Kapitel 3: Berechnungstheorie

Gliederung

Zielstellung

- Gegenstand der Betrachtung

Wir sehen uns ein einfaches, aber überraschend leistungsfähiges Berechnungsmodell an.

- Dieses Modell wurde von Alan Turing entwickelt und war das erste Berechnungsmodell überhaupt.
- Die Programme in diesem Berechnungsmodell nennt man Turing-Maschinen.

Turing-Maschinen werden benutzt, um Funktionen zu berechnen.

Turing-Maschinen kann man zu mehr benutzen, als nur Entscheidungsprobleme zu lösen.

Turing A., On computable numbers with an application to the Entscheidungsproblem,
Proc. London Math. Soc., Vol. 42 (1936), 230-265.

- Historisches (aus einem Gutachten zum Artikel von A. Turing)

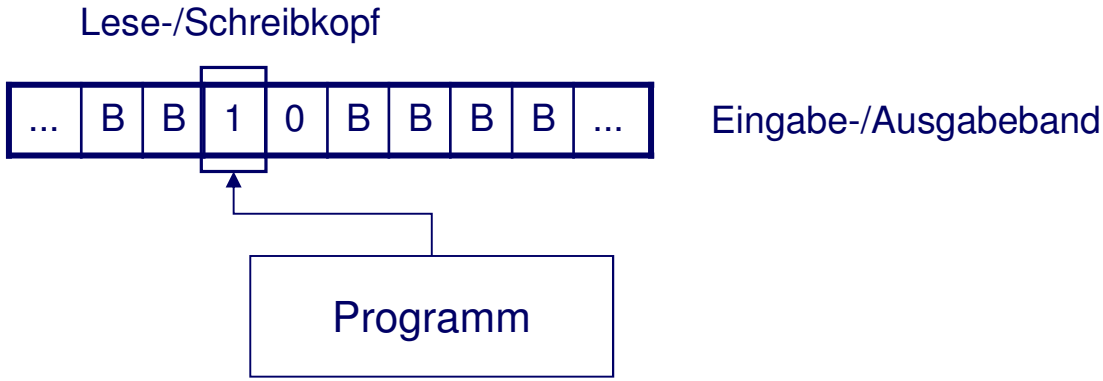
> *This is a bizarre paper. It begins by defining a computing device absolutely unlike anything I have seen, then proceeds to show – I haven't quite followed the needlessly complicated formalism – that there are numbers that it can't compute. As I see it, there are two alternatives that apply to any machine that will ever be built: Either these numbers are too big to be represented in the machine, in which case the conclusion is obvious, or they are not; in that case, a machine that can't compute them is simply broken!*

Berechnungsmodelle

- Zur Präzisierung einer Sprache zur Beschreibung von Algorithmen sind u.a. zu klären:
  - Welche elementare Operationen stehen zur Verfügung?
  - Wie viel Speicher steht zur Verfügung, und wie geht man damit um?
  - Wie verläuft die Eingabe?
  - Wie verläuft die Ausgabe?
- Festlegung der Semantik
  - Welche möglichen Ausgaben berechnet ein „Programm“ in dem gewählten Berechnungsmodell aus den Eingaben?

  bzw. hier genauer (ohne alternative Ausgaben)
  - Welche totale oder partielle Funktion f berechnet ein Algorithmus (in dem vorliegenden Berechenbarkeitsmodell) eigentlich?

Turing-Maschinen



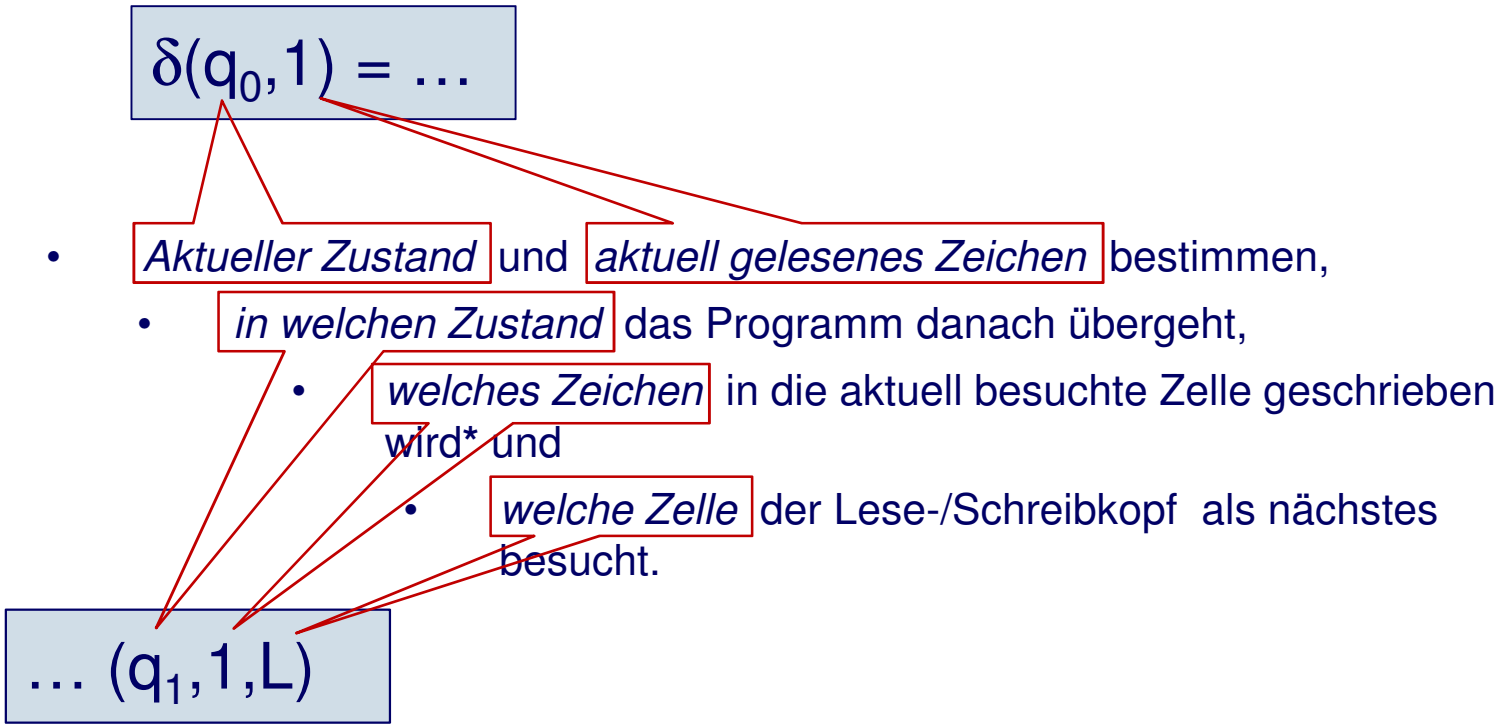
- Im Unterschied zu einem endlichen Automaten hat eine Turing-Maschine eine Eingabe-/Ausgabeband (auch Speicherband, Arbeitsband), das aus Zellen besteht
- Eine Turing-Maschine kann
  - den Inhalt der Zellen des Arbeitsbands lesen,
  - in die Zellen schreiben,
  - alle Zellen mehrfach benutzen, denn sie kann auch
  - den Lese-/Schreibkopf nach links und rechts bewegen.

- informelle Beschreibung
  - Arbeitsband
    - linearer Speicher; besteht aus Zellen;
    - in jede Zelle passt ein Zeichen;
    - auf dem Arbeitsband steht die zu verarbeitende Eingabe und nach der Berechnung die Ausgabe;
    - vor und hinter der Eingabe stehen in den Zellen nur spezielle Leerzeichen (hier ´B´)
  - Lese-/Schreibkopf
    - steht zu Beginn auf dem ersten Zeichen der Eingabe – außer sie ist leer, dann auch auf einem B.
    - liest das Zeichen in der aktuell besuchten Zelle
    - schreibt ein Zeichen in die aktuell besuchte Zelle
    - bewegt sich nach dem Schreiben um eine Zelle nach links oder rechts oder gar nicht
    - steht nach der Berechnung auf dem ersten Zeichen der Ausgabe

- informelle Beschreibung (cont.)
  - Programm
    - befindet sich in unterschiedlichen Zuständen
    - hat zwei ausgezeichnete Zustände
      - einen Anfangszustand q_0
      - einen Endzustand q_e
    - Sobald der Endzustand erreicht wird, wird gestoppt.
    - Aktueller Zustand und aktuell gelesenes Zeichen bestimmen,
      - in welchen Zustand das Programm danach übergeht,
      - welches Zeichen in die aktuell besuchte Zelle geschrieben wird* und
      - welche Zelle der Lese-/Schreibkopf als nächstes besucht.

*) „*Zeichen stehen lassen*“ wird erreicht durch „*dasselbe Zeichen hineinschreiben*“ – wie auf dem Stapel des Kellerautomaten.

- Programmschritte nochmal etwas formaler und ganz ausführlich …



- informelle Beschreibung (cont.)
  - direkte Ausgabe
    - Die Zeichenkette, die sich auf dem Arbeitsband befindet, nachdem der Endzustand q_e erreicht wird, wird als Ausgabe interpretiert.
    - Genauer gesagt:
      - Die längste ununterbrochene Σ-Zeichenkette, die in der aktuellen Zelle beginnt, zählt als Ergebnis.
      - Steht dort kein Σ-Zeichen, ist das Ergebniswort ε.
      - Mit Mehraufwand kann man dafür sorgen, dass außer dem Ergebniswort nur noch B's davor und dahinter stehen. Das ist bei „Unterprogrammen" hilfreich, deren Ergebnis man weiter verwenden möchte.

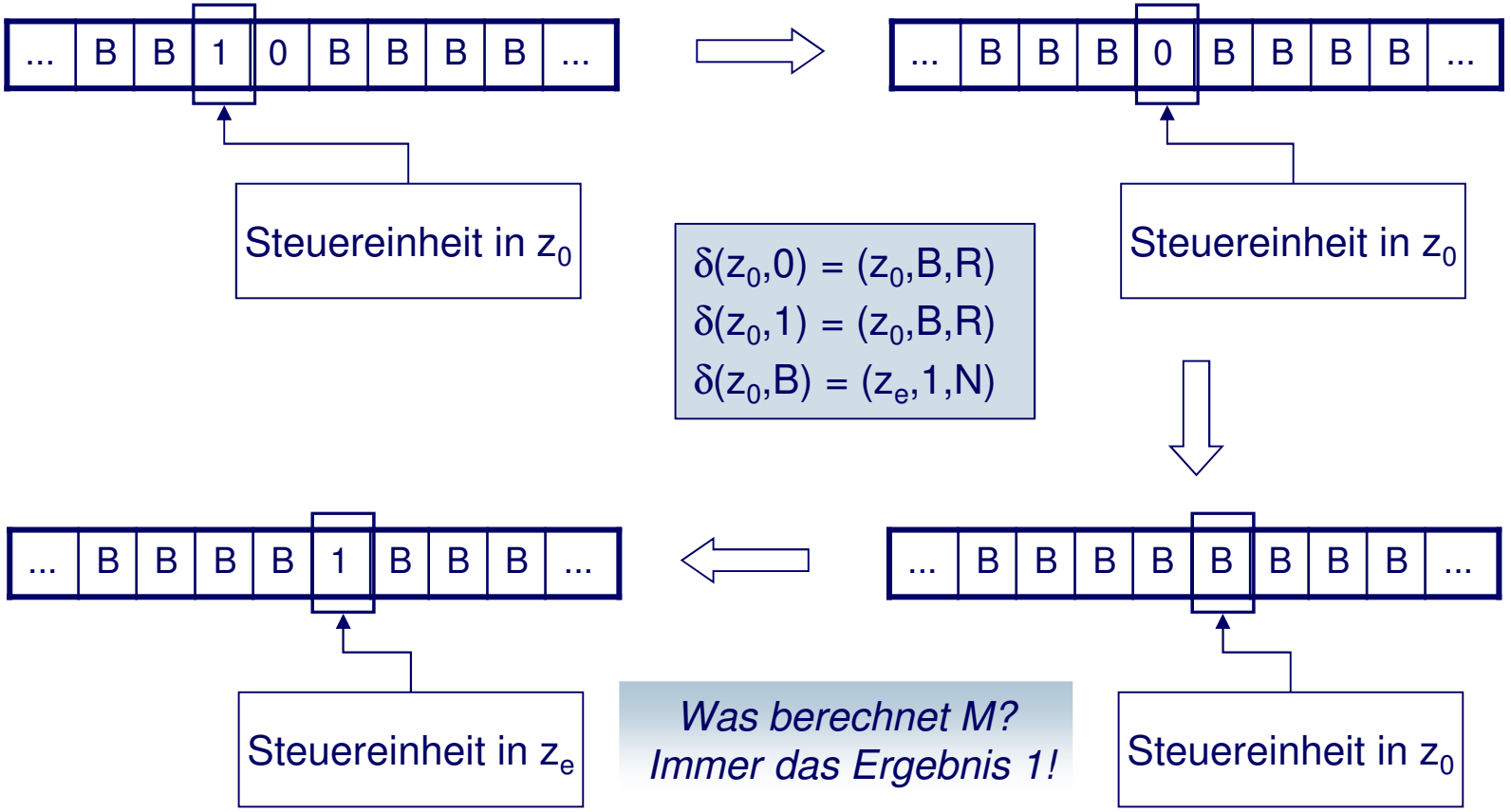
Beispiel 0

- Alphabet $\Sigma = \{0,1\}$
- Zustandsmenge $Z = \{ z_0, z_e \}$
- Anfangszustand z_0
- Endzustand z_e
- Zustandsüberführungsfunktion δ (Beschreibung der Aktionen)

$$\delta(z_0,0) = (z_0,B,R)$$
$$\delta(z_0,1) = (z_0,B,R)$$
$$\delta(z_0,B) = (z_e,1,N)$$

Was berechnet M?

- Illustration der Arbeitsweise von M



Beispiel 1

- Wir wollen eine Turing-Maschine M angeben, die die Funktion berechnet, die jeder natürlichen Zahl x (incl. 0) die Zahl x + 1 zuordnet.
- Um uns zunächst das Leben einfacher zu machen, werden natürliche Zahlen hier unär kodiert, d.h.
  - die Zahl n wird durch eine Folge von n Einsen kodiert.
- das Programm der Turing-Maschine M:

$\delta(q_0,B) = (q_e,1,N)$ $\quad \delta(q_0,1) = (q_1,1,L)$
$\delta(q_1,B) = (q_e,1,N)$

Beispiel 2

- Wir wollen nochmals eine Turing-Maschine M angeben, die die Funktion berechnet, die jeder natürlichen Zahl x (incl. 0) die Zahl x + 1 zuordnet.
- Jetzt werden natürliche Zahlen aber binär kodiert, wobei es keine führenden Nullen geben soll, d.h. als 0, 1, 10, 11, 100 usw.
- Programm der Turing-Maschine M:

Was geschieht da?

$\delta(q_0,0) = (q_e,1,N)$	$\delta(q_0,1) = (q_1,1,R)$	
$\delta(q_1,B) = (q_2,B,L)$	$\delta(q_1,0) = (q_1,0,R)$	$\delta(q_1,1) = (q_1,1,R)$
$\delta(q_2,B) = (q_e,1,N)$	$\delta(q_2,0) = (q_3,1,L)$	$\delta(q_2,1) = (q_2,0,L)$
$\delta(q_3,B) = (q_e,B,R)$	$\delta(q_3,0) = (q_3,0,L)$	$\delta(q_3,1) = (q_3,1,L)$

Beispiel 2

- Wir wollen nochmals eine Turing-Maschine M angeben, die die Funktion berechnet, die jeder natürlichen Zahl x (incl. 0) die Zahl x + 1 zuordnet.
- Jetzt werden natürliche Zahlen aber binär kodiert, wobei es keine führenden Nullen geben soll, d.h. als 0, 1, 10, 11, 100 usw.
- Programm der Turing-Maschine M:

Was geschieht da?

$\delta(q_0,0) = (q_e,1,N)$	$\delta(q_0,1) = (q_1,1,R)$		$0 \mapsto 1$, sonst: →→
$\delta(q_1,B) = (q_2,B,L)$	$\delta(q_1,0) = (q_1,0,R)$	$\delta(q_1,1) = (q_1,1,R)$	
$\delta(q_2,B) = (q_e,1,N)$	$\delta(q_2,0) = (q_3,1,L)$	$\delta(q_2,1) = (q_2,0,L)$	+1, ggf. Übertrag ←
$\delta(q_3,B) = (q_e,B,R)$	$\delta(q_3,0) = (q_3,0,L)$	$\delta(q_3,1) = (q_3,1,L)$	erledigt ←

gehe zu / bleibe auf erstem Zeichen

- formale Beschreibung
  - Bestandteile einer Turing-Maschine $M = [Z,\Sigma,\Gamma,q_0,q_e,\delta]$
    - endliche Menge Z von Zuständen
      - Startzustand $q_0 \in Z$
      - Endzustand $q_e \in Z$,
    - endliches Ein-/Ausgabealphabet Σ,
    - endliches Bandalphabet Γ mit $B \in \Gamma \setminus \Sigma$ und $\Sigma \subseteq \Gamma$,
    - Überführungsfunktion δ: $Z' \times \Gamma \rightarrow Z \times \Gamma \times \{ R,N,L \}$ mit $Z' = Z \setminus \{ q_e \}$.
- Verwendung
  - Eingaben und Ausgaben stehen auf dem Arbeitsband.
  - Programm "=" Überführungsfunktion δ
  - Wenn sich M im Endzustand befindet, werden bestimmte Zeichen auf dem Arbeitsband als Ausgabe interpretiert.

- komplizierteres Beispiel 3

  - Sei $\Sigma = \{ 0,1 \}$.
  - Wir wollen eine Turing-Maschine M angeben, die bei Eingabe einer Zeichenkette der Form 0^{2^i} mit $i \in N_0$ eine Eins ausgibt (sonst eine Null).
  - zugrunde liegende Idee:

1) M überprüft, ob auf dem Eingabeband genau eine Null vorkommt (falls ja, geht M zu (4)).
2) M überprüft, ob sich auf dem Eingabeband gerade viele Nullen befinden (falls nein, geht M zu (5)).
3) M streicht jede zweite Null auf dem Eingabeband und geht zu (1).
4) M schreibt die Ausgabe 1 aufs Arbeitsband.
5) M schreibt die Ausgabe 0 aufs Arbeitsband.

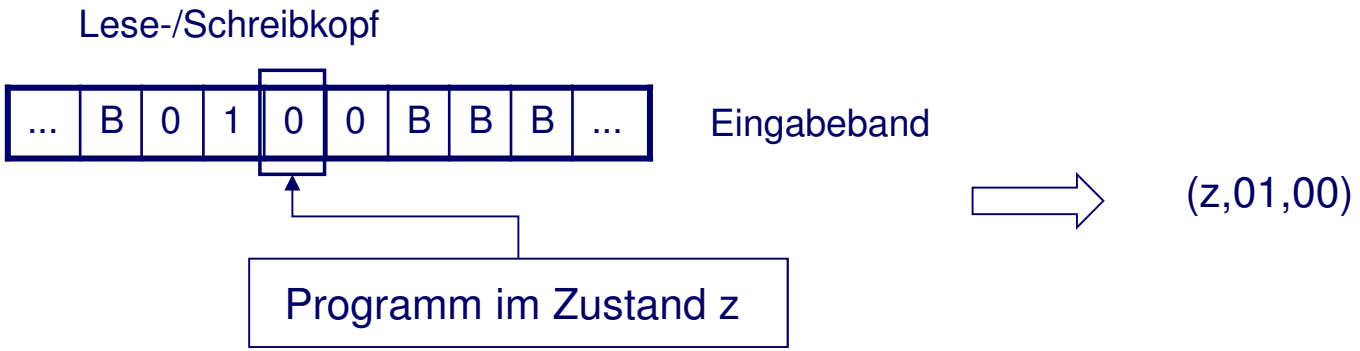
- komplizierteres Beispiel 3 (cont.)

$\delta(q_0,B) = (q_n,B,R)$	$\delta(q_0,0) = (q_1,0,R)$	$\delta(q_0,/) = (q_0,/,R)$
$\delta(q_1,B) = (q_p,B,R)$	$\delta(q_1,0) = (q_2,0,R)$	$\delta(q_1,/) = (q_1,/,R)$
$\delta(q_2,B) = (q_4,B,L)$	$\delta(q_2,0) = (q_3,0,R)$	$\delta(q_2,/) = (q_2,/,R)$
$\delta(q_3,B) = (q_n,B,R)$	$\delta(q_3,0) = (q_2,0,R)$	$\delta(q_3,/) = (q_3,/,R)$
$\delta(q_4,B) = (q_0,B,R)$	$\delta(q_4,0) = (q_5,0,L)$	$\delta(q_4,/) = (q_4,/,L)$
	$\delta(q_5,0) = (q_4,/,L)$	$\delta(q_5,/) = (q_5,/,L)$
$\delta(q_p,B) = (q_e,1,N)$		
$\delta(q_n,B) = (q_e,0,N)$		

$/\in \Gamma$ für „gestrichen“
... q_2 (gerade viele Nullen gelesen); q_3 (ungerade viele);
q_4 / q_5 (auf Rückweg Null abwechselnd überspringen/streichen)

Berechnungen einer Turing-Maschine – formal

- Sei $M = [Z,\Sigma,\Gamma,q_0,q_e,\delta]$ eine Turing-Maschine.
- Eine Konfiguration (Situation) in einer Berechnung von M wird beschrieben durch ein Tripel (z,u,v) mit $z \in Z$, $u \in \Gamma^*$ und $v \in \Gamma^*$.
  - z ist der aktuelle Zustand
  - u ist das Wort auf dem Eingabeband links vom Lese-/Schreibkopf (Unbesuchte Zellen …BBB werden nicht berücksichtigt.)
  - v ist das Wort auf dem Eingabeband „rechts“ vom Lese-/Schreibkopf (Unbesuchte Zellen BBB… werden nicht berücksichtigt.)



Berechnungen einer Turing-Maschine – formal (cont.)

- Sei $M = [Z,\Sigma,\Gamma,q_0,q_e,\delta]$ eine Turing-Maschine.
- Sei $z \in Z$ und $u, v \in \Gamma^*$
- Dann nennen wir die Konfiguration (z,u,v)
  - wenn $z = q_0$ und $u = \varepsilon$ gilt, eine Anfangskonfiguration,
  - wenn $z = q_e$ gilt, eine Endkonfiguration.
- Wenn (z,u,v) eine Anfangskonfiguration ist, gilt $v \in \Sigma^*$, und v ist die Eingabe von M
- Wenn (z,u,v) eine Endkonfiguration ist, gilt $v \in \Gamma^*$, und das längste Präfix v‘ von v mit $v' \in \Sigma^*$ ist die von M berechnete Ausgabe.

Berechnungen einer Turing-Maschine – formal (cont.)

- Sei $M = [Z,\Sigma,\Gamma,q_0,q_e,\delta]$ eine Turing-Maschine.
- M definiert eine Relation $\rightarrow_M$ über der Menge $Z \times \Gamma^* \times \Gamma^*$.

  Diese Relation beschreibt die einzelnen Rechenschritte von M:
  Was ist die nächste Konfiguration nach (z,u,v)?

- $(z,u,v) \rightarrow_M (z',ub,v')$, falls $v = av'$ und $\delta(z,a) = (z',b,R)$
- $(z,u,v) \rightarrow_M (z',ub,\varepsilon)$, falls $v = \varepsilon$ und $\delta(z,B) = (z',b,R)$
- $(z,u,v) \rightarrow_M (z',u,bv')$, falls $v = av'$ und $\delta(z,a) = (z',b,N)$
- $(z,u,v) \rightarrow_M (z',u,b)$, falls $v = \varepsilon$ und $\delta(z,B) = (z',b,N)$

- $(z,u,v) \rightarrow_M (z',u',cbv')$, falls $u = u'c$, $v = av'$ und $\delta(z,a) = (z',b,L)$
- $(z,u,v) \rightarrow_M (z',u',cb)$, falls $u = u'c$, $v = \varepsilon$ und $\delta(z,B) = (z',b,L)$
- $(z,u,v) \rightarrow_M (z',\varepsilon,Bbv')$, falls $u = \varepsilon$, $v = av'$ und $\delta(z,a) = (z',b,L)$
- $(z,u,v) \rightarrow_M (z',\varepsilon,Bb)$, falls $u = v = \varepsilon$ und $\delta(z,B) = (z',b,L)$

Anmerkungen

- Sei M eine Turing-Maschine und $x \in \Sigma^*$
- Bei der Verarbeitung von x kann M entweder
  - eine Endkonfiguration erreichen, oder aber auch …
  - keine Endkonfiguration erreichen,

  d.h. in eine Endlosschleife gehen oder mangels passender Anweisungen für die vorliegende Konfiguration steckenbleiben.

- zentraler Hilfsbegriff

  - sei $M = [Z,\Sigma, \Gamma,q_0,q_e,\delta]$ eine Turing-Maschine
  - die von M berechnete (partielle) Funktion f_M über Σ^* ist wie folgt definiert

Sei $x \in \Sigma^*$. Dann gilt:

- $f_M(x) = y$, falls M bei Eingabe x eine Endkonfiguration erreicht und die Ausgabe y bestimmt, d.h. es gilt $(q_0,\varepsilon,x) \rightarrow^*_M (q_e,u,yv)$, wobei $v = \varepsilon$ ist, oder der erste Buchstabe von v nicht zu Σ gehört.
- $f_M(x)$ ist undefiniert, wenn M bei Eingabe x keine Endkonfiguration erreicht.

... $(z,u,v) \rightarrow^*_M (z',u',v')$ bedeutet, dass M eine Konfiguration (z,u,v) in endlich vielen Rechenschritten in die Konfiguration (z',u',v') überführt

Anmerkung

- Meist interessieren wir uns nur für „vollständig“ definierte Funktionen
- Mit Turing-Maschinen kann man aber „partiell“ definierte Funktionen berechnen, d.h. Funktionen f, die nicht jedem x einen Funktionswert f(x) zuordnen

Das ist kein Unfall: Jedes hinreichend leistungsfähige Berechnungsmodell muss Programme enthalten, mit denen man „partiell“ definierte Funktionen berechnen kann,

und es muss möglich sein, „Endlosschleifen“ zu programmieren

- Drastisches Beispiel zum Thema *partielle* Funktion

Es sei das folgende Programm einer Turing-Maschine M für $\Sigma=\{0,1\}$ gegeben:

$$\delta(z_0,0) = (z_0,0,L)$$
$$\delta(z_0,1) = (z_0,1,L)$$
$$\delta(z_0,B) = (z_0,B,L)$$

- Drastisches Beispiel zum Thema *partielle* Funktion

Es sei das folgende Programm einer Turing-Maschine M für $\Sigma=\{0,1\}$ gegeben:

$$\delta(z_0,0) = (z_0,0,L)$$
$$\delta(z_0,1) = (z_0,1,L)$$
$$\delta(z_0,B) = (z_0,B,L)$$

M liefert für keinen Input ein Ergebnis und „rechnet“ stets ewig.
Die von M berechnete Funktion f_M auf Σ^* ist nirgendwo definiert .

- der formale Berechenbarkeits-Begriff

  - Sei Σ ein endliches Alphabet und f eine vollständig definierte Funktion über Σ^*
  - Dann definieren wir:

> Die Funktion f ist Turing-berechenbar, falls es eine Turing-Maschine M gibt, so dass für alle $x \in \Sigma^*$ gilt:
>
> - $f_M(x) = f(x)$

Diskussion

- beabsichtigte Verwendung
  - Wir wissen bereits aus Mächtigkeitsgründen:
    - Es gibt vollständig definierte Funktionen, die nicht Turing-berechenbar sind.
  - Da das Berechnungsmodell Turing-Maschine präzise definiert ist, kann man jetzt daran gehen, folgende Frage zu beantworten:
    - Gibt es „interessante“ vollständig definierte Funktionen, die nicht Turing-berechenbar sind?

Man muss „nur“ noch zeigen, dass es keine Turing-Maschine gibt, die man benutzen kann, um die entsprechende Funktion zu berechnen.