes:
totale Abbildung $b : \mathbb{Z} \rightarrow X$
 - ▶ aktuelle Position $p \in \mathbb{Z}$ des Kopfes
- ▶ \mathcal{C}_T : Menge aller Konfigurationen von T

- ▶ Bandbeschriftung: ein „potenziell unendliches Gebilde“
- ▶ In weiten Teilen der Informatik interessieren
 - ▶ endliche Berechnungen, die
 - ▶ aus endlichen Eingaben
 - ▶ endliche Ausgaben

berechnen.

- ▶ „Fast“ das ganze Band ist immer „leer“:
 - ▶ Bandalphabet enthält das sogenannte **Blanksymbol**
 - ▶ $\square \in X$ geschrieben
 - ▶ hier: Bei allen vorkommenden Bandbeschriftungen sind nur endlich viele Felder nicht mit \square beschriftet.

Eine technische Vorbemerkung

Turingmaschinen

Berechnungen

Eingaben für Turingmaschinen

Ergebnisse von Turingmaschinen

Berechnungskomplexität

Komplexitätsmaße

Komplexitätsklassen

Schwere Probleme

Unentscheidbare Probleme

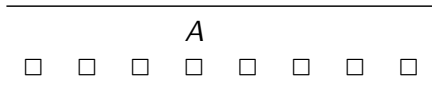
Codierungen von Turingmaschinen

Das Halteproblem

Die Busy-Beaver-Funktion

- ▶ Sei $c = (z, b, p)$ die aktuelle Konfiguration einer TM T .
- ▶ Wenn für das Paar $(z, b(p))$ die Funktionen f , g und m definiert sind,
- ▶ dann kann die TM **einen Schritt machen**.
- ▶ Nachfolgekonfiguration $c' = (z', b', p')$ ist wie folgt definiert:
 - ▶ $z' = f(z, b(p))$
 - ▶ $\forall i \in \mathbb{Z} : b'(i) = \begin{cases} b(i) & \text{falls } i \neq p \\ g(z, b(p)) & \text{falls } i = p \end{cases}$
 - ▶ $p' = p + m(z, b(p))$
- ▶ schreiben $c' = \Delta_1(c)$, also
- ▶ $\Delta_1 : \mathcal{C}_T \dashrightarrow \mathcal{C}_T$

	<i>A</i>	<i>B</i>	<i>C</i>	<i>H</i>
□	1, <i>R</i> , <i>B</i>	□, <i>R</i> , <i>C</i>	1, <i>L</i> , <i>C</i>	
1	1, <i>R</i> , <i>H</i>	1, <i>R</i> , <i>B</i>	1, <i>L</i> , <i>A</i>	



	<i>A</i>	<i>B</i>	<i>C</i>	<i>H</i>
□	1, <i>R</i> , <i>B</i>	□, <i>R</i> , <i>C</i>	1, <i>L</i> , <i>C</i>	
1	1, <i>R</i> , <i>H</i>	1, <i>R</i> , <i>B</i>	1, <i>L</i> , <i>A</i>	



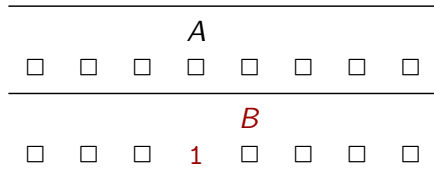
	<i>A</i>	<i>B</i>	<i>C</i>	<i>H</i>
□	1, <i>R</i> , <i>B</i>	□, <i>R</i> , <i>C</i>	1, <i>L</i> , <i>C</i>	
1	1, <i>R</i> , <i>H</i>	1, <i>R</i> , <i>B</i>	1, <i>L</i> , <i>A</i>	



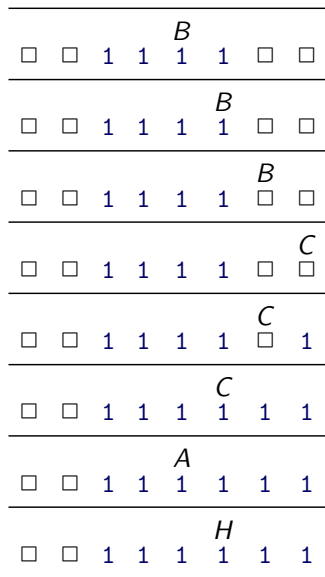
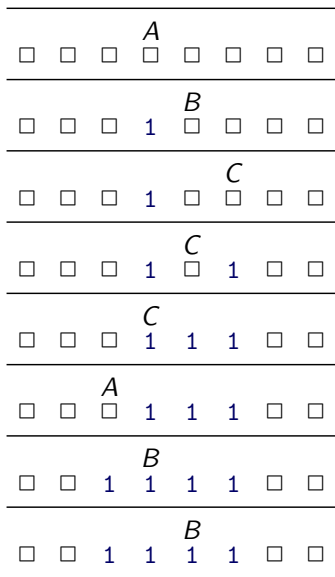
	<i>A</i>	<i>B</i>	<i>C</i>	<i>H</i>
□	1, <i>R</i> , <i>B</i>	□, <i>R</i> , <i>C</i>	1, <i>L</i> , <i>C</i>	
1	1, <i>R</i> , <i>H</i>	1, <i>R</i> , <i>B</i>	1, <i>L</i> , <i>A</i>	



	<i>A</i>	<i>B</i>	<i>C</i>	<i>H</i>
□	1, <i>R</i> , <i>B</i>	□, <i>R</i> , <i>C</i>	1, <i>L</i> , <i>C</i>	
1	1, <i>R</i> , <i>H</i>	1, <i>R</i> , <i>B</i>	1, <i>L</i> , <i>A</i>	



Längere Beispielberechnung



- ▶ c ist **Endkonfiguration**, falls $\Delta_1(c)$ nicht definiert ist.
- ▶ **endliche Berechnung**:
 - ▶ endliche Folge von Konfigurationen $(c_0, c_1, c_2, \dots, c_t)$, wobei für alle $0 < i \leq t$ gilt $c_i = \Delta_1(c_{i-1})$
- ▶ **haltende Berechnung**:
 - ▶ endliche Berechnung,
 - ▶ deren letzte Konfiguration eine Endkonfiguration ist
- ▶ **unendliche Berechnung**:
 - ▶ unendliche Folge von Konfigurationen (c_0, c_1, c_2, \dots) , wobei für alle $0 < i$ gilt $c_i = \Delta_1(c_{i-1})$
 - ▶ heißt auch **nicht haltend**
 - ▶ simples Beispiel
 - ▶ $f(z, x) = z$,
 - ▶ $g(z, x) = x$ und
 - ▶ $m(z, x) = 1$
 - ▶ Kann man so etwas nicht einfach „wegmachen“? ...

- ▶ analog zu Δ_1 allgemein für $t \in \mathbb{N}_0$ Abbildung $\Delta_t : \mathcal{C}_T \dashrightarrow \mathcal{C}_T$

$$\Delta_0 = \text{Id}$$

$$\Delta_{t+1} = \Delta_1 \circ \Delta_t$$

- ▶ Zu jeder Konfiguration c gibt es genau eine Berechnung, die mit c startet.
- ▶ Wenn diese Berechnung hält, dann ist der Zeitpunkt zu dem das geschieht natürlich auch eindeutig.
- ▶ Wir schreiben Δ_* für die partielle Abbildung $\mathcal{C}_T \dashrightarrow \mathcal{C}_T$ mit

$$\Delta_*(c) = \begin{cases} \Delta_t(c) & \text{falls } \Delta_t(c) \text{ definiert und} \\ & \text{Endkonfiguration ist} \\ \text{undefiniert} & \text{falls } \Delta_t(c) \text{ für alle } t \in \mathbb{N}_0 \text{ definiert ist} \end{cases}$$

Eine technische Vorbemerkung

Turingmaschinen

Berechnungen

Eingaben für Turingmaschinen

Ergebnisse von Turingmaschinen

Berechnungskomplexität

Komplexitätsmaße

Komplexitätsklassen

Schwere Probleme

Unentscheidbare Probleme

Codierungen von Turingmaschinen

Das Halteproblem

Die Busy-Beaver-Funktion

analog zu endlichen Automaten

- ▶ Berechnung von Funktionen
- ▶ Erkennung formaler Sprachen
 - ▶ Turingmaschinenakzeptoren
 - ▶ Entscheidungsprobleme

- ▶ **Eingabealphabet** $A \subset X \setminus \{\square\}$ spezifiziert ist.
 - ▶ Das Blanksymbol also nie zum Eingabealphabet.
- ▶ **Anfangskonfiguration** $c_0(w) = (z, b, p)$ für Eingabe $w \in A^*$
 - ▶ $z = z_0$
 - ▶ $b = b_w : \mathbb{Z} \rightarrow X$
$$b_w(i) = \begin{cases} \square & \text{falls } i < 0 \vee i \geq |w| \\ w(i) & \text{falls } 0 \leq i \wedge i < |w| \end{cases}$$
 - ▶ $p = 0$
 - ▶ Kopf auf dem ersten Eingabesymbol (falls $w \neq \varepsilon$)
- ▶ Anfangskonfiguration bei Eingabe evtl. mehrerer Zahlen
 - ▶ geeignet harmlos, z. B.
 - ▶ $\square\square\square 1011 \square\square$ oder
 - ▶ $\square\square\square (1011, 101) \square\square$ oder
 - ▶ ...

Eine technische Vorbemerkung

Turingmaschinen

Berechnungen

Eingaben für Turingmaschinen

Ergebnisse von Turingmaschinen

Berechnungskomplexität

Komplexitätsmaße

Komplexitätsklassen

Schwere Probleme

Unentscheidbare Probleme

Codierungen von Turingmaschinen

Das Halteproblem

Die Busy-Beaver-Funktion

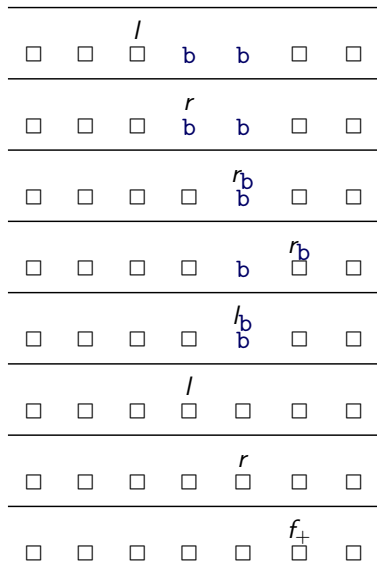
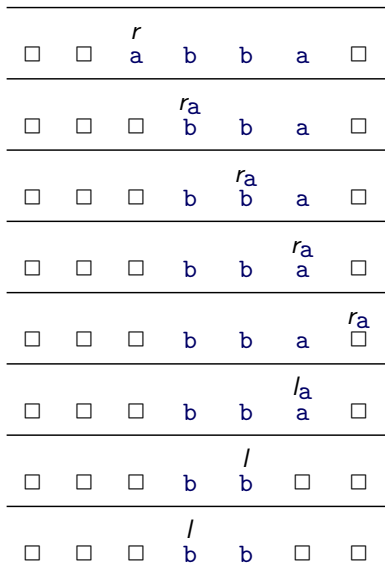
analog zu endlichen Automaten

- ▶ Berechnung von Funktionen: ein Ausgabewort
- ▶ Erkennung formaler Sprachen: ein Bit akzeptiert/abgelehnt

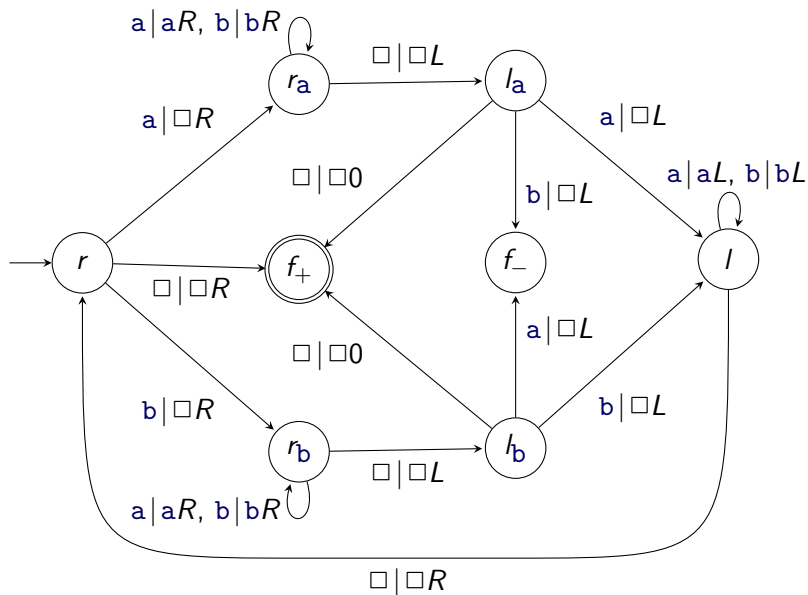
- ▶ Teilmenge $F \subset Z$ *akzeptierender Zustände*
- ▶ T akzeptiert w , wenn
 - ▶ T für Eingabe w hält und
 - ▶ der Zustand der Endkonfiguration $\Delta_*(c_0(w))$ akzeptierend ist.
- ▶ $L(T)$: Menge der akzeptierten Wörter

- ▶ zwei Möglichkeiten, wenn w von T *nicht* akzeptiert wird:
 1. T hält für Eingabe w , aber Endzustand ist nicht akzeptierend.
 2. T hält für Eingabe w nicht.
- ▶ Was weiß man?
 1. T ist fertig und lehnt die Eingabe ab.
 2. T ist noch nicht fertig.
Ob T irgendwann w noch akzeptiert oder ablehnt, ist unklar.
- ▶ zwei Definitionen
 1. L heißt *aufzählbare Sprache*, wenn es eine Turingmaschine gibt, die L akzeptiert.
 2. L heißt *entscheidbare Sprache*, wenn es eine Turingmaschine gibt, die *immer hält* und L akzeptiert.
- ▶ werden sehen:
Entscheidbarkeit ist eine stärkere Forderung

Palindromerkennung: Beispielberechnung



Palindromerkennung: Beispielturingmaschine



Das sollten Sie mitnehmen:

- ▶ Turingmaschinen
 - ▶ Steuereinheit endlich
 - ▶ Band
 - ▶ unendlich, aber
 - ▶ nur endlich viel nicht leer
- ▶ alles endlich beschreibbar
- ▶ die klassische Formalisierung des Algorithmusbegriffs
- ▶ Berechnungen
 - ▶ haltende
 - ▶ nicht haltende

Das sollten Sie üben:

- ▶ Beispielturingmaschinen konstruieren
- ▶ Beispielturingmaschinen verstehen

Eine technische Vorbemerkung

Turingmaschinen

Berechnungen

Eingaben für Turingmaschinen

Ergebnisse von Turingmaschinen

Berechnungskomplexität

Komplexitätsmaße

Komplexitätsklassen

Schwere Probleme

Unentscheidbare Probleme

Codierungen von Turingmaschinen

Das Halteproblem

Die Busy-Beaver-Funktion

- ▶ In diesem Abschnitt (und im nächsten) Annahme: alle **Turingmaschinen halten für jede Eingabe.**
- ▶ Versprechen: Für die Fragestellungen hier (Komplexitätstheorie) ist das in Ordnung
- ▶ Vorsicht: Für die Fragestellungen in Abschnitt 16.5 (Berechenbarkeit) ist das *nicht* mehr in Ordnung.

Eine technische Vorbemerkung

Turingmaschinen

Berechnungen

Eingaben für Turingmaschinen

Ergebnisse von Turingmaschinen

Berechnungskomplexität

Komplexitätsmaße

Komplexitätsklassen

Schwere Probleme

Unentscheidbare Probleme

Codierungen von Turingmaschinen

Das Halteproblem

Die Busy-Beaver-Funktion

- ▶ Beurteilung des Zeitbedarfs einer Turingmaschine
- ▶ definiere

$$\text{time}_{\mathcal{T}} : A^+ \rightarrow \mathbb{N}_+$$

$$\text{Time}_{\mathcal{T}} : \mathbb{N}_+ \rightarrow \mathbb{N}_+$$

wie folgt:

$$\text{time}_{\mathcal{T}}(w) = \text{dasjenige } t \text{ mit } \Delta_t(c_0(w)) = \Delta_*(c_0(w))$$

$$\text{Time}_{\mathcal{T}}(n) = \max\{\text{time}(w) \mid w \in A^n\}$$

- ▶ $\text{Time}_{\mathcal{T}}$ heißt *Zeitkomplexität* der Turingmaschine
- ▶ Zeitbedarf beschrieben
 - ▶ nicht für jede Eingabe einzeln, sondern
 - ▶ in Abhängigkeit von der Länge der Eingabe
 - ▶ der schlimmste Fall
- ▶ Zeitkomplexität einer Turingmaschine *polynomiell*, wenn ein Polynom $p(n)$ gibt mit $\text{Time}_{\mathcal{T}}(n) \in O(p(n))$.

- ▶ Für Eingabe der Länge $n \geq 2$ schlimmstenfalls
 1. erstes und letztes Symbol miteinander vergleichen, und weil übereinstimmend, anschließend
 2. für Teilwort der Länge $n - 2$ ohne Randsymbole wieder ein Palindromtest
- ▶ Zeitbedarf:
 1. $2n - 1$ Schritte
 2. $O(\text{Time}(n - 2))$

insgesamt also

$$\text{Time}(n) \leq 2n - 1 + \text{Time}(n - 2)$$

- ▶ Zeitaufwand für Wörter der Längen 0 und 1 durch Konstante c beschränkt
- ▶ also polynomielle, nämlich quadratische, Zeitkomplexität

$$\text{Time}(n) \in O(n^2)$$

- ▶ Beurteilung des Speicherplatzbedarfs einer Turingmaschine
- ▶ definiere

$$\text{space}_{\mathcal{T}}(w) : A^+ \rightarrow \mathbb{N}_+$$

$$\text{Space}_{\mathcal{T}}(n) : \mathbb{N}_+ \rightarrow \mathbb{N}_+$$

wie folgt

$\text{space}_{\mathcal{T}}(w)$ = die Anzahl der Felder, die während der
Berechnung für Eingabe w benötigt werden

$$\text{Space}_{\mathcal{T}}(n) = \max\{\text{space}(w) \mid w \in A^n\}$$

- ▶ Ein Feld wird „benötigt“, wenn es anfangs ein Eingabesymbol enthält oder einmal vom Kopf der TM besucht wird.
- ▶ $\text{Space}_{\mathcal{T}}$ heißt die *Raumkomplexität* oder *Platzkomplexität* der Turingmaschine

- ▶ benötigte Felder:
 - ▶ n Felder mit den Eingabesymbolen
 - ▶ jeweils ein weiteres Feld links bzw. rechts davon werden besucht
- ▶ polynomieller, nämlich linearer, Platzbedarf

$$\text{Space}(n) = n + 2 \in \Theta(n)$$

- ▶ Wenn T für Eingabe w genau $\text{time}(w)$ Schritte macht,
- ▶ dann kann T höchstens $1 + \text{time}(w)$ Felder besuchen.
- ▶ Folglich immer

$$\text{space}(w) \leq \max(|w|, 1 + \text{time}(w)) .$$

- ▶ Jede Turingmaschine mit polynomieller Laufzeit hat auch nur polynomiellen Platzbedarf.

nun umgekehrt von Raum- zu Zeitkomplexität:

- ▶ Auf k Feldern können $(|X| - 1)^k$ „interessante“ verschiedene Inschriften stehen.
- ▶ Das sind

$$\left(\frac{|X| - 1}{|A|} \right)^k$$

viele je Eingabewort

- ▶ Man kann sehr wohl Turingmaschinen konstruieren, die zwar polynomielle Raumkomplexität, aber exponentielle Zeitkomplexität haben.
- ▶ Eine Turingmaschine mit polynomiellem Platzbedarf kann exponentiellen Zeitbedarf haben.

Eine technische Vorbemerkung

Turingmaschinen

Berechnungen

Eingaben für Turingmaschinen

Ergebnisse von Turingmaschinen

Berechnungskomplexität

Komplexitätsmaße

Komplexitätsklassen

Schwere Probleme

Unentscheidbare Probleme

Codierungen von Turingmaschinen

Das Halteproblem

Die Busy-Beaver-Funktion

- ▶ Eine *Komplexitätsklasse* ist eine Menge von Problemen.
- ▶ hier: nur Entscheidungsprobleme, also formale Sprachen
- ▶ Charakterisierung durch Beschränkung der zur Verfügung stehen Ressourcen, also z. B. **Schranken für Zeitkomplexität** oder/und **Raumkomplexität**
- ▶ Beispiel: alle Entscheidungsprobleme, i. e. formale Sprachen, die von Turingmaschinen entschieden werden können, bei denen gleichzeitig
 - ▶ Zeitkomplexität in $O(n^3)$ und
 - ▶ Raumkomplexität in $O(n^{3/2} \log n)$ ist

wobei n die Größe der Probleminstanz, also die Länge des Eingabewortes ist.

- ▶ **P** ist die Menge aller formaler Sprachen, die von Turingmaschinen entschieden werden können, deren Zeitkomplexität polynomiell ist.
- ▶ **PSPACE** ist die Menge aller formaler Sprachen, die von Turingmaschinen entschieden werden können, deren Raumkomplexität polynomiell ist.

Beispiele

- ▶ „Palindromerkennung“ bzw. die formale Sprache aller Palindrome ist in **P**
- ▶ „Äquivalenz regulärer Ausdrücke“ ist in **PSPACE**

- ▶ polynomielle Laufzeit impliziert polynomiellen Platzbedarf
- ▶ Eine Turingmaschine, die belegt, dass ein Problem in \mathbf{P} ist, belegt gleichzeitig auch, dass das Problem in \mathbf{PSPACE} ist.
- ▶ Also

$$\mathbf{P} \subseteq \mathbf{PSPACE} .$$

- ▶ Und umgekehrt? **Vorsicht!**
 - ▶ Eine Turingmaschine mit polynomielltem Platzbedarf kann exponentiell viele Schritte machen kann.
 - ▶ Solche Turingmaschinen gibt es.
 - ▶ Es könnte aber sein, dass es immer eine andere Turingmaschine gibt, die viel schneller ist.
- ▶ Bei \mathbf{P} und \mathbf{PSPACE} geht es um formale Sprachen, nicht um Turingmaschinen.
- ▶ großes offenes wissenschaftliches Problem:
 $\mathbf{P} = \mathbf{PSPACE}$ oder $\mathbf{P} \neq \mathbf{PSPACE}$?

- ▶ polynomielle Laufzeit impliziert polynomiellen Platzbedarf
- ▶ Eine Turingmaschine, die belegt, dass ein Problem in \mathbf{P} ist, belegt gleichzeitig auch, dass das Problem in \mathbf{PSPACE} ist.
- ▶ Also

$$\mathbf{P} \subseteq \mathbf{PSPACE} .$$

- ▶ Und umgekehrt? **Vorsicht!**
 - ▶ Eine Turingmaschine mit polynomielltem Platzbedarf kann exponentiell viele Schritte machen kann.
 - ▶ Solche Turingmaschinen gibt es.
 - ▶ Es könnte aber sein, dass es immer eine andere Turingmaschine gibt, die viel schneller ist.
- ▶ Bei \mathbf{P} und \mathbf{PSPACE} geht es um formale Sprachen, nicht um Turingmaschinen.
- ▶ großes offenes wissenschaftliches Problem:
 $\mathbf{P} = \mathbf{PSPACE}$ oder $\mathbf{P} \neq \mathbf{PSPACE}$?

- ▶ polynomielle Laufzeit impliziert polynomiellen Platzbedarf
- ▶ Eine Turingmaschine, die belegt, dass ein Problem in P ist, belegt gleichzeitig auch, dass das Problem in $PSPACE$ ist.
- ▶ Also

$$P \subseteq PSPACE .$$

- ▶ Und umgekehrt? **Vorsicht!**
 - ▶ Eine Turingmaschine mit polynomielltem Platzbedarf kann exponentiell viele Schritte machen kann.
 - ▶ Solche Turingmaschinen gibt es.
 - ▶ Es könnte aber sein, dass es immer eine andere Turingmaschine gibt, die viel schneller ist.
- ▶ Bei P und $PSPACE$ geht es um formale Sprachen, nicht um Turingmaschinen.
- ▶ **großes offenes wissenschaftliches Problem:**
 $P = PSPACE$ oder $P \neq PSPACE$?

Das sollten Sie mitnehmen:

- ▶ Zeit und Speicherplatz als wertvolle Ressourcen
- ▶ **Zeitkomplexität** und **Raumkomplexität**
 - ▶ üblicherweise in Abhängigkeit von Eingabegröße der schlimmste Fall
 - ▶ evtl. nur obere Schranke
 - ▶ es gibt auch noch andere Komplexitätsmaße ...
- ▶ **Komplexitätsklassen**
 - ▶ durch **Beschränkung der zur Verfügung stehen Ressourcen**, also
 - ▶ z. B. Schranken für Zeitkomplexität oder/und Raumkomplexität
 - ▶ wichtig (neben anderen wie **NP** ...)
 - ▶ **P**
 - ▶ **PSPACE**

Das sollten Sie üben:

- ▶ Abschätzung der Zeit- und Raumkomplexität von TM

Eine technische Vorbemerkung

Turingmaschinen

Berechnungen

Eingaben für Turingmaschinen

Ergebnisse von Turingmaschinen

Berechnungskomplexität

Komplexitätsmaße

Komplexitätsklassen

Schwere Probleme

Unentscheidbare Probleme

Codierungen von Turingmaschinen

Das Halteproblem

Die Busy-Beaver-Funktion

- ▶ Es seien L_1 und L_2 formale Sprachen über Alphabet A .
- ▶ L_1 ist auf L_2 (*polynomialzeit-*)reduzierbar, wenn
 - ▶ eine in Polynomialzeit berechenbare Funktion $f : A^* \rightarrow A^*$ existiert
 - ▶ mit

$$w \in L_1 \iff f(w) \in L_2 .$$

- ▶ Eine solche Funktion kann auch wieder nur polynomiell viel Platz benötigen.

- ▶ Algorithmus:
 - ▶ gegeben $w \in L_1$
 - ▶ berechne zuerst $f(w)$
 - ▶ überprüfe dann, ob $f(w) \in L_2$ ist
 - ▶ liefere das als Ergebnis
- ▶ Wenn $L_2 \in \mathbf{P}$ ist, dann ist auch $L_1 \in \mathbf{P}$.
- ▶ Wenn $L_2 \in \mathbf{PSPACE}$ ist, dann ist auch $L_1 \in \mathbf{PSPACE}$.

- ▶ Ein Problem C heißt **PSPACE-hart** oder **PSPACE-schwer**, wenn es für jedes Problem $L \in \mathbf{PSPACE}$ eine Polynomialzeitreduktion auf C gibt.
- ▶ Ein Problem heißt **PSPACE-vollständig**, wenn
 - ▶ es **PSPACE-hart** ist und
 - ▶ selbst in **PSPACE** liegt.
- ▶ **PSPACE-vollständige** Probleme gehören also sozusagen „zu den schwersten in **PSPACE**“.
- ▶ Beispiel: die Äquivalenz regulärer Ausdrücke zu entscheiden ist **PSPACE-vollständig**

- ▶ Konsequenz der Reduzierbarkeit:
 - ▶ Wenn man für ein **PSPACE**-vollständiges Problem einen Polynomialzeitalgorithmus findet,
 - ▶ dann kann man *alle* Probleme aus **PSPACE** in Polynomialzeit entscheiden.
 - ▶ also ist dann $\mathbf{PSPACE} \subseteq \mathbf{P}$, also $\mathbf{P} = \mathbf{PSPACE}$.

- ▶ Man kennt für kein einziges **PSPACE**-vollständiges Problem einen Polynomialzeit-Algorithmus,
- ▶ aber man kann auch nicht beweisen, dass keine existieren.

- ▶ Bei **PSPACE** ist noch lange nicht Schluss.
- ▶ Es gibt beweisbar gaaaaanz schwere Probleme,
- ▶ z. B. solche, für deren Lösung jede Turingmaschine Zeitbedarf

$$2^{2^{2^{2^n}}}$$

hat.

- ▶ Solche Probleme
 - ▶ kann man aber prinzipiell lösen.
 - ▶ (Man „will“ nur nicht so lange warten.)

Das sollten Sie mitnehmen:

- ▶ Begriff der **PSPACE**-Vollständigkeit
- ▶ Bedeutung/Stand der Wissenschaft

Das sollten Sie üben:

- ▶ ausnahmsweise mal nichts ...

Eine technische Vorbemerkung

Turingmaschinen

Berechnungen

Eingaben für Turingmaschinen

Ergebnisse von Turingmaschinen

Berechnungskomplexität

Komplexitätsmaße

Komplexitätsklassen

Schwere Probleme

Unentscheidbare Probleme

Codierungen von Turingmaschinen

Das Halteproblem

Die Busy-Beaver-Funktion

Eine technische Vorbemerkung

Turingmaschinen

Berechnungen

Eingaben für Turingmaschinen

Ergebnisse von Turingmaschinen

Berechnungskomplexität

Komplexitätsmaße

Komplexitätsklassen

Schwere Probleme

Unentscheidbare Probleme

Codierungen von Turingmaschinen

Das Halteproblem

Die Busy-Beaver-Funktion

- ▶ Ziel: beschreibe jede Turingmaschine durch ein Wort
- ▶ im folgenden beispielhaft eine Möglichkeit, das zu tun
- ▶ für Beschreibungen Alphabet $A = \{ [,], 0, 1 \}$
 - ▶ es reicht aber auch $A = \{ 0, 1 \}$
 - ▶ oder sogar $A = \{ 1 \}$
- ▶ zu den Wörtern gehören Zahlen
 - ▶ Nummerierung von Turingmaschinen
 - ▶ sogenannte **Gödelisierung** nach Kurt Gödel (1906-1978)
 - ▶ wesentliche Arbeit:
Kurt Gödel (1931): *Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I*,
Monatshefte für Mathematik und Physik, **38**, S. 173-198.
www.springerlink.com/content/p03501kn35215860/

- ▶ Ziel: beschreibe jede Turingmaschine durch ein Wort
- ▶ im folgenden beispielhaft eine Möglichkeit, das zu tun
- ▶ für Beschreibungen Alphabet $A = \{ [,], 0, 1 \}$
 - ▶ es reicht aber auch $A = \{ 0, 1 \}$
 - ▶ oder sogar $A = \{ 1 \}$
- ▶ zu den Wörtern gehören Zahlen
 - ▶ Nummerierung von Turingmaschinen
 - ▶ sogenannte **Gödelisierung** nach Kurt Gödel (1906-1978)
 - ▶ wesentliche Arbeit:
Kurt Gödel (1931): *Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I*,
Monatshefte für Mathematik und Physik, **38**, S. 173-198.
www.springerlink.com/content/p03501kn35215860/

Turingmaschine $T = (Z, z_0, X, \square, f, g, m)$

Codierung der Zustände:

- ▶ Die Zustände von T werden ab 0 durchnummeriert.
- ▶ Der Anfangszustand bekommt Nummer 0.
- ▶ Alle Zustände werden durch gleich lange Binärdarstellungen ihrer Nummern, umgeben von einfachen eckigen Klammern, repräsentiert.
- ▶ Wir schreiben $\text{cod}_Z(z)$ für die Codierung von Zustand z .
- ▶ Beispiele:
 - ▶ $\text{cod}_Z(z_0) = [0000]$
 - ▶ $\text{cod}_Z(z_1) = [0001]$
 - ▶ ...

Turingmaschine $T = (Z, z_0, X, \square, f, g, m)$

Codierung der Symbole:

- ▶ Bandsymbole ab 0 durchnummeriert.
- ▶ Blankensymbol bekommt Nummer 0.
- ▶ Alle Bandsymbole durch gleich lange Binärdarstellungen ihrer Nummern, umgeben von einfachen eckigen Klammern, repräsentiert.
- ▶ Wir schreiben $\text{cod}_X(x)$ für die Codierung von Bandsymbol x .
- ▶ Beispiele:
 - ▶ $\text{cod}_X(\square) = [0000]$
 - ▶ $\text{cod}_X(x_1) = [0001]$
 - ▶ ...

Turingmaschine $T = (Z, z_0, X, \square, f, g, m)$

Codierung der Kopfbewegungen:

- ▶ mögliche Bewegungsrichtungen des Kopfes durch die Wörter [10], [00] und [01] repräsentiert.
- ▶ Wir schreiben $\text{cod}_M(r)$ für die Codierung der Bewegungsrichtung r .
- ▶ also
 - ▶ $\text{cod}_M(-1) = [10]$
 - ▶ $\text{cod}_M(0) = [00]$
 - ▶ $\text{cod}_M(1) = [01]$

Turingmaschine $T = (Z, z_0, X, \square, f, g, m)$

Codierung einzelner Funktionswerte von f , g und m

- ▶ wenn für Argumentpaar (z, x) nicht definiert, Codierung $\text{cod}_{fgm}(z, x) = [\text{cod}_Z(z) \text{cod}_X(x) \square \square \square]$.
- ▶ wenn für Argumentpaar (z, x) definiert, Codierung $\text{cod}_{fgm}(z, x) = [\text{cod}_Z(z) \text{cod}_X(x) \text{cod}_Z(f(z, x)) \text{cod}_X(g(z, x)) \text{cod}_M(m(z, x))]$.
- ▶ Beispiel (siehe BB3):
 - ▶ wenn $(f, g, m)(A, \square) = (B, 1, R)$
dann $\text{cod}_{fgm}(A, \square) = [[00] [0] [01] [1] [01]]$
 - ▶ wenn $(f, g, m)(C, 1)$ undefiniert
dann $\text{cod}_{fgm}(C, 1) = [[10] [1] \square \square \square]$

Turingmaschine $T = (Z, z_0, X, \square, f, g, m)$

- ▶ Codierung der gesamten Funktionen:
Konkatenation aller $\text{cod}_{fgm}(z, x)$ für alle $z \in Z, x \in X$.
- ▶ Codierung der gesamten Turingmaschine:
Konkatenation von
 - ▶ Codierung des Zustands mit der größten Nummer,
 - ▶ Codierung des Bandsymbols mit der größten Nummer und
 - ▶ Codierung der gesamten Funktionen f, g und m .
- ▶ Schreibe T_w für die Turingmaschine mit Codierung w .

- ▶ **einfache Syntaxanalyse** ist möglich
 - ▶ man kann Turingmaschine konstruieren, die für $w \in A^*$ feststellt, ob es die Codierung einer Turingmaschine ist oder nicht.
 - ▶ Mehr brauchen wir im folgenden nicht.
- ▶ **universelle Turingmaschine** U existiert
 - ▶ erhält als Eingabe zwei Argumente, etwa als Wort $[w_1][w_2]$,
 - ▶ prüft, ob w_1 Codierung einer Turingmaschine T ist,
 - ▶ falls nein: diese Mitteilung und halt.
 - ▶ falls ja:
 - ▶ U simuliert Schritt für Schritt die Arbeit, die T für Eingabe w_2 durchführen würde,
 - ▶ und falls T endet, liefert U am Ende als Ergebnis das, was T liefern würde.

Eine technische Vorbemerkung

Turingmaschinen

Berechnungen

Eingaben für Turingmaschinen

Ergebnisse von Turingmaschinen

Berechnungskomplexität

Komplexitätsmaße

Komplexitätsklassen

Schwere Probleme

Unentscheidbare Probleme

Codierungen von Turingmaschinen

Das Halteproblem

Die Busy-Beaver-Funktion

Das *Halteproblem* ist die formale Sprache

$$H = \{w \in A^* \mid w \text{ ist eine TM-Codierung und } T_w(w) \text{ hält.}\}$$

Satz

Das Halteproblem ist unentscheidbar, d. h. es gibt keine Turingmaschine, die H entscheidet.

Es gibt Probleme, die man nicht mit dem Rechner, also algorithmisch, lösen kann!

- ▶ Beweis der Unentscheidbarkeit von H benutzt Diagonalisierung.
- ▶ Idee von Georg Ferdinand Ludwig Philipp Cantor (1845–1918)
 - ▶ für Überabzählbarkeit von \mathbb{R} benutzt
- ▶ betrachten erst die Kernidee, danach die Anwendung auf Halteproblem

„zweidimensionale unendliche Tabelle“

- ▶ Zeilen mit Funktionen f_i ($i \in \mathbb{N}_0$) indiziert
- ▶ Spalten mit Argumenten x_j ($j \in \mathbb{N}_0$) indiziert
- ▶ Eintrag in Zeile i und Spalte j : Funktionswert $f_i(x_j)$

	x_0	x_1	x_2	x_3	x_4	\dots
f_0	$f_0(x_0)$	$f_0(x_1)$	$f_0(x_2)$	$f_0(x_3)$	$f_0(x_4)$	\dots
f_1	$f_1(x_0)$	$f_1(x_1)$	$f_1(x_2)$	$f_1(x_3)$	$f_1(x_4)$	\dots
f_2	$f_2(x_0)$	$f_2(x_1)$	$f_2(x_2)$	$f_2(x_3)$	$f_2(x_4)$	\dots
f_3	$f_3(x_0)$	$f_3(x_1)$	$f_3(x_2)$	$f_3(x_3)$	$f_3(x_4)$	\dots
f_4	$f_4(x_0)$	$f_4(x_1)$	$f_4(x_2)$	$f_4(x_3)$	$f_4(x_4)$	\dots
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\ddots

Diagonalisierung: Kernidee (2)

	x_0	x_1	x_2	x_3	x_4	\dots
f_0	$f_0(x_0)$	$f_0(x_1)$	$f_0(x_2)$	$f_0(x_3)$	$f_0(x_4)$	\dots
f_1	$f_1(x_0)$	$f_1(x_1)$	$f_1(x_2)$	$f_1(x_3)$	$f_1(x_4)$	\dots
f_2	$f_2(x_0)$	$f_2(x_1)$	$f_2(x_2)$	$f_2(x_3)$	$f_2(x_4)$	\dots
f_3	$f_3(x_0)$	$f_3(x_1)$	$f_3(x_2)$	$f_3(x_3)$	$f_3(x_4)$	\dots
f_4	$f_4(x_0)$	$f_4(x_1)$	$f_4(x_2)$	$f_4(x_3)$	$f_4(x_4)$	\dots
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\ddots

Diagonalisierung: Kernidee (2)

	x_0	x_1	x_2	x_3	x_4	\dots
f_0	$f_0(x_0)$	$f_0(x_1)$	$f_0(x_2)$	$f_0(x_3)$	$f_0(x_4)$	\dots
f_1	$f_1(x_0)$	$f_1(x_1)$	$f_1(x_2)$	$f_1(x_3)$	$f_1(x_4)$	\dots
f_2	$f_2(x_0)$	$f_2(x_1)$	$f_2(x_2)$	$f_2(x_3)$	$f_2(x_4)$	\dots
f_3	$f_3(x_0)$	$f_3(x_1)$	$f_3(x_2)$	$f_3(x_3)$	$f_3(x_4)$	\dots
f_4	$f_4(x_0)$	$f_4(x_1)$	$f_4(x_2)$	$f_4(x_3)$	$f_4(x_4)$	\dots
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\ddots
d	$f_0(x_0)$	$f_1(x_1)$	$f_2(x_2)$	$f_3(x_3)$	$f_4(x_4)$	\dots

Diagonalisierung: Kernidee (2)

	x_0	x_1	x_2	x_3	x_4	\dots
f_0	$f_0(x_0)$	$f_0(x_1)$	$f_0(x_2)$	$f_0(x_3)$	$f_0(x_4)$	\dots
f_1	$f_1(x_0)$	$f_1(x_1)$	$f_1(x_2)$	$f_1(x_3)$	$f_1(x_4)$	\dots
f_2	$f_2(x_0)$	$f_2(x_1)$	$f_2(x_2)$	$f_2(x_3)$	$f_2(x_4)$	\dots
f_3	$f_3(x_0)$	$f_3(x_1)$	$f_3(x_2)$	$f_3(x_3)$	$f_3(x_4)$	\dots
f_4	$f_4(x_0)$	$f_4(x_1)$	$f_4(x_2)$	$f_4(x_3)$	$f_4(x_4)$	\dots
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\ddots
d	$f_0(x_0)$	$f_1(x_1)$	$f_2(x_2)$	$f_3(x_3)$	$f_4(x_4)$	\dots
\bar{d}	$\overline{f_0(x_0)}$	$\overline{f_1(x_1)}$	$\overline{f_2(x_2)}$	$\overline{f_3(x_3)}$	$\overline{f_4(x_4)}$	\dots

$$\bar{d}(x_i) = \overline{f_i(x_i)} = \begin{cases} 1 & \text{falls } f_i(x_i) = 0 \\ 0 & \text{sonst} \end{cases}$$

	x_0	x_1	x_2	x_3	x_4	\dots
f_0	$f_0(x_0)$	$f_0(x_1)$	$f_0(x_2)$	$f_0(x_3)$	$f_0(x_4)$	\dots
f_1	$f_1(x_0)$	$f_1(x_1)$	$f_1(x_2)$	$f_1(x_3)$	$f_1(x_4)$	\dots
f_2	$f_2(x_0)$	$f_2(x_1)$	$f_2(x_2)$	$f_2(x_3)$	$f_2(x_4)$	\dots
f_3	$f_3(x_0)$	$f_3(x_1)$	$f_3(x_2)$	$f_3(x_3)$	$f_3(x_4)$	\dots
f_4	$f_4(x_0)$	$f_4(x_1)$	$f_4(x_2)$	$f_4(x_3)$	$f_4(x_4)$	\dots
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\ddots
d	$f_0(x_0)$	$f_1(x_1)$	$f_2(x_2)$	$f_3(x_3)$	$f_4(x_4)$	\dots
\bar{d}	$\overline{f_0(x_0)}$	$\overline{f_1(x_1)}$	$\overline{f_2(x_2)}$	$\overline{f_3(x_3)}$	$\overline{f_4(x_4)}$	\dots

\bar{d} unterscheidet sich von jeder Zeile der Tabelle.

Satz

Es gibt keine Turingmaschine, die

$$H = \{w \in A^* \mid w \text{ ist eine TM-Codierung und } T_w(w) \text{ hält.}\}$$

entscheidet.

Beachte:

- ▶ Es geht um „entscheidet“, nicht nur um „erkennt“.
- ▶ Es geht um Turingmaschinen, die *immer* halten.

- ▶ benutze (eine Variante der) Diagonalisierung
- ▶ in der Tabelle
 - ▶ x_i : alle Codierungen von Turingmaschinen
 - ▶ f_i : die Funktion, die von Turingmaschine T_{x_i} berechnet wird.
- ▶ beliebige Turingmaschinen: manche der $f_i(x_j)$ nicht definiert
- ▶ da x_i *alle* Codierungen von Turingmaschinen sind, in der Tabelle für *jede* Turingmaschine eine Zeile
- ▶ **indirekter Beweis**: Annahme es gibt Turingmaschine T_h , die das Halteproblem entscheidet
 - ▶ d. h. für *jede* Eingabe x_i hält und
 - ▶ als Ergebnis mitteilt, ob T_{x_i} für Eingabe x_i hält oder nicht.
- ▶ **Widerspruch** durch „verdorbene Diagonale“ \bar{d} :
 - ▶ Einerseits unterscheidet sich \bar{d} von jeder Zeile der Tabelle, also von jeder von einer Turingmaschine berechneten Funktion.
 - ▶ Andererseits kann auch \bar{d} von einer Turingmaschine berechnet werden.

- ▶ benutze (eine Variante der) Diagonalisierung
- ▶ in der Tabelle
 - ▶ x_i : alle Codierungen von Turingmaschinen
 - ▶ f_i : die Funktion, die von Turingmaschine T_{x_i} berechnet wird.
- ▶ beliebige Turingmaschinen: manche der $f_i(x_j)$ nicht definiert
- ▶ da x_i *alle* Codierungen von Turingmaschinen sind, in der Tabelle für *jede* Turingmaschine eine Zeile
- ▶ **indirekter Beweis**: Annahme es gibt Turingmaschine T_h , die das Halteproblem entscheidet
 - ▶ d. h. für *jede* Eingabe x_i hält und
 - ▶ als Ergebnis mitteilt, ob T_{x_i} für Eingabe x_i hält oder nicht.
- ▶ **Widerspruch** durch „verdorbene Diagonale“ \bar{d} :
 - ▶ Einerseits unterscheidet sich \bar{d} von jeder Zeile der Tabelle, also von jeder von einer Turingmaschine berechneten Funktion.
 - ▶ Andererseits kann auch \bar{d} von einer Turingmaschine berechnet werden.

- ▶ benutze (eine Variante der) Diagonalisierung
- ▶ in der Tabelle
 - ▶ x_i : alle Codierungen von Turingmaschinen
 - ▶ f_i : die Funktion, die von Turingmaschine T_{x_i} berechnet wird.
- ▶ beliebige Turingmaschinen: manche der $f_i(x_j)$ nicht definiert
- ▶ da x_i *alle* Codierungen von Turingmaschinen sind, in der Tabelle für *jede* Turingmaschine eine Zeile
- ▶ **indirekter Beweis**: Annahme es gibt Turingmaschine T_h , die das Halteproblem entscheidet
 - ▶ d. h. für *jede* Eingabe x_i hält und
 - ▶ als Ergebnis mitteilt, ob T_{x_i} für Eingabe x_i hält oder nicht.
- ▶ **Widerspruch** durch „verdorbene Diagonale“ \bar{d} :
 - ▶ Einerseits unterscheidet sich \bar{d} von jeder Zeile der Tabelle, also von jeder von einer Turingmaschine berechneten Funktion.
 - ▶ Andererseits kann auch \bar{d} von einer Turingmaschine berechnet werden.

- ▶ Wenn die Turingmaschine T_h existiert
- ▶ dann kann auch das folgende eine Turingmaschine $T_{\bar{d}}$:
 - ▶ Für Eingabe x_i berechnet $T_{\bar{d}}$ zunächst, welches Ergebnis T_h für diese Eingabe liefern würde.
 - ▶ Dann:
 - ▶ Wenn T_h mitteilt, dass $T_{x_i}(x_i)$ hält, dann geht $T_{\bar{d}}$ in eine Endlosschleife.
 - ▶ Wenn T_h mitteilt, dass $T_{x_i}(x_i)$ nicht hält, dann hält $T_{\bar{d}}$ (und liefert irgendein Ergebnis, etwa 0).
 - ▶ andere Möglichkeiten gibt es nicht
 - ▶ in beiden Fällen verhält sich $T_{\bar{d}}$ für Eingabe x_i anders als T_{x_i}
- ▶ Wenn Turingmaschine T_h existiert, dann existiert auch Turingmaschine $T_{\bar{d}}$,
- ▶ aber jede Turingmaschine (T_{x_i}) verhält sich für mindestens eine Eingabe (x_i) anders als $T_{\bar{d}}$.
- ▶ **Widerspruch**
- ▶ Also ist die Annahme falsch, dass es die Turingmaschine T_h gibt, die H entscheidet.

- ▶ Wenn die Turingmaschine T_h existiert
- ▶ dann kann auch das folgende eine Turingmaschine $T_{\bar{d}}$:
 - ▶ Für Eingabe x_i berechnet $T_{\bar{d}}$ zunächst, welches Ergebnis T_h für diese Eingabe liefern würde.
 - ▶ Dann:
 - ▶ Wenn T_h mitteilt, dass $T_{x_i}(x_i)$ hält, dann geht $T_{\bar{d}}$ in eine Endlosschleife.
 - ▶ Wenn T_h mitteilt, dass $T_{x_i}(x_i)$ nicht hält, dann hält $T_{\bar{d}}$ (und liefert irgendein Ergebnis, etwa 0).
 - ▶ andere Möglichkeiten gibt es nicht
 - ▶ in beiden Fällen verhält sich $T_{\bar{d}}$ für Eingabe x_i anders als T_{x_i}
- ▶ Wenn Turingmaschine T_h existiert, dann existiert auch Turingmaschine $T_{\bar{d}}$,
- ▶ aber jede Turingmaschine (T_{x_i}) verhält sich für mindestens eine Eingabe (x_i) anders als $T_{\bar{d}}$.
- ▶ **Widerspruch**
- ▶ Also ist die Annahme falsch, dass es die Turingmaschine T_h gibt, die H entscheidet.

- ▶ Wenn die Turingmaschine T_h existiert
- ▶ dann kann auch das folgende eine Turingmaschine $T_{\bar{d}}$:
 - ▶ Für Eingabe x_i berechnet $T_{\bar{d}}$ zunächst, welches Ergebnis T_h für diese Eingabe liefern würde.
 - ▶ Dann:
 - ▶ Wenn T_h mitteilt, dass $T_{x_i}(x_i)$ hält, dann geht $T_{\bar{d}}$ in eine Endlosschleife.
 - ▶ Wenn T_h mitteilt, dass $T_{x_i}(x_i)$ nicht hält, dann hält $T_{\bar{d}}$ (und liefert irgendein Ergebnis, etwa 0).
 - ▶ andere Möglichkeiten gibt es nicht
 - ▶ in beiden Fällen verhält sich $T_{\bar{d}}$ für Eingabe x_i anders als T_{x_i}
- ▶ Wenn Turingmaschine T_h existiert, dann existiert auch Turingmaschine $T_{\bar{d}}$,
- ▶ aber jede Turingmaschine (T_{x_i}) verhält sich für mindestens eine Eingabe (x_i) anders als $T_{\bar{d}}$.
- ▶ **Widerspruch**
- ▶ Also ist die Annahme falsch, dass es die Turingmaschine T_h gibt, die H entscheidet.

Eine technische Vorbemerkung

Turingmaschinen

Berechnungen

Eingaben für Turingmaschinen

Ergebnisse von Turingmaschinen

Berechnungskomplexität

Komplexitätsmaße

Komplexitätsklassen

Schwere Probleme

Unentscheidbare Probleme

Codierungen von Turingmaschinen

Das Halteproblem

Die Busy-Beaver-Funktion

- ▶ Bandalphabet ist $X = \{\square, 1\}$.
- ▶ Turingmaschine hat $3 + 1$ Zustände
 - ▶ in 3 Zuständen für jedes Bandsymbol Fortsetzung definiert
 - ▶ einer dieser 3 Zustände ist Anfangszustand
 - ▶ in Zustand 4 für kein Bandsymbol Fortsetzung („Haltezustand“).
- ▶ Wenn man die Turingmaschine auf dem leeren Band startet, dann hält sie nach endlich vielen Schritten.

n -Bibermaschine:

- ▶ Bandalphabet ist $X = \{\square, 1\}$.
- ▶ Turingmaschine hat $n + 1$ Zustände
 - ▶ in n Zuständen für jedes Bandsymbol Fortsetzung definiert
 - ▶ einer dieser n Zustände ist Anfangszustand
 - ▶ in Zustand $n + 1$ für kein Bandsymbol Fortsetzung („Haltezustand“).
- ▶ Wenn man die Turingmaschine auf dem leeren Band startet, dann hält sie nach endlich vielen Schritten.
- ▶ Im folgenden geht es immer nur um die Berechnung, die auf dem vollständig leeren Band beginnt.

n -Bibermaschine:

- ▶ Bandalphabet ist $X = \{\square, 1\}$.
- ▶ Turingmaschine hat $n + 1$ Zustände
 - ▶ in n Zuständen für jedes Bandsymbol Fortsetzung definiert
 - ▶ einer dieser n Zustände ist Anfangszustand
 - ▶ in Zustand $n + 1$ für kein Bandsymbol Fortsetzung („Haltezustand“).
- ▶ Wenn man die Turingmaschine auf dem leeren Band startet, dann hält sie nach endlich vielen Schritten.
- ▶ Im folgenden geht es immer nur um die Berechnung, die auf dem vollständig leeren Band beginnt.

- ▶ n -Bibermaschine heißt *fleißiger Biber*,
- ▶ wenn sie am Ende die maximale Anzahl Einsen auf dem Band hinterlässt unter allen n -Bibermaschinen.
- ▶ *Busy-Beaver-Funktion* (oder Radó-Funktion)

$$\text{bb} : \mathbb{N}_+ \rightarrow \mathbb{N}_+$$

$\text{bb}(n)$ = die Anzahl von Einsen, die eine fleißige
 n -Bibermaschine am Ende auf dem Band hinterlässt

n	$bb(n)$
1	1
2	
3	
4	
5	
6	
\vdots	

n	$bb(n)$	
1	1	
2	4	Radó (1963)
3		
4		
5		
6		
\vdots		

n	$bb(n)$	
1	1	
2	4	Radó (1963)
3	6	Radó (1963)
4		
5		
6		
\vdots		

n	$bb(n)$	
1	1	
2	4	Radó (1963)
3	6	Radó (1963)
4	13	Brady (1974(?))
5		
6		
⋮		

n	$bb(n)$	
1	1	
2	4	Radó (1963)
3	6	Radó (1963)
4	13	Brady (1974(?))
5	≥ 4098	Marxen/Buntrock (1990)
6		
\vdots		

n	$bb(n)$	
1	1	
2	4	Radó (1963)
3	6	Radó (1963)
4	13	Brady (1974(?))
5	≥ 4098	Marxen/Buntrock (1990)
6	$> 4.6 \cdot 10^{1439}$	Ligocki/Ligocki (2007)
\vdots	\vdots	

Satz

Für jede berechenbare Funktion $f : \mathbb{N}_+ \rightarrow \mathbb{N}_+$ gibt es ein n_0 , so dass für alle $n \geq n_0$ gilt: $\text{bb}(n) > f(n)$.

Korollar

Die Busy-Beaver-Funktion $\text{bb}(n)$ ist nicht berechenbar.

Das sollten Sie mitnehmen:

- ▶ Das Halteproblem ist unentscheidbar.
- ▶ viele andere interessierende Probleme auch
- ▶ Die Busy-Beaver-Funktion wächst schneller als jede berechenbare Funktion.

Das sollten Sie üben:

- ▶ sich klar machen, dass informelle algorithmische Beschreibungen in Turingmaschinen überführt werden können

- ▶ Turingmaschinen sind eine formale Präzisierung des Algorithmusbegriffs.
- ▶ Komplexitätsmaße und Komplexitätsklassen
 - ▶ insbesondere **P** und **PSPACE**
 - ▶ später: **P** \subseteq **NP** \subseteq **PSPACE**
- ▶ Es gibt Probleme, die anscheinend großen algorithmischen Aufwand erfordern.
- ▶ Es gibt Probleme, die beweisbar sehr großen algorithmischen Aufwand erfordern.
- ▶ Es gibt Probleme, die algorithmisch überhaupt nicht lösbar sind.