TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik, Institut für Künstliche Intelligenz, Professur Computational Logic

THEORETISCHE INFORMATIK UND LOGIK

2. Vorlesung: Berechenbarkeit und Unentscheidbarkeit

Hannes Straß

TU Dresden, 7. April 2022

Was bisher geschah . . .

Hilbert: „Die Mathematik hat mehr als ein Problem. Wir brauchen ein formales Fundament! Dann kann jedes mathematische Problem durch endlich viele ‚Rechenschritte' gelöst werden!"

Turing: „Es gibt Dinge, die man nicht berechnen kann. Ich kann das beweisen . . . aber erst einmal muss ich definieren, was ‚berechnen' eigentlich bedeutet."

Church und Turing (im Chor mit Kleene und Rosser):
„Alle Computer sind gleich!"

Wichtige Typen von Turingmaschinen

- Deterministische Turingmaschine (DTM)
- Nichtdeterministische Turingmaschine (NTM)
- Mehrband-Turingmaschine
  Wir ändern die Definition so, dass statt einem Band in jedem Schritt $k \geq 2$ Bänder gelesen und beschrieben werden.
  Jedes Band hat einen unabhängigen Lese-/Schreibkopf.
  Die Eingabe wird auf das erste Band geschrieben.

Satz: Deterministische und nichtdeterministische Turingmaschinen mit einer beliebigen Anzahl von Bändern können die gleichen Berechnungen ausführen.

Details siehe Vorlesung Formale Systeme (Vorlesungen 18 und 19)

Grundbegriffe: Berechnen und Entscheiden

TMs, die Mengen definieren

Eine Turingmaschine kann eine Menge von akzeptierten Eingaben definieren:

Die von einer TM $\mathcal{M}$ erkannte Sprache $\mathbf{L}(\mathcal{M})$ ist die Menge aller Wörter, die von einer TM akzeptiert werden, d.h., bei deren Eingabe $\mathcal{M}$ in einem Endzustand hält (DTM) / halten könnte (NTM).

TMs, die Mengen definieren

Eine Turingmaschine kann eine Menge von akzeptierten Eingaben definieren:

Die von einer TM $\mathcal{M}$ erkannte Sprache $\mathbf{L}(\mathcal{M})$ ist die Menge aller Wörter, die von einer TM akzeptiert werden, d.h., bei deren Eingabe $\mathcal{M}$ in einem Endzustand hält (DTM) / halten könnte (NTM).

Es gibt zwei mögliche Gründe für Nichtakzeptanz von Wörtern:

(1) TM hält in einem Zustand, der kein Endzustand ist

(2) TM hält nicht (Endlosschleife)

Es ist praktisch, wenn eine TM garantiert hält, da man Fall (2) meist nicht sicher erkennen kann (man weiß nicht, ob die TM irgendwann doch noch anhält).

Eine TM ist genau dann ein Entscheider, wenn sie bei jeder Eingabe garantiert hält. (Bei NTMs: In jedem möglichen Lauf.)
Wir sagen in diesem Fall, dass die TM die von ihr erkannte Sprache entscheidet.

TMs, die Funktionen definieren

Eine Turingmaschine kann eine Funktion von Eingabewörtern auf Ausgabewörter definieren:

Eine DTM $\mathcal{M}$ berechnet eine partielle Funktion $f_{\mathcal{M}} : \Sigma^* \to \Sigma^*$ wie folgt. Für ein Wort $w \in \Sigma^*$ ist genau dann $f_{\mathcal{M}}(w) = v$, wenn $\mathcal{M}$ bei Eingabe w mit einem Band anhält, auf dem nur v steht, d.h., wenn der Bandinhalt nach dem Halten die Form $v\sqcup\sqcup\sqcup\cdots$ hat.

TMs, die Funktionen definieren

Eine Turingmaschine kann eine Funktion von Eingabewörtern auf Ausgabewörter definieren:

Eine DTM $\mathcal{M}$ berechnet eine partielle Funktion $f_{\mathcal{M}} : \Sigma^* \to \Sigma^*$ wie folgt. Für ein Wort $w \in \Sigma^*$ ist genau dann $f_{\mathcal{M}}(w) = v$, wenn $\mathcal{M}$ bei Eingabe w mit einem Band anhält, auf dem nur v steht, d.h., wenn der Bandinhalt nach dem Halten die Form $v\sqcup\sqcup\sqcup\cdots$ hat.

Es gibt also zwei Fälle, in denen $f_{\mathcal{M}}(w)$ undefiniert ist:

(1) $\mathcal{M}$ hält bei Eingabe w mit einem Band, das nicht die geforderte Form hat;

(2) $\mathcal{M}$ hält bei Eingabe w gar nicht.

Das heißt: Wenn $f_{\mathcal{M}}$ eine totale Funktion ist, dann muss $\mathcal{M}$ immer halten.

(Frage zur Selbstkontrolle: Gilt auch die Umkehrung – d.h. ist $f_{\mathcal{M}}$ immer eine totale Funktion, wenn $\mathcal{M}$ stets hält?)

Entscheidbarkeit und Berechenbarkeit

Wir interessieren uns für Funktionen, die man berechnen kann:

Eine totale oder partielle Funktion $f : \Sigma^* \to \Sigma^*$ heißt genau dann berechenbar, wenn es eine DTM $\mathcal{M}$ gibt, die f berechnet, d.h. für die $f = f_{\mathcal{M}}$ gilt.

- Berechenbare totale Funktionen heißen rekursiv;
- berechenbare partielle Funktionen heißen partiell rekursiv.

Entscheidbarkeit und Berechenbarkeit

Wir interessieren uns für Funktionen, die man berechnen kann:

Eine totale oder partielle Funktion $f : \Sigma^* \to \Sigma^*$ heißt genau dann berechenbar, wenn es eine DTM $\mathcal{M}$ gibt, die f berechnet, d.h. für die $f = f_{\mathcal{M}}$ gilt.

- Berechenbare totale Funktionen heißen rekursiv;
- berechenbare partielle Funktionen heißen partiell rekursiv.

Bei Sprachen unterscheiden wir mehrere Fälle:

Eine Sprache **L** ist entscheidbar (=berechenbar=rekursiv), wenn es eine TM $\mathcal{M}$ gibt, die ihr Wortproblem entscheidet, d.h. $\mathcal{M}$ ist Entscheider und **L** = **L**($\mathcal{M}$).
Andernfalls heißt **L** unentscheidbar.

L ist semi-entscheidbar (=Turing-erkennbar=Turing-akzeptierbar=rekursiv aufzählbar) wenn es eine TM $\mathcal{M}$ mit **L** = **L**($\mathcal{M}$) gibt, auch wenn $\mathcal{M}$ kein Entscheider ist.

Warum heißt es „rekursiv aufzählbar“?

Bei rekursiv aufzählbaren Sprachen **L** kann man eine TM konstruieren, die alle Elemente von **L** der Reihe nach „aufzählt“:

Ein Aufzähler ist eine deterministische Turingmaschine $\mathcal{M}$ mit einem speziellen Zustand q_{Ausgabe}. Immer wenn $\mathcal{M}$ in den Zustand q_{Ausgabe} gelangt, wird der aktuelle Bandinhalt – vom Anfang bis zum ersten Zeichen, nach dem nur noch $\sqcup$ folgen – ausgegeben.

Die durch $\mathcal{M}$ aufgezählte Sprache ist die Menge aller Wörter aus Σ^*, die $\mathcal{M}$ ausgibt, wenn $\mathcal{M}$ auf dem leeren Band gestartet wird.

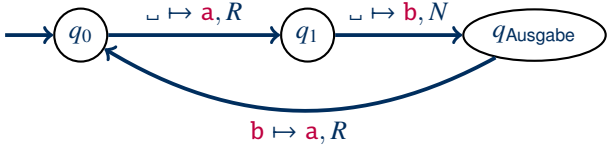
(Anmerkung: Die Definition erlaubt die Ausgabe von Wörtern aus $\Gamma^* \setminus \Sigma^*$. Diese gehören für uns nicht zur aufgezählten Sprache.)

Die durch $\mathcal{M}$ aufgezählte Sprache kann unendlich sein, wenn $\mathcal{M}$ auf der leeren Eingabe nicht hält.

Es ist erlaubt, dass Wörter mehr als einmal in der Aufzählung vorkommen.

Beispiel für einen Aufzähler

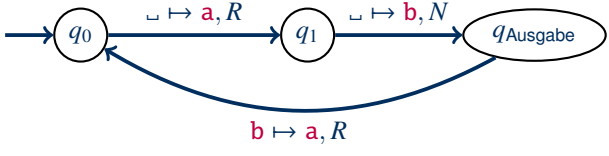
Wir betrachten das Alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}\}$.



Frage: Welche Menge zählt diese TM auf?

Beispiel für einen Aufzähler

Wir betrachten das Alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}\}$.



Frage: Welche Menge zählt diese TM auf?

Antwort: $\left\{\mathtt{a}^{2n+1}\mathtt{b} \;\middle|\; 0 \leq n\right\}$

Aufgabe zur Selbstkontrolle: Schreiben Sie einen Aufzähler für diese Sprache, ohne dabei die Bewegungsrichtung N zu verwenden.

Quiz: Aufzähler

Quiz: Wir betrachten den grafisch angegebenen Aufzähler $\mathcal{M}$ über $\Sigma = \{\mathtt{0}, \mathtt{1}\}$: ...

Aufzählbar = semi-entscheidbar (1)

Satz:
Eine Sprache **L** ist genau dann semi-entscheidbar, wenn es einen Aufzähler für **L** gibt.

Aufzählbar = semi-entscheidbar (1)

Satz:
Eine Sprache **L** ist genau dann semi-entscheidbar, wenn es einen Aufzähler für **L** gibt.

Beweis: „⇐“ (Vom Aufzähler zur TM) Wenn es für **L** einen Aufzähler gibt, dann können wir **L** wie folgt semi-entscheiden:

- Simuliere den Aufzähler auf einem leeren Band (wir dürfen eine Mehrband-TM verwenden).
- Immer wenn q_{Ausgabe} erreicht wird: vergleiche das Eingabeband mit dem Aufzählerband und akzeptiere, wenn beide das gleiche Wort beinhalten.
- Andernfalls fahre mit der Aufzählung fort.
- Falls die Aufzählung terminiert (ohne, dass die Eingabe gefunden wurde), dann verwirf.

Aufzählbar = semi-entscheidbar (2)

Satz:
Eine Sprache **L** ist genau dann semi-entscheidbar, wenn es einen Aufzähler für **L** gibt.

Aufzählbar = semi-entscheidbar (2)

Satz:
Eine Sprache **L** ist genau dann semi-entscheidbar, wenn es einen Aufzähler für **L** gibt.

Beweis: „⇒“ (Von der TM zum Aufzähler) Wenn **L** durch eine TM $\mathcal{M}$ erkannt wird, dann können wir **L** wie folgt aufzählen:

- Betrachte eine systematisch berechenbare Aufzählung aller Wörter $w_1, w_2, w_3, \ldots$ aus Σ^*.
- Für jede natürliche Zahl $n \geq 1$:
  - Für jedes $i \in \{1, \ldots, n\}$:
    - Simuliere $\mathcal{M}$ auf Eingabe w_i für n Schritte.
    - Falls $\mathcal{M}$ bei dieser Simulation terminiert und w_i akzeptiert, dann gib w_i aus.

Anmerkung: Der so konstruierte Aufzähler terminiert nicht (selbst wenn die aufgezählte Menge endlich ist). □

Von der TM zum Aufzähler

Für jede natürliche Zahl $n \geq 1$:

- Für jedes $i \in \{1, \ldots, n\}$:
  - Simuliere $\mathcal{M}$ auf Eingabe w_i für n Schritte.
  - Falls $\mathcal{M}$ bei dieser Simulation terminiert und w_i akzeptiert, dann gib w_i aus.

Diese Art und Weise der Simulation kann folgendermaßen veranschaulicht werden:

$n \setminus \Sigma^*$	w_1	w_2	w_3	w_4	$\ldots$
1					$\ldots$
2					$\ldots$
3					$\ldots$
4					$\ldots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

L = {

Von der TM zum Aufzähler

Für jede natürliche Zahl $n \geq 1$:

- Für jedes $i \in \{1, \ldots, n\}$:
  - Simuliere $\mathcal{M}$ auf Eingabe w_i für n Schritte.
  - Falls $\mathcal{M}$ bei dieser Simulation terminiert und w_i akzeptiert, dann gib w_i aus.

Diese Art und Weise der Simulation kann folgendermaßen veranschaulicht werden:

$n \setminus \Sigma^*$	w_1	w_2	w_3	w_4	$\ldots$
1	?				$\ldots$
2					$\ldots$
3					$\ldots$
4					$\ldots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

L = {

Von der TM zum Aufzähler

Für jede natürliche Zahl $n \geq 1$:

- Für jedes $i \in \{1, \ldots, n\}$:
  - Simuliere $\mathcal{M}$ auf Eingabe w_i für n Schritte.
  - Falls $\mathcal{M}$ bei dieser Simulation terminiert und w_i akzeptiert, dann gib w_i aus.

Diese Art und Weise der Simulation kann folgendermaßen veranschaulicht werden:

$n \setminus \Sigma^*$	w_1	w_2	w_3	w_4	$\ldots$
1	?				$\ldots$
2	?				$\ldots$
3					$\ldots$
4					$\ldots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

L = {

Von der TM zum Aufzähler

Für jede natürliche Zahl $n \geq 1$:

- Für jedes $i \in \{1, \ldots, n\}$:
  - Simuliere $\mathcal{M}$ auf Eingabe w_i für n Schritte.
  - Falls $\mathcal{M}$ bei dieser Simulation terminiert und w_i akzeptiert, dann gib w_i aus.

Diese Art und Weise der Simulation kann folgendermaßen veranschaulicht werden:

$n \setminus \Sigma^*$	w_1	w_2	w_3	w_4	$\ldots$
1	?				$\ldots$
2	?	?			$\ldots$
3					$\ldots$
4					$\ldots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

L = {

Von der TM zum Aufzähler

Für jede natürliche Zahl $n \geq 1$:

- Für jedes $i \in \{1, \ldots, n\}$:
  - Simuliere $\mathcal{M}$ auf Eingabe w_i für n Schritte.
  - Falls $\mathcal{M}$ bei dieser Simulation terminiert und w_i akzeptiert, dann gib w_i aus.

Diese Art und Weise der Simulation kann folgendermaßen veranschaulicht werden:

$n \setminus \Sigma^*$	w_1	w_2	w_3	w_4	$\ldots$
1	?				$\ldots$
2	?	?			$\ldots$
3	?				$\ldots$
4					$\ldots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

L = {

Von der TM zum Aufzähler

Für jede natürliche Zahl $n \geq 1$:

- Für jedes $i \in \{1, \ldots, n\}$:
  - Simuliere $\mathcal{M}$ auf Eingabe w_i für n Schritte.
  - Falls $\mathcal{M}$ bei dieser Simulation terminiert und w_i akzeptiert, dann gib w_i aus.

Diese Art und Weise der Simulation kann folgendermaßen veranschaulicht werden:

$n \setminus \Sigma^*$	w_1	w_2	w_3	w_4	$\ldots$
1	?				$\ldots$
2	?	?			$\ldots$
3	?	?			$\ldots$
4					$\ldots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

L = {

Von der TM zum Aufzähler

Für jede natürliche Zahl $n \geq 1$:

- Für jedes $i \in \{1, \ldots, n\}$:
  - Simuliere $\mathcal{M}$ auf Eingabe w_i für n Schritte.
  - Falls $\mathcal{M}$ bei dieser Simulation terminiert und w_i akzeptiert, dann gib w_i aus.

Diese Art und Weise der Simulation kann folgendermaßen veranschaulicht werden:

$n \setminus \Sigma^*$	w_1	w_2	w_3	w_4	$\ldots$
1	?				$\ldots$
2	?	?			$\ldots$
3	?	?	$\in$		$\ldots$
4					$\ldots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

$\mathbf{L} = \{ w_3$

Von der TM zum Aufzähler

Für jede natürliche Zahl $n \geq 1$:

- Für jedes $i \in \{1, \ldots, n\}$:
  - Simuliere $\mathcal{M}$ auf Eingabe w_i für n Schritte.
  - Falls $\mathcal{M}$ bei dieser Simulation terminiert und w_i akzeptiert, dann gib w_i aus.

Diese Art und Weise der Simulation kann folgendermaßen veranschaulicht werden:

$n \setminus \Sigma^*$	w_1	w_2	w_3	w_4	$\ldots$
1	?				$\ldots$
2	?	?			$\ldots$
3	?	?	$\in$		$\ldots$
4	$\notin$				$\ldots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

$\mathbf{L} = \{ w_3$

Von der TM zum Aufzähler

Für jede natürliche Zahl $n \geq 1$:

- Für jedes $i \in \{1, \ldots, n\}$:
  - Simuliere $\mathcal{M}$ auf Eingabe w_i für n Schritte.
  - Falls $\mathcal{M}$ bei dieser Simulation terminiert und w_i akzeptiert, dann gib w_i aus.

Diese Art und Weise der Simulation kann folgendermaßen veranschaulicht werden:

$n \setminus \Sigma^*$	w_1	w_2	w_3	w_4	$\ldots$
1	?				$\ldots$
2	?	?			$\ldots$
3	?	?	$\in$		$\ldots$
4	$\notin$	?			$\ldots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

$\mathbf{L} = \{ w_3$

Von der TM zum Aufzähler

Für jede natürliche Zahl $n \geq 1$:

- Für jedes $i \in \{1, \ldots, n\}$:
  - Simuliere $\mathcal{M}$ auf Eingabe w_i für n Schritte.
  - Falls $\mathcal{M}$ bei dieser Simulation terminiert und w_i akzeptiert, dann gib w_i aus.

Diese Art und Weise der Simulation kann folgendermaßen veranschaulicht werden:

$n \setminus \Sigma^*$	w_1	w_2	w_3	w_4	$\ldots$
1	?				$\ldots$
2	?	?			$\ldots$
3	?	?	$\in$		$\ldots$
4	$\notin$	?	$\in$		$\ldots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

$\mathbf{L} = \{ w_3$

Von der TM zum Aufzähler

Für jede natürliche Zahl $n \geq 1$:

- Für jedes $i \in \{1, \ldots, n\}$:
  - Simuliere $\mathcal{M}$ auf Eingabe w_i für n Schritte.
  - Falls $\mathcal{M}$ bei dieser Simulation terminiert und w_i akzeptiert, dann gib w_i aus.

Diese Art und Weise der Simulation kann folgendermaßen veranschaulicht werden:

$n \setminus \Sigma^*$	w_1	w_2	w_3	w_4	$\ldots$
1	?				$\ldots$
2	?	?			$\ldots$
3	?	?	$\in$		$\ldots$
4	$\notin$	?	$\in$	$\in$	$\ldots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

$\mathbf{L} = \{ w_3, w_4$

Von der TM zum Aufzähler

Für jede natürliche Zahl $n \geq 1$:

- Für jedes $i \in \{1, \ldots, n\}$:
  - Simuliere $\mathcal{M}$ auf Eingabe w_i für n Schritte.
  - Falls $\mathcal{M}$ bei dieser Simulation terminiert und w_i akzeptiert, dann gib w_i aus.

Diese Art und Weise der Simulation kann folgendermaßen veranschaulicht werden:

$n \setminus \Sigma^*$	w_1	w_2	w_3	w_4	$\ldots$
1	?				$\ldots$
2	?	?			$\ldots$
3	?	?	$\in$		$\ldots$
4	$\notin$	?	$\in$	$\in$	$\ldots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

$\mathbf{L} = \{ w_3, w_4, \ldots \}$

Berechnungen jenseits von Σ^*

Berechnungen jenseits von Σ^*

Wir können den Berechnungsbegriff leicht auf beliebige Objekte ausdehnen, die als Wörter kodiert werden können.

Wichtige Fälle:

- Natürliche Zahlen $\mathbb{N}$ können z.B. binär als Wörter über $\{\mathtt{0}, \mathtt{1}\}$ kodiert werden.
- Tupel (Listen) von Wörtern (oder natürlichen Zahlen, ...), können kodiert werden, indem man zum Eingabealphabet ein zusätzliches Trennzeichen # hinzufügt.

Berechnungen jenseits von Σ^*

Wir können den Berechnungsbegriff leicht auf beliebige Objekte ausdehnen, die als Wörter kodiert werden können.

Wichtige Fälle:

- Natürliche Zahlen $\mathbb{N}$ können z.B. binär als Wörter über $\{\mathtt{0}, \mathtt{1}\}$ kodiert werden.
- Tupel (Listen) von Wörtern (oder natürlichen Zahlen, ...), können kodiert werden, indem man zum Eingabealphabet ein zusätzliches Trennzeichen # hinzufügt.

Beispiel: Mit Hilfe dieser Kodierungen können wir z.B. von berechenbaren Funktionen $\mathbb{N} \to \mathbb{N}$ oder von semi-entscheidbaren Teilmengen von $\mathbb{N} \times \mathbb{N}$ sprechen.

Anmerkung: Oft gibt es viele denkbare Kodierungen eines Objektes als Wort. Vorerst sollen uns die Details nicht interessieren, solange klar ist, dass eine TM die Kodierung entschlüsseln kann.

Zusammenhang von Sprache und Funktion

Berechenbarkeit von Funktionen und Sprachen sind eng verwandt.

Sei $\mathbf{L} \subseteq \Sigma^*$ eine Sprache. Die charakteristische Funktion von $\mathbf{L}$ ist $f_{\mathbf{L}} : \Sigma^* \to \Sigma^*$ mit

$$f_{\mathbf{L}}(w) = \begin{cases} 1 & \text{falls } w \in \mathbf{L}, \\ 0 & \text{falls } w \notin \mathbf{L}. \end{cases}$$

(O.B.d.A. sei $\{0, 1\} \subseteq \Sigma$.)

Zusammenhang von Sprache und Funktion

Berechenbarkeit von Funktionen und Sprachen sind eng verwandt.

Sei $\mathbf{L} \subseteq \Sigma^*$ eine Sprache. Die charakteristische Funktion von $\mathbf{L}$ ist $f_{\mathbf{L}} : \Sigma^* \to \Sigma^*$ mit

$$f_{\mathbf{L}}(w) = \begin{cases} 1 & \text{falls } w \in \mathbf{L}, \\ 0 & \text{falls } w \notin \mathbf{L}. \end{cases}$$

(O.B.d.A. sei $\{0, 1\} \subseteq \Sigma$.)

Satz: Eine Sprache $\mathbf{L}$ ist genau dann entscheidbar, wenn ihre charakteristische Funktion berechenbar ist.

Zusammenhang von Sprache und Funktion

Berechenbarkeit von Funktionen und Sprachen sind eng verwandt.

Sei $\mathbf{L} \subseteq \Sigma^*$ eine Sprache. Die charakteristische Funktion von $\mathbf{L}$ ist $f_{\mathbf{L}} : \Sigma^* \to \Sigma^*$ mit

$$f_{\mathbf{L}}(w) = \begin{cases} 1 & \text{falls } w \in \mathbf{L}, \\ 0 & \text{falls } w \notin \mathbf{L}. \end{cases} \qquad \text{(O.B.d.A. sei } \{0, 1\} \subseteq \Sigma.\text{)}$$

Satz: Eine Sprache $\mathbf{L}$ ist genau dann entscheidbar, wenn ihre charakteristische Funktion berechenbar ist.

Beweisskizze: „⇒“ Ein Entscheider für $\mathbf{L}$ kann in eine TM für $f_{\mathbf{L}}$ umgebaut werden. Dazu verwendet man „Subroutinen“, die den Bandinhalt löschen und mit einem einzelnen Zeichen 1 oder 0 ersetzten. Diese Routinen werden aufgerufen, wenn der ursprüngliche Entscheider halten würde: die 1-Routine beim Halten in einem Endzustand, die 0-Routine andernfalls.

Eventuell muss man den Entscheider außerdem so modifizieren, dass das Zeichen ␣ nur am Ende des verwendeten Bandinhaltes vorkommen kann (sonst wird das Löschen des gesamten Bandes problematisch).

„⇐“ Eine TM, die $f_{\mathbf{L}}$ berechnet, kann in einen Entscheider für $\mathbf{L}$ umgebaut werden. Die Idee ist wie zuvor, aber die Subroutinen prüfen jetzt, ob das Band 1 oder 0 enthält und wechseln entsprechend in einen akzeptierenden oder nicht-akzeptierenden Zustand. □

Zusammenhang von Funktion und Sprache (1)

Für die Umkehrung stellen wir Funktionen als Mengen dar:

Satz:
Eine partielle Funktion $f : \Sigma^* \to \Sigma^*$ ist genau dann berechenbar, wenn die Sprache

$$\text{Graph}_f = \{\langle w, f(w)\rangle \mid w \in \Sigma^*,\ f(w) \text{ definiert}\}$$

semi-entscheidbar ist. Ist f total, dann ist Graph_f sogar entscheidbar.

Zusammenhang von Funktion und Sprache (1)

Für die Umkehrung stellen wir Funktionen als Mengen dar:

Satz:
Eine partielle Funktion $f : \Sigma^* \to \Sigma^*$ ist genau dann berechenbar, wenn die Sprache

$$\text{Graph}_f = \{\langle w, f(w)\rangle \mid w \in \Sigma^*,\ f(w) \text{ definiert}\}$$

semi-entscheidbar ist. Ist f total, dann ist Graph_f sogar entscheidbar.

Beweis: „⇒“ Sei $f : \Sigma^* \to \Sigma^*$ berechenbar. Dann kann man Graph_f wie folgt erkennen:

- Für eine Eingabe $\langle w, v\rangle$, berechne $f(w)$ (als Subroutine).
- Wenn die Berechnung von $f(w)$ terminiert, dann akzeptiere falls $f(w) = v$.
- Wenn die Berechnung von $f(w)$ nicht terminiert, dann wird auch die Erkennung von Graph_f nicht halten. Dieses Verhalten ist korrekt, da in diesem Fall $f(w)$ undefiniert ist und $\langle w, v\rangle \notin \text{Graph}_f$.

Zusammenhang von Funktion und Sprache (1)

Für die Umkehrung stellen wir Funktionen als Mengen dar:

Satz:
Eine partielle Funktion $f : \Sigma^* \to \Sigma^*$ ist genau dann berechenbar, wenn die Sprache

$$\text{Graph}_f = \{\langle w, f(w)\rangle \mid w \in \Sigma^*,\ f(w) \text{ definiert}\}$$

semi-entscheidbar ist. Ist f total, dann ist Graph_f sogar entscheidbar.

Beweis: „⇒“ Sei $f : \Sigma^* \to \Sigma^*$ berechenbar. Dann kann man Graph_f wie folgt erkennen:

- Für eine Eingabe $\langle w, v\rangle$, berechne $f(w)$ (als Subroutine).
- Wenn die Berechnung von $f(w)$ terminiert, dann akzeptiere falls $f(w) = v$.
- Wenn die Berechnung von $f(w)$ nicht terminiert, dann wird auch die Erkennung von Graph_f nicht halten. Dieses Verhalten ist korrekt, da in diesem Fall $f(w)$ undefiniert ist und $\langle w, v\rangle \notin \text{Graph}_f$.

Falls f total ist, dann hält auch die Erkennung von Graph_f.

Zusammenhang von Funktion und Sprache (2)

Für die Umkehrung stellen wir Funktionen als Mengen dar:

Satz:
Eine partielle Funktion $f : \Sigma^* \to \Sigma^*$ ist genau dann berechenbar, wenn die Sprache

$$\text{Graph}_f = \{\langle w, f(w)\rangle \mid w \in \Sigma^*,\ f(w) \text{ definiert}\}$$

semi-entscheidbar ist. Ist f total, dann ist Graph_f sogar entscheidbar.

Zusammenhang von Funktion und Sprache (2)

Für die Umkehrung stellen wir Funktionen als Mengen dar:

Satz:
Eine partielle Funktion $f : \Sigma^* \to \Sigma^*$ ist genau dann berechenbar, wenn die Sprache

$$\text{Graph}_f = \{\langle w, f(w)\rangle \mid w \in \Sigma^*,\ f(w) \text{ definiert}\}$$

semi-entscheidbar ist. Ist f total, dann ist Graph_f sogar entscheidbar.

Beweis: „$\Leftarrow$“ Sei Graph_f semi-entscheidbar. Dann kann man f wie folgt berechnen:

- Wir konstruieren einen Aufzähler für Graph_f wie zuvor gezeigt.
- Der Aufzähler wird auf einem eigenen Band simuliert.
- Immer wenn der Aufzähler ein Paar $\langle w, f(w)\rangle$ ausgibt, vergleichen wir w mit dem Inhalt des Eingabebandes.
- Wenn die Eingabe mit w übereinstimmt, dann wird $f(w)$ als Ergebnis verwendet und die TM hält; andernfalls wird die Simulation der Aufzählung fortgesetzt. □

Anmerkung: Wir haben nicht formal definiert, was bei einer Mehrband-TM die Ausgabe ist, aber das kann man leicht tun (oder die Mehrband-TM auf einem Band simulieren).

Unentscheidbare Probleme

Es gibt unentscheidbare Probleme

Satz: Es gibt Sprachen und Funktionen, die nicht berechenbar sind.

Es gibt unentscheidbare Probleme

Satz: Es gibt Sprachen und Funktionen, die nicht berechenbar sind.

Dies kann man wie folgt zeigen:

- Die Menge der Turingmaschinen ist abzählbar.
- Die Menge der Sprachen über jedem Alphabet ist überabzählbar.
- Also muss es Sprachen geben, die durch keine TM entschieden werden.

Es gibt unentscheidbare Probleme

Satz: Es gibt Sprachen und Funktionen, die nicht berechenbar sind.

Dies kann man wie folgt zeigen:

- Die Menge der Turingmaschinen ist abzählbar.
- Die Menge der Sprachen über jedem Alphabet ist überabzählbar.
- Also muss es Sprachen geben, die durch keine TM entschieden werden.

Das Argument funktioniert analog mit (partiellen) Funktionen, deren Menge ebenfalls überabzählbar groß ist.

Das Argument zeigt zudem auch, dass die meisten Sprachen nicht einmal semi-entscheidbar sind (Kontrollfrage: Warum?).

Die Menge der TMs ist abzählbar

Die Menge der TMs ist abzählbar

Dies folgt in zwei Schritten:

- Jede TM kann leicht durch ein endliches Wort kodiert werden (z.B. binär).
- Es gibt abzählbar viele Wörter. Man kann sie zum Beispiel wie folgt abzählen:
  - Beginne mit dem leeren Wort ϵ.
  - Reihe dann alle Wörter der Länge 1 auf
  - Reihe dann alle Wörter der Länge 2 auf
  - Reihe dann alle Wörter der Länge 3 auf
  - ...

  (Vgl. dazu auch Formale Systeme, Vorlesung 2)

Die Menge der Sprachen ist überabzählbar (1)

Die Menge der Sprachen ist überabzählbar (1)

Dies folgt aus einer Konstruktion von Georg Cantor:

- Beweis durch Widerspruch: Wir nehmen an, dass die Menge aller Sprachen abzählbar ist.
- Sei $\mathbf{L_1}, \mathbf{L_2}, \mathbf{L_3}, \ldots$ eine entsprechende Aufzählung aller Sprachen.
- Wir reihen außerdem alle Wörter in Σ^* auf: $w_1, w_2, w_3, \ldots$.

Die Menge der Sprachen ist überabzählbar (2)

- Man kann sich die Relation $\in$ zwischen Wörtern und Sprachen jetzt als unendliche Tabelle vorstellen:

	w_1	w_2	w_3	w_4	$\dots$
$\mathbf{L_1}$	$\times$	$-$	$-$	$\times$	$\dots$
$\mathbf{L_2}$	$-$	$\times$	$-$	$\times$	$\dots$
$\mathbf{L_3}$	$-$	$\times$	$-$	$-$	$\dots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Die Menge der Sprachen ist überabzählbar (2)

- Man kann sich die Relation $\in$ zwischen Wörtern und Sprachen jetzt als unendliche Tabelle vorstellen:

	w_1	w_2	w_3	w_4	$\dots$
$\mathbf{L_1}$	$\times$	$-$	$-$	$\times$	$\dots$
$\mathbf{L_2}$	$-$	$\times$	$-$	$\times$	$\dots$
$\mathbf{L_3}$	$-$	$\times$	$-$	$-$	$\dots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$
$\mathbf{L_d}$	$-$	$-$	$\times$	$\dots$	

- Wir konstruieren eine Sprache $\mathbf{L_d}$ durch Diagonalisierung.
  Formal: Wir setzen $w_i \in \mathbf{L_d}$ genau dann, wenn $w_i \notin \mathbf{L_i}$.

Die Menge der Sprachen ist überabzählbar (2)

- Man kann sich die Relation $\in$ zwischen Wörtern und Sprachen jetzt als unendliche Tabelle vorstellen:

	w_1	w_2	w_3	w_4	$\dots$
$\mathbf{L_1}$	$\times$	$-$	$-$	$\times$	$\dots$
$\mathbf{L_2}$	$-$	$\times$	$-$	$\times$	$\dots$
$\mathbf{L_3}$	$-$	$\times$	$-$	$-$	$\dots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$
$\mathbf{L_d}$	$-$	$-$	$\times$	$\dots$	

- Wir konstruieren eine Sprache $\mathbf{L_d}$ durch Diagonalisierung.
  Formal: Wir setzen $w_i \in \mathbf{L_d}$ genau dann, wenn $w_i \notin \mathbf{L_i}$.

Dann kommt $\mathbf{L_d}$ in der Tabelle nicht vor. Widerspruch.

Nichtwissen ≠ Unentscheidbarkeit (1)

Wie finden wir konkrete unentscheidbare Probleme?

Nichtwissen ≠ Unentscheidbarkeit (1)

Wie finden wir konkrete unentscheidbare Probleme?

Es reicht nicht aus, dass wir nicht wissen, wie ein Problem algorithmisch gelöst werden kann!

Beispiel: Sei $\mathbf{L}_\pi$ die Menge aller endlichen Ziffernfolgen, die in der Dezimaldarstellung von π vorkommen. Zum Beispiel gilt $14159265 \in \mathbf{L}_\pi$ und $41 \in \mathbf{L}_\pi$.

Nichtwissen ≠ Unentscheidbarkeit (1)

Wie finden wir konkrete unentscheidbare Probleme?

Es reicht nicht aus, dass wir nicht wissen, wie ein Problem algorithmisch gelöst werden kann!

Beispiel: Sei $\mathbf{L}_\pi$ die Menge aller endlichen Ziffernfolgen, die in der Dezimaldarstellung von π vorkommen. Zum Beispiel gilt $14159265 \in \mathbf{L}_\pi$ und $41 \in \mathbf{L}_\pi$.

Wir wissen nicht, ob man die Sprache $\mathbf{L}_\pi$ entscheiden kann, aber sie könnte dennoch entscheidbar sein (z.B. wenn jede endliche Ziffernfolge irgendwo in π vorkommt, was aber bisher nicht bekannt ist).

Nichtwissen ≠ Unentscheidbarkeit (2)

Es gibt sogar Fälle, in denen wir sicher sind, dass ein Problem entscheidbar ist, aber trotzdem nicht wissen, wie man es löst.

Nichtwissen ≠ Unentscheidbarkeit (2)

Es gibt sogar Fälle, in denen wir sicher sind, dass ein Problem entscheidbar ist, aber trotzdem nicht wissen, wie man es löst.

Beispiel: Sei $\mathbf{L}_{\pi 7}$ die Menge aller Ziffernfolgen der Form 7^n, die in der Dezimaldarstellung von π vorkommen.

Nichtwissen ≠ Unentscheidbarkeit (2)

Es gibt sogar Fälle, in denen wir sicher sind, dass ein Problem entscheidbar ist, aber trotzdem nicht wissen, wie man es löst.

Beispiel: Sei $\mathbf{L}_{\pi 7}$ die Menge aller Ziffernfolgen der Form 7^n, die in der Dezimaldarstellung von π vorkommen.

$\mathbf{L}_{\pi 7}$ ist entscheidbar:

- Möglichkeit 1: π enthält beliebig lange Ketten der Ziffer 7. Dann wird $\mathbf{L}_{\pi 7}$ durch eine TM entschieden, die alle Wörter der Form 7^n akzeptiert.
- Möglichkeit 2: π enthält Ketten der Ziffer 7 nur bis zu einer maximalen Länge ℓ. Dann wird $\mathbf{L}_{\pi 7}$ durch eine TM entschieden, die alle Wörter der Form 7^n mit $n \leq \ell$ akzeptiert.

Für jeden denkbaren Fall gibt es einen Algorithmus – wir wissen nur nicht, welcher davon korrekt ist.

Quiz: Berechenbare Funktion

Quiz: Wir definieren die Funktion $f_{\mathbf{R}} : \{\mathtt{0}, \mathtt{1}\}^* \to \{\mathtt{0}, \mathtt{1}\}^*$ wie folgt: ...

Ein erstes unentscheidbares Problem (1)

Frage: Falls eine TM anhält, wie lange kann das im schlimmsten Fall dauern?

Ein erstes unentscheidbares Problem (1)

Frage: Falls eine TM anhält, wie lange kann das im schlimmsten Fall dauern?

Antwort: Beliebig lange, weil:

(a) die Eingabe beliebig groß sein kann;

(b) die TM beliebig groß sein kann.

Ein erstes unentscheidbares Problem (2)

Frage: Falls eine TM mit n Zuständen und einem zwei-elementigen Arbeitsalphabet $\Gamma = \{\mathtt{x}, \sqcup\}$ auf einem leeren Band anhält, wie lange kann das im schlimmsten Fall dauern?

Ein erstes unentscheidbares Problem (2)

Frage: Falls eine TM mit n Zuständen und einem zwei-elementigen Arbeitsalphabet $\Gamma = \{\mathtt{x}, \sqcup\}$ auf einem leeren Band anhält, wie lange kann das im schlimmsten Fall dauern?

Antwort: Das kommt auf n an ...

Ein erstes unentscheidbares Problem (2)

Frage: Falls eine TM mit n Zuständen und einem zwei-elementigen Arbeitsalphabet $\Gamma = \{\mathtt{x}, \sqcup\}$ auf einem leeren Band anhält, wie lange kann das im schlimmsten Fall dauern?

Antwort: Das kommt auf n an ...

Wir definieren $S(n)$ als die maximale Zahl an Schritten, die eine DTM mit n Zuständen und dem Arbeitsalphabet $\Gamma = \{\mathtt{x}, \sqcup\}$ auf dem leeren Band ausführt, bevor sie schließlich hält.

Beobachtung: S ist wohldefiniert.

- Die Zahl der TMs mit maximal n Zuständen ist endlich.
- Unter den relevanten n-Zustand-TMs gibt es eine maximale Anzahl an Schritten bis zum Halten (nicht haltende TMs werden ignoriert).

Fleißige Biber

Tibor Radó, BB-Erfinder

Eine leichte Abwandlung des Schrittzählers ist das Busy-Beaver-Problem:

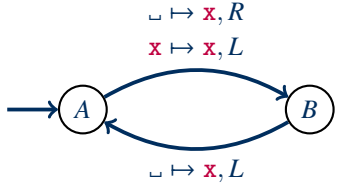
Die Busy-Beaver-Funktion $\mathbf{\Sigma} : \mathbb{N} \to \mathbb{N}$ ist eine totale Funktion, wobei $\mathbf{\Sigma}(n)$ die maximale Zahl an $\mathbf{x}$ ist, die eine DTM mit höchstens n Zuständen und dem Arbeitsalphabet $\Gamma = \{\mathbf{x}, \sqcup\}$ beginnend mit dem leeren Band schreiben kann, bevor sie schließlich hält.

Anmerkung: Der genaue Wert von $\mathbf{\Sigma}(n)$ hängt von Details der TM-Definition ab.

Üblich ist hier insbesondere die Verwendung eines zweiseitig unendlichen Bands, das man bei Bedarf nach links und rechts erweitern kann.

Beispiel

Die Busy-Beaver-Zahl $\Sigma(2)$ ist 4, wenn man ein beidseitig unendliches Band annimmt.
Die folgende TM realisiert dieses Verhalten:



Wir erhalten: $A\text{␣} \vdash \mathtt{x}B\text{␣} \vdash A\mathtt{xx} \vdash B\text{␣}\mathtt{xx} \vdash A\text{␣}\mathtt{xxx} \vdash \mathtt{x}B\mathtt{xxx}$

Busy-Beaver berechnen?

Satz: Die Busy-Beaver-Funktion ist nicht berechenbar.

[1] Bei der Hintereinanderausführung kann man End- und Anfangszustand jeweils verschmelzen.

Busy-Beaver berechnen?

Satz: Die Busy-Beaver-Funktion ist nicht berechenbar.

Beweisskizze: Nehmen wir an, Σ wäre berechenbar.

[1] Bei der Hintereinanderausführung kann man End- und Anfangszustand jeweils verschmelzen.

Busy-Beaver berechnen?

Satz: Die Busy-Beaver-Funktion ist nicht berechenbar.

Beweisskizze: Nehmen wir an, Σ wäre berechenbar.

- Dann kann man eine TM $\mathcal{M}_\Sigma$ erzeugen, die mit dem Alphabet $\{\mathtt{x}, \sqcup\}$ arbeitet und die Funktion $\mathtt{x}^n \mapsto \mathtt{x}^{\Sigma(n)}$ berechnet.

[1] Bei der Hintereinanderausführung kann man End- und Anfangszustand jeweils verschmelzen.

Busy-Beaver berechnen?

Satz: Die Busy-Beaver-Funktion ist nicht berechenbar.

Beweisskizze: Nehmen wir an, Σ wäre berechenbar.

- Dann kann man eine TM $\mathcal{M}_\Sigma$ erzeugen, die mit dem Alphabet $\{\mathtt{x}, \sqcup\}$ arbeitet und die Funktion $\mathtt{x}^n \mapsto \mathtt{x}^{\Sigma(n)}$ berechnet.
- Sei $\mathcal{M}_{+1}$ eine TM, welche die Funktion $\mathtt{x}^n \mapsto \mathtt{x}^{n+1}$ berechnet.

[1] Bei der Hintereinanderausführung kann man End- und Anfangszustand jeweils verschmelzen.

Busy-Beaver berechnen?

Satz: Die Busy-Beaver-Funktion ist nicht berechenbar.

Beweisskizze: Nehmen wir an, Σ wäre berechenbar.

- Dann kann man eine TM $\mathcal{M}_\Sigma$ erzeugen, die mit dem Alphabet $\{\mathtt{x}, \sqcup\}$ arbeitet und die Funktion $\mathtt{x}^n \mapsto \mathtt{x}^{\Sigma(n)}$ berechnet.
- Sei $\mathcal{M}_{+1}$ eine TM, welche die Funktion $\mathtt{x}^n \mapsto \mathtt{x}^{n+1}$ berechnet.
- Sei $\mathcal{M}_{*2}$ eine TM, welche die Funktion $\mathtt{x}^n \mapsto \mathtt{x}^{2n}$ berechnet.

[1] Bei der Hintereinanderausführung kann man End- und Anfangszustand jeweils verschmelzen.

Busy-Beaver berechnen?

Satz: Die Busy-Beaver-Funktion ist nicht berechenbar.

Beweisskizze: Nehmen wir an, Σ wäre berechenbar.

- Dann kann man eine TM $\mathcal{M}_\Sigma$ erzeugen, die mit dem Alphabet $\{\mathtt{x}, \sqcup\}$ arbeitet und die Funktion $\mathtt{x}^n \mapsto \mathtt{x}^{\Sigma(n)}$ berechnet.
- Sei $\mathcal{M}_{+1}$ eine TM, welche die Funktion $\mathtt{x}^n \mapsto \mathtt{x}^{n+1}$ berechnet.
- Sei $\mathcal{M}_{*2}$ eine TM, welche die Funktion $\mathtt{x}^n \mapsto \mathtt{x}^{2n}$ berechnet.
- Sei k die Gesamtzahl der Zustände in $\mathcal{M}_\Sigma$, $\mathcal{M}_{+1}$ und $\mathcal{M}_{*2}$. Es gibt eine TM $\mathcal{I}_k$ mit $k+1$ Zuständen, die das Wort $\mathtt{x}^k$ auf das leere Band schreibt.

[1] Bei der Hintereinanderausführung kann man End- und Anfangszustand jeweils verschmelzen.

Busy-Beaver berechnen?

Satz: Die Busy-Beaver-Funktion ist nicht berechenbar.

Beweisskizze: Nehmen wir an, Σ wäre berechenbar.

- Dann kann man eine TM $\mathcal{M}_\Sigma$ erzeugen, die mit dem Alphabet $\{\mathtt{x}, \sqcup\}$ arbeitet und die Funktion $\mathtt{x}^n \mapsto \mathtt{x}^{\Sigma(n)}$ berechnet.
- Sei $\mathcal{M}_{+1}$ eine TM, welche die Funktion $\mathtt{x}^n \mapsto \mathtt{x}^{n+1}$ berechnet.
- Sei $\mathcal{M}_{*2}$ eine TM, welche die Funktion $\mathtt{x}^n \mapsto \mathtt{x}^{2n}$ berechnet.
- Sei k die Gesamtzahl der Zustände in $\mathcal{M}_\Sigma$, $\mathcal{M}_{+1}$ und $\mathcal{M}_{*2}$. Es gibt eine TM $\mathcal{I}_k$ mit $k+1$ Zuständen, die das Wort $\mathtt{x}^k$ auf das leere Band schreibt.
- Wenn man nun hintereinander $\mathcal{I}_k$, $\mathcal{M}_{*2}$, $\mathcal{M}_\Sigma$ und $\mathcal{M}_{+1}$ ausführt, dann erhält man eine TM mit $\leq 2k$ Zuständen,[1] die insgesamt $\Sigma(2k)+1$ mal $\mathtt{x}$ schreibt und dann hält.

[1] Bei der Hintereinanderausführung kann man End- und Anfangszustand jeweils verschmelzen.

Busy-Beaver berechnen?

Satz: Die Busy-Beaver-Funktion ist nicht berechenbar.

Beweisskizze: Nehmen wir an, Σ wäre berechenbar.

- Dann kann man eine TM $\mathcal{M}_\Sigma$ erzeugen, die mit dem Alphabet $\{\mathtt{x}, \sqcup\}$ arbeitet und die Funktion $\mathtt{x}^n \mapsto \mathtt{x}^{\Sigma(n)}$ berechnet.
- Sei $\mathcal{M}_{+1}$ eine TM, welche die Funktion $\mathtt{x}^n \mapsto \mathtt{x}^{n+1}$ berechnet.
- Sei $\mathcal{M}_{*2}$ eine TM, welche die Funktion $\mathtt{x}^n \mapsto \mathtt{x}^{2n}$ berechnet.
- Sei k die Gesamtzahl der Zustände in $\mathcal{M}_\Sigma$, $\mathcal{M}_{+1}$ und $\mathcal{M}_{*2}$. Es gibt eine TM $\mathcal{I}_k$ mit $k+1$ Zuständen, die das Wort $\mathtt{x}^k$ auf das leere Band schreibt.
- Wenn man nun hintereinander $\mathcal{I}_k$, $\mathcal{M}_{*2}$, $\mathcal{M}_\Sigma$ und $\mathcal{M}_{+1}$ ausführt, dann erhält man eine TM mit $\leq 2k$ Zuständen,[1] die insgesamt $\Sigma(2k)+1$ mal $\mathtt{x}$ schreibt und dann hält.
- Also ist $\Sigma(2k) \geq \Sigma(2k)+1$ – Widerspruch. □

[1] Bei der Hintereinanderausführung kann man End- und Anfangszustand jeweils verschmelzen.

Bemerkungen zum Beweis

Anmerkung 1: Der Beweis verwendet die interessante Idee, dass man TMs als „Subroutinen“ von anderen TMs verwenden kann. Wir werden das noch an anderer Stelle verwenden.

Anmerkung 2: Der Schritt von einer beliebigen Berechnung einer Funktion $f : \mathbb{N} \to \mathbb{N}$ zu einer TM, die eine Funktion $\mathtt{x}^n \mapsto \mathtt{x}^{f(n)}$ berechnet, ist nicht schwer; man ändert nur die Kodierung der Ein- und Ausgabe von binär auf unär.

Anmerkung 3: Der Schritt von einer beliebigen TM zu einer, die auf dem Alphabet $\{\mathtt{x}, \sqcup\}$ arbeitet, ist etwas kniffliger, aber machbar.

Theorie und Praxis

„Unentscheidbarkeit ist doch eine rein theoretische Eigenschaft! In der Praxis ist es egal, ob wir $\Sigma(n)$ für beliebig große n berechnen können. Die praktisch relevanten Fälle können wir sicher klären.“

Theorie und Praxis

„Unentscheidbarkeit ist doch eine rein theoretische Eigenschaft! In der Praxis ist es egal, ob wir $\Sigma(n)$ für beliebig große n berechnen können. Die praktisch relevanten Fälle können wir sicher klären.“

Nun ja . . . seit den 1960ern ist man noch nicht so weit gekommen:

n:	1	2
$\Sigma(n)$:	1	4

Theorie und Praxis

„Unentscheidbarkeit ist doch eine rein theoretische Eigenschaft! In der Praxis ist es egal, ob wir $\Sigma(n)$ für beliebig große n berechnen können. Die praktisch relevanten Fälle können wir sicher klären.“

Nun ja . . . seit den 1960ern ist man noch nicht so weit gekommen:

n:	1	2	3
$\Sigma(n)$:	1	4	6

Theorie und Praxis

„Unentscheidbarkeit ist doch eine rein theoretische Eigenschaft! In der Praxis ist es egal, ob wir $\Sigma(n)$ für beliebig große n berechnen können. Die praktisch relevanten Fälle können wir sicher klären.“

Nun ja . . . seit den 1960ern ist man noch nicht so weit gekommen:

n:	1	2	3	4
$\Sigma(n)$:	1	4	6	13

Theorie und Praxis

„Unentscheidbarkeit ist doch eine rein theoretische Eigenschaft! In der Praxis ist es egal, ob wir $\Sigma(n)$ für beliebig große n berechnen können. Die praktisch relevanten Fälle können wir sicher klären.“

Nun ja . . . seit den 1960ern ist man noch nicht so weit gekommen:

n:	1	2	3	4	5
$\Sigma(n)$:	1	4	6	13	≥ 4098

Theorie und Praxis

„Unentscheidbarkeit ist doch eine rein theoretische Eigenschaft! In der Praxis ist es egal, ob wir $\Sigma(n)$ für beliebig große n berechnen können. Die praktisch relevanten Fälle können wir sicher klären.“

Nun ja . . . seit den 1960ern ist man noch nicht so weit gekommen:

n:	1	2	3	4	5	6
$\Sigma(n)$:	1	4	6	13	≥ 4098	$\geq 3{,}5 \cdot 10^{18267}$

Theorie und Praxis

„Unentscheidbarkeit ist doch eine rein theoretische Eigenschaft! In der Praxis ist es egal, ob wir $\Sigma(n)$ für beliebig große n berechnen können. Die praktisch relevanten Fälle können wir sicher klären.“

Nun ja . . . seit den 1960ern ist man noch nicht so weit gekommen:

n:	1	2	3	4	5	6	7
$\Sigma(n)$:	1	4	6	13	≥ 4098	$\geq 3{,}5 \cdot 10^{18267}$	riesig

Theorie und Praxis

„Unentscheidbarkeit ist doch eine rein theoretische Eigenschaft! In der Praxis ist es egal, ob wir $\Sigma(n)$ für beliebig große n berechnen können. Die praktisch relevanten Fälle können wir sicher klären.“

Nun ja . . . seit den 1960ern ist man noch nicht so weit gekommen:

n:	1	2	3	4	5	6	7	8
$\Sigma(n)$:	1	4	6	13	≥ 4098	$\geq 3{,}5 \cdot 10^{18267}$	riesig	irrsinnig

Theorie und Praxis

„Unentscheidbarkeit ist doch eine rein theoretische Eigenschaft! In der Praxis ist es egal, ob wir $\Sigma(n)$ für beliebig große n berechnen können. Die praktisch relevanten Fälle können wir sicher klären.“

Nun ja . . . seit den 1960ern ist man noch nicht so weit gekommen:

n:	1	2	3	4	5	6	7	8
$\Sigma(n)$:	1	4	6	13	≥ 4098	$\geq 3{,}5 \cdot 10^{18267}$	riesig	irrsinnig

Für $n = 10$ lässt sich eine untere Schranke der Form $\Sigma(10) > 3^{3^{3^{\cdot^{\cdot^{\cdot^{3}}}}}}$ angeben, wobei der komplette Ausdruck über $7{,}6$ Billionen mal die Zahl 3 enthält.

Zusammenfassung und Ausblick

Turingmaschinen sind auf viele Arten verwendbar (und definierbar).

Funktionen und Mengen können berechenbar sein, aber die meisten sind es nicht.

Semi-entscheidbare Mengen können durch Aufzähler erzeugt werden.

Die Busy-Beaver-Funktion ist nicht berechenbar und wächst irrsinnig schnell.

Was erwartet uns als nächstes?

- Relevantere Probleme
- Reduktionen
- Rechenmodelle, die nicht auf TMs beruhen

Bildrechte

Folie 2: gemeinfrei
Folie 21: Fotografie von 1870, gemeinfrei
Folie 28: Ausschnitt aus einer Fotografie von 1928,
`http://www.bibl.u-szeged.hu/sztegy/photo/778.jpg`, CC-By-SA 3.0