Grundbegriffe der Informatik

Kapitel 16: Turingmaschinen

Mattias Ulbrich
(basierend auf Folien von Thomas Worsch)

KIT · Institut für Theoretische Informatik

Wintersemester 2023/2024

Überblick

Karlsruher Institut für Technologie

Eine technische Vorbemerkung

Turingmaschinen

Berechnungskomplexität

Unentscheidbare Probleme

Wo sind wir?

Karlsruher Institut für Technologie

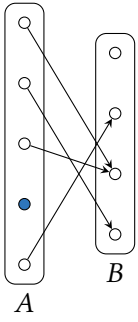
Eine technische Vorbemerkung

Turingmaschinen

Berechnungskomplexität

Unentscheidbare Probleme

Erinnerung: partielle Funktionen von A nach B



- rechtseindeutige Relation $f \subseteq A \times B$
- für jedes a gibt es höchstens ein $b \in B$ mit $(a, b) \in f$
  - totale Funktionen sind Spezialfall
- ggf. wieder $b = f(a)$ geschrieben
  - andernfalls ist „$f(a)$ undefiniert“
- Notation $f : A \dashrightarrow B$

Wo sind wir?

Eine technische Vorbemerkung

Turingmaschinen

Berechnungskomplexität

Unentscheidbare Probleme

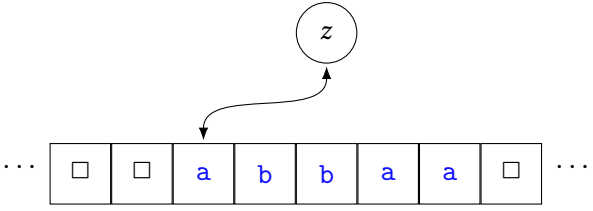
Vom endlichen Automaten augehend ...

- **Ziel:** Beschreibe formal: «eine Maschine führt einen Algorithmus aus.»
- **Idee:** endlichen Moore-Automaten, ...
  - ...für Ausgabefunktion: $o : X^* \to Y^*$ berechnen.
  - ...um reguläre Sprache $L \subseteq X^*$ zu akzeptieren.
- **Aber:** Mächtigkeit beschränkt, weil «Speicher» auf Zustände beschränkt.
- **Also:** Ergänze endliche Automaten um ein Konzept von Speicher
- Turingmaschinen sind im wesentlichen endliche Automaten, deren Ein-/und Ausgabe auf einem Speicher operieren.

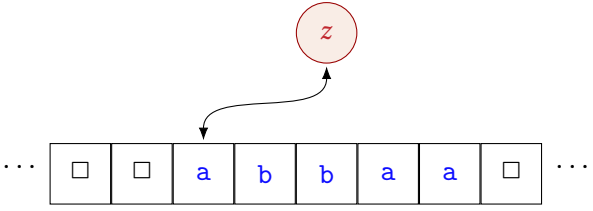
Turingmaschinen: Ursprung

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

Eine Turingmaschine im Bild

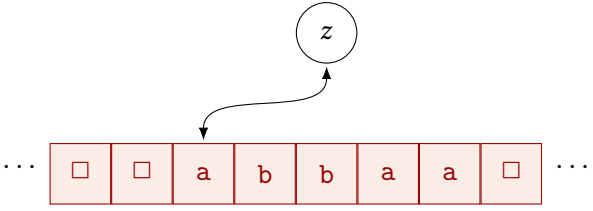


Eine Turingmaschine im Bild



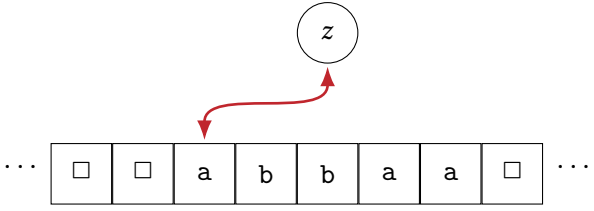
- Steuereinheit: endliche Zustandsmenge Z

Eine Turingmaschine im Bild



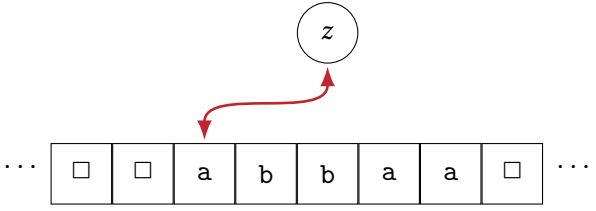
- endliche Zustandsmenge Z
- Band, in Felder unterteilt:
  beschriftet mit Symbolen aus Bandalphabet X

Eine Turingmaschine im Bild



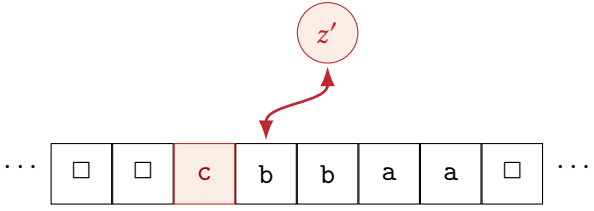
- endliche Zustandsmenge Z
- Bandalphabet X
- nächste Aktion festgelegt durch
  - aktuellen Zustand z und
  - aktuell gelesenes Symbol x

Eine Turingmaschine im Bild



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

Eine Turingmaschine im Bild

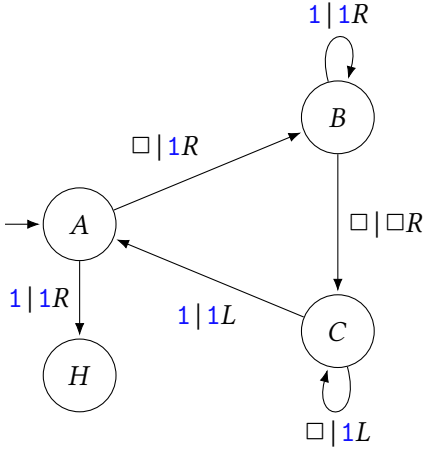


- endliche Zustandsmenge Z
- Bandalphabet X
- Schritt
  - neue Feldbeschriftung $g(z, \mathtt{a}) = \mathtt{c}$
  - neuer Zustand $f(z, \mathtt{a}) = z'$
  - Kopfbewegung $m(z, \mathtt{a}) = +1$

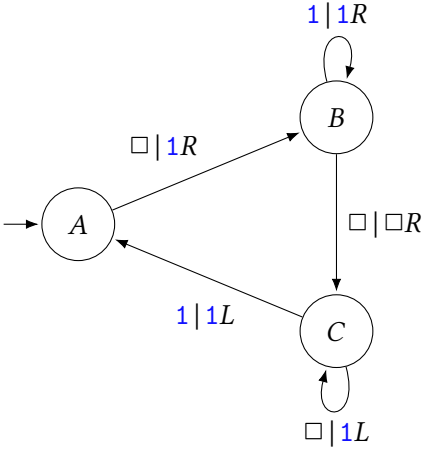
Turingmaschine formalisiert als $T = (Z, z_0, X, f, g, m)$

- Zustandsmenge Z (endlich!)
  - Anfangszustand $z_0 \in Z$
- Bandalphabet X (endlich!)
  - meist mit Blanksymbol $\square$
- partielle Zustandsüberführungsfunktion
  $f : Z \times X \dashrightarrow Z$
- partielle Ausgabefunktion
  $g : Z \times X \dashrightarrow X$ und
- partielle Bewegungsfunktion
  $m : Z \times X \dashrightarrow \{-1, 0, 1\}$ oder $\{L, 0, R\}$
- f, g, m mit gleichem Definitionsbereich

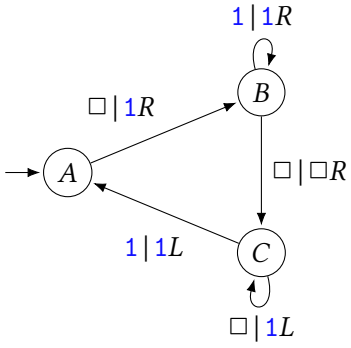
Turingmaschinen: graphische Darstellung (TM BB3)



Turingmaschinen: graphische Darstellung



Turingmaschinen: tabellarische Darstellung



- so:

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

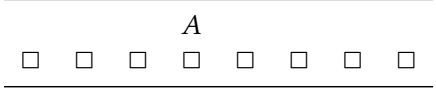
- oder so:

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

- oder …

Beispielberechnung (1)

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



Beispielberechnung (1)

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

Beispielberechnung (1)

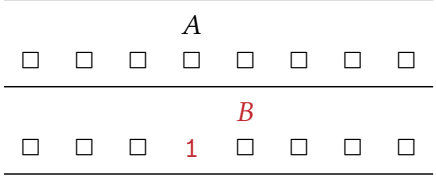
	A	B	C	H
□	$1, R, B$	$□, R, C$	$1, L, C$	
1	$1, R, H$	$1, R, B$	$1, L, A$	

Beispielberechnung (1)

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

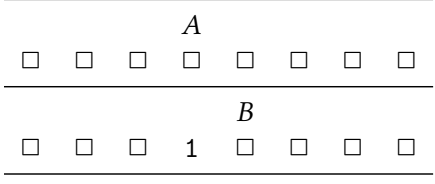
Beispielberechnung (2)

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



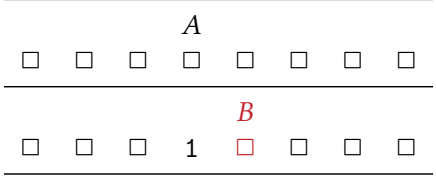
Beispielberechnung (3)

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



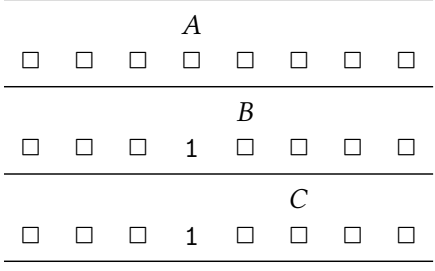
Beispielberechnung (4)

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



Beispielberechnung (5)

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



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
- C_T: Menge aller Konfigurationen von T, also $C_T = Z \times X^{\mathbb{Z}} \times \mathbb{Z}$

Turingmaschinen: «überschaubare» Bandbeschriftungen

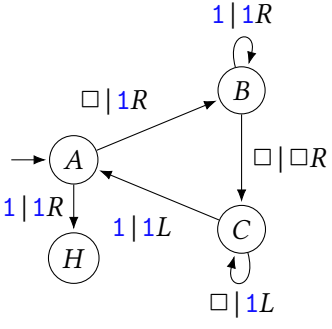
- Bandbeschriftung: ein «potenziell unendliches Gebilde»
- In weiten Teilen der Informatik interessieren
  - endliche Berechnungen, die
  - aus endlichen Eingaben
  - endliche Ausgaben

  berechnen.
- «Fast» das ganze Band ist immer «leer»:
  - Bandalphabet enthält Blanksymbol $\square \in X$
  - Bandbeschriftungen: immer nur endlich viele Felder $\neq \square$

Ein Schritt einer Turingmaschine

- $c = (z, b, p)$ aktuelle Konfiguration einer TM T
- aktuell gelesenes Symbol: $b(p)$
- wenn für $(z, b(p))$ die Funktionen f, g und m definiert, dann kann die TM *einen Schritt machen*
- Nachfolgekonfiguration $c' = (z', b', p')$:
  - $z' = f(z, b(p))$
  - $\forall i \in \mathbb{Z} : b'(i) = \begin{cases} b(i) & \text{falls } i \neq p \\ g(z, b(p)) & \text{falls } i = p \end{cases}$
  - $p' = p + m(z, b(p))$
- schreiben $c' = \Delta_1(c)$, also
  - $\Delta_1 : C_T \dashrightarrow C_T$

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	□	□

						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: $(c_0, c_1, c_2, \ldots, c_t)$
  - 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 (nicht haltend):
  - unendliche Folge $(c_0, c_1, c_2, \ldots)$ mit $\forall i > 0 \colon c_i = \Delta_1(c_{i-1})$
  - simple Beispiel-TM
    - $f(z, x) = z$,
    - $g(z, x) = x$ und
    - $m(z, x) = 1$
  - Kann man so etwas «wegkonstruieren» ?

Rechnen bis zur Endkonfiguration

- für $t \in \mathbb{N}_0$ Abbildung $\Delta_t : C_T \dashrightarrow C_T$

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

- von jedem c gibt es genau eine maximal lange Berechnung
  - wenn Berechnung endlich: Haltezeitpunkt eindeutig
- $\Delta_* : C_T \dashrightarrow C_T$ mit

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

zwei Arten von Turingmaschinen

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

Eingaben und Anfangskonfigurationen

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

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

  - $p = 0$
- Anfangskonfiguration bei Eingabe mehrerer Argumente
  - geeignet harmlos
  - z. B. $\square\square\square$ `[1011]` `[101]` $\square\square$ oder …

Ergebnisse von Turingmaschinenberechnungen

- Berechung von Funktionen
  - ein Ausgabewort
  - auf dem ansonsten leeren Band
- Erkennung formaler Sprachen: ein Bit akzeptiert/abgelehnt
- Turingmaschinenakzeptor T:
  - Teilmenge $F \subseteq Z$ akzeptierender Zustände
  - *T akzeptiert w*, wenn
    - T für Eingabe w hält und
    - Zustand der Endkonfiguration $\Delta_*(c_0(w))$ akzeptierend
  - $L(T)$: Menge der akzeptierten Wörter
    die von der Turingmaschine akzeptierte Sprache

Beispiel: Palindromerkennung

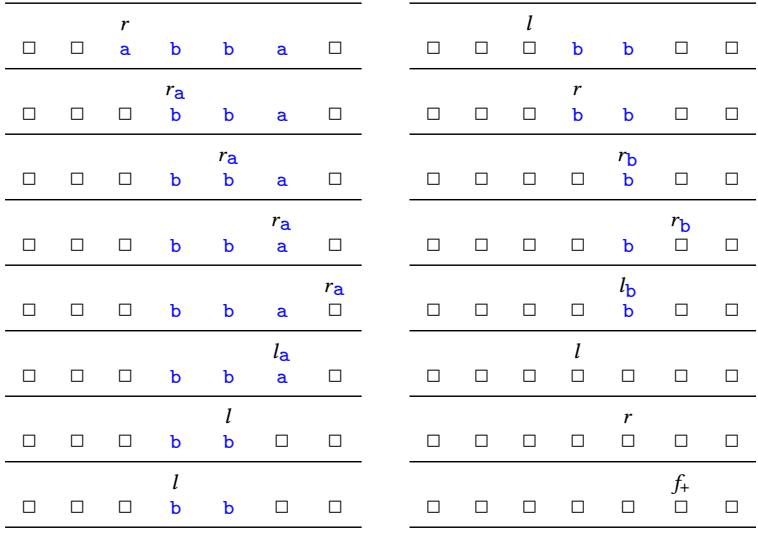
- ein Palindrom ist ein Wort, das seinem Spiegelbild gleicht
- also von hinten und von vorne gelesen «gleich»
  - z. B. `NEBEN`, `OTTO`, `RENTNER`, …
- bei dem also
  - erstes und letztes Symbol übereinstimmen
  - zweites und vorletztes Symbol übereinstimmen
  - usw.

Beispiel: Palindromerkennung

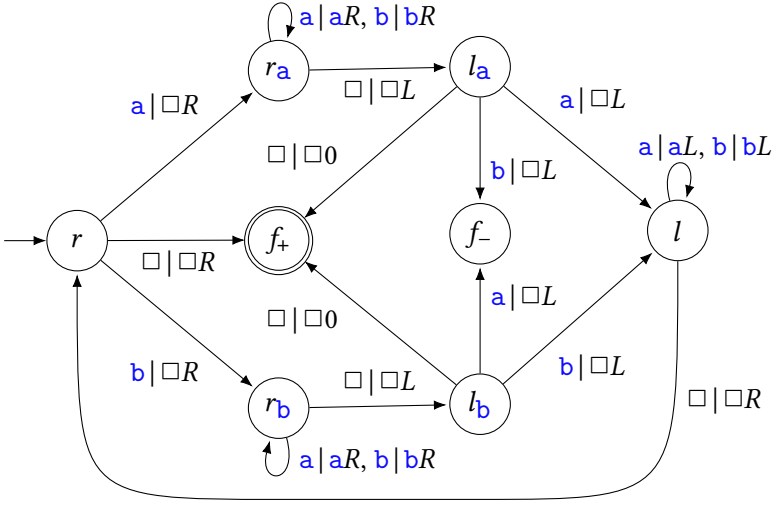
- ein Palindrom ist ein Wort, das seinem Spiegelbild gleicht
- also von hinten und von vorne gelesen «gleich»
  - z. B. `NEBEN`, `OTTO`, `RENTNER`, …
- bei dem also
  - erstes und letztes Symbol übereinstimmen
  - zweites und vorletztes Symbol übereinstimmen
  - usw.
- bei dem also
  - erstes und letztes Symbol übereinstimmen
  - und dazwischen ein Palindrom steht
- formale Sprache der Palindrome nicht regulär wenn $|X| \geq 2$

Palindromerkennung: Berechnung einer Beispiel-TM

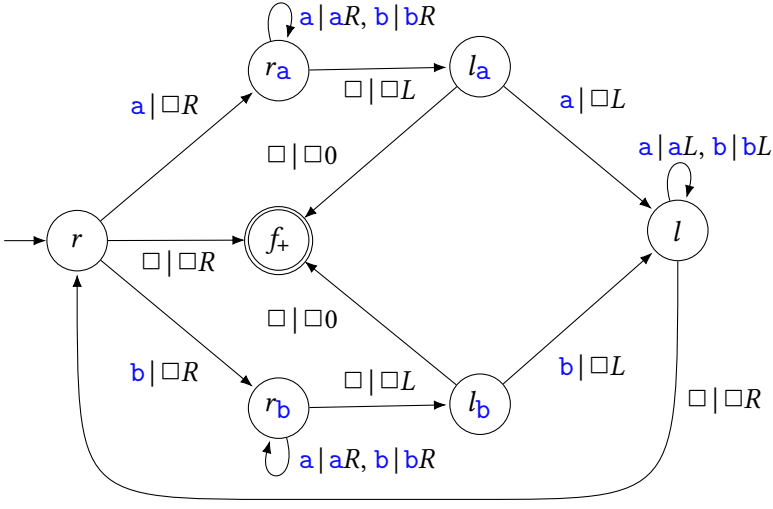
Palindromerkennung: Berechnung einer Beispiel-TM



Palindromerkennung: Beispiel-TM



Palindromerkennung: Beispiel-TM



Entscheidbare und aufzählbare Sprachen

- zwei Möglichkeiten, wenn w von T *nicht* akzeptiert wird:
  1. T hält für Eingabe w, aber letzter Zustand nicht akzeptierend
  2. T hält für Eingabe w nicht
- Was weiß man?
  1. T ist fertig, Eingabe abgelehnt
  2. T ist noch nicht fertig.
     Ob T irgendwann w noch akzeptiert oder ablehnt, ist unklar.
- man unterscheide
  1. L heißt entscheidbare Sprache, wenn es eine TM gibt, die L akzeptiert und *immer hält!*
  2. L heißt aufzählbare Sprache, wenn es eine TM gibt, die L akzeptiert.

Entscheidbare und aufzählbare Sprachen

- zwei Möglichkeiten, wenn w von T *nicht* akzeptiert wird:
  1. T hält für Eingabe w, aber letzter Zustand nicht akzeptierend
  2. T hält für Eingabe w nicht
- Was weiß man?
  1. T ist fertig, Eingabe abgelehnt
  2. T ist noch nicht fertig.
     Ob T irgendwann w noch akzeptiert oder ablehnt, ist unklar.
- man unterscheide
  1. L heißt entscheidbare Sprache, wenn es eine TM gibt, die L akzeptiert und *immer hält!*
  2. L heißt aufzählbare Sprache, wenn es eine TM gibt, die L akzeptiert.
- Entscheidbarkeit ist eine echt stärkere Forderung

Was ist wichtig

- Das sollten Sie mitnehmen:
  - Turingmaschinen
    - Steuereinheit endlich
    - Band unendlich, aber nur endlich viel «nicht leer»
  - alles endlich beschreibbar
  - klassische Formalisierung von «Algorithmus»
  - Berechnungen
    - haltende
    - nicht haltende
- Das sollten Sie üben:
  - Beispielturingmaschinen konstruieren
  - Beispielturingmaschinen verstehen

Wo sind wir?

Karlsruher Institut für Technologie

Eine technische Vorbemerkung

Turingmaschinen

Berechnungskomplexität

Unentscheidbare Probleme

Achtung Achtung Achtung Achtung Achtung Achtung

Achtung Achtung Achtung Achtung Achtung Achtung

- Annahme in diesem Abschnitt:
  Turingmaschinen halten für jede Eingabe.
  Aber *nur* in diesem Abschnitt!
- 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.

Zeitkomplexität — der Rechenzeitbedarf einer TM

- definiere $\quad \text{time}_T : A^+ \to \mathbb{N}_0$

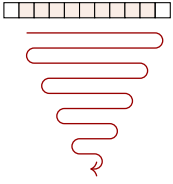
  $w \mapsto$ das t, für das $\Delta_t(c_0(w))$ Endkonfiguration

  $\text{Time}_T : \mathbb{N}_+ \to \mathbb{N}_0$

  $n \mapsto \max\{\text{time}_T(w) \mid w \in A^n\}$

- Time_T heißt Zeitkomplexität der Turingmaschine
  - «max» $\hat{=}$ «worst case»
- Zeitkomplexität von T polynomiell,
  wenn $\text{Time}_T(n) \in O(p(n))$ für ein Polynom $p(n)$

Zeitkomplexität der TM für Palindromerkennung



- Für Eingabe der Länge $n \geq 2$ schlimmstenfalls
  1. erstes und letztes Symbol miteinander vergleichen, und weil übereinstimmend, zurücklaufen und dann
  2. für Teilwort der Länge $n-2$ ohne Randsymbole wieder ein Palindromtest
- für $n \geq 2$: $\text{Time}(n) \leq 2n + 1 + \text{Time}(n-2)$
- für Wörter der Längen 1 und 2 Zeitaufwand konstant
- $\text{Time}(n) \in O\left(n^2\right)$, d. h. polynomielle Zeitkomplexität

Platzkomplexität oder Raumkomplexität einer TM

- definiere $\text{space}_T(w) : A^+ \to \mathbb{N}_+$

  $w \mapsto$ Anzahl Felder, die während der Berechnung für Eingabe w «benötigt»

  $\text{Space}_T(n) : \mathbb{N}_+ \to \mathbb{N}_+$

  $n \mapsto \max\{\text{space}_T(w) \mid w \in A^n\}$

- Feld «benötigt», wenn
  - anfangs dort Eingabesymbol oder
  - Feld mindestens einmal vom TM-Kopf besucht

Raumkomplexität der TM für Palindromerkennung

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

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

Zeitkomplexität versus Raumkomplexität (1)

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

$$\text{space}(w) \leq |w| + \text{time}(w)$$

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

Zeitkomplexität versus Raumkomplexität (2)

- nun 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.

Eine Komplexitätsklasse ist eine Menge von *Problemen*

- hier nur formale Sprachen (Entscheidungsprobleme)
- Festlegung oft durch Beschränkung verfügbarer Ressourcen, z. B. *Schranke für Zeitkomplexität* oder *Raumkomplexität* (oder beides)
- Beispiel: alle formalen Sprachen, die von Turingmaschinen entschieden werden können, bei denen gleichzeitig
  - Zeitkomplexität in $O\left(n^3\right)$ und
  - Raumkomplexität in $O\left(n^{3/2}\log n\right)$ ist

  wobei n die Länge des Eingabewortes ist.

P und **PSPACE** — zwei wichtige Komplexitätsklassen

- **P**
  Menge aller formaler Sprachen, die von TM entschieden werden können, deren Zeitkomplexität polynomiell ist.
- **PSPACE**
  Menge aller formaler Sprachen, die von TM entschieden werden können, deren Raumkomplexität polynomiell ist.
- Beispiele
  - «Palindrome» ist in **P**
  - «Äquivalenz regulärer Ausdrücke» ist in **PSPACE**

Beziehung zwischen **P** und **PSPACE** — unklar

- polynomielle Laufzeit $\Longrightarrow$ polynomieller Platzbedarf:
  $\mathbf{P} \subseteq \mathbf{PSPACE}$

Beziehung zwischen **P** und **PSPACE** — unklar

- polynomielle Laufzeit $\Longrightarrow$ polynomieller Platzbedarf:
  $\mathbf{P} \subseteq \mathbf{PSPACE}$
- Und umgekehrt? Vorsicht!

Beziehung zwischen **P** und **PSPACE** — unklar

- polynomielle Laufzeit $\Longrightarrow$ polynomieller Platzbedarf:
  $\mathbf{P} \subseteq \mathbf{PSPACE}$
- Und umgekehrt? Vorsicht!
  - TM mit polynomiellem Platzbedarf kann exponentiell viele Schritte machen kann.
  - Solche Turingmaschinen gibt es.

Beziehung zwischen **P** und **PSPACE** — unklar

- polynomielle Laufzeit $\Longrightarrow$ polynomieller Platzbedarf:
  $\mathbf{P} \subseteq \mathbf{PSPACE}$
- Und umgekehrt? Vorsicht!
  - TM mit polynomiellem Platzbedarf kann exponentiell viele Schritte machen kann.
  - Solche Turingmaschinen gibt es.
- Aber!
  - trotzdem könnte $\mathbf{PSPACE} \subseteq \mathbf{P}$ sein, wenn immer eine viel schnellere äquivalente TM existiert

Beziehung zwischen **P** und **PSPACE** — unklar

- polynomielle Laufzeit $\Longrightarrow$ polynomieller Platzbedarf:
  $\mathbf{P} \subseteq \mathbf{PSPACE}$
- Und umgekehrt? Vorsicht!
  - TM mit polynomiellem Platzbedarf kann exponentiell viele Schritte machen kann.
  - Solche Turingmaschinen gibt es.
- Aber!
  - trotzdem könnte $\mathbf{PSPACE} \subseteq \mathbf{P}$ sein, wenn immer eine viel schnellere äquivalente TM existiert
- Bei **P** und **PSPACE** geht es um formale Sprachen, nicht um Turingmaschinen.

Beziehung zwischen **P** und **PSPACE** — unklar

- polynomielle Laufzeit $\Longrightarrow$ polynomieller Platzbedarf:
  $\mathbf{P} \subseteq \mathbf{PSPACE}$
- Und umgekehrt? Vorsicht!
  - TM mit polynomiellem Platzbedarf kann exponentiell viele Schritte machen kann.
  - Solche Turingmaschinen gibt es.
- Aber!
  - trotzdem könnte $\mathbf{PSPACE} \subseteq \mathbf{P}$ sein, wenn immer eine viel schnellere äquivalente TM existiert
- Bei **P** und **PSPACE** geht es um formale Sprachen, nicht um Turingmaschinen.
- unbekannt, ob $\mathbf{PSPACE} \subseteq \mathbf{P}$ oder $\mathbf{PSPACE} \not\subseteq \mathbf{P}$!
- großes offenes Problem:
  $\mathbf{P} = \mathbf{PSPACE}$ oder $\mathbf{P} \neq \mathbf{PSPACE}$?

Was ist wichtig

- Das sollten Sie mitnehmen:
  - Zeitkomplexität und Raumkomplexität
    - 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: **P** und **PSPACE**
- Das sollten Sie üben:
  - Abschätzung der Zeit- und Raumkomplexität von TM

Wo sind wir?

Karlsruher Institut für Technologie

Eine technische Vorbemerkung

Turingmaschinen

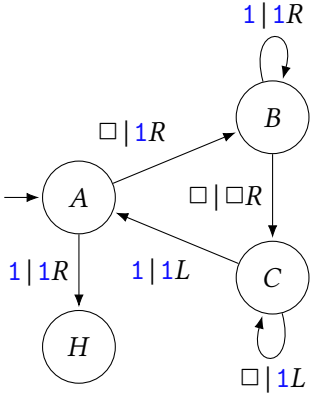
Berechnungskomplexität

Unentscheidbare Probleme

Ausgelagert ...

Die Inhalte dieses Abschnitts finden Sie in der Foliensammlung “16a”.

Erinnerung: BB3



- 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.

Bibermaschinen

- 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 TM 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

- Busy-Beaver-Funktion (oder Radó-Funktion)

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

- fleißiger Biber: n-Bibermaschine die $\mathrm{bb}(n)$ Einsen schafft

Busy-Beaver-Funktion — Schranken für einige Funktionswerte

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

Busy-Beaver-Funktion — Schranken für einige Funktionswerte

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

Busy-Beaver-Funktion — Schranken für einige Funktionswerte

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

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		
$\vdots$		

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		
$\vdots$		

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	$> 3.514 \cdot 10^{18267}$	Kropitz (2010)
$\vdots$	$\vdots$	

Die Busy-Beaver-Funktion ist nicht berechenbar

Satz

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

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
  - Busy-Beaver-Funktion wächst schneller als jede berechenbare Funktion.
- Das sollten Sie üben:
  - informelle algorithmische Beschreibungen in Turingmaschinen überführen

Steam-Powered Turing Machine ;-)

- http://www.cs.washington.edu/homes/ruzzo/
- *„[…] principal research project involves the construction and programming […] of 32 steam-powered Turing machines“*
- *„Graduate students have played an important role in the construction and operation of the engine […] advanced undergraduates are occasionally allowed to polish the brass gauges.“*
- *“Originally intended as a general computing engine, restrictions imposed by the Pollution Control and Noise Abatement Boards require that only algorithms running in polynomial time may be used.“*
- *„one of [the] students slipped on a mouldering stack of ungraded homework exercises and fell under the write head of one of the machines. Now permanently embossed with a series of 1's and 0's, the student is suing to have the machine dismantled.“*
- https://www.cs.washington.edu/art/SPTM

Zusammenfassung

- Turingmaschinen sind eine formale Präzisierung des Algorithmusbegriffs.
- Komplexitätsmaße und Komplexitätsklassen
  - insbesondere **P** und **PSPACE**
  - im 3. Semester: **P** $\subseteq$ **NP** $\subseteq$ **PSPACE**
- Es gibt Probleme, die *anscheinend* sehr großen algorithmischen Aufwand erfordern.
- Es gibt Probleme, die *beweisbar* sehr großen algorithmischen Aufwand erfordern.
- Es gibt Probleme, die algorithmisch *gar nicht* lösbar sind.