Inhaltsverzeichnis

Ein- und Ausgabebefehle:

`LOAD` i ,	$i \in \mathbb{N}$	$b' = b + 1$	$c'_0 = c_i$
`ILOAD` i ,	$i \in \mathbb{N}$	$b' = b + 1$	$c'_0 = c_{c_i}$
`CLOAD` i ,	$i \in \mathbb{N}_0$	$b' = b + 1$	$c'_0 = i$
`STORE` i ,	$i \in \mathbb{N}$	$b' = b + 1$	$c'_i = c_0$
`ISTORE` i ,	$i \in \mathbb{N}$	$b' = b + 1$	$c'_{c_i} = c_0$

Arithmetische Befehle:

`ADD` i ,	$i \in \mathbb{N}$	$b' = b + 1$	$c'_0 = c_0 + c_i$
`CADD` i ,	$i \in \mathbb{N}$	$b' = b + 1$	$c'_0 = c_0 + i$
`SUB` i ,	$i \in \mathbb{N}$	$b' = b + 1$	$c'_0 = \begin{cases} c_0 - c_i & \textit{für } c_0 \geq c_i \\ 0 & \textit{sonst} \end{cases}$
`CSUB` i ,	$i \in \mathbb{N}$	$b' = b + 1$	$c'_0 = \begin{cases} c_0 - i & \textit{für } c_0 \geq i \\ 0 & \textit{sonst} \end{cases}$
`MULT` i ,	$i \in \mathbb{N}$	$b' = b + 1$	$c'_0 = c_0 * c_i$
`CMULT` i ,	$i \in \mathbb{N}$	$b' = b + 1$	$c'_0 = c_0 + i$
`DIV` i ,	$i \in \mathbb{N}$	$b' = b + 1$	$c'_0 = \lfloor c_0 / c_i \rfloor$
`CDIV` i ,	$i \in \mathbb{N}$	$b' = b + 1$	$c'_0 = \lfloor c_0 / i \rfloor$

Sprungbefehle:

`GOTO` i ,	$i \in \mathbb{N}$	$b' = i$
`IF` $c_0 = 0$ `GOTO` i ,		$b' = \begin{cases} i & \textit{falls } c_0 = 0 \\ b + 1 & \textit{sonst} \end{cases}$

Stoppbefehl:

`END`

Eine Registermaschine lässt sich entsprechend Abb. 1.4 veranschaulichen.
Bei den Eingabebefehlen `LOAD` i bzw. `CLOAD` i wird der Wert des i-ten Registers bzw. die Zahl i in den Akkumulator geladen; bei `STORE` i wird der Wert des Akkumulators in das i-te Speicherregister eingetragen. Es sei j der Inhalt des i-ten Registers (d.h. $c_i = j$); dann werden durch die Befehle `ILOAD` i bzw. `ISTORE` i mit indirekter Adressierung der Inhalt des Registers j in den Akkumulator geladen bzw. der Inhalt des Akkumulators in das j-te Register gespeichert.
Bei den Befehlen `ADD` i, `SUB` i, `MULT` i und `DIV` i erfolgt eine Addition, Subtraktion, Multiplikation und Division des Wertes des Akkumulators mit dem Wert des i-ten Speicherregisters. Da die Operationen nicht aus dem Bereich der natürlichen Zahlen herausführen sollen, wird die Subtraktion nur dann wirklich ausgeführt, wenn der Subtrahend nicht kleiner als der Minuend ist und sonst 0 ausgegeben; analog erfolgt die Division nur ganzzahlig.
Die Befehle `CADD` i, `CSUB` i, `CMULT` i und `CDIV` i arbeiten analog, nur dass anstelle des Wertes des i-ten Registers die natürliche Zahl i benutzt wird. Dadurch werden auch arithmetische Operationen mit Konstanten möglich.
In all diesen Fällen wird der Wert des Befehlsregisters um 1 erhöht, d.h. der nächste Befehl des Programms wird abgearbeitet. Dies ist bei den Sprungbefehlen grundsätzlich anders.

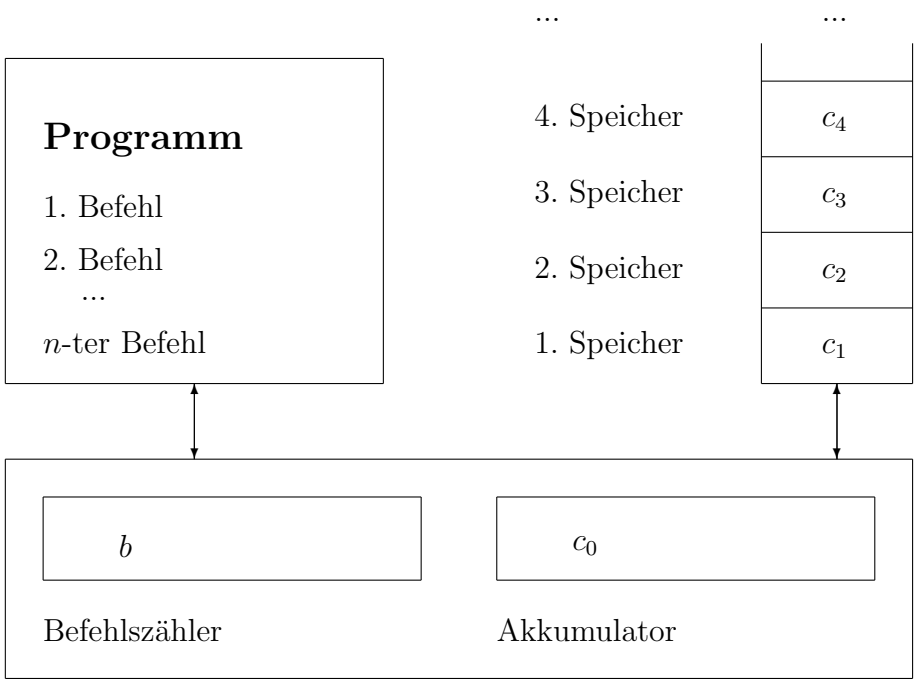

Abbildung 1.4: Registermaschine

Bei `GOTO` i wird als nächster Befehl der i-te Befehl des Programms festgelegt, während bei der `IF`-Anweisung in Abhängigkeit von dem Erfülltsein der Bedingung $c_0 = 0$ der nächste Befehl der i-te bzw. der im Programm auf die `IF`-Anweisung folgende Befehl des Programms ist.
Der Befehl `END` ist ein Stoppbefehl.

Definition 1.9 *Es sei M eine Registermaschine wie in Definition 1.8. Die von M induzierte Funktion $f_M : \mathbb{N}_0^n \longrightarrow \mathbb{N}_0$ ist wie folgt definiert: $f(x_1, x_2, \ldots x_n) = y$ gilt genau dann, wenn M ausgehend von der Konfiguration $(1, 0, x_1, x_2, \ldots, x_n, 0, 0, \ldots)$ die Konfiguration $(b, c_0, y, c_2, c_3, \ldots)$ für gewisse $b, c_0, c_2, c_3, \ldots$ erreicht und der b-te Befehl des Programm* `END` *ist.*

Entsprechend dieser Definition gehen wir davon aus, dass zu Beginn der Arbeit der Registermaschine die ersten n Speicherregister die Werte $x_1, x_2, \ldots, x_n$ und der Akkumulator und alle anderen Speicherregister den Wert 0 enthalten, die Abarbeitung des Programms mit dem ersten Befehl begonnen wird und bei Erreichen eines `END`-Befehls beendet wird und dann das Ergebnis y im ersten Speicherregister abgelegt ist (die Inhalte der anderen Register interessieren nicht).

Wir geben nun drei Beispiele.

Beispiel 1.5 Wir betrachten die Registermaschine M_1 mit dem Programm aus Abb. 1.5.

```
 1  CLOAD 1
 2  STORE 3
 3  LOAD 2
 4  IF c_0 = 0 GOTO 12
 5  LOAD 3
 6  MULT 1
 7  STORE 3
 8  LOAD 2
 9  CSUB 1
10  STORE 2
11  GOTO 4
12  LOAD 3
13  STORE 1
14  END
```

Abbildung 1.5: Programm der Registermaschine aus Beispiel 1.5

Das Programm geht davon aus, dass im ersten und zweiten Register Werte stehen (Befehle 3 bzw. 6), sodass wir davon ausgehen, daá M_1 eine zweistellige Funktion $f_{M_1}(x, y)$ berechnet, wobei x und y zu Beginn im ersten bzw. zweiten Speicherregister stehen.
M_1 verhält sich wie folgt: Mittels der ersten zwei Befehle wird der Wert 1 in das dritte Register geschrieben. Die Befehle 4 – 11 bilden eine Schleife, die sooft durchlaufen wird, wie y angibt, denn bei jedem Durchlauf wird y um 1 verringert (Befehle 8 – 10). Ferner erfolgt bei jedem Durchlauf der Schleife eine Multiplikation des Inhalts des dritten Registers mit x (Befehle 5 – 7). Abschließend wird der Inhalt des dritten Registers in das erste umgespeichert, weil dort nach Definition das Ergebnis zu finden ist. Folglich induziert diese Registermaschine die Funktion

$$f_{M_1}(x, y) = 1 \cdot \underbrace{x \cdot x \cdot \ldots \cdot x}_{y \ mal} = x^y.$$

M_1 berechnet also die Potenzfunktion.

Beispiel 1.6 Wir betrachten die Registermaschine M_2 mit dem Programm aus Abb. 1.6 und zu Beginn der Arbeit stehe nur im ersten Register ein möglicherweise von Null verschiedener Wert (in den anderen Registern steht also eine Null).
Steht im ersten Register eine Null, so werden wegen des zweiten Befehl die Befehle 12–14 abgearbeitet durch die die Ausgabe 0 erzeugt wird. Anderenfalls erfolgt ein Durchlaufen der Befehle 3–5, durch die der Inhalt des Registers 2 um ein erhöht wird, und anschließend der Befehle 6–7, durch die eine Addition der Werte der Register 2 (nach der Erhöhung) und 3 erfolgt, deren Resultat wieder in Register 3 abgelegt wird. Danach wird der Wert des Registers 1 um 1 erniedrigt. Diese Befehle werden solange durchgeführt, wie in Register 1 keine 0 steht, d.h. diese Schleife wird n mal durchlaufen, wenn n zu Beginn in Register 1 steht, da dieses Register bei jedem Durchlauf um 1 verringert wird. In Register 2 stehen während der Durchläufe nacheinander die Zahlen 1, 2, $\ldots n$, die in Register 3 aufaddiert

```
 1  LOAD 1
 2  IF c_0 = 0 GOTO 12
 3  LOAD 2
 4  CADD 1
 5  STORE 2
 6  ADD 3
 7  STORE 3
 8  LOAD 1
 9  CSUB 1
10  STORE 1
11  GOTO 1
12  LOAD 3
13  STORE 1
14  END
```

Abbildung 1.6: Programm der Registermaschine aus Beispiel 1.6

werden. Da der Inhalt des dritten Registers das Resultat liefert, erhalten wir

$$f_{M_2}(n) = \sum_{i=1}^{n} i\,.$$

Beispiel 1.7 Wir gehen jetzt umgekehrt vor. Wir geben uns eine Funktion vor und wollen eine Registermaschine konstruieren, die diese Funktion induziert. Dazu betrachten wir die auf einem Feld (oder Vektor) $(x_1, x_2, \ldots, x_n)$ und seiner Länge n durch

$$f(n, x_1, x_2, \ldots, x_n) = \begin{cases} \sum_{i=1}^{n} x_i & n \geq 1 \\ 0 & n = 0 \end{cases} \tag{1.1}$$

definierte Funktion.
Wir konstruieren eine Registermaschine, bei der zu Beginn n im ersten Register, x_i im $i+1$-ten Register und 0 in allen anderen Registern steht. Die Addition der Elemente des Feldes realisieren wir, indem wir zum Inhalt des zweiten Registers der Reihe nach die Werte $x_n, x_{n-1}, \ldots, x_2$ aus den Registern $n+1, n, \ldots, 3$ addieren. Hierbei greifen wir durch indirekte Adressierung immer auf das entsprechende i-te Register zu, indem wir das erste Register zuerst auf $n+1$ setzen und dann bei jeder Addition um 1 verringern. Die Addition erfolgt solange, wie die Registernummer (im ersten Register), auf die wir zugreifen wollen, mindestens 3 ist. Für diesen ganzen Prozess konstruieren wir eine Schleife.
Die beiden Sonderfälle, $n = 0$ und $n = 1$ (bei denen eigentlich keine Addition erfolgt) lassen sich einfach dadurch realisieren, dass der Inhalt des zweiten Registers (0 bei $n = 0$ und x_1 bei $n = 1$) direkt in das Ergebnisregister 1 umgespeichert wird. Diese beiden Fällen werden wir außerhalb der Schleife vorab erledigen.
Abschließend speichern wir das Ergebnis, das in jedem Fall im zweiten Register steht in das erste Register um.
Formal ergibt dies das Programm aus Abb. 1.7.

```
 1  LOAD 1
 2  CSUB 1              Befehle 1–3 testen, ob Sonderfall vorliegt
 3  IF c_0 = 0 GOTO 15
 4  LOAD 1
 5  CADD 1              Befehle 4–5 setzen Registernummer für Addition auf n + 1
 6  STORE 1
 7  ILOAD 1
 8  ADD 2               Befehle 7–9 addieren c_{i+1} = x_i, n ≥ i ≥ 2, zu c_2
 9  STORE 2
10  LOAD 1
11  CSUB 3              Befehle 10–12 testen, ob c_3 = x_2 schon addiert wurde
12  IF c_0 = 0 GOTO 15
13  CADD 2              Verringern der Registernummer i + 1 um 1
14  GOTO 6
15  LOAD 2
16  STORE 1             Befehle 15 und 16 speichern das Ergebnis in das erste Register
17  END
```

Abbildung 1.7: Programm einer Registermaschine zur Berechnung von (1.1)

Wir wollen nun zeigen, dass Registermaschinen die Funktionen, die von **LOOP/WHILE**-Programmen induziert werden, berechnen können.

Satz 1.8 *Zu jedem* **LOOP/WHILE**-*Programm* Π *gibt es eine Registermaschine* M *derart, dass* $f_M = \Phi_{\Pi,1}$ *gilt.*

Beweis. Wir beweisen den Satz mittels Induktion über die Tiefe der **LOOP/WHILE**-Programme.
Induktionsanfang $k = 0$. Dann ist die gegebene Funktion eine der Wertzuweisungen.
Ist $x_i := 0$ die Anweisung, so liefert die Registermaschine mit dem Programm

```
1  CLOAD 0
2  STORE i
3  END
```

bereits das gewünschte Verhalten.
Ist $x_i := S(x_j)$, so leistet die Registermaschine mit dem Programm

```
1  LOAD j
2  CADD 1
3  STORE i
4  END
```

die gewünschte Simulation.
Für $x_i := P(x_j)$ und $x_i := x_j$ geben wir analoge Konstruktionen.
Induktionsschritt von $< k$ *auf* k. Wir haben zwei Fälle zu unterscheiden, nämlich ob als letzte Operation beim Aufbau der Programme eine Hintereinanderausführung oder eine

WHILE-Schleife angewendet wurde (auf die Betrachtung der **LOOP**-Schleife können wir wegen der Bemerkung am Ende des Abschnitts 1.1.1 verzichten).
Hintereinanderausführung. Es sei $\Pi = \Pi_1; \Pi_2$, und für $i \in \{1, 2\}$ sei M_i die nach Induktionsvoraussetzung existierende Registermaschine mit $f_{M_i} = \Phi_{\Pi_i,1}$. Ferner habe M_i das Programm P_i, das aus r_i Befehlen bestehen möge. Weiterhin bezeichnen wir mit $p_{j,i}$ den j-ten Befehl von P_i. Ohne Beschränkung der Allgemeinheit nehmen wir an, dass jedes der Programme P_i nur einen `END`-Befehl enthält, der am Ende des Programms steht. Wir modifizieren nun die Befehle des Programms P_2 dahingehend, dass wir alle Befehlsnummer j in einem Sprungbefehl oder einem bedingten Befehl durch $j + r_1 - 1$ ersetzen. Dadurch entstehe q_j aus $p_{j,2}$ für $1 \leq j \leq r_2$. Dann berechnet die Registermaschine mit dem Programm

1	$p_{1,1}$
2	$p_{2,1}$
...	...
$r_1 - 1$	$p_{r_1-1,1}$
r_1	q_1
$r_1 + 1$	q_2
...	...
$r_1 + r_2 - 1$	q_{r_2}

die Funktion $\Phi_{\Pi,1}$.
WHILE-*Schleife* Es sei $\Pi' =$ **WHILE** $x_i \neq 0$ **BEGIN** Π **END**. Außerdem seien $p_1, p_2, \ldots p_r$ die Befehle einer Registermaschine M mit $f_M = \Phi_{\Pi,1}$, wobei wiederum $p_r =$ `END` gelte. Für $1 \leq i \leq m$ sei q_i der Befehl, der aus p_i entsteht, indem jede in ihm vorkommende Befehlsnummern um 2 erhöht werden. Dann berechnet das Programm

1	`LOAD`i
2	`IF` $c_0 = 0$ `GOTO` $r + 3$
3	q_1
4	q_2
...	...
$r + 1$	q_{r-1}
$r + 2$	`GOTO`1
$r + 3$	`END`

die von Π' indizierte Funktion. □

1.1.4 TURING-Maschinen

Die Registermaschine stellt eine Modellierung des realen Rechners dar. Sie ist aber hinsichtlich der folgenden Aspekte ein relativ komplexes Modell.

- Die verwendeten Operationen Addition, Multiplikation usw. sind nicht so elementar sind wie die Operationen, die wir z.B. bei **LOOP**/**WHILE**-Programmen oder partiell-rekursiven Funktionen als Basisoperationen benutzt haben.
- Der Abstand zwischen den Registern, die von einem Lade- oder Speicherbefehl betroffen sind, kann groß sein.
- Die in den Registern enthaltenen Objekte sind natürliche Zahlen, d.h. Folgen von Ziffern, und daher komplexer als einfache Ziffern oder Symbole.

Wir wollen nun eine Formalisierung des Berechenbarkeitsbegriffs auf der Basis einer TURING-Maschine[1] geben, die bezüglich der obigen Aspekte einfacher ist. In den Zellen, die den Registern entsprechen, werden nur Ziffern bzw. Symbole aus einer endlichen Menge gespeichert. Die im wesentlichen einzige Operation besteht im Ändern des Inhalts einer Zelle, d.h. im Ersetzen einer Ziffer durch eine andere. Ferner kann nach Änderung des Inhalts einer Zelle nur zu den beiden benachbarten Zellen gegangen werden.
Eine Zahl wird dann als Folge von Ziffern, die in aufeinanderfolgenden Zellen stehen, interpretiert. Will man mehrere Zahlen betrachten, so ist es erforderlich, diese durch spezielle Trennsymbole voneinander zu trennen. Daher kann der Grundbereich der Symbole nicht nur aus Ziffern bestehen. Es ist somit sinnvoll beliebige endliche Mengen als Grundbereiche zuzulassen.
Die folgenden Begriffe dienen dazu, um für diesen Fall die notwendige Terminologie bereitzustellen.
Unter einem Alphabet verstehen wir eine endliche nichtleere Menge. Die Elemente eines Alphabets heißen Buchstaben. Endliche Folgen von Buchstaben des Alphabets V nennen wir Wörter über V; Wörter werden durch einfaches Hintereinanderschreiben der Buchstaben angegeben. Unter der Länge $|w|$ eines Wortes w verstehen wir die Anzahl der in w vorkommenden Buchstaben, wobei jeder Buchstabe sooft gezählt wird, wie er in w vorkommt. λ bezeichnet das Leerwort, das der leeren Folge entspricht, also aus keinem Buchstaben besteht und die Länge 0 hat. Mit V^* bezeichnen wir die Menge aller Wörter über V (einschließlich λ) und setzen $V^+ = V^* \setminus \{\lambda\}$.
In V^* definieren wir ein Produkt w_1w_2 der Wörter w_1 und w_1 durch einfaches Hintereinanderschreiben. Für alle Wörter $w, w_1, w_2, w_3 \in V^*$ gelten dann folgende Beziehungen:

$$\begin{aligned} w_1(w_2w_3) &= (w_1w_2)w_3 = w_1w_2w_3 \quad \text{(Assoziativgesetz)}, \\ w\lambda &= \lambda w, \\ |w_1w_2| &= |w_1| + |w_2|. \end{aligned}$$

Dagegen gilt im allgemeinen $w_1w_2 \neq w_2w_1$ (entsprechend der Definition von Wörtern als Folgen müssen w_1w_2 und w_2w_1 als Folgen gleich sein, was z.B. für $w_1 = ab$, $w_2 = ba$ und damit $w_1w_2 = abba$, $w_2w_1 = baab$ nicht gegeben ist).
Wir geben nun die formale Definition einer TURING-Maschine.

Definition 1.10 *Eine* TURING*-Maschine ist ein Quintupel*

$$M = (X, Z, z_0, Q, \delta),$$

[1] ALAN TURING (1912-1953), englischer Mathematiker

wobei
- X und Z Alphabete sind,
- $z_0 \in Z$ und $\emptyset \subseteq Q \subseteq Z$ gelten,
- δ eine Funktion von $(Z \setminus Q) \times (X \cup \{*\})$ in $Z \times (X \cup \{*\}) \times \{R, L, N\}$ ist, und $* \notin X$ gilt.

Um den Begriff „Maschine“ zu rechtfertigen, geben wir folgende Interpretation. Eine TURING-Maschine besteht aus einem beidseitig unendlichen, in Zellen unterteilten Band und einem „Rechenwerk“ mit einem Lese-/Schreibkopf. In jeder Zelle des Bandes steht entweder ein Element aus X oder das Symbol $*$; insgesamt stehen auf dem Band höchstens endlich viele Elemente aus X. Der Lese-/Schreibkopf ist in der Lage, das auf dem Band in einer Zelle stehende Element zu erkennen (zu lesen) und in eine Zelle ein neues Element einzutragen (zu schreiben). Das „Rechenwerk“ kann intern Informationen in Form von Elementen der Menge Z, den Zuständen, speichern. z_0 bezeichnet den Anfangszustand, in dem sich die Maschine zu Beginn ihrer Arbeit befindet. Q ist die Menge der Zustände, in denen die Maschine ihre Arbeit stoppt.
Ein Arbeitsschritt der Maschine besteht nun in folgendem: Die Maschine befindet sich in einem Zustand z, ihr Kopf befindet sich über einer Zelle i und liest deren Inhalt x; hiervon ausgehend berechnet die Maschine einen neuen Zustand z', schreibt in die Zelle i ein aus z und x berechnetes Element x' und bewegt den Kopf um eine Zelle nach rechts (R) oder nach links (L) oder bewegt den Kopf nicht (N). Dies wird durch

$$\delta(z, x) = (z', x', r) \quad \text{mit} \quad z, z' \in Z, x, x' \in X \cup \{*\}, r \in \{R, L, N\}$$

beschrieben.

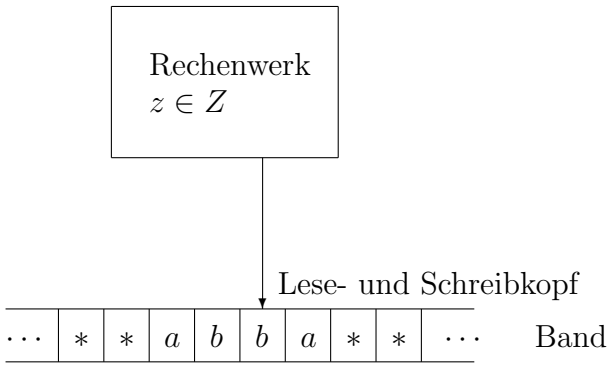

Abbildung 1.8: TURING-Maschine

Die aktuelle Situation, in der sich eine TURING-Maschine befindet, wird also durch das Wort (die Wörter) über X auf dem Band, den internen Zustand und die Stelle an der der Kopf steht, beschrieben. Formalisiert wird dies durch folgende Definition erfasst.

Definition 1.11 *Es sei M eine* TURING*-Maschine wie in Definition 1.10. Eine Konfiguration K der* TURING*-Maschine M ist ein Tripel*

$$K = (w_1, z, w_2),$$

wobei w_1 und w_2 Wörter über $X \cup \{*\}$ sind und $z \in Z$ gilt.
Eine Anfangskonfiguration liegt vor, falls $w_1 = \lambda$ und $z = z_0$ gelten.
Eine Endkonfiguration ist durch $z \in Q$ gegeben.

Wir interpretieren dies wie folgt: Auf dem Band steht das Wort w_1w_2; alle Zellen vor und hinter denjenigen, in denen w_1w_2 steht, sind mit $*$ belegt; der Kopf steht über der Zelle, in der der erste Buchstabe von w_2 steht; und die Maschine befindet sich im Zustand z.
Wir bemerken, dass eine Situation durch mehrere Konfigurationen beschrieben werden kann, z.B. beschreiben (λ, z, ab), $(*, z, ab)$ und $(**, z, ab*)$ alle die Situation, dass auf dem Band ab steht und der Kopf über a positioniert ist. Bei den nachfolgenden Definitionen und Beispielen wird jeweils unter den verschiedenen äquivalenten Konfigurationen eine geeignete Konfiguration ausgewählt.
Die folgende Definition formalisiert nun die Konfigurationsänderung, wenn die Maschine einen Schritt entsprechend δ ausführt.

Definition 1.12 *Es sei M eine* Turing-*Maschine wie in Definition 1.10, und es seien $K_1 = (w_1, z, w_2)$ und $K_2 = (v_1, z', v_2)$ Konfigurationen von M. Wir sagen, dass K_1 durch M in K_2 überführt wird (und schreiben dafür $K_1 \models K_2$), wenn eine der folgenden Bedingungen erfüllt ist:*

$$v_1 = w_1, w_2 = xu, v_2 = x'u, \delta(z, x) = (z', x', N)$$

oder

$$w_1 = v, v_1 = vx', w_2 = xu, v_2 = u, \delta(z, x) = (z', x', R)$$

oder

$$w_1 = vy, v_1 = v, w_2 = xu, v_2 = yx'u, \delta(z, x) = (z', x', L)$$

für gewisse $x, x', y \in X \cup \{*\}$ und $u, v \in (X \cup \{*\})^*$.

Offenbar kann eine Endkonfiguration in keine weitere Konfiguration überführt werden, da die Funktion δ für Zustände aus Q und beliebige $x \in X \cup \{*\}$ nicht definiert ist.

Definition 1.13 *Es sei M eine* Turing-*Maschine wie in Definition 1.10. Die durch M induzierte Funktion f_M aus X^* in X^* ist wie folgt definiert: $f_M(w) = v$ gilt genau dann, wenn es für die Anfangskonfiguration $K = (\lambda, z_0, w)$ eine Endkonfiguration $K' = (v_1, q, v_2)$, natürliche Zahlen r, s und t und Konfigurationen $K_0, K_1, \ldots, K_t$ derart gibt, dass $*^r v *^s = v_1v_2$ und*

$$K = K_0 \models K_1 \models K_2 \models \ldots \models K_t = K'$$

gelten.

Interpretiert bedeutet dies, dass sich durch mehrfache Anwendung von Überführungsschritten aus der Anfangskonfiguration, bei der w auf dem Band steht, eine Endkonfiguration ergibt, in der v auf dem Band steht. Falls in der Endkonfiguration (v_1, q, v_2) der Kopf über einer Zelle von v steht, so gelten $v = v_1v_2$ und $r = s = 0$; steht der Kopf dagegen r Zellen vor v bzw. s Zellen hinter v, so gelten $*^r v = v_1v_2$, $v_1 = \lambda$ und $s = 0$ bzw. $v*^s = v_1v_2$, $v_2 = \lambda$ und $r = 0$.
Wir bemerken ferner, dass für solche Wörter w, bei denen die Maschine nie ein Stopzustand aus Q erreicht, kein zugeordneter Funktionswert $f_M(w)$ definiert ist. Somit kann f_M auch eine partielle Funktion sein.

Beispiel 1.8 Um eine TURING-Mschine zu beschreiben, werden wir nachfolgend die Funktion δ immer durch eine Tabelle angeben, bei der im Schnittpunkt der zu $x \in X$ bzw. $*$ gehörenden Zeile und der zu $z \in Z \setminus Q$ gehörenden Spalte das Tripel $\delta(z, x)$ steht.
a) Es sei

$$M_1 = (\{a, b\}, \{z_0, q, z_a, z_b\}, z_0, \{q\}, \delta)$$

eine TURING-Maschine, und es sei δ durch die Tabelle aus Abb. 1.9 gegeben.

δ	z_0	z_a	z_b
$*$	$(q, *, N)$	(q, a, N)	(q, b, N)
a	$(z_a, *, R)$	(z_a, a, R)	(z_b, a, R)
b	$(z_b, *, R)$	(z_a, b, R)	(z_b, b, R)

Abbildung 1.9:

Wir starten mit dem Wort *abba* auf dem Band. Dann ergeben sich die folgenden Konfigurationen mittels Überführungen (um Übereinstimmung mit Definition 1.12 zu erreichen, haben wir die Konfiguration immer in die Form umgewandelt, die benötigt wird):

$$\begin{aligned}(\lambda, z_0, abba) &\models (*, z_a, bba) \models (*b, z_a, ba) = (b, z_a, ba) \models (bb, z_a, a) = (bb, z_a, a*) \\ &\models (bba, z_a, *) \models (bba, q, a).\end{aligned}$$

Folglich gilt

$$f_{M_1}(abba) = bbaa.$$

Ausgehend von *bab* erhalten wir

$$(\lambda, z_0, bab) \models (*, z_b, ab) \models (a, z_b, b) \models (ab, z_b, *) \models (ab, q, b)$$

und damit

$$f_{M_1}(bab) = abb.$$

Allgemein ergibt sich

$$f_{M_1}(x_1 x_2 \dots x_n) = x_2 x_3 \dots x_n x_1$$

(den zu Beginn gestrichenen Buchstaben x_1 merkt sich die Maschine in Form des Zustandes z_{x_1} und schreibt ihn an das Ende des Wortes).
b) Es sei

$$M_2 = (\{a, b\}, \{z_0, z_1, q\}, z_0, \{q\}, \delta),$$

wobei δ durch Abb. 1.10 gegeben sei.
Für *abb* und *abba* ergeben sich

$$(\lambda, z_0, abb) \models (a, z_1, bb) \models (ab, z_0, b) \models (abb, z_1, *) \models (abb, q, *)$$

und

$$\begin{aligned}(\lambda, z_0, abba) &\models (a, z_1, bba) \models (ab, z_0, ba) \models (abb, z_1, b) \\ &\models (abba, z_0, *) \models (abba, z_0, *) \models (abba, z_0, *) \models \dots \ .\end{aligned}$$

δ	z_0	z_1
$*$	$(z_0, *, N)$	$(q, *, N)$
a	(z_1, a, R)	(z_0, a, R)
b	(z_1, b, R)	(z_0, b, R)

Abbildung 1.10:

Folglich gilt

$$f_{M_2}(abb) = abb,$$

und $f_{M_2}(abba)$ ist nicht definiert. Es gilt

$$f_{M_2}(x_1x_2\ldots x_n) = \begin{cases} x_1x_2\ldots x_n & n \text{ ungerade} \\ nicht\ definiert & \text{sonst.} \end{cases}$$

c) Wir betrachten die TURING-Maschine $M_3 = (\{a, b, c, d\}, \{z_0, z_1, z_2, z_3, q, z_a, z_b\}, z_0, \{q\}, \delta)$ mit

δ	z_0	z_1	z_2	z_3	z_a	z_b
$*$	$(z_0, *, N)$	$(z_1, *, N)$	$(z_3, *, L)$			
a	(z_0, a, N)	(z_1, a, N)	(z_2, a, R)	$(z_a, *, L)$	(z_a, a, L)	(z_a, b, L)
b	(z_0, b, N)	(z_1, b, N)	(z_2, b, R)	$(z_b, *, L)$	(z_b, a, L)	(z_b, b, L)
c	(z_1, c, R)	(z_1, c, N)	(z_2, c, N)			
d	(z_0, d, N)	(z_2, d, R)	(z_2, d, N)	(z_3, d, N)	(q, a, N)	(q, b, N)

Für diese TURING-Maschine ergibt sich

$$f_{M_3}(w) = \begin{cases} cx_1x_2\ldots x_n & \text{für } w = cdx_1x_2\ldots x_n, x_i \in \{a, b\}, 1 \leq i \leq n, n \geq 0 \\ undefiniert & \text{sonst} \end{cases}$$

Zur Begründung merken wir Folgendes an: Wir laufen zuerst über das Wort, ändern den Zustand in z_1 bzw. z_2 wenn wir als ersten bzw. zweiten Buchstaben ein c bzw. ein d lesen und bleiben im Zustand z_2, wenn wir danach nur Buchstaben aus $\{a, b\}$ lesen. Damit wissen wir, dass das Eingabewort die Form hat, bei der eine Ausgabe definiert ist. Bei Erreichen des Wortendes gehen wir in den Zustand z_3. Jetzt laufen wir von rechts nach links über das Wort, merken uns jeweils einen gelesenen Buchstaben und schreiben diesen in die links davon befindliche Zelle. Dadurch verschieben wir das Wort über $\{a, b\}$ um eine Zelle nach links. Nach Lesen des d stoppt die Maschine.
Wir bemerken, dass M_3 im Wesentlichen den Buchstaben d löscht und die dadurch entstehende Lücke durch Verschiebung wieder füllt. Wir werden im Folgenden mehrfach davon Gebrauch machen, dass das Streichen, Einfügen und Verschieben von Wörtern/Teilwörtern von TURING-Maschinen realisiert werden können, ohne dies dann explizit auszuführen.

d) Wir wollen eine TURING-Maschine konstruieren, deren induzierte Funktion die Nachfolgerfunktion (bzw. die Addition von 1) ist, wobei wir die Dezimaldarstellung für Zahlen verwenden wollen.
Offenbar muss das Eingabealphabet aus den Ziffern $0, 1, 2, \ldots, 9$ bestehen. Wir werden als Grundidee die schriftliche Addition verwenden, d.h. wir verwenden den Anfangszustand

z_0, um das Wortende zu finden, indem wir bis zum ersten $*$ nach rechts laufen; danach verwenden wir einen Zustand $+$ zur Addition von 1 bei der Ziffer, über der der Kopf gerade steht; die Addition kann abgebrochen werden, falls die Addition nicht zur Ziffer 9 erfolgt, bei der der entstehende Übertrag 1 zur Fortsetzung der Addition von 1 zu der links davon stehenden Ziffer notwendig wird. Formal ergibt sich die Maschine

$$M_+ = (\{0, 1, 2, \ldots, 9\}, \{z_0, +, q\}, z_0, \{q\}, \delta)$$

mit δ aus Abb. 1.11.

δ	z_0	$+$
$*$	$(+, *, L)$	$(q, 1, N)$
0	$(z_0, 0, R)$	$(q, 1, N)$
1	$(z_0, 1, R)$	$(q, 2, N)$
2	$(z_0, 2, R)$	$(q, 3, N)$
3	$(z_0, 3, R)$	$(q, 4, N)$
4	$(z_0, 4, R)$	$(q, 5, N)$
5	$(z_0, 5, R)$	$(q, 6, N)$
6	$(z_0, 6, R)$	$(q, 7, N)$
7	$(z_0, 7, R)$	$(q, 8, N)$
8	$(z_0, 8, R)$	$(q, 9, N)$
9	$(z_0, 9, R)$	$(+, 0, L)$

Abbildung 1.11:

Wir geben nun eine Normalform für TURING-Maschinen.

Lemma 1.9 *Zu jeder* TURING*-Maschine* $M = (X, Z, z_0, Q, \delta)$ *gibt es eine* TURING*-Maschine*

$$M' = (X \cup \{\S, \#\}, Z', z_0', \{q'\}, \delta')$$

mit

$$f_{M'}(w) = \begin{cases} f_M(w) & \text{für } w \in X^* \\ \text{nicht definiert} & \text{sonst} \end{cases}$$

derart, dass jede Endkonfiguration von M' *die Form* (λ, q', v) *für* $v \neq \lambda$ *oder* $(\lambda, q', *)$ *hat (d.h. die Maschine* M' *hat genau einen Stoppzustand, stoppt nur auf Wörtern über* X *und stoppt stets über dem ersten Buchstaben des Ergebnisses* v*).*

Beweis. Es sei die TURING-Maschine $M = (X, Z, z_0, Q, \delta)$ gegeben. Wir konstruieren dann die TURING-Maschine

$$M' = (X \cup \{\S, \#\}, Z \cup (Z \times \{\#\}) \cup (Z \times \{\S\}) \cup \{z_0', z_0'', q_1, q_2, q_3, q'\}, z_0', \{q'\}, \delta'),$$

wobei δ' wie folgt definiert ist:

(1) $\delta'(z_0', x) = (z_0', x, R)$ für $x \in X$,
$\delta'(z_0', \S) = (z_0', \S, N)$,

$\delta'(z_0', \#) = (z_0', \#, N)$,
$\delta'(z_0', *) = (z_0'', \#, L)$,
$\delta'(z_0'', x) = (z_0'', x, L)$ für $x \in X$,
$\delta'(z_0'', *) = (z_0, \S, R)$,

(2) $\delta'(z, x) = (z', x', r)$ für $x \in X \cup \{*\}, z \in Z \setminus Q, \delta(z, x) = (z', x', r), r \in \{R, L, N\}$,

(3) $\delta'(z, \S) = ((z, \S), *, L)$ für $z \in Z$,
$\delta'((z, \S), *) = (z, \S, R)$ für $z \in Z$,
$\delta'(z, \#) = ((z, \#), *, R)$ für $z \in Z$,
$\delta'((z, \#), *) = (z, \#, L)$ für $z \in Z$,

(4) $\delta'(q, x) = (q, x, R)$ für $x \in X \cup \{*\}, q \in Q$,
$\delta'(q, \#) = (q_1, *, L)$,
$\delta'(q_1, *) = (q_1, *, L)$,
$\delta'(q_1, x) = (q_2, x, L)$ für $x \in X$,
$\delta'(q_1, \S) = (q', *, N)$,
$\delta'(q_2, x) = (q_2, x, L)$ für $x \in X \cup \{*\}$,
$\delta'(q_2, \S) = (q_3, *, R)$,
$\delta'(q_3, *) = (q_3, *, R)$,
$\delta'(q_3, x) = (q', x, N)$ für $x \in X$

(für die Paare, für die δ' nicht definiert, kann ein beliebiges Tripel als Wert festgelegt werden, da die Arbeitsweise von M' sichert, dass solche Paare nicht erreicht werden können). Dass M' allen Bedingungen genügt, die in Lemma 1.9 gefordert werden, ist aus folgenden Überlegungen zu ersehen: Entsprechend der Definition von δ' im Teil (1) wird zuerst getestet, ob das Wort w auf dem Band ein § oder ein # enthält. Ist dies der Fall, so wird eine Schleife erreicht (die Konfiguration wird stets in sich selbst überführt) und damit kein Ergebnis von $f_{M'}$ erreicht. Ist kein § und kein # in w, so wird hinter das Wort ein # und vor das Wort ein § auf das Band geschrieben. Danach verhält sich die TURING-Maschine M' wegen der Definition von δ' in (2) genauso wie M', wobei mittels der Festlegungen in (3) gesichert wird, dass die Anfangsmarkierung § und die Endmarkierung # stets um eine Zelle nach links bzw. rechts verschoben wird, wenn dies erforderlich ist (d.h. wird ein § erreicht, so wird es durch $*$ ersetzt wird und danach wird die Arbeit in der Zelle, in der ursprünglich § stand und jetzt $*$ steht, mit dem Zustand z fortgesetzt, den die Maschine hatte, als sie diese Zelle betrat, da z in der ersten Komponente von $(z, \$)$ gespeichert wurde; analog wird bei # verfahren). Während M in Zuständen aus Q ihre Arbeit beendet, bewegt M' in diesen Zuständen den Kopf nach rechts, bis # erreicht wird, löscht #, geht dann nach links, bis § erreicht wird. M' löscht § und stoppt, falls zwischen § und # kein Symbol aus X stand, oder geht nach rechts bis zum ersten Buchstaben aus X und stoppt. □

Definition 1.14 *Eine Funktion $f : X_1* \to X_2*$ heißt* TURING*-berechenbar, wenn es eine* TURING*-Maschine $M = (X, Z, z_0, Q, \delta)$ mit $X_1 \subseteq X$ und $X_2 \subseteq X$ und*

$$f_M(x) = \begin{cases} f(x) & \text{falls } f(x) \text{ definiert ist} \\ \text{nicht definiert} & \text{sonst} \end{cases}$$

gibt.

Wir beweisen nun, dass eine Eigenschaft, die partiell-rekursive Funktionen nach Definition (und damit auch **LOOP/WHILE**-berechenbare Funktionen) haben, auch TURING-berechenbare Funktionen haben.

Lemma 1.10 *Sind $f_1 : X^* \to X^*$ und $f_2 : X^* \to X^*$ zwei* TURING*-berechenbare Funktionen, so ist auch deren Komposition $f : X^* \to X^*$ mit $f(w) = f_2(f_1(w))$ eine* TURING*-berechenbare Funktion.*

Beweis. Nach Definition 1.14 und Lemma 1.9 gibt es TURING-Maschinen

$$M_1 = (X \cup \{\S, \#\}, Z_1, z_{0,1}, \{q_1\}, \delta_1)$$

und

$$M_2 = (X \cup \{\S, \#\}, Z_2, z_{0,2}, \{q_2\}, \delta_2)$$

mit

$$f_{M_1}(w) = \begin{cases} f_1(w) & \text{für } w \in X^* \\ \text{nicht definiert} & \text{sonst} \end{cases}$$

und

$$f_{M_2}(w) = \begin{cases} f_2(w) & \text{für } w \in X^* \\ \text{nicht definiert} & \text{sonst} \end{cases}$$

und der Eigenschaft, dass beide TURING-Maschinen über dem ersten Buchstaben des Ergebnisses stoppen. Ohne Beschränkung der Allgemeinheit nehmen wir an, dass Z_1 und Z_2 kein Element gemeinsam haben, und betrachten die TURING-Maschine

$$M = (X \cup \{\S, \#\}, Z_1 \cup Z_2, z_{0,1}, \{q_2\}, \delta)$$

mit

$$\delta(z, x) = \begin{cases} \delta_1(z, x) & \text{für } z \in Z_1, z \neq q_1 \\ (z_{0,2}, x, N) & \text{für } z = q_1 \\ \delta_2(z, x) & \text{für } z \in Z_2, z \neq q_2 \end{cases}.$$

Da der Anfangszustand von M in Z_1 liegt, beginnt M auf der Eingabe w wie M_1 zu arbeiten, bis der Endzustand q_1 von M_1 erreicht wird und damit die Konfiguration $(\lambda, q_1, f_{M_1}(w))$ vorliegt. Für M ist q_1 kein Endzustand und erreicht nach Definition von δ die Konfiguration $(\lambda, z_{0,2}, f_{M_1}(w))$, die gerade die Anfangskonfiguration von M_2 bei Eingabe von $f_{M_1}(w)$ ist. Nun verhält sich M wie M_2 und stoppt mit der Konfiguration $(\lambda, q_2, f_{M_2}(f_{M_1}(w)))$. Damit ergibt sich

$$f_M(w) = f_{M_2}(f_{M_1}(w))) = f_2(f_1(w)) = f(w).$$

Daher wird f von einer TURING-Maschine induziert und ist damit TURING-berechenbar. □

Entsprechend der Definition hat die TURING-Maschine ein Arbeitsband, das sowohl als Eingabe- als auch Ausgabeband dient. Wir wollen nun eine Variante der TURING-Maschine behandeln, bei der ein Eingabeband, ein Ausgabeband und mehrere zusätzliche Bänder zur Rechnung zur Verfügung stehen. Dies führt in der Regel zu einer einfacheren Berechnung von Funktionen.

Definition 1.15 *Eine k-Band-*TURING*-Maschine ist ein 6-Tupel*

$$M = (k, X, Z, z_0, Q, \delta),$$

wobei $k \geq 1$ eine natürliche Zahl ist, X, Z, z_0 und Q wie bei einer TURING*-Maschine definiert sind, δ eine Funktion*

$$(Z \setminus Q) \times (X \cup \{*\})^{k+1} \longrightarrow Z \times (X \cup \{*\})^{k+1} \times \{R, L, N\}^{k+1} \times \{R, N\}$$

ist und $* \notin X$ gilt.

Die k-Band-TURING-Maschine verfügt über ein Eingabeband, auf dem nur gelesen werden darf, ein Ausgabeband, auf das nur von links nach rechts geschrieben werden darf, und k Arbeitsbändern mit jeweils einem Lese-Schreibkopf. Wir interpretieren X, Z, z_0, Q und die Elemente aus $\{R, L, N\}$ wie bei einer TURING-Maschine. Falls

$$\delta(z, x_e, x_1, x_2, \ldots, x_k) = (z', y_1, y_2, \ldots, y_k, y_a, r_e, r_1, r_2, \ldots, r_k, r_a)$$

gilt, so interpretieren wir dies wie folgt: Die Maschine liest im Zustand z auf dem Eingabeband den Buchstaben x_e, auf dem i-ten Arbeitsband den Buchstaben x_i, $1 \leq i \leq k$, geht in den Zustand z' über, schreibt den Buchstaben y_i auf das i-te Arbeitsband, $1 \leq i \leq k$, und y_a auf das Arbeitsband und der Lesekopf des Eingabebandes bewegt sich nach $r_e \in \{R, N, L\}$, der Schreibkopf des Ausgabebandes nach $r_a \in \{R, N\}$ und der Kopf des i-ten Arbeitsbandes nach $r_i \in \{R, L, N\}$, $1 \leq i \leq k$.

Definition 1.16 *Es sei M eine k-Band-*TURING*-Maschine wie in Definition 1.15.*
Eine Konfiguration von M ist ein $2k + 5$-Tupel

$$(z, w_e, w'_e, w_1, w'_1, w_2, w'_2, \ldots, w_k, w'_k, w_a, w'_a), \tag{1.2}$$

wobei $z \in Z$, $w_e, w'_e, w_a, w'_a \in (X \cup \{*\})^*$ und $w_i, w'_i \in (X \cup \{*\})^*$ für $1 \leq i \leq k$ gelten.
Eine Konfiguration heißt Anfangskonfiguration, falls $z = z_0$, $w_e = w_a = w_1 = w_2 = \ldots = w_k = \lambda$ und $w'_a = w'_1 = w'_2 = \ldots = w'_k = *$ gelten.
Eine Konfiguration heißt Endkonfiguration, falls z in Q liegt.

Wir interpretieren eine Konfiguration (1.2) wie folgt: Die Maschine befindet sich im Zustand z, auf dem Eingabeband steht $w_e w'_e$ und der Lesekopf steht über dem ersten Buchstaben von w'_e auf dem Ausgabeband steht $w_a w'_a$ und der Schreibkopf steht über dem ersten Buchstaben von w'_a und für $1 \leq i \leq k$ steht auf dem i-ten Arbeitsband $w_i w'_i$ und steht der Kopf über dem ersten Buchstaben von w'_i.

Wir überlassen dem Leser eine formale Definition der Änderung der Konfiguration K_1 in die Konfiguration K_2, die sich aus dem bisher gesagtem in Analogie zu Definition 1.12 ergibt und die wir wieder mit $K_1 \vdash K_2$ bezeichnen.

Definition 1.17 *Es sei M eine k-Band-*TURING*-Maschine wie in Definition 1.15. Die durch M induzierte Funktion f_M aus X^* in X^* ist wie folgt definiert: $f_M(w) = v$ gilt genau dann, wenn es für die Anfangskonfiguration*

$$K = (z_0, \lambda, w, \lambda, *, \lambda, *, \ldots, \lambda, *)$$

eine Endkonfiguration

$$K' = (q, w_e, w'_e, w_1, w'_1, w_2, w'_2, \ldots, w_k, w'_k, w_a, w'_a),$$

und Konfigurationen $K_0, K_1, \ldots, K_t$ *derart gibt, dass*

$$v = w_a w'_a$$

und

$$K = K_0 \models K_1 \models K_2 \models \ldots \models K_t = K'$$

gelten.

Wir betrachten einige Beispiele.

Beispiel 1.9 a) Wir betrachten die 2-Band-TURING-Maschine M mit

$$X = \{a, b\}, \quad Z = \{z_0, z_1, z_2, q\}, Q = \{q\}$$

und der Funktion δ, die durch

$$\begin{array}{ll}
(a1) & \delta(z_0, x, *, *) = (z_0, x, x, *, R, R, R, N) \quad \text{für } x \in X, \\
(a2) & \delta(z_0, *, *, *) = (z_1, *, *, *, N, L, L, N), \\
(a3) & \delta(z_1, *, x, y) = (z_1, x, y, *, N, L, N, N) \quad \text{für } x, y \in X, \\
(a4) & \delta(z_1, *, *, y) = (z_2, *, y, *, N, R, N, N) \quad \text{für } y \in X, \\
(a5) & \delta(z_2, *, x, x) = (z_2, x, x, *, N, R, L, N) \quad \text{für } x \in X, \\
(a6) & \delta(z_2, *, x, y) = (q, x, y, b, N, N, N, N) \quad \text{für } x, y \in X,\ x \neq y, \\
(a7) & \delta(z_2, *, *, *) = (q, *, *, a, N, N, N, N)
\end{array}$$

und

$$\delta(z, x, y, v) = (z, y, v, *, N, N, N, N)$$

in allen sonstigen Fällen gegeben ist.
Entsprechend $(a1)$ wird zuerst das Wort w auf dem Eingabeband vollständig gelesen und dabei sowohl auf das erste als auch das zweite Arbeitsband kopiert. Dann wird mittels $(a2)$ in den Zustand z_1 gegangen, der aufgrund von $(a3)$ und $(a4)$ bewirkt, dass auf dem ersten Arbeitsband zum Wortanfang gegangen wird, während der Kopf des zweiten Arbeitsbandes über dem letzten Buchstaben stehen bleibt. Dabei wird Zustand z_2 erreicht, in dem nun verglichen wird, ob der i-te Buchstabe von w von vorn mit dem i-ten Buchstaben von w von hinten übereinstimmt (der Kopf auf dem ersten Arbeitsband geht beim ersten Buchstaben von w beginnend nach rechts, während der Kopf auf dem zweiten Arbeitsband mit dem letzten Buchstaben beginnend nach links geht). Wird keine Übereinstimmung festgestellt, so geht die Maschine in den Stoppzustand und gibt b aus. Wird Übereinstimmung festgestellt, so wird der Vergleich beim nächsten Buchstaben fortgesetzt. Sind alle Buchstaben verglichen worden, so geht die Maschine in den Stoppzustand und gibt a aus. Folglich berechnet M die Funktion

$$f_M(w) = \begin{cases} a & w \text{ ist Palindrom} \\ b & \text{sonst} \end{cases}$$

(ein Wort $w = x_1x_2 \dots x_{n-1}x_n$ heißt Palindrom, falls für $1 \leq i \leq n$ die Relation $x_i = x_{n-i+1}$ gilt, d.h. w ändert sich nicht, wenn man es von hinten liest).
b) Wir betrachten die 2-Band-TURING-Maschine M' mit

$$X = X' \cup \{\#\},\ X' = \{0, 1, 2, \dots 9\}, \quad Z = \{z_0, z_1, z_2, z_+, z'_+, q\}, \quad Q = \{q\}$$

und der Funktion δ, die durch

$(b1)$ $\delta(z_0, x, *, *) = (z_0, x, *, *, R, R, N, N)$ für $x \in X'$,
$(b2)$ $\delta(z_0, \#, *, *) = (z_1, *, *, *, R, L, N, N)$,
$(b3)$ $\delta(z_1, x, y, *) = (z_1, y, x, *, R, N, R, N)$ für $x, y \in X'$,
$(b4)$ $\delta(z_1, *, y, *) = (z_+, y, *, *, N, N, L, N)$ für $y \in X'$,
$(b5)$ $\delta(z_+, *, x, y) = (z_+, x + y, *, *, N, L, L, N)$ für $x, y \in X', x + y < 10$,
$(b6)$ $\delta(z_+, *, x, y) = (z'_+, s, *, *, N, L, L, N)$ für $x, y \in X, x + y = 10 + s, s \geq 0$,
$(b7)$ $\delta(z'_+, *, x, y) = (z_+, x + y + 1, *, *, N, L, L, N)$ für $x, y \in X', x + y + 1 < 10$,
$(b8)$ $\delta(z'_+, *, x, y) = (z'_+, s, *, *, N, L, L, N)$ für $x, y \in X, x + y + 1 = 10 + s, s \geq 0$,
$(b9)$ $\delta(z_+, *, x, *) = (z_+, x, *, *, N, L, L, N)$ für $x \in X'$,
$(b10)$ $\delta(z'_+, *, x, *) = (z_+, x + 1, *, *, N, L, L, N)$ für $x \in X', x \neq 9$,
$(b11)$ $\delta(z'_+, *, 9, *) = (z'_+, 0, *, *, N, L, L, N, N)$
$(b12)$ $\delta(z_+, *, *, x) = (z_+, x, *, *, N, L, L, N)$ für $x \in X'$,
$(b13)$ $\delta(z'_+, *, *, x) = (z_+, x + 1, *, *, N, L, L, N)$ für $x \in X', x \neq 9$,
$(b14)$ $\delta(z'_+, *, *, 9) = (z'_+, 0, *, *, N, L, L, N)$
$(b15)$ $\delta(z_+, *, *, *) = (z_2, *, *, *, N, R, N, N)$
$(b16)$ $\delta(z'_+, *, *, *) = (z_2, 1, *, *, N, N, N, N)$
$(b17)$ $\delta(z_2, *, x, *) = (z_2, x, *, x, N, R, N, R)$ für $x \in X'$,
$(b18)$ $\delta(z_2, *, *, *) = (q, *, *, *, N, N, N, N)$

und

$$\delta(z, x, y, v) = (z, y, v, *, N, N, N, N)$$

in allen sonstigen Fällen gegeben ist.
Wir betrachten nur das Ergebnis der Rechnung für eine Eingabe zweier Dezimalzahlen, d.h. für den Fall, dass $w_1\#w_2$ auf dem Eingabeband steht, wobei # ein Trennsymbol ist. Solange M' im Zustand z_0 ist, wird w_1 auf das erste Arbeitsband übertragen. In z_1 erfolgt analog die Übertragung von w_2 auf das zweite Arbeitsband. Ist dies erfolgt, so stehen die Köpfe der Arbeitsbänder jeweils über den letzten Buchstaben von w_1 bzw. w_2 und M. Außerdem ist im Zustand z_+, der nun die Addition der beiden Zahlen mit den Darstellungen w_1 bzw.w_2 in Analogie zur schriftlichen Addition beginnt. Im Zustand z_+ liegt kein Übertrag vor, während z'_+ die Berücksichtigung des Übertrages von 1 realisiert. Das Ergebnis der Addition wird auf das erste Arbeitsband geschrieben. Ist die Addition vollständig ausgeführt, d.h. von beiden Bändern wird ein $*$ gelesen, so geht die Maschine in den Zustand z_2, in dem sie das Ergebnis der Addition auf das Ausgabeband überträgt. Folglich gilt

$$f_{M'}(dec(n_1)\#dec(n_2)) = dec(n_1 + n_2),$$

wobei wir mit $dec(x)$ die Dezimaldarstellung von x bezeichnen.

Definition 1.18 *Eine Funktion $f : X_1* \to X_2*$ heißt k-*TURING*-berechenbar, wenn es eine k-Band*TURING*-Maschine $M = (k, X, Z, z_0, Q, \delta)$ mit $X_1 \subseteq X$ und $X_2 \subseteq X$ und*

$$f_M(x) = \begin{cases} f(x) & \text{falls } f(x) \text{ definiert ist} \\ \text{nicht definiert} & \text{sonst} \end{cases}$$

gibt.

Wir wollen nun einen Zusammenhang zwischen den von TURING-Maschinen und Registermaschinen induzierten Funktionen herstellen. Eine Gleichheit ist nicht möglich, da die von TURING-Maschinen induzierten Funktionen Wörter in Wörter abbilden, während die von Registermaschinen berechneten Funktionen Tupel natürlicher Zahlen auf natürliche Zahlen abbilden. Jedoch kann jede natürliche Zahl durch eine Folge von Ziffern beschrieben werden, d.h. durch ein Wort über der Menge der Ziffern.
Für eine natürliche Zahl m bezeichne $dec(m)$ die Dezimaldarstellung von n.
Der folgende Satz besagt nun, dass es zu jeder Registermaschine M eine mehrbändige TURING-Maschine gibt, die im wesentlichen dasselbe wie M leistet, d.h. auf eine Eingabe der Dezimaldarstellungen von $m_1, m_2, \ldots m_n$ liefert die TURING-Maschine die Dezimaldarstellung von $f_M(m_1, m_2, \ldots, m_n)$, also die Dezimaldarstellung des Ergebnisses der Berechnung von M.

Satz 1.11 *Es sei M eine Registermaschine M mit $f_M : \mathbb{N}_0^n \to \mathbb{N}_0$. Dann gibt es eine 3-Band-*TURING*-Maschine M', deren Eingabealphabet außer den Ziffern 0,1,2,... 9 noch das Trennsymbol # und das Fehlersymbol F enthält und deren induzierte Funktion*

$$f_{M'}(w) = \begin{cases} dec(f_M(m_1, m_2, m_3 \ldots, m_n)) & w = dec(m_1)\#dec(m_2)\#dec(m_3)\ldots\#dec(m_n) \\ F & sonst \end{cases}$$

gilt (auf einer Eingabe, die einem Zahlentupel entspricht, verhält sich M' wie M und gibt bei allen anderen Eingaben eine Fehlermeldung).

Beweis. Wir geben hier keinen vollständigen formalen Beweis; wir geben nur die Idee des Beweises wider; der formale Beweis lässt sich unter Verwendung der Konstruktionen und Ideen aus den Beweisen der Lemmata 1.9 und 1.10 erbringen.
Wir konstruieren eine 3-Band-TURING-Maschine M', die schrittweise die Arbeit von M simuliert:
Auf dem ersten Arbeitsband speichern wir im Wesentlichen die Konfiguration der Registermaschine. Da diese ein unendliches Tupel ist, kann dies nicht direkt geschehen. Wir geben dort im wesentlichen die Folge der Nummern und Inhalte der Register an, die im Laufe der schon simulierten Schritte belegt worden sind. Formal steht auf dem ersten Band das Wort

$$\#\#0\#dec(c_0)\#\#dec(k_1)\#dec(c_{k_1})\#\#dec(k_2)\#dec(c_{k_2})\ldots\#\#dec(k_s)\#dec(c_{k_s})\#\#$$

(d.h. durch ## werden die verschiedenen Register voneinander getrennt; es wird stets die Nummer 0 bzw. k_i des Registers, $1 \leq i \leq s$, und der Inhalt c_0 bzw. c_{k_i} angegeben die durch ein # getrennt sind; in allen anderen Registern steht eine 0; da stets nur endlich viele Register einer Registermaschine mit von Null verschiedenen Werten belegt werden, enthält das erste Band stets nur endlich viele Symbole).

Die gegebene Registermaschine M habe ein Programm mit r Befehlen. Für $1 \leq i \leq r$ konstruieren wir eine TURING-Maschine M_i, die eine Änderung des ersten Bandes entsprechend dem i-ten Befehl vornimmt. M_i arbeitet nur auf den drei Arbeitsbändern. Nach der eigentlichen Simulation des Befehls der Registermaschine werden das zweite und dritte Arbeitsband stets geleert und auf dem ersten Arbeitsband der Kopf zum Anfang bewegt (d.h. er steht über dem ersten #). $z_{i,0}$ sei der Anfangszustand und q_i der Stoppzustand von M_i. Um festzuhalten, welcher Befehl gerade simuliert wird, haben die Zustände von M' die Form (i, z), wobei $1 \leq i \leq r$ gilt und z ein Zustand von M_i ist. Für eine Anfangsphase werden noch weitere Zustände benötigt.
M' arbeitet nun wie folgt: Zuerst testet M', ob die Eingabe die Form $w_1\#w_2\#\ldots\#w_n$, wobei für $1 \leq i \leq n$ entweder $w_i = 0$ oder $w_i = x_iy_i$ mit $x_i \in \{1, 2, \ldots, 9\}$ und $y_i \in \{0, 1, \ldots, 9\}^*$ gilt, d.h. ob die Eingabe die Kodierung eines n-Tupels natürlicher Zahlen ist.
Ist dies nicht der Fall, so schreibt M' das Fehlersymbol F auf das Ausgabeband und stoppt.
Im anderen Fall schreibt M' auf das erste Arbeitsband die entsprechend modifizierte Eingabe

$$\#\#0\#0\#\#1\#dec(m_1)\#\#2\#dec(m_2)\ldots\#\#dec(n)\#dec(m_n)\#\#,$$

geht in den Zustand $(1, z_{1,0})$ über, und M_1 beginnt mit der Simulation des ersten Befehls. M_i, $1 \leq i \leq r$, beendet seine Simulation im Zustand (i, q_i), da während der Simulation nur die zu M_i gehörende zweite Komponente geändert wird. M' geht nun in den Zustand $(j, z_{j,0})$, wobei j der nach dem i-ten Befehl abzuarbeitende Befehl ist.

Wir geben nun die Arbeitsweise einiger TURING-Maschinen zu Befehlen der Registermaschine an, wobei wir nur die Änderung des Inhalts des ersten Arbeitsbandes angeben (es folgen dann noch die Löschungen auf den anderen Arbeitsbändern und die Kopfbewegung zum Anfang des ersten Arbeitsbandes).
a) Es sei `LOAD` t der i-te Befehl. Dann schreibt M_i zuerst $dec(t)$ auf das zweite Arbeitsband und testet dann, ob im t-ten Register schon etwas gespeichert ist. Dazu läuft sie über das Wort auf dem ersten Arbeitsband und vergleicht stets ob nach zwei aufeinanderfolgenden # das Wort $dec(t)$ folgt. Hinter $dec(t)$ steht dann $\#dec(c_t)\#$ auf dem ersten Band. M_i leert das zweite Band und kopiert $dec(c_t)$ auf das zweite Band. Dann ersetzt M_i den Inhalt des Akkumulators (zwischen dem dritten und vierten # auf dem Band durch den Inhalt $dec(c_t)$ des zweiten Registers. Findet M_i keinen Eintrag im t-ten Register (ist bei Erreichen eines $*$ gegeben), so wird eine 0 in den Akkumulator geschrieben.
b) Im Fall der indirekten Adressierung `ILOAD` t schreiben wir zuerst den Inhalt des t-ten Registers in Dezimaldarstellung auf das zweite Band (dieser wird analog zu a) gesucht) und mit diesem Wert anstelle von $dec(t)$ fahren wir wie bei a).
c) Ist der i-te Befehl `CLOAD` t, so schreiben wir gleich $dec(t)$ auf das zweite Band und kopieren dies in den Akkumulator (zwischen dritten und viertem #).
d) Ist `STORE` t der i-te Befehl, so kopiert M_i den Inhalt $dec(c_0)$ des Akkumulators auf das dritte Band, schreibt $dec(t)$ auf das zweite Band, sucht die Stelle, wo der Inhalt des t-Registers steht, (dies steht hinter $\#\#dec(t)\#$) und ersetzt diesen durch den Inhalt des dritten Bandes. Wird $\#\#dec(t)\#$ nicht gefunden (d.h. es erfolgte noch kein Eintrag in dies Register, so wird $dec(t)\#dec(c_0)\#\#$ an das Wort auf dem ersten Band angefügt.

e) Ist der i-te Befehl `ADD` t, so wird zuerst der Inhalt des t-ten Registers gesucht und auf das zweite Band geschrieben. Durch ein # getrennt schreibt M_i den Inhalt des Akkumulators dahinter und addiert beide Zahlen, wobei das Ergebnis auf das dritten Band geschrieben wird. Dies Ergebnis schreiben wir in den Akkumulator.
f) Beim Befehl `GOTO` t ändern wir direkt den Zustand $(i, z_{i,0})$ zu $(t, z_{t,0})$.
g) Beim Befehl `IF` $c_o \neq 0$ `GOTO` t testet M_i, ob zwischen dem dritten und vierten # genau eine 0 steht. Ist dies der Fall geht M' in den Zustand $(t, z_{t,0})$, andererfalls in $(i+1, z_{i+1,0})$. Die Konstruktionen für die anderen Fälle sind analog.
h) Beim Befehl `END` wird der Inhalt des ersten Registers (zwischen dem sechsten und siebenten #) auf das Ausgabeband geschrieben und gestoppt.

Aus diesen Erklärungen folgt sofort, dass M' bei Eingabe von der Kodierung eines Zahlentupels schrittweise die Befehle der Registermaschine simuliert und am Ende die Kodierung des Inhalts des ersten Registers, in dem das Ergebnis der Berechnung der Registermaschine steht, ausgibt. □

Satz 1.12 *Zu jeder k-Band-*Turing*-Maschine M gibt es eine* Turing*-Maschine M' derart, dass $f_{M'} = f_M$ gilt.*

Beweis. Wir geben auch hier keinen formalen Beweis, sondern erläutern nur die Idee der Konstruktion. Im Wesentlichen schreiben wir den Inhalt des Eingabebandes, der Arbeitsbänder und des Ausgabebandes von M hintereinander auf das eine Band der Turimg-Maschine M' und markieren die Stellen, an denen die Köpfe stehen, dadurch, dass anstelle des eigentlichen Inhalt x der Zelle unterm Kopf den Inhalt (x, k) verwenden. Um einen Überführungsschritt von M zu simulieren, suchen wir nacheinander die markierten Stellen auf den Wörtern auf, führen das aus, was auf dem zum Wort gehörigen Band zu tun wäre, und markieren die neue Stelle des Kopfes. Bei Erreichen eines Stoppzustandes setzen wir die Arbeit noch fort, indem wir alle Wörtes mit Ausnahme der Ausgabe selbst streichen. □

Wir wollen nun zeigen, dass die Turing-berechenbaren Funktionen im Wesentlichen partiell-rekursive Funktionen sind. Hierbei haben wir die Schwierigkeit, dass Turing-berechenbare Funktionen als Definitionsbereich Mengen von Wörtern über einem beliebigen Alphabet haben, während der Definitionsbereich einer beliebigen partiell-rekursiven Funktion eine Teilmenge von $\mathbf{N}^r$ für ein $r \geq 0$ ist; gleiches gilt auch für den Wertevorrat. Daher müssen wir Kodierungen betrachten mittels derer die jeweiligen Eingangsgrößen ineinander überführt werden.
Es sei $M = (X, Z, z_0, Q, \delta)$ eine Turing-Maschine. Ohne Beschränkung der Allgemeinheit können wir annehmen, dass X und Z disjunkt sind und $* \notin Z$ gilt. Es sei

$$X \cup Z \cup \{*\} = \{a_1, a_2, \ldots, a_p\}.$$

Dann definieren wir die Funktion $\psi : (X \cup Z \cup \{*\})^* \to \mathbf{N}$ durch

$$\psi(a_{i_1} a_{i_2} \ldots a_{i_n}) = \sum_{j=1}^{n} i_j (p+1)^{n-j},\ a_{i_j} \in (X \cup Z)$$

($i_1 i_2 \dots i_n$ ist die $(p+1)$-adische Darstellung von $\psi(a_{i_1} a_{i_2} \dots a_{i_n})$). Ist umgekehrt x eine beliebige natürliche Zahl, in deren $(p+1)$-adischer Darstellung $i_1 i_2 \dots i_n$ keine 0 vorkommt, so gilt $\psi^{-1}(x) = a_{i_1} a_{i_2} \dots a_{i_n}$. Daher ist ψ eine eineindeutige Abbildung von $(X \cup Z)^+$ auf die Menge aller natürlichen Zahlen, in deren $(p+1)$-adischer Darstellung keine 0 vorkommt. (Bei Benutzung einer p-adischen Darstellung müsste ein Element aus $X \cup Z \cup \{*\}$ der Null zugeordnet werden, wodurch Darstellungen mit führenden Nullen möglich sind, was ausgeschlossen sein soll.)
Es seien $w = b_1 b_2 \dots b_n$ mit $b_i \in X \cup Z$ für $1 \leq i \leq n$ und $w' \in (X \cup Z)^*$. Dann gelten – wie man leicht nachrechnet – folgende Beziehungen für die folgenden Funktionen:

$$\begin{aligned}
Lg(\psi(w)) &= |w| = min\{m : (p+1)^m > \psi(w)\}, \\
Prod(\psi(w), \psi(w')) &= \psi(ww') = \psi(w)(p+1)^{Lg(\psi(w'))} + \psi(w'), \\
Anfang(\psi(w), i) &= \psi(b_1 b_2 \dots b_i) = \psi(w) \; div \; (p+1)^{n-i} \; (ganzzahligeDivision), \\
Ende(\psi(w), i) &= \psi(b_i b_{i+1} \dots b_n) = \psi(w) \; mod \; (p+1)^{n-i+1}, \\
Elem(\psi(w), i) &= \psi(b_i) = Ende(Anfang(\psi(w), i), 1)
\end{aligned}$$

(für die Definition von *div* und *mod* siehe Übungsaufgabe 20), d.h. aus der Kenntnis von $\psi(w)$ und $\psi(w')$ können wir den Wert von ψ auf Anfangsstücken und Endstücken von w sowie ww' berechnen. Man erkennt aus den angegebenen Funktionen, den Beispielen und Übungsaufgaben, dass die Berechnung des Wertes auf Anfangsstücken, Endstücken und Produkten mittels partiell-rekursiver Funktionen möglich ist.
Zuküftig schreiben wir $Prod(x, y, z)$ für $Prod(Prod(x, y), z)$ (dies ist möglich, da $Prod$ assoziativ ist, wie man leicht nachrechnet.)
Wegen $(X \cup \{*\}) \cap Z = \emptyset$ können wir eine Konfiguration (u, z, v) auch als Wort $K = uzv$ angeben, da der Zustand in K eindeutig bestimmt ist. Wir zeigen nun, dass wir die Stelle, an der z steht, mittels partiell-rekursiver Funktionen auch aus $\psi(K)$ berechnen können.
Es sei $f : \mathbb{N}_0 \to \mathbb{N}_0$ die Funktion

$$f(t) = \begin{cases} 0 & \psi^{-1}(t) \in Z \\ 1 & \text{sonst} \end{cases} .$$

f ist partiell-rekursiv (siehe Übungsaufgabe 8). Nun definieren wir die Funktion $g : \mathbb{N}_0 \to \mathbb{N}_0$ durch

$$g(\psi(K)) = (\mu i)[f(Elem(\psi(K), i)) = 0],$$

d.h. für ein Wort $w = x_1 x_2 \dots x_n$ wird der minimale (bei Konfigurationen sogar einzige) Index i mit $x_i \in Z$ bestimmt. Damit ist gezeigt, dass die Stelle in einer Konfiguration w, an der der Zustand z steht, mittels partiell-rekursiver Funktionen ermittelt werden kann.
Wir definieren die folgenden partiell-rekursiven Funktionen:

$$\begin{aligned}
r(x) &= Anfang(x, g(x) \ominus 2), \\
s(x) &= Prod(Elem(x, g(x) \ominus 1), Elem(x, g(x)), Elem(x, g(x)+1)), \\
t(x) &= Ende(x, g(x)+2).
\end{aligned}$$

Für ein Wort $K = u'azbv'$ mit $a, b \in X$, $u', v' \in X^*$ und $z \in Z$, das eine Konfiguration beschreibt, ergeben sich folgende Werte:

$$r(\psi(K)) = \psi(u'), \quad s(\psi(K)) = \psi(azb), \quad t(\psi(K)) = \psi(v').$$

Ferner gilt

$$\psi(K) = Prod(r(\psi(K)), s(\psi(K)), t(\psi(K)).$$

Wir definieren Δ auf der Menge der Kodierungen von Konfigurationen durch

$$\Delta(\psi(K_1)) = \begin{cases} \psi(K_2) & K_1 = azb, a, b \in X, z \in Z, K_1 \models K_2 \\ nicht\ definiert & \text{sonst} \end{cases}.$$

Nach Übungsaufgabe 8 ist Δ partiell-rekursiv.
Damit ist die Konfiguration K', in die K mittels δ überführt wird wie folgt kodiert:

$$\begin{aligned} \psi(K') &= Prod(\psi(u'), \Delta(\psi(azb)), \psi(v')), \\ &= Prod(r(K), \Delta(s(K)), t(K)). \end{aligned}$$

Entsprechende Relationen lassen sich auch herleiten, wenn die Konfiguration nicht durch ein Wort der Form $K = u'azbv'$ beschrieben wird. Insgesamt ergibt sich dann, dass die Funktion $\overline{\Delta}$ mit $\overline{\Delta}(\psi(K)) = \psi(K')$ partiell-rekursiv ist. Damit haben wir gezeigt, dass die Relation $\models$ mittels der partiell-rekursiven $\overline{\Delta}$ simuliert werden kann.
Wir erweitern dies wie folgt auf die Iteration von $\models$, indem wir die Funktion D mittels Rekursionsschema durch

$$\begin{aligned} D(x, 0) &= x, \\ D(x, n+1) &= \overline{\Delta}(D(x, n)) \end{aligned}$$

definieren. Damit gilt $D(\psi(K), n) = \psi(K'')$, wobei K'' die Konfiguration ist, die aus K mittels n-facher direkter Überführung entsteht.

Entsprechend der Definition der TURING-Berechenbarkeit wird einem Wort w das Wort w' zugeordnet, das bei Erreichen einer Endkonfiguration auf dem Band steht. Intuitiv werden also folgende Schritte unternommen:

1. Aus w ergibt sich die Anfangskonfiguration $K_0 = z_0 w$.
2. Aus K_0 wird die Folge der Konfigurationen berechnet bis eine Endkonfiguration $\overline{K}$ vorliegt.
3. Aus $\overline{K}$ ermitteln wir das Wort auf dem Band.

Offenbar ist der Schritt 1 durch eine partiell-rekursive Funktion, die $\psi(w)$ auf $\psi(K_0)$ abbildet, simulierbar. Die Folge der Kodierungen der Konfigurationen ist nach Obigem ebenfalls mittels partiell-rekursiver Funktionen berechenbar. Da die Maschine bei Erreichen der ersten Endkonfiguration stoppt, kann die Endkonfiguration mittels der Funktionen

$$\begin{aligned} Stop_1(\psi(K)) &= \begin{cases} 0 & K \text{ ist Endkonfiguration} \\ 1 & \text{sonst} \end{cases}, \\ Stop_2(\psi(K)) &= (\mu i)[Stop_1(D(K, i)) = 0], \\ Stop_3(\psi(K)) &= D(K, Stop_2(K)) \end{aligned}$$

berechnet werden ($Stop_1$ stellt fest, ob $\psi(K)$ eine Kodierung einer Endkonfiguration ist, $Stop_2$ berechnet die Anzahl der Schritte bis zum Erreichen einer Endkonfiguration bei Start mit K, und $Stop_3$ berechnet dann die Kodierung der zu K gehörigen Endkonfiguration). Unter Verwendung der obigen Ideen ist leicht zu zeigen, dass $Stop_1$ partiell-rekursiv

ist (wir bestimmen zuerst den Zustand in K und testen dann, ob er in Q liegt). Dann sind nach Konstruktion auch $Stop_2$ und $Stop_3$ partiell-rekursiv. Wenn wir nun noch beachten, dass auch der obige dritte Schritt mittels partiell-rekursiver Funktionen simuliert werden kann (wir können ausgehend von $\psi(v_1qv_2)$ sowohl $\psi(v_1), \psi(v_2)$ als auch $\psi(v_1v_2)$ und damit $\psi(v)$ berechnen), ist damit gezeigt, dass die Funktion h, die jeder Zahl $\psi(w)$, $w \in X^*$, den Wert $\psi(f_M(w))$ zuordnet, partiell-rekursiv ist.

Aus diesen Überlegungen resultiert der folgende Satz.

Satz 1.13 *Es seien M eine* TURING*-Maschine und ψ die zugehörige Kodierung. Dann ist die Funktion $f : \mathbb{N}_0 \to \mathbb{N}_0$ mit $f(\psi(w)) = \psi(f_M(w))$ partiell-rekursiv.* □

Fassen wir unsere Ergebnisse über Beziehungen zwischen Berechenbarkeitsbegriffen zusammen, so ergibt sich folgender Satz.

Satz 1.14 *Für eine Funktion f sind die folgenden Aussagen gleichwertig:*

- *f ist durch ein* **LOOP/WHILE***-Programm berechenbar.*
- *f ist partiell-rekursiv.*
- *f ist durch eine Registermaschine berechnenbar.*
- *f ist bis auf Konvertierung der Zahlendarstellung durch eine* TURING*-Maschine berechenbar.*
- *f ist bis auf Konvertierung der Zahlendarstellung durch eine k-Band-*TURING*-Maschine berechenbar.* □

Damit erhalten wir auch die folgende Folgerung.

Folgerung 1.15 *Es gibt Funktionen, die nicht* TURING*-berechenbar sind.* □

Der amerikanische Logiker A. CHURCH hat nun die nach ihm benannte These aufgestellt, dass eine Funktion, die berechenbar in irgendeinem Sinn (der den eingangs formulierten intuitiven Bedingungen genügt) ist, auch TURING-berechenbar (und damit partiell-rekursiv und **LOOP/WHILE**-berechenbar und berechenbar durch Registermaschinen) ist, d.h. dass die von uns hier eingeführten Berechenbarkeitsbegriffe die allgemeinsten sind. Auch alle anderen bisher betrachteten Berechenbarkeiten lieferten tatsächlich nur TURING-berechenbare Funktionen. Daher wird die CHURCHsche These heute allgemein akzeptiert. (Die These kann nicht bewiesen werden, da eine allgemeine Formalisierung des intuitiven Algorithmenbegriffs nicht möglich ist; sie ließe sich aber widerlegen, indem man zeigt, dass bei einer speziellen Formalisierung Funktionen als berechenbar gelten, die nicht TURING-berechenbar sind.)

1.2 Entscheidbarkeit von Problemen

Unter einem Problem (genauer einem Entscheidungsproblem) P verstehen wir im folgenden stets eine Aussageform, d.h. einen Ausdruck $A(x_1, x_2, \ldots, x_n)$, der eine oder mehrere Variable x_i, $1 \leq i \leq n$, enthält und der bei Ersetzen der Variablen x_i durch Elemente a_i aus dem zugehörigen Grundbereich X_i, $1 \leq i \leq n$, in eine Aussage $A(a_1, a_2, \ldots, a_n)$ überführt wird, die den Wahrheitswert „wahr“, repräsentiert durch 1, oder den Wahrheitswert „falsch“, repräsentiert durch 0, annimmt. Wir beschreiben ein Problem im folgenden daher

- durch ein *“Gegeben:“*, das eine Belegung $a_1, a_2, \ldots, a_n$ der Variablen angibt, und
- durch die *“Frage:“* nach der Gültigkeit der Aussage $A(a_1, a_2, \ldots, a_n)$.

Beim *Halteproblem für* TURING-*Maschinen* wird die Aussageform

x stoppt bei Abarbeitung von y.

mit den Variablen x und y betrachtet. Dabei ist x mit einer TURING-Maschine und y mit einem Wort zu belegen. Damit können wir das Halteproblem durch

Gegeben: TURING-Maschine M, Wort w
Frage: Gilt „M stoppt bei Abarbeitung von w“ ?

beschreiben. Offenbar ist die folgende Beschreibung dazu gleichwertig, da bei ihr nur die hinter dem Problem stehende Aussage schon als Frage formuliert wird.

Gegeben: TURING-Maschine M, Wort w
Frage: Stoppt M bei Abarbeitung von w ?

Eine andere Beschreibung des Halteproblems ist durch

Gegeben: TURING-Maschine M, Wort w
Frage: Ist $f_M(w)$ definiert?

gegeben, bei der nur die obige Frage durch eine gleichwertige ersetzt wurde.
Betrachten wir den Ausdruck

x ist eine Primzahl.

so ergeben sich

Gegeben: natürliche Zahl n
Frage: Gilt „n ist eine Primzahl“ ?

und

Gegeben: natürliche Zahl n
Frage: Ist n eine Primzahl?

als mögliche Beschreibung des Problems.

Diese Beschreibung eines Problems ist intuitiv meist die verständlichste, und daher werden wir sie in diesem Abschnitt bevorzugen. Aber sie gestattet kaum eine präzise Fassung des Begriffs der (algorithmischen) Entscheidbarkeit. Daher geben wir noch weitere Formen zur Beschreibung von problemen an, die dies gestatten.

Eine Menge M lässt sich in der Regel durch eine Eigenschaft angeben, die (genau) ihren Elementen zukommt. Formal wird dies durch

$$M = \{x : x \in X \text{ und } A(x)\} \tag{1.3}$$

ausgedrückt, wobei X der Grundbereich ist, dem die Elemente x zu entnehmen sind, und A ein Ausdruck ist, der die Eigenschaft beschreibt. Unter Verwendung dieser Schreibweise lässt sich ein Problem P, das durch den Ausdruck A beschrieben ist, auch als Menge

$$M = \{(a_1, a_2, \ldots, a_n) : a_i \in X_i \text{ für } 1 \leq i \leq n \text{ und } A(a_1, a_2, \ldots, a_n)\}$$

angeben. Wenn die Grundbereiche aus dem Kontext klar sind, lassen wir diese fort.

Für unsere beiden obigen Beispiele ergibt ergeben sich die Mengen

$$M_{halt} = \{(M, w) : f_M(w) \text{ ist definiert}\}$$

und

$$P = \{n : n \text{ ist prim}\}.$$

Für die Grundbereiche ist im Fall der Menge der Primzahlen die Menge $\mathbf{N}$ der natürlichen Zahlen zu wählen. Beim Halteproblem gehen wir zur Bestimmung des Grundbereichs wie folgt vor: Für ein Alphabet X sei M_X die Menge aller TURING-Maschinen mit Eingabealphabet X. Dann muss

$$M_{halt} \subseteq \bigcup_X M_X \times X$$

gefordert werden.

Eine weitere Beschreibung von Problemen kann durch Funktionen vorgenommen werden. Dabei gehen wir von der Mengendarstellung (1.3) aus und definieren die Funktion

$$\varphi_M(x) = \begin{cases} 1 & x \in M \\ 0 & \text{sonst} \end{cases},$$

die charakteristische Funktion der Menge M genannt wird. Für unsere beiden Beispiele ergeben sich (bei Fortlassung der Grundbereiche), über denen die Funktion definiert ist,

$$\varphi_1(M, w) = \begin{cases} 1 & M \text{ stoppt auf } w \\ 0 & \text{sonst} \end{cases}$$

und

$$\varphi_2(n) = \begin{cases} 1 & n \text{ ist Primzahl} \\ 0 & \text{sonst} \end{cases}.$$

Auf diese Weise gehören zu jedem Problem P ein Ausdruck A_P, eine Menge M_P und eine Funktion φ_P mit

$$M_P = \{(a_1, a_2, \ldots, a_n) : A_P(a_1, a_2, \ldots, a_n)\} \quad \text{und} \quad \varphi_P = \begin{cases} 1 & A_P(a_1, a_2, \ldots, a_n) \\ 0 & \text{sonst} \end{cases}.$$

Offenbar beschreiben umgekehrt jede Menge und jede Funktion mit Wertevorrat $\{0,1\}$ auch ein Problem.
Wir werden in den folgenden Ausführungen stets die Beschreibung des Problems so wählen, wie wir es für günstig in dem Zusammenhang halten.

Definition 1.19 *Wir sagen, dass ein Problem P algorithmisch entscheidbar (oder kurz nur entscheidbar) ist, wenn die entsprechend dieser Konstruktion zum Problem gehörende charakteristische Funktion φ_P* TURING*-berechenbar ist. Anderenfalls heißt P (algorithmisch) unentscheidbar.*

Natürlich ist dies gleichwertig zu der Forderung, dass f_P **LOOP**/**WHILE**-berechenbar oder partiell-rekursiv oder durch Registermaschinen berechenbar ist.
Da Probleme als Mengen interpretiert werden können, ist klar, dass wir anstelle der Entscheidbarkeit von Problemen auch die von Mengen definieren können, wofür auch der Begriff der Rekursivität der Menge benutzt wird.

Definition 1.19' *Wir sagen, dass eine Menge M (algorithmisch) entscheidbar (oder rekursiv) ist, wenn die zugehörige charakteristische Funktion φ_M* TURING*-berechenbar ist. Anderenfalls heißt M (algorithmisch) unentscheidbar.*

Offenbar gilt, dass *ein Problem P genau dann entscheidbar ist, wenn die zugehörige Menge M_P entscheidbar ist,* da die zugehörigen charakteristischen Funktionen identisch sind.

Wir wollen noch eine Bemerkung zu dem „sonst“ machen. Wir gehen immer davon aus, dass es eine Universalmenge U gibt, in der die uns interessierenden Elemente und Mengen liegen. Bei Aussagen über wörtern über einem Alphabet X, ist dies die Menge X^*, bei natürlichen Zahlen $\mathbb{N}_0$. Dann ist mit „sonst“ immer das Komplement von M bez. der Universalmenge, also $U \setminus M$ gemeint (für die hier meist vorkommenden Fälle also $X^* \setminus M$ und $\mathbb{N}_0 \setminus M$). Bei der TURING-Berechenbarkeit kann es aber sein, dass die Turing-Maschine, die φ_P bzw. φ_M berechnet, ein Alphabet X' für die Eingaben verwendet, dass X echt enthält. In diesem Fall verlangen wir, dass für alle Wörter, die mindestens einen Buchstaben aus $X' \setminus X$ enthalten, auch eine 0 ausgegeben wird. Diese Forderung lässt sich leicht realisieren, indem die Maschine zuerst testet, ob ein Wort aus X^* auf dem band steht; ist es nicht der Fall, wird eine 0 ausgegeben; ansonsten kehrt die Turing-Maschine zum Wortanfang zurück und arbeitet auf der eingabe aus X^*.
Selbstverständlich können anstelle von 0 und 1 auch zwei anderen feste Elemente aus U genommen werden, falls 0 und/oder 1 nicht zu U gehören.

Bisher haben wir Entscheidungsprobleme behandelt, bei denen der Ausdruck nach Belegung nur einen der beiden Wahrheitswerte annehmen kann. Daneben gibt es natürlich auch noch Berechnungsprobleme, bei denen eine Funktion $f : X \to Y$ gegeben ist und nach dem Wert $f(x)$ für ein gegebenes x gefragt wird. Wir wollen dabei hier annehmen, dass die Funktion f für jedes $x \in X$ definiert ist. (Die einfache Erweiterung auf den Fall partieller Funktionen bleibt dem Leser überlassen.) Formal wird eine Funktion als Menge definiert, und zwar als

$$M_f = \{(x, y) : f(x) = y\}.$$

Dies ist offensichtlich die Mengenbeschreibung des Entscheidungsproblems

Gegeben: $x \in X$ und $y \in Y$
Frage: Nimmt f an der Stelle x den Wert y an?

dessen charakteristische Funktion durch

$$\varphi(x,y) = \begin{cases} 1 & f(x) = y \\ 0 & \text{sonst} \end{cases}$$

gegeben ist. Somit reicht es im Folgenden, nur Entscheidungsprobleme zu betrachten, da Berechnungsprobleme in solche umformuliert werden können.
Als ein Beispiel für ein entscheidbares Problem geben wir das folgende an:

Gegeben: $w \in \{0,1\}$
Frage: Hat w ungerade Länge?

Mittels einer leichtem Modifikation der TURING-Maschine aus Beispiel 1.8 b) lässt sich die TURING-Berechenbarkeit von

$$\varphi_P(w) = \begin{cases} 1 & w \text{ hat ungerade Länge} \\ 0 & \text{sonst} \end{cases}$$

zeigen.
Andererseits folgt aus Obigem und Folgerung 1.2 sofort, dass es ein unentscheidbares Problem gibt. Jedoch scheint das aus Folgerung 1.2 resultierende Problem relativ künstlich zu sein, da die im Beweis von Satz 1.2 konstruierte Funktion auf den ersten Blick keinen praktischen Sinn hat. Daher wollen wir nun ein weiteres unentscheidbares Problem angeben, das (zumindest in gewissen Grenzen) eine Interpretation für Programmiersprachen besitzt.

Satz 1.16 *Das Halteproblem für* TURING*-Maschinen ist unentscheidbar.*

Beweis: Es sei $M = (X, Z, z_0, Q, \delta)$ eine TURING-Maschine. Zur vollständigen Angabe von M reicht es offenbar aus, alle Elemente aus X, alle Elemente aus $Z \setminus Q$ und alle Elemente aus der Menge $\delta \subseteq (Z \setminus Q) \times (X \cup \{*\}) \times Z \times (X \cup \{*\}) \times \{R, L, N\}$ (jede Funktion $f : X \to Y$ kann als Relation $R \subseteq X \times Y$ aufgefasst werden) anzugeben, wenn wir ohne Beschränkung der Allgemeinheit vereinbaren, bei der Angabe der Elemente aus Z mit z_0 anzufangen. (Q lässt sich dann als die Menge der Zustände ermitteln, die in der dritten Komponente von δ, aber nicht in $Z \setminus Q$ vorkommen.) Es seien $X = \{x_1, x_2, \ldots, x_n\}$, $Z = \{z_0, z_1, \ldots, z_m\}$ und $Q = \{z_{k+1}, z_{k+2}, \ldots, z_m\}$. Wir setzen $x_0 = *$. Für $0 \leq i \leq k$ und $0 \leq j \leq n$ setzen wir ferner $\delta_{ij} = (z_i, x_j, z_{ij}, x_{ij}, r_{ij})$, falls $\delta(z_i, x_j) = (z_{ij}, x_{ij}, r_{ij})$ gilt. Damit lässt sich M durch

$$x_1, x_2, \ldots, x_n, z_0, z_1, \ldots, z_k, \delta_{00}, \delta_{01}, \ldots, \delta_{0n}, \delta_{10}, \delta_{11}, \ldots, \delta_{1n}, \ldots, \delta_{kn}$$

beschreiben. Um aus dieser Beschreibung ein Wort zu erhalten, betrachten wir die Kodierung, die durch folgende eineindeutige Zuordnung gegeben ist:

$$\begin{aligned}
&x_j \to 01^{j+1}0 \quad \text{für } 0 \leq j \leq n, \\
&z_i \to 01^{i+1}0^2 \quad \text{für } 0 \leq i \leq k, \\
&R \to 010^3, \quad L \to 01^20^3, \quad N \to 01^30^3, \\
&(\to 010^4, \quad) \to 01^20^4, \\
&, \to 010^5.
\end{aligned}$$

Man beachte, dass durch die letzten drei Zuordnungen den zur Beschreibung eines Quintupels δ_{ij} notwendigen Zeichen „Klammer auf“, „Klammer zu“ und „Komma“ jeweils ein Wort zugeordnet wird. Durch diese Kodierung wird es möglich, M durch ein Wort über $\{0,1\}$ zu beschreiben.
Wir illustrieren diese Konstruktion anhand der TURING-Maschine aus Beispiel 1.8 b). Zuerst erhalten wir (mit $x_1 = a, x_2 = b, z_2 = q$) die Beschreibung

$$a, b, z_0, z_1, (z_0, *, z_0, *, N), (z_0, a, z_1, a, R), (z_0, b, z_1, b, R), (z_1, *, q, *, N),$$
$$(z_1, a, z_0, a, R), (z_1, b, z_0, b, R)$$

und nach der Kodierung mittels

$$* \to 010, a \to 01^20, b \to 01^30, z_0 \to 010^2, z_1 \to 01^20^2, q \to 01^30^2,$$
$$R \to 010^3, L \to 01^20^3, N \to 01^30^3, (\to 010^4,) \to 01^20^4, , \to 010^5$$

die Beschreibung durch das Wort

$$01^20010^501^30010^5010^2010^501^20^2010^5$$
$$010^4010^2010^5010010^5010^2010^5010010^501^30^301^20^4010^5$$
$$010^4010^2010^501^20010^501^20^2010^501^20010^5010^301^20^4010^5$$
$$010^4010^2010^501^30010^501^20^2010^501^30010^5010^301^20^4010^5$$
$$010^401^20^2010^5010010^501^30^2010^5010010^501^30^301^20^4010^5$$
$$010^401^20^2010^501^20010^5010^2010^501^20010^5010^301^20^4010^5$$
$$010^401^20^2010^501^30010^5010^2010^501^30010^5010^301^20^4.$$

Mit $\mathcal{S}$ bezeichnen wir die Menge aller TURING-Maschinen $M = (X, Z, z_0, Q, \delta)$ mit $X = \{0,1\}$, $Z = \{z_0, z_1, \ldots, z_m\}$ und $Q = \{z_m\}$ für ein $m \geq 1$ (wegen Lemma 1.9 können wir ohne Beschränkung der Allgemeinheit annehmen, dass Q einelementig ist). Für eine TURING-Maschine $M \in \mathcal{S}$ sei w_M das Wort, das M nach obiger Kodierung beschreibt. M bestimmt w_M eindeutig, und ist umgekehrt $w \in \{0,1\}^*$ die Beschreibung einer TURING-Maschine aus $\mathcal{S}$, so ist die TURING-Maschine $M \in \mathcal{S}$ mit $w = w_M$ eindeutig bestimmt. Ferner ist die Kodierung w_M von $M \in \mathcal{S}$ eine mögliche Eingabe für die TURING-Maschine M.

Hilfssatz 1. Das Problem

Gegeben:	$w \in \{0,1\}^*$
Frage:	Ist w Kodierung einer TURING-Maschine aus S ?

ist entscheidbar.

Wir geben nur die Idee des Beweises, die Realisierung der Idee durch eine formale TURING-Maschine ist aufwendig und bleibt dem Leser überlasssen.
Um festzustellen, ob das Eingabealphabet der TURING-Maschine aus $x_1 = 0$ und $x_2 = 1$ besteht und Z mindestens zwei Zustände enthält, ist nur zu testen, ob w das Anfangsstück $01^20010^501^30010^5010^2010^5$ hat. Dann wird getestet, ob nach diesem Anfang (von der Kodierung der Kommas abgesehen) Kodierungen von Zuständen und Quadrupeln δ_{ij} folgen und ob die in den Quadrupeln auftauchenden Eingabesymbole und Zustände auch

in X bzw. Z vorhanden sind. Abschließend wird getestet, ob für jedes Paar $(z_i, x_j), z_i \in Z \setminus Q, x_j \in X \cup \{*\}$, ein Quintupel δ_{ij} existiert.
Wir betrachten nun die Funktion $f : \{0,1\}^* \to \{0,1\}$, die durch

$$f(w) = \begin{cases} 0 & w = w_M \text{ für ein } M \in \mathcal{S},\ f_M(w_M) \text{ ist nicht definiert} \\ \text{nicht definiert} & \text{sonst} \end{cases}$$

gegeben ist.

Hilfssatz 2. f ist nicht TURING*-berechenbar.*

Beweis: Wir führen den Beweis indirekt, d.h. wir nehmen an, dass f TURING-berechenbar ist und leiten einen Widerspruch her.
Wenn f TURING-berechenbar ist, so gibt es nach Definition eine TURING-Maschine N mit $f_N = f$. Da offensichtlich $N \in \mathcal{S}$ gilt, existiert eine Kodierung w_N von N.
Wenn für die Kodierung w_N von N der Wert $f(w_N)$ definiert ist, so folgt aus der Definition von f, dass $f_N(w_N)$ nicht definiert ist. Dies widerspricht aber $f = f_N$.
Ist dagegen $f(w_N)$ nicht definiert, so besagt die Definition von f gerade, dass $f_N(w_N)$ definiert sein muss. Damit erhalten wir erneut einen Widerspruch zu $f = f_N$.

Da es nach Hilfssatz 1 entscheidbar ist, ob ein Wort $w \in \{0,1\}^*$ eine Kodierung einer TURING-Maschine ist, kann es nicht entscheidbar sein, ob $f_M(w_M)$ definiert ist oder nicht, da sonst die Funktion aus Hilfssatz 2 TURING-berechenbar wäre. Damit ist die Behauptung von Satz 1.16 bewiesen (es ist sogar noch mehr gezeigt worden, da nicht beliebige Wörter x sondern nur Kodierungen von TURING-Maschinen betrachtet wurden). □

Satz 1.17 *Das Problem*

Gegeben: **LOOP/WHILE**-*Programm* Π, $n \in \mathbf{N}$
Frage: *Ist* $\Phi_{\Pi,1}(n)$ *definiert?*

ist unentscheidbar.

Beweis: Zu jeder TURING-Maschine M können wir entsprechend den Beweisen von Satz 1.13 und Satz 1.6 ein **LOOP/WHILE**-Programm Π mit $\psi(f_M(w)) = \Phi_{\Pi,1}(\psi(w))$ konstruieren. Die Funktion ψ aus dem Beweis von Satz 1.13 und ihre Umkehrung sind TURING-berechenbar. Wäre nun das Problem aus Satz 1.17 entscheidbar, so wäre auch entscheidbar, ob $\psi(f_M(w))$ und damit $f_M(w)$ definiert ist oder nicht. Dies widerspricht aber Satz 1.16. □

Wir merken an, dass die Aussage von Satz 1.17 wie folgt gedeutet werden kann: Sind in einer Programmiersprache Konstrukte vorhanden, die der **LOOP**- bzw. **WHILE**-Anweisung entsprechen, so kann für ein beliebiges Programm nicht entschieden werden, ob es bei einer beliebigen Eingabe ein Resultat liefert.

Definition 1.20 *i) Zwei* TURING*-Maschinen M_1 und M_2 heißen äquivalent, wenn $f_{M_1} = f_{M_2}$ gilt.*
ii) Zwei **LOOP/WHILE**-*Programme Π_1 und Π_2 heißen äquivalent, wenn $\Phi_{\Pi_1,1} = \Phi_{\Pi_2,1}$ gilt.*

Es ist leicht zu sehen, dass diese beiden Äquivalenzen die Eigenschaften einer Äquivalenzrelation erfüllen.

Es sei $\mathcal{M}$ eine Menge, in der eine Äquivalenz erklärt ist. Das *Äquivalenzproblem* für Elemente aus $\mathcal{M}$ ist durch

Gegeben: zwei Elemente A_1 und A_2 aus $\mathcal{M}$
Frage: Sind A_1 und A_2 äquivalent?

gegeben.

Satz 1.18 *Das Äquivalenzproblem für* TURING*-Maschinen bzw.* **LOOP/WHILE***-Programme ist unentscheidbar.*

Beweis: Wir geben den Beweis nur für TURING-Maschinen. Die Übertragung auf den Fall der **LOOP/WHILE**-Programme erfolgt analog zum Beweis von Satz 1.17.
Es seien eine TURING-Maschine M und ein Wort w über dem Eingabealphabet von M gegeben. Wir konstruieren zunächst in Abhängigkeit von M und w die TURING-Maschinen M_1 und N, deren induzierte Funktionen f_{M_1} und f_N durch

$$f_{M_1}(v) = \begin{cases} 1 & v = w \\ \text{nicht definiert} & \text{sonst} \end{cases}$$

und

$$f_N(v) = \begin{cases} v & f_M(v) \text{ ist definiert} \\ \text{nicht definiert} & \text{sonst} \end{cases}$$

gegeben sind. (f_{M_1} ist nach Übungsaufgabe 9 aus Abschnitt 1.1 partiell-rekursiv, und wegen Satz 1.14 gibt es daher eine solche TURING-Maschine M_1; N ergibt sich aus M durch folgende Modifikation: zuerst kopiert N das Eingabewort v auf das Band, arbeitet auf v wie M, und falls ein Endzustand erreicht wird, wird das erhaltene Resultat $f_M(v)$ gelöscht, so dass auf dem Band nur noch die Kopie von v steht.)
Ferner können wir nun eine TURING-Maschine M_2 konstruieren, die die Komposition von f_N und f_{M_1} als induzierte Funktion besitzt (da aus partiell-rekursiven Funktionen durch Komposition wieder nur partiell-rekursive und damit TURING-berechenbare Funktionen entstehen). Offenbar gilt

$$f_{M_2}(v) = f_{M_1}(f_N(v)) = \begin{cases} 1 & v = w \text{ und } f_M(w) \text{ ist definiert} \\ \text{nicht definiert} & \text{sonst} \end{cases} .$$

Damit gilt $f_{M_1} = f_{M_2}$ genau dann, wenn $f_M(w)$ definiert ist. Wenn die Äquivalenz von M_1 und M_2 entscheidbar wäre, so wäre auch entscheidbar, ob $f_M(w)$ definiert ist. Wegen Satz 1.16 ist daher das Äquivalenzproblem für TURING-Maschinen unentscheidbar. □

Auch hier ist wieder festzustellen, dass eine Interpretation von Satz 1.18 dahingehend möglich ist, dass es unentscheidbar ist, ob zwei Programme die gleichen Abbildung der Eingaben in Ausgaben realisieren.

Bei den Beweisen von Satz 1.17 und 1.18 wurde die gleiche Methode benutzt. Es erfolgte eine Reduktion des zu betrachtenden Problems auf ein Problem, dessen Unentscheidbarkeit bereits gezeigt wurde, in der Weise, dass aus der Entscheidbarkeit des betrachteten Problems auch die des unentscheidbaren Problems folgen würde. Diese Methode ist die am meisten benutzte, um Unentscheidbarkeiten zu zeigen.

Im Folgenden wollen wir zuerst zwei weitere Probleme angeben, die unentscheidbar sind und in natürlicher Weise entstehen. Hierbei werden wir auif die Beweise der Unentscheidbarkeit aus Platzgründen verzichten. Unter Verwendung des zweiten dieser Probleme zeigen wir dann die Unentscheidbarkeit von Problemen der Prädikatenlogik.

Im Jahre 1900 hielt der deutsche Mathematiker DAVID HILBERT auf dem Internationalen Mathematikerkongress einen Hauptvortrag, in dem er 23 Probleme vorstellte, die nach seiner Meinung von besonders großer Bedeutung für die Mathematik waren. Das 10. Problem lautet:

Gegeben:	eine natürliche Zahl $n \geq 1$, ein Polynom $p(x_1, x_2, \ldots, x_n) = \sum c_{i_1 i_2 \ldots i_n} x_1^{i_1} x_2^{i_2} \ldots x_n^{i_n}$ in n Variablen mit ganzzahligen Koeffizienten
Frage:	Gibt es eine Lösung von $p(x_1, x_2, \ldots, x_n) = 0$ in $\mathbf{Z}^n$?

Zum Beispiel hat

$$p(x, y, z) = 3xyz^2 + 5xy^2 - 4x^2yz = 0$$

die Lösung $x = 2, y = 1, z = 1$, während

$$p(x, y, z) = 2x^4y^2 + 3x^2z^2 + 2y^2z^6 - 1 = 0$$

keine ganzzahlige Lösung besitzt (da geradezahlige Potenzen von ganzen Zahlen stets nichtnegative ganze Zahlen und somit die ersten drei Summanden 0 oder ≥ 2 sind).
Genauer gesagt, HILBERT fragte nach einem Algorithmus zur Lösung des eben genannten Problems. In unserer Terminologie stellte er die Frage nach der Entscheidbarkeit des Problems. Die Lösung des Problems wurde nach Vorarbeiten von ROBINSON im Jahre 1960 vom JU.V.MATIJASEVIC gegeben.

Satz 1.19 *Das 10.* HILBERT*sche Problem ist unentscheidbar.* □

Entsprechend diesem Ergebnis gibt es keinen Algorithmus, der für alle Polynome die richtige Antwort gibt. Auf der anderen Seite gibt es natürlich Teilmengen, die nur spezielle Polynome enthalten, für die es dann Algorithmen gibt. Wir erwähnen hier zwei solche Fälle.

- Wir beschränken die Menge der Polynome, indem wir Linearität fordern, d.h. die Polynome sind von der Form

  $$p(x_1, x_2, \ldots, x_n) = a_0 + a_1x_1 + a_2x_2 + \ldots + a_nx_n.$$

  Dann gibt es genau dann eine ganzzahlige Lösung, wenn der größte gemeinsame Teiler d der Koeffizienten $a_1, a_2, \ldots, a_n$ ein Teiler von a_0 ist. (Es sei zuerst $(b_1, b_2, \ldots, b_n) \in \mathbf{Z}^n$ eine Lösung. Dann ist d ein Teiler von $a_1b_1 + a_2b_2 + \ldots + a_nb_n$. Wegen $-a_0 = a_1b_1 + a_2b_2 + \ldots + a_nb_n$ ist d damit ein Teiler von a_0. Umgekehrt gibt es für den größten gemeinsamen Teiler d von $a_1, a_2, \ldots, a_n$ eine Darstellung der Form $d = a_1c_1 + a_2c_2 + \ldots + a_nc_n$ mit gewissen ganzen Zahlen $c_1, c_2, \ldots, c_n$. Falls $a_0 = kd$, dann ist $(kc_1, kc_2, \ldots, kc_n)$ eine Lösung.) Da der größte gemeinsame Teiler nach dem EUKLIDischen Algorithmus bestimmt werden kann und es entscheidbar ist, ob eine ganze Zahl Teiler einer anderen ganzen Zahl ist, ist es auch entscheidbar, ob ein Polynom der obigen speziellen Art eine Nulstelle in $\mathbf{Z}^n$ hat.

- Wir betrachten nur Polynome in einer Variablen, d.h. Polynome der Form

  $$p(x) = a_0 + a_1x + a_2x^2 + \ldots + a_nx^n.$$

Wenn $-a_0 = a_1x + a_2x^2 + \ldots + a_nx^n$ gelten soll, muss offenbar x ein Teiler von a_0 sein. Da es nur endlich viele Teiler von a_0 gibt, lässt sich mittels Durchtesten aller dieser Teiler feststellen, ob einer von ihnen Nullstelle von p ist. Dies liefert offensichtlich einen Algorithmus zur Beantwortung, ob p eine ganzzahlige Nullstelle hat.

Wir betrachten nun das POST*sche Korrespondenzproblem*:

Gegeben:	Alphabet X mit mindestens zwei Buchstaben, $n \geq 1$, Menge $\{(u_1, v_1), (u_2, v_2), \ldots, (u_n, v_n)\}$ von Paaren mit $u_i, v_i \in X^+$ für $1 \leq i \leq n$
Frage:	Gibt es eine Folge $i_1 i_2 \ldots i_m$ mit $1 \leq i_j \leq n$ für $1 \leq j \leq m$ derart, dass $u_{i_1} u_{i_2} \ldots u_{i_m} = v_{i_1} v_{i_2} \ldots v_{i_m}$ gilt?

Beispiel 1.10 a) Es seien $n = 3$, $X = \{a, b, c\}$ und die Menge $\{(aa, a), (bc, ab), (c, cca)\}$ von Paaren gegeben. Dann ist $i_1 i_2 i_3 i_4 = 1231$ eine Folge der gesuchten Art, denn es gilt

$$u_1u_2u_3u_1 = aa \cdot bc \cdot c \cdot aa = a \cdot ab \cdot cca \cdot a = v_1v_2v_3v_1.$$

b) Für $n = 2$, $X = \{a, b\}$ und die Menge $\{(aab, aa), (ab, ba)\}$ von Paaren gibt es dagegen keine derartige Folge. Dies ist wie folgt zu sehen: Wegen $|u_1| > |v_1|$ und $|u_2| = |v_2|$ kann die Folge keine 1 enthalten, und die Folge kann nicht nur aus dem Symbol 2 bestehen, da u_2 mit a und v_2 mit b anfängt.

Zur Motivation des POSTschen Korrespondenzproblems betrachten wir zwei Kodierungen ϕ_1 und ϕ_2 der Menge $\{1, 2, \ldots, n\}$, die in der Abbildung 1.12 gegeben sind.

	ϕ_1		ϕ_2	
u_1	$\leftarrow$	1	$\rightarrow$	v_1
u_2	$\leftarrow$	2	$\rightarrow$	v_2
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
u_n	$\leftarrow$	n	$\rightarrow$	v_n

Abbildung 1.12:

Dann gelten

$$\phi_1(i_1 i_2 \ldots i_m) = u_{i_1} u_{i_2} \ldots u_{i_m} \quad \text{und} \quad \phi_2(i_1 i_2 \ldots i_m) = v_{i_1} v_{i_2} \ldots v_{i_m}.$$

Folglich ist die Frage des POSTschen Korrespondenzproblems damit gleichwertig, zu fragen, ob eine Folge von Elementen aus $\{1, 2, \ldots, n\}$ existiert, die bei beiden Kodierungen ϕ_1 und ϕ_2 auf das gleiche Wort abgebildet wird.

Satz 1.20 *Das* POST*sche Korrespondenzproblem ist unentscheidbar.* □

Für die Diskussion von Spezialfällen verweisen wir auf die Übungsaufgaben.
Wir werden im Laufe dieser Vorlesung mehrere Probleme als unentscheidbar nachweisen, indem wir die Unentscheidbarkeit des Postschen Korrespondenzproblems ausnutzen.
Abschließend geben wir zwei unentscheidbare Probleme der Prädikatenlogik an.

Satz 1.21 *i) Es ist unentscheidbar, ob ein prädikatenlogischer Ausdruck eine Tautologie ist.*
ii) Es ist unentscheidbar, ob ein prädikatenlogischer Ausdruck erfüllbar ist.

Übungsaufgaben

1. Welche Funktionen werden durch die nachfolgenden Programme berechnet?

   a) $x_2 := P(x_2);\ x_2 := P(x_2);\ x_2 := P(x_2);$
   WHILE $x_2 \neq 0$ **BEGIN**
   LOOP x_1 **BEGIN** $x_3 := S(x_3)$ **END**;
   $x_2 := P(x_2)$
   END;
   $x_1 := x_3$

   b) $x_2 := x_1;$
   LOOP x_1 **BEGIN** $x_3 := P(x_3)$ **END**;
   WHILE $x_3 \neq 0$ **BEGIN** $x_1 := x_3; x_3 := 0$ **END**

2. Berechnen Sie, welchen Wert die Variable x_1 nach Abarbeitung des folgenden Programms bei gegebener Eingabe x_1 annimmt.

   $x_2 := S(x_1);\ x_3 := 0;\ x_4 := x_1;$
   WHILE $x_1 \neq 0$ **BEGIN**
   $x_1 := P(x_1); x_1 := P(x_1); x_2 := P(x_2); x_2 := P(x_2)$
   END;
   WHILE $x_2 \neq 0$ **BEGIN** $x_3 := S(x_3); x_2 := P(x_2)$ **END**;
   WHILE $x_3 \neq 0$ **BEGIN**
   LOOP x_4 **BEGIN** $x_4 := S(x_4)$ **END**;
   $x_3 := P(x_3)$
   END;
   $x_1 := x_4$

3. Bestimmen sie Funktionen $\Phi_{\Pi,i}$ für $i \in \{1,2,3\}$ und das folgende Programm Π.

   $x_2 := S(x_1);\ x_3 := x_1;$
   WHILE $x_1 \neq 0$ **BEGIN**
   $x_1 := P(x_1);\ x_1 := P(x_1);\ x_2 := P(x_2); x_1 := P(x_1)$
   END;
   WHILE $x_2 \neq 0$ **BEGIN**
   LOOP x_3 **BEGIN** $x_3 := S(x_3)$ **END**;
   $x_2 := P(x_2)$
   END

4. Konstruieren Sie **LOOP**/**WHILE**-Programme für folgende Funktionen:

   a) $f(x_1) = 2^{x_1}$,

   b) $f(x_1) = x_1^a$, wobei a eine feste natürliche Zahl ist.

5. Zeigen Sie, dass die Funktionen $subt : \mathbb{N}_0^2 \to \mathbb{N}_0$, $sg : \mathbb{N}_0 \to \mathbb{N}_0$ und $\overline{sg} : \mathbb{N}_0 \to \mathbb{N}_0$, die durch

$$\begin{aligned} subt(x_1, x_2) &= \begin{cases} x_1 - x_2 & \text{für } x_1 \geq x_2 \\ 0 & \text{sonst} \end{cases} \\ sg(x_1) &= \begin{cases} 1 & \text{für } x_1 > 0 \\ 0 & \text{sonst} \end{cases} \\ \overline{sg}(x_1) &= \begin{cases} 0 & \text{für } x_1 > 0 \\ 1 & \text{sonst} \end{cases} \end{aligned}$$

definiert sind, **LOOP**-berechenbar sind, indem Sie jeweils ein **LOOP**-Programm (d.h. ohne Benutzung des **WHILE**-Befehls) angeben, dass die Funktion induziert.

6. Geben Sie **LOOP/WHILE**-Programme für folgende Konstrukte aus Programmiersprachen an:

   a) **IF** $x_2 > 2$ **THEN** $x_1 := x_1 + x_2$ **ELSE** $x_1 := 0$,

   b) **FOR** $i = 10$ **TO** 20 **DO** $x_1 := i * x_1$.

7. Es seien, a_1, a_2, a_3, b_1, b_2 und B_3 Zahlen aus $\mathbb{N}$, wobei a_1, a_2 und a_3 paarweise verschieden sind. Zeigen Sie, dass die Funktion $f : \mathbb{N}_0 \to \mathbb{N}_0$ mit

$$f(x_1) = \begin{cases} b_i & \text{für } x_1 = a_i, 1 \leq i \leq 3 \\ 0 & \text{sonst} \end{cases}$$

   durch ein **LOOP**-Programm berechnet werden kann.

8. Zeigen Sie, dass die Funktionen $\text{div} : \mathbb{N}_0^2 \to \mathbb{N}_0$ und $\text{mod} : \mathbb{N}_0^2 \to \mathbb{N}_0$ vermöge

$$(x, y) \mapsto \text{div}(x, y) = \left\lfloor \frac{x}{y} \right\rfloor \qquad \text{sowie} \qquad (x, y) \mapsto \text{mod}(x, y) = x - \left\lfloor \frac{x}{y} \right\rfloor \cdot y$$

   LOOP/WHILE-berechenbar sind.

   Sind die angegebenen Funktionen div und mod (mit eventuellen Änderungen) auch **LOOP**-berechenbar?

9. Beweisen Sie, dass eine Funktion $f : \mathbb{N}_0 \to \mathbb{N}_0$, die nur an endlich vielen Stellen definiert ist, **LOOP/WHILE**-berechenbar ist.

10. Beweisen Sie, dass es für jede Funktion $f : \mathbb{N}_0^n \to \mathbb{N}_0$ eine natürliche Zahl t derart existiert, dass es für jede natürliche Zahl $t' \geq t$ ein Programm $\Pi_{f,t'}$ so gibt, dass

$$\Phi_{\Pi_{f,t'},1} = f \text{ und } t(\Pi_{f,t'}) = t$$

    gelten.

11. Es sei $t \geq 1$.

    a) Beweisen Sie, dass jedes **LOOP/WHILE**-Programm der Tiefe t höchstens $2t$ Variable benutzt.

    b) Bestimmen Sie die Anzahl der **LOOP/WHILE**-Programme der Tiefe t, die genau die Variablen $x_1, x_2, \ldots x_{2t}$ benutzen.

12. Zeigen Sie, dass folgende Funktionen primitiv-rekursiv sind:

    a) $pot(x, y) = x^y$,

    b) $f(x, y) = \begin{cases} 1 & x = y \\ 0 & \text{sonst} \end{cases}$.

13. a) Man zeige, dass es zu jeder natürlichen Zahl $n \geq 1$ eine totale n-stellige Funktion von $\mathbb{N}_0^n$ in $\mathbb{N}_0$ gibt, die nicht **LOOP/WHILE**-berechenbar ist. (In der Vorlesung wurde nur gezeigt, dass es ein- und zweistellige Funktionen gibt, die nicht **LOOP/WHILE**-berechenbar sind.)

    b) Zeigen Sie, dass jede nullstellige Funktion in $\mathbb{N}_0$ **LOOP**-berechenbar ist.

14. Geben Sie die primitiv-rekursiven Funktionen f und g an, die folgendermaßen definiert sind:

$$\begin{aligned} f(0) &= S(Z) \\ f(y+1) &= P(P_2^2(y, f(y))) \end{aligned}$$

$$\begin{aligned} g(0) &= Z \\ g(y+1) &= f(P_2^2(y, g(y))) \end{aligned}$$

15. Welche Funktion wird durch das Schema

$$\begin{aligned} f(0) &= 1, \\ f(n+1) &= h(n, f(n)) = f(n) + pot(2, n) \end{aligned}$$

    berechnet, wobei

$$h(x, y) = y + pot(2, x) \text{ und } pot(x, y) = x^y.$$

16. Die Ackermann-Funktion $A : \mathbf{N}^2 \to \mathbf{N}$ sei durch

$$\begin{aligned} A(0, y) &= y + 1, \\ A(x+1, 0) &= A(x, 1), \\ A(x+1, y+1) &= A(x, A(x+1, y)) \end{aligned}$$

    definiert. Zeigen Sie

    a) $A(1, y) = y + 2$,

    b) $A(2, y) = 2y + 3$,

    c) $A(3, y) > 2^{y+1}$.

    (Wir erwähnen hier, dass die Ackermann-Funktion eine totale Funktion ist, die nicht primitiv-rekursiv ist.)

17. Beweisen Sie folgende Aussagen:
a) Eine totale Funktion, die nur an endlich vielen Stellen einen von 0 verschiedenen Wert annimmt, ist partiell-rekursiv.
b) Seien N eine endliche Menge und $f : N \to N'$ eine totale Funktion. Dann sind die Funktionen f' und f'' mit

$$f'(x) = \begin{cases} 0 & x \in N \\ 1 & \text{sonst} \end{cases}$$

und

$$f''(x) = \begin{cases} f(x) & x \in N \\ nicht\ definiert & \text{sonst} \end{cases}$$

partiell-rekursiv.

18. Es sei $g : \mathbb{N}_0 \to \mathbb{N}_0$ eine partiell-rekursive Funktion. Zeigen Sie, dass die Funktion $f : \mathbb{N}_0 \to \mathbb{N}_0$ mit

$$f(x) = \begin{cases} g(x) & \text{falls } g(x) \text{ definiert und } g(x) > 0, \\ \text{nicht definiert} & \text{sonst} \end{cases}$$

ebenfalls partiell-rekursiv ist.

19. Beweisen Sie, dass die Funktion $f\colon \mathbb{N}_0 \to \mathbb{N}_0$ mit

$$f(x) = \begin{cases} x + 2 & \text{für } x > 0 \\ 0 & \text{sonst} \end{cases}$$

primitiv-rekursiv ist!

20. Es sei $g\colon \mathbb{N}_0 \to \mathbb{N}_0$ eine primitiv-rekursive Funktion. Zeigen Sie, dass dann auch die Funktion f, definiert durch

$$f(n_1, n_2) = \begin{cases} n_1 & \text{für } n_2 = 0 \\ \underbrace{g(g(\ldots g(}_{n_2-\text{mal}} n_1)\ldots)) & \text{sonst} \end{cases},$$

primitiv-rekursiv ist.

21. a) Es sei $g(x) = x \bmod 2$. Bestimmen Sie die Funktion $h(x) = (\mu y)[add(g(P_1^2(x, y)), P_2^2(x, y))]$.
b) Zeigen Sie, dass die Funktion

$$geq(x, y) = \begin{cases} 1 & \text{falls } x \geq y, \\ 0 & \text{sonst,} \end{cases}$$

primitiv-rekursiv ist und bestimmen Sie $(\mu y)[\overline{sg}(geq(pot(m + 2, y), n + 1))]$.

22. Gegeben sei die Registermaschine mit dem folgenden Programm

```
 1  CLOAD 0
 2  STORE 2
 3  LOAD 2
 4  MULT 2
 5  STORE 3
 6  LOAD 1
 7  SUB 3
 8  IF c_o = 0 GOTO 12
 9  LOAD 2
10  CADD 1
11  GOTO 2
12  END
```

a) Geben Sie die Folge der Konfigurationen, die bei Abarbeitung des Programms beginnend mit $(1, 0, 15, 0, 0, \ldots)$ entsteht. (Beschränkung auf die ersten vier Speicherregister ist möglich).

b) Bestimmen Sie die von der Registermaschine berechnete einstellige Funktion.

23. Es sei die Registermaschine mit dem Programm

```
 1  LOAD 1                 7  STORE 2
 2  CSUB 3                 8  GOTO 1
 3  IF c_0 = 0 GOTO 9      9  LOAD 2
 4  STORE 1               10  STORE 1
 5  LOAD 2                11  END
 6  CADD 1
```

a) Bestimmen Sie die folge der Konfigurationen für den Eingabewert 5 (es reicht eine Beschränkung auf die ersten fünf Komponenten).

b) Welche einstellige Funktion wird durch die Registermaschine induziert.

24. Bestimmen Sie die von den Registermaschinen mit den folgenden Programmen berechneten zweistelligen Funktionen und beschreiben Sie die berechneten Funktionen durch Konstrukte einer Programmiersprache.

a)
```
 1  LOAD 2
 2  IF c_0 = 0 GOTO 10
 3  LOAD 1
 4  CADD 1
 5  STORE 1
 6  LOAD 2
 7  CSUB 1
 8  STORE 2
 9  GOTO 2
10  END
```

b)
```
 1  LOAD 2
 2  CSUB 5
 3  IF c_0 = 0 GOTO 8
 4  LOAD 2
 5  MULT 1
 6  STORE 1
 7  GOTO 11
 8  LOAD 2
 9  ADD 1
10  STORE 1
11  END
```

25. Geben Sie Registermaschinen an, die die gleichen Funktionen berechnen wie die folgenden Konstrukte der Programmiersprache C :

a)
```
if      (x[2] <= 5)
        x[1] = x[1] + x[2] ;
else
        x[1] = x[1] * x[2] ;
```

b)
```
for     (x[1] = 10; x[1] <= 20; x[1] = x[1] + 1)
        x[2] = x[2] + x[1] ;
```

c)
```
while   (x[1] > 0)
        { x[2] = x[2] + x[1]; x[1] = x[1] - 1 ; }
```

26. Geben Sie eine Registermaschine an, die entscheidet. ob eine gegebene Zahl eine Quadratzahl ist.

27. Welche einstellige (!) Funktion wird durch eine Registermaschine mit dem folgenden Programm berechnet?

```
 1  LOAD 1                 8  LOAD 2
 2  IF c_0 = 0 GOTO 11     9  SUB 3
 3  SUB 3                 10  GOTO 2
 4  STORE 2               11  LOAD 3
 5  LOAD 3                12  STORE 1
 6  CADD 1                13  END
 7  STORE 3
```

28. Konstruieren Sie eine Registermaschine für die zweistellige Funktion, die das **LOOP/WHILE**-Programm

$$\textbf{WHILE } x_2 \neq 0 \textbf{ BEGIN } x_1 := S(x_1);\ x_2 := P(x_2) \textbf{ END}$$

bei der ersten Variablen berechnet.

29. Man zeige, dass es zu jeder Registermaschine M eine Registermaschine M' gibt, die folgende Bedingungen erfüllt:
– $f_{M'} = f_M$,

– Das Programm von M' enthält nur einen Stoppbefehl.
– Wenn das Programm von M' aus r Befehlen besteht, so ist `END` der r-te Befehl (d.h. der einzige Stoppbefehl steht am Ende des Programms.

30. Bestimmen Sie für die Turing-Maschine $M = (\{z_0, z_1, z_2, z_3, q\},\ \{a, b\},\ z_0,\ \{q\},\ \delta)$, wobei δ durch

δ	z_0	z_1	z_2	z_3
$*$	$(q, *, N)$	$(q, *, N)$	$(z_2, *, N)$	$(z_0, *, R)$
a	(z_0, a, R)	(z_2, b, L)	(z_2, a, N)	(z_3, a, L)
b	(z_1, b, R)	(z_1, b, R)	(z_3, a, L)	(z_3, b, L)

gegeben ist,
a) $f_M(abba)$, $f_M(bbaa)$ und $f_M(aabb)$,
b) die von M induzierte Funktion f_M.

31. Es sei $M = (\{a, b\}, \{z_0, z_1, z_2, z_3, q\}, z_0, \{q\}, \delta)$ eine TURING–Maschine, bei der die Funktion δ durch folgende Tabelle gegeben ist.

δ	z_0	z_1	z_2	z_3
$*$	$(z_2, *, L)$	$(q, *, N)$	$(q, *, N)$	$(q, *, N)$
a	(z_1, a, R)	(z_0, a, R)	(z_3, a, L)	(z_2, b, L)
b	(z_1, a, R)	(z_0, b, R)	(z_3, b, L)	(z_2, b, L)

a) Bestimmen Sie $f_M(abaabb)$.
b) Bestimmen Sie die induzierte Funktion $f_M \colon \{a, b\}^* \to \{a, b\}^*$.

32. Durch die beiden folgenden Tabellen sei jeweils eine TURING-Maschine beschrieben:

a)

	z_0	z_1
$*$	$(z_0, *, N)$	$(q, *, N)$
a	(z_1, a, R)	(z_0, a, R)
b	(z_1, b, R)	(z_0, b, R)

b)

	z_0	z_a^1	z_b^1	z_a^2	z_b^2	z_1	z_2	z_3
$*$	$(q, *, N)$	$(z_a^2, *, R)$	$(z_b^2, *, R)$	(z_1, a, L)	(z_1, b, L)	$(z_2, *, L)$	$(z_3, *, R)$	$(q, *, N)$
a	$(z_a^1, *, R)$	(z_a^1, a, R)	(z_b^1, a, R)	(z_a^2, a, R)	(z_b^2, a, R)	(z_1, a, L)	(z_2, a, L)	$(z_0, *, R)$
b	$(z_b^1, *, R)$	(z_a^1, b, R)	(z_b^1, b, R)	(z_a^2, b, R)	(z_b^2, b, R)	(z_1, b, L)	(z_2, b, L)	$(z_0, *, R)$

Berechnen Sie die von diesen TURING-Maschinen induzierten Funktionen $\{a, b\}^* \to \{a, b\}^*$.

33. Man konstruiere eine Turing-Maschine M, deren induzierte Funktion f_M durch $f_M(\lambda) = \lambda$ und

$$f_M(x_1 x_2 \dots x_n) = x_1 x_1 x_2 x_2 \dots x_n x_n = x_1^2 x_2^2 \dots x_n^2$$

für $x_i \in \{a, b\}$ mit $1 \le i \le n$ gegeben ist.

34. Man konstruiere eine Turing-Maschine M, deren induzierte Funktion f_M die Funktion $P : \mathbb{N}_0 \to \mathbb{N}_0$, definiert durch

$$P(n) = \begin{cases} 0 & \text{für } n = 0, \\ n - 1 & \text{für } n \ge 1, \end{cases}$$

verwirklicht.
Dabei sei die verwendete Zahlendarstellung
a) die unäre Zahlendarstellung („Strichkode“, Eingabealphabet $X = \{|\}$, n wird durch $|^n$ dargestellt),
b) die binäre Zahlendarstellung (Eingabealphabet $X = \{0, 1\}$).

35. Beweisen Sie, dass es zu jeder TURING-Maschine M eine TURING-Maschine M' mit

$$f_{M'}(x) = \begin{cases} 1 & f_M(x) \text{ ist definiert} \\ \textit{nicht definiert} & \text{sonst} \end{cases}$$

gibt.

36. Geben Sie eine TURING-Maschine an, die 1 bei einem Palindrom und sonst 0 ausgibt.

37. Man konstruiere eine Turing-Maschine M mit einem Eingabealphabet X, $\{a, b, \#\} \subseteq X$, deren induzierte Funktion f_M durch

$$f_M(w) = \begin{cases} w\#w & \text{für } w \in \{a, b\}^*, \\ \text{nicht definiert} & \text{sonst} \end{cases}$$

definiert ist.

38. Es sei $M = (2, X, Z, z_0, Q, \delta)$ eine zwei Band Turing-Maschine. Ferner sei

$$(z, w_e, xw'_e, w_1, x_1w'_1, w_2, x_2w'_2, w_a, \lambda)$$

mit $z \in Z$, $w_e, w'_e, w_1, w'_1, w_2, w'_2, w_a \in X^*$ und $x, x_1, x_2 \in X$ eine Konfiguration von M. Geben Sie die Konfiguration von M an, die durch Anwendung von

$$\delta(z, x, x_1, x_2) = (z', x'_1, x'_2, x', L, R, N, R)$$

entsteht.

39. Konstruieren Sie eine 4-Band-TURING-Maschine zur Multiplikation von Zahlen in Dezimaldarstellung.

40. Zeigen Sie, dass es zu jeder Mehrband-TURING-Maschine M (d.h. M ist k-Band-TURING-Maschine für ein $k \geq 1$) eine Mehrband-TURING-Maschine M' so gibt, dass $f_M = f_{M'}$ gilt und M' auf dem Eingabeband keine Bewegung des Kopfes nach links vollführt.

41. Mit *div* bzw. *mod* seien die ganzzahlige Division bzw. der dabei auftretende Rest bezeichnet. Ferner sei die Funktion $\ominus : \mathbf{N}^2 \to \mathbf{N}$ durch

$$x \ominus y = \begin{cases} x - y & \text{für } x \geq y \\ 0 & \text{sonst} \end{cases}$$

gegeben. Beweisen Sie jeweils mittels der Definitionen (d.h. ohne Benutzung von Aussagen mittels derer eine Berechenbarkeit in eine andere überführt wird), dass diese drei Funktionen
a) **LOOP**-berechenbar,
b) primitiv-rekursiv,
c) TURING-berechenbar
sind.

42. Es sei X ein Alphabet mit $\{0,1\} \subseteq X$. Zeigen Sie, dass die Menge aller Wörter $w \in X^*$, die mit 0 beginnen und enden und in denen mindestens eine 1 vorkommt, entscheidbar ist.

43. Zeigen Sie, dass die Menge der Primzahlen (als Teilmenge von $\mathbb{N}_0$) entscheidbar ist.

44. Beweisen Sie, dass aus der Entscheidbarkeit von $M \subseteq X^*$ die Entscheidbarkeit von $X^* \setminus M$ folgt.

45. Es sei X ein Alphabet mit $1 \in X$. Eine Teilmenge M von X^* heißt rekursiv-aufzählbar, wenn die Funktion

$$\varphi'_M(x) = \begin{cases} 1 & x \in M \\ \text{nicht definiert} & x \in X^* \setminus M \end{cases}$$

TURING-berechenbar ist.

a) Beweisen Sie, dass jede entscheidbare Menge $M \subseteq X^*$ rekursiv-aufzählbar ist.

b) Beweisen Sie, dass eine Menge $M \subseteq X^*$ genau dann entscheidbar ist, wenn M und $X^* \setminus M$ rekursiv-aufzählbar sind. (Es reicht, eventuell benötigte Turing-Maschinen informal zu beschreiben.)

46. Beweisen Sie, dass das Problem

Gegeben: Alphabet X, $n \geq 1$,
$\{(u_1, v_1), (u_2, v_2), \ldots, (u_n, v_n)\}$ mit $u_i, v_i \in X^+$ und $|u_i| = |v_i|$ für $1 \leq i \leq n$,
Frage: Gibt es eine Folge $i_1 i_2 \ldots i_k$ mit $k \geq 1$, $1 \leq i_j \leq n$ für $1 \leq j \leq k$ und
$u_{i_1} u_{i_2} \ldots u_{i_k} = v_{i_1} v_{i_2} \ldots v_{i_k}$?

entscheidbar ist.

47. Beweisen Sie, dass das Problem

Gegeben: Alphabet X mit $|X| = 1$, $n \geq 1$,
$\{(u_1, v_1), (u_2, v_2), \ldots, (u_n, v_n)\}$ mit $u_i, v_i \in X^+$ für $1 \leq i \leq n$,
Frage: Gibt es eine Folge $i_1 i_2 \ldots i_k$ mit $k \geq 1$, $1 \leq i_j \leq n$ für $1 \leq j \leq k$ und
$u_{i_1} u_{i_2} \ldots u_{i_k} = v_{i_1} v_{i_2} \ldots v_{i_k}$?

entscheidbar ist.

48. Untersuchen Sie, ob das 10. Hilbertsche Problem für folgende Fälle eine Lösung besitzt:

(a) $x^3 - 3x^2 - 6x + 18 = 0$,

(b) $2x^3y + 4xz^2 - 2y + 1 = 0$,

(c) $x^4 - 2x^2y^2 + 2y^4 - 3 = 0$.

49. Beweisen Sie, dass das Problem

    Gegeben: Turing-Maschine M,
    Frage: Stoppt M bei leerem Wort als Eingabe?

    unentscheidbar ist.

    Hinweis: Verwenden Sie die Unentscheidbarkeit des Halteproblems für Turing-Maschinen.

Literaturverzeichnis

[1] J.Albert, Th.Ottmann: Automaten, Sprachen und Maschinen für Anwender. B.-I.-Wissenschaftsverlag, 1983.

[2] A.Aho, J.E.Hopcroft, J.D.Ullman: The Design and Analysis of Algorithms. Reading, Mass., 1974.

[3] A.Aho, R.Sethi, J.D.Ullman: Compilerbau. Band 1 und 2, Addison-Wesley, 1990.

[4] A.Asteroth, Ch.Baier: Theoretische Informatik. Pearson Studium, 2002.

[5] L.Balke, K.H.Böhling: Einführung in die Automatentheorie und Theorie formaler Sprachen. B.-I.-Wissenschaftsverlag, 1993.

[6] W.Bucher, H.Maurer: Theoretische Grundlagen der Programmiersprachen. B.-I.-Wissenschaftsverlag, 1983.

[7] J.Carrol, D.Long: Theory of Finite Automata (with an Introduction to Formal Languages). Prentice Hall, London, 1983.

[8] E.Engeler, P.Läuchli: Berechnungstheorie für Informatiker. Teubner-Verlag, 1988.

[9] M.R.Garey, D.S.Johnson: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, 1979.

[10] J.Hopcroft, J.Ullman: Einführung in die Automatentheorie, formale Sprachen und Komplexitätstheorie. 2. Aufl., Addison-Wesley, 1990.

[11] E.Horowitz, S.Sahni: Fundamentals of Computer Algorithms. Computer Science Press, 1978.

[12] D.E.Knuth: The Art of Computer Programming. Volumes 1-3, Addison-Wesley, 1968-1975.

[13] U.Manber, Introduction to Algorithms. Addison-Wesley, 1990.

[14] K.Mehlhorn: Effiziente Algorithmen. Teubner-Verlag, 1977.

[15] Ch.Meinel: Effiziente Algorithmen. Fachbuchverlag Leipzig, 1991.

[16] W.Paul: Komplexitätstheorie. Teubner-Verlag, 1978.

[17] Ch.Posthoff, K.Schulz: Grundkurs Theoretische Informatik. Teubner-Verlag, 1992.

[18] U.Schöning: Theoretische Informatik kurz gefaßt. B.I.Wissenschaftsverlag, 1992.

[19] R.Sedgewick: Algorithmen. Addison-Wesley, 1990.

[20] B.A.Trachtenbrot: Algorithmen und Rechenautomaten. Berlin, 1977.

[21] G. Vossen, K.-U. Witt: Grundlagen der Theoretischen Informatik mit Anwendungen. Vieweg-Verlag, Braunschweig, 2000.

[22] K.Wagner: Einführung in die Theoretische Informatik. Springer-Verlag, 1994.

[23] D.Wätjen: Theoretische Informatik. Oldenbourg-Verlag, 1994.

[24] I.Wegener: Theoretische Informatik. Teubner-Verlag, 1993.

[25] D.Wood: Theory of Computation. Harper & Row Publ., 1987.