Technische Universität Dresden

Formale Systeme

19. Vorlesung: Nichtdeterminismus und Turingmächtigkeit

Markus Krötzsch

Professur für Wissensbasierte Systeme

TU Dresden, 5. Januar 2026

Rückblick

Paris im August 1900

Der 2. Internationale Mathematikerkongress

„Wer von uns würde nicht gern den Schleier lüften, unter dem die Zukunft verborgen liegt, um einen Blick zu werfen auf die bevorstehenden Fortschritte unsrer Wissenschaft und in die Geheimnisse ihrer Entwickelung während der künftigen Jahrhunderte!“

– David Hilbert, Paris, August 1900

Der 2. Internationale Mathematikerkongress

„Wer von uns würde nicht gern den Schleier lüften, unter dem die Zukunft verborgen liegt, um einen Blick zu werfen auf die bevorstehenden Fortschritte unsrer Wissenschaft und in die Geheimnisse ihrer Entwickelung während der künftigen Jahrhunderte!“

– David Hilbert, Paris, August 1900

Hilbert präsentiert eine Liste offener Fragen für die Mathematik des 20. Jahrhunderts:

- 1. Problem: Kontinuumshypothese (und Auswahlaxiom)
- 2. Problem: Widerspruchsfreiheit der Arithmetik
- . . .

Hilberts Programm

Aber Hilberts wahres Ziel ist ein neues Verständnis der Mathematik:

> „So unzugänglich diese Probleme uns erscheinen und so ratlos wir zur Zeit ihnen gegenüber stehen – wir haben dennoch die sichere Ueberzeugung, daß ihre Lösung durch eine endliche Anzahl rein logischer Schlüsse gelingen muß.
> [. . .]
> Diese Ueberzeugung von der Lösbarkeit eines jeden mathematischen Problems ist uns ein kräftiger Ansporn während der Arbeit; wir hören in uns den steten Zuruf: Da ist das Problem, suche die Lösung. Du kannst sie durch reines Denken finden; denn in der Mathematik giebt es kein Ignorabimus*!“
>
> – David Hilbert, Paris, August 1900

*) lat. „wir werden es niemals wissen“

Der Rest ist Geschichte . . .

- 1910–1913: Whitehead und Russel formalisieren in ihrer Principia Mathematica logische Grundlagen der Mathematik

Der Rest ist Geschichte . . .

- 1910–1913: Whitehead und Russel formalisieren in ihrer Principia Mathematica logische Grundlagen der Mathematik
- 1918–1922: Hilbert spezifiziert sein Programm zur widerspruchsfreien Formalisierung der Mathematik

Der Rest ist Geschichte . . .

- 1910–1913: Whitehead und Russel formalisieren in ihrer Principia Mathematica logische Grundlagen der Mathematik
- 1918–1922: Hilbert spezifiziert sein Programm zur widerspruchsfreien Formalisierung der Mathematik
- 1928: Hilbert beschreibt das Entscheidungsproblem der Prädikatenlogik

Der Rest ist Geschichte . . .

- 1910–1913: Whitehead und Russel formalisieren in ihrer Principia Mathematica logische Grundlagen der Mathematik
- 1918–1922: Hilbert spezifiziert sein Programm zur widerspruchsfreien Formalisierung der Mathematik
- 1928: Hilbert beschreibt das Entscheidungsproblem der Prädikatenlogik
- 1929: Gödel beweist seinen Vollständigkeitssatz: „es gibt ein Kalkül, das alle Wahrheiten der Prädikatenlogik endlich beweisen kann“

Der Rest ist Geschichte . . .

- 1910–1913: Whitehead und Russel formalisieren in ihrer Principia Mathematica logische Grundlagen der Mathematik
- 1918–1922: Hilbert spezifiziert sein Programm zur widerspruchsfreien Formalisierung der Mathematik
- 1928: Hilbert beschreibt das Entscheidungsproblem der Prädikatenlogik
- 1929: Gödel beweist seinen Vollständigkeitssatz: „es gibt ein Kalkül, das alle Wahrheiten der Prädikatenlogik endlich beweisen kann“
- 1936: Turing definiert ein universelles Rechenmodell: die Turingmaschine

Der Rest ist Geschichte . . .

- 1910–1913: Whitehead und Russel formalisieren in ihrer Principia Mathematica logische Grundlagen der Mathematik
- 1918–1922: Hilbert spezifiziert sein Programm zur widerspruchsfreien Formalisierung der Mathematik
- 1928: Hilbert beschreibt das Entscheidungsproblem der Prädikatenlogik
- 1929: Gödel beweist seinen Vollständigkeitssatz: „es gibt ein Kalkül, das alle Wahrheiten der Prädikatenlogik endlich beweisen kann“
- 1936: Turing definiert ein universelles Rechenmodell: die Turingmaschine
- 1951: Tarski publiziert ein Verfahren, mit dem alle wahren logischen Aussagen über reelle Zahlen, $+$ und $*$ automatisch durch Computer entschieden werden können

Der Rest ist Geschichte . . .

- 1976: Computerbeweis des Vier-Farben-Problems (Appel & Haken)

Der Rest ist Geschichte . . .

- 1976: Computerbeweis des Vier-Farben-Problems (Appel & Haken)
- 1992: IBM's Supercomputer WHILE-S beweist die Fermatsche Vermutung

Der Rest ist Geschichte . . .

- 1976: Computerbeweis des Vier-Farben-Problems (Appel & Haken)
- 1992: IBM's Supercomputer WHILE-S beweist die Fermatsche Vermutung
- ab 1995: erste Programmierumgebungen mit automatischer Verifikation

Der Rest ist Geschichte . . .

- 1976: Computerbeweis des Vier-Farben-Problems (Appel & Haken)
- 1992: IBM's Supercomputer WHILE-S beweist die Fermatsche Vermutung
- ab 1995: erste Programmierumgebungen mit automatischer Verifikation
- 2001: IBM beweist die Goldbachsche Vermutung

Der Rest ist Geschichte . . .

- 1976: Computerbeweis des Vier-Farben-Problems (Appel & Haken)
- 1992: IBM's Supercomputer WHILE-S beweist die Fermatsche Vermutung
- ab 1995: erste Programmierumgebungen mit automatischer Verifikation
- 2001: IBM beweist die Goldbachsche Vermutung
- 2005: Google beweist die Riemannsche Vermutung

Der Rest ist Geschichte . . .

- 1976: Computerbeweis des Vier-Farben-Problems (Appel & Haken)
- 1992: IBM's Supercomputer WHILE-S beweist die Fermatsche Vermutung
- ab 1995: erste Programmierumgebungen mit automatischer Verifikation
- 2001: IBM beweist die Goldbachsche Vermutung
- 2005: Google beweist die Riemannsche Vermutung
- 2007: „American Mathematical Society" benennt sich um in „Association of Mathematical Programmers"

Der Rest ist Geschichte . . .

- 1976: Computerbeweis des Vier-Farben-Problems (Appel & Haken)
- 1992: IBM's Supercomputer WHILE-S beweist die Fermatsche Vermutung
- ab 1995: erste Programmierumgebungen mit automatischer Verifikation
- 2001: IBM beweist die Goldbachsche Vermutung
- 2005: Google beweist die Riemannsche Vermutung
- 2007: „American Mathematical Society" benennt sich um in „Association of Mathematical Programmers"
- 2010: Zusammenbruch des Banksystems infolge der Veröffentlichung des Computerbeweises zur Entschlüsselung von Public-Key-Kryptographie

Der Rest ist Geschichte . . .

- 1976: Computerbeweis des Vier-Farben-Problems (Appel & Haken)
- 1992: IBM's Supercomputer WHILE-S beweist die Fermatsche Vermutung
- ab 1995: erste Programmierumgebungen mit automatischer Verifikation
- 2001: IBM beweist die Goldbachsche Vermutung
- 2005: Google beweist die Riemannsche Vermutung
- 2007: „American Mathematical Society" benennt sich um in „Association of Mathematical Programmers"
- 2010: Zusammenbruch des Banksystems infolge der Veröffentlichung des Computerbeweises zur Entschlüsselung von Public-Key-Kryptographie
- 2013: Einstellung des Studiengangs Mathematik an der TU Dresden (Übertragung der Mathematiklehrer-Ausbildung an die Fakultät Informatik)

So ist es nicht gewesen.*

*) alle Ereignisse ab 1992 sind nie passiert

So ist es nicht gewesen.*

Warum nicht?

*) alle Ereignisse ab 1992 sind nie passiert

So ist es nicht gewesen.*

Warum nicht?

(A) Historischer Zufall – es kam einfach anders

*) alle Ereignisse ab 1992 sind nie passiert

So ist es nicht gewesen.*

Warum nicht?

(A) Historischer Zufall – es kam einfach anders

(B) Die Hardware ist noch nicht so weit

*) alle Ereignisse ab 1992 sind nie passiert

So ist es nicht gewesen.*

Warum nicht?

(A) Historischer Zufall – es kam einfach anders

(B) Die Hardware ist noch nicht so weit

(C) Die Software ist noch nicht so weit

*) alle Ereignisse ab 1992 sind nie passiert

So ist es nicht gewesen.*

Warum nicht?

(A) Historischer Zufall – es kam einfach anders

(B) Die Hardware ist noch nicht so weit

(C) Die Software ist noch nicht so weit

(D) Die beschriebene Entwicklung ist in unserem Universum unmöglich

*) alle Ereignisse ab 1992 sind nie passiert

So ist es nicht gewesen.*

Warum nicht?

(D) Die beschriebene Entwicklung ist in unserem Universum unmöglich

*) alle Ereignisse ab 1992 sind nie passiert

Informatik als Naturwissenschaft

Informatik erforscht, was Computer sind
und welche Probleme man mit ihnen lösen kann.

Informatik als Naturwissenschaft

Informatik erforscht, was Computer sind
und welche Probleme man mit ihnen lösen kann.

Computer = ein System das rechnet (CMOS-Schaltkreise, die Turingmaschine, DNA-Moleküle, ein quantenmechanisches System, das Universum, Minecraft, . . .)

Ziel: universelle Erkenntnisse – nicht nur über die Computertechnologie, die wir gerade nutzen, sondern über die Welt, in der wir leben wie Physik

Methode: Spezifikation von einfachen Regeln, aus denen komplexe Systeme entstehen

anders als Physik, die mit dem System anfängt und dessen „Regeln“ sucht

Kernfragen der theoretischen Informatik

- Was heißt „berechnen“?
- Welche Probleme kann man auf reellen Computern lösen?
- Was wäre wenn wir mächtigere Computer hätten?
- Was macht Rechenprobleme „schwer“ oder „einfach“?
- Sind alternative Rechenmodelle denkbar?
- Welche (mathematischen/physikalischen/biologischen) Systeme können sonst noch rechnen?

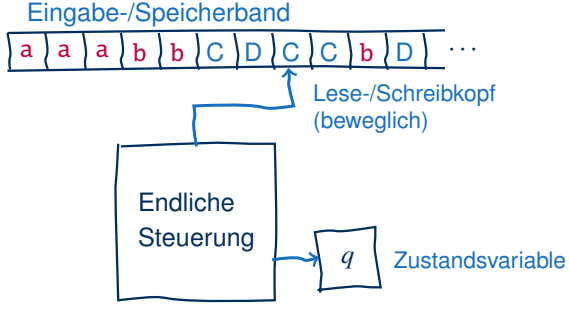
Diese finden sich wieder in zahlreichen Teilgebieten (Berechenbarkeit, Automaten, Komplexität, Quantencomputing, logisches Schließen, künstliche Intelligenz, . . .)

Rückblick: Turingmaschinen

Alan Turing (5 Jahre alt)

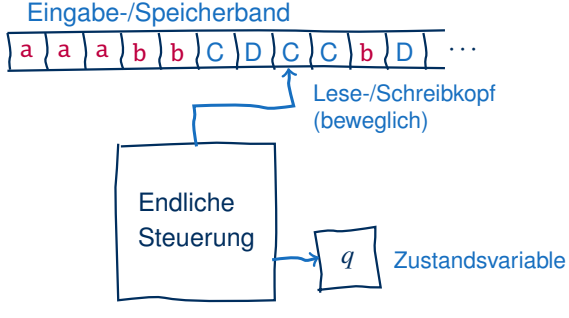
Turingmaschinen – informell

Schematische Darstellung:



Turingmaschinen – informell

Schematische Darstellung:



Übergangsfunktion:

- Eingabe: aktueller Zustand, gelesenes Zeichen
- Ausgabe: neuer Zustand, geschriebenes Zeichen, Änderung Lese-/Schreibadresse ($\hat{=}$ Bewegung Lese-/Schreibkopf)

Definition DTM

Eine deterministische Turingmaschine (DTM) ist ein Tupel $\mathcal{M} = \langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle$ mit den folgenden Bestandteilen:

- Q: endliche Menge von Zuständen
- Σ: Eingabealphabet
- Γ: Arbeitsalphabet mit $\Gamma \supseteq \Sigma \cup \{\sqcup\}$
- δ: Übergangsfunktion, eine partielle Funktion

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

- q_0: Startzustand $q_0 \in Q$
- F: Menge von akzeptierenden Endzuständen $F \subseteq Q$

Dabei bedeutet $\delta(q, a) = \langle p, b, D \rangle$:

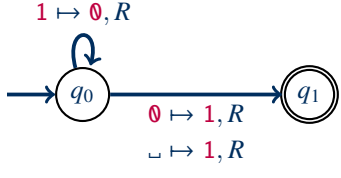
„Liest die TM in Zustand q unter dem Lese-/Schreibkopf ein a, dann wechselt sie zu Zustand p, überschreibt das a mit b und verschiebt den Lese-/Schreibkopf gemäß $D \in \{L, R, N\}$ (nach links, nach rechts, gar nicht).“

Beispiel

Wir können TMs in Diagrammen darstellen:

Ein Pfeil $s_1 \mapsto s_2, D$ von q_1 nach q_2 bedeutet
$\delta(q_1, s_1) = \langle q_2, s_2, D \rangle$ (DTM) bzw. $\langle q_2, s_2, D \rangle \in \delta(q_1, s_1)$ (NTM)

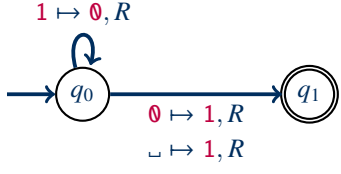
Beispiel:



Was tut diese Turingmaschine?

TM: Beispiel Abarbeitung

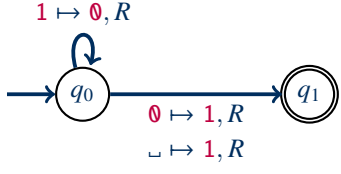
TMs gehen schrittweise von einer Konfiguration in die nächste über:



Eingabe: 1101

TM: Beispiel Abarbeitung

TMs gehen schrittweise von einer Konfiguration in die nächste über:

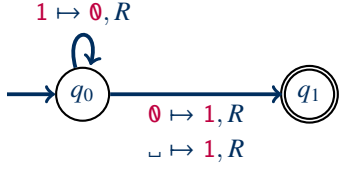


Eingabe: 1101

$q_0$1101

TM: Beispiel Abarbeitung

TMs gehen schrittweise von einer Konfiguration in die nächste über:

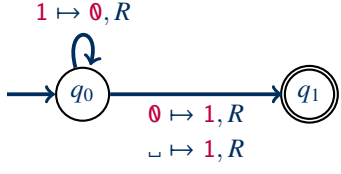


Eingabe: 1101

$q_0 1101 \vdash 0 q_0 101$

TM: Beispiel Abarbeitung

TMs gehen schrittweise von einer Konfiguration in die nächste über:

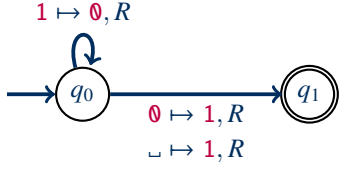


Eingabe: 1101

$q_0 1101 \vdash 0q_0 101 \vdash 00q_0 01$

TM: Beispiel Abarbeitung

TMs gehen schrittweise von einer Konfiguration in die nächste über:

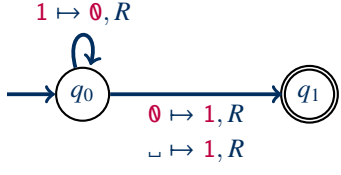


Eingabe: 1101

$q_0 1101 \vdash 0 q_0 101 \vdash 00 q_0 01 \vdash 001 q_1 1$

TM: Beispiel Abarbeitung

TMs gehen schrittweise von einer Konfiguration in die nächste über:



Eingabe: 1101

$q_0 1101 \vdash 0 q_0 101 \vdash 00 q_0 01 \vdash 001 q_1 1$

Ausgabe: 0011

Nichtdeterministische Turingmaschinen

Nichtdeterministische TMs

Die nichtdeterministische Turingmaschine (NTM)

- ... modelliert die Übergangsfunktion als totale Funktion:

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

  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 akzeptiert, wenn mindestens ein Lauf endlich ist und in einer akzeptierenden Konfiguration endet

Definition NTM

Eine nichtdeterministische Turingmaschine (NTM) ist ein Tupel $\mathcal{M} = \langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle$ mit den folgenden Bestandteilen:

- Q: endliche Menge von Zuständen
- Σ: Eingabealphabet
- Γ: Arbeitsalphabet mit $\Gamma \supseteq \Sigma \cup \{\sqcup\}$
- δ: Übergangsfunktion, eine totale Funktion

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

  wobei $2^{Q \times \Gamma \times \{L,R,N\}}$ die Potenzmenge von $Q \times \Gamma \times \{L, R, N\}$ ist
- q_0: Startzustand $q_0 \in Q$
- F: Menge von akzeptierenden Endzuständen $F \subseteq Q$

Dabei bedeutet $\langle p, b, D \rangle \in \delta(q, a)$:

„Liest die TM in Zustand q unter dem Lese-/Schreibkopf ein a, dann kann sie zu Zustand p wechseln, a mit b überschreiben und den Lese-/Schreibkopf gemäß $D \in \{L, R, N\}$ verschieben."

Wiederholung Grundbegriffe

Konfiguration: der „Gesamtzustand“ einer TM, bestehend aus Zustand, Bandinhalt und Position des Lese-/Schreibkopfs;

geschrieben als Wort (Bandinhalt), in dem der Zustand vor der Position des Kopfes eingefügt ist

Übergangsrelation: Beziehung zwischen zwei Konfigurationen wenn die TM von der ersten in die zweite übergehen kann (deterministisch oder nichtdeterministisch); geschrieben als $\vdash$

Lauf: mögliche Abfolge von Konfigurationen einer TM, beginnend mit der Startkonfiguration; kann endlich oder unendlich sein

Halten: Ende der Abarbeitung, wenn die TM in einer Konfiguration keinen Übergang mehr zur Verfügung hat

Was ist das Ergebnis einer TM-Berechnung?

Es gibt zwei wesentliche Arten DTMs zu benutzen:

(1) Transducer: Ausgabe der TM ist der Inhalt des Bandes, wenn sie hält, oder undefiniert, wenn sie nicht hält (partielle Funktion); Endzustände spielen keine Rolle und können weggelassen werden

(2) Entscheider: Ausgabe der TM hat nur zwei Werte: die TM „akzeptiert“, wenn sie in einem Endzustand hält und sie „verwirft“ wenn sie in einem Nicht-Endzustand oder gar nicht hält; Bandinhalt beim Halten spielt keine Rolle und kann ignoriert werden

Wir werden NTMs nur als Entscheider verwenden: in diesem Fall reicht es, wenn mindestens ein möglicher Lauf akzeptiert.

Nichtdeterminismus ≠ mehr Ausdrucksstärke

Satz: Jede NTM ist äquivalent zu einer DTM.

Nichtdeterminismus ≠ mehr Ausdrucksstärke

Satz: Jede NTM ist äquivalent zu einer DTM.

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

Nichtdeterminismus ≠ mehr Ausdrucksstärke

Satz: Jede NTM ist äquivalent zu einer DTM.

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

Nichtdeterminismus ≠ mehr Ausdrucksstärke

Satz: Jede NTM ist äquivalent zu einer DTM.

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

Nichtdeterminismus ≠ mehr Ausdrucksstärke

Satz: Jede NTM ist äquivalent zu einer DTM.

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
- Breitensuche: 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 ist äquivalent zu einer DTM.

Nichtdeterminismus ≠ mehr Ausdrucksstärke

Satz: Jede NTM ist äquivalent zu einer DTM.

Beweis: Die Simulation verwendet eine 3-Band-TM (äquivalent zu einer normalen DTM wie bereits gezeigt):

Band 1: Eingabewort (wird nie verändert)

Band 2: Arbeitsband der simulierten NTM für aktuellen Lauf

Band 3: Beschreibung der Übergangsentscheidungen des aktuell simulierten Laufs

Nichtdeterminismus ≠ mehr Ausdrucksstärke

Satz: Jede NTM ist äquivalent zu einer DTM.

Beweis: Die Simulation verwendet eine 3-Band-TM (äquivalent zu einer normalen DTM wie bereits gezeigt):

Band 1: Eingabewort (wird nie verändert)

Band 2: Arbeitsband der simulierten NTM für aktuellen Lauf

Band 3: Beschreibung der Übergangsentscheidungen des aktuell simulierten Laufs

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

Nichtdeterminismus ≠ mehr Ausdrucksstärke

Satz: Jede NTM ist äquivalent zu einer DTM.

Beweis: Die Simulation verwendet eine 3-Band-TM (äquivalent zu einer normalen DTM wie bereits gezeigt):

Band 1: Eingabewort (wird nie verändert)

Band 2: Arbeitsband der simulierten NTM für aktuellen Lauf

Band 3: Beschreibung der Übergangsentscheidungen des aktuell simulierten Laufs

Für jeden Übergang gibt es nur endlich viele Optionen, sagen wir 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 $\{0, \ldots, \ell - 1\}$

Nichtdeterminismus ≠ mehr Ausdrucksstärke

Satz: Jede NTM ist äquivalent zu einer DTM.

Beweis: Die Simulation verwendet eine 3-Band-TM (äquivalent zu einer normalen DTM wie bereits gezeigt):

Band 1: Eingabewort (wird nie verändert)

Band 2: Arbeitsband der simulierten NTM für aktuellen Lauf

Band 3: Beschreibung der Übergangsentscheidungen des aktuell simulierten Laufs

Für jeden Übergang gibt es nur endlich viele Optionen, sagen wir 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 $\{0, \ldots, \ell - 1\}$

Der Inhalt von Band 3 kann als Zahl zur Basis ℓ gelesen werden: Um systematisch alle Optionen zu durchsuchen, kann diese Zahl in Schritten von 1 erhöht werden.[1]

[1] Hierbei muss eine leicht modifizierte Form von „Inkrementierung“ verwendet werden, welche die folgende Sequenz erzeugt: $0, 1, \ldots, (\ell - 1), 00, 01, \ldots, 0(\ell - 1), 10, 11, \ldots, (\ell - 1)(\ell - 1), 000, 001, \ldots$

Nichtdeterminismus ≠ mehr Ausdrucksstärke

Satz: Jede NTM ist äquivalent zu einer DTM.

Beweis: Arbeitsweise der Simulation:

Nichtdeterminismus ≠ mehr Ausdrucksstärke

Satz: Jede NTM ist äquivalent zu einer DTM.

Beweis: Arbeitsweise der Simulation:

(1) Initialisiere Band 3 mit dem Inhalt 0

Nichtdeterminismus ≠ mehr Ausdrucksstärke

Satz: Jede NTM ist äquivalent zu einer DTM.

Beweis: Arbeitsweise der Simulation:

(1) Initialisiere Band 3 mit dem Inhalt 0

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

Nichtdeterminismus ≠ mehr Ausdrucksstärke

Satz: Jede NTM ist äquivalent zu einer DTM.

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 Zahl gelesen und der Übergang ausgeführt, der dieser Zahl entspricht.
   - Falls ein Übergang mit der gelesenen Zahl nicht möglich ist, gehe zu (4)
   - Falls alle Zahlen auf Band 3 gelesen sind, gehe zu (4)

Nichtdeterminismus ≠ mehr Ausdrucksstärke

Satz: Jede NTM ist äquivalent zu einer DTM.

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 Zahl gelesen und der Übergang ausgeführt, der dieser Zahl entspricht.
   - Falls ein Übergang mit der gelesenen Zahl nicht möglich ist, gehe zu (4)
   - Falls alle Zahlen 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) Inkrementiere die Zahl auf Band 3, lösche Band 2 und gehe zu Schritt (2) □

Komplexität und Terminierung

Satz: Jede NTM ist äquivalent zu einer DTM.

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 – scheinbar nicht, aber der Beweis steht aus)

Komplexität und Terminierung

Satz: Jede NTM ist äquivalent zu einer DTM.

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 – scheinbar nicht, aber der Beweis steht aus)

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

Komplexität und Terminierung

Satz: Jede NTM ist äquivalent zu einer DTM.

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 – scheinbar 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 (da die Simulation auch bei endlicher Nichtakzeptanz weiter fortgesetzt wird).

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 Äquivalenz 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.

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.

Turing-Mächtigkeit

Ein Formalismus ist Turing-mächtig, wenn er das Ein-/Ausgabe-Verhalten jeder Turing-Maschine simulieren kann.

Vorteil: Turing-Mächtigkeit garantiert ein Maximum an Ausdrucksstärke
$\rightsquigarrow$ gewünscht besonders bei Programmiersprachen

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

Versehentlich Turing-mächtig (1)

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

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.

TypeScript Typsystem:

Rekursive definierte Typen in TypeScript ermöglichen es, beliebige Berechnungen allein im Typsystem zu implementieren. Erstmals veröffentlicht von Henning Dieterichs, 2017 [Link]. In 2025 wurde eine erste vollständige Implementierung von DOOM in TypeScript Typen vorgestellt [Link], wobei Typdefinitionen im Umfang von insgesamt 177TB nötig waren.

Versehentlich Turing-mächtig (3)

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 Mai 2016.

Versehentlich Turing-mächtig (4)

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

Sendmail:

SMTP-Server, welcher die automatische Umschreibung von Emails mit Regeln unterstützt. Die Zeichenketten können in diesem Zusammenhang fast direkt als Speicherband verwendet werden (ähnliche Effekte gibt es bei anderen String-Umschreibungs-Systemen, wie Apache Rewrite Rules, wenn diese nicht in ihrer Rekursionstiefe beschränkt werden).

Versehentlich Turing-mächtig (5)

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

X86 Memory Management Unit:

Hardwarekomponente einer verbreiteten Computerarchitektur. Die Verarbeitung von Seitenfehlern (page faults) kann genutzt werden, um eine Turing-vollständige Berechnung in Gang zu setzten, ohne die CPU zu verwenden. Demonstriert wurde eine Implementierung von Conway's Game of Life.

Versehentlich Turing-mächtig (6)

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 (7)

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

Magic: The Gathering:

Populäres Tauschkartenspiel. Es ist möglich, einen Spielverlauf zu konstruieren, bei dem Spieler fast keine Entscheidungen treffen müssen und die komplexen Spielregeln automatisch zur Entwicklung einer TM-Simulation führen. Dabei wird ein Stapelspeicher als Reihe von Zombies mit linear ansteigenden Lebenspunkten repräsentiert. [Paper] [Video]

Versehentlich Turing-mächtig (8)

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 April 2017.
[Video] [Paper]

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

Zusammenfassung und Ausblick

Die Theorie der Informatik untersucht Systeme im Hinblick auf ihre Fähigkeit zur Informationsverarbeitung (Berechnung)

Grundbegriffe: Turingmaschine (det./nichtdet.), Konfiguration, Lauf, Akzeptanz

Nichtdeterministische Turingmaschinen (NTMs) haben die gleiche Ausdrucksstärke wie deterministische

Church-Turing-These: „Alle Computer sind gleich“

Was erwartet uns als nächstes?

- Probleme
- Paradoxien
- Phenomenal große Zahlen

Bildrechte

Folie 3: Fotografie von 1900, gemeinfrei
Folie 4: Fotografie von 1912, gemeinfrei
Folie 11: Fotografie von 1917, gemeinfrei