Grundbegriffe der Informatik

Einheit 16: Turingmaschinen

Prof. Dr. Tanja Schultz

Karlsruher Institut für Technologie, Fakultät für Informatik

Wintersemester 2012/2013

Überblick

Eine technische Vorbemerkung

Turingmaschinen
- Berechnungen
- Eingaben für Turingmaschinen
- Ergebnisse von Turingmaschinen

Berechnungskomplexität
- Komplexitätsmaße
- Komplexitätsklassen

Unentscheidbare Probleme
- Codierungen von Turingmaschinen
- Das Halteproblem
- Die Busy-Beaver-Funktion

Überblick

Eine technische Vorbemerkung

Turingmaschinen
- Berechnungen
- Eingaben für Turingmaschinen
- Ergebnisse von Turingmaschinen

Berechnungskomplexität
- Komplexitätsmaße
- Komplexitätsklassen

Unentscheidbare Probleme
- Codierungen von Turingmaschinen
- Das Halteproblem
- Die Busy-Beaver-Funktion

Erinnerung: Partielle Funktionen

- 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

Überblick

Eine technische Vorbemerkung

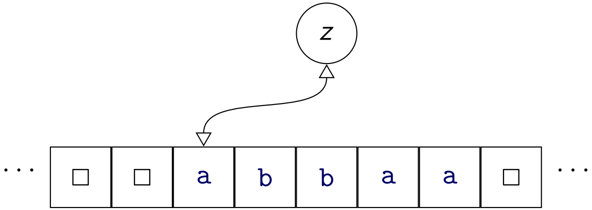
Turingmaschinen
- Berechnungen
- Eingaben für Turingmaschinen
- Ergebnisse von Turingmaschinen

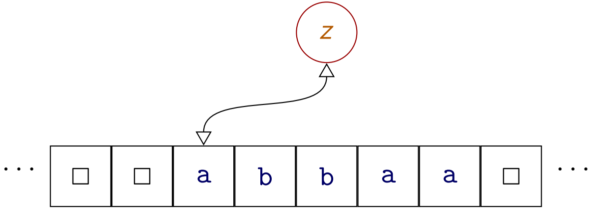
Berechnungskomplexität
- Komplexitätsmaße
- Komplexitätsklassen

Unentscheidbare Probleme
- Codierungen von Turingmaschinen
- Das Halteproblem
- Die Busy-Beaver-Funktion

- eingeführt von Alan Turing (1912 – 1954)
  `http://www.turing.org.uk/turing/index.html`
- *„On computable numbers, with an application to the Entscheidungsproblem"*
  Proceedings of the London Mathematical Society **42**, 1936, S. 230–265.

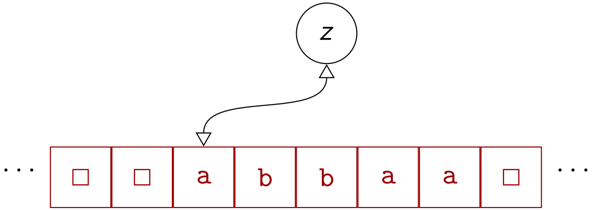
- *„The informal arguments [. . .] are as lucid and convincing now as they were then. [. . .] the best introduction to the subject [. . .] superior piece of expository writing."*
  `http://www.scholarpedia.org/article/Turing_machine`


z
... □ □ a b b a a □ ...



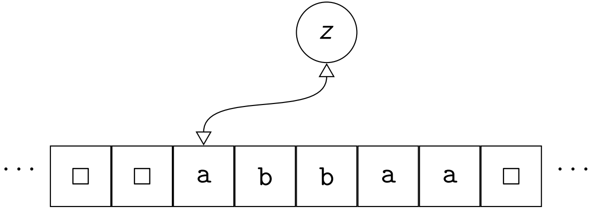
- Steuereinheit: endliche Zustandsmenge Z

Turingmaschinen: das Bild

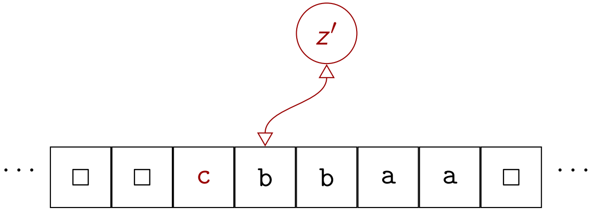


- endliche Zustandsmenge Z
- Band, in Felder unterteilt:
  beschriftet mit Symbolen aus Bandalphabet X

Turingmaschinen: das Bild



- endliche Zustandsmenge Z
- Bandalphabet X
- Schritt:
  - neue Feldbeschriftung $g(z, \mathtt{a})$
  - neuer Zustand $f(z, \mathtt{a})$
  - Kopfbewegung $m(z, \mathtt{a})$

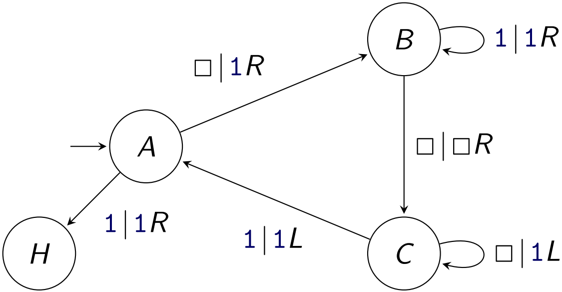


- endliche Zustandsmenge Z
- Bandalphabet X
- Schritt:
  - neue Feldbeschriftung $g(z, \mathtt{a}) = \mathtt{c}$
  - neuer Zustand $f(z, \mathtt{a}) = z'$
  - Kopfbewegung $m(z, \mathtt{a}) = +1$

Turingmaschinen: Formalisierung

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

- eine Zustandsmenge Z
- einen Anfangszustand $z_0 \in Z$
- ein Bandalphabet X
  - meist mit Blanksymbol $\square$
- eine partielle Zustandsüberführungsfunktion
  $f : Z \times X \dashrightarrow Z$
- eine partielle Ausgabefunktion
  $g : Z \times X \dashrightarrow X$ und
- eine partielle Bewegungsfunktion
  $m : Z \times X \dashrightarrow \{-1, 0, 1\}$ oder $\{L, 0, R\}$
- f, g, m für die gleichen Paare
  $(z, x) \in Z \times X$ definiert bzw. nicht definiert



	A	B	C	H
$\square$	$B, 1, R$	$C, \square, R$	$C, 1, L$	
1	$H, 1, R$	$B, 1, R$	$A, 1, L$	

Beispielberechnung (1)

	A	B	C	H
$\square$	$B, 1, R$	$C, \square, R$	$C, 1, L$	
1	$H, 1, R$	$B, 1, R$	$A, 1, L$	

			A				
$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$

Beispielberechnung (1)

	A	B	C	H
$\square$	$B, 1, R$	$C, \square, R$	$C, 1, L$	
1	$H, 1, R$	$B, 1, R$	$A, 1, L$	

			A				
$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$

Beispielberechnung (1)

	A	B	C	H
$\square$	$B, 1, R$	$C, \square, R$	$C, 1, L$	
1	$H, 1, R$	$B, 1, R$	$A, 1, L$	

			A				
$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$

	A	B	C	H
$\square$	$B, 1, R$	$C, \square, R$	$C, 1, L$	
1	$H, 1, R$	$B, 1, R$	$A, 1, L$	

			A				
$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$

	A	B	C	H
$\square$	$B, 1, R$	$C, \square, R$	$C, 1, L$	
1	$H, 1, R$	$B, 1, R$	$A, 1, L$	

			A				
$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$
				B			
$\square$	$\square$	$\square$	1	$\square$	$\square$	$\square$	$\square$

Beispielberechnung (3)

	A	B	C	H
$\square$	$B, 1, R$	$C, \square, R$	$C, 1, L$	
1	$H, 1, R$	$B, 1, R$	$A, 1, L$	

			A				
$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$
				B			
$\square$	$\square$	$\square$	1	$\square$	$\square$	$\square$	$\square$

Beispielberechnung (4)

	A	B	C	H
$\square$	$B, 1, R$	$C, \square, R$	$C, 1, L$	
1	$H, 1, R$	$B, 1, R$	$A, 1, L$	

			A				
$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$
				B			
$\square$	$\square$	$\square$	1	$\square$	$\square$	$\square$	$\square$

Beispielberechnung (5)

	A	B	C	H
$\square$	$B, 1, R$	$C, \square, R$	$C, 1, L$	
1	$H, 1, R$	$B, 1, R$	$A, 1, L$	

			A				
$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$
				B			
$\square$	$\square$	$\square$	1	$\square$	$\square$	$\square$	$\square$
					C		
$\square$	$\square$	$\square$	1	$\square$	$\square$	$\square$	$\square$

Turingmaschinen: Konfigurationen

- *Konfiguration:* „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 $b : \mathbb{Z} \to X$
  - aktuelle Position $p \in \mathbb{Z}$ des Kopfes
- $\mathcal{C}_T$: Menge aller Konfigurationen von T

Turingmaschinen: „überschaubare“ Bandbeschriftungen

- 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
  - $\Box \in X$ geschrieben
  - hier: Bei allen vorkommenden Bandbeschriftungen sind nur endlich viele Felder nicht mit $\Box$ beschriftet.

Überblick

Eine technische Vorbemerkung

Turingmaschinen
- Berechnungen
- Eingaben für Turingmaschinen
- Ergebnisse von Turingmaschinen

Berechnungskomplexität
- Komplexitätsmaße
- Komplexitätsklassen

Unentscheidbare Probleme
- Codierungen von Turingmaschinen
- Das Halteproblem
- Die Busy-Beaver-Funktion

Ein Schritt einer Turingmaschine

- 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
- $\Delta_1 : \mathcal{C}_T \dashrightarrow \mathcal{C}_T$

Längere Beispielberechnung von BB3

			A				
□	□	□	□	□	□	□	□
				B			
□	□	□	1	□	□	□	□
					C		
□	□	□	1	□	□	□	□
				C			
□	□	□	1	□	1	□	□
			C				
□	□	□	1	1	1	□	□
		A					
□	□	□	1	1	1	□	□
			B				
□	□	1	1	1	1	□	□
				B			
□	□	1	1	1	1	□	□

				B			
□	□	1	1	1	1	□	□
					B		
□	□	1	1	1	1	□	□
						B	
□	□	1	1	1	1	□	□
							C
□	□	1	1	1	1	□	□
						C	
□	□	1	1	1	1	□	1
					C		
□	□	1	1	1	1	1	1
				A			
□	□	1	1	1	1	1	1
					H		
□	□	1	1	1	1	1	1

Berechnungen und Endkonfigurationen

- c ist Endkonfiguration, falls $\Delta_1(c)$ nicht definiert ist.
- *endliche Berechnung*:
  - endliche Folge von Konfigurationen $(c_0, c_1, c_2, \ldots, c_t)$, 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 $(c_0, c_1, c_2, \ldots)$, 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 „wegkonstruieren"? ...

Rechnen bis zur Endkonfiguration

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

$$\Delta_0 = \mathrm{I}$$
$$\Delta_{t+1} = \Delta_1 \circ \Delta_t$$

- Zu jeder Konfiguration c gibt es genau eine Berechnung, die mit c startet und möglichst lange dauert.
- 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

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

Überblick

Eine technische Vorbemerkung

Turingmaschinen
- Berechnungen
- Eingaben für Turingmaschinen
- Ergebnisse von Turingmaschinen

Berechnungskomplexität
- Komplexitätsmaße
- Komplexitätsklassen

Unentscheidbare Probleme
- Codierungen von Turingmaschinen
- Das Halteproblem
- Die Busy-Beaver-Funktion

analog zu endlichen Automaten

- Berechung von Funktionen
- Erkennung formaler Sprachen
  - Turingmaschinenakzeptoren
  - Entscheidungsprobleme

Eingaben und Anfangskonfigurationen

- *Eingabealphabet* $A \subset X \setminus \{\square\}$ spezifiziert ist.
  - Blanksymbol nicht dabei
- *Anfangskonfiguration* $c_0(w) = (z, b, p)$ für Eingabe $w \in A^*$
  - $z = z_0$
  - $b = b_w : \mathbb{Z} \to X$

$$b_w(i) = \begin{cases} \square & \text{falls } i < 0 \vee i \geq |w| \\ w(i) & \text{falls } 0 \leq i \wedge i < |w| \end{cases}$$

  - $p = 0$
    - Kopf auf dem ersten Eingabesymbol (falls $w \neq \varepsilon$)
- Anfangskonfiguration bei Eingabe evtl. mehrerer Zahlen
  - geeignet harmlos, z. B.
  - □□□`1011`□□ oder
  - □□□`[1011][101]`□□ oder
  - □□□`[1011,101]`□□ oder
  - ...

Überblick

Eine technische Vorbemerkung

Turingmaschinen
- Berechnungen
- Eingaben für Turingmaschinen
- Ergebnisse von Turingmaschinen

Berechnungskomplexität
- Komplexitätsmaße
- Komplexitätsklassen

Unentscheidbare Probleme
- Codierungen von Turingmaschinen
- Das Halteproblem
- Die Busy-Beaver-Funktion

Ausgaben

analog zu endlichen Automaten

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

- Turingmaschinenakzeptor T:
  - 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
    die von der Turingmaschine akzeptierte Sprache

Aufzählbare und entscheidbare Sprachen

- 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.
- Entscheidbarkeit ist eine stärkere Forderung

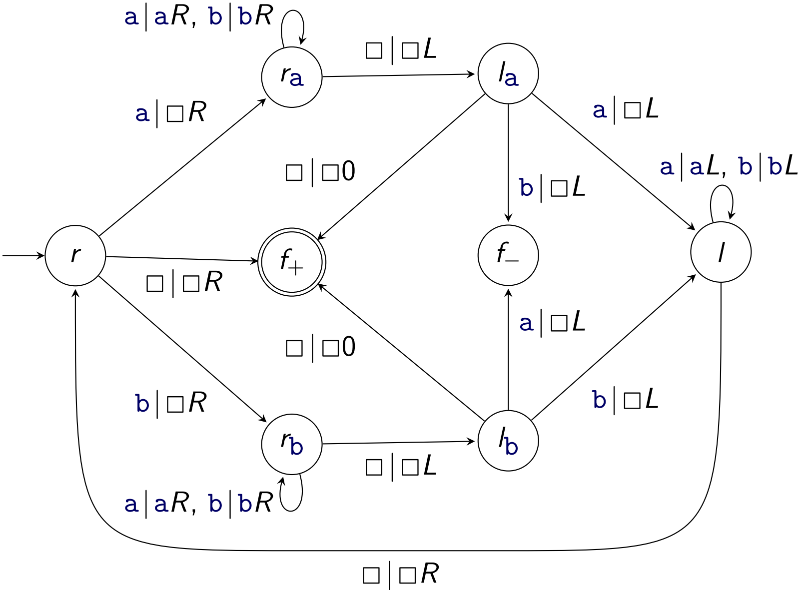
		r				
$\square$	$\square$	a	b	b	a	$\square$
			r_a			
$\square$	$\square$	$\square$	b	b	a	$\square$
				r_a		
$\square$	$\square$	$\square$	b	b	a	$\square$
					r_a	
$\square$	$\square$	$\square$	b	b	a	$\square$
						r_a
$\square$	$\square$	$\square$	b	b	a	$\square$
					l_a	
$\square$	$\square$	$\square$	b	b	a	$\square$
				l		
$\square$	$\square$	$\square$	b	b	$\square$	$\square$
			l			
$\square$	$\square$	$\square$	b	b	$\square$	$\square$

		l				
$\square$	$\square$	$\square$	b	b	$\square$	$\square$
			r			
$\square$	$\square$	$\square$	b	b	$\square$	$\square$
				r_b		
$\square$	$\square$	$\square$	$\square$	b	$\square$	$\square$
					r_b	
$\square$	$\square$	$\square$	$\square$	b	$\square$	$\square$
				l_b		
$\square$	$\square$	$\square$	$\square$	b	$\square$	$\square$
			l			
$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$
				r		
$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$
					f_+	
$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$

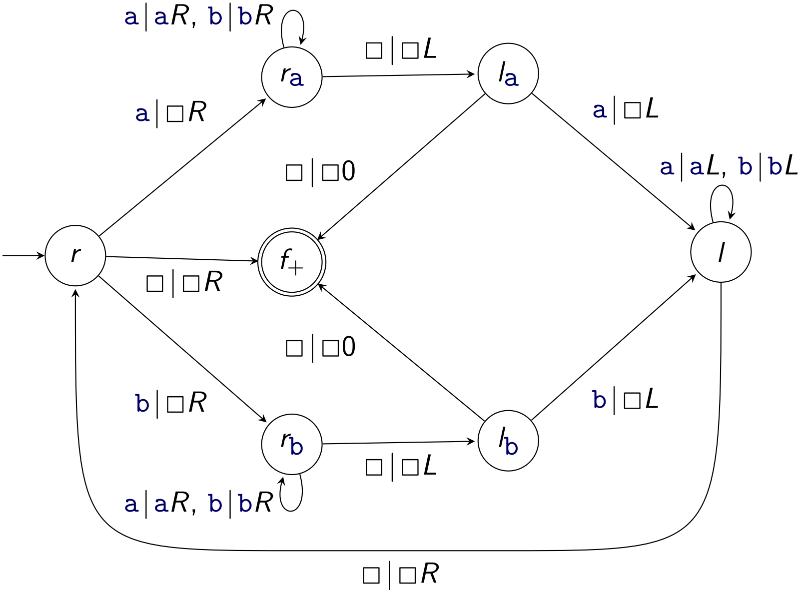
Palindromerkennung: Beispielberechnung

		r				
□	□	a	b	b	a	□
			r_a			
□	□	□	b	b	a	□
				r_a		
□	□	□	b	b	a	□
					r_a	
□	□	□	b	b	a	□
						r_a
□	□	□	b	b	a	□
					l_a	
□	□	□	b	b	a	□
				l		
□	□	□	b	b	□	□
			l			
□	□	□	b	b	□	□

		l				
□	□	□	b	b	□	□
			r			
□	□	□	b	b	□	□
				r_b		
□	□	□	□	b	□	□
					r_b	
□	□	□	□	b	□	□
				l_b		
□	□	□	□	b	□	□
			l			
□	□	□	□	□	□	□
				r		
□	□	□	□	□	□	□
					f_+	
□	□	□	□	□	□	□



a|aR, b|bR
□|□L
a|□R
a|□L
□|□0
a|aL, b|bL
b|□L
r
f₊
f₋
l
□|□R
a|□L
□|□0
b|□R
b|□L
□|□L
a|aR, b|bR
□|□R


a|aR, b|bR
□|□L
a|□R
a|□L
□|□0
a|aL, b|bL
r
f+
l
□|□R
□|□0
b|□R
b|□L
□|□L
a|aR, b|bR
□|□R

Was ist wichtig

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

Überblick

Eine technische Vorbemerkung

Turingmaschinen
- Berechnungen
- Eingaben für Turingmaschinen
- Ergebnisse von Turingmaschinen

Berechnungskomplexität
- Komplexitätsmaße
- Komplexitätsklassen

Unentscheidbare Probleme
- Codierungen von Turingmaschinen
- Das Halteproblem
- Die Busy-Beaver-Funktion

Achtung

- Annahme in diesem Abschnitt: alle Turingmaschinen halten für jede Eingabe.
- Versprechen: Für die Fragestellungen in diesem Abschnitt (Komplexitätstheorie) ist das in Ordnung
- Vorsicht: Für die Fragestellungen im Abschnitt über unentscheidbare Probleme ist das *nicht* mehr in Ordnung.

Überblick

Eine technische Vorbemerkung

Turingmaschinen
- Berechnungen
- Eingaben für Turingmaschinen
- Ergebnisse von Turingmaschinen

Berechnungskomplexität
- Komplexitätsmaße
- Komplexitätsklassen

Unentscheidbare Probleme
- Codierungen von Turingmaschinen
- Das Halteproblem
- Die Busy-Beaver-Funktion

Zeitkomplexität

- Beurteilung des Zeitbedarfs einer Turingmaschine
- definiere

$$\mathrm{time}_T : A^+ \to \mathbb{N}_+$$
$$\mathrm{Time}_T : \mathbb{N}_+ \to \mathbb{N}_+$$

  wie folgt:

$$\mathrm{time}_T(w) = \text{dasjenige } t \text{ mit } \Delta_t(c_0(w)) = \Delta_*(c_0(w))$$
$$\mathrm{Time}_T(n) = \max\{\mathrm{time}_T(w) \mid w \in A^n\}$$

- Time_T heißt *Zeitkomplexität* der Turingmaschine
  - der schlimmste Fall in Abhängigkeit von der Länge der Eingabe
- Zeitkomplexität einer Turingmaschine *polynomiell*, wenn es ein Polynom $p(n)$ gibt mit $\mathrm{Time}_T(n) \in \mathrm{O}(p(n))$.

Zeitkomplexität der Beispielturingmaschine für Palindromerkennung

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

  insgesamt also

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

- Zeitaufwand für Wörter der Länge $n=1$ auch gerade $2n+1$
- also

$$\mathrm{Time}(n) \in \mathrm{O}\left(n^2\right)$$

  d. h. polynomielle, genauer quadratische, Zeitkomplexität

- Beurteilung des Speicherplatzbedarfs einer Turingmaschine
- definiere

$$\mathrm{space}_T(w) : A^+ \to \mathbb{N}_+$$
$$\mathrm{Space}_T(n) : \mathbb{N}_+ \to \mathbb{N}_+$$

wie folgt

$$\begin{aligned}\mathrm{space}_T(w) &= \text{die Anzahl der Felder, die während der} \\ &\qquad \text{Berechnung für Eingabe } w \text{ benötigt werden} \\ \mathrm{Space}_T(n) &= \max\{\mathrm{space}_T(w) \mid w \in A^n\}\end{aligned}$$

- Ein Feld wird „benötigt“, wenn es anfangs ein Eingabesymbol enthält oder einmal vom Kopf der TM besucht wird.
- Space_T heißt die *Raumkomplexität* oder *Platzkomplexität* der Turingmaschine

Raumkomplexität der Beispielturingmaschine für Palindromerkennung

- benötigte Felder:
  - n Felder mit den Eingabesymbolen
  - ein weiteres Feld rechts davon
- polynomieller, nämlich linearer, Platzbedarf

$$\mathrm{Space}(n) = n + 1 \in \Theta(n)$$

Zusammenhänge zwischen Zeit- und Raumkomplexität (1)

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

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

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

Zusammenhänge zwischen Zeit- und Raumkomplexität (2)

umgekehrt von Raum- zu Zeitkomplexität:

- Auf k Feldern können $(|X|-1)^k$ „interessante“ verschiedene Inschriften stehen.
- Es gibt Turingmaschinen mit
  - polynomieller Raumkomplexität aber
  - exponentieller Zeitkomplexität.

Überblick

Eine technische Vorbemerkung

Turingmaschinen
- Berechnungen
- Eingaben für Turingmaschinen
- Ergebnisse von Turingmaschinen

Berechnungskomplexität
- Komplexitätsmaße
- Komplexitätsklassen

Unentscheidbare Probleme
- Codierungen von Turingmaschinen
- Das Halteproblem
- Die Busy-Beaver-Funktion

Komplexitätsklassen

- Eine *Komplexitätsklasse* ist eine Menge von *Problemen*.
  - hier nur formale Sprachen (Entscheidungsprobleme)

- Charakterisierung durch Beschränkung der zur Verfügung stehenden Ressourcen, also z. B. Schranken für Zeitkomplexität oder Raumkomplexität (oder beides)

- Beispiel: alle formalen Sprachen, die von Turingmaschinen entschieden werden können, bei denen gleichzeitig
  - Zeitkomplexität in $\mathrm{O}\left(n^3\right)$ und
  - Raumkomplexität in $\mathrm{O}\left(n^{3/2} \log n\right)$ ist

  wobei n die Länge des Eingabewortes ist.

Zwei wichtige Komplexitätsklassen

- **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**

Zusammenhang zwischen **P** und **PSPACE**

- polynomielle Laufzeit impliziert polynomiellen Platzbedarf
- Also

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

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

$$\mathbf{P} = \mathbf{PSPACE} \text{ oder } \mathbf{P} \neq \mathbf{PSPACE}\ ?$$

Zusammenhang zwischen **P** und **PSPACE**

- polynomielle Laufzeit impliziert polynomiellen Platzbedarf
- Also

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

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

$$\mathbf{P} = \mathbf{PSPACE} \text{ oder } \mathbf{P} \neq \mathbf{PSPACE}\ ?$$

Zusammenhang zwischen **P** und **PSPACE**

- polynomielle Laufzeit impliziert polynomiellen Platzbedarf
- Also

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

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

$$\mathbf{P} = \mathbf{PSPACE} \text{ oder } \mathbf{P} \neq \mathbf{PSPACE} \, ?$$

Zusammenhang zwischen **P** und **PSPACE**

- polynomielle Laufzeit impliziert polynomiellen Platzbedarf
- Also

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

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

  $\mathbf{P} = \mathbf{PSPACE}$ oder $\mathbf{P} \neq \mathbf{PSPACE}$?

Was ist wichtig

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
- 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

Überblick

Eine technische Vorbemerkung

Turingmaschinen
- Berechnungen
- Eingaben für Turingmaschinen
- Ergebnisse von Turingmaschinen

Berechnungskomplexität
- Komplexitätsmaße
- Komplexitätsklassen

Unentscheidbare Probleme
- Codierungen von Turingmaschinen
- Das Halteproblem
- Die Busy-Beaver-Funktion

Achtung

Ab sofort ist es wieder von Bedeutung, dass Turingmaschinen in „Endlosschleifen“ laufen können.

Überblick

Eine technische Vorbemerkung

Turingmaschinen
- Berechnungen
- Eingaben für Turingmaschinen
- Ergebnisse von Turingmaschinen

Berechnungskomplexität
- Komplexitätsmaße
- Komplexitätsklassen

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 $A = \{\texttt{[}, \texttt{]}, \texttt{0}, \texttt{1}\}$
  - es reicht aber auch $A = \{\texttt{0}, \texttt{1}\}$
  - oder sogar $A = \{\texttt{1}\}$
- 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 $A = \{\texttt{[}, \texttt{]}, \texttt{0}, \texttt{1}\}$
  - es reicht aber auch $A = \{\texttt{0}, \texttt{1}\}$
  - oder sogar $A = \{\texttt{1}\}$
- 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/`

Beispielcodierung (1)

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 $\mathrm{cod}_Z(z)$ für die Codierung von Zustand z.
- Beispiele:
  - $\mathrm{cod}_Z(z_0) =$ `[0000]`
  - $\mathrm{cod}_Z(z_1) =$ `[0001]`
  - $\ldots$

Beispielcodierung (2)

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

Codierung der Symbole:

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

Beispielcodierung (3)

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 $\mathrm{cod}_M(r)$ für die Codierung der Bewegungsrichtung r.
- also
  - $\mathrm{cod}_M(-1) =$ [10]
  - $\mathrm{cod}_M(0) =$ [00]
  - $\mathrm{cod}_M(1) =$ [01]

Beispielcodierung (4)

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 $\mathrm{cod}_{fgm}(z, x) = \texttt{[}\,\mathrm{cod}_Z(z)\,\mathrm{cod}_X(x)\,\texttt{[]}\,\texttt{[]}\,\texttt{[]]}$.
- wenn für Argumentpaar (z, x) definiert, Codierung $\mathrm{cod}_{fgm}(z, x) =$ $\texttt{[}\,\mathrm{cod}_Z(z)\,\mathrm{cod}_X(x)\,\mathrm{cod}_Z(f(z, x))\,\mathrm{cod}_X(g(z, x))\,\mathrm{cod}_M(m(z, x))\texttt{]}$.

- Beispiel (siehe BB3):
  - wenn $(f, g, m)(A, \square) = (B, 1, R)$ dann $\mathrm{cod}_{fgm}(A, \square) = \texttt{[[00] [0] [01] [1] [01]]}$
  - wenn $(f, g, m)(C, 1)$ undefiniert dann $\mathrm{cod}_{fgm}(C, 1) = \texttt{[[10] [1] [] [] []]}$

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

- Codierung der gesamten Funktionen:
  Konkatenation aller $\mathrm{cod}_{fgm}(z, x)$ 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.

Eigenschaften dieser und ähnlicher Codierungen

- 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 $\texttt{[}w_1\texttt{]}\texttt{[}w_2\texttt{]}$,
  - 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.

Überblick

Eine technische Vorbemerkung

Turingmaschinen
- Berechnungen
- Eingaben für Turingmaschinen
- Ergebnisse von Turingmaschinen

Berechnungskomplexität
- Komplexitätsmaße
- Komplexitätsklassen

Unentscheidbare Probleme
- Codierungen von Turingmaschinen
- Das Halteproblem
- Die Busy-Beaver-Funktion

Das Halteproblem ist unentscheidbar

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!

Es gibt Probleme, die

NICHT

algorithmisch

gelöst werden können!

Diagonalisierung

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

Diagonalisierung: Kernidee (2)

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

Diagonalisierung: Kernidee (2)

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

Diagonalisierung: Kernidee (2)

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

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

Diagonalisierung: Kernidee (2)

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

$\overline{d}$ unterscheidet sich von jeder Zeile f_i der Tabelle.

Das Halteproblem

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" $\overline{d}$:
  - Einerseits unterscheidet sich $\overline{d}$ von jeder Zeile der Tabelle, also von jeder von einer Turingmaschine berechneten Funktion.
  - Andererseits kann auch $\overline{d}$ von einer Turingmaschine berechnet werden.

Beweis des Halteproblems (1)

- 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" $\overline{d}$:
  - Einerseits unterscheidet sich $\overline{d}$ von jeder Zeile der Tabelle, also von jeder von einer Turingmaschine berechneten Funktion.
  - Andererseits kann auch $\overline{d}$ von einer Turingmaschine berechnet werden.

Beweis des Halteproblems (1)

- 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“ $\overline{d}$:
  - Einerseits unterscheidet sich $\overline{d}$ von jeder Zeile der Tabelle, also von jeder von einer Turingmaschine berechneten Funktion.
  - Andererseits kann auch $\overline{d}$ von einer Turingmaschine berechnet werden.

Beweis des Halteproblems (2)

- Wenn die Turingmaschine T_h existieren würde,
- dann könnte man „die verdorbene Diagonale“ als Turingmaschine bauen, d. h.
- es gäbe eine Turingmaschine $T_{\overline{d}}$, deren Arbeitsweise sich von der jeder Turingmaschine T_{x_i} unterscheiden würde !?!?!?

genauer ...

Beweis des Halteproblems (3)

- Wenn es Turingmaschine T_h gäbe,
- dann auch folgende Turingmaschine $T_{\overline{d}}$:
  - Für Eingabe x_i berechnet $T_{\overline{d}}$ zunächst, welches Ergebnis T_h für diese Eingabe liefern würde.
  - Dann:
    - Wenn T_h mitteilt, dass $T_{x_i}(x_i)$ hält, dann geht $T_{\overline{d}}$ in eine Endlosschleife.
    - Wenn T_h mitteilt, dass $T_{x_i}(x_i)$ nicht hält, dann hält $T_{\overline{d}}$ (und liefert irgendein Ergebnis, etwa 0).
  - in beiden Fällen verhält sich $T_{\overline{d}}$ für Eingabe x_i anders als T_{x_i}
- Wenn TM T_h existiert, dann auch TM $T_{\overline{d}}$,
- aber jede TM T_{x_i} verhält sich für Eingabe x_i anders als $T_{\overline{d}}$.
- Widerspruch
- Also gibt es keine TM T_h, die H entscheidet.

Beweis des Halteproblems (3)

- Wenn es Turingmaschine T_h gäbe,
- dann auch folgende Turingmaschine $T_{\overline{d}}$:
  - Für Eingabe x_i berechnet $T_{\overline{d}}$ zunächst, welches Ergebnis T_h für diese Eingabe liefern würde.
  - Dann:
    - Wenn T_h mitteilt, dass $T_{x_i}(x_i)$ hält, dann geht $T_{\overline{d}}$ in eine Endlosschleife.
    - Wenn T_h mitteilt, dass $T_{x_i}(x_i)$ nicht hält, dann hält $T_{\overline{d}}$ (und liefert irgendein Ergebnis, etwa 0).
  - in beiden Fällen verhält sich $T_{\overline{d}}$ für Eingabe x_i anders als T_{x_i}
- Wenn TM T_h existiert, dann auch TM $T_{\overline{d}}$,
- aber jede TM T_{x_i} verhält sich für Eingabe x_i anders als $T_{\overline{d}}$.
- Widerspruch
- Also gibt es keine TM T_h, die H entscheidet.

Beweis des Halteproblems (3)

- Wenn es Turingmaschine T_h gäbe,
- dann auch folgende Turingmaschine $T_{\overline{d}}$:
  - Für Eingabe x_i berechnet $T_{\overline{d}}$ zunächst, welches Ergebnis T_h für diese Eingabe liefern würde.
  - Dann:
    - Wenn T_h mitteilt, dass $T_{x_i}(x_i)$ hält, dann geht $T_{\overline{d}}$ in eine Endlosschleife.
    - Wenn T_h mitteilt, dass $T_{x_i}(x_i)$ nicht hält, dann hält $T_{\overline{d}}$ (und liefert irgendein Ergebnis, etwa 0).
  - in beiden Fällen verhält sich $T_{\overline{d}}$ für Eingabe x_i anders als T_{x_i}
- Wenn TM T_h existiert, dann auch TM $T_{\overline{d}}$,
- aber jede TM T_{x_i} verhält sich für Eingabe x_i anders als $T_{\overline{d}}$.
- Widerspruch
- Also gibt es keine TM T_h, die H entscheidet.

Weitere unentscheidbare Probleme

- Varianten des Halteproblems
  - Beispiel:
    Hält gegebene TM, wenn das Band zu Beginn völlig leer ist?
- Äquivalenzproblem:
  Liefern zwei TM für jede Eingabe die gleiche Ausgabe?
  - automatischer Vergleich mit „Musterlösungen" unmöglich
- Wird ein bestimmter Zustand einer Turingmaschine jemals gebraucht?
  - Erreichbarkeit von Codestücken unentscheidbar
- vieles vieles vieles vieles vieles vieles vieles mehr

- Beachte: statt Turingmaschine kann man immer Java-Programm einsetzen

Überblick

Eine technische Vorbemerkung

Turingmaschinen
- Berechnungen
- Eingaben für Turingmaschinen
- Ergebnisse von Turingmaschinen

Berechnungskomplexität
- Komplexitätsmaße
- Komplexitätsklassen

Unentscheidbare Probleme
- Codierungen von Turingmaschinen
- Das Halteproblem
- Die Busy-Beaver-Funktion

Erinnerung: BB3

- Bandalphabet ist $X = \{\square, \mathtt{1}\}$.
- 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 $X = \{\square, \mathtt{1}\}$.
- 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 zu Beginn immer vollständig leeres Band

Fleißige Biber und die Busy-Beaver-Funktion

- 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)

$$\begin{aligned} \mathrm{bb} &: \mathbb{N}_+ \to \mathbb{N}_+ \\ \mathrm{bb}(n) &= \text{die Anzahl von Einsen, die eine fleißige} \\ &\qquad n\text{-Bibermaschine am Ende auf dem Band hinterlässt} \end{aligned}$$

Die Busy-Beaver-Funktion: Schranken für einige Funktionswerte

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

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

Die Busy-Beaver-Funktion: Schranken für einige Funktionswerte

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

Die Busy-Beaver-Funktion: Schranken für einige Funktionswerte

n	$\mathrm{bb}(n)$	
1	1	
2	4	Radó (1963)
3	6	Radó (1963)
4	13	Brady (1974(?))
5		
6		
$\vdots$		

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

Die Busy-Beaver-Funktion: Schranken für einige Funktionswerte

n	$\mathrm{bb}(n)$	
1	1	
2	4	Radó (1963)
3	6	Radó (1963)
4	13	Brady (1974(?))
5	≥ 4098	Marxen/Buntrock (1990)
6	$> 3.514 \cdot 10^{18276}$	Kropitz (2010)
$\vdots$	$\vdots$	

Satz

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

Korollar

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

Was ist wichtig

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**
  - im 3. Semester: $\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE}$
- 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.