

Theoretische Informatik

Rainer Schrader

Zentrum für Angewandte Informatik Köln

22. April 2009

1/95

Turingmaschinen

2/95

Turingmaschinen

Gliederung

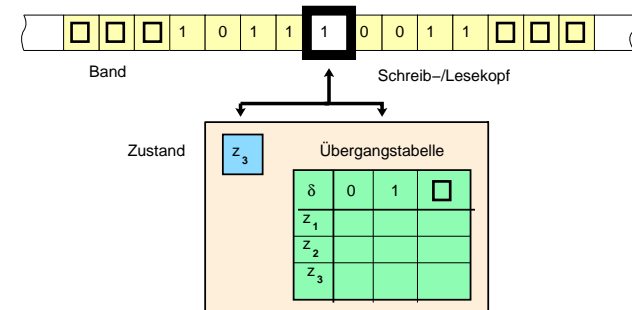
- **Aufbau und Eigenschaften**
- Maße für Zeit- und Platzbedarf
- Programmierung von DTM's
- universelle Turing-Maschinen
- Berechenbarkeit
- Nichtentscheidbarkeit

3/95

Turingmaschinen

Eine (deterministische) Turingmaschine

- besteht aus:
 - einer Recheneinheit
 - einem zweiseitig unbeschränkten Band (= Speicher)
 - einem Schreib-/Lesekopf
 - einer Übergangstabelle (Programm)



4/95

Turingmaschinen

Eine (deterministische) Turingmaschine

- besteht aus:
 - einer Recheneinheit
 - einem zweiseitig unbeschränkten Band (= Speicher)
 - einem Schreib-/Lesekopf
 - einer Übergangstabelle (Programm)
- ist stets in einem von endlich vielen **Zuständen** $z \in Z$
- besitzt zwei ausgezeichnete Typen von Zuständen:
 - den **Startzustand** z_0
 - eine Teilmenge $E \subseteq Z$ von **Endzuständen**
- ist zu Beginn im Startzustand z_0
- hält (wenn sie hält), sobald sie einen Endzustand erreicht

5/95

Turingmaschinen

- das Band ist in einzelne Zellen unterteilt
- jede Zelle enthält einen Buchstaben aus dem Alphabet $A = \{0, 1\} \cup \square$, wobei $\square = \text{blank}$
- zu Beginn steht die Eingabe x_1, \dots, x_n mit $x_i \neq \square$ auf den Feldern $i = 1, \dots, n$
- alle anderen Felder enthalten \square
- der Kopf steht über dem Feld 1
- in jedem Schritt liest der Kopf auf dem Feld, über dem er steht, einen Buchstaben

6/95

Turingmaschinen

- das Programm besteht aus der Übergangstabelle
- die Tabelle hat $|Z \setminus E|$ Zeilen und drei Spalten
- die Tabelle definiert eine **Übergangsfunktion**
$$\delta : Z \setminus E \times A \longrightarrow Z \times A \times \{L, R, N\}$$
- für jeden Zustand z und für jedes mögliche Zeichen a auf dem Band beschreibt die Tabelle die nächste Aktion $\delta(z, a)$

$\delta(z, a) = (z', b, x)$ bedeutet:

- M ist im Zustand z und unter dem L/S-Kopf steht a ,
- M überschreibt a mit b ,
- und bewegt den L/S-Kopf nach links ($x = L$), rechts ($x = R$), gar nicht ($x = N$)
- und geht in den Zustand z' über

7/95

Turingmaschinen

Eine **Turingmaschine (DTM)** ist ein 7-Tupel

$$M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E) \quad \text{mit}$$

Z :	endliche Zustandsmenge
Σ :	Eingabealphabet
$\Gamma \supseteq \Sigma$:	Arbeitsalphabet
$\delta : Z \times \Gamma \rightarrow Z \times \Gamma \times \{L, R, N\}$:	Überföhrungsfunktion
$z_0 \in Z$:	Startzustand
$\square \in \Gamma \setminus \Sigma$:	Blank
$E \subseteq Z$:	Menge der Endzustände

8/95

Turingmaschinen

- eine **Konfiguration** einer Turingmaschine ist ein Wort $k \in \Gamma^* Z \Gamma^*$
- informell ist eine Konfiguration eine „Momentaufnahme“ der DTM
- Interpretation von $k = \alpha z \beta$:
 - $\alpha \beta$ ist der nicht-leere bzw. bereits besuchte Teil des Bandes
 - z ist der Maschinenzustand
 - der L/S-Kopf befindet sich auf dem ersten Zeichen von β
- die Startkonfiguration ist
 - $z_0 x$: die Eingabe $x \in \Sigma^*$ steht auf dem sonst leeren Band, der L/S-Kopf auf erstem Zeichen von x .

9/95

Turingmaschinen

- auf der Menge der Konfigurationen ist eine zweistellige Relation \vdash definiert
- sie beschreibt den Übergang von einer Konfiguration zur nächsten
- sei $k = \alpha a z b \beta$ eine Konfiguration mit $|\alpha a| = m$ und $|b \beta| = n$
- dann ist $k \vdash k'$ mit

$$k' = \begin{cases} \alpha a z' c \beta, & \text{falls } \delta(z, b) = (z', c, N), \quad (m \geq 0, n \geq 1) \\ \alpha a c z' \beta, & \text{falls } \delta(z, b) = (z', c, R), \quad (m \geq 0, n \geq 2) \\ \alpha z' a c \beta, & \text{falls } \delta(z, b) = (z', c, L), \quad (m \geq 1, n \geq 1) \end{cases}$$

- Für $n = 1$: $\alpha a z b \vdash \alpha a c z' \square$, falls $\delta(z, b) = (z', c, R)$
- Für $m = 0$: $z b \beta \vdash z' \square c \beta$, falls $\delta(z, b) = (z', c, L)$
- k' ist die **Nachfolgekonfiguration** von k
- \vdash^* bezeichne die reflexive und transitive Hülle von \vdash

10/95

Turingmaschinen

Beispiel

Die folgende DTM interpretiert $x \in \{0, 1\}^*$ als Dualzahl und addiert 1 hinzu:

Lesezustand	$\delta(z_0, 0) = (z_0, 0, R)$ $\delta(z_0, 1) = (z_0, 1, R)$ $\delta(z_0, \square) = (z_1, \square, L)$	/* lies die Zahl
Übertragszustand	$\delta(z_1, 0) = (z_2, 1, L)$ $\delta(z_1, 1) = (z_1, 0, L)$ $\delta(z_1, \square) = (e, 1, N)$	/* addiere 1
kein Übertrag	$\delta(z_2, 0) = (z_2, 0, L)$ $\delta(z_2, 1) = (z_2, 1, L)$ $\delta(z_2, \square) = (e, \square, R)$	

Start mit Eingabe 101:

$z_0 101 \vdash 1 z_0 01 \vdash 10 z_0 1 \vdash 101 z_0 \square \vdash 10 z_1 1 \square \vdash 1 z_1 00 \square \vdash z_2 110 \square \vdash z_2 \square 110 \square \vdash \square z_e 110 \square$

11/95

Turingmaschinen

Bemerkung

- wir können stets annehmen, dass $\Gamma \setminus \{\square\} = \Sigma = \{0, 1\}$:
- sei $\Gamma = \{a_0, a_1, \dots, a_k\}$
- jedem a_i ordnen wir das Wort $\# \text{bin}(i) \#$ zu
- (hierbei ist $\text{bin}(i)$ die Binärdarstellung von i)
- das Resultat ist eine Kodierung der DTM über $\{0, 1, \#\}$
- mit den Substitutionen $0 \rightarrow 00$, $1 \rightarrow 01$ und $\# \rightarrow 11$ erhalten wir das Alphabet $\{0, 1\}$

12/95

Turingmaschinen

- die Übergangsfunktion kann durch eine einfache Programmiersprache beschrieben werden
- die Anweisungen lauten:
 - **left**
 - **right**
 - **write a** für $a \in \{0, 1, \square\}$
 - **case a goto i** für $a \in \{0, 1, \square\}$ und $i \in \mathbb{N}$
 - **stop**
- die Programmzeilen sind nummeriert
- die Zustände werden von den Programmzeilen übernommen

13/95

Turingmaschinen

Beispiel: binäre Addition einer 1:

```

/*   gehe nach rechts bis □ gefunden
10   case □ goto 160
20   right
30   case 0 goto 20
40   case 1 goto 20
/*   Übertrag
50   left
60   case 1 goto 100
70   case 0 goto 120
80   write 1
90   case 1 goto 160
100  write 0
110  case 0 goto 50
120  write 1
/*   kein Übertrag
130  left
140  case 0 goto 130
150  case 1 goto 130
160  stop
    
```

14/95

Turingmaschinen

- wenn die DTM M hält, so fassen wir das Wort rechts vom Kopf bis zum ersten \square als Ergebnis auf
- damit berechnet M die folgende partielle Funktion:

$$f_M : \{0, 1\}^* \xrightarrow{\text{part}} \{0, 1\}^*$$

$$f_M(x) = \begin{cases} \beta, & \text{es existiert } k' = \alpha z \beta \text{ mit } z_0 x \vdash^* k' \text{ und } z \in E \\ \text{nd}, & \text{andernfalls} \end{cases}$$

15/95

Turingmaschinen

- sind die Antworten binär (0 oder 1), so sagen wir, dass M eine Input **akzeptiert** oder **verwirft**
- wir können dann auf die Ausgabe verzichten und dafür neue Endzustände Z_{akzept} und Z_{verwirf} einführen
- dann wird ein Input x akzeptiert, wenn die Berechnung in einem Zustand $Z_{\text{akzept}} \in E$ endet

16/95

Turingmaschinen

Gliederung

- Aufbau und Eigenschaften
- **Maße für Zeit- und Platzbedarf**
- Programmierung von DTM's
- universelle Turing-Maschinen
- Berechenbarkeit
- Nichtentscheidbarkeit

17/95

Turingmaschinen

Idee

- Zeit- und Platzbedarf werden Funktionen vom Input x sein
- wie in „Informatik I“ sind wir weniger an konkreten Aussagen für jedes x interessiert
- sondern am Verhalten für alle Inputs gleicher Länge
- wir werden daher Schranken in Abhängigkeit von der Anzahl der Bits des Inputs herleiten
- d.h. in Abhängigkeit von der **Länge** $|x|$
- wir messen somit den maximal benötigten Zeit-/Platzbedarf bei einem Input mit n bits

18/95

Turingmaschinen

Laufzeitmaß

Sei M eine DTM

- ein **Rechenschritt** entspricht einem direkten Konfigurationsübergang
- wir zählen die Rechenschritte, die die Maschine durchführt
- setze:

$$T^M(x) = \begin{cases} \# \text{ Rechenschritte,} & \text{wenn } M \text{ einen Endzustand erreicht} \\ \infty, & \text{sonst} \end{cases}$$

19/95

Turingmaschinen

- setze:

$$T^M(x) = \begin{cases} \# \text{ Rechenschritte,} & \text{wenn } M \text{ einen Endzustand erreicht} \\ \infty, & \text{sonst} \end{cases}$$

- sei $t : \mathbb{N} \rightarrow \mathbb{N}$
- M heißt **$t(n)$ -zeitbeschränkt**, falls $T^M(x) \leq t(|x|)$
- anschaulich:
 - M stoppt immer
 - für alle Inputs x der Länge $|x| = n$ gilt:
 - M hält nach höchstens $t(n)$ Schritten

20/95

Turingmaschinen

- für den benötigten Platz einer Maschine zählen wir die maximale Anzahl der belegten Speicherplätze
- für jedes x liefert dies einen Wert $S^M(x)$
- für eine DTM ist dies das längste Wort $\alpha\beta$ einer Konfiguration $\alpha z\beta$
- formal:

21/95

Turingmaschinen

Speicherplatzmaß

Sei M eine DTM

- sei

$$S^M(x) = \max\{|k_i| : z_0 x \vdash^* k_i\}$$

- M heißt **$s(n)$ -platzbeschränkt**, falls $S^M(x) \leq s(|x|)$ für alle x .
- offensichtlich gilt: der Speicherplatzbedarf ist nie größer als der Zeitbedarf
- denn bei jeder Vergrößerung des Speichers tritt ein Rechenschritt auf

22/95

Turingmaschinen

Gliederung

- Aufbau und Eigenschaften
- Maße für Zeit- und Platzbedarf
- **Programmierung von DTM's**
- universelle Turing-Maschinen
- Berechenbarkeit
- Nichtentscheidbarkeit

23/95

Turingmaschinen

Datenspeicherung

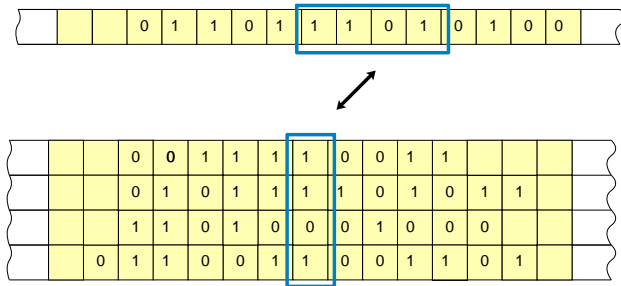
- sei p ein Parameter der Werte aus einer endlichen Menge A annimmt
- angenommen, wir wollen den Wert von p immer schnell verfügbar haben
- wir erweitern dazu die Zustandsmenge Z zu $Z' = Z \times A$
- Zustandswechsel beinhalten dann eine Wechsel des eigentlichen Zustands und des Parameters

24/95

Turingmaschinen

Verwendung mehrerer Spuren

Wie ein Magnetband können wir das Band in k Spuren S_1, \dots, S_k unterteilen:



- wir identifizieren dazu S_i mit den Zellen $\{j : j \equiv i \pmod k\}$
- und erweitern das Alphabet zu Σ^k
- wir werden gleich zeigen, wie wir k Spuren auf einer normalen DTM simulieren können

25/95

Turingmaschinen

einseitig beschränktes Band

- das unendliche Band kann durch ein einseitig unendliches Band ersetzt werden
- dazu identifizieren wir:
 - die negativen ganzen Zahlen | mit den geraden natürlichen Zahlen
 - die positiven ganzen Zahlen | mit den ungeraden natürlichen Zahlen

26/95

Turingmaschinen

Markieren von Speicherstellen

- wir wollen einzelne Zellen des Bandes markieren
- dazu führen wir eine zweite Spur ein
 - die erste Spur übernimmt die Funktion des ursprünglichen Bandes
 - die zweite Spur nimmt die Markierungen auf

27/95

Turingmaschinen

Eindeutiger Endzustand / leeres Band im Endzustand

- aus formalen Gründen ist es oft einfacher, annehmen zu können, dass der Endzustand eindeutig ist
- ebenfalls erwünscht ist bisweilen ist das vorhergehende Löschen des Bandes
- beides sind Spezialfälle der folgenden Operation

28/95

Turingmaschinen

Hintereinanderschaltung von DTM's

- seien M_1, M_2 zwei DTM's mit identischen Alphabeten und disjunkten Zustandsmengen
- wir wollen eine DTM M bauen, die:
 - M_1 ausführt
 - danach M_2 auf der Ausgabe von M_1 startet
- seien Z_i die Zustandsmenge von M_i
- setze $Z = Z_1 \cup Z_2$ und setze δ entsprechend
- setze $\delta(e, a) = (z_{0,2}, a, N)$ für $e \in E_1, a \in \Gamma$ und $z_{0,2}$ Startzustand von M_2

29/95

Turingmaschinen

Verwendung von Unterprogrammen

- seien M_1, M_2 zwei DTM's mit identischen Alphabeten und disjunkten Zustandsmengen
- M_1 soll M_2 als Unterprogramm verwenden dürfen
- wir gehen wie bei der Hintereinanderschaltung vor
- und erweitern Z_1 um zwei neue Zustände z_{call}, z_{return}
- und erweitern δ u.a. um $\delta(z_{call}, \cdot) = (z_{0,2}, \cdot, N)$ sowie $\delta(e_{0,2}, \cdot) = (z_{return}, \cdot, N)$
- Parameter können über eine dem Unterprogramm zugeordnete Spur übergeben werden

30/95

Turingmaschinen

Schleifen

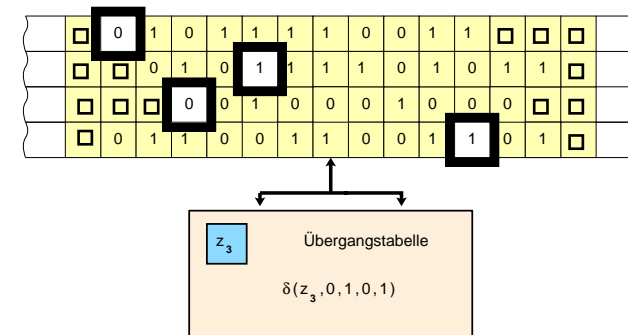
- lassen sich als spezielle Unterprogramme mit Abbruchkriterien auffassen
- bei **while**-Schleifen muss das Abbruchkriterium überprüft werden
- bei **for**-Schleifen wird ein Zähler verändert und mit dem Schleifenende verglichen

31/95

Turingmaschinen

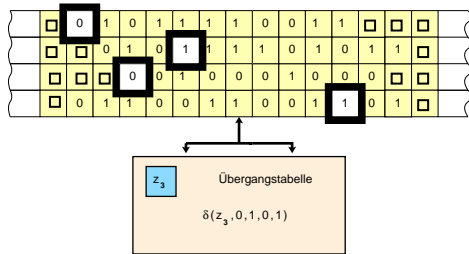
Mehrband-Turingmaschinen

- sei $k \in \mathbb{N}$ fest gewählt
- eine **k -Band-Turingmaschine** hat k Bänder und k Rechenköpfe, die sich unabhängig voneinander bewegen können



32/95

Turingmaschinen



- die Zustandsüberföhrungsfunktion geht dann von $Z \times A^k \rightarrow Z \times A^k \times \{L, R, N\}^k$
- in Abhangigkeit vom aktuellen Zustand und den k Zeichen unter den Kopfen wird unabhangig voneinander
 - jedes Band neu beschreiben
 - jeder Kopf bewegt und
 - ein neuer Zustand angenommen.

Turingmaschinen

Lemma

- sei M eine $t(n)$ -zeit- und $s(n)$ -platzbeschrankte k -Band-Turingmaschine
- M kann durch eine 1-Band-Turingmaschine M' simuliert werden,
- die $\mathcal{O}(t^2(n))$ -zeit- und $\mathcal{O}(s(n))$ -platzbeschrankt ist.

Beweisidee:

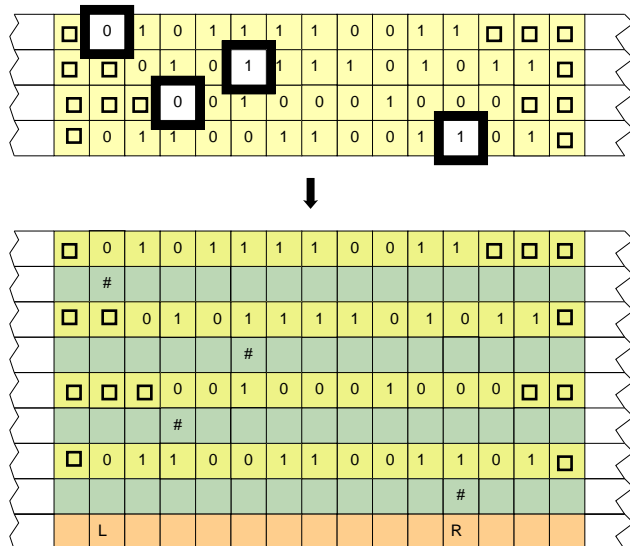
- M' verwendet $2k + 1$ „virtuelle“ Spuren
- die Zustandsmenge von M' enthalt einen endlichen Speicher $q_{i,a}$ mit der Bedeutung

$q_{i,a}$ die Zelle unter Kopf i enthalt das Zeichen $a \in \Gamma$

- dies sind nur konstant viele, da $k|\Gamma|$ eine Konstante ist

Turingmaschinen

Beweisidee:



Turingmaschinen

- nach der Simulation eines Schrittes von M durch M' soll gelten:
- die Spuren $1, 3, \dots, 2k - 1$ enthalten die Inhalte der Bander von M
- die Spuren $2, 4, \dots, 2k$ enthalten jeweils eine Markierung fur die Kopfpositionen von M
- die Spur $2k + 1$ enthalt zwei Markierungen L und R mit der Bedeutung:
 - L ist die am weitesten links liegende Position
 - R ist die am weitesten rechts liegende Position
- der Zustand von M' entspricht dem Zustand von M

Turingmaschinen

Zu Beginn eines Simulationsschrittes steht der Kopf von M' über der Zelle, die in der letzten Spur L enthält

- M' läuft nach rechts bis zur Markierung R
- unterwegs merkt sich M' in seinem endlichen Speicher, was die Köpfe von M lesen
- damit kennt M' den Zustand und die Bandinhalte von M und weiß, was M machen würde
- der Kopf von M' wandert wieder nach links
- wenn er eine markierte Kopfposition von der Kopf von M erreicht:
 - führt er die lokale Operation des Kopfes aus
 - ändert die Markierung der Kopfposition entsprechend
 - und schreibt ggf. L und R fort
- am Schluss nimmt M' den Zustand von M an.

Damit steht der Kopf wieder über der Zelle, die in der letzten Spur L enthält.

37/95

Turingmaschinen

- die Initialisierung von M' kann in $\mathcal{O}(1)$ durchgeführt werden
- in den $2k + 1$ werden in jedem Schritt höchstens so viele Zelle wie zwischen L und R belegt
- der Abstand zwischen L und R wächst in einem Schritt nur dann, wenn auch der Speicherbedarf von M wächst
- da dieser durch $s(n)$ beschränkt ist, ist M' $\mathcal{O}(s(n))$ -platzbeschränkt
- damit benötigt die Simulation eines Schritte von M auch nur $\mathcal{O}(s(n))$ Schritte
- somit ist M' $\mathcal{O}(t^2(n))$ -zeitbeschränkt. \square

38/95

Turingmaschinen

Gliederung

- Aufbau und Eigenschaften
- Maße für Zeit- und Platzbedarf
- Programmierung von DTM's
- **universelle Turing-Maschinen**
- Berechenbarkeit
- Nichtentscheidbarkeit

39/95

Turingmaschinen

DTM-Programmierung

- bisher ist das Programm in einer DTM fest installiert
- bei von-Neumann-Rechnern wird das Programm jedoch wie die Daten eingelesen und vom Rechner interpretiert
- wir wollen also versuchen, eine DTM zu konstruieren, die programmierbar ist

40/95

Turingmaschinen

- ein Programm besteht aus Zeilennummern und Befehlen B_i der Befehlsliste
- zur Vereinfachung unterstellen wir, dass die Zeilennummern den Zahlen $1, 2, \dots$ entsprechen
- jeder Befehl wird durch seine Nummer und einen Parameter kodiert

Befehl	BefehlsNr	Parameter	Bedeutung
left	1	0,0	beide ignorieren
right	2	0,0	beide ignorieren
write a	3	a,0	zu schreibendes Zeichen, ignorieren
case a goto i	4	a, i	gelesenes Zeichen/Sprungadresse
stop	5	0,0	beide ignorieren

- sei $M = (1, B_1, 2, B_2, \dots, k, B_k)$ ein Programm
- die **Kodierung** von M ist das Tupel
 $(k, 1, \text{BefehlsNr}(B_1), \text{Parameter}(B_1), \dots, k, \text{BefehlsNr}(B_k), \text{Parameter}(B_k), 2, 2)$
- alle Zahlen sind binär kodiert und durch Zweien getrennt

41/95

Turingmaschinen

- wie früher identifizieren wir das Alphabet über $\{0, 1, 2\}$ mit dem Alphabet über $\{0, 1\}$

- dabei setzen wir

$$\begin{aligned} 00 &\cong 0 \\ 01 &\cong 1 \\ 10 &\cong 2. \end{aligned}$$

- die binäre Kodierung eines Programmes M bezeichnen wir mit $G(M)$

42/95

Turingmaschinen

- die Eingabe besteht dann aus zwei Teilen: dem Programm gefolgt vom Input für das Programm
- der erste Schritt beim Ablauf eines Programms könnte eine **Syntaxanalyse** sein
- verläuft die Syntaxanalyse erfolgreich, wird danach das Programm gestartet
- die Syntaxanalyse kann mit einer DTM durchgeführt werden:

Lemma

Es gibt eine DTM, die bei Eingabe von $x \in \{0, 1\}^*$ testet, ob x die Kodierung eines Programms ist.

43/95

Turingmaschinen

Lemma

Es gibt eine DTM, die bei Eingabe von $x \in \{0, 1\}^*$ testet, ob x die Kodierung eines Programms ist.

Beweisskizze:

- interpretiere das Wort vor der ersten 2 als Binärkodierung einer Dezimalzahl k
- in einer **for**-Schleife der Länge k :
 - lies die nächsten vier Binärkodierungen von Dezimalzahlen i, B, O_1, O_2 ein
 - überprüfe, ob $1 \leq B \leq 5$
 - überprüfe, ob $O_1 = 0$ für $B = 1, 2, 5$
 - überprüfe, ob $O_2 = 0$ für $B = 1, 2, 3, 5$
 - überprüfe, ob $O_1 \in \Gamma$ für $B = 3, 4$
 - überprüfe, ob $1 \leq O_2 \leq k$ für $B = 4$
 - falls nicht oder Fehler im Input auftreten, antworte „syntaktisch nicht korrekt“
- falls der restliche Teil von x 2 Zweien, antworte „syntaktisch korrekt“
- andernfalls „syntaktisch nicht korrekt“ □

Wir werden im weiteren eine DTM mit ihrem Programm identifizieren.

44/95

Turingmaschinen

- sei M_U eine DTM
- M_U heißt **universell** oder **programmierbar**, falls:
- für jedes Programm M und jeden Input $x = (x_1, \dots, x_n) \in \{0, 1\}^*$ gilt:
- M_U „simuliert“ M , d.h.
- M_U gefüttert mit der Kodierung $G(M)$ von M gefolgt von x liefert $M(x)$

$$f_{M_U}(G(M), x) = f_M(x)$$

Satz

Es existiert eine universelle DTM M_U .

45/95

Turingmaschinen

Beweisidee:

- wir verwenden zur besseren Veranschaulichung eine DTM M_U mit vier Bändern
- wir wissen bereits, dass wir M_U durch eine klassische DTM simulieren können
- der Input $G(M), x$ steht auf Band 1
- M_U führt zuerst eine Syntaxanalyse auf $G(M)$ durch
- ist $G(M)$ syntaktisch korrekt, wird $G(M)$ auf Band 2 kopiert
- der Kopf von Band 1 steht über dem ersten Buchstaben von x
- Band 3 enthält die Zeilennummer des nächsten Befehls (zu Beginn 1)
- Band 4 enthält die Parameter des nächsten Befehls

46/95

Turingmaschinen

- für jeden der 5 Befehlstypen von M enthält M_U ein Unterprogramm
- die benötigten Parameter stehen auf Band 4
- ist der Befehl von M **stop**, so stoppt auch M_U
- die anderen Befehle erhöhen den Inhalt von Band 3 um 1 bzw. ersetzen ihn durch die Sprungadresse

47/95

Turingmaschinen

Durchführung der Simulation:

- suche auf Band 2 die nächste Anweisung i
- kopiere die zugehörigen Parameter auf Band 4
- rufe das zugehörige Unterprogramm auf

Die Laufzeit von M_U unterscheidet sich nur um einen konstanten Faktor von der Laufzeit von M . □

48/95

Turingmaschinen

Gliederung

- Aufbau und Eigenschaften
- Maße für Zeit- und Platzbedarf
- Programmierung von DTM's
- universelle Turing-Maschinen
- **Berechenbarkeit**
- Nichtentscheidbarkeit

49/95

Turingmaschinen

- jeder hat eine intuitive Vorstellung davon, was berechenbar ist
- aber mit intuitiven Berechenbarkeitsbegriffen können wir die Grenzen der Berechenbarkeit nicht ausloten
- in den folgenden Betrachtungen beschränken wir uns auf Funktionen der Form $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$
- der intuitive Berechenbarkeitsbegriff besagt dann etwa, dass eine solche Funktion f berechenbar ist, falls ein Algorithmus (oder ein Computerprogramm) existiert, der f berechnet
- d.h. für jede Eingabe $(x_1, x_2, \dots, x_k) \in \{0, 1\}^k$ stoppt das Verfahren nach endlich vielen Schritten mit der Ausgabe $f(x_1, x_2, \dots, x_k)$
- ist die Funktion f nur partiell definiert, so soll die Berechnung für undefinierte Eingaben nicht stoppen (Endlosschleife)

50/95

Turingmaschinen

Beispiel

Das Programm

```
1 case 0 goto 1
2 case 1 goto 1
3 case □ goto 3
```

berechnet die überall undefinierte Funktion $\Omega : n \mapsto \text{undef.}$

Beispiel

Die Funktion

$$f(n) = \begin{cases} 1, & \text{falls } n \text{ Anfangsabschnitt der Dezimalbruchentwicklung} \\ & \text{von } \pi \text{ ist} \\ 0, & \text{sonst} \end{cases}$$

(z.B. $f(314) = 1$, $f(50) = 0$) ist berechenbar, denn es gibt Näherungsverfahren für π .

51/95

Turingmaschinen

Beispiel

- sei f die folgende Funktion

$$f(n) = \begin{cases} 1, & \text{falls } n \text{ irgendwo in der Dezimalbruchentwicklung} \\ & \text{von } \pi \text{ vorkommt} \\ 0, & \text{sonst} \end{cases}$$

- wir wissen nicht, ob f berechenbar ist
- die Ziffernfolge in $\pi = 3, 14159 \dots$ erscheint so zufällig, dass vielleicht jede endliche Ziffernfolge vorkommt
- dann würde $g(n) = 1$ für alle n gelten und als konstante Funktion wäre g natürlich berechenbar

52/95

Turingmaschinen

Beispiel

- sei f die folgende Funktion

$$f(n) = \begin{cases} 1, & \text{falls in der Dezimalbruchentwicklung von } \pi \text{ irgendwo} \\ & n\text{-mal hintereinander eine 7 vorkommt} \\ 0, & \text{sonst} \end{cases}$$

- dann gilt:
 - entweder gibt es in der Dezimalbruchentwicklung von π beliebig lange 7er Folgen, so dass $h(n) = 1$ für alle $n \in \mathbb{N}$ gilt
 - oder es existiert ein $n_0 \in \mathbb{N}$, so dass in der Dezimalbruchentwicklung von π eine Folge mit n_0 „7“ existiert, aber keine mit $n_0 + 1$ „7“, d.h.

$$h(n) = \begin{cases} 1, & \text{falls } n \leq n_0, \\ 0, & \text{sonst.} \end{cases}$$

53/95

Turingmaschinen

$$f(n) = \begin{cases} 1, & \text{falls in der Dezimalbruchentwicklung von } \pi \text{ irgendwo} \\ & n\text{-mal hintereinander eine 7 vorkommt} \\ 0, & \text{sonst} \end{cases}$$

- dann gilt:
 - entweder gibt es in der Dezimalbruchentwicklung von π beliebig lange 7er Folgen, so dass $h(n) = 1$ für alle $n \in \mathbb{N}$ gilt
 - oder es existiert ein $n_0 \in \mathbb{N}$, so dass in der Dezimalbruchentwicklung von π eine Folge mit n_0 „7“ existiert, aber keine mit $n_0 + 1$ „7“, d.h.

$$h(n) = \begin{cases} 1, & \text{falls } n \leq n_0, \\ 0, & \text{sonst.} \end{cases}$$

- f ist berechenbar, denn in jedem Fall gibt es einen Algorithmus zur Berechnung von h ,
- aber wir können ihn nicht angeben
- man beachte: für die Berechenbarkeit reicht es aus, die Existenz eines Algorithmus nachzuweisen, der Algorithmus muss nicht unbedingt explizit angegeben werden

54/95

Turingmaschinen

- wir wollen jetzt den Begriff der Berechenbarkeit formalisieren
- wir folgen dabei unserer intuitiven Vorstellung (es muss ein Programm existieren ...)
- verwenden aber Turingmaschinen als zugelassene Programme
- anschaulich ist eine Funktion f berechenbar, wenn:
 - eine Turingmaschine existiert,
 - die bei Input x hält
 - und $f(x)$ auf das Band geschrieben hat

55/95

Turingmaschinen

- sei $f_M : \{0, 1\}^* \xrightarrow{\text{part}} \{0, 1\}^*$ eine partielle Funktion
- f heißt **berechenbar** (genauer **DTM-berechenbar**), wenn es eine DTM M gibt mit $f_M = f$, d.h.
 - ist $f(x)$ definiert, so schreibt M $f(x)$ auf das Ausgabeband
 - ist $f(x)$ nicht definiert, so hält M nicht

Beispiele

- die Nachfolgerfunktion $f(n) = n + 1$ ist berechenbar (vgl. einführendes Beispiel zur Addition einer 1)
- die nirgends definierte Funktion **nd** ist durch eine DTM M_{nd} , die nie hält, berechenbar

56/95

Turingmaschinen

- die DTM-Berechenbarkeit scheint mit unserer intuitiven Vorstellung von Berechenbarkeit übereinzustimmen
- unabhängig davon ist der Begriff „Berechenbarkeit“ mit verschiedenen Konzepten formuliert worden
- wir werden im nächsten Kapitel einige dieser Konzepte vorstellen
- es hat sich herausgestellt, dass all diese Konzepte äquivalent sind
- obwohl sie alle Abstraktionen von existierenden Maschinen sind, hat Church die Vermutung geäußert, dass diese Abstraktionen nicht mächtiger sind als die realen Maschinen:

57/95

Turingmaschinen

Church'sche These:

Die in einem intuitiven Sinne berechenbaren Funktionen sind genau die berechenbaren Funktionen.

- dies ist keine Aussage im Sinne der Mathematik
- insbesondere ist der unpräzise Begriff „intuitiv berechenbar“ nicht definiert
- er soll ausdrücken, dass alles, was auf einem abstrakten Rechnermodell berechnet werden kann, auch auf realen Maschinen berechnet werden kann

58/95

Turingmaschinen

- im weiteren interessiert uns auch das folgende speziellere Problem:
- ist die Antwort auf eine gegebene Frage „ja“ oder „nein“?
- genauer gesagt: können wir entscheiden, ob die Antwort auf eine kodierte Frage $x \in \{0, 1\}^*$ 0 oder 1 ist?
- äquivalent dazu ist: gegeben $x \in \{0, 1\}^*$ entscheide, ob $x \in f^{-1}(1)$
- sei $L = f^{-1}(1)$ die Menge der Urbilder der 1
- L ist als Menge von Wörtern eine Sprache
- wir fragen also, ob das **Spracherkennungsproblem**:

gegeben $L \subseteq \{0, 1\}^*$ und $x \in \{0, 1\}^*$, ist $x \in L$?

mittels einer DTM beantwortet werden kann.

59/95

Turingmaschinen

- sei $L \subseteq \{0, 1\}^*$ eine Sprache
- L heißt **entscheidbar**, falls die charakteristische Funktion von L

$$\chi_L : \{0, 1\}^* \rightarrow \{0, 1\} \quad \chi_L(w) = \begin{cases} 1, & \text{falls } w \in L \\ 0, & \text{falls } w \notin L \end{cases}$$

berechenbar ist

- L heißt **semi-entscheidbar** oder **aufzählbar**, falls die „halbe“ charakteristische Funktion von L

$$\chi'_L : \{0, 1\}^* \rightarrow \{0, 1\} \quad \chi'_L(w) = \begin{cases} 1, & \text{falls } w \in L \\ \text{undef}, & \text{falls } w \notin L \end{cases}$$

berechenbar ist

60/95

Turingmaschinen

- für eine entscheidbare Sprache existiert somit eine DTM M , für die gilt:
 - M stoppt in jedem Fall
 - $x \in L \iff M$ akzeptiert
 - $x \notin L \iff M$ verwirft
- für eine semi-entscheidbare Sprache existiert somit eine DTM M , für die gilt:
 - $x \in L \iff M$ akzeptiert
 - $x \notin L \iff M$ verwirft oder gerät in eine Endlosschleife

61/95

Turingmaschinen

Illustration

- Entscheidbarkeit:



- Semi-Entscheidbarkeit:



Den Begriff der Semi-Entscheidbarkeit kennen wir im Prinzip schon:

62/95

Turingmaschinen

Lemma

Eine Sprache ist genau dann semi-entscheidbar, wenn sie von einer DTM akzeptiert wird.

Beweis:

- sei L eine Sprache, die von einer DTM M akzeptiert wird
- konstruiere aus M eine DTM M' , die, wenn M akzeptiert, das Band löscht, eine 1 schreibt und in einem Endzustand hält
- dann berechnet M' die charakteristische Funktion

$$\chi'_L(w) = \begin{cases} 1, & \text{falls } w \in L \\ \text{undef}, & \text{sonst} \end{cases}$$

- sei umgekehrt M eine DTM, die die halbe charakteristische Funktion einer Sprache L berechnet
- dann erreicht M genau für die Wörter aus L einen akzeptierenden Zustand
- d.h. die von M akzeptierte Sprache ist L . □

63/95

Turingmaschinen

Satz

Eine Sprache L ist genau dann entscheidbar, wenn L und \bar{L} semi-entscheidbar sind.

Beweis:

- sei M eine DTM, die L entscheidet
- baue zwei DTM's M_1, M_2 , die das folgende leisten:
- für einen gegebenen Input x lassen beide M laufen
- gibt M eine 1 aus, so:
 - gibt M_1 ebenfalls 1 aus und
 - M_2 gerät in eine Endlosschleife
- gibt M eine 0 aus, so:
 - gerät M_1 in eine Endlosschleife und
 - M_2 gibt eine 1 aus
- dann akzeptiert M_1 die Sprache L und M_2 die Sprache \bar{L} .

64/95

Turingmaschinen

- seien umgekehrt M_1 und M_2 zwei DTM's, die L bzw. \bar{L} semi-entscheiden
- baue eine neue DTM M , die zu einem gegebenen Input x das folgende leistet:
 - sie lässt abwechselnd M_1 und M_2 jeweils einen Rechenschritt ausführen
 - eine der beiden Maschinen gerät irgendwann in einen Endzustand
 - ist dies M_1 , so gibt M eine 1 aus und hält
 - ist dies M_2 , so gibt M eine 0 aus und hält
- dann ist $f_M = L$. □

65/95

Turingmaschinen

Gliederung

- Aufbau und Eigenschaften
- Maße für Zeit- und Platzbedarf
- Programmierung von DTM's
- universelle Turing-Maschinen
- Berechenbarkeit
- **Nichtentscheidbarkeit**

66/95

Turingmaschinen

- nach der Church'schen These sind die berechenbaren Funktionen genau die Funktionen, die wir auf realen Rechnern berechnen können
- was lässt sich also DTM-berechnen?
- wir werden zeigen, dass es Funktionen gibt, die auf keiner DTM berechnet werden können
- der Beweis beruht auf der folgenden Tatsache:
 - es gibt überabzählbar unendlich viele Funktionen $f : \{0, 1\}^* \rightarrow \{0, 1\}$
 - aber nur abzählbar unendlich viele DTM's

67/95

Turingmaschinen

- sei A eine Menge
- A heißt **abzählbar**, falls $A = \emptyset$ oder es gibt eine Funktion $f : \mathbb{N} \rightarrow A$, so dass

$$A = \{f(0), f(1), f(2), \dots\}$$

(Berechenbarkeit wird nicht verlangt)

Beispiele

- für ein Alphabet Σ ist Σ^* abzählbar. (z.B. durchnummerieren in lexikographischer Ordnung.)
- die Menge aller Sprachen über Σ^*

$$2^{\Sigma^*} = \{L \mid L \subseteq \Sigma^*\}$$

ist nicht abzählbar („überabzählbar“), denn:

Satz

Die Potenzmenge einer abzählbar unendlichen Menge M ist nicht abzählbar.

68/95

Turingmaschinen

Beweis:

- wir identifizieren jede Teilmenge $A \subseteq M$ mit ihrer Indikatorfunktion $f_A : M \rightarrow \{0, 1\}$ mittels $f_A(m) = 1 \iff m \in A$

- dann ist die Behauptung äquivalent zu

$$\{f \mid f : M \rightarrow \{0, 1\}\}$$

ist nicht abzählbar

- da M abzählbar ist, existiert eine Abzählung g von M (o.B.d.A. ohne Wiederholungen)
- mit $g(i) = m_i$ lässt sich M schreiben als $M = \{m_0, m_1, m_2, \dots\}$ mit $m_i \neq m_j$ für $i \neq j$

69/95

Turingmaschinen

Annahme: $\{f \mid f : M \rightarrow \{0, 1\}\}$ ist abzählbar

- d.h. die Funktionen lassen sich aufzählen f_0, f_1, f_2, \dots
- dann können wir die folgende Tabelle aufstellen:

	m_0	m_1	m_2	\dots
f_0	$f_0(m_0)$	$f_0(m_1)$	$f_0(m_2)$	\dots
f_1	$f_1(m_0)$	$f_1(m_1)$	$f_1(m_2)$	\dots
f_2	$f_2(m_0)$	$f_2(m_1)$	$f_2(m_2)$	\dots
\vdots	\vdots	\vdots	\vdots	\vdots

- definiere die Funktion $g : M \rightarrow \{0, 1\}$ als $g(m_i) = 1 - f_i(m_i)$
- g kommt aber in der Aufzählung nicht vor:
- denn angenommen $g = f_j$
- dann folgt $g(m_j) = f_j(m_j)$ im Widerspruch zu $g(m_j) = 1 - f_j(m_j)$. \square

70/95

Turingmaschinen

Damit wissen wir folgendes:

- die Menge der $\{0, 1\}$ -Folgen ist abzählbar
- die DTM's können wir über ihre Kodierungen als Teilmenge der $\{0, 1\}$ -Folgen auffassen
- damit ist die Menge der DTM's abzählbar
- die berechenbaren Funktionen $f : \{0, 1\}^* \rightarrow \{0, 1\}$ können wir wiederum als Teilmenge der DTM's auffassen
- damit ist die Menge der berechenbaren $\{0, 1\}$ -wertigen Funktionen abzählbar
- die Menge aller Funktionen $f : \{0, 1\}^* \rightarrow \{0, 1\}$ ist dagegen überabzählbar

71/95

Turingmaschinen

Damit haben wir die folgende Aussage bewiesen:

Satz

Es gibt überabzählbar viele Funktionen $f : \{0, 1\}^* \rightarrow \{0, 1\}$, von denen nur abzählbar viele berechenbar sind. \square

Der Beweis ist analog zur Diagonalisierungsmethode von Cantor zum Nachweis der Überabzählbarkeit der reellen Zahlen.

72/95

Turingmaschinen

- die Diagonalisierung liefert eine (abstrakt definierte) Funktion, die nicht berechenbar ist
- wir wollen jetzt versuchen, konkrete nicht berechenbare Funktionen zu finden
- wir haben gesehen, dass wir mit Hilfe der Syntaxanalyse entscheiden können, ob ein $w \in \{0, 1\}^*$ die Kodierung eines syntaktisch korrekten DTM-Programms ist
- wir wenden uns jetzt der Frage zu, ob wir einem DTM-Programm ansehen können, ob es bei gegebenem Input hält

73/95

Turingmaschinen

Unter dem **speziellen Halteproblem** oder **Selbstanwendungsproblem** verstehen wir die Sprache

$$K = \{w \in \{0, 1\}^* \mid M_w \text{ angesetzt auf } w \text{ hält}\}.$$

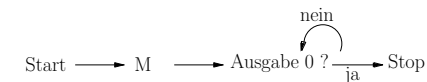
Satz

Das spezielle Halteproblem ist nicht entscheidbar.

Beweis:

Annahme: K ist entscheidbar.

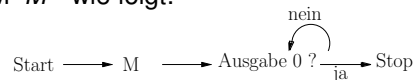
- dann ist χ_K berechenbar
- sei M eine DTM, die χ_K berechnet
- konstruiere eine DTM M' wie folgt:



74/95

Turingmaschinen

- konstruiere eine DTM M' wie folgt:



- d.h. es gilt: M' hält $\iff M$ hält mit Ausgabe 0
- sei w' die binäre Kodierung von M'
- dann folgt:

$$\begin{aligned}
 & M' \text{ angesetzt auf } w' \text{ hält} \\
 \iff & M \text{ angesetzt auf } w' \text{ gibt 0 aus} \\
 \iff & \chi_K(w') = 0 \\
 \iff & w' \notin K \\
 \iff & M_{w'} = M' \text{ angesetzt auf } w' \text{ hält nicht} \quad \zeta \quad \square
 \end{aligned}$$

75/95

Turingmaschinen

- seien $A \subseteq \Sigma^*$ und $B \subseteq \Gamma^*$ Sprachen
- dann heißt A **auf B reduzierbar** ($A \preceq B$), falls es eine totale berechenbare Funktion $f: \Sigma^* \rightarrow \Gamma^*$ gibt mit:

$$x \in A \iff f(x) \in B \quad \text{für alle } x \in \Sigma^*$$

Lemma

„ \preceq “ ist transitiv.

Beweis:

- sei L reduzierbar auf L' mittels der berechenbaren Funktion f
- sei L' reduzierbar auf L'' mittels der berechenbaren Funktion g
- dann gilt: $x \in L \iff f(x) \in L'$ und $y \in L' \iff g(y) \in L''$
- damit ist $x \in L \iff g(f(x)) \in L''$.
- $g \circ f$ ist berechenbar (schalte die beiden Programme hintereinander)
- damit folgt $L \preceq L''$. □

76/95

Turingmaschinen

Entscheidbarkeit setzt sich auf reduzierbare Sprachen fort:

Lemma

Aus $A \preceq B$ und B (semi-)entscheidbar folgt auch A (semi-)entscheidbar.

Beweis:

- sei f die Funktion, die A auf B reduziert
- da B entscheidbar, ist χ_B berechenbar
- es gilt

$$\chi_A(x) = \begin{cases} 1 & \text{falls } x \in A \\ 0 & \text{falls } x \notin A \end{cases} = \begin{cases} 1 & \text{falls } f(x) \in B \\ 0 & \text{falls } f(x) \notin B \end{cases} = \chi_B(f(x))$$

- also ist χ_A berechenbar und somit A entscheidbar
- für den Fall der Semientscheidbarkeit ersetze χ durch χ' und 0 durch „undefiniert“.

□

77/95

Turingmaschinen

Wir benutzen das letzte Lemma in umgekehrter Richtung:

- wir wissen bereits, dass A unentscheidbar ist
- wir zeigen $A \preceq B$ und schließen, dass B unentscheidbar ist

Das (allgemeine) **Halteproblem** ist die Sprache

$$H = \{w\#x \mid M_w \text{ angesetzt auf } x \text{ hält}\}.$$

Satz

Das Halteproblem ist nicht entscheidbar.

Beweis:

- sei K das spezielle Halteproblem
- nach der obigen Bemerkung reicht es zu zeigen: $K \preceq H$
- wähle dafür $f(w) = w\#w$. Dann gilt:

$$w \in K \Leftrightarrow f(w) \in H. \quad \square$$

78/95

Turingmaschinen

Bemerkung

- H ist aber semi-entscheidbar:
- simuliere M_w angesetzt auf x , halte und akzeptiere, falls M_w hält.

Auch eine speziellere Variante des Halteproblems ist nicht entscheidbar:

Satz

Das Halteproblem auf leerem Band ist nicht entscheidbar.

79/95

Turingmaschinen

Beweis:

- sei $H_e = \{w \mid M_w \text{ angesetzt auf das leere Band hält}\}$
- wir zeigen $H \preceq H_e$
- dazu ordnen wir jedem Wort $w\#x$ eine Turingmaschine $M_{w,x}$ zu, die wie folgt arbeitet:
 - sie ignoriert ihren Input (d.h. sie startet mit dem leeren Band)
 - sie löscht ihren Input
 - sie schreibt x aufs Band
 - und simuliert danach M_w angesetzt auf x
- sei $f(w, x)$ die Kodierung von $M_{w,x}$
- f ist berechenbar (im Wesentlichen durch die Simulation gegeben)
- wir ergänzen f (beliebig) zu einer totalen Funktion
- dann gilt:

$$\begin{aligned} w\#x \in H &\Leftrightarrow M_w \text{ angesetzt auf } x \text{ hält} \\ &\Leftrightarrow M_{w,x} \text{ angesetzt auf leerem Band hält} \\ &\Leftrightarrow f(w\#x) \in H_e. \quad \square \end{aligned}$$

80/95

Turingmaschinen

- der letzte Satz ist Spezialfall ein allgemeineren Aussage
- sei \mathcal{R} die Menge aller berechenbaren Funktionen.
- eine Teilmenge $E \subseteq \mathcal{R}$ heie **Eigenschaft** der berechenbaren Funktionen
- E ist **trivial**, wenn $E = \emptyset$ oder $E = \mathcal{R}$

Satz (Rice)

Jede nicht-triviale Eigenschaft von berechenbaren Funktionen ist nicht entscheidbar.

81/95

Turingmaschinen

Beweis:

- sei $\Omega \in \mathcal{R}$ die berall undefinierte Funktion
- dann gilt entweder $\Omega \in \mathcal{S}$ oder $\Omega \notin \mathcal{S}$
- und wir unterscheiden die beiden Flle

Fall 1 $\Omega \in \mathcal{S}$:

- wegen $\mathcal{S} \neq \mathcal{R}$ gibt es eine Funktion $q \in \mathcal{R} \setminus \mathcal{S}$
- sei Q eine Turingmaschine, die q berechnet
- wir ordnen jedem Wort $w \in \{0, 1\}^*$ eine DTM M zu, die wie folgt arbeitet:
 - gestartet mit Eingabe y ignoriert M die Eingabe zunchst und verhlt sich wie M_w gestartet mit dem leeren Band
 - falls die Berechnung stoppt, verhlt sich M danach wie Q gestartet mit y

82/95

Turingmaschinen

- M berechnet also folgende Funktion:

$$g = \begin{cases} \Omega, & \text{falls } M_w \text{ auf leerem Band nicht stoppt,} \\ q, & \text{sonst.} \end{cases}$$

- Die Abbildung f , die w eine Kodierung von M zuordnet, ist total und berechenbar, und es gilt:

$$\begin{aligned} w \in H_\varepsilon &\Rightarrow M_w \text{ angesetzt auf das leere Band stoppt} \\ &\Rightarrow M = M_{f(w)} \text{ berechnet die Funktion } q \\ &\Rightarrow \text{die von } M_{f(w)} \text{ berechnete Funktion liegt nicht in } \mathcal{S} \\ &\Rightarrow f(w) \notin \mathcal{C}(\mathcal{S}) \end{aligned}$$

$$\begin{aligned} w \notin H_\varepsilon &\Rightarrow M_w \text{ angesetzt auf das leere Band stoppt nicht} \\ &\Rightarrow M = M_{f(w)} \text{ berechnet die Funktion } \Omega \\ &\Rightarrow \text{die von } M_{f(w)} \text{ berechnete Funktion liegt in } \mathcal{S} \\ &\Rightarrow f(w) \in \mathcal{C}(\mathcal{S}) \end{aligned}$$

83/95

Turingmaschinen

- d.h. die Funktion f liefert eine Reduktion von $\overline{H_\varepsilon}$ nach $\mathcal{C}(\mathcal{S})$
- d.h. $\overline{H_\varepsilon} \preceq \mathcal{C}(\mathcal{S})$
- damit folgt:

$$\begin{aligned} H_\varepsilon \text{ ist nicht entscheidbar} &\Rightarrow \overline{H_\varepsilon} \text{ ist nicht entscheidbar} \\ &\Rightarrow \mathcal{C}(\mathcal{S}) \text{ ist nicht entscheidbar} \end{aligned}$$

Fall 2 $\Omega \notin \mathcal{S}$

- wir knnen hier analog zeigen: $H_\varepsilon \preceq \mathcal{C}(\mathcal{S})$. □

84/95

Turingmaschinen

Beispiel

- es ist nicht möglich, zu entscheiden, ob eine gegebene Turingmaschine eine konstante Funktion berechnet:

- wähle

$$\mathcal{S} = \{f \in \mathcal{R} \mid f \text{ ist eine konstante Funktion}\}$$

- dann ist

$$\mathcal{C}(\mathcal{S}) = \{w \mid M_w \text{ berechnet eine konstante Funktion}\}$$

nicht entscheidbar.

weitere Beispiele

- $H_{\text{überhaupt}}$: hält eine DTM für irgendein Wort als Input?
- H_{immer} : akzeptiert eine DTM alle Inputwörter?

85/95

Turingmaschinen

Bemerkung

- es gibt „noch unlösbarere“ Probleme als das Halteproblem, z.B. das **Äquivalenzproblem für Turingmaschinen**:

$$H_{\bar{a}} = \{u\#v \mid M_u \text{ berechnet dieselbe Funktion wie } M_v\}$$

- wir reduzieren H_{immer} auf $H_{\bar{a}}$:
- sei dazu M^0 eine (triviale) DTM, die immer akzeptiert
- die Reduktion f ordnet einer beliebigen DTM M das Paar (M, M^0) zu
- f ist berechenbar und es gilt:

$$M \in H_{\text{immer}} \iff (M, M^0) \in H_{\bar{a}}$$

$$M \notin H_{\text{immer}} \iff (M, M^0) \notin H_{\bar{a}}$$

- damit kann $H_{\bar{a}}$ nicht entscheidbar sein. □

86/95

Turingmaschinen

Folgerung:

- zur Überprüfung eines Programms im Programmierpraktikum könnten man eine Musterlösung schreiben
- wir könnten jedoch nicht feststellen, ob ein abgegebenes Programm das gleiche macht, wie die Musterlösung
- entsprechend könnten wir nicht prüfen, ob ein Programm die gleiche Funktion hat, wie ein bereits bekannter Virus

87/95

Turingmaschinen

- der Satz von Rice lässt sich auch für DTM's anstelle von Funktionen formulieren
- jeder berechenbaren Funktion entspricht eine Menge von äquivalenten DTM's (die die Funktion berechnen)
- eine **semantische Eigenschaft** eines Programms ist eine Eigenschaft, die allen äquivalenten Programmen gemein ist
- **Beispiele:**
 - bei Eingabe 0 halten
 - immer halten
 - zieht die Wurzel aus dem Input
 - berechnet die Funktion f
- nicht semantische Eigenschaften:
 - verwendet keinen Sprungbefehl
 - enthält mehr als 19 Anweisungen

88/95

Turingmaschinen

Satz (Rice)

Eine semantische Eigenschaft E von DTMs ist genau dann entscheidbar, wenn alle DTMs diese Eigenschaft haben oder keine.

Beweis:

- haben alle DTMs die Eigenschaft E , so ist E mittels Syntaxanalyse zu entscheiden
- gleiches gilt, falls $E = \emptyset$
- d.h. in diesen beiden Fällen ist E zu entscheiden
- umgekehrt ist die Fragestellung äquivalent dazu, E für die zugehörigen berechenbaren Funktionen zu entscheiden
- nach dem Satz von Rice geht dies nur für triviale Eigenschaften \square

89/95

Turingmaschinen

- für DTM's ist das Halteproblem nicht entscheidbar
- wenn wir uns auf platzbeschränkte Turingmaschinen beschränken, können wir jedoch eine ähnliche Fragestellung beantworten
- das folgende Lemma beschreibt, wann eine Turingmaschine nicht hält:

Lemma

Sei M eine $s(n)$ -platzbeschränkte Turingmaschine über dem Alphabet A . Hält M gestartet mit x nach höchstens

$$|Z| \cdot s(|x|) \cdot |A|^{s(|x|)}$$

Schritten nicht, so hält M überhaupt nicht.

90/95

Turingmaschinen

Lemma

Sei M eine $s(n)$ -platzbeschränkte Turingmaschine über dem Alphabet A . Hält M gestartet mit x nach höchstens

$$|Z| \cdot s(|x|) \cdot |A|^{s(|x|)}$$

Schritten nicht, so hält M überhaupt nicht.

Beweis:

- für jede Konfiguration $w_1 z_i w_2$ gilt $|w_1 w_2| \leq s(|x|)$
- es gibt höchstens $|A|^{s(|x|)}$ viele verschiedene Bandinschriften
- es gibt höchstens $s(|x|)$ Kopfpositionen, $|Q|$ Zustände
- somit gibt es höchstens $|Q|s(|x|)|A|^{s(|x|)}$ verschiedene Konfigurationen
- d.h. nach spätestens $|Q|s(|x|)|A|^{s(|x|)}$ Schritten wird eine Konfiguration zum zweiten Male erreicht
- ab dann gerät das Programm in eine Endlosschleife. \square

91/95

Turingmaschinen

- bisher haben wir Probleme betrachtet, die nach dem Halten oder Nichthalten von Turingmaschinen fragen
- unter den nichtentscheidbaren Problemen gibt es auch natürlichere Fragestellungen

Postisches Korrespondenzproblem

- **gegeben:** eine Menge von Paaren binärer Wörter $\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$
- **gesucht:** Indices $i_j, j = 1, \dots, k$ mit $1 \leq i_j \leq n$, so dass $x_{i_1} x_{i_2} \dots x_{i_k} = y_{i_1} y_{i_2} \dots y_{i_k}$

92/95

Turingmaschinen

Bemerkungen:

- es ist zugelassen, dass Indizes der Eingabefolge mehrmals auftauchen
- die Anzahl k der zu bestimmenden Paare ist a priori nicht beschränkt
- es sind Umsortierungen erlaubt, d.h. $i_{j+1} < i_j$

Beispiel:

- die Paare $(111, 1), (1110, 1110111), (01, 101)$
- haben die Lösung $2, 1, 1, 3$:
- $x_2 x_1 x_1 x_3 = 111011111101$
- $y_2 y_1 y_1 y_3 = 111011111101$

93/95

Turingmaschinen

Satz

Das Postsche Korrespondenzproblem ist nicht entscheidbar. \square

- der Beweis beruht auf einer Reduktion des Halteproblems auf das Korrespondenzproblem
- einer DTM M und einer Eingabe w werden dazu Paare (x_i, y_i) zugeordnet
- in den y -Komponenten wird die Berechnung von M simuliert
- in den x -Komponenten wird die Berechnung um eine Konfigurationen verschoben simuliert
- wenn M hält, kann diese Verschiebung rückgängig gemacht werden.

94/95

Turingmaschinen

- wir werden uns später mit der Erfüllbarkeit von Booleschen Formeln befassen
- dazu betrachten wir die folgende Sprache

$$\text{SAT} = \{B : B \text{ ist eine erfüllbare KNF}\}$$

- ähnliche Fragestellungen sind:

$$\text{TAUTOLOGIE} = \{B : B \text{ ist für alle Belegungen wahr}\}$$

$$\text{FOLGERUNG} = \{(A, B) : A \text{ ist eine Menge von Booleschen Formeln, } B \text{ folgt aus } A\}$$

- d.h. wann immer alle Aussagen in A wahr sind, soll auch B wahr sein
- alle diese Fragen können offensichtlich entschieden werden, indem man alle Belegungen testet
- die Situation ändert sich drastisch, wenn man von der Aussagen- zur Prädikatenlogik übergeht
- die entsprechenden Fragestellungen sind dann nicht mehr entscheidbar (und nur noch semi-entscheidbar).

95/95