TECHNISCHE UNIVERSITÄT DRESDEN

# FORMALE SYSTEME

**19. Vorlesung: Nichtdeterminismus und Unentscheidbarkeit**

**Markus Krötzsch**
**Professur für Wissensbasierte Systeme**

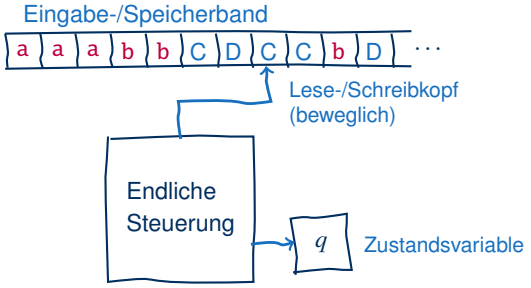
TU Dresden, 18. Dezember 2017

## 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 $\Sigma$, Arbeitsalphabet $\Gamma \supseteq \Sigma \cup \{\sqcup\}$, Startzustand $q_0 \subseteq Q$, Endzuständen $F \subseteq Q$, und einer partiellen Übergangsfunktion

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



## 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 gleichstark:

- Turingmaschinen in vielen Varianten (deterministisch/nichtdeterministisch, Einband/Mehrband, einseitig/zweiseitig unendlich, mit/ohne wahlfreiem Zugriff, ...)
- $\lambda$-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:

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

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



## Nichtdeterminismus ≠ mehr Ausdrucksstärke

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

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 $\ell$.

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 $\ell$ gelesen werden: Um systematisch alle Optionen zu durchsuchen, kann diese Zahl in Schritten von $1$ erhöht werden

## Nichtdeterminismus ≠ mehr Ausdrucksstärke

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 um $1$, lösche Band 2 und gehe zu Schritt (2) □



## Komplexität und Terminierung

Jede NTM ist äquivalent zu einer DTM.

**Komplexität:** Wenn die NTM einen akzeptierenden Lauf der Länge $n$ hat, dann findet ihn die DTM nach spätestens $\ell^n$ 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.

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

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



# Unentscheidbare Probleme

## (Un)Entscheidbarkeit

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 $\mathbf{L} = \mathbf{L}(\mathcal{M})$. Andernfalls heißt **L** unentscheidbar.

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

Beispiel: Wir haben bereits erwähnt, dass die Äquivalenz zweier kontextfreier Grammatiken (CFGs) unentscheidbar ist. Sei 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 $\{0, 1, \#\}$
- 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 $\Gamma$ 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 $\Sigma$ binär kodieren.

Damit kann man formal auch sagen:

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 $\{0, 1, \#\}^*$, 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

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 Eingaben enc($\mathcal{M}$)##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 $w \in \{0, 1, \#\}$ zunächst in die Form $w$##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, dann geht $\mathcal{D}$ in eine Endlosschleife und hält nicht.

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



## Unentscheidbarkeit des Halteproblems (2)

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.

### 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 (3)

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 (4)

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 (5)

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.



## Versehentlich Turing-mächtig (6)

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 (7)

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 Animtationen, 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

Nichtdeterministische Turingmaschinen (NTMs) haben die gleiche Ausdruckstärke wie deterministische

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

Es gibt sehr viele Turing-mächtige Systeme. Das ist nicht immer erwünscht.

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?



## Frohe Weihnachten!