EINE EINFÜHRUNG ZUM SEMINAR MATHEMATIK & INFORMATIK

Philipp Rohde

Wintersemester 2001/02

ZUSAMMENFASSUNG. Wir wiederholen die Definition einer Turing-Maschine (deterministisch und nicht-deterministisch) und betrachten deren Zeit- und Raumbedarf. Dabei werden die grundlegenden Komplexitätsklassen wie P und NP eingeführt. Abschnitt 4 behandelt das Problem der Unentscheidbarkeit. Im letzten Abschnitt gehen wir kurz auf die Verbindung von Turing-Maschinen und Logik ein.

Als Grundlage dienten Kapitel 1 bis 6 aus

Papadimitriou, Christos M.: ***Computational Complexity*****, Addison-Wesley 1994**.

1. Turing-Maschinen

Es existieren verschieden Ansätze, den Begriff der Turing-Maschine formal zu definieren. Wir legen hier die folgende Interpretation zugrunde: eine Turing-Maschine besteht aus einem Speichermedium und einem Schreib-Lese-Kopf (Cursor). Als Speichermedium fungiert ein Band mit einer, in eine Richtung potentiell unendlichen Ausdehnung. Das Band ist in Abschnitte – sogenannte Bandquadrate – unterteilt und jedes Bandquadrat enthält genau ein Symbol aus einer festgelegten Menge. Der Cursor befindet sich stets genau über einem der Bandquadrate, dessen Inhalt er lesen und verändern kann. Die Maschine selbst befindet sich stets in genau einem von endlich vielen möglichen Zuständen und sie wird durch ein Programm gesteuert: der Cursor liest den Inhalt des aktuellen Bandquadrats und in Abhängigkeit vom dem Zustand, in dem sie sich aktuell befindet, führt sie die folgenden drei Handlungen aus:

(1) Übergang in einen (eventuell neuen) Zustand;
(2) Schreiben eines Symbols in das aktuelle Bandquadrat. Dabei wird das vorherige Symbol überschrieben;
(3) Verweilen des Cursors an der aktuellen Position oder Bewegung des Cursors um genau ein Quadrat nach rechts oder links.

Formal definieren wir dazu:

(1.1) Definition: Eine **Turing-Maschine** ist ein Quadrupel $M = (K, \Sigma, \delta, s)$. Dabei sei K eine endliche Menge von **Zuständen**. $s \in K$ heißt **Initialzustand**. Σ sei eine endliche Menge von **Symbolen** (das **Alphabet** von M). Wir setzen voraus, daß K und Σ disjunkt sind. Σ enthält stets die beiden Symbole **Blanko** $\sqcup$ und **Start** $\triangleright$. Das **Programm** δ ist eine Funktion

$$\delta : K \times \Sigma \to (K \cup \{\text{STOPP}, \text{JA}, \text{NEIN}\}) \times \Sigma \times \{\leftarrow, \to, -\}.$$

Die **Endzustände** sind: der **Haltezustand** STOPP, der **Zustand der Annahme** JA und der **Zustand der Zurückweisung** NEIN. Die **Cursorrichtungen** sind: **links** $\leftarrow$, **rechts** $\to$ und **keine Bewegung** $-$. Wir setzen voraus, daß die drei Endzustände und die drei Cursorrichtungen weder in K noch in Σ enthalten sind.

Mathematisches Institut, Universität Bonn, Beringstr. 4, 53115 Bonn
e-mail: rohde@math.uni-bonn.de

Damit haben wir das oben skizzierte Bild formalisiert, denn für jeden aktuellen Zustand $q \in K$ der Turing-Maschine M und jedes gelesene Symbol $\sigma \in \Sigma$ liefert die Funktion δ die Anweisung für die drei Handlungen: ist $\delta(q, \sigma) = (p, \rho, d)$, dann gibt p den Zustand an, in den die Maschine übergeht, ρ das Symbol, mit dem σ überschrieben wird und $d \in \{\leftarrow, \rightarrow, -\}$ die Richtung, in welche der Cursor bewegt wird.

Wir setzen ferner voraus: ist $\delta(q, \triangleright) = (p, \rho, d)$ für beliebige Zustände q und p, dann ist $\rho = \triangleright$ und $d = \rightarrow$. So ist garantiert, daß sich der Cursor nicht links über $\triangleright$ hinaus bewegen kann und $\triangleright$ nicht gelöscht wird. Wir erhalten damit das Bild eines links begrenzten Bandes:[1]

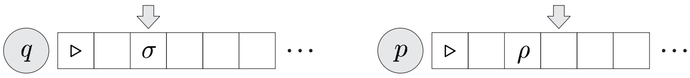

ABB. 1. Eine Turing-Maschine mit $\delta(q, \sigma) = (p, \rho, \rightarrow)$

Eine **Berechnung** der Maschine M ist der folgende Prozeß: zu Beginn enthält das Band das Startsymbol $\triangleright$, gefolgt von einer endlichen Zeichenkette $x \in (\Sigma - \{\sqcup\})^*$ (die **Eingabe**) und rechts daneben nur noch $\sqcup$. Die Maschine selbst befindet sich im Zustand s. Nun führt die Maschine sukzessive die drei Handlungen gemäß dem Programm δ aus. Da δ eine totale Funktion ist, ist die nächste Handlung stets definiert.

Die einzige Möglichkeit, so daß die Berechnung nicht fortgesetzt werden kann, ist das Eintreten eines der drei Endzustände STOPP, JA oder NEIN. In diesem Fall **hält** die Maschine. Die **Ausgabe** $M(x)$ sei wie folgt definiert: falls JA eintritt, dann **akzeptiert** M die Eingabe und es sei $M(x) = $ JA. Bei NEIN wird die Eingabe **zurückgewiesen** und wir setzen $M(x) = $ NEIN. Wenn die Maschine mit STOPP hält, dann sei $M(x) = y$ das beschriebene Band zu diesem Zeitpunkt, d.h. die endliche Zeichenkette y zwischen $\triangleright$ (exklusive) und dem letzten Symbol ungleich $\sqcup$ (inklusive). y ist hier stets eine endliche, möglicherweise leere Folge. Falls M bei Eingabe x überhaupt nicht hält, dann schreiben wir $M(x) = \uparrow$. Z.B. zeigt Abb. 2 eine Berechnung mit Eingabe $x = 010011$ und Ausgabe $M(x) = 10 \sqcup 1100$. Man beachte, daß im Gegensatz zur Eingabe die Ausgabe durchaus Blankosymbole enthalten kann.

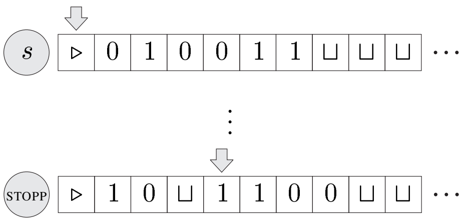

ABB. 2. Anfang und Ende einer Berechnung

(1.2) Definition: Eine Teilmenge $L \subseteq (\Sigma - \{\sqcup\})^*$ heißt **Sprache**.[2] Sei M eine Turing-Maschine mit Alphabet Σ. Wir sagen, M **entscheidet** L, falls $M(x) = $ JA für alle $x \in L$ und $M(x) = $ NEIN für alle $x \notin L$ gilt. Eine Sprache L heißt **rekursiv**, falls sie von einer Turing-Maschine entschieden wird.

M **akzeptiert** L, falls $M(x) = $ JA für alle $x \in L$ und $M(x) = \uparrow$ für alle $x \notin L$ gilt. Eine Sprache L heißt **rekursiv aufzählbar**, falls sie von einer Turing-Maschine akzeptiert wird.

Für eine Turing-Maschine M sei $L(M)$ die Sprache, die von M akzeptiert wird. Man beachte, daß nicht alle Turing-Maschinen eine Sprache akzeptieren (Beispiel: die Maschine hält im Zustand NEIN). In diesem Fall setzen wir $L(M) = \emptyset$.

---

[1] In der Literatur werden häufig auch Turing-Maschinen mit einem in beiden Richtungen endlosen Band betrachtet.

[2] Nicht zu Verwechseln mit dem üblichen Begriff der Logik.

Offensichtlich gilt:

(1.3) Lemma: *Ist eine Sprache L rekursiv, dann ist sie auch rekursiv aufzählbar.*

Beweis: Sei $M = (K, \Sigma, \delta, s)$ eine Turing-Maschine, die L entscheidet. Wir definieren eine Maschine M^*, die L akzeptiert. M^* verhält sich exakt wie M mit einem Unterschied: wenn M die Eingabe zurückweisen würde, dann geht M^* anstelle dessen in einen neuen Zustand über, der eine kontinuierliche Rechtsbewegung auslöst. Formal wählen wir einen Zustand p mit $p \notin K$. Dann sei $\delta^*(q, \sigma) = (p, \rho, d)$, falls $\delta(q, \sigma) = (\text{NEIN}, \rho, d)$; $\delta^*(p, \sigma) = (p, \sigma, \rightarrow)$ für alle $\sigma \in \Sigma$ und $\delta^*(q, \sigma) = \delta(q, \sigma)$ sonst. Dann leistet $M^* = (K \cup \{p\}, \Sigma, \delta^*, s)$ das gewünschte. ■

Eine mehrbändige Turing-Maschine besitzt pro Band einen Cursor, der unabhängig von den anderen bewegt werden kann. Die Maschine befindet sich weiterhin zu jedem Zeitpunkt in genau einem Zustand. Formal definieren wir:

(1.4) Definition: Sei $k \geq 1$. Eine **Turing-Maschinen mit k Bändern** ist ein Quadrupel $M = (K, \Sigma, \delta, s)$ mit K, Σ und s wie bisher. Das Progamm δ ist nun eine Funktion:

$$\delta : K \times \Sigma^k \rightarrow (K \cup \{\text{STOPP}, \text{JA}, \text{NEIN}\}) \times (\Sigma \times \{\leftarrow, \rightarrow, -\})^k .$$

Jedes Band beginnt mit $\triangleright$ und ist nach rechts hin offen. Wir setzen analog zu oben voraus, daß die Cursor nicht links über $\triangleright$ hinaus bewegt werden können. Die Eingabe der Maschine ist auf dem ersten Band enthalten. Hält die Maschine im Endzustand STOPP, dann enthält das letzte Band die Ausgabefolge. Alle anderen Begriffe werden entsprechend übertragen.

Wir werden in Satz (2.3) sehen, daß dies keine wesentliche Erweiterung ist.

2. Zeit- und Raumbedarf von Turing-Maschinen

(2.1) Definition: Wir betrachten hier stets Funktionen $\mathbb{N} \rightarrow \mathbb{N}$ als obere Schranken für den Zeit- und Raumbedarf von Turing-Maschinen. n sei stets das Argument dieser Funktionen. Bei Funktionen wie $\sqrt{n}$, $\log(n)$ etc. sei der Funktionswert stets aufgerundet, d.h. für $f : \mathbb{N} \rightarrow \mathbb{R}$ ersetzen wir $f(n)$ durch $\max\{\lceil f(n) \rceil, 0\}$.
Seien f und g Funktionen von $\mathbb{N}$ nach $\mathbb{N}$. Wir schreiben $f(n) = \mathcal{O}(g(n))$, falls f nicht schneller als g wächst, d.h. falls es natürliche Zahlen c und N gibt mit $f(n) \leq c \cdot g(n)$ für alle $n \geq N$. Wir schreiben $f(n) = \Omega(g(n))$, falls $g(n) = \mathcal{O}(f(n))$ gilt und $f(n) = \Theta(g(n))$, falls sowohl $f(n) = \mathcal{O}(g(n))$ als auch $f(n) = \Omega(g(n))$ gilt.

(2.2) Beispiel: Sei $p(n)$ ein Polynom vom Grad d. Dann gilt $p(n) = \Theta(n^d)$. Für $b \in \mathbb{N}$ mit $b > 1$ gilt stets $p(n) = \mathcal{O}(b^n)$, aber nicht $p(n) = \Omega(b^n)$, d.h. jedes Polynom wächst echt langsamer als jede Exponentialfunktion mit Basis > 1.

I. Zeit. Die **benötigte Zeit** bei Eingabe x ist die Anzahl der Schritte der Berechnung, ausgehend vom Initialzustand bei Eingabe x bis zu dem Zeitpunkt, an dem die Maschine hält. Im Fall $M(x) = \uparrow$ denken wir uns die benötige Zeit als unendlich. Eine Funktion $f : \mathbb{N} \rightarrow \mathbb{N}$ heißt **Zeitschranke** für M, falls für jede Eingabe x die benötigte Zeit höchstens $f(|x|)$ ist. Dabei sei $|x|$ die Länge der Eingabefolge. Man beachte, daß für jede „sinnvolle“ Zeitschranke $f(n) \geq n$ gilt, da jede Turing-Maschine schon n Schritte benötigt, um die gesamte Eingabe zu lesen.
$\mathsf{TIME}(f(n))$ sei die Menge der Sprachen, die von Turing-Maschinen mit Zeitschranke $f(n)$ entschieden werden. $\mathsf{TIME}(f(n))$ ist eine sogenannte **Komplexitätsklasse**.

(2.3) Satz: *Eine Turing-Maschine M mit k Bändern und Zeitschranke $f(n)$ kann durch eine Turing-Maschine M^* mit einem Band und Zeitschranke $\mathcal{O}(f(n)^2)$ simuliert werden.*

Beweis: Wir kodieren die Zeichenketten der k Bänder von $M = (K, \Sigma, \delta, s)$ durch Konkatenation, allerdings ohne die $\triangleright$-Symbole. Außerdem müssen die Positionen der Cursor und das jeweilige rechte Ende der einzelnen Bänder kodiert werden. Dazu wählen wir $\Sigma^* = \Sigma \cup \underline{\Sigma} \cup \{\diamond, \underline{\diamond}, \triangleleft\}$ mit $\underline{\Sigma} = \{\underline{\sigma} \mid \sigma \in \Sigma\}$. Die $\underline{\sigma}$ kodieren die Symbole, die gerade von einem Cursor gelesen werden, $\diamond$ ist die Kodierung von $\triangleright$ (und kann nach links überquert werden), $\triangleleft$ markiert das rechte Ende des simulierten Bandes. Das folgend Beispiel illustriert die Kodierung einer dreibändigen Maschine:

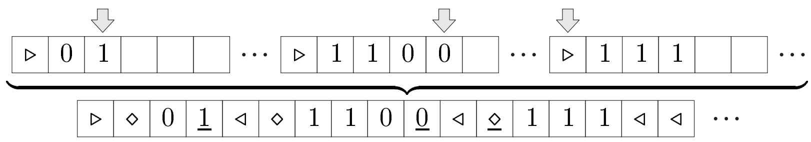

ABB. 3. Kodierung einer mehrbändigen Turing-Maschine

Am Anfang der Simulation wandelt M^* den Input

$$\triangleright x_0 x_1 x_2 \dots x_n \quad \text{in} \quad \triangleright\diamond x_0 x_1 x_2 \dots x_n \triangleleft\diamond\triangleleft \dots \diamond\triangleleft\triangleleft$$

um. Ein Schritt von M wird wie folgt simuliert: M^* fährt ihren Cursor zweimal über das beschriebene Band hin- und her. Beim ersten Mal merkt sie sich die k unterstrichenen Symbole durch Zustände (ein neuer für jede der möglichen Kombinationen inkl. dem aktuellen Zustand von M). Beim zweiten Mal simuliert M^* mit dieser Information das Programm von M, indem sie hintereinander die k unterstrichenen Symbole verändert und gegebenenfalls den Unterstrich auf einen der jeweiligen Nachbarn „verschiebt". Dabei muß eventuell eines der k simulierten Bänder rechts verlängert werden. Dies kann durch weitere neue Zustände und ein neues Symbol zur Markierung bewerkstelligt werden. Am Ende der Simulation löscht M^* die ersten $k-1$ Zeichenketten der simulierten Bänder, so daß nur noch die Zeichenkette des k-ten simulierten Bandes als Ausgabe übrigbleibt.

M hält nach höchstens $f(|x|)$ Schritten. Daher kann keine der k Zeichenketten länger als $f(|x|)$ werden (eine Turing-Maschine kann nicht mehr Raum als Zeit verbrauchen) und die Länge der Zeichenkette von M^* ist höchstens $k \cdot f(|x|) + k + 1$. Diese wird für jeden simulierten Schritt von M zweimal überquert. Die gesamte Simulation benötigt also $\mathcal{O}(k^2 \cdot f(|x|)^2)$ Schritte. Da k unabhängig von x ist, erhalten wir die gewünschte Schranke. ■

Wir können jede Turing-Maschine linear „beschleunigen". Die berechtigt uns auch zum Gebrauch der $\mathcal{O}$-Notation:

(2.4) Satz (Lineare Beschleunigung): *Sei $L \in \mathsf{TIME}(f(n))$ und $\varepsilon > 0$. Dann ist auch $L \in \mathsf{TIME}(\varepsilon \cdot f(n) + n + 2)$.*

Beweis: Sei M eine Turing-Maschine mit Zeitschranke $f(n)$, die L entscheidet. Wir simulieren M durch eine Turing-Maschine M^*, wobei jedes Symbol von M^* mehrere Symbole von M kodiert (die Anzahl k hängt von ε ab und wird vorher fixiert). Daneben besitzt M^* die k-fache Anzahl von Zuständen zur Markierung der Cursorposition innerhalb eines k-Tupels von Symbolen. Auf diese Art können wir k Schritte von M durch eine feste Anzahl von Schritten von M^* simulieren. Daraus erhalten wir die gewünschte lineare Beschleunigung (der lineare Term entsteht aus einer notwendigen Anfangsphase). ■

Wir definieren nun eine wichtige Komplexitätsklasse:

(2.5) Definition: $\mathsf{P} := \bigcup\{\mathsf{TIME}(p(n)) \mid p \text{ ist ein Polynom in } n\}$.

II. Raum. Da sie nicht Teil der eigentlichen Berechnung sind, wollen wir das reine Lesen der Eingabe („read-only") sowie das reine Schreiben der Ausgabe („write-only") nicht zum Raumverbrauch hinzurechnen. Wir definieren daher:

(2.6) Definition: Sei $k > 2$. Eine k-bändige Turing-Maschine **mit Ein- und Ausgabeband** (kurz **I/O-Maschine**) ist eine Turing-Maschine mit folgender Einschränkung: ist $\delta(q, \sigma_1, \ldots, \sigma_k) = (p, \rho_1, d_1, \ldots, \rho_k, d_k)$, dann gilt $\rho_1 = \sigma_1$, $d_k \neq \leftarrow$ und aus $\sigma_1 = \sqcup$ folgt $d_1 = \leftarrow$. Damit ist gewährleistet, daß von dem ersten Band nur gelesen wird und daß sich der Cursor auf dem letzten Band nur nach rechts bewegen kann, dieses also als reines Ausgabeband betrachtet werden kann.

(2.7) Lemma: *Eine Turing-Maschine M mit k Bändern und Zeitschranke $f(n)$ kann durch eine I/O-Maschine M^* mit $k+2$ Bändern und Zeitschranke $\mathcal{O}(f(n))$ simuliert werden.*

Beweis: M^* kopiert zunächst den Inhalt des ersten Bandes auf das zweite und simuliert dann die k Bänder von M auf ihrem zweiten bis $(k+1)$-ten Band. Wenn M hält, kopiert M^* ihre Ausgabe auf das letzte Band und hält ebenfalls. ■

Damit können wir den Raumverbrauch ohne Ein- und Ausgabe definieren:

(2.8) Definition: Sei M eine k-bändige I/O-Turing-Maschine. Der **benötigte Raum** bei Eingabe x ist die Summe $\sum_{i=2}^{k-1} |x_i|$, wobei x_i der Inhalt i-ten Bandes ist. Eine Funktion $f : \mathbb{N} \to \mathbb{N}$ heißt **Raumschranke** für M, falls für jede Eingabe x der benötigte Raum höchstens $f(|x|)$ ist.
$\mathsf{SPACE}(f(n))$ sei die Menge der Sprachen, die von einer I/O-Turing-Maschine mit Raumschranke $f(n)$ entschieden werden.
Im Gegensatz zu den Zeitschranken sind bei den Raumschranken Funktionen mit $f(n) < n$ durchaus sinnvoll, z.B. $\mathsf{L} := \mathsf{SPACE}(\log(n))$.

Durch eine entsprechende Modifikation des Beweises von Satz (2.4) erhalten wir:

(2.9) Satz (Lineares Platzsparen): *Sei $L \in \mathsf{SPACE}(f(n))$ und $\varepsilon > 0$. Dann ist auch $L \in \mathsf{TIME}(\varepsilon \cdot f(n) + 2)$.* ■

Damit ist auch hier der Gebrauch der $\mathcal{O}$-Notation gerechtfertigt. Wir definieren wieder:

(2.10) Definition: $\mathsf{PSPACE} := \bigcup\{\mathsf{SPACE}(p(n)) \mid p$ ist ein Polynom in $n\}$.

3. Nicht-deterministische Maschinen

Wir betrachten in diesem Abschnitt ein Maschinenmodell, das eher eine theoretische als eine praktische Relevanz hat:

(3.1) Definition: Eine **nicht-deterministische Turing-Maschinen** $N = (K, \Sigma, \Delta, s)$ wird wie bisher definiert, allerdings ist das Programm keine Funktion mehr, sondern eine Relation:

$$\Delta \subseteq \big(K \times \Sigma\big) \times \big((K \cup \{\text{STOPP}, \text{JA}, \text{NEIN}\}) \times \Sigma \times \{\leftarrow, \rightarrow, -\}\big).$$

Eine derartige Maschine besitzt zu einem gegebenen Zeitpunkt eventuell keine eindeutige nächste Handlung mehr. Es können mehrere (allerdings nur endlich viele) Fortsetzung der Berechnung existieren – oder auch gar keine.
Eine Eingabe wird von N **akzeptiert**, falls eine Berechnung existiert, die Eingabe akzeptiert, d.h. falls an jedem „Verzweigungspunkt" eine Entscheidung der Maschine getroffen werden kann, die zum Zustand JA führt. Man beachte aber, daß andere Berechnungen die Eingabe zurückweisen können. Entsprechend wird eine Eingabe nur dann **zurückgewiesen**, wenn es keine Folge von Entscheidungen gibt, die mit Zustand JA enden.
Eine Sprache L wird von N **entschieden**, falls für jede Eingabe x gilt: $x \in L \iff N$ akzeptiert x und $x \notin L \iff N$ weist x zurück.

Eine Funktion $f : \mathbb{N} \to \mathbb{N}$ heißt **Zeitschranke** (**Raumschranke**) für N, falls für jede Eingabe x die benötigte Zeit (der benötigte Raum) jeder Berechnung – d.h. bei jeder Folge von Entscheidungen – höchstens $f(|x|)$ ist.[3]
Die Komplexitätsklassen $\mathsf{NTIME}(f(n))$ und NP bzw. $\mathsf{NSPACE}(f(n))$ und $\mathsf{NPSPACE}$ werden analog zum deterministischen Fall definiert.

Da Funktionen spezielle Relationen sind, erhalten wir unmittelbar:

(3.2) Lemma: *Es gilt* $\mathsf{P} \subseteq \mathsf{NP}$. ■

Die Umkehrung ist ein berühmtes offenes Problem, dessen Lösung 1 Million US-Dollar wert ist:

(3.3) Frage: Gilt $\mathsf{P} = \mathsf{NP}$?

Es gibt eine Vielzahl guter Gründe, warum diese Gleichheit wahrscheinlich *nicht* gilt. Wir werden im Laufe des Seminars noch öfters mit dieser Frage konfrontiert werden.

(3.4) Beispiel: Ein wichtiger Vertreter der Klasse NP ist das **Problem des Handlungsreisenden** (**traveling salesman problem**, TSP): Gegeben seien n Städte und ihre Entfernungen untereinander. Gesucht ist die kürzeste Tour, so daß alle Städte mindestens einmal besucht werden. Die „Entscheidungsversion" $\mathsf{TSP}(B)$ ist die Frage, ob es zu einem gegebenen Budget B eine Tour gibt, die höchstens so lang wie B ist. Es ist offen, ob diese Probleme in P liegen, aber man sieht leicht, daß sie in NP liegen. Eine entsprechende Maschine erhält als Eingabe die Anzahl der Städte n und die Entfernungen (z.B. als Matrix). Zuerst schreibt sie eine zufällige Zeichenkette (Länge hängt von n ab) auf ein Band. Dann überprüft sie, ob es sich um eine Permutation der Zahlen $1, \ldots, n$ handelt und wenn ja, ob diese Permutation eine Tour mit Länge $\leq B$ ist. Falls ja, dann hält die Maschine mit Zustand JA, ansonsten mit NEIN. Diese Berechnungen können zeitlich innerhalb von $\mathcal{O}(n^2)$ ausgeführt werden. Die Maschine entscheidet $\mathsf{TSP}(B)$ im nicht-deterministischen Sinne: genau dann, wenn es eine entsprechende Tour gibt, kann die Maschine diese „raten" und die Eingabe akzeptieren. Dabei ist es unerheblich, ob in anderen Berechnungen Touren überprüft werden, die länger als B sind.

Wir können nicht-deterministische Maschinen simulieren, allerdings benötigt man dazu im Allgemeinen exponentiell viel Zeit:

(3.5) Satz: *Eine nicht-deterministische Turing-Maschine N mit Zeitschranke $f(n)$ kann durch eine deterministische 3-Band-Turing-Maschine M mit Zeitschranke $\mathcal{O}(c^{f(n)})$ simuliert werden. Dabei ist $c > 1$ eine Konstante, die nur von N abhängt:*

$$\mathsf{NTIME}(f(n)) \subseteq \bigcup_{c>1} \mathsf{TIME}(c^{f(n)}).$$

Beweis: Sei $N = (K, \Sigma, \Delta, s)$. Der **Grad der Nicht-Determiniertheit** d sei die maximale Anzahl von Verzweigungen:

$$d := \max_{(q,\sigma) \in K \times \Sigma} |\{(p, \rho, d) \mid ((q, \sigma), (p, \rho, d)) \in \Delta\}|.$$

Wir können eine Berechnung der Länge m als Folge aus d^m kodieren. Die Simulation beginnt mit $m = 1$ (M merkt sich m auf dem dritten Band). M generiert auf dem zweiten Band alle Folgen aus d^m und simuliert die kodierte Berechnung auf dem ersten Band (dabei stellt sie auch fest, ob die Folge überhaupt eine Berechnung von N kodiert). Wenn N mit JA halten würde, dann hält M ebenfalls akzeptierend. In den anderen Fällen merkt sich M das

[3] Wir können bei gegebener Folge von Entscheidungen einer nicht-deterministischen Maschine den resultierenden „Pfad der Berechnung" als deterministisch ansehen und die Begriffe Zeit- und Raumverbrauch übertragen.

Verhalten von N auf dem dritten Band (Halten mit NEIN, Halten mit STOPP, nicht Halten). Wenn es keine Fortsetzung der Berechnung gibt, d.h. jede so simulierte Berechnung einer festen Länge m endet mit NEIN oder STOPP, dann hält M zurückweisend. Ansonsten wird m erhöht, der Rest vom dritten Band wird gelöscht und die Simulation beginnt von neuem. Da N durch $f(n)$ zeitlich beschränkt ist, muß auch M halten. M muß im ungünstigen Fall $\Sigma_{m=1}^{f(n)} d^m = \mathcal{O}(d^{f(n)})$ Folgen erzeugen. Das Generieren und Überprüfen kann in einer Zeit von $\mathcal{O}(2^{f(n)})$ realisiert werden. ■

(3.6) Bemerkung: Es gibt noch viele andere Maschinentypen, die aber häufig mit geringem Verlust simuliert werden können. Beispiele:

- Maschinen mit einem zwei-dimensionalem Band (Simulation mit quadratischem Verlust);
- Maschinen, die auf ihrem Band Zeichen löschen und einfügen können (Simulation mit quadratischem Verlust);
- Nicht-deterministischen Maschinen mit k Bändern (Simulation durch nicht-deterministische Maschinen mit zwei Bändern ohne Verlust).

4. Unentscheidbarkeit

Um ein Diagonalargument anwenden zu können, benötigen wir eine Turing-Maschine, die sämtliche möglichen Turing-Maschinen simulieren kann:

(4.1) Definition: Die **universelle Turing-Maschine** U interpretiert ihre Eingabe als Kodierung einer Turing-Maschine M und einer Eingabe x und simuliert dann die Berechnung von M mit Eingabe x. Alle nötigen Informationen werden durch natürliche Zahlen in Binärdarstellung kodiert: $1, \ldots, m$ seien die Symbole von M; $m+1, \ldots, m+k$ die Zustände; 1 kodiert $\triangleright$; der Initialzustand sei $m+1$. Die Zahlen $m+k+1, \ldots, m+k+6$ kodieren $\leftarrow, \rightarrow, -$, STOPP, JA, NEIN. Die Kodierung der Maschine M beginnt mit k und m, gefolgt von der vollständigen Beschreibung des Programms δ in Form von hintereinander geschriebener Paare $((q,\sigma),(p,\rho,d))$ mit natürlichen Zahlen als Einträge (zur Trennung sind weitere Symbole von U notwendig). U besitzt zwei Bänder: das erste Band enthält hintereinander die Kodierungen von M und x (wir schreiben dafür $M;x$). Das zweite Band enthält die aktuelle Konfiguration von M in Form von '$w;q;u$', wobei w die Zeichenkette links vom Cursor bis einschließlich unter dem Cursor, q den aktuellen Zustand und u die Zeichenkette rechts vom Cursor angibt. U schreibt zunächst die Folge '$1;m+1;$' gefolgt von x auf das zweite Band. In jedem Schritt der Berechnung liest U vom zweiten Band den aktuellen Zustand, die Cursorposition und das gelesene Symbol von M, sucht auf dem ersten Band die entsprechende Regel und wendet diese auf dem zweiten Band an. Wenn M hält, dann hält U im gleichen Zustand und modifiziert das zweite Band derart, daß es nur noch die Ausgabe von M enthält.

Jede Maschine und jede Eingabe kann auf diese Weise von U simuliert werden. Dies führt uns zu folgendem Problem:

(4.2) Definition: Gegeben sei eine Turing-Maschine M und eine Eingabe x. Hält M bei Eingabe x? Als Sprache über dem Alphabet von U wird diese Frage als **Halteproblem** H bezeichnet:

$$\mathsf{H} := \{M;x \,|\, U(M;x) \neq \uparrow\}.$$

Wir kommen nun zu dem bereits erwähnten Diagonalargument:

(4.3) Satz: H *ist rekursiv aufzählbar, aber nicht rekursiv.*

Beweis: H ist rekursiv aufzählbar. Wir modifizieren dazu die universelle Turing-Maschine U wie folgt: immer wenn die Maschine hält, dann hält sie im Zustand JA. Diese Maschine hält genau dann, wenn die Eingabe in H liegt, d.h. sie akzeptiert H.

Angenommen, H wäre auch rekursiv. Sei M_{H} eine Maschine, die H entscheidet. Wir definieren eine Maschine D wie folgt: bei Eingabe von M simuliert D die Maschine M_{H} mit Eingabe $M; M$. Da sie das Halteproblem entscheidet, hält M_{H} irgendwann. Falls M_{H} akzeptieren würde, dann bewegt sich D für immer nach rechts. Falls M_{H} zurückweisen würde, dann hält D im Zustand JA.

Wie verhält sich nun U bei Eingabe $D; D$? Falls $U(D; D) = \uparrow$ gilt, dann würde nach Definition von D die Maschine M_{H} die Eingabe $D; D$ akzeptieren. Da aber M_{H} die Sprache H entscheidet, heißt das nach Definition von H gerade $U(D; D) \neq \uparrow$. Widerspruch.

Gilt andererseits $U(D; D) \neq \uparrow$, dann würde M_{H} die Eingabe $D; D$ zurückweisen, also wäre $D; D \notin \mathsf{H}$ und damit $U(D; D) = \uparrow$. Widerspruch.

Also kann H nicht rekursiv sein. ■

(4.4) Bemerkung: Das Problem H ist **vollständig** für die Klasse der rekursiv aufzählbaren Sprachen: wenn wir einen Algorithmus besitzen, um H zu entscheiden, dann erhalten wir auch einen Algorithmus, um jede rekursiv aufzählbare Sprachen zu entscheiden. Denn ist L eine Sprache, die von der Turing-Maschine M akzeptiert wird, dann gilt $x \in L \iff M; x \in \mathsf{H}$.

(4.5) Lemma: *Ist eine Sprache L rekursiv, dann ist auch das Komplement $\bar{L}$ rekursiv.*

Beweis: Vertausche die Zustände JA und NEIN bei einer Turing-Maschine, die L entscheidet. ■

(4.6) Lemma: *Eine Sprache L ist genau dann rekursiv, wenn L und $\bar{L}$ rekursiv aufzählbar sind.*

Beweis: ($\Rightarrow$) Nach Lemma (4.5) sind L und $\bar{L}$ rekursiv, nach Lemma (1.3) sind damit beide rekursiv aufzählbar.

($\Leftarrow$) Seien M und $\bar{M}$ akzeptierende Maschinen für L und $\bar{L}$. Die Turing-Maschine D sei wie folgt definiert: D besitzt zwei Bänder, beide enthalten zu Beginn die Eingabe x. Die Zustände von D sind alle Paare der Zustände von M und $\bar{M}$. D führt abwechselnd auf dem ersten/zweiten Band die Operationen von $M/\bar{M}$ gemäß der ersten/zweiten Koordinate des Zustandes aus. Die andere Koordinate und das andere Band bleiben jeweils unverändert. Da jede Eingabe entweder von M oder von $\bar{M}$ akzeptiert wird, hält eine der beiden Maschinen. D hält dann mit JA oder NEIN, je nachdem, ob M oder $\bar{M}$ angehalten hat. ■

(4.7) Bemerkung: Sei R die Klasse der rekursiven Sprachen, RE die Klasse der rekursiv aufzählbaren Sprachen und coRE die Klasse der Komplemente der rekursiv aufzählbaren Sprachen. Diese Klassen sind gerade die ersten Stufen der **Arithmetischen Hierarchie**: $\mathsf{R} = \Delta_1^0$, $\mathsf{RE} = \Sigma_1^0$ und $\mathsf{coRE} = \Pi_1^0$.

Man beachte, daß schon aus Kardinalitätsgründen der obere Bereich von Abb. 4 nicht leer ist: es gibt überabzählbar viele Sprachen (Teilmengen endlicher Folgen natürlicher Zahlen), aber nur abzählbar viele Turing-Maschinen (jede Turing-Maschine kann wie in Definition (4.1) als endliche Folge natürlicher Zahlen kodiert werden).

Tatsächlich ist jede nicht-triviale Eigenschaft von Turing-Maschinen unentscheidbar:

(4.8) Satz (Rice): *Sei $\mathcal{C}$ eine echte, nicht-leere Teilmenge von* RE. *Dann ist das folgende Problem unentscheidbar: Gilt $L(M) \in \mathcal{C}$ für eine gegebene Turing-Maschine M?*

Beweis: O.E. sei $\emptyset \notin \mathcal{C}$ (ansonsten betrachte $\mathsf{RE} - \mathcal{C}$). Nach Voraussetzung ist $\mathcal{C} \neq \emptyset$. Sei $L \in \mathcal{C}$ und M_L eine Turing-Maschine, die L akzeptiert. Sei M_{H} eine Maschine, die das

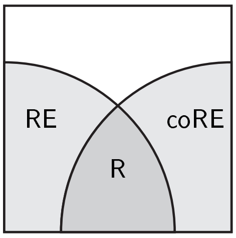

ABB. 4. Klassen von Sprachen

Halteproblem akzeptiert, vgl. Satz (4.3). Für eine beliebige Eingabe x definieren wir die Maschine M_x wie folgt: bei Eingabe y simuliert M_x zunächst M_{H} mit Eingabe x. Falls M_{H} hält, dann simuliert M_x danach die Maschine M_L mit Eingabe y.

BEH.: $x \in \mathsf{H} \iff L(M_x) \in \mathcal{C}$.

Ein Algorithmus, der $\mathcal{C}$ entscheidet, entscheidet also auch H. Aus der Unentscheidbarkeit von H folgt daher die von $\mathcal{C}$.

BEW.: ($\Rightarrow$) Ist $x \in \mathsf{H}$, dann gilt $M_{\mathsf{H}}(x) = \text{JA}$. Also simuliert M_x die Maschine M_L und es gilt $L(M_x) = L(M_L) = L \in \mathcal{C}$ nach Voraussetzung.

($\Leftarrow$) Ist $x \notin \mathsf{H}$, dann gilt $M_{\mathsf{H}} = \uparrow$, also auch $M_x(y) = \uparrow$ für jede Eingabe y und damit $L(M_x) = \emptyset \notin \mathcal{C}$. ■

5. Logik

In diesem Abschnitt betrachten wir Maschinen, die logische Probleme lösen sollen.

I. Boolsche Logik. Wir kümmern uns zunächst um Boolsche Ausdrücke:

(5.1) Definition: Das Problem SAT sei wie folgt definiert: ist ein gegebener Boolscher Ausdruck φ erfüllbar?

Wir erhalten unmittelbar:

(5.2) Lemma: *Es gilt* $\mathsf{SAT} \in \mathsf{TIME}(\mathcal{O}(n^2 \cdot 2^n))$ *und* $\mathsf{SAT} \in \mathsf{NP}$. *Dabei gibt n die Anzahl der Variablen in einem gegebenen Ausdruck φ an.*

Beweis: Eine deterministische Maschine kann hintereinander alle möglichen Belegungen (Zuordnung von Wahrheitswerten) der Variablen ausprobieren. Die Überprüfung, ob eine gegebene Belegung den Ausdruck erfüllt, benötigt höchstens n^2 Schritte.

Eine nicht-deterministische Maschine „tippt“ eine Belegung und überprüft diese. ■

(5.3) Frage: Gilt auch $\mathsf{SAT} \in \mathsf{P}$? Sehr wahrscheinlich nicht, siehe Frage (3.3).

Ein wichtiger Spezialfall von SAT kann in polynomieller Zeit gelöst werden:

(5.4) Definition: Eine **Horn-Klausel** ist eine Klausel mit höchstens einem positiven Literal. Beispiele: $(\neg v_1 \vee v_3)$, $(\neg v_1 \vee \neg v_2 \vee \neg v_3)$ oder (v_1). Horn-Klauseln sind logisch äquivalent zu Implikationen der Form

(1) $$v_{i_1} \wedge \cdots \wedge v_{i_k} \Longrightarrow v_j$$

(2) $$\mathsf{wahr} \Longrightarrow v_j$$

(3) $$v_{i_1} \wedge \cdots \wedge v_{i_k} \Longrightarrow \mathsf{falsch}.$$

Ein **Horn-Ausdruck** ist eine Konjunktion von Horn-Klauseln. Das Problem HORNSAT ist der Spezialfall von SAT für Horn-Ausdrücke.

(5.5) Lemma: *Es gilt* $\mathsf{HORNSAT} \in \mathsf{P}$.

Beweis: Sei φ ein Horn-Ausdruck. Die Maschine konstruiert sukzessive eine Menge T von Variablen. Die Belegung β_T sei definiert durch $\beta_T(v_i) = \mathsf{wahr} \Longleftrightarrow v_i \in T$. Zu Beginn sei $T = \emptyset$. Solange es noch Klauseln in φ der Form (1) oder (2) gibt, die unter der Belegung β_T falsch sind, fügt die Maschine v_j zu T hinzu. Wie man leicht zeigt, gilt $T \subseteq T^*$ für jede Menge T^* von Variablen, so daß β_{T^*} den Ausdruck φ erfüllt. Andererseits gilt: wird φ nicht von β_T erfüllt, dann nur durch eine Klausel der Form (3) mit $v_{i_1}, \ldots, v_{i_k} \in T$. Also ist φ genau dann erfüllbar, wenn β_T eine Belegung ist, die φ erfüllt. ■

II. Logik erster Stufe. Ein wichtiges Gebiet für die Anwendung von Turing-Maschinen ist die Theorie der Graphen. Das zugehörige Vokabular[4] Σ_G enthält keine Funktions- oder Konstantensymbole und neben $=$ nur eine zweistellige Relation G. $G(x, y)$ wird interpretiert als das Bestehen einer Kante zwischen x und y. Für einen Ausdruck φ der Logik erster Stufe sei φ-GRAPH das folgende Problem: gegeben sei ein Graph G und eine Belegung der freien Variablen von φ mit Knoten aus G. Ist G ein Modell von φ? Wir setzten hier eine geeignete Repräsentation von Graphen als Eingabe voraus. Dabei sei n die Anzahl der Knoten von G.

Dieses Problem hat die folgende Komplexität:

(5.6) Satz: *Für jeden Ausdruck φ der Logik erster Stufe über Σ_G gilt φ-$\mathsf{GRAPH} \in \mathsf{P}$.*

Beweis: Durch Induktion über den Aufbau von φ. Atomare Ausdrücke können direkt überprüft werden.

$\varphi \equiv \neg\psi$: Die Maschine simuliert den Algorithmus für ψ-$\mathsf{GRAPH} \in \mathsf{TIME}(p(n))$ und vertauscht die Zustände JA und NEIN. Damit ist auch φ-$\mathsf{GRAPH} \in \mathsf{TIME}(p(n))$.

$\varphi \equiv \psi_1 \vee \psi_2$: Die Maschine simuliert erst den Algorithmus für ψ_1-$\mathsf{GRAPH} \in \mathsf{TIME}(p_1(n))$. Bei JA endet sie selbst mit JA, bei NEIN simuliert sie den Algorithmus für ψ_2-$\mathsf{GRAPH} \in \mathsf{TIME}(p_2(n))$ und endet mit dem gleichen Zustand wie dieser. Also ist φ-$\mathsf{GRAPH} \in \mathsf{TIME}(p_1(n) + p_2(n))$.

$\varphi \equiv \forall x \psi(x)$: Die Maschine simuliert für jeden Knoten x von G den Algorithmus ψ-$\mathsf{GRAPH} \in \mathsf{TIME}(p(n))$. Sie hält genau dann mit JA, wenn alle Instanzen ein JA ergeben haben. Da G genau n Knoten besitzt, gilt φ-$\mathsf{GRAPH} \in \mathsf{TIME}(n \cdot p(n))$. ■

III. Logik zweiter Stufe. Viele bedeutende Probleme der Theorie der Graphen können aber nicht in der ersten Stufe ausgedrückt werden, z.B. REACHABILITY (gibt es einen Pfad zwischen zwei gegebenen Knoten eines Graphen) und HAMILTONPATH (gibt es zu einem Graphen G einen Pfad, der jeden Knoten genau einmal durchläuft). Diese Probleme lassen sich durch Ausdrücke der Logik zweiter Stufe mit einem zweitstufigen Existenzquantor der Form $\varphi \equiv \exists P\, \psi$ ausdrücken, wobei ψ ein Ausdruck erster Stufe über dem Vokabular ist, daß P als neues Relationssymbol enthält. Für derartige Ausdrücke erhalten wir:

(5.7) Satz: *Es gilt φ-$\mathsf{GRAPH} \in \mathsf{NP}$.*

Beweis: Sei $G = (V, E)$ ein Graph mit $|V| = n$. Eine entsprechende nicht-deterministische Turing-Maschine „tippt“ die Relation $P^G \subseteq V^r$. Dabei sei r die Stelligkeit der Relation P. Zur Überprüfung, ob der erststufige Ausdruck $\psi(P^G)$ erfüllt ist, genügt nach Satz (5.6) eine deterministische Maschine mit polynomieller Zeitschranke. Da höchstens n^r Elemente geraten werden müssen, arbeitet die so definierte Maschine in polynomieller Zeit. ■

---

[4] In der Logik üblicherweise als Sprache bezeichnet.