# Berechenbarkeit und Komplexität

**Vorlesung SS 2013**

## 1. Einführung

I. Berechenbarkeitsbegriff, typische Fragen:
- wann ist eine Funktion berechenbar?
- wie lässt sich der intuitive Berechenbarkeitsbegriff formal präzisieren?
- wie verhalten sich unterschiedliche Formalisierungen zueinander?

II. Grenzen der Berechenbarkeit, typische Fragen:
- sind alle mathematisch exakt beschreibbaren Funktionen berechenbar?
- falls nein, welche nicht?

III. Komplexitätstheorie, typische Fragen:
- wie lässt sich die Komplexität eines Problems beschreiben?
- was sind relevante theoretische Konzepte hierfür?

hier zunächst vorläufige Grobgliederung von I, II:

1. Motivation, was will Berechenbarkeitstheorie, unlösbare Probleme, Whlg. Turingmaschinen
2. Turing-Berechenbarkeit, Mehrband-Turingmaschinen, Äquivalenz zu normaler TM, Verknüpfung von TMs
3. LOOP, WHILE und GOTO - Berechenbarkeit
4. Äquivalenz TM, WHILE, GOTO
5. Rekursive Funktionen, Äquivalenz primitiv-rekursiv - LOOP - berechenbar
6. Äquivalenz μ-rekursiv - WHILE - berechenbar, Ackermannfunktion
7. Unentscheidbarkeit, Halteproblem
8. Weitere unentscheidbare Probleme, Satz von Gödel

### 1.1 Was will Berechenbarkeitstheorie?

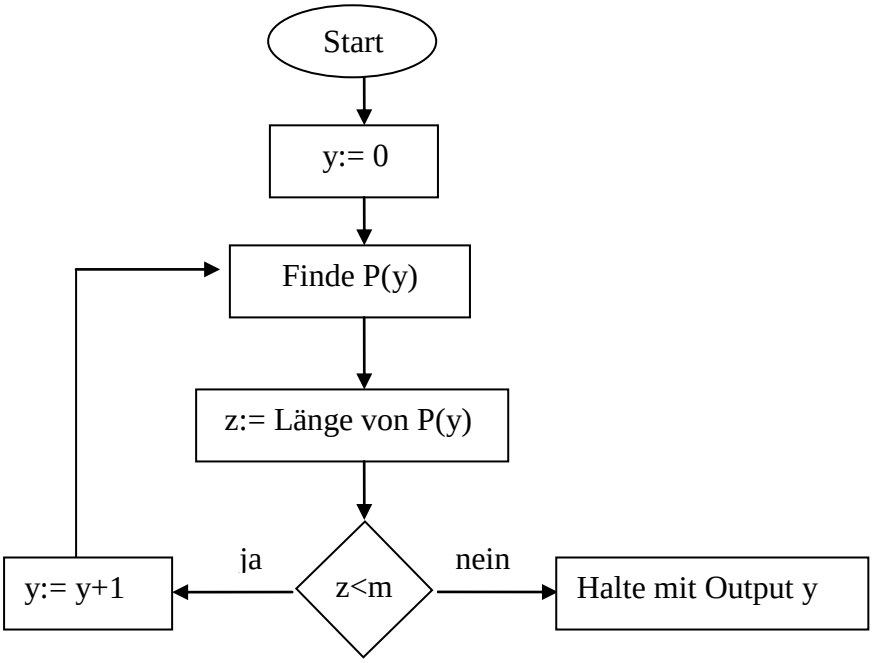
Zur Motivation:

Frage: gibt es ein Programm in einer beliebigen Programmiersprache L, das bei Eingabe einer Zahl n als Ausgabe das kürzeste L-Programm liefert, das (ohne Eingabe) n ausdruckt?

Antwort: Nein!

Beweis:
Voraussetzungen:
zu jedem n gibt es ein Programm, das konstante Funktion n berechnet,
Länge eines Programms bestimmbar,
es gibt nur endlich viele verschiedene Programme bestimmter Länge,
Konstanten werden codiert zu Basis 2.

Nehmen wir an, es gäbe so ein Programm. Nennen wir es P. Aus P ließe sich zu jedem m folgendes Programm $Q_m$ konstruieren:




Was tut $Q_m$?
Berechnet kleinstes i, so dass i nicht durch Programm kürzer als m ausgedruckt werden kann.

Länge der Programme $Q_m$? Konstant bis auf Darstellung von m in Programmtext.
Also gibt es c mit Länge von $Q_m$ = c + Länge der Darstellung von m
$\leq c + 1 + \log_2 m$

Für genügend großes m gilt damit: Länge von $Q_m < m$.

Aber dann druckt $Q_m$ die kleinste Zahl, die nicht mit Programm kürzer als m gedruckt werden kann, ist aber selbst kürzer. Widerspruch!!

<u>Probleme und Sprachen:</u>

in der theoretischen Informatik werden Probleme und Sprachen (genauer: das Enthaltensein eines Wortes in einer Sprache) oft gleich gesetzt:
Beispiel: ist eine vorgegebene natürliche Zahl n Primzahl?
entspricht: gehört Binärcodierung von n zur Sprache

$$L = \{w \in \{0,1\}^* \mid w \text{ Binärcodierung einer Primzahl}\}$$

oder: ist f(x,y) = z für eine 2stellige Funktion f?
entspricht: gehört $x_b\$y_b\$z_b$ , wobei $a_b$ Binärcodierung von a ist, zur Sprache

$$L = \{w_1\$w_2\$w_3 \mid w_i \in \{0,1\}^* \text{ Binärcodierungen von } n_1, n_2, n_3 \text{ so dass } f(n_1,n_2) = n_3\}$$

Wie viele Sprachen über einem Alphabet Σ gibt es? Sprache Teilmenge von Σ*, also #Sprachen = | Pot(Σ*)|

Eine Menge M heißt
abzählbar unendlich: gleichmächtig mit $\mathbb{N}$, d.h. es gibt bijektive Funktion f: $\mathbb{N} \rightarrow$ M
höchstens abzählbar: endlich oder abzählbar unendlich
überabzählbar: nicht höchstens abzählbar

Σ* abzählbar unendlich: Beispiel Σ = {a,b}
ε, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, bab, bba, bbb, aaaa, ...

Pot(Σ*) ist überabzählbar:

Beweis (Diagonalverfahren):
Sei f(0), f(1), ... Abzählung von Σ*.
Angenommen, es gäbe Abzählung g(0), g(1), ... von Pot(Σ*).

Definiere D = {f(j) | f(j) $\notin$ g(j)}

Da D eine Sprache über Σ ist, müsste es ein n geben mit g(n) = D.
Es gilt aber: f(n) $\in$ D <=> f(n) $\notin$ g(n) <=> f(n) $\notin$ D! Widerspruch.

Veranschaulichung:

| | f(0) | f(1) | f(2) | f(3) | f(4) | f(5) | … |
|---|---|---|---|---|---|---|---|
| g(0) | 0 | x | x | x | x | x | … |
| g(1) | x | 1 | x | x | x | x | … |
| g(2) | x | x | 1 | x | x | x | … |
| g(3) | x | x | x | 0 | x | x | … |
| g(4) | x | x | x | x | 0 | x | … |
| g(5) | x | x | x | x | x | 1 | … |
| … | | | | | | | |

der j-te Wert in der i-ten Zeile gibt an, ob f(j) Element der Sprache g(i) ist: 1 bedeutet ja, 0 bedeutet nein (x: hier irrelevant).

Die oben definierte Sprache D enthält f(n) genau dann, wenn in der Diagonale in Zeile n eine 0 steht (im Beispiel wären f(0), f(3), f(4), ... enthalten). Die Sprache D unterscheidet sich von jeder Sprache g(k) in der Abzählung an der k-ten Stelle. Damit ist D selbst nicht in der Abzählung enthalten.

Es gibt also überabzählbar viele Sprachen, aber nur abzählbar unendlich viele Programme in einer vorgegebenen Programmiersprache. Also: es kann nicht für jede Sprache L ein Programm geben, das L akzeptiert.

Was heißt berechenbar?

Intuitive Vorstellung:
es gibt ein Verfahren, das bei bestimmter Eingabe gesuchte Ausgabe produziert.
Vage, muss präzisiert werden
(dabei sollen Größe des Speichers und Schnelligkeit der Maschine keine Rolle spielen)

Bisher: Automaten, die "berechnen" ob $w \in L$.
Entspricht Funktion $f_L: \Sigma^* \to \{0,1\}$, die 1 liefert gdw. $w \in L$, 0 sonst.
(diese Funktion wird auch charakteristische Funktion genannt).

Des Weiteren: Funktionen über $\mathbb{N}$
n-stellige partielle Funktion:
Teilmenge f von $\mathbb{N}^{n+1}$ so dass $(x_1, ..., x_n, q) \in f$ und $(x_1, ..., x_n, r) \in f$ impliziert $q = r$.
totale Funktion: für jedes $x_1, ..., x_n$ gibt es y mit $(x_1, ..., x_n, y) \in f$.
nicht-totale Funktionen heißen echt partiell.
übliche Schreibweise: $f(x_1, ..., x_n) = q$ statt $(x_1, ..., x_n, q) \in f$.

Funktion f ist berechenbar, wenn es ein Verfahren (Algorithmus, Programm) gibt, das gestartet mit Eingabe $x_1, ..., x_n$, nach endlich vielen Schritten mit Ausgabe $f(x_1, ..., x_n)$ stoppt.
Ist $f(x_1, ..., x_n)$ undefiniert, so läuft das Verfahren endlos.

Intuitive Beispiele:

1. total undefinierte Funktion berechenbar:

```
INPUT(n);
REPEAT UNTIL FALSE;
```

2. $h(n) = \begin{cases} 1 & \text{falls 5 n-mal hintereinander in } \pi \text{ vorkommt.} \\ 0 & \text{sonst.} \end{cases}$

berechenbar? Ja, denn
falls es beliebig lange 5er-Ketten in $\pi$ gibt, dann ist $h(n) = 1$ für alle n, berechenbar
falls es längste 5er-Kette gibt und diese die Länge k hat, dann ist

$h(n) = \begin{cases} 1 & \text{falls } n \leq k \\ 0 & \text{sonst} \end{cases}$

Auch diese Funktion ist für alle k berechenbar. Wir wissen nur nicht, welche dieser berechenbaren Funktionen die richtige ist.

Merke: für die Berechenbarkeit einer Funktion genügt es, dass ein entsprechendes Verfahren existiert, wir müssen es nicht angeben können!!

3. ähnlich: $i(n) = \begin{cases} 1 & \text{falls deterministische und nichtdet. LBAs äquivalent} \\ 0 & \text{sonst} \end{cases}$

berechenbar, denn sowohl die konstante Funktion 1, wie die konstante Funktion 0 sind berechenbar.

4. Sei r beliebige reelle Zahl, definiere

$f_r(n) = 1$ falls n Anfangsstück der Dezimalbruchentwicklung von r ist
$0$ sonst

Kann nicht für alle r berechenbar sein, denn: Rechenverfahren immer durch endlichen Text beschrieben, deshalb abzählbar. Dagegen überabzählbar viele reelle Zahlen.

Präzisierungsvorschläge für den intuitiven Berechenbarkeitsbegriff :

Turing-Berechenbarkeit, While-Berechenbarkeit, Goto-Berechenbarkeit, μ-Rekursivität

Hauptresultat: alle äquivalent

Churchsche These:

*Die durch die Begriffe Turing-Berechenbarkeit (While-Berechenbarkeit, Goto-Berechenbarkeit, μ-Rekursivität) erfassten Funktionen sind genau die im intuitiven Sinne berechenbaren Funktionen.*

## 2. Turing-Berechenbarkeit

Eine der Standardpräzisierungen des Berechenbarkeitsbegriffs verwendet Turing Maschinen (TMs). (Wiederholung):

*Def.: Eine Turingmaschine ist ein 7-Tupel M = (Z, Σ, Γ, δ, $z_0$, □, E). Hierbei ist*

*Z endliche Zustandsmenge*
*Σ Eingabealphabet*
*Γ Arbeitsalphabet*
*δ (partielle) Überführungsfunktion*
*$z_0$ Anfangszustand*
*□ Leerzeichen (Blank)*
*E Endzustände*

falls M deterministisch: $\delta: Z \times \Gamma \rightarrow Z \times \Gamma \times \{L,R,N\}$
falls M nichtdeterministisch: $\delta: Z \times \Gamma \rightarrow Pot(Z \times \Gamma \times \{L,R,N\})$

Man kann zeigen: jede nichtdeterministische TM kann durch deterministische modelliert werden => deshalb hier zunächst immer deterministische betrachtet.

*Def.: Eine Konfiguration einer TM ist ein Wort $k \in \Gamma^*Z\Gamma^*$.*

Die Eingabe x der TM steht auf dem Band: Startkonfiguration $z_0x$

Def.: Die Konfigurationsübergangsrelation |-- ist die kleinste Relation, so dass:

$a_1...a_m z b_1...b_n$ |-- $a_1...a_m z' c b_2...b_n$ falls $\delta(z,b_1) = (z', c, N)$, $m \geq 0$, $n \geq 1$
$a_1...a_m z b_1...b_n$ |-- $a_1...a_m c z' b_2...b_n$ falls $\delta(z,b_1) = (z', c, R)$, $m \geq 0$, $n \geq 2$
$a_1...a_m z b_1...b_n$ |-- $a_1...a_{m-1} z' a_m c b_2...b_n$ falls $\delta(z,b_1) = (z', c, L)$, $m \geq 1$, $n \geq 1$

Sonderfälle, bei denen neues Feld besucht wird:

$a_1...a_m z b_1...b_n$ |-- $a_1...a_m c z'$□ falls $\delta(z,b_1) = (z', c, R)$, $m \geq 0$, $n = 1$
$z b_1...b_n$ |-- $z'$□$c b_2...b_n$ falls $\delta(z,b_1) = (z', c, L)$, $m = 0$, $n \geq 1$

Def.: Die von einer TM M akzeptierte Sprache ist:

$$T(M) = \{x \in \Sigma^* \mid z_0 x \text{ |--* } \alpha z \beta, z \in E, \alpha, \beta \in \Gamma^* \}$$

|--* ist die reflexive, transitive Hülle von |--, d.h. die kleinste Relation, für die gilt:

1) c |--* c, (reflexiv, null-malige Anwendung von |--)
2) $c_1$ |--* $c_2$ und $c_2$ |-- $c_3$ impliziert $c_1$ |--* $c_3$. (transitiv)

Def.: Eine Funktion f: $\mathbb{N}^k \rightarrow \mathbb{N}$ heißt Turing-berechenbar, wenn es eine deterministische TM M gibt, so dass für alle $n_1, ..., n_k, m \in \mathbb{N}$ gilt:

$f(n_1, ..., n_k) = m$ genau dann wenn
$z_0 bin(n_1)\# bin(n_2)\#... \#bin(n_k)$ |--* □...□$z_e bin(m)$ □...□

wobei $z_e$ Endzustand, bin(x) Binärdarstellung von x (ohne führende Nullen).

Def.: Eine Funktion f: $\Sigma^* \rightarrow \Sigma^*$ heißt Turing-berechenbar, wenn es eine (deterministische) TM M gibt, so dass für alle $x, y \in \Sigma^*$ gilt:

$$f(x) = y \text{ genau dann wenn } z_0 x \text{ |--* □...□} z_e y \text{ □...□}$$

wobei $z_e$ Endzustand.

Beispiel: TM, die die Funktion f(x) = x+1 berechnet:

$M_3 = (\{q_0, q_1, q_2, q_f\}, \{0,1\}, \{0,1, □\}, \delta, q_0, □, \{q_f\})$

mit $\delta(q_0,0) = (q_0,0,R)$
$\delta(q_0,1) = (q_0,1,R)$
$\delta(q_0,□) = (q_1,□,L)$

$\delta(q_1,0) = (q_2,1,L)$
$\delta(q_1,1) = (q_1,0,L)$
$\delta(q_1,□) = (q_f,1,N)$

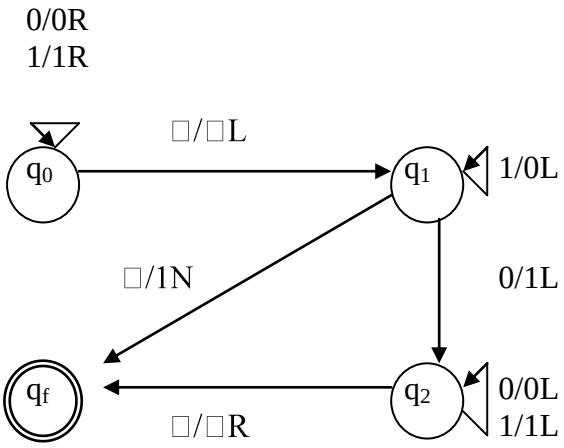
$\delta(q_2,0) = (q_2,0,L)$
$\delta(q_2,1) = (q_2,1,L)$
$\delta(q_2,□) = (q_f, □,R)$

Intuitiv: Wandern an rechtes Ende der Eingabe, Ersetzen der am weitesten rechts stehenden 0 durch 1, aller rechts davon stehenden 1en durch 0; wenn keine 0 vorkommt, dann 1 vor Eingabe schreiben und alle 1en in 0 umwandeln: entspricht +1 in Binärdarstellung

$q_0 10 \vdash 1q_0 0 \vdash 10q_0\square \vdash 1q_1 0\square \vdash q_2 11\square \vdash q_2\square 11\square \vdash \square q_f 11\square$

$q_0 11 \vdash 1q_0 1 \vdash 11q_0\square \vdash 1q_1 1\square \vdash q_1 10\square \vdash q_1\square 00\square \vdash q_f 100\square$

Graphische Repräsentation. Übergangsdiagramme:




X/YZ an Pfeil bedeutet: bei Lesen von X wird X durch Y ersetzt und nach Z gegangen. Pfeil gibt Nachfolgezustand an.

Mehrband-Turingmaschinen

intuitiv: k Bänder, k Schreib-Lese-Köpfe, können sich unabhängig voneinander bewegen (partielle) Übergangsfunktion:

$$\delta: Z \times \Gamma^k \rightarrow Z \times \Gamma^k \times \{L, R, N\}^k$$

Es muss jeweils festgelegt werden, welche Bänder für die Eingabe verwendet werden und welches für die Ausgabe. Bei k-stelligen Funktionen oft: k+1 Bänder, Eingabe auf den ersten k Bändern, Ausgabe wird auf Band k+1 produziert.

Beispiel: 3-Band TM, die unäre Multiplikation realisiert: Band 3 := Band 2 * Band 1

Startkonfiguration: Band 3 leer, Band 1 und 2 enthalten Multiplikanden, unär codiert mit a's. Lesekopf Band 1,2 auf jeweils erstem a, Startzustand $z_0$. x und y sind beliebige Bandsymbole.

$(z_0, a, a, x) \rightarrow (z_1, a, a, a, R, N, R)$
$(z_0, a, \square, x) \rightarrow (z_e, a, \square, x, N, N, N)$
$(z_0, \square, a, x) \rightarrow (z_e, \square, a, x, N, N, N)$

$(z_1, a, x, y) \rightarrow (z_1, a, x, a, R, N, R)$
$(z_1, \square, x, y) \rightarrow (z_2, \square, x, y, L, N, N)$

$(z_2, a, x, y) \rightarrow (z_2, a, x, y, L, N, N)$
$(z_2, \square, x, y) \rightarrow (z_0, \square, \square, y, R, R, N)$

Idee: $z_0$: Startzustand, $z_1$: Inhalt von Band 1 auf Band 3 kopieren, $z_2$: auf Band 1 nach links gehen, wenn Blank gelesen 1 a auf Band 2 löschen und in $z_1$ gehen. Ende wenn Band 2 leer.

Beispiel:

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 3: | □ | a□ | aa□ | aaa□ | aaa□ | | aaa□ | aaa□ | | aaaaaa□ |
| 2: | aa | aa | aa | aa | aa | → | aa | □a | → | □□□ |
| 1: | aaa | aaa | aaa | aaa□ | aaa□ | | □aaa□ | □aaa□ | | □aaa□ |
| | $z_0$ | $z_1$ | $z_1$ | $z_1$ | $z_2$ | | $z_2$ | $z_0$ | | $z_e$ |

Satz: Zu jeder Mehrband-Turingmaschine M gibt es eine (1-Band) Turingmaschine M', so dass

1. T(M) = T(M') falls M akzeptierend, bzw.
2. M und M' berechnen dieselbe Funktion f falls M berechnende TM

Beweisidee: Band von M' hat 2k „Spuren":

aus:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| … | $a_{1,1}$ | $a_{1,2}$ | $a_{1,3}$ | $a_{1,4}$ | $a_{1,5}$ | $a_{1,6}$ | $a_{1,7}$ | … |
| … | $a_{2,1}$ | $a_{2,2}$ | $a_{2,3}$ | $a_{2,4}$ | $a_{2,5}$ | $a_{2,6}$ | $a_{2,7}$ | … |
| … | $a_{3,1}$ | $a_{3,2}$ | $a_{3,3}$ | $a_{3,4}$ | $a_{3,5}$ | $a_{3,6}$ | $a_{3,7}$ | … |

3 Bänder

wird:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| … | $a_{1,1}$ □ | $a_{1,2}$ □ | $a_{1,3}$ * | $a_{1,4}$ □ | $a_{1,5}$ □ | $a_{1,6}$ □ | $a_{1,7}$ □ | … |
| … | $a_{2,1}$ □ | $a_{2,2}$ □ | $a_{2,3}$ □ | $a_{2,4}$ □ | $a_{2,5}$ * | $a_{2,6}$ □ | $a_{2,7}$ □ | … |
| … | $a_{3,1}$ □ | $a_{3,2}$ □ | $a_{3,3}$ □ | $a_{3,4}$ * | $a_{3,5}$ □ | $a_{3,6}$ □ | $a_{3,7}$ □ | … |

1 Band, 6 Spuren

$\Gamma' = \Gamma \cup (\Gamma \cup \{*\})^{2k}$
* ist neues Symbol zur Markierung der Position des Lesekopfes der k-Band TM.

Grundidee der Modellierung von M durch M':
1. Starte M' mit Eingabe $x_1 \ldots x_n \in \Sigma*$.
2. Erzeuge Spurendarstellung der Startkonfiguration von M, Lesekopf links von erstem *.
3. Simuliere einen Schritt von M durch mehrere Schritte von M':
   gehe nach rechts bis * gefunden
   führe entsprechende Änderungen in der jeweiligen Spur des Bandes durch
   merke (im Zustand), dass entsprechende Spur abgearbeitet ist
   wenn alle Spuren abgearbeitet, gehe nach links hinter erstes * zurück
4. Wenn Endzustand von M erreicht, produziere Ausgabe: ersetze Bandinhalt durch Inhalt der Spur, die dem Ausgabeband von M entspricht.

Bemerkung: führt zu immenser Zahl von Zuständen. Unpraktisch, aber für theoretische Überlegungen zur Berechenbarkeit nicht relevant. Es muss gelten: $|Z'| \geq |Z \times \Gamma^k|$

Wir wollen im Folgenden zeigen, dass sich TMs miteinander auf verschiedene Weise verknüpfen lassen. Wir verwenden hierzu Bezeichnungen bzw. eine graphische Notation, wie man sie aus imperativen Programmiersprachen bzw. von Flussdiagrammen kennt.

Notation:
Wenn M 1-Band TM, so ist M(i,k) die k-Band-TM, die auf Band i M simuliert, alle anderen Bänder unverändert lässt (k kann entfallen, wenn es implizit gegeben ist)

| | | |
|---|---|---|
| Nenne: | +1 TM (binäres Inkrement): | "Band:= Band+1" |
| | "Band:= Band+1"(i): | "Band i := Band i + 1" |

Folgende Mehrband-TM (mit intuitiver Bedeutung) sind leicht zu erstellen:

"Band i := Band i - 1"
"Band i := 0"
"Band i := Band j"

Hintereinanderschaltung von Turing-Maschinen:

Seien $M_1 = (Z_1, \Sigma, \Gamma_1, \delta_1, z_{11}, \square, E_1)$
$M_2 = (Z_2, \Sigma, \Gamma_2, \delta_2, z_{21}, \square, E_2)$

Turing-Maschinen, so dass $Z_1 \cap Z_2 = \varnothing$ und $M_1$ ein Wort aus $\Sigma*$ berechnet .

start → $M_1$ → $M_2$ → stop

oder: $M_1; M_2$

bezeichnet die TM $M = (Z_1 \cup Z_2, \Sigma, \Gamma_1 \cup \Gamma_2, \delta, z_{11}, \square, E_2)$ mit

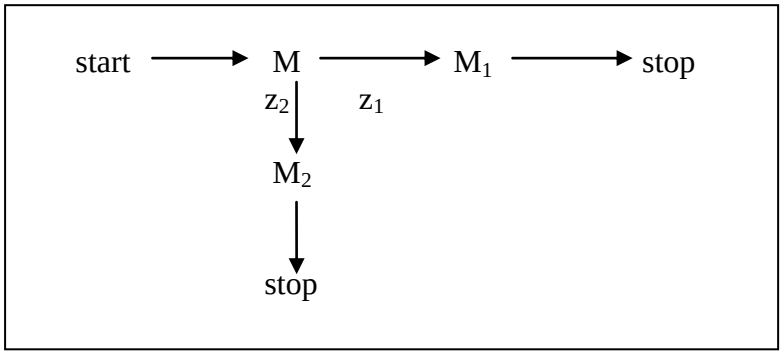
$\delta = \delta_1 \cup \delta_2 \cup \{(z_e a \rightarrow z_{21} a N \mid z_e \in E_1, a \in \Gamma_1\}$

Beispiel:

start → “Band:= Band + 1” → “Band:= Band + 1” → stop

TM, die Bandinhalt um 2 inkrementiert.

Fallunterscheidungen lassen sich ähnlich modellieren:



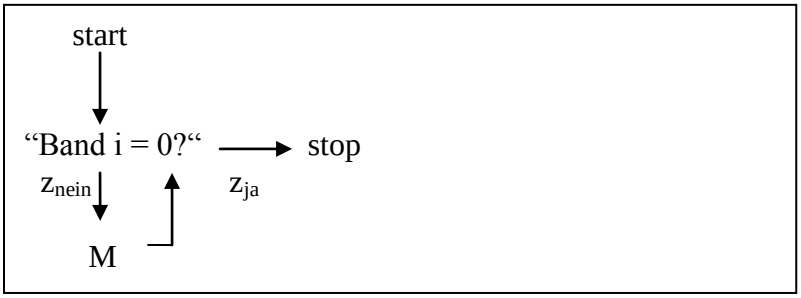

bezeichnet die TM, die $M_1$ ausführt, wenn M in $z_1$ terminiert, $M_2$, wenn M in $z_2$ terminiert.

Eine Turingmaschine, genannt “Band = 0?“, die testet, ob auf dem Band die Eingabe 0 steht, lässt sich wie folgt definieren:

$Z = \{z_0, z_1, z_{ja}, z_{nein}\}$, Startzustand $z_0$, Endzustände $z_{ja}$, $z_{nein}$.

| | |
|---|---|
| $z_0a \rightarrow z_{nein}aN$ | für $a \neq 0$ |
| $z_00 \rightarrow z_10R$ | |
| $z_1a \rightarrow z_{nein}aL$ | für $a \neq \square$ |
| $z_1\square \rightarrow z_{ja}\square L$ | |

Wir nennen die TM “Band = 0?“ (i): “Band i = 0?“
Sei M beliebige TM. Wir nennen folgende TM “While Band i ≠ 0 Do M“:




Wir können also Turing-Maschinen mit Konstrukten verknüpfen, wie wir sie von üblichen imperativen Programmiersprachen her kennen.