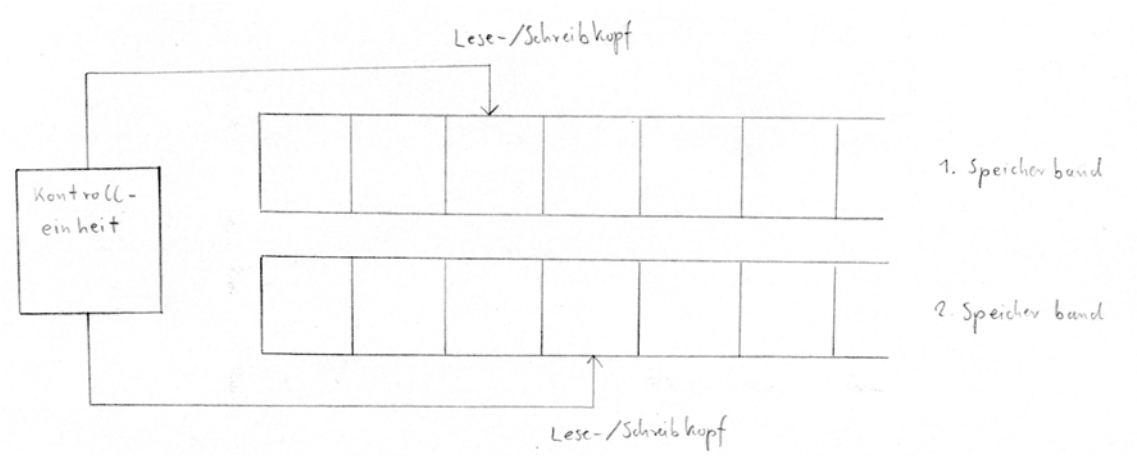
Kapitel 2: Turingmaschine, Komplexitätsklassen (Grundlagen)

Turingmaschine

Um eine präzise Grundlage für die weitere Vorlesung zu schaffen, definieren wir zunächst das Modell der Turingmaschine und einige Komplexitätsklassen.

Eine Turingmaschine ist ein mathematisches Modell einer universellen Rechenmaschine, auf der alle überhaupt von Menschen berechenbaren Probleme berechnet werden können. Dieses Modell ist benannt nach dem englischen Mathematiker Alan Turing, der es 1936 in seiner Schrift „On Computable Numbers, with an Application to the Entscheidungsproblem“[1] vorstellte.

Die Turingmaschine besteht aus einer Kontrolleinheit, die Lese-/Schreibköpfe auf einem oder mehreren Speicherbändern steuert. Wir interessieren uns für Turingmaschinen mit mehr als einem Band, sogenannte k-Band-Turingmaschinen. (Das k steht für die Anzahl der Bänder, entsprechend gibt es 1-Band-, 2-Band-, 3-Band-Turingmaschinen etc.) Mehrband-Turingmaschinen sind für die Komplexitätstheorie ein besseres Modell als 1-Band-Maschinen, da damit die Modellierung der Zeit näher an der Realität ist.



2-Band-Turingmaschine

Definition k-Band-Turingmaschine:

$M = (Q, \Sigma, I, q_0, F)$

In Worten: Eine k-Band-Turingmaschine M ist ein 5-Tupel bestehend aus:

1. Einer endlichen Menge von Zuständen Q. Diese kann man sich vorstellen als interne Zustände der Kontrolleinheit.

2. Einem endlichen Alphabet Σ, welches in jedem Fall ein leeres Symbol □ enthält: □ ∈ Σ. („Blank-Symbol“)

[1] Der deutsche Mathematiker David Hilbert postulierte 1900 eine Liste von 23 ungelösten, mathematischen Problemen, darunter das „Entscheidungsproblem“. Dieses „Hilbertprogramm“ fand international Beachtung in der Mathematik. Daher taucht das Wort im Titel von Turings Abhandlung auf Deutsch auf.

3. Einem Anfangszustand $q_0 \in Q$.

4. Einer Menge von Endzuständen $F \subset Q$.

5. Einer Übergangsrelation I.

Die Übergangsrelation I kann man sich als Tabelle vorstellen, die für jeden Zustand der Maschine und für alle möglicherweise von den Bändern zu lesenden Zeichen auflistet, welche neuen Zeichen geschrieben werden sollen, und wie anschließend die Köpfe verschoben werden sollen (und in welchem neuen Zustand sich die Maschine daraufhin befindet). Die Übergangsrelation ist definiert als:

$I \subseteq Q \times \Sigma^k \times \Sigma^k \times \{l, r, s\}^k \times Q$

Dabei steht Q für die Menge der Zustände, aus der jedes Element der Übergangsrelation einen Anfangs- und ein Endzustand enthält. Σ^k sind Mengen zu lesender bzw. zu schreibender Zeichen (je ein Zeichen pro Band). Die Menge {l, r, s} ist die Menge der möglichen Positionsänderungen der Köpfe, nämlich „links“, „rechts“ und „stehen bleiben“. Pro Band gibt es einen Kopf, daher enthält jedes Element der Übergangsrelation k solche Positionsänderungen.

Ein Element aus der Übergangsrelation I ist also wiederum ein 5-Tupel (q, s, s’, m, q’) bestehend aus:

1. dem aktuellen Zustand q,

2. den Zeichen s, die die Lese-/Schreibköpfe von der derzeitigen Position lesen,

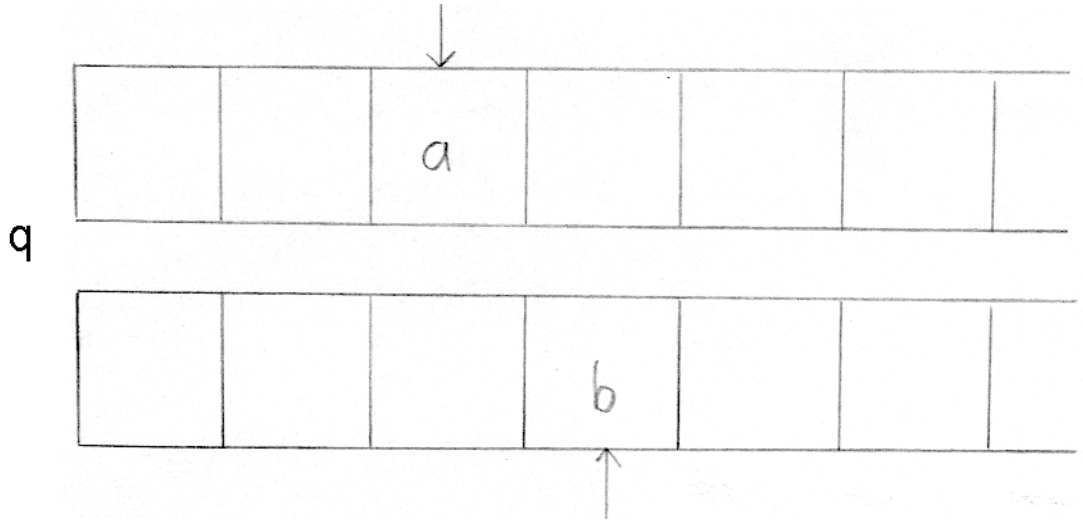
3. den Zeichen s’, die die Lese-/Schreibköpfe an die derzeitige Position schreiben,

4. den Positionsänderungen m der Lese-/Schreibköpfe und

5. dem Folgezustand q’.

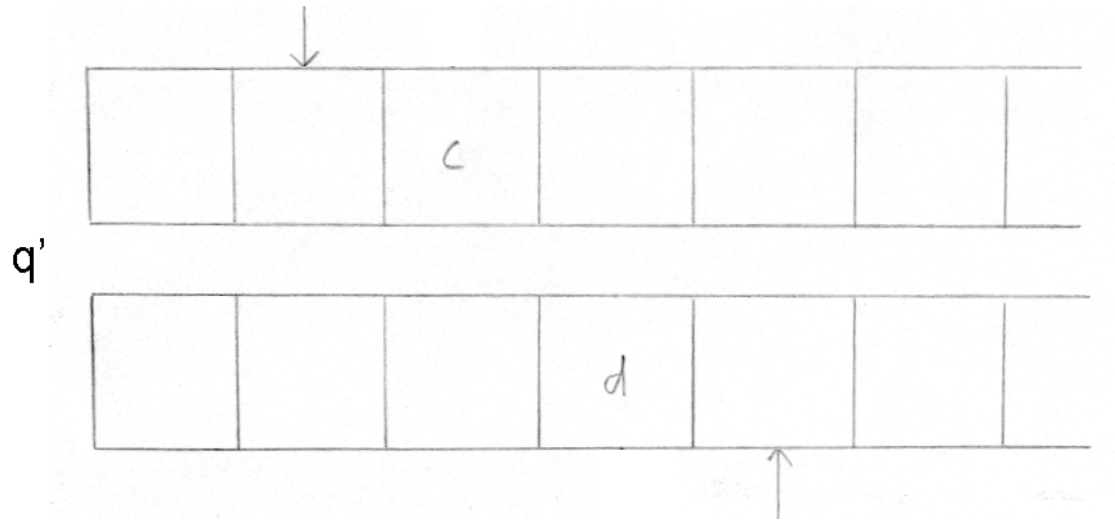
Ein solches Element der Übergangsrelation lässt sich als Anweisung an die Turingmaschine lesen: „Wenn sich die Maschine in Zustand q befindet und die Köpfe die Zeichen s lesen, dann schreibe die Zeichen s’ und verschiebe die Köpfe um m!“ Anschließend befindet sich die Maschine dann im Zustand q’.

Beispiel: Gegeben sei folgende Turingmaschine mit zwei Speicherbändern:



Die Position der Köpfe ist durch die Pfeile markiert. Das erste Speicherband enthält das Zeichen „a“ an der derzeitigen Position seines Kopfes, das zweite das Zeichen „b“. Der Zustand, in dem sich die Maschine im Moment befindet, heiße q.

Ein Beispielelement (q, (a, b), (c, d), (l, r), q’) der Übergangsrelation überführt die Maschine in den neuen Zustand q’, wobei an den derzeitigen Positionen der Köpfe die Zeichen „c“ und „d“ geschrieben werden. Anschließend wird der Kopf des ersten Bandes nach links bewegt, der des zweiten Bandes nach rechts:



Akzeptor-Maschinen

Manche Turingmaschinen sind sogenannte Akzeptor-Maschinen, die ähnlich wie endliche Automaten formale Sprachen akzeptieren. Diese zeichnen sich durch zwei spezielle Zustände namens ACC („accept“) und REJ („reject“) aus, die die einzigen Endzustände sind:

F = { ACC, REJ }

Deterministische Turingmaschinen (DTM)

Eine Turingmaschine heißt deterministisch, wenn für jedes Paar $(q, s) \in Q \times \Sigma^k$ aus aktuellem Zustand und Bandbeschriftung an den aktuellen Kopfpositionen höchstens ein Tupel (q, s, s', m, q') in der Übergangsrelation I existiert.

Vollständige Beschreibung einer Turingmaschine

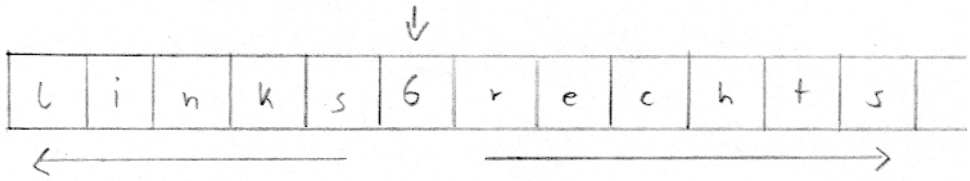
Eine Turingmaschine wird durch ihren globalen Zustand (auch „Konfiguration“ genannt) vollständig beschrieben. Dieser ist ein Tupel:

$(q, b) \in Q \times Band^k$

Dabei ist Band ein Objekt, das den Inhalt eines Bandes und die Position des zugehörigen Kopfes beschreibt. Es beantwortet also die Fragen:

- An welcher Position steht der Kopf im Moment?
- Welches Zeichen steht dort?
- Was steht links von dieser Position?
- Was steht rechts davon?

Beispiel:



Der Kopf in diesem Band steht auf Position 6. An dieser Stelle enthält das Band auch das Zeichen „6“. Links davon steht die Zeichenfolge „links“ auf dem Band, auf der rechten Seite die Zeichenfolge „rechts“.

Als Konvention gilt, dass rechts vom angegebenen Inhalt nur noch leere Symbole vorkommen.

Eine mögliche Definition des Objekts Band ist z.B.:

$Band = \Sigma^* \times \Sigma \times \Sigma^*$

Eine Instanz eines solchen Objektes wäre also ein 3-Tupel (w, o, w’), bestehend aus dem Wort am Anfang des Bandes, dem Zeichen an der Kopfposition und dem Wort am Ende des Bandes.

(Anmerkung zur Implementierung: Bei einem solchen Band-Objekt bietet es sich an, den Bandinhalt links vom Kopf rückwärts geschrieben abzuspeichern. Dann lässt sich bei einer Bewegung des Kopfes um eins nach links einfacher herausfinden, welches Zeichen an der neuen Position steht.)

Eine alternative Definition des Objekts Band wäre:

$Band = \Sigma^* \times N$

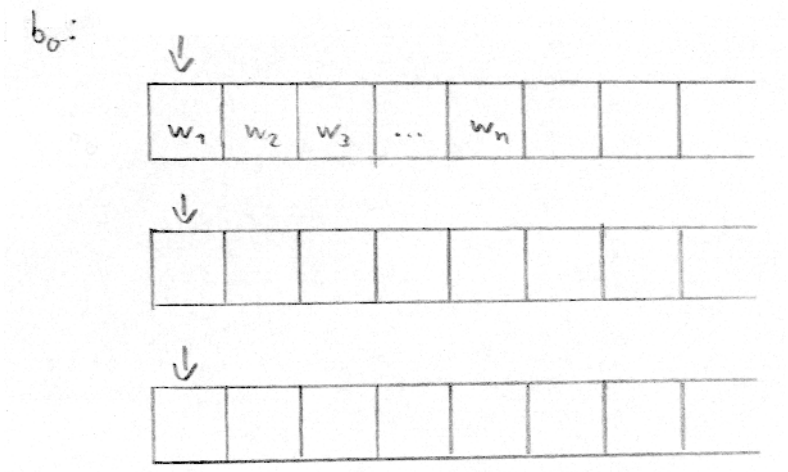
Ein Element aus Band wäre dann ein 2-Tupel (w, n), bestehend aus dem gesamten Bandinhalt w und der Kopfposition n. Dabei gilt naheliegenderweise $n \leq |w|$

Globaler Anfangszustand

Bevor die Maschine anfängt zu rechnen, befindet sie sich im Anfangszustand s_0. Dieser ist abhängig vom Eingabewert $w \in \Sigma^*$, welcher gleich zu Beginn auf dem ersten Band der Maschine gespeichert ist. Der Eingabewert besteht aus einer Folge von Zeichen: $w = w_1 \ldots w_n$.

$s_0(w) = (q_0, b_0)$

b_0 ist ein Bandobjekt, in dem auf dem ersten Band der Eingabewert liegt, während alle anderen Bänder leer sind, und die Köpfe sich am Anfang der Bänder befinden:



Übergangsrelation für globale Zustände

Für globale Zustände gibt es eine Übergangsrelation, die von der bereits bekannten Übergangsrelation I induziert wird und durch einen Pfeil dargestellt wird:

$s \rightarrow s'$

Entsprechend ist Berechnung, die eine DTM bei einer Eingabe w durchführt, eine Folge von globalen Zuständen:

$s_0(w) \rightarrow s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \ldots$

Dass die Turingmaschine deterministisch ist, bedeutet in diesem Zusammenhang, dass die Folge von globalen Zuständen für ein Eingabewort w eindeutig bestimmt ist. (Sie kann dabei endlich oder unendlich sein.)

Für die Vorlesung gilt die Konvention, dass eine endliche DTM sich nach der Berechnung stets im Endzustand befindet, bei einer Akzeptor-Maschine also in einem der Zustände ACC oder REJ. Durch diese Konvention sind Deadlocks ausgeschlossen.

Man sagt, eine deterministische Akzeptor-Maschine M akzeptiert oder „erkennt“ ein Wort w $\in \Sigma^*$, wenn die Berechnung von M auf w in ACC endet. Damit kann man auch eine Sprache L zu einer Maschine M definieren:

L(M) = {w, M akzeptiert w}

Ein Beispiel

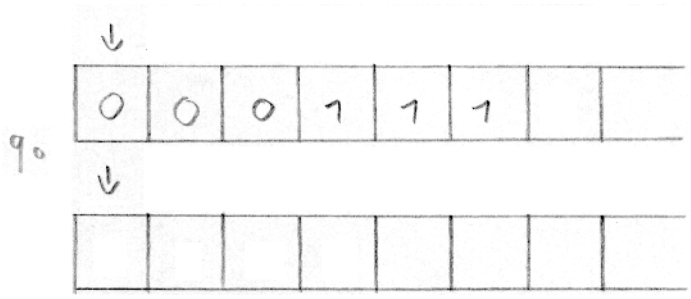
Als Beispiel soll eine Maschine für die Sprache der Wörter aus n Nullen gefolgt von ebenso vielen Einsen (0^n1^n) dienen. Diese habe zwei Speicherbänder (k = 2), die Zustandsmenge $Q = \{q_0, q_1, q_2, q_3, q_4\}$ mit q_1 = ACCEPT, q_2 = REJECT und ein Alphabet $\Sigma = \{0, 1, \#, \square\}$. Der Anfangszustand sei q_0, die Endzustände seien {ACCEPT, REJECT}.

Die Übergangsrelation ist in folgender Tabelle gegeben:

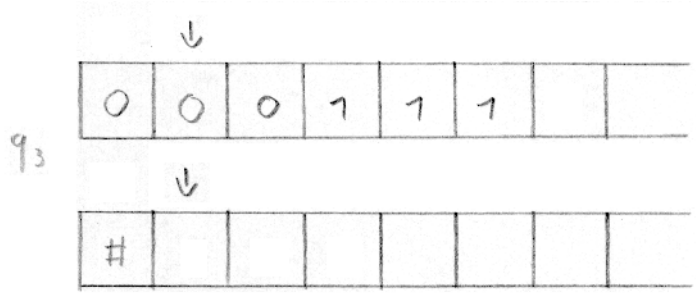
Nr.	**q**	**s**	**s’**	**m**	**q’**
1	q_0	(□, □)	(#, #)	(s, s)	q_1
2	q_0	(1, □)	(1, #)	(s, s)	q_2
3	q_0	(0, □)	(0, #)	(r, r)	q_3
4	q_3	(0, □)	(0, 0)	(r, r)	q_3
5	q_3	(□, □)	(#, #)	(s, s)	q_2
6	q_3	(1, □)	(1, #)	(r, l)	q_4
7	q_4	(0, 0)	(0, 0)	(s, s)	q_2
8	q_4	(0, #)	(0, #)	(s, s)	q_2
9	q_4	(1, 0)	(1, 0)	(r, l)	q_4
10	q_4	(1, #)	(1, #)	(s, s)	q_2
11	q_4	(□, 0)	(#, 0)	(s, s)	q_2
12	q_4	(□, #)	(#, #)	(s, s)	q_1

Als Eingabewert sei auf dem ersten Band 000111 gespeichert.

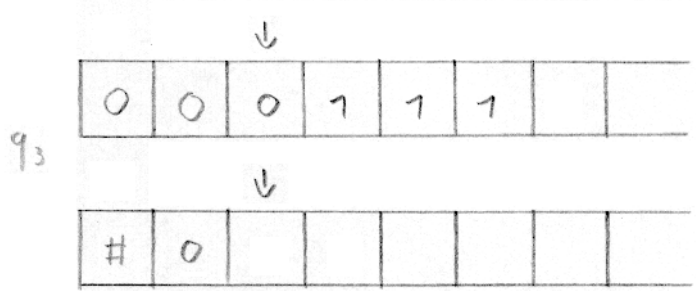
Der Ablauf ab dem Anfangszustand ist dann wie folgt:



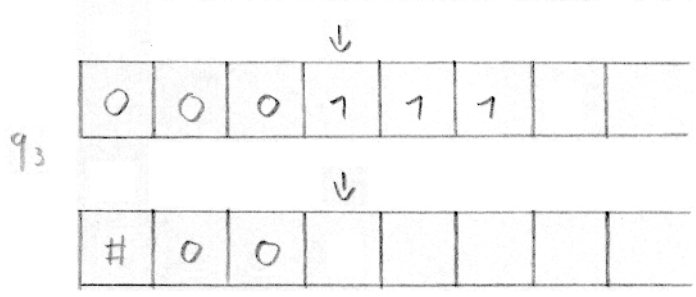
(1) Anfangszustand



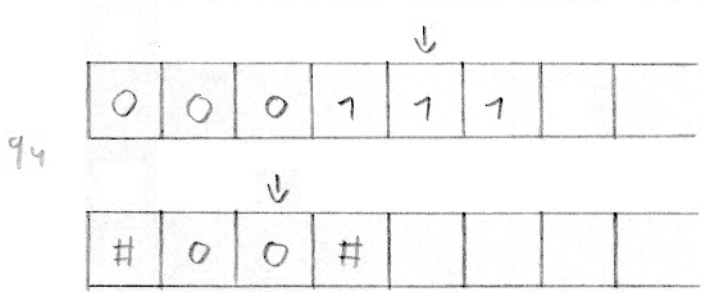
(2)



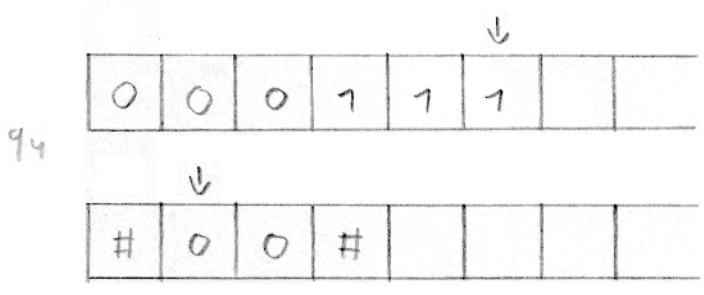
(3)



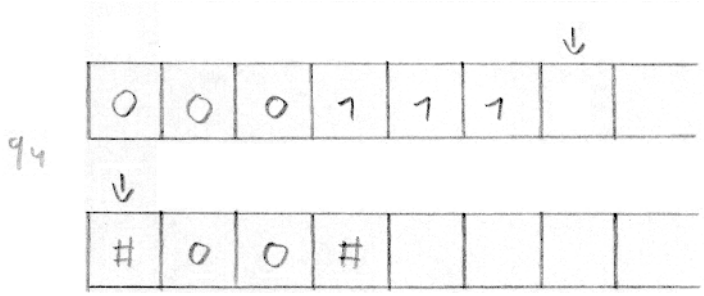
(4)



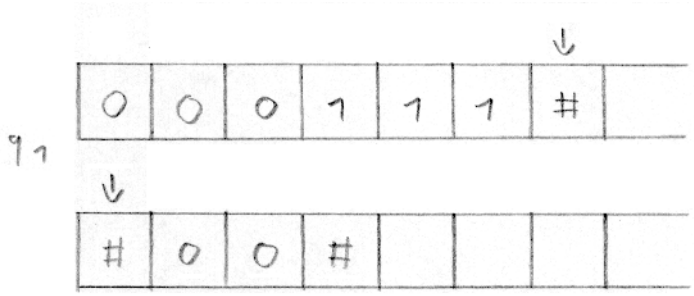
(5)



(6)



(7)



(8) Endzustand

Element Nr. 3 aus der Übergangstabelle beschreibt gewissermaßen eine while-Schleife:

```
while(q3 AND (0, □)) {
     write(0, 0);
     move(r, r);
}
```

(Siehe Bilder (2) und (3))

Komplexitätsklassen

Als nächstes werden wir einige deterministische, von der Zeit abhängige Komplexitätsklassen von Problemen definieren. Als Maß für die Zeit dient dabei die Länge der Berechnungsfolge des Problems auf einer DTM.

Gegeben sei also eine deterministische Turingmaschine T mit einem Eingabewort $w \in \Sigma^*$. Für die Länge der Berechnung auf T, also die Länge der eindeutig bestimmten Folge der globalen Zustände, schreibt man dann: $TIME_T(w) \in \mathbf{N} \cup \{\infty\}$.[2]

Als erste Komplexitätsklasse definieren wir die Klasse der deterministisch in Polynomialzeit entscheidbaren Probleme P:

$P := \{L \in \Sigma^*, \exists \text{ DTM } T: k \in \mathbf{N} \text{ mit } L(T) = L \text{ und } \forall w\ TIME_T(w) = O(|w|^k)\}$

Der Term am Ende der Definition beschreibt den maximalen Zeitaufwand.[3]

---

[2] (fett gesetztes N) **N** := Menge der natürlichen Zahlen

[3] Die „O-Notation“ wird u.a. in der deutschen Wikipedia unter dem Stichwort „Landau-Symbole“ erklärt.

In Worten: P ist die Menge aller Sprachen L, für die eine deterministische Turingmaschine T existiert, so dass L die Sprache der Turingmaschine ist und für alle Eingabewörter w die Länge der Berechnung höchstens $|w|^{k \in \mathbf{N}}$ („polynomial“) ist.

Informell könnte man sagen, P erkennt Wörter w und hält nach höchstens $|w|^k$ Schritt an.

Die natürliche Zahl k nennt man den „Grad“. (Hat nichts mit der Anzahl der Bänder k einer Turingmaschine aus dem vorhergehenden Abschnitt zu tun!)

Weitere Komplexitätsklassen sind z.B.:

- Die Klasse der deterministisch in exponentieller Zeit entscheidbaren Probleme:

  $E := \{L \in \Sigma^*, \exists \text{ DTM } T: k \in \mathbf{N} \text{ mit } L(T) = L \text{ und } \forall w \; TIME_T(w) = O(2^{k|w|})\}$

  (E wird gelegentlich auch ETIME oder EXP genannt, ist aber nicht dasselbe wie die folgende, EXP genannte Klasse.)

- Die Klasse der deterministisch in polynomial-exponentieller Zeit entscheidbaren Probleme:

  $EXP := \{L \in \Sigma^*, \exists \text{ DTM } T: k \in \mathbf{N} \text{ mit } L(T) = L \text{ und } \forall w \; TIME_T(w) = O(2^{|w|\hat{}k})\}$

  (EXP wird gelegentlich auch EXPTIME genannt)

- Die Klasse der deterministisch in doppelt exponentieller Zeit lösbaren Probleme:

  $2E := \{L \in \Sigma^*, \exists \text{ DTM } T: k \in \mathbf{N} \text{ mit } L(T) = L \text{ und } \forall w \; TIME_T(w) = O(2^{2\hat{}|w|k})\}$

Die genannten Klassen unterscheiden sich nur in den am Ende der Definitionen gegebenen, zeitbeschränkenden Funktionen. Da diese von P bis 2E immer größer werden, gilt naheliegenderweise:

$P \subseteq E \subseteq EXP \subseteq 2E$

Die gegebenen Definitionen beschreiben eigentlich Mengen von Sprachen über einem endlichen Alphabet. Modelliert werden in der Komplexitätstheorie aber Entscheidungsprobleme, bei denen die Eingabe beispielsweise Graphen oder ähnliche Objekte sind. Andere Probleme, bei denen mehr als ein Bit ausgegeben wird, lassen sich in den meisten Fällen auf die Komplexität von Entscheidungsproblemen zurückführen.

Die definierten Klassen sind robust, d.h. sie ändern sich nicht bei Variation des zu Grunde liegenden Maschinenmodells. Wenn ein Modell nur sequentielle, deterministische Berechnungen ermöglicht, ist die Klasse der darauf in polynomieller Zeit lösbaren Probleme gleich der oben definierten Klasse P. Dies kann man zeigen, indem man das Modell auf einer Turingmaschine simuliert. Ausnahmen sind „unrealistische“ Modelle wie das Random-Access-Modell, das Multiplikation in einem Berechnungsschritt erlaubt.

Übersetzer-Maschinen

Übersetzer- oder Transducer-Maschinen sind deterministische Turingmaschinen mit folgenden Eigenschaften:

- Es gibt nur einen Endzustand q_F ($F = \{q_F\}$). Jede endliche Berechnung endet per Konvention in q_F. Der globale Endzustand ist (q_F, b), wobei in b alle Köpfe ganz links stehen.

- Der Output steht auf einem ausgezeichneten Ausgabeband.

- Übersetzermaschinen berechnen für alle Wörter $w \in \Sigma^*$ eine Funktion $f : \Sigma^* \to \Sigma^*$
  Informell: Wenn der Eingabewert ein gültiges Wort w ist, dann ist nach der Berechnung der Wert auf dem Ausgabeband f(w).

Komplexitätsklassen von Funktionen

Analog zu den Komplexitätsklassen von Sprachen kann man auch solche für Funktionen definieren, z.B. die Klasse der in polynomialer Zeit berechenbaren Funktionen FP:

$FP := \{f : \Sigma^* \to \Sigma^* : \exists \text{ DTM T mit: T berechnet f wobei } \forall w \text{ gilt: } TIME_T(w) = O(|w|^k)\}$

Insbesondere gilt hier auch, dass der in polynomialer Zeit berechnete Output natürlich selbst nur polynomial groß sein kann. Daran lassen sich gegebenenfalls auch Funktionen erkennen, die nicht zu FP gehören.

Analog zu den Komplexitätsklassen für Sprachen lassen sich auch weitere Komplexitätsklassen für Funktionen definieren, z.B. FE.

Diskussion

P und FP sind interessant, weil es die anerkannt „effizient“ berechenbaren Klassen von Problemen sind. Die dahinterstehende Vorstellung ist, dass Probleme „effizient“ lösbar bzw. Funktionen „effizient“ berechenbar sind, wenn dies in polynomialer Zeit möglich ist. Ein Vorteil der definierten Komplexitätsklassen ist zudem, dass die Definitionen mathematisch präzise sind und auf einem robusten Konzept beruhen.

Trotzdem ist die Beschreibung als effizient relativ zu sehen: Eine Funktion, die in $O(n^{23507})$ berechenbar ist, ist nicht wirklich „effizient“ zu berechnen. Andererseits wächst die Berechnungsdauer einer in $O(1.000053^n)$ berechenbaren Funktion selbst für große n moderat. In der Praxis vorkommende, „natürliche“ Probleme in P sind typischerweise in $O(n^3)$, $O(n^4)$ berechenbar. Die Frage, ob eine gegebene Zahl Primzahl ist, lässt sich beispielsweise in $O(n^{12})$ beantworten. Auch sind in der Praxis effizient lösbare Probleme meist in polynomialer Zeit lösbar, so dass sie in die Klasse P fallen. (Andererseits gibt es auch wieder Probleme, für die es bekanntermaßen einen Algorithmus mit polynomialem Zeitaufwand gibt, der aber selbst unbekannt ist ...)