Grundbegriffe der Informatik

Kapitel 20: Turingmaschinen

Thomas Worsch

KIT, Institut für Theoretische Informatik

Wintersemester 2015/2016

Überblick

Wo sind wir?

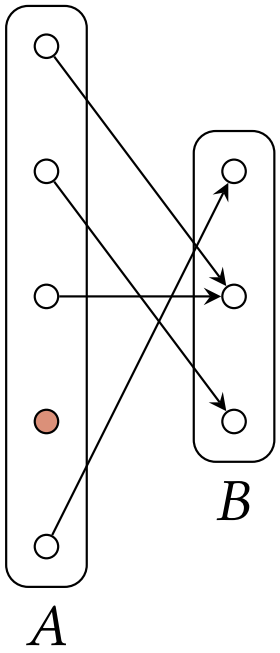
Eine technische Vorbemerkung

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

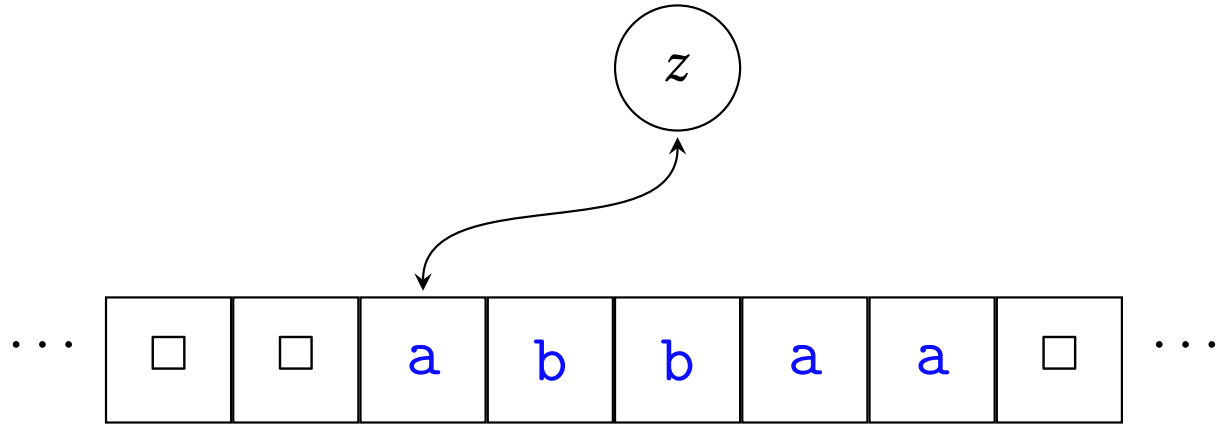
Turingmaschinen: Ursprung

eingeführt von Alan Turing (1912 – 1954)
`http://www.turing.org.uk/turing/index.html`

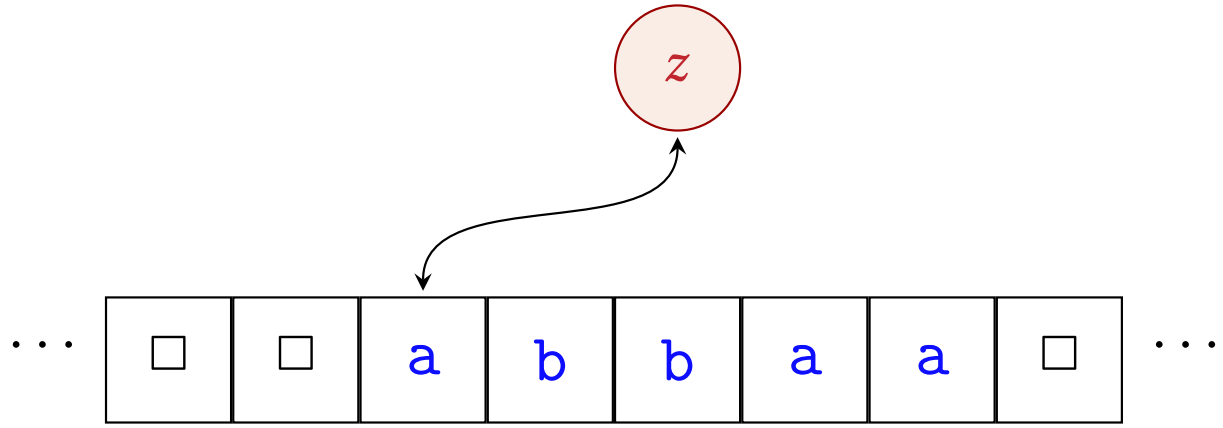
„On computable numbers, with an application to the Entscheidungsproblem“
Proceedings of the London Mathematical Society **42**, 1936, S. 230–265.

„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`

Eine Turingmaschine im Bild

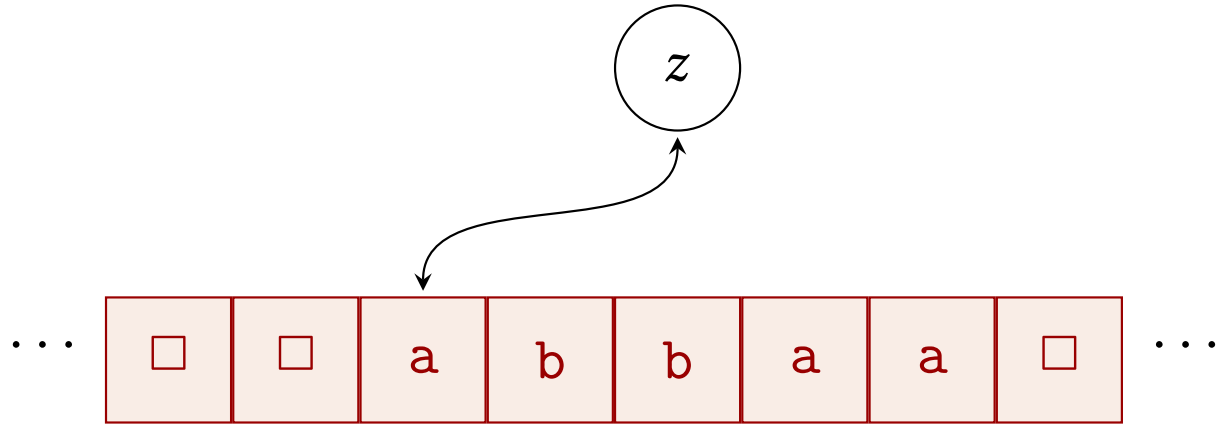


Eine Turingmaschine im Bild



Steuereinheit: endliche Zustandsmenge Z

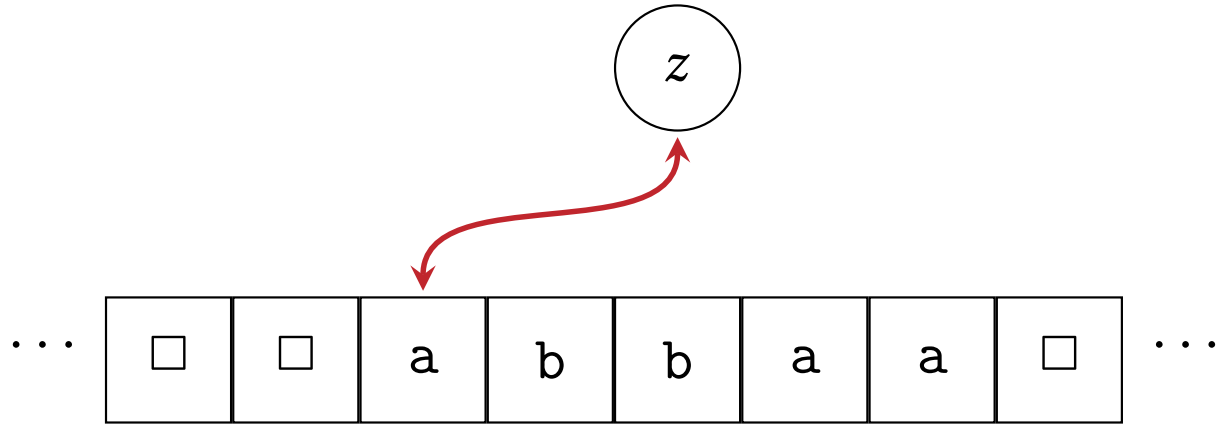
Eine Turingmaschine im Bild



endliche Zustandsmenge Z

Band, in Felder unterteilt:
beschriftet mit Symbolen aus Bandalphabet X

Eine Turingmaschine im Bild



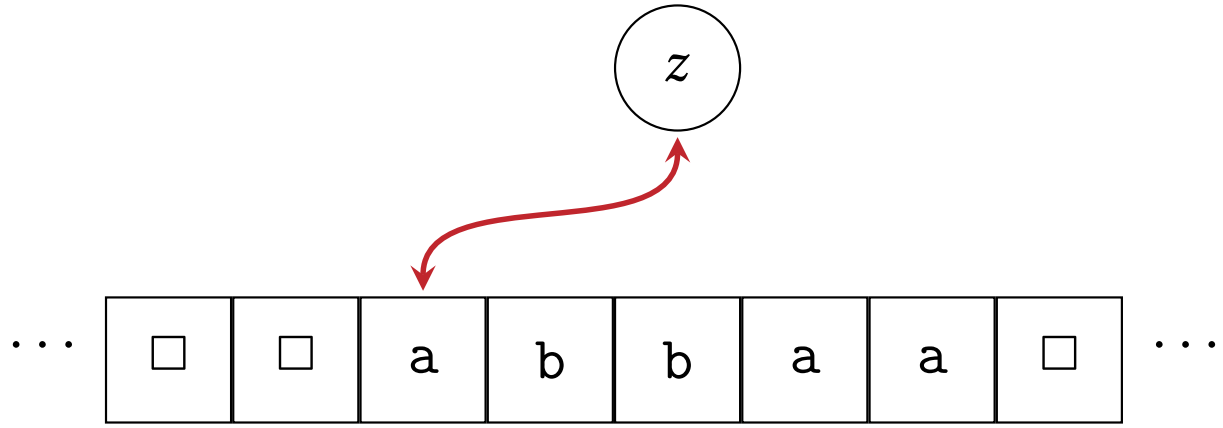
endliche Zustandsmenge Z

Bandalphabet X

nächste Aktion festgelegt durch

- aktuellen Zustand z und
- aktuell gelesenes Symbol x

Eine Turingmaschine im Bild



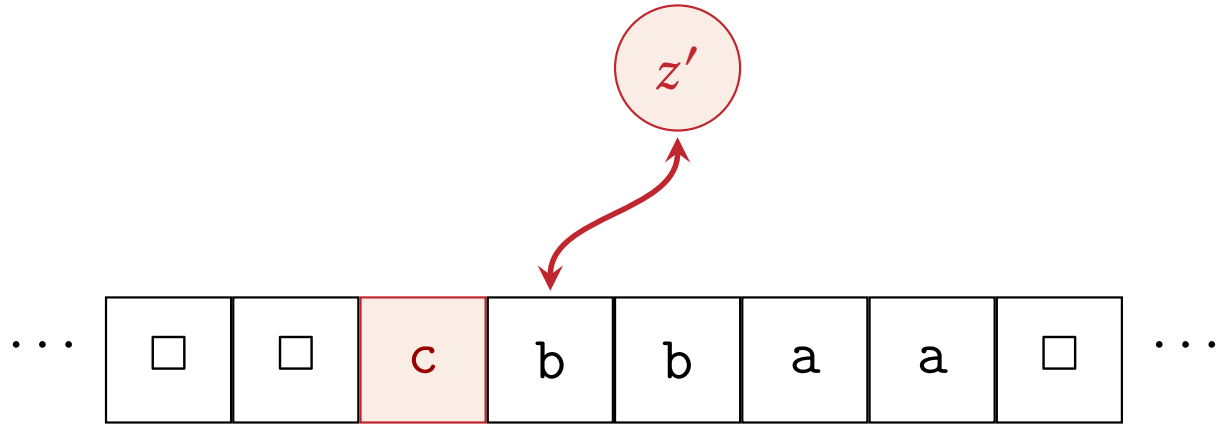
endliche Zustandsmenge Z

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

- Anfangszustand $z_0 \in Z$

Bandalphabet X

- meist mit Blanksymbol $\square$

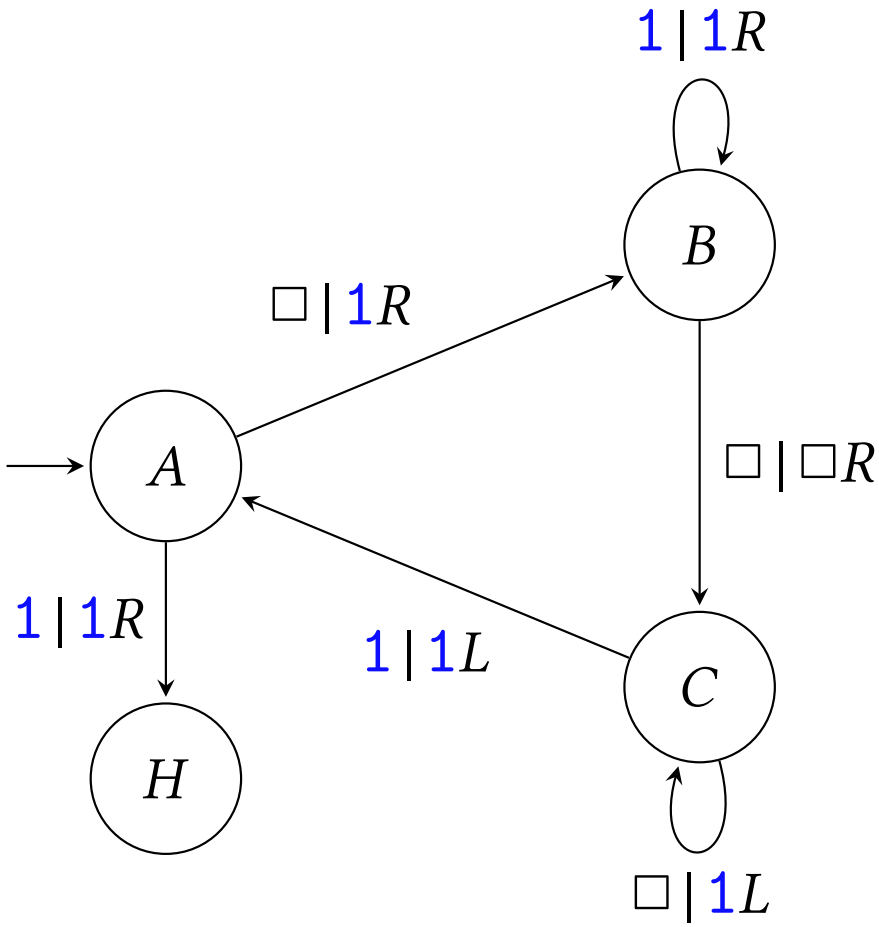
partielle Zustandsüberführungsfunktion
$f : Z \times X \dashrightarrow Z$

partielle Ausgabefunktion
$g : Z \times X \dashrightarrow X$ und

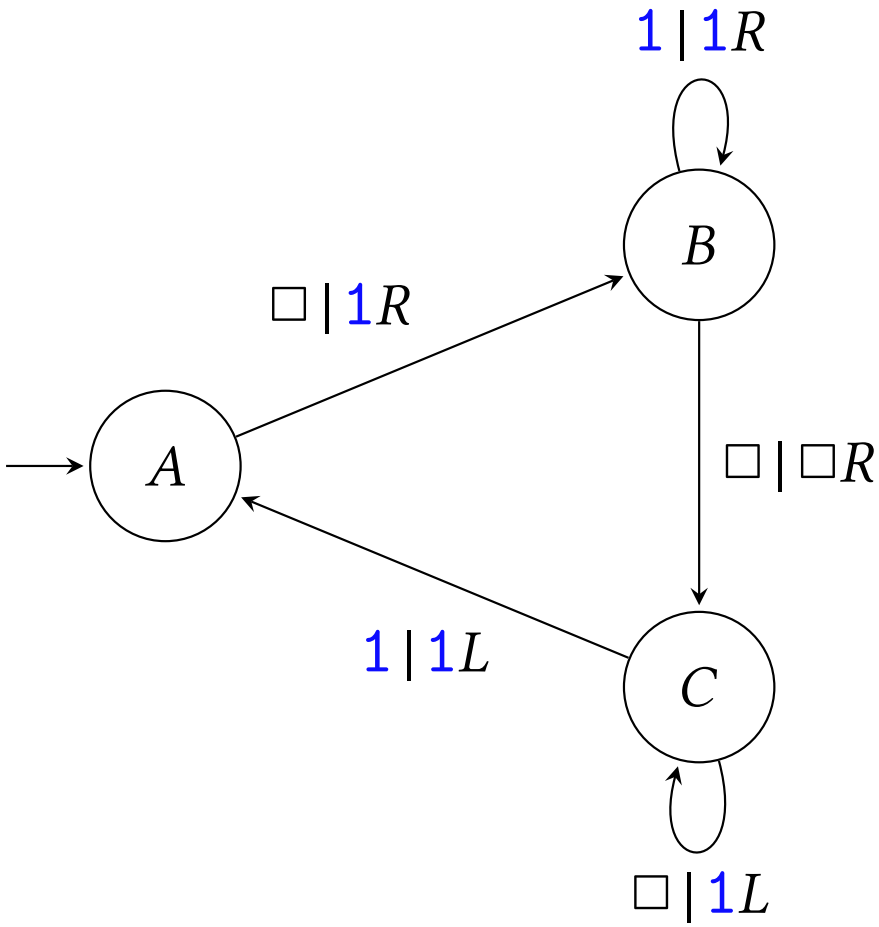
partielle Bewegungsfunktion
$m : Z \times X \dashrightarrow \{-1, 0, 1\}$ oder $\{L, 0, R\}$

f, g, m mit gleichem Definitionsbereich

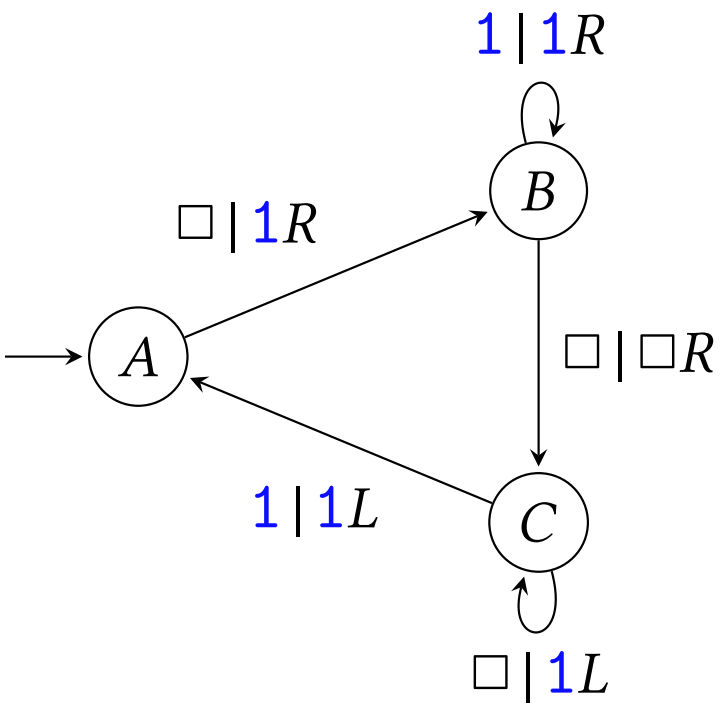
Turingmaschinen: graphische Darstellung (TM BB3)



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

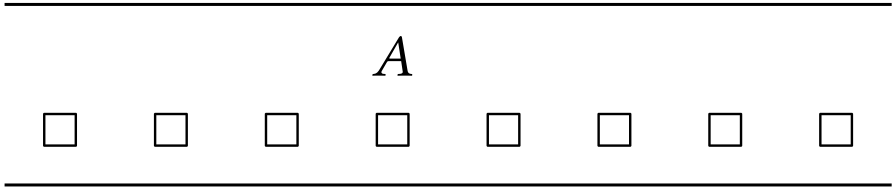
oder so:

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

oder ...

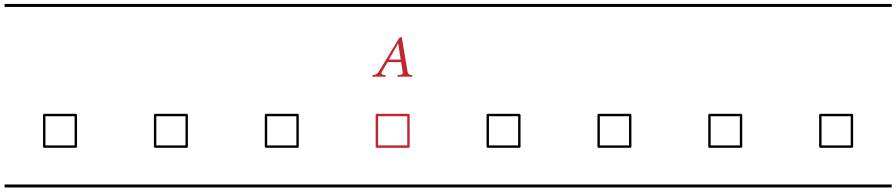
Beispielberechnung (1)

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



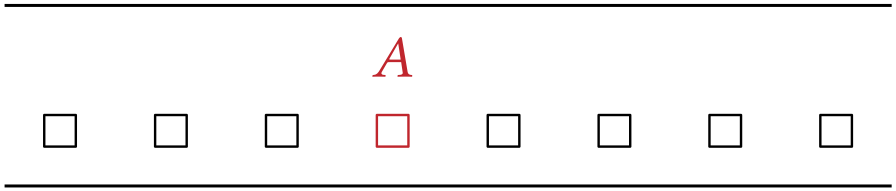
Beispielberechnung (1)

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



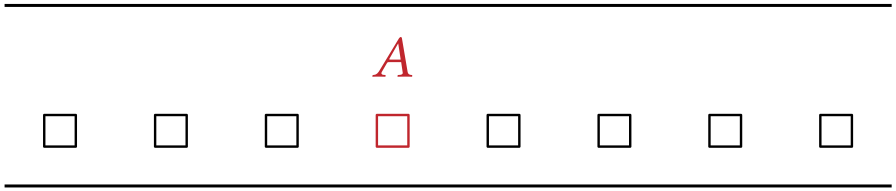
Beispielberechnung (1)

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



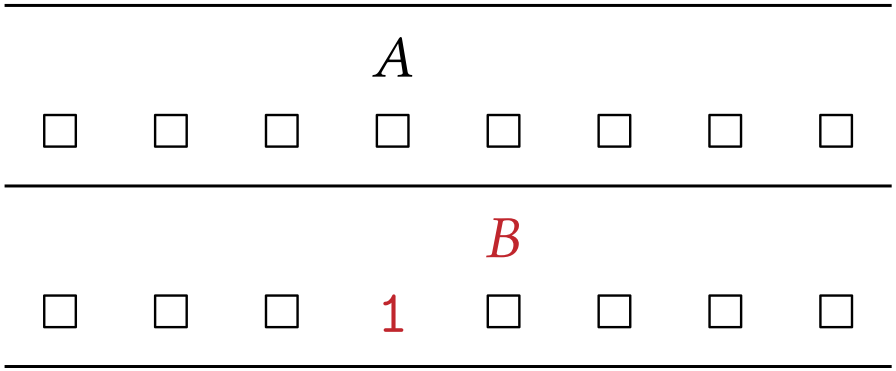
Beispielberechnung (1)

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

A

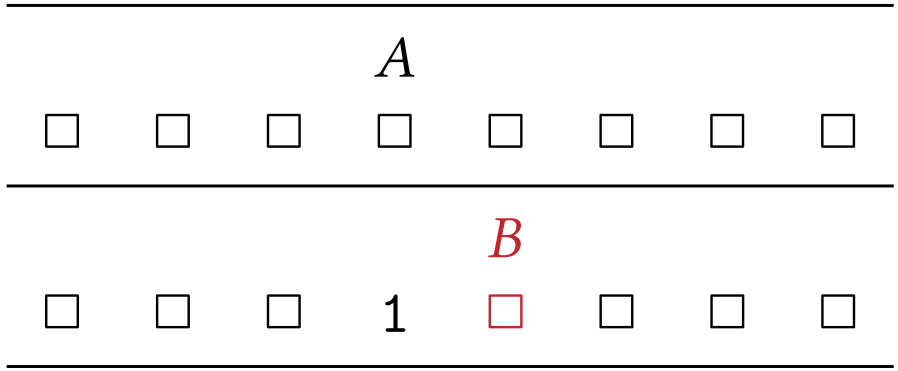
$\square \quad \square \quad \square \quad \square \quad \square \quad \square \quad \square \quad \square$

B

$\square \quad \square \quad \square \quad 1 \quad \square \quad \square \quad \square \quad \square$

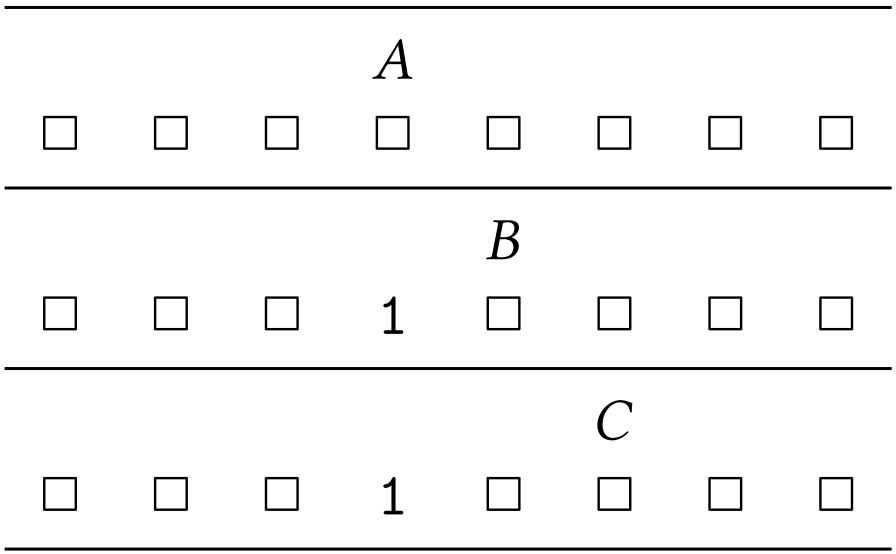
Beispielberechnung (4)

	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 (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

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

wenn für $(z, b(p))$ die Funktionen f, g und m definiert,
dann kann die TM einen Schritt machen

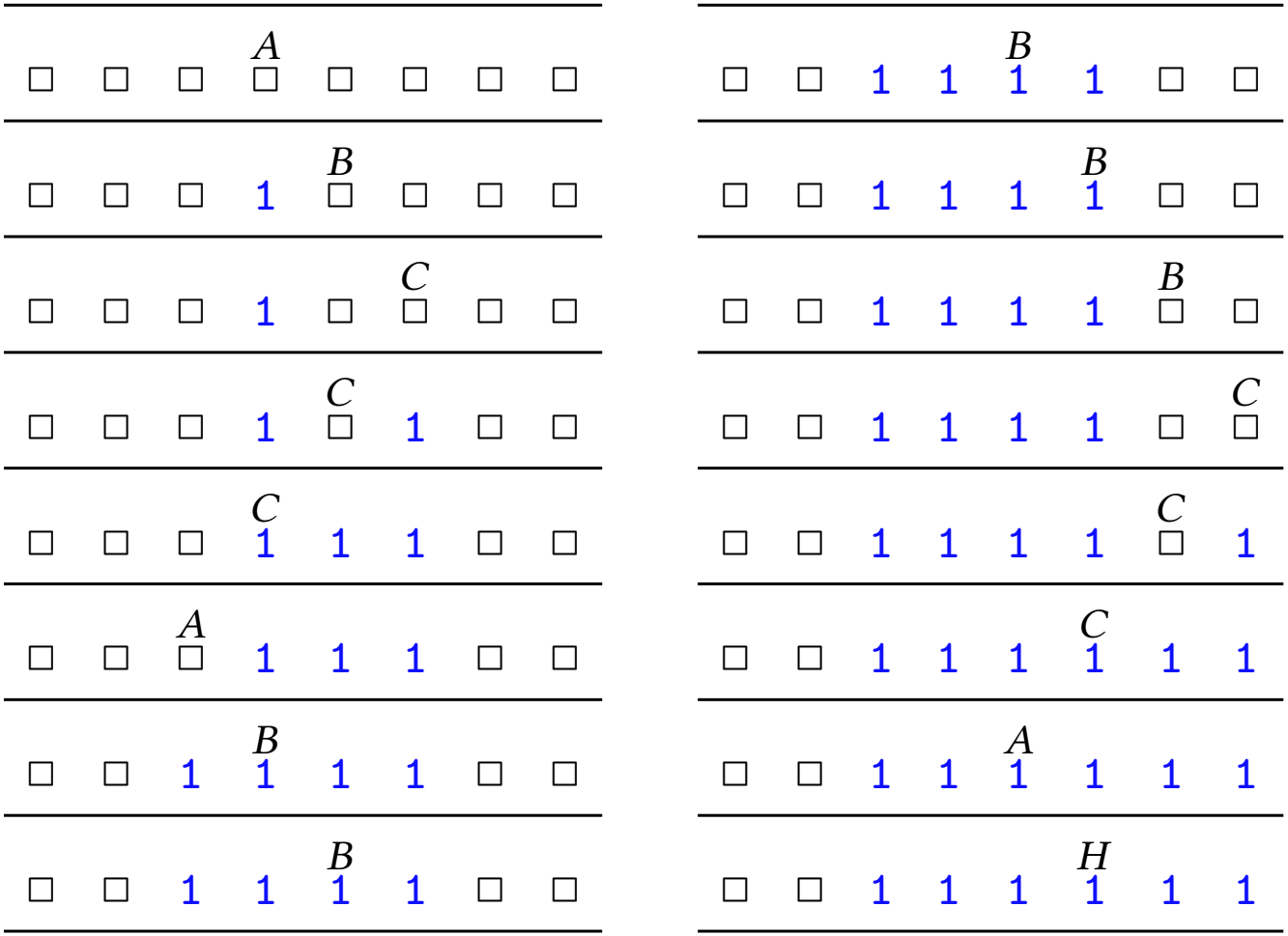
Nachfolgekonfiguration $c' = (z', b', p')$:

- $z' = f(z, b(p))$
- $\forall i \in \mathbb{Z} : b'(i) = \begin{cases} b(i) & \text{falls } i \neq p \\ g(z, b(p)) & \text{falls } i = p \end{cases}$
- $p' = p + m(z, b(p))$

schreiben $c' = \Delta_1(c)$, also

- $\Delta_1 : C_T \dashrightarrow C_T$

Längere Beispielberechnung von BB3



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 $\forall i > 0 : c_i = \Delta_1(c_{i-1})$
- simples Beispiel
  - $f(z, x) = z$,
  - $g(z, x) = x$ 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

- Haltezeitpunkt gegebenenfalls 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
- Entscheidungsprobleme

Eingaben und Anfangskonfigurationen

Eingabealphabet $A \subset X \setminus \{\square\}$

- 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. □□□`[1011]``[101]`□□ oder …

Ergebnisse von Turingmaschinenberechnungen

Berechung von Funktionen

- ein Ausgabewort
- auf dem ansonsten leeren Band

Erkennung formaler Sprachen: ein Bit akzeptiert/abgelehnt

Turingmaschinenakzeptor T:

- Teilmenge $F \subset Z$ 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»

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»

bei dem also

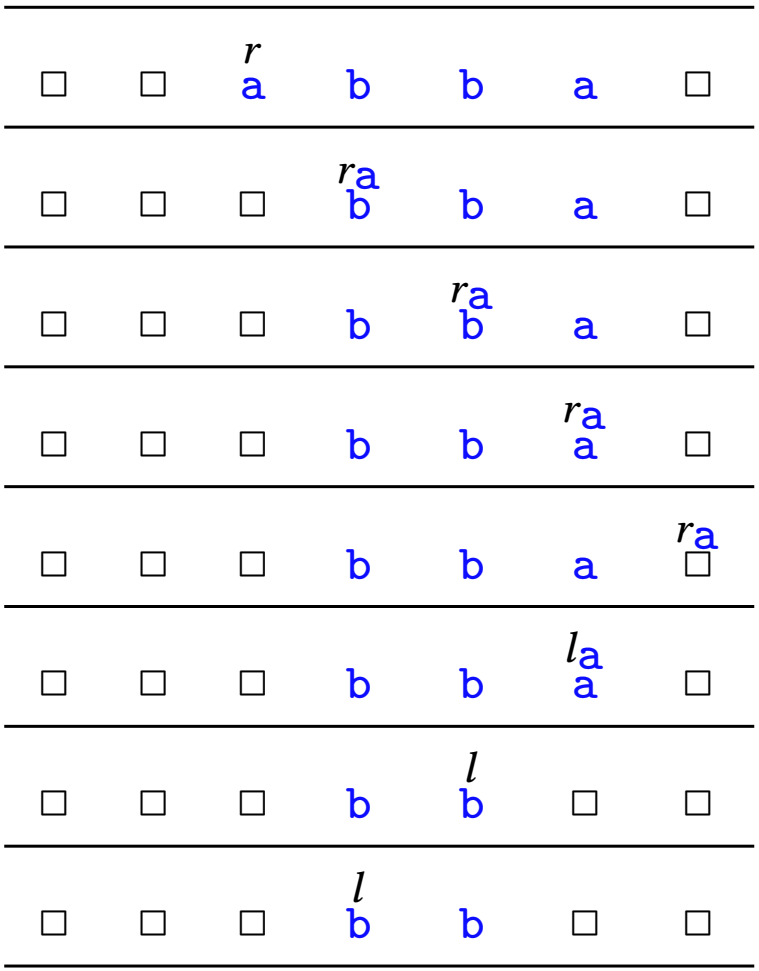
- erstes und letztes Symbol übereinstimmen
- zweites und vorletztes Symbol übereinstimmen
- usw.

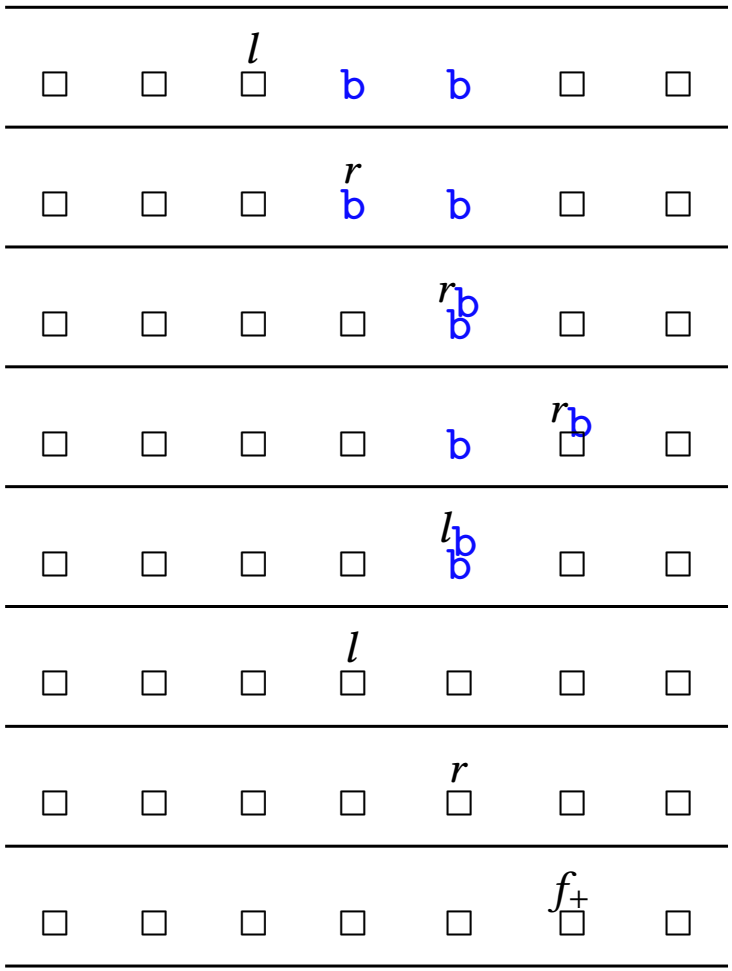
bei dem also

- erstes und letztes Symbol übereinstimmen
- und dazwischen ein Palindrom steht

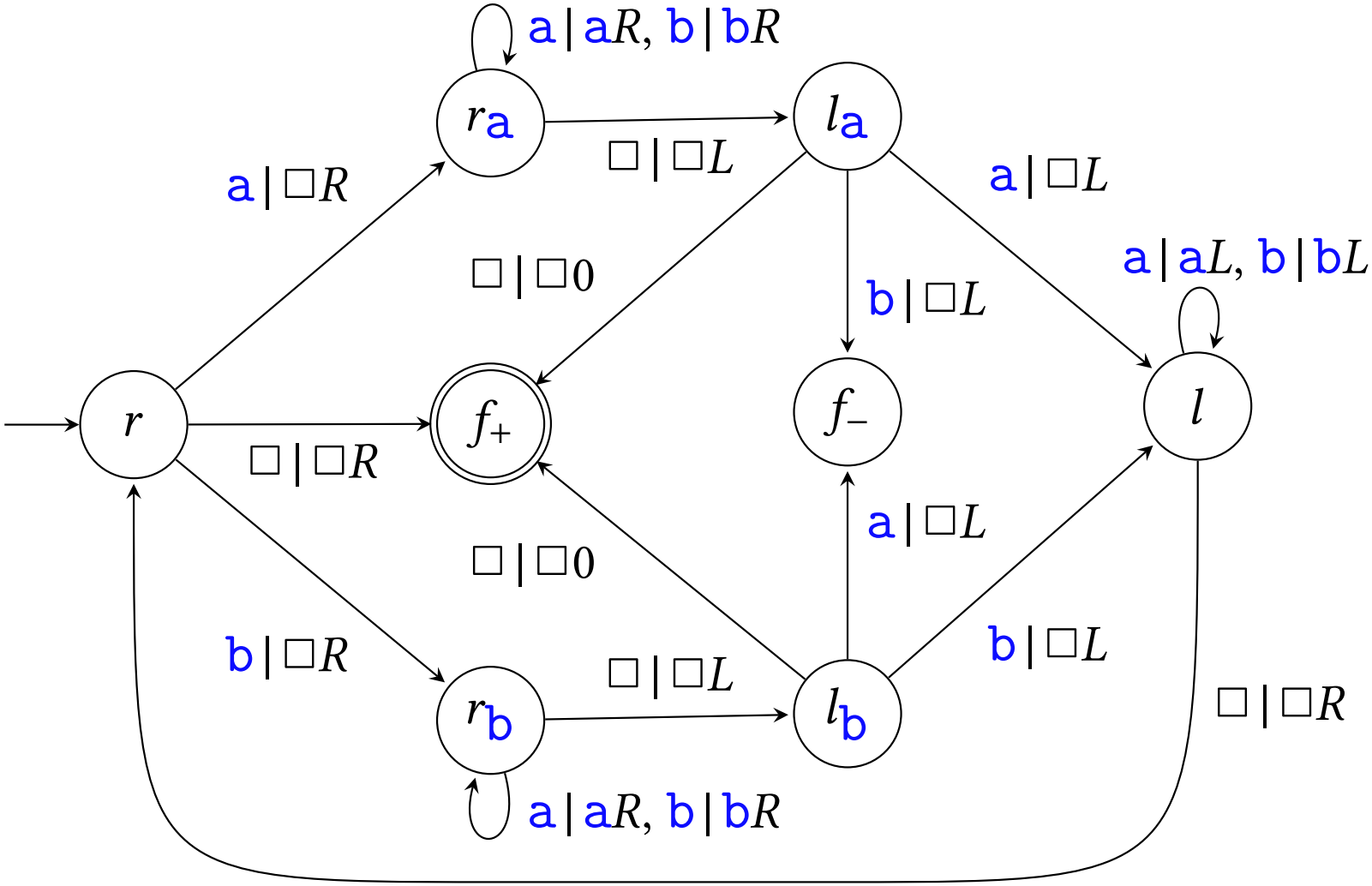
Palindromerkennung: Beispielberechnung

Palindromerkennung: Beispielberechnung

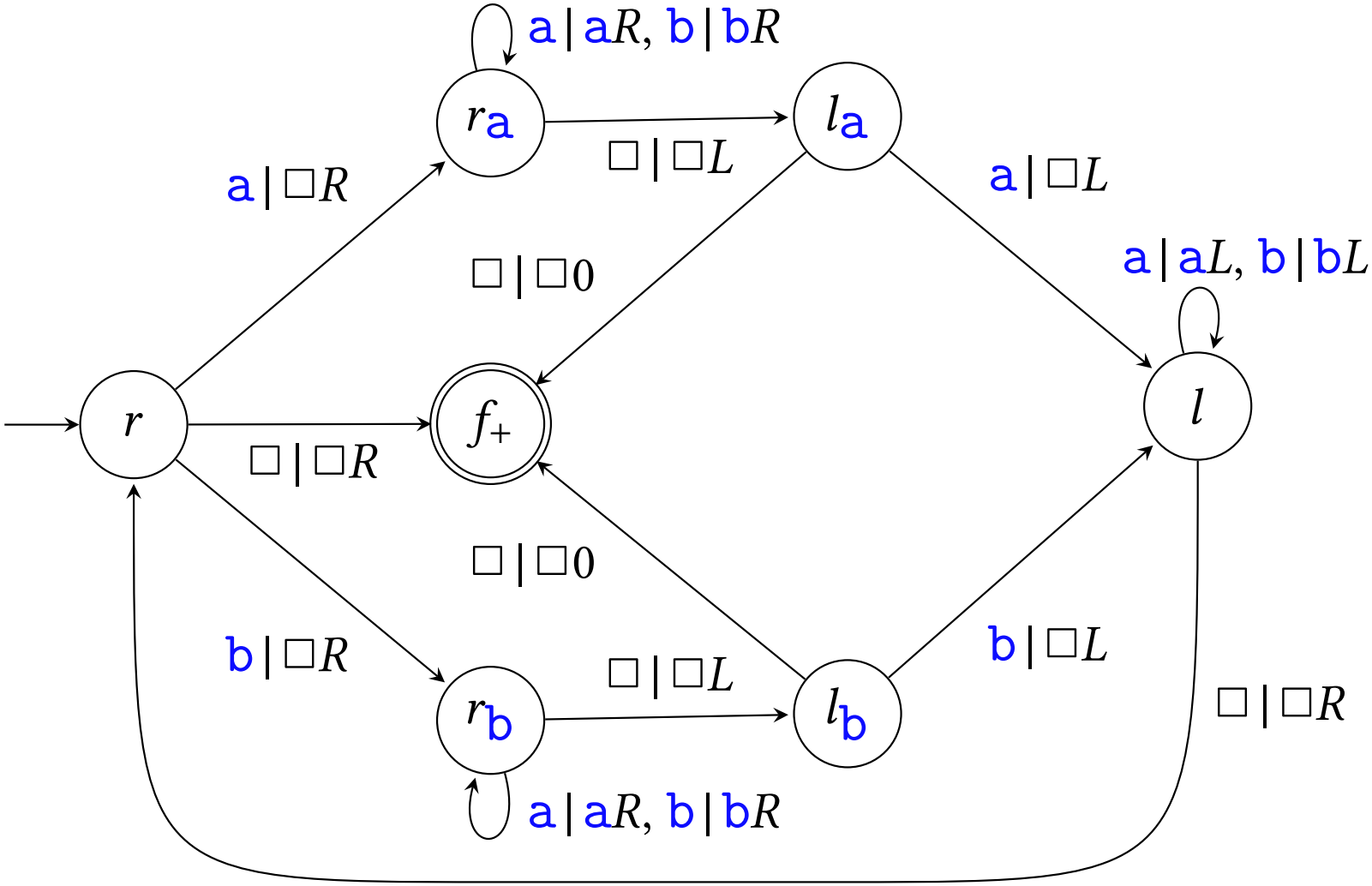




Palindromerkennung: Beispielturingmaschine



Palindromerkennung: Beispielturingmaschine



Aufzählbare und entscheidbare 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 Turingmaschine gibt, *die immer hält* und L akzeptiert.
2. L heißt aufzählbare Sprache, wenn es eine Turingmaschine gibt, die L akzeptiert.

Entscheidbarkeit ist eine 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?

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

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

wie folgt:

$\text{time}_T(w)$ = das t, für das $\Delta_t(c_0(w))$ Endkonfiguration

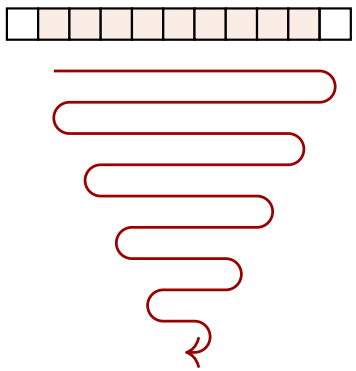
$\text{Time}_T(n) = \max\{\text{time}_T(w) \mid w \in A^n\}$

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

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

wie folgt

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

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 der zur Verfügung stehen 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 $O\left(n^3\right)$ 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 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 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 viel schnellere äquivalente TM existiert

Bei **P** und **PSPACE** geht es um formale Sprachen, nicht um Turingmaschinen.

großes offenes Problem:

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

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?

Achtung Achtung Achtung Achtung Achtung Achtung

Ab sofort ist es wieder von Bedeutung, dass
Turingmaschinen in «Endlosschleifen» laufen können.

Codierungen von Turingmaschinen

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 $A = \{\texttt{1}\}$

Codierungen von Turingmaschinen

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) — Zustände

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

Codierung der Zustände:

- Zustände ab 0 durchnummeriert.
- Anfangszustand Nummer 0.

Repräsentation der Zustände durch
gleich lange Binärdarstellungen der Nummern in [ ]

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

Beispielcodierung (2) — Symbole

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

Codierung der Symbole:

- Bandsymbole ab 0 durchnummeriert.
- Blanksymbol bekommt Nummer 0.

Repräsentation der Bandsymbole durch
gleich lange Binärdarstellungen der Nummern in `[ ]`

- schreibe $\mathrm{cod}_X(x)$ für Codierung von Bandsymbol x
- Beispiele:
  - $\mathrm{cod}_X(\square) =$ `[0000]`
  - $\mathrm{cod}_X($`a`$) =$ `[0001]`
  - ...

Beispielcodierung (3) — Kopfbewegungen

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

Codierung der Kopfbewegungen

- `[10]`, `[00]` und `[01]` für -1, 0, $+1$ resp.
- schreibe $\mathrm{cod}_M(r)$ für Codierung der Bewegungsrichtung r.
- also
  - $\mathrm{cod}_M(-1) =$ `[10]`
  - $\mathrm{cod}_M(0) =$ `[00]`
  - $\mathrm{cod}_M(+1) =$ `[01]`

Beispielcodierung (4) — einzelne Funktionswerte

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

Beispielcodierung (5) — eine ganze Turingmaschine

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

Codierung der gesamten Funktionen:
Konkatenation aller $\text{cod}_{fgm}(z, x)$ für alle $z \in Z$, $x \in X$.

Codierung der gesamten Turingmaschine:
Konkatenation von

- Codierung des Zustands mit der größten Nummer,
- Codierung des Bandsymbols mit der größten Nummer und
- Codierung der gesamten Funktionen f, g und m
- alles eingerahmt von [ ]

T_w bezeichne die Turingmaschine mit Codierung w

Eigenschaften dieser und ähnlicher Codierungen

einfache Syntaxanalyse ist möglich

- man kann TM konstruieren, die für $w \in A^*$ feststellt, ob es die Codierung einer TM ist oder nicht.

Eigenschaften dieser und ähnlicher Codierungen

einfache Syntaxanalyse ist möglich

- man kann TM konstruieren, die für $w \in A^*$ feststellt, ob es die Codierung einer TM ist oder nicht.

universelle Turingmaschine U existiert

- erhält als Eingabe Wort $[w_1]\,[w_2]$
- prüft, ob
  - w_1 Codierung einer Turingmaschine T ist und gegebenenfalls
  - w_2 Codierung einer Eingabe für T_{w_1}

Eigenschaften dieser und ähnlicher Codierungen

einfache Syntaxanalyse ist möglich

- man kann TM konstruieren, die für $w \in A^*$ feststellt, ob es die Codierung einer TM ist oder nicht.

universelle Turingmaschine U existiert

- erhält als Eingabe Wort $[w_1]\,[w_2]$
- prüft, ob
  - w_1 Codierung einer Turingmaschine T ist und gegebenenfalls
  - w_2 Codierung einer Eingabe für T_{w_1}
- falls nein: entsprechende Mitteilung und HALT

Eigenschaften dieser und ähnlicher Codierungen

einfache Syntaxanalyse ist möglich

- man kann TM konstruieren, die für $w \in A^*$ feststellt, ob es die Codierung einer TM ist oder nicht.

universelle Turingmaschine U existiert

- erhält als Eingabe Wort $[w_1]\,[w_2]$
- prüft, ob
  - w_1 Codierung einer Turingmaschine T ist und gegebenenfalls
  - w_2 Codierung einer Eingabe für T_{w_1}
- falls nein: entsprechende Mitteilung und HALT
- falls ja:
  - U simuliert Schritt für Schritt die Arbeit, die T_{w_1} für Eingabe w_2 durchführen würde,
  - und falls T_{w_1} endet, liefert U am Ende als Ergebnis das, was T_{w_1} liefern würde.

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** algorithmisch
also **NICHT MIT DEM RECHNER** 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) (oder früher …)

- für Überabzählbarkeit von $\mathbb{R}$ benutzt

betrachten erst die Kernidee,
danach die Anwendung auf Halteproblem

Diagonalisierung: Kernidee (1)

«zweidimensionale unendliche Tabelle»

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

	w_0	w_1	w_2	w_3	w_4	$\cdots$
f_0	$f_0(w_0)$	$f_0(w_1)$	$f_0(w_2)$	$f_0(w_3)$	$f_0(w_4)$	$\cdots$
f_1	$f_1(w_0)$	$f_1(w_1)$	$f_1(w_2)$	$f_1(w_3)$	$f_1(w_4)$	$\cdots$
f_2	$f_2(w_0)$	$f_2(w_1)$	$f_2(w_2)$	$f_2(w_3)$	$f_2(w_4)$	$\cdots$
f_3	$f_3(w_0)$	$f_3(w_1)$	$f_3(w_2)$	$f_3(w_3)$	$f_3(w_4)$	$\cdots$
f_4	$f_4(w_0)$	$f_4(w_1)$	$f_4(w_2)$	$f_4(w_3)$	$f_4(w_4)$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Diagonalisierung: Kernidee (2)

	w_0	w_1	w_2	w_3	w_4	$\cdots$
f_0	$f_0(w_0)$	$f_0(w_1)$	$f_0(w_2)$	$f_0(w_3)$	$f_0(w_4)$	$\cdots$
f_1	$f_1(w_0)$	$f_1(w_1)$	$f_1(w_2)$	$f_1(w_3)$	$f_1(w_4)$	$\cdots$
f_2	$f_2(w_0)$	$f_2(w_1)$	$f_2(w_2)$	$f_2(w_3)$	$f_2(w_4)$	$\cdots$
f_3	$f_3(w_0)$	$f_3(w_1)$	$f_3(w_2)$	$f_3(w_3)$	$f_3(w_4)$	$\cdots$
f_4	$f_4(w_0)$	$f_4(w_1)$	$f_4(w_2)$	$f_4(w_3)$	$f_4(w_4)$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Diagonalisierung: Kernidee (2)

	w_0	w_1	w_2	w_3	w_4	$\cdots$
f_0	$f_0(w_0)$	$f_0(w_1)$	$f_0(w_2)$	$f_0(w_3)$	$f_0(w_4)$	$\cdots$
f_1	$f_1(w_0)$	$f_1(w_1)$	$f_1(w_2)$	$f_1(w_3)$	$f_1(w_4)$	$\cdots$
f_2	$f_2(w_0)$	$f_2(w_1)$	$f_2(w_2)$	$f_2(w_3)$	$f_2(w_4)$	$\cdots$
f_3	$f_3(w_0)$	$f_3(w_1)$	$f_3(w_2)$	$f_3(w_3)$	$f_3(w_4)$	$\cdots$
f_4	$f_4(w_0)$	$f_4(w_1)$	$f_4(w_2)$	$f_4(w_3)$	$f_4(w_4)$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$
d	$f_0(w_0)$	$f_1(w_1)$	$f_2(w_2)$	$f_3(w_3)$	$f_4(w_4)$	$\cdots$

Diagonalisierung: Kernidee (2)

	w_0	w_1	w_2	w_3	w_4	$\cdots$
f_0	$f_0(w_0)$	$f_0(w_1)$	$f_0(w_2)$	$f_0(w_3)$	$f_0(w_4)$	$\cdots$
f_1	$f_1(w_0)$	$f_1(w_1)$	$f_1(w_2)$	$f_1(w_3)$	$f_1(w_4)$	$\cdots$
f_2	$f_2(w_0)$	$f_2(w_1)$	$f_2(w_2)$	$f_2(w_3)$	$f_2(w_4)$	$\cdots$
f_3	$f_3(w_0)$	$f_3(w_1)$	$f_3(w_2)$	$f_3(w_3)$	$f_3(w_4)$	$\cdots$
f_4	$f_4(w_0)$	$f_4(w_1)$	$f_4(w_2)$	$f_4(w_3)$	$f_4(w_4)$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$
d	$f_0(w_0)$	$f_1(w_1)$	$f_2(w_2)$	$f_3(w_3)$	$f_4(w_4)$	$\cdots$
$\overline{d}$	$\overline{f_0(w_0)}$	$\overline{f_1(w_1)}$	$\overline{f_2(w_2)}$	$\overline{f_3(w_3)}$	$\overline{f_4(w_4)}$	$\cdots$

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

Diagonalisierung: Kernidee (2)

	w_0	w_1	w_2	w_3	w_4	$\cdots$
f_0	$f_0(w_0)$	$f_0(w_1)$	$f_0(w_2)$	$f_0(w_3)$	$f_0(w_4)$	$\cdots$
f_1	$f_1(w_0)$	$f_1(w_1)$	$f_1(w_2)$	$f_1(w_3)$	$f_1(w_4)$	$\cdots$
f_2	$f_2(w_0)$	$f_2(w_1)$	$f_2(w_2)$	$f_2(w_3)$	$f_2(w_4)$	$\cdots$
f_3	$f_3(w_0)$	$f_3(w_1)$	$f_3(w_2)$	$f_3(w_3)$	$f_3(w_4)$	$\cdots$
f_4	$f_4(w_0)$	$f_4(w_1)$	$f_4(w_2)$	$f_4(w_3)$	$f_4(w_4)$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$
d	$f_0(w_0)$	$f_1(w_1)$	$f_2(w_2)$	$f_3(w_3)$	$f_4(w_4)$	$\cdots$
$\overline{d}$	$\overline{f_0(w_0)}$	$\overline{f_1(w_1)}$	$\overline{f_2(w_2)}$	$\overline{f_3(w_3)}$	$\overline{f_4(w_4)}$	$\cdots$

$\overline{d}$ 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.

Beweis der Unentscheidbarkeit des Halteproblems (1)

benutze (eine Variante der) Diagonalisierung

in der Tabelle

- Spalten w_i: *alle* Codierungen von Turingmaschinen
- Zeilen f_i Abbildung mit

$$f_i(w_j) = \begin{cases} 1, & \text{falls } T_{w_i} \text{ für Eingabe } w_j \text{ hält} \\ 0, & \text{sonst} \end{cases}$$

für *jede* Turingmaschine eine Zeile

Beweis der Unentscheidbarkeit des Halteproblems (1)

benutze (eine Variante der) Diagonalisierung

in der Tabelle

- Spalten w_i: *alle* Codierungen von Turingmaschinen
- Zeilen f_i Abbildung mit

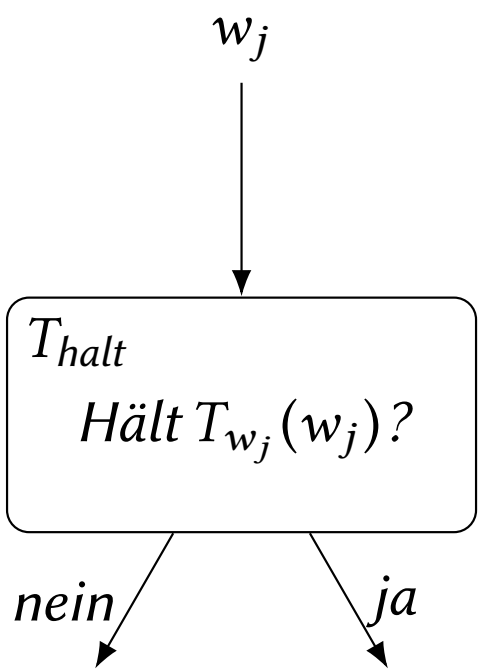
$$f_i(w_j) = \begin{cases} 1, & \text{falls } T_{w_i} \text{ für Eingabe } w_j \text{ hält} \\ 0, & \text{sonst} \end{cases}$$

für *jede* Turingmaschine eine Zeile

indirekter Beweis:

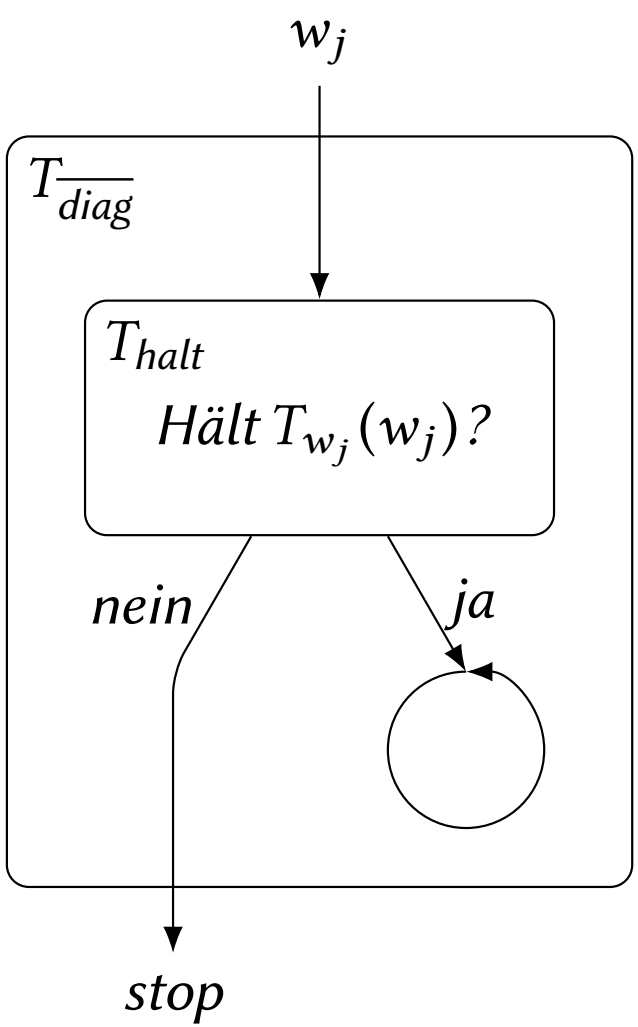
- Annahme: eine TM T_{halt} *entscheidet* H
- dann existiert TM $T_{\overline{diag}}$ für die «verdorbene Diagonale»
- Widerspruch: verdorbene Diagonale nicht in Tabelle

Beweis der Unentscheidbarkeit des Halteproblems (2)



wenn es Turingmaschine T_{halt} gäbe,

Beweis der Unentscheidbarkeit des Halteproblems (2)

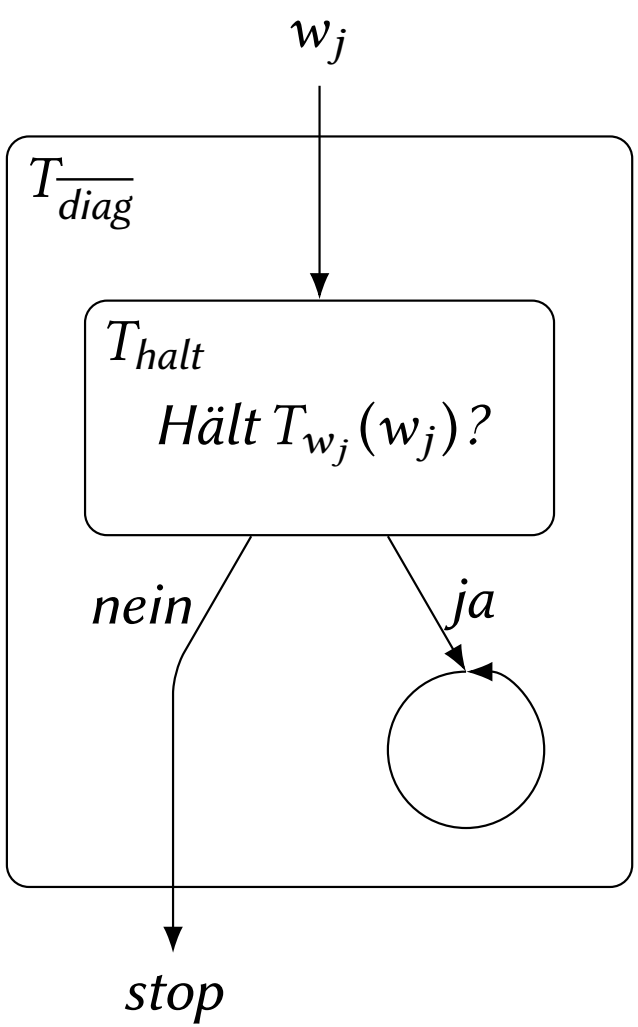


wenn es Turingmaschine T_{halt} gäbe,

dann auch Turingmaschie $T_{\overline{diag}}$

- für Eingabe w_j
  berechne $T_{halt}(w_j)$
- anschließend
  - würde $T_{w_j}(w_j)$ nicht halten, dann halte
  - würde $T_{w_j}(w_j)$ halten, dann halte nicht

Beweis der Unentscheidbarkeit des Halteproblems (2)



wenn es Turingmaschine T_{halt} gäbe,

dann auch Turingmaschie $T_{\overline{diag}}$

- für Eingabe w_j
  berechne $T_{halt}(w_j)$
- anschließend
  - würde $T_{w_j}(w_j)$ nicht halten, dann halte
  - würde $T_{w_j}(w_j)$ halten, dann halte nicht
- für jedes w_j in beiden Fällen
  verhält sich $T_{\overline{diag}}$ anders als T_{w_j}
- also wäre TM $T_{\overline{diag}}$ anders als jede TM

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 TM jemals gebraucht?

- Erreichbarkeit von Codestücken unentscheidbar

vieles vieles vieles vieles vieles vieles vieles mehr

Beachte: statt Turingmaschine kann man immer Java-Programm einsetzen

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)

$$\mathrm{bb} : \mathbb{N}_+ \to \mathbb{N}_+$$

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

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		
$\vdots$		

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		
⋮		

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	$> 3.514 \cdot 10^{18276}$	Kropitz (2010)
$\vdots$	$\vdots$	

Die Busy-Beaver-Funktion ist nicht berechenbar

Satz

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

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.“*

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.