Grundbegriffe der Informatik

Einheit 16: Turingmaschinen

Thomas Worsch

Universität Karlsruhe, Fakultät für Informatik

Wintersemester 2008/2009

Überblick

Eine technische Vorbemerkung

Turingmaschinen
- Berechnungen
- Eingaben für Turingmaschinen
- Ergebnisse von Turingmaschinen

Berechnungskomplexität
- Komplexitätsmaße
- Komplexitätsklassen

Schwere Probleme

Unentscheidbare Probleme
- Universelle Turingmaschinen
- Das Halteproblem
- Die Busy-Beaver-Funktion

Eine technische Vorbemerkung

Turingmaschinen
- Berechnungen
- Eingaben für Turingmaschinen
- Ergebnisse von Turingmaschinen

Berechnungskomplexität
- Komplexitätsmaße
- Komplexitätsklassen

Schwere Probleme

Unentscheidbare Probleme
- Universelle 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

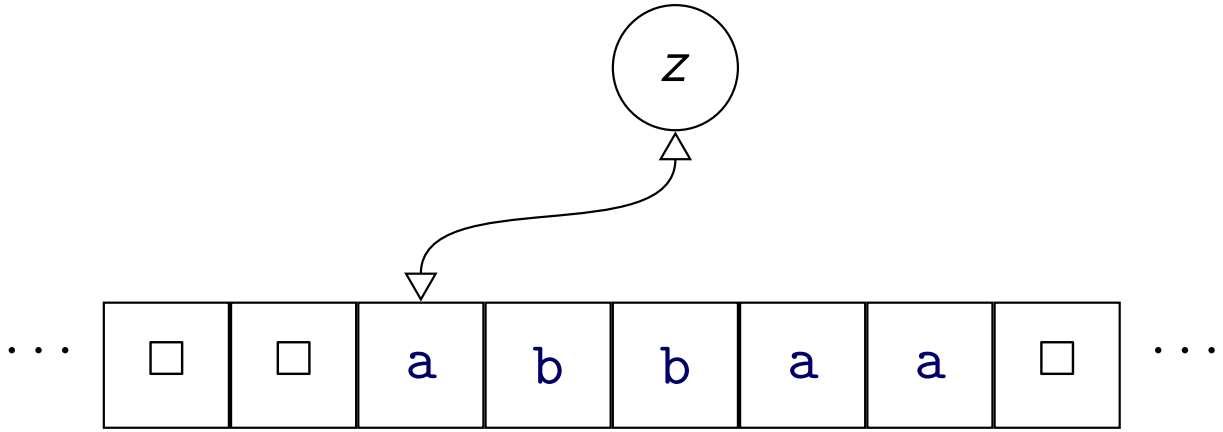
Schwere Probleme

Unentscheidbare Probleme
- Universelle Turingmaschinen
- Das Halteproblem
- Die Busy-Beaver-Funktion

Turingmaschinen: Ursprung

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

Turingmaschinen: das Bild

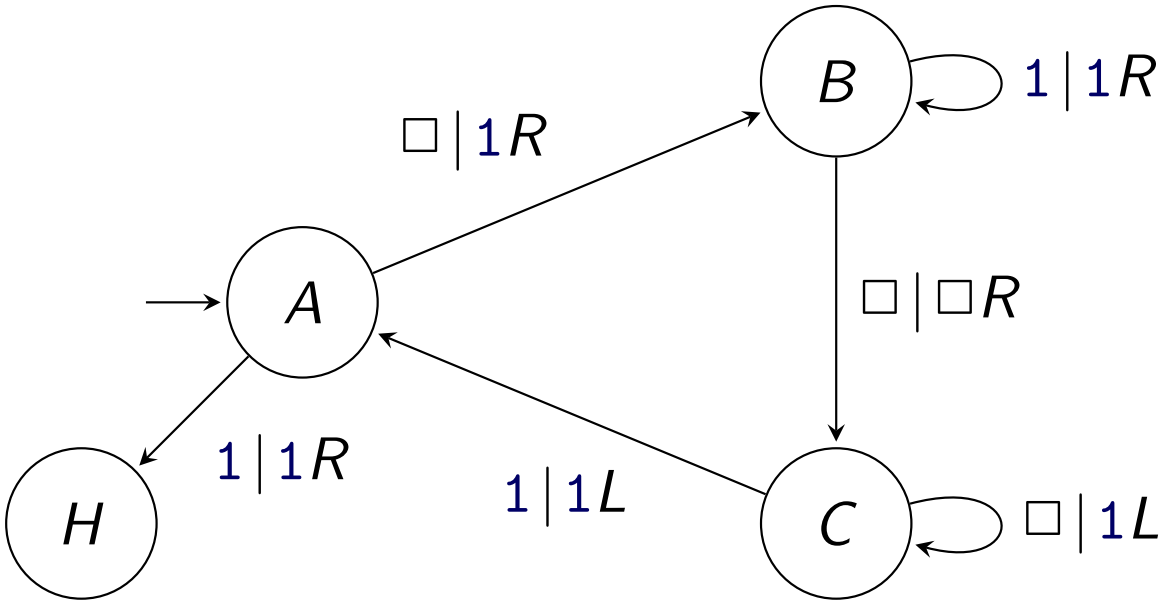


Turingmaschinen: Formalisierung

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

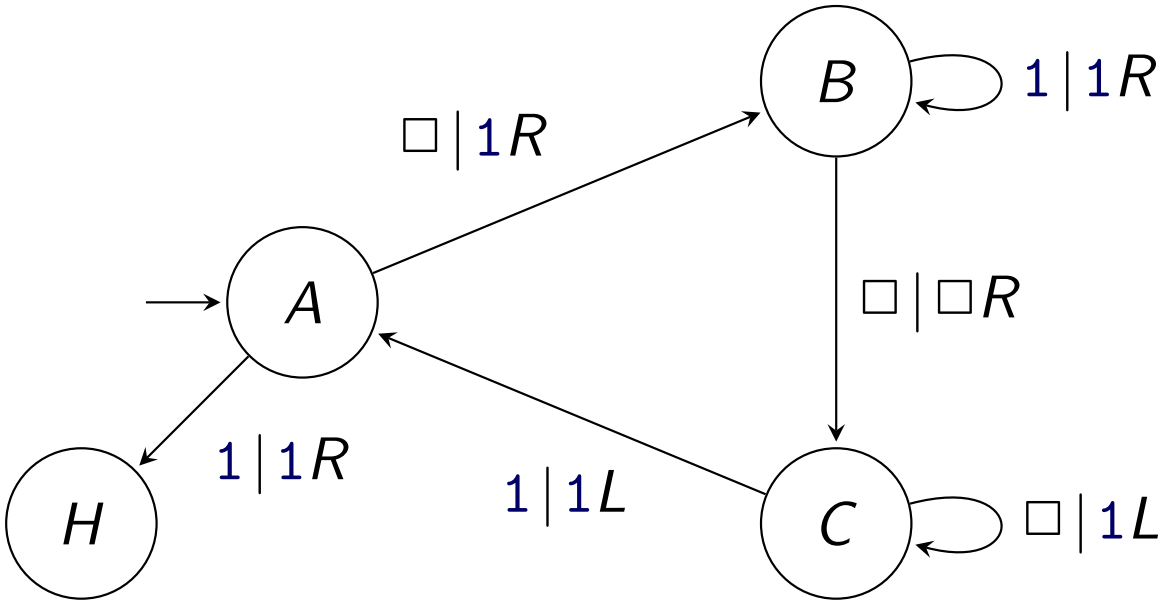
- eine Zustandsmenge Z
- einen Anfangszustand $z_0 \in Z$
- ein Bandalphabet X
- eine partielle Zustandsüberführungsfunktion $f : Z \times X \dashrightarrow Z$
- eine partielle Ausgabefunktion $g : Z \times X \dashrightarrow X$ und
- eine partielle Bewegungsfunktion $m : Z \times X \dashrightarrow \{-1,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

Turingmaschinen: Darstellung



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

Turingmaschinen: Darstellung



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

Turingmaschinen: Konfigurationen

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

Überblick

Eine technische Vorbemerkung

Turingmaschinen
- Berechnungen
- Eingaben für Turingmaschinen
- Ergebnisse von Turingmaschinen

Berechnungskomplexität
- Komplexitätsmaße
- Komplexitätsklassen

Schwere Probleme

Unentscheidbare Probleme
- Universelle 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
- $\Delta_1 : \mathcal{C}_T \dashrightarrow \mathcal{C}_T$

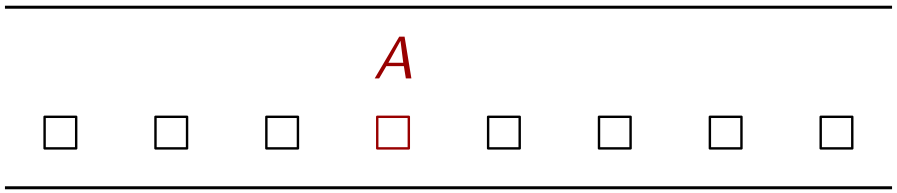
Beispiel

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

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

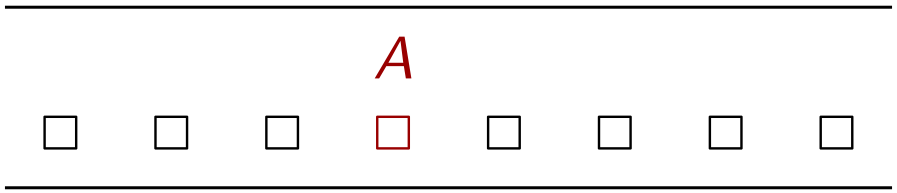
Beispiel

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



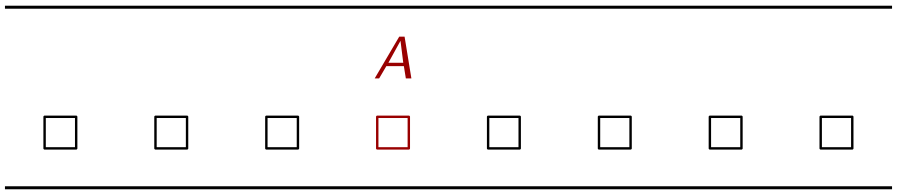
Beispiel

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



Beispiel

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

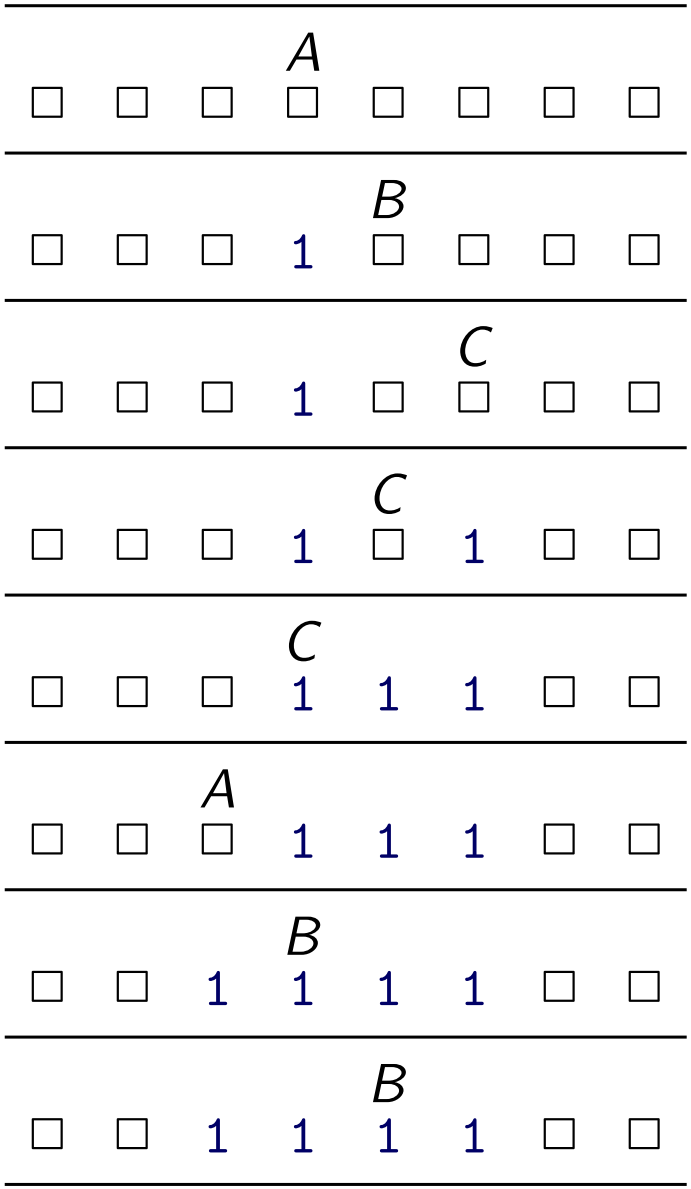


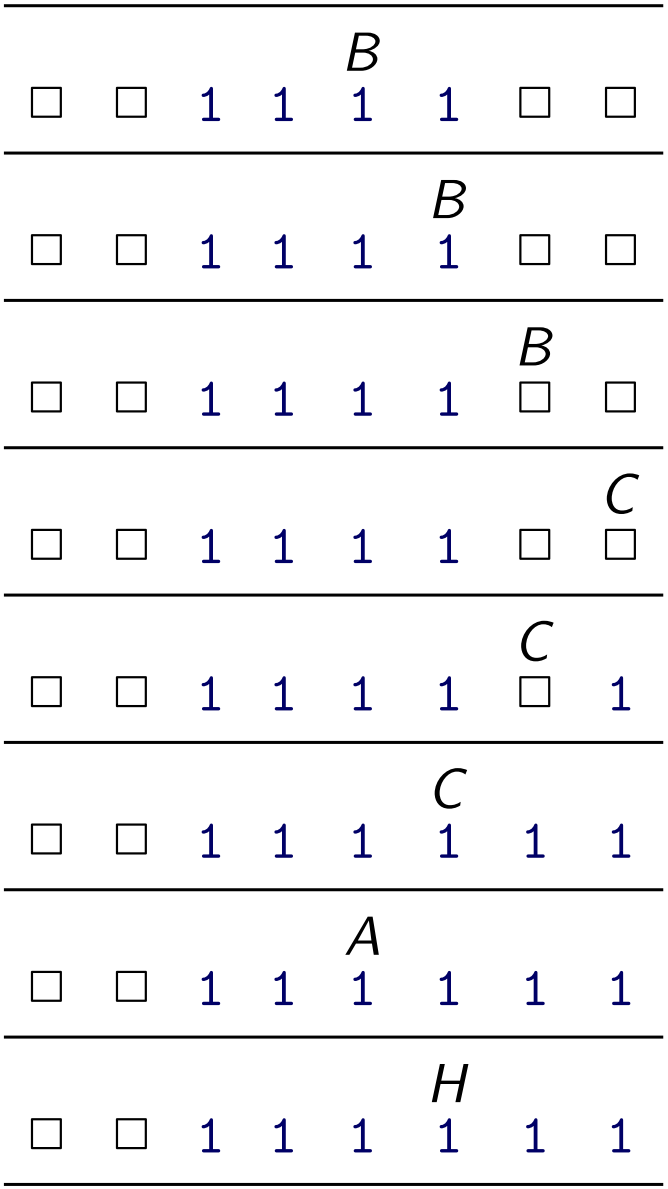
Beispiel

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

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

Längere Beispielberechnung





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 „wegmachen“? ...

Rechnen bis zur Endkonfiguration

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

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

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

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

Überblick

Eine technische Vorbemerkung

Turingmaschinen
- Berechnungen
- Eingaben für Turingmaschinen
- Ergebnisse von Turingmaschinen

Berechnungskomplexität
- Komplexitätsmaße
- Komplexitätsklassen

Schwere Probleme

Unentscheidbare Probleme
- Universelle Turingmaschinen
- Das Halteproblem
- Die Busy-Beaver-Funktion

zwei Arten von Turingmaschinen

analog zu endlichen Automaten

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

Eingaben und Anfangskonfigurationen

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

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

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

Überblick

Eine technische Vorbemerkung

Turingmaschinen
- Berechnungen
- Eingaben für Turingmaschinen
- Ergebnisse von Turingmaschinen

Berechnungskomplexität
- Komplexitätsmaße
- Komplexitätsklassen

Schwere Probleme

Unentscheidbare Probleme
- Universelle Turingmaschinen
- Das Halteproblem
- Die Busy-Beaver-Funktion

Ausgaben

analog zu endlichen Automaten

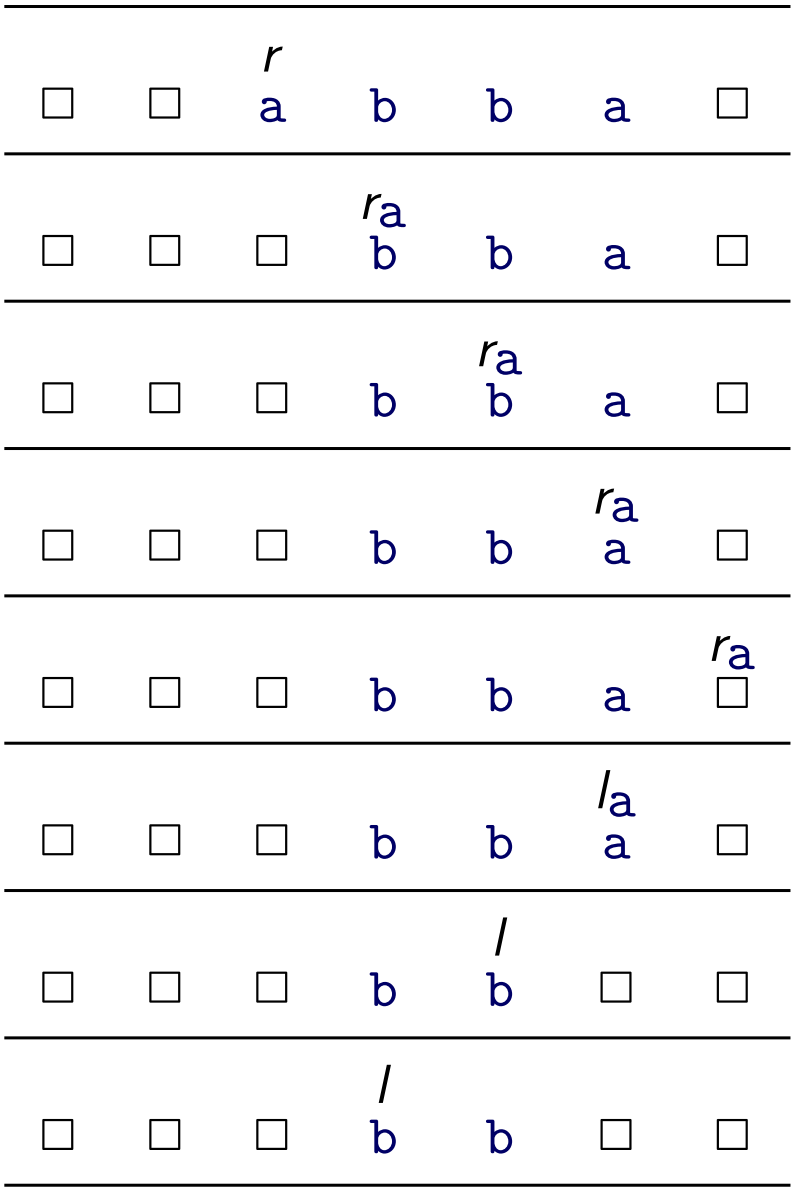
- Berechung von Funktionen: ein Ausgabewort
- Erkennung formaler Sprachen: ein Bit akzeptiert/abgelehnt

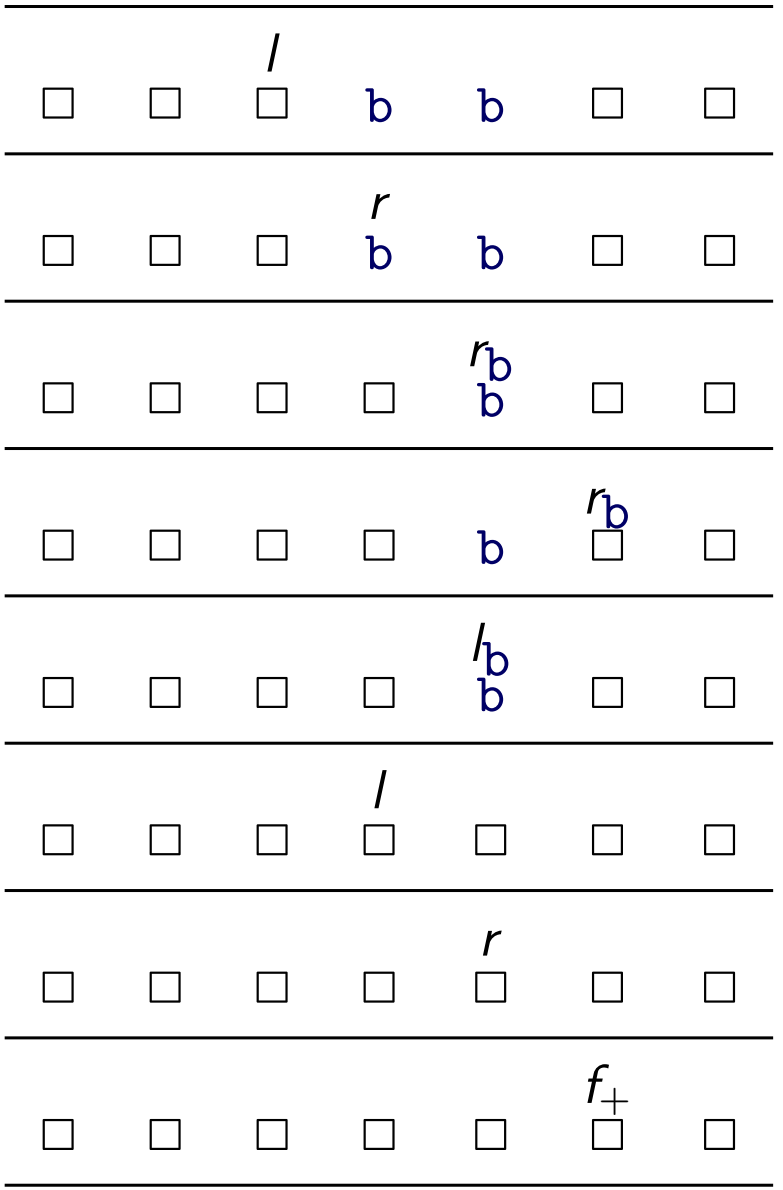
- Teilmenge $F \subset Z$ *akzeptierender Zustände*
- T akzeptiert w, wenn
  - T für Eingabe w hält und
  - der Zustand der Endkonfiguration $\Delta_*(c_0(w))$ akzepierend ist.
- $L(T)$: Menge der akzeptierten Wörter

Aufzählbare und entscheidbare Sprachen

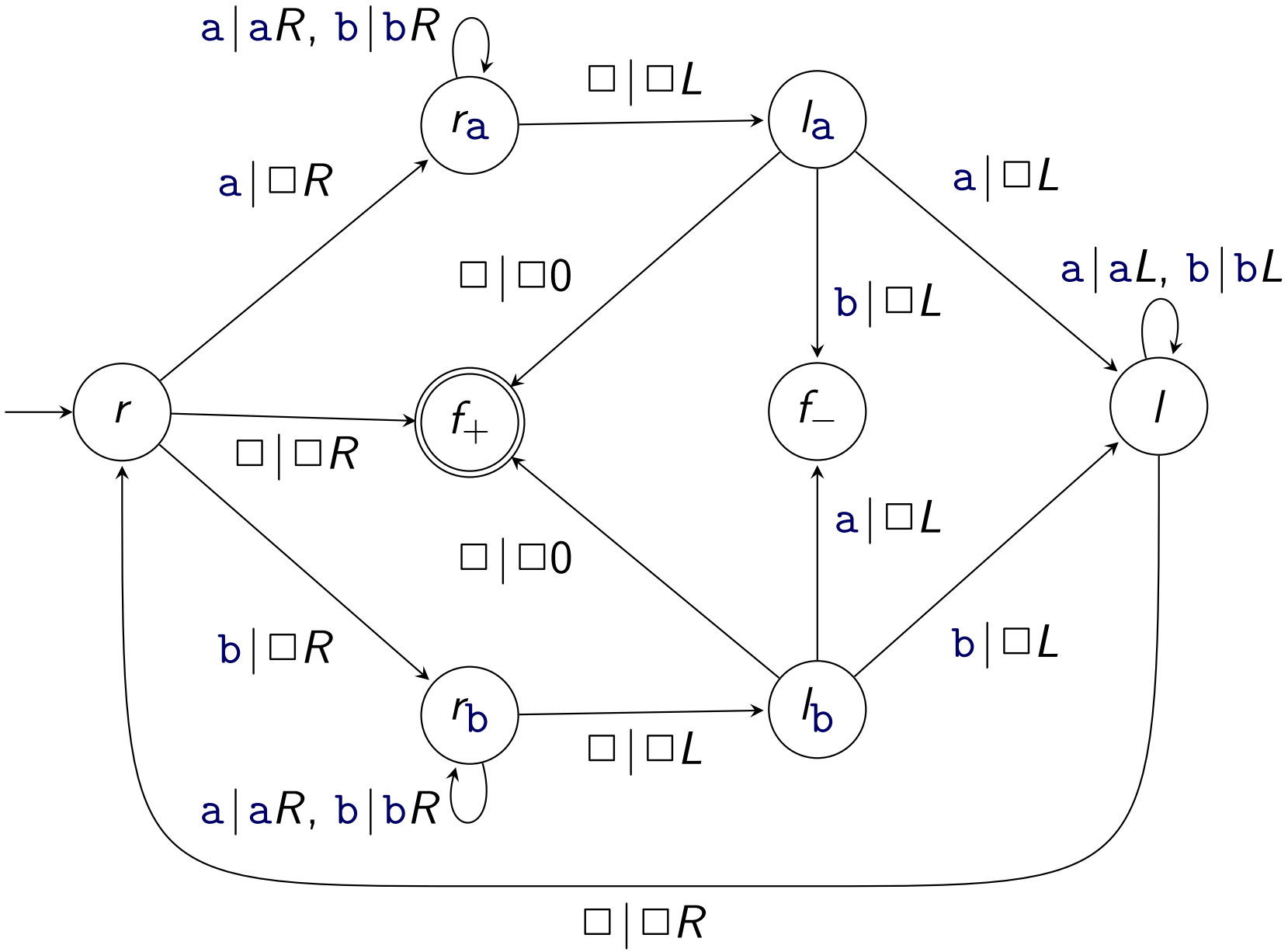
- zwei Möglichkeiten, wenn w von T *nicht* akzeptiert wird:
  1. T hält für Eingabe w, aber
     Endzustand ist nicht akzeptierend.
  2. T hält für Eingabe w nicht.
- Was weiß man?
  1. T ist fertig und lehnt die Eingabe ab.
  2. T ist noch nicht fertig.
     Ob T irgendwann w noch akzeptiert oder ablehnt, 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.
- werden sehen:
  Entscheidbarkeit ist eine stärkere Forderung

Palindromerkennung: Beispielberechnung





Palindromerkennung: Beispielturingmaschine



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

Schwere Probleme

Unentscheidbare Probleme
- Universelle Turingmaschinen
- Das Halteproblem
- Die Busy-Beaver-Funktion

Achtung

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

Überblick

Eine technische Vorbemerkung

Turingmaschinen
- Berechnungen
- Eingaben für Turingmaschinen
- Ergebnisse von Turingmaschinen

Berechnungskomplexität
- Komplexitätsmaße
- Komplexitätsklassen

Schwere Probleme

Unentscheidbare Probleme
- Universelle 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:

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

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

Zeitkomplexität der Beispielturingmaschine für Palindromerkennung

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

  insgesamt also

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

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

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

Raumkomplexität

- 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

$\mathrm{space}_T(w) =$ die Anzahl der Felder, die während der Berechnung für Eingabe w benötigt werden

$\mathrm{Space}_T(n) = \max\{\mathrm{space}(w) \mid w \in A^n\}$

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

Raumkomplexität der Beispielturingmaschine für Palindromerkennung

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

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

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)

nun umgekehrt von Raum- zu Zeitkomplexität:

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

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

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

Überblick

Eine technische Vorbemerkung

Turingmaschinen
- Berechnungen
- Eingaben für Turingmaschinen
- Ergebnisse von Turingmaschinen

Berechnungskomplexität
- Komplexitätsmaße
- Komplexitätsklassen

Schwere Probleme

Unentscheidbare Probleme
- Universelle Turingmaschinen
- Das Halteproblem
- Die Busy-Beaver-Funktion

Komplexitätsklassen

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

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

Zwei wichtige Komplexitätsklassen

- $\mathbf{P}$ ist die Menge aller formaler Sprachen, die von Turingmaschinen entschieden werden können, deren Zeitkomplexität polynomiell ist.
- $\mathbf{PSPACE}$ 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 $\mathbf{P}$
- „Äquivalenz regulärer Ausdrücke“ ist in $\mathbf{PSPACE}$

Zusammenhang zwischen $\mathbf{P}$ und $\mathbf{PSPACE}$

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

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

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

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

Zusammenhang zwischen $\mathbf{P}$ und $\mathbf{PSPACE}$

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

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

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

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

Zusammenhang zwischen $\mathbf{P}$ und $\mathbf{PSPACE}$

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

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

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

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

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
  - es gibt auch noch andere Komplexitätsmaße ...
- Komplexitätsklassen
  - durch Beschränkung der zur Verfügung stehen Ressourcen, also
  - z. B. Schranken für Zeitkomplexität oder/und Raumkomplexität
  - wichtig (neben anderen wie **NP** ...)
    - **P**
    - **PSPACE**

Das sollten Sie üben:

- Abschätzung der Zeit- und Raumkomplexität von TM

Überblick

Eine technische Vorbemerkung

Turingmaschinen
- Berechnungen
- Eingaben für Turingmaschinen
- Ergebnisse von Turingmaschinen

Berechnungskomplexität
- Komplexitätsmaße
- Komplexitätsklassen

Schwere Probleme

Unentscheidbare Probleme
- Universelle Turingmaschinen
- Das Halteproblem
- Die Busy-Beaver-Funktion

Reduzierbarkeit

- Es seien L_1 und L_2 formale Sprachen über Alphabet A.
- *L_1 ist auf L_2 (polynomialzeit-)reduzierbar*, wenn
  - eine in Polynomialzeit berechenbare Funktion $f : A^* \to A^*$ existiert
  - mit
$$w \in L_1 \iff f(w) \in L_2 \ .$$
  - Eine solche Funktion kann auch wieder nur polynomiell viel Platz benötigen.

Folgen von Reduzierbarkeit

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

Schwere und vollständige Probleme

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

Der Stand der Wissenschaft

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

- Man kennt für kein einziges $\mathbf{PSPACE}$-vollständiges Problem einen Polynomialzeit-Algorithmus,
- aber man kann auch nicht beweisen, dass keine existieren.

Damit kein falscher Eindruck entsteht

- Es gibt beweisbar gaaaaanz schwere Probleme,
- z. B. solche, für deren Lösung jede Turingmaschine Zeitbedarf

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

  hat.
- Solche Probleme
  - kann man prinzipiell lösen,
  - aber man „will“ nicht so lange warten

Was ist wichtig

Das sollten Sie mitnehmen:

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

Das sollten Sie üben:

- ausnahmsweise mal nichts . . .

Überblick

Eine technische Vorbemerkung

Turingmaschinen
- Berechnungen
- Eingaben für Turingmaschinen
- Ergebnisse von Turingmaschinen

Berechnungskomplexität
- Komplexitätsmaße
- Komplexitätsklassen

Schwere Probleme

Unentscheidbare Probleme
- Universelle Turingmaschinen
- Das Halteproblem
- Die Busy-Beaver-Funktion

Überblick

Eine technische Vorbemerkung

Turingmaschinen
- Berechnungen
- Eingaben für Turingmaschinen
- Ergebnisse von Turingmaschinen

Berechnungskomplexität
- Komplexitätsmaße
- Komplexitätsklassen

Schwere Probleme

Unentscheidbare Probleme
- Universelle Turingmaschinen
- Das Halteproblem
- Die Busy-Beaver-Funktion

Überblick

Eine technische Vorbemerkung

Turingmaschinen
- Berechnungen
- Eingaben für Turingmaschinen
- Ergebnisse von Turingmaschinen

Berechnungskomplexität
- Komplexitätsmaße
- Komplexitätsklassen

Schwere Probleme

Unentscheidbare Probleme
- Universelle Turingmaschinen
- Das Halteproblem
- Die Busy-Beaver-Funktion

Überblick

Eine technische Vorbemerkung

Turingmaschinen
- Berechnungen
- Eingaben für Turingmaschinen
- Ergebnisse von Turingmaschinen

Berechnungskomplexität
- Komplexitätsmaße
- Komplexitätsklassen

Schwere Probleme

Unentscheidbare Probleme
- Universelle Turingmaschinen
- Das Halteproblem
- Die Busy-Beaver-Funktion

Was ist wichtig

Das sollten Sie mitnehmen:

- alles endlich
- alles endlich beschreibbar
- im Beispiel Getränkeautomat:
  - aktueller Zustand und aktuelle Eingabe legen immer und eindeutig fest:
    - nächsten Zustand
    - Ausgabe

Das sollten Sie üben:

- Zustandsdiagramm lesen

Zusammenfassung

- Turingmaschinen