TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik, Institut für Künstliche Intelligenz, Professur für Computational Logic

FORMALE SYSTEME

19. Vorlesung: Nichtdeterminismus und Unentscheidbarkeit

Hannes Straß

TU Dresden, 13. Dezember 2021

Organisatorisches

In der Woche vor Weihnachten

20.–22.12.2021

und der Woche nach dem Jahreswechsel

05.–07.01.2022

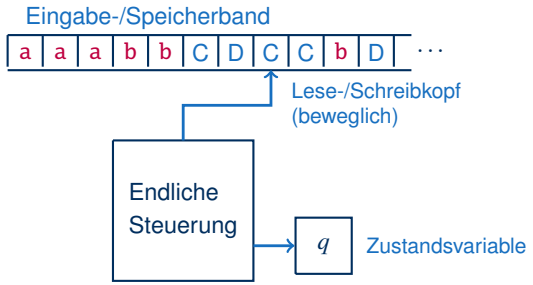
findet keine Präsenzlehre statt.

Ihre Tutor:innen kontaktieren Sie mit Informationen zur digitalen Durchführung der Lehrveranstaltungen.

Rückblick

Alan Turing (5 Jahre alt)

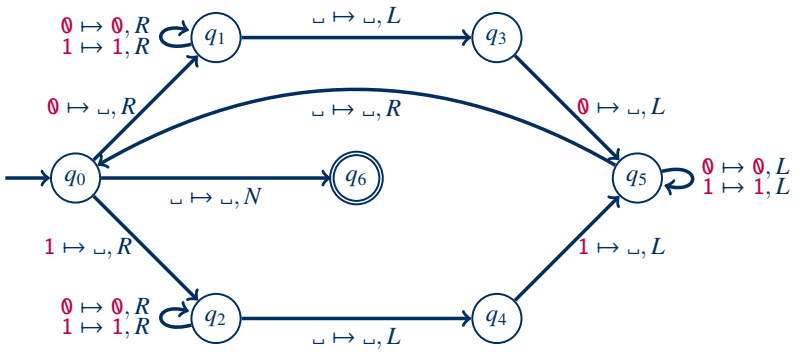
Die Turingmaschine



Eine (deterministische) Turingmaschine (DTM) ist ein Tupel $\mathcal{M} = \langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle$ bestehend aus Zustandsmenge Q, Eingabealphabet Σ, Arbeitsalphabet $\Gamma \supseteq \Sigma \cup \{\sqcup\}$, Startzustand $q_0 \in Q$, Endzuständen $F \subseteq Q$, und einer partiellen Übergangsfunktion

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, N\}.$$

Deterministische Turingmaschine: Beispiel



Beispielkonfigurationsübergangsfolgen:

- $\sqcup q_5\, 11 \vdash q_5\, \sqcup 11 \vdash \sqcup q_0\, 11 \vdash \sqcup\sqcup q_2\, 1 \vdash \sqcup\sqcup 1\, q_2 \sqcup \vdash \sqcup\sqcup q_4 1 \vdash \sqcup q_5 \sqcup \vdash \sqcup\sqcup q_0 \sqcup$
- $q_5\, 11 \vdash q_5\, 11 \vdash \ldots$

Skizzenhafte Beschreibung der Arbeitsweise:

(1) Ist das Arbeitsband leer, akzeptiere.

Die Church-Turing-These

Was ist mit dem intuitiven Begriff der „mechanisch berechenbaren“ Funktion gemeint?

- Gödel/Herbrand (1934): allgemeine rekursive Funktionen
- Church (1936): λ-Kalkül
- Turing (1936): Turingmaschine (ursprünglich „a-machine“)
- Kleene/Rosser/Church/Turing: Die drei Ansätze beschreiben die gleiche Klasse von Funktionen!

Church-Turing-These: Eine Funktion ist genau dann im intuitiven Sinne berechenbar, wenn es eine Turingmaschine gibt, die für jede mögliche Eingabe den Wert der Funktion auf das Band schreibt und anschließend hält.

- **Lesart 1:** Vorschlag einer mathematischen Definition der intuitiven Idee von Berechenbarkeit.
- **Lesart 2:** „Naturgesetz“ über die Möglichkeiten und Grenzen des Rechnens an sich.

Berechnung vs. Worterkennung

Wir haben TMs als Automaten zur Worterkennung definiert:

- Die Eingabe ist am Anfang auf dem Band gegeben.
- Die TM kann (a) anhalten und die Eingabe akzeptieren, (b) anhalten und die Eingabe ablehnen, oder (c) nicht anhalten.

Wie kann man TMs als allgemeines Rechenmodell verstehen?

(1) Berechnungsfragen können als Wortproblem kodiert werden.
   - Eingabewörter als Kodierung beliebiger Eingaben (z.B. als Binärdatei)
   - Berechnung einer Booleschen Funktion $\rightsquigarrow$ Entscheidungsproblem

(2) Alternative Definition: Der Inhalt des Bandes beim Halten der TM wird als Ausgabe interpretiert (keine Endzustände nötig).
Dann kodieren TMs partielle Funktionen $\Sigma^* \to \Gamma^*$ (partiell, da die TM nicht immer anhalten muss).

Computer sind kompliziert

Das Modell der Turingmaschine ist einfach,
das Verhalten von Turingmaschinen ist es nicht!

Betrachten wir einen ganz einfachen Sonderfall:

- TMs mit nur fünf Zuständen
- und nur zwei Zeichen im Bandalphabet
- bei Eingabe ϵ (leeres Wort).

Bereits diese „Spielzeugbeispiele“ werden auch 85 Jahre nach Einführung der TM noch nicht völlig verstanden:

- Es gibt TMs dieser Art, die bei Eingabe ϵ erst nach über 47 Millionen Übergängen[1] anhalten.
- Es gibt TMs dieser Art, von denen wir bis heute nicht wissen, ob sie bei Eingabe ϵ jemals anhalten.

[1] Für ein TM-Modell, das am linken und am rechten Bandende Speicher allokieren kann (beidseitig unendliches Band); siehe „busy beaver“.

Varianten von Turingmaschinen

Es gibt sehr viele alternative Definitionen von Turingmaschinen:

- Alternative Akzeptanzbedinungen (Ausgabe der Antwort aufs Band, totale Übergangsfunktion + explizite Stoppzustände für Akzeptanz & Ablehnung, ...)
- Alternative Bewegungsregeln (keine N-Übergänge, anderes Verhalten am Rand des Bandes, ...)
- TMs mit beidseitig unendlichem Band
- TMs mit mehreren Bändern
- nichtdeterministische Turingmaschinen
- TMs mit wahlfreiem Speicherzugriff (RAM)
- ...

All diese Varianten können die selben Funktionen berechnen – wenn auch zum Teil mit unterschiedlichem Aufwand.

$\rightsquigarrow$ Untermauerung der Church-Turing-These

TMs mit mehreren Bändern

Die Mehrband-Turingmaschine

- ... verwendet eine vorher festgelegte Zahl $k \geq 2$ von (einseitig unendlichen) Speicherbändern;
- ... erweitert die Übergangsfunktion entsprechend:

$$\delta : Q \times \Gamma^k \to Q \times (\Gamma \times \{L, R, N\})^k$$

- ... erhält die Eingabe auf dem ersten Band, während die anderen anfänglich leer sind.

Wichtig: Die Mehrband-TM hat einen unabhängigen Lese-/Schreibkopf für jedes Band, damit sind unterschiedliche Bewegungen/Positionen möglich.

Beispiel: Ein deterministischer PDA kann leicht durch eine 2-Band-TM simuliert werden. Dabei wird vom ersten Band nur gelesen, während auf dem zweiten Band der aktuelle Kellerinhalt gespeichert wird.

Zusätzliche Bänder führen nicht zu mehr Ausdrucksstärke

Satz: Für jedes $k \geq 1$ können k-Band-TM durch (1-Band-)TM simuliert werden.

Beweis: Für $k = 1$ ist die Aussage klar.

Es können aber auch mehrere Bänder auf einem simuliert werden:

- Für jedes der k Bänder gibt es je ein endliches Wort (Bandinhalt) und eine Position (Lese-/Schreibkopf)
- Speicherung auf einem Band:
  - Bandinhalte werden hintereinander gespeichert, getrennt durch ein Sonderzeichen $\#$.
  - Steht der Kopf über einer Zelle mit Symbol s, dann wird dort stattdessen ein markiertes Symbol $\hat{s}$ gespeichert.
- Die kodierte Startkonfiguration der k-Band-TM bei Eingabe $a_1 \cdots a_n$ ist also: $\#\hat{a}_1 a_2 \cdots a_n \# \hat{\sqcup} \# \cdots \# \hat{\sqcup} \#$.

Zusätzliche Bänder führen nicht zu mehr Ausdrucksstärke (2)

Satz: Für jedes $k \geq 1$ können k-Band-TM durch (1-Band-)TM simuliert werden.

Beweis (Fortsetzung): Mit dieser Kodierung kann die TM einzelne Schritte der Mehrband-TM simulieren:

- Intitialisierung: Die Eingabe wird in die Kodierung der k-Band-Konfiguration umgeschrieben.
- Berechnungsschritt: Die TM läuft über das gesamte Band und liest die Symbole an den Kopf-Positionen; die k gelesenen Symbole werden in der Zustandsinformation kodiert; dann läuft die TM nochmals über das Band und aktualisiert alle k Kodierungen entsprechend der Übergangsfunktion.
- Falls der Speicher für ein Band erweitert werden muss (Verlassen des bisher verwendeten Speichers nach rechts), verschiebt die TM alle darauf folgenden Zellen entsprechend, kehrt zurück und setzt die Simulation fort.

Zusätzliche Bänder führen nicht zu mehr Ausdrucksstärke (3)

Satz: Für jedes $k \geq 1$ können k-Band-TM durch (1-Band-)TM simuliert werden.

Beweis (Fortsetzung): Die skizzierte Simulation benötigt viele Zustände, um alle relevanten Informationen zwischenspeichern zu können (Zustand der simulierten Maschine, gelesene Symbole unter den k Köpfen, Arbeitszustand bei Hilfsoperationen wie Speicherverschiebung, ...).

$\rightsquigarrow$ Details umständlich, aber im Prinzip machbar. □

Komplexität?

- Die Zahl der Schritte zur Simulation eines Schrittes ist proportional zur Gesamtlänge des beschriebenen Speichers.
- Der maximal beschriebene Speicher pro Band ist proportional zur Zahl der bereits durchgeführten Schritte.
- Die Simulation von n Schritten benötigt also $O(k \cdot n^2)$ Schritte.
  (Sofern n nicht kürzer ist als die Eingabe w; allgemeiner also $O(k \cdot n \cdot \max(n, |w|))$ Schritte.)

Church-Turing-These

Church-Turing-These: Eine Funktion ist genau dann im intuitiven Sinne berechenbar, wenn es eine Turingmaschine gibt, die für jede mögliche Eingabe den Wert der Funktion auf das Band schreibt und anschließend hält.

In der Tat sind eine große Menge von Ansätzen genau gleich stark:

- Turingmaschinen in vielen Varianten (deterministisch/nichtdeterministisch, Einband/Mehrband, einseitig/zweiseitig unendlich, mit/ohne wahlfreiem Zugriff, ...)
- λ-Kalkül nach Church
- Gödel und Herbrands allgemeine rekursive Funktionen
- alle bekannten Programmiersprachen[1]
- Typ-0-Sprachen
- Prädikatenlogik (erster Stufe)

[1] Sofern wir eventuelle technische Beschränkungen der maximalen verwendbaren Speichergröße ignorieren.

Nichtdeterministische Turingmaschinen

Nichtdeterministische TMs

Die nichtdeterministische Turingmaschine (NTM)

... modelliert die Übergangsfunktion als totale Funktion

$$\delta : Q \times \Gamma \to 2^{Q \times \Gamma \times \{L,R,N\}}$$

wobei $2^{Q \times \Gamma \times \{L,R,N\}}$ die Potenzmenge von $Q \times \Gamma \times \{L, R, N\}$ ist;

... kann weiterhin mit einem einzigen Anfangszustand arbeiten.

Läufe werden wie bei DTMs definiert, aber jetzt kann es pro Eingabe viele Läufe geben.

Die Eingabe wird genau dann akzeptiert, wenn mindestens ein Lauf endlich ist und in einer akzeptierenden Konfiguration endet.

Quiz: Nichtdeterministische TM

Quiz: Betrachten Sie die folgende NTM $\mathcal{M} = \langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle$ mit $\Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$: ...

Nichtdeterminismus ≠ mehr Ausdrucksstärke

Satz: Jede NTM kann von einer DTM simuliert werden.

Beweis: Allgemeine Idee:

- Wir simulieren systematisch einen Lauf nach dem anderen.
- Die simulierende TM akzeptiert die Eingabe, wenn ein akzeptierender Lauf gefunden wird;
- andernfalls hält sie nicht an.

Wie kann man systematisch alle möglichen Läufe testen?

- Tiefensuche: Berechne zunächst einen Lauf; falls dieser fehlschlägt, dann gehe zum letzten Entscheidungspunkt zurück und teste eine andere Möglichkeit.
  $\rightsquigarrow$ Problem: Nicht akzeptierende Läufe können unendlich sein.
- Iterativ vertiefende Tiefensuche: Berechne alle Läufe bis zu einer gewissen Tiefe, für immer größere Tiefen.
  $\rightsquigarrow$ Simulation eines Laufs wird bei Maximaltiefe abgebrochen.

Nichtdeterminismus ≠ mehr Ausdrucksstärke

Satz: Jede NTM kann von einer DTM simuliert werden.

Beweis: Die Simulation verwendet eine 3-Band-TM (und kann von einer 1-Band-DTM simuliert werden wie bereits gezeigt):

Band 1: Eingabewort (wird nie verändert)

Band 2: Arbeitsband der simulierten NTM für den aktuellen Lauf

Band 3: Beschreibung der Übergangsentscheidungen des aktuell simulierten Laufs

Für jeden Übergang gibt es nur endlich viele Optionen, sagen wir höchstens ℓ.

Dann kann eine Folge von Entscheidungen als Sequenz von Zahlen in $\{0, \ldots, \ell - 1\}$ beschrieben werden.
$\rightsquigarrow$ Band 3 enthält solch ein Wort über dem Alphabet $\Sigma_\ell = \{0, \ldots, \ell - 1\}$.

Um systematisch alle Optionen für Entscheidungsfolgen zu durchsuchen, werden auf Band 3 alle Wörter über Σ_ℓ in aufsteigender Länge aufgezählt. (Also z.B. $0, 1, \ldots, \ell - 1, 00, 01, \ldots, 0\ell - 1, 10, 11, \ldots, \ell - 1\ell - 1, 000, 001, \ldots$)

Nichtdeterminismus ≠ mehr Ausdrucksstärke

Satz: Jede NTM kann von einer DTM simuliert werden.

Beweis: Arbeitsweise der Simulation:

(1) Initialisiere Band 3 mit dem Inhalt 0.

(2) Kopiere die Eingabe von Band 1 nach Band 2.

(3) Simuliere einen Lauf der NTM auf Band 2. In jedem Schritt wird von Band 3 eine Ziffer aus Σ_ℓ gelesen und der Übergang ausgeführt, der dieser Ziffer entspricht.
   - Falls ein Übergang mit der gelesenen Ziffer nicht möglich ist, gehe zu (4).
   - Falls alle Ziffern auf Band 3 gelesen sind, gehe zu (4).

(4) Prüfe, ob die simulierte NTM in einem Endzustand angehalten hat und akzeptiere in diesem Fall, andernfalls:

(5) Berechne das nächste Element der Aufzählung auf Band 3, lösche Band 2 und gehe zu Schritt (2). □

Komplexität und Terminierung

Satz: Jede NTM kann von einer DTM simuliert werden.

Komplexität: Wenn die NTM einen akzeptierenden Lauf der Länge n hat, dann findet ihn die DTM nach $O(\ell^n)$ Schritten.

$\rightsquigarrow$ Exponentielle Komplexität
(Es ist bis heute unbekannt, ob es eine effizientere Simulation geben könnte – vermutlich nicht, aber der Beweis steht aus.)

Terminierung: Wenn die NTM ein Entscheider ist (auch bei Nichtakzeptanz garantiert hält), dann ist die simulierende DTM . . . nicht unbedingt ein Entscheider.

Der Beweis kann allerdings so abgewandelt werden, dass diese Eigenschaft gilt, also:

Satz: Jede Sprache die von einer NTM entschieden wird, kann auch von einer DTM entschieden werden.

TM, DFA und PDA

Mehrband-NTMs und ihre Gleichmächtigkeit zu 1-Band-NTMs sind analog zum deterministischen Fall.

Damit ist leicht zu sehen:

- Ein DFA kann als DTM aufgefasst werden, welche die Eingabe auf dem Band nur in einer Richtung liest und niemals beschreibt.
- Ein PDA kann als 2-Band-NTM aufgefasst werden, die das zweite Band als Kellerspeicher verwendet.

Unentscheidbare Probleme

(Un)Entscheidbarkeit

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

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

Beispiel: Wir haben bereits erwähnt, dass die Äquivalenz zweier kontextfreier Grammatiken (CFGs) unentscheidbar ist. Sei $\text{enc}(G)$ eine (beliebige) binäre Kodierung der Regeln einer Grammatik G. Das erwähnte Resultat bedeutet formal, dass die Sprache

$$\{\text{enc}(G_1)\#\text{enc}(G_2) \mid \mathbf{L}(G_1) = \mathbf{L}(G_2) \text{ oder } G_1 \text{ bzw. } G_2 \text{ keine CFG}\}$$

über dem Alphabet $\{0, 1, \#\}$ unentscheidbar ist.

Ein konkretes unentscheidbares Problem

Wir wissen bereits:

Satz: Es gibt abzählbar viele Turingmaschinen (Computerprogramme, Algorithmen) aber überabzählbar viele Sprachen. Also sind die meisten Sprachen unentscheidbar.

Das liefert allerdings kein konkretes Beispiel für so eine Sprache.

Wir benötigen ein „erstes“ unentscheidbares Problem. Das folgende ist klassisch:

Das Halteproblem besteht in der folgenden Frage:
Gegeben eine TM $\mathcal{M}$ und ein Wort w, wird $\mathcal{M}$ für die Eingabe w jemals anhalten?

Wir wollen zeigen, dass dieses Problem unentscheidbar ist.

Turingmaschinen kodieren

Will man das Halteproblem als Wortproblem ausdrücken, dann muss man eine Kodierung für TMs als Wörter festlegen.

Jede vernünftige Kodierung einer TM $\mathcal{M} = \langle Q, \Sigma, \Gamma, \delta, q_0, F\rangle$ ist nutzbar, zum Beispiel die folgende:

- Wir verwenden das Alphabet $\{\mathtt{0}, \mathtt{1}, \#\}$.
- Zustände werden in beliebiger Reihenfolge nummeriert (mit Startzustand q_0) und binär kodiert:
  $Q = \{q_0, \ldots, q_n\} \rightsquigarrow \text{enc}(Q) = \text{bin}(0)\#\cdots\#\text{bin}(n)$
- Wir kodieren auch Γ (also auch Σ) und die Bewegungsrichtungen $\{R, L, N\}$ binär.
- Ein Übergang $\delta(q_i, \sigma_n) = \langle q_j, \sigma_m, D\rangle$ wird als 5-Tupel kodiert:
  $\text{enc}(q_i, \sigma_n) = \text{bin}(i)\#\text{bin}(n)\#\text{bin}(j)\#\text{bin}(m)\#\text{bin}(D)$
- Die Übergangsfunktion wird kodiert als Liste aller dieser Tupel, getrennt mit #:
  $\text{enc}(\delta) = (\text{enc}(q_i, \sigma_n)\#)_{q_i\in Q, \sigma_i\in\Gamma}$
- Insgesamt setzten wir $\text{enc}(\mathcal{M}) = \text{enc}(Q)\#\#\text{enc}(\Sigma)\#\#\text{enc}(\Gamma)\#\#\text{enc}(\delta)\#\#\text{enc}(F)$.

Das Halteproblem, formal

Analog kann man auch Wörter über beliebigen Σ binär kodieren.

Damit können wir nun formalisieren:

Das Halteproblem ist das Wortproblem für die Sprache

$$\{\text{enc}(\mathcal{M})\#\#\text{enc}(w) \mid \mathcal{M} \text{ hält bei Eingabe } w\}.$$

Anmerkung: Wörter aus $\{\mathtt{0}, \mathtt{1}, \#\}^*$, die nicht die Form $\text{enc}(\mathcal{M})\#\#\text{enc}(w)$ haben (z.B. weil die Kodierungen fehlerhaft sind), werden laut dieser Definition abgelehnt.

$\rightsquigarrow$ typisch für Entscheidungsprobleme:
Akzeptanz: Antwort auf Frage ist „ja“
Ablehnung: Antwort auf Frage ist „nein“ oder
die Eingabe kodiert gar keine Frage

Unentscheidbarkeit des Halteproblems

Satz: Das Halteproblem ist unentscheidbar.

Beweis: Mittels Widerspruch: Angenommen, es gäbe eine DTM $\mathcal{H}$, die das Halteproblem entscheidet, d.h. $\mathcal{H}$ hält immer und akzeptiert Eingabe $\text{enc}(\mathcal{M})\#\#\text{enc}(w)$ genau dann, wenn $\mathcal{M}$ auf w hält.

Wir verwenden $\mathcal{H}$ als Subroutine einer neuen DTM $\mathcal{D}$, die sich wie folgt verhält:

(1) $\mathcal{D}$ schreibt eine Eingabe $w \in \{\mathtt{0}, \mathtt{1}, \#\}$ zunächst in die Form $w\#\#\text{enc}(w)$ um.
(2) Dann führt $\mathcal{D}$ die TM $\mathcal{H}$ auf dieser Eingabe aus.
(3) Wenn $\mathcal{H}$ die Eingabe nicht akzeptiert, dann hält $\mathcal{D}$ und akzeptiert.
(4) Andernfalls (wenn $\mathcal{H}$ die Eingabe akzeptiert) geht $\mathcal{D}$ in eine Endlosschleife und hält nicht.

Idee: $\mathcal{D}$ hält (und akzeptiert) eine TM-Kodierung $\text{enc}(\mathcal{M})$ genau dann, wenn $\mathcal{M}$ auf der Eingabe $\text{enc}(\mathcal{M})$ nicht hält.

Unentscheidbarkeit des Halteproblems (2)

Satz: Das Halteproblem ist unentscheidbar.

Beweis: $\mathcal{D}$ hält (und akzeptiert) eine TM-Kodierung enc($\mathcal{M}$) genau dann, wenn $\mathcal{M}$ auf der Eingabe enc($\mathcal{M}$) nicht hält.

Was tut $\mathcal{D}$ für die Eingabe enc($\mathcal{D}$)?

- Falls $\mathcal{D}$ auf Eingabe enc($\mathcal{D}$) hält, dann akzeptiert $\mathcal{H}$ die Eingabe enc($\mathcal{D}$)##enc(enc($\mathcal{D}$)), doch dann (laut Definition) hält $\mathcal{D}$ nicht!
- Falls $\mathcal{D}$ auf Eingabe enc($\mathcal{D}$) nicht hält, dann akzeptiert $\mathcal{H}$ die Eingabe enc($\mathcal{D}$)##enc(enc($\mathcal{D}$)) nicht, doch dann (laut Definition) hält $\mathcal{D}$!

Widerspruch. □

Die Universelle Turingmaschine

Satz: Es gibt eine Turingmaschine $\mathcal{U}$, die für Eingaben der Form enc($\mathcal{M}$)##enc(w) das Verhalten von $\mathcal{M}$ auf w simuliert:

- Falls $\mathcal{M}$ auf w hält, dann hält $\mathcal{U}$ auf enc($\mathcal{M}$)##enc(w) mit dem gleichen Ergebnis
- Falls $\mathcal{M}$ auf w nicht hält, dann hält $\mathcal{U}$ auf enc($\mathcal{M}$)##enc(w) ebenfalls nicht

Eine TM $\mathcal{U}$ wie im Satz heißt Universelle Turingmaschine (UTM).

Es ist etwas aufwändig, aber nicht sehr kompliziert, eine UTM zu beschreiben.

Intuitiv gleichwertig: Programmieren eines TM-Simulators in einer beliebigen Programmiersprache.

Semi-Entscheidbarkeit des Halteproblems

Satz: Das Halteproblem ist semi-entscheidbar.

Beweisskizze: Die gesuchte TM startet eine universelle TM auf der Eingabe und akzeptiert, wenn diese UTM hält (egal mit welchem Ergebnis). □

Anmerkung: Falls die Antwort auf das Halteproblem „nein“ ist, dann wird auch diese TM nicht halten.

Nicht Turing-erkennbare Probleme

Das Komplement des Halteproblems („Wird die gegebene TM nicht anhalten?“) liefert ein Beispiel für ein Problem, welches unentscheidbar und nicht semi-entscheidbar ist:

Satz: Das Komplement des Halteproblems ist nicht Turing-erkennbar.

Beweisskizze: Angenommen das Problem wäre Turing-erkennbar. Dann könnten wir das Halteproblem entscheiden, indem wir die Semi-Entscheidungsalgorithmen für Halteproblem und sein Komplement parallel abarbeiten. Einer der beiden Algorithmen muss halten und liefert dann das gesuchte Ergebnis. Widerspruch. □

Turing-Mächtigkeit

Ein Formalismus ist Turing-mächtig, wenn er das Ein-/Ausgabe-Verhalten jeder Turing-Maschine simulieren kann (äquivalent: wenn er eine UTM kodieren kann).

Vorteil: Turing-Mächtigkeit garantiert ein Maximum an Ausdrucksstärke
⇝ gewünscht besonders bei Programmiersprachen

Nachteil: Turing-Mächtigkeit bedeutet, dass alle interessanten Fragen in Bezug auf die berechnete Funktion unentscheidbar sind (z.B. Äquivalenz zweier Darstellungen)
⇝ zumeist unerwünscht, wenn nicht programmiert wird

Versehentlich Turing-mächtig (1)

Immer wieder stellen sich bestimmte Technologien und formale Systeme unerwartet als Turing-mächtig heraus.

C++-Templates

Ein Mechanismus zur generischen Programmierung in C++, bei dem zur Compilezeit (beliebig viele) Code-Templates instantiiert werden. Damit lassen sich TMs simulieren. Daher ist das Halteproblem für C++-Compiler unentscheidbar. Sogar die Frage, ob eine gegebene Textdatei ein gültiges C++-Programm ist, ist unentscheidbar. Praktisch wurde demonstriert, wie der Compiler Primzahlen berechnen und als Compilerfehler ausgeben kann.

Versehentlich Turing-mächtig (2)

Immer wieder stellen sich bestimmte Technologien und formale Systeme unerwartet als Turing-mächtig heraus.

Typsystem von TypeScript:

TypeScript ist eine Erweiterung der Programmiersprache JavaScript mit einem statischen Typsystem. Das Typsystem erlaubt rekursive Typen, Indizierung, Typen beliebiger Größe und weitere Features, mit deren Hilfe TMs simuliert werden können. Damit ist die Frage, ob ein gegebenes TypeScript-Programm korrekt getypt ist, unentscheidbar. Praktisch wurde demonstriert, wie innerhalb TypeScripts Typsystem eine Busy-Beaver-TM simuliert werden kann. Publiziert im März 2017.

Versehentlich Turing-mächtig (3)

Immer wieder stellen sich bestimmte Technologien und formale Systeme unerwartet als Turing-mächtig heraus.

SQL:

Verbreitete Anfragesprache für relationale Datenbanken. Mit Hilfe von rekursiven Hilfstabellen (Common Table Expressions/`WITH RECURSIVE`) kann eine einzelne Abfrage Turingmaschinen simulieren.

Versehentlich Turing-mächtig (4)

Immer wieder stellen sich bestimmte Technologien und formale Systeme unerwartet als Turing-mächtig heraus.

Magic: The Gathering:

Populäres, bekannterweise kompliziertes Tauschkartenspiel. Es ist möglich, eine TM so in ein Spiel zu übersetzen, dass der erste Spieler genau dann gewinnt, wenn die TM hält. Dabei ist sogar sämtliche Entscheidungsfreiheit der Spieler ausgeschlossen. Es ist also sogar unentscheidbar, wer ein gegebenes Spiel gewinnt, selbst wenn kein Spieler eine nichttriviale Entscheidung treffen kann. Erstmals publiziert im März 2019.

Versehentlich Turing-mächtig (5)

Immer wieder stellen sich bestimmte Technologien und formale Systeme unerwartet als Turing-mächtig heraus.

Java Generics:

Mechanismus zur generischen Programmierung in Java. Sollte die Turing-Vollständigkeit von C++-Templates vermeiden. Offenbar ist das nicht gelungen. Erstmals publiziert im Mai 2016.

Versehentlich Turing-mächtig (6)

Immer wieder stellen sich bestimmte Technologien und formale Systeme unerwartet als Turing-mächtig heraus.

Microsoft Powerpoint: Programm zum Erstellen von Präsentationen. Simulationen von Turing-Maschinen auf beliebigem, aber begrenztem Speicher können allein durch Animationen, Links und AutoShapes (ohne VB Makros etc.) realisiert werden. Praktisch wurde lediglich demonstriert, wie man Palindrome gerader Länge erkennen kann. Veröffentlicht im April 2017.
[Video] [Paper]

(Weitere Beispiele siehe `http://beza1e1.tuxen.de/articles/accidentally_turing_complete.html`.)

Zusammenfassung und Ausblick

Die Church-Turing-These besagt, dass jeder Algorithmus mit Turingmaschinen beschrieben werden kann.

Zahlreiche Varianten von TMs führen zur gleichen Ausdrucksstärke, konkret z.B. Mehrband-DTMs oder NTMs.

Das Halteproblem ist unentscheidbar und sein Komplement ist nicht einmal semi-entscheidbar.

Es gibt sehr viele Turing-mächtige Systeme. Das ist nicht immer erwünscht (oder überhaupt bekannt).

Offene Fragen:

- Wie ergibt sich die Korrespondenz von TMs und Typ-0-Sprachen?
- Wenn alle Modelle Typ 0 liefern, was ist dann mit Typ 1?
- Unterscheiden sich Typ 0 und Typ 1 überhaupt?