LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Formale Sprachen und Komplexität
Sommersemester 2019

Berechenbarkeitstheorie: Teil I

Prof. Dr. David Sabel

LFE Theoretische Informatik

Letzte Änderung der Folien: 27. Juni 2019

Inhalt

- Intuitive Berechenbarkeit und Churchsche These
- Turingmaschinen und Turingberechenbarkeit
- Konstruktion von Turingmaschinen
- LOOP-, WHILE-, GOTO-Programme und -Berechenbarkeit

Intuitive Berechenbarkeit

Motivation / Fragen

- Was kann mit einem Computerprogramm berechnet werden?
- Was kann nicht mit einem Computerprogramm berechnet werden?
- Aus der (Programmier-)Erfahrung:
  Gefühl dafür, was berechnet werden kann (und was nicht).
- Das ist der intuitive Begriff der Berechenbarkeit.
- Wie beweist man, dass etwas nicht berechnet werden kann?

Historie

- Berühmte Mathematiker / Informatiker versuchten in den 1930er Jahren den Begriff der Berechenbarkeit zu formalisieren
- dafür entwarfen sie verschiedene Modelle
- insbesondere sind zu nennen:
  - Alan Turing
  - Alonzo Church

Berechenbare Funktion

Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ nennen wir berechenbar, wenn

- es gibt einen Algorithmus einer modernen Programmiersprache, der f berechnet:
- Bei Eingabe $(n_1, \ldots, n_k)$ stoppt der Algorithmus nach endlich vielen Berechnungsschritten und liefert den Wert von $f(n_1, \ldots, n_k)$ als Ausgabe.
- Wenn $f(n_1, \ldots, n_k)$ undefiniert ist (f ist also eine partielle Funktion), dann stoppt der Algorithmus nicht.

Beispiele

Der Algorithmus

Eingabe: Zahl $n \in \mathbb{N}$
Beginn
  solange *true* **tue**
    skip

berechnet $f_1 : \mathbb{N} \to \mathbb{N}$ mit $f_1(x) =$ undefiniert für alle x

Der Algorithmus

Eingabe: Zahlen $n_1, n_2 \in \mathbb{N}$
Beginn
  hilf $:= n_1 + n_2$;
  return *hilf*

berechnet die Funktion $f_2 : (\mathbb{N} \times \mathbb{N}) \to \mathbb{N}$ mit $f_2(x, y) = x + y$.

f_1 und f_2 sind daher berechenbar

Beispiele (2)

$$f_3(n) = \begin{cases} 1, & \text{falls } n \text{ ein Präfix der Ziffern der Dezimalzahl-} \\ & \text{darstellung von } \pi \text{ ist} \\ 0, & \text{sonst} \end{cases}$$

Z.B. gilt

- $f_3(31) = 1$ und $f_3(314) = 1$
- $f_3(2) = 0$ und $f_3(315) = 0$

Ist f_3 berechenbar?

Ja. Sei n eine k-stellige Zahl. Es gibt Algorithmen, die die ersten k Stellen von π berechnen. Danach kann man vergleichen.

Beispiele (3)

$$f_4(n) = \begin{cases} 1, & \text{falls } n \text{ ein Teilwort der Ziffern} \\ & \text{der Dezimalzahldarstellung von } \pi \text{ ist} \\ 0, & \text{sonst} \end{cases}$$

Ist f_4 berechenbar?

Unklar. Es gibt die Vermutung, dass jede x-beliebige Ziffernfolge irgendwann als Folge in π auftaucht, aber die Frage ist bisher ungelöst.

Beispiele (4)

$$f_5(n) = \begin{cases} 1, & \text{falls die Dezimaldarstellung von } \pi \text{ das Wort } 3^m \\ & \text{mit } m \geq n \text{ als Teilwort besitzt.} \\ 0, & \text{sonst} \end{cases}$$

Ist f_5 berechenbar?

- Entweder: $f_5(n) = 1$ für alle n
- Oder: Es gibt m_0, sodass π 3^{m_0} als Teilwort hat, aber alle Teilworte 3^m mit $m > m_0$ nicht mehr besitzt.
  Dann ist $f_5(n) = \begin{cases} 1, \text{falls } n \leq m_0 \\ 0, \text{falls } n > m_0 \end{cases}$
- Egal, welcher Fall zutrifft, wir können für beide Fälle Algorithmen angeben, die f_5 berechnen.

Beachte: Unsere Definition von Berechenbarkeit ist nicht konstruktiv:
Wir müssen keinen Algorithmus liefern, sondern nur einen Beweis, dass der Algorithmus existiert.

Beispiele (5)

$$f_6(n) = \begin{cases} 1, & \text{falls deterministische LBAs genau die gleichen} \\ & \text{Sprachen erkennen wie nichtdeterministische LBAs} \\ 0, & \text{sonst} \end{cases}$$

Ist f_6 berechenbar?

Ja. Entweder ist $f_6(x) = 1$ oder $f_6(x) = 0$.
Beide Funktionen sind berechenbar.
(Unabhängig davon, ob das 1. LBA Problem eine positive oder negative Lösung hat)

Beispiele (6)

$$f^r(n) = \begin{cases} 1, & \text{falls } n \text{ ein Präfix der Ziffern der} \\ & \text{Dezimalzahldarstellung von } r \text{ ist} \\ 0, & \text{sonst} \end{cases}$$

Ist f_r für jedes $r \in \mathbb{R}$ berechenbar?

Nein. Wir bräuchten genauso viele verschiedene Algorithmen wie es reelle Zahlen gibt.
Es gibt nur abzählbar viele Algorithmen einer Programmiersprache, aber überabzählbar viele reelle Zahlen

Churchsche These

Im folgenden untersuchen wir Modelle zur Berechenbarkeit

- Turingmaschinen
- WHILE-Programme
- GOTO-Programme
- μ-rekursive Funktionen

Resultat: Alle führen zum selben Begriff der Berechenbarkeit.

Churchsche These:

Die Klasse der Turingberechenbaren (äquivalent WHILE-berechenbaren, GOTO-berechenbaren, μ-rekursiven) Funktionen stimmt genau mit der Klasse der intuitiv berechenbaren Funktionen überein.

Die Churchsche These kann man nicht beweisen, da der Begriff „intuitiv berechenbar“ nicht formal gefasst werden kann.

Turingmaschinen: Wiederholung

- Ein Turingmaschine ist ein 7-Tupel $(Z, \Sigma, \Gamma, \delta, z_0, \square, E)$ mit Zuständen Z, Eingabealphabet Σ, Bandalphabet $\Gamma \supset \Sigma$, Blank-Symbol $\square$, Startzustand z_0, Endzuständen $E \subseteq Z$, und Überführungsfunktion δ.
- DTM: $\delta : (Z \times \Gamma) \to (Z \times \Gamma \times \{L, R, N\})$
- NTM: $\delta : (Z \times \Gamma) \to \mathcal{P}(Z \times \Gamma \times \{L, R, N\})$
- TM-Konfiguration $a_1 \cdots a_m z a_{m+1} \cdots a_n$, wobei $a_1 \cdots a_n \in \Gamma^*$ der entdeckte Teil des Bands, TM ist in Zustand z und Kopf ist unter a_{m+1}.

Turingberechenbarkeit

Definition (Turingberechenbarkeit)

Sei $bin(n)$ die Binärdarstellung von $n \in \mathbb{N}$.

Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ heißt Turingberechenbar, falls es eine (deterministische) Turingmaschine $M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$ gibt, so dass für alle $n_1, \ldots, n_k, m \in \mathbb{N}$ gilt:

$$f(n_1, \ldots, n_k) = m$$
g.d.w.
$$z_0 bin(n_1) \# \ldots \# bin(n_k) \vdash^* \square \ldots \square z_e bin(m) \square \ldots \square \text{ mit } z_e \in E.$$

Eine Funktion $f : \Sigma^* \to \Sigma^*$ heißt Turingberechenbar, falls es eine (deterministische) Turingmaschine $M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$ gibt, so dass für alle $u, v \in \Sigma^*$ gilt

$$f(u) = v \text{ g.d.w. } z_0 u \vdash^* \square \ldots \square z_e v \square \ldots \square \text{ mit } z_e \in E.$$

Konsequenzen

Eine Konsequenz der Definition ist:

Falls $f(n_1, \ldots, n_k)$ bzw. $f(u)$ undefiniert ist,
dann muss die Maschine in eine Endlosschleife gehen,
d. h. sie darf nicht in einen Endzustand gehen

Beispiele

Nachfolgerfunktion

Die Funktion $f(x) = x + 1$ für alle $x \in \mathbb{N}$ ist Turingberechenbar.
Wir haben eine entsprechende Turingmaschine bereits angegeben.

Identitätsfunktion

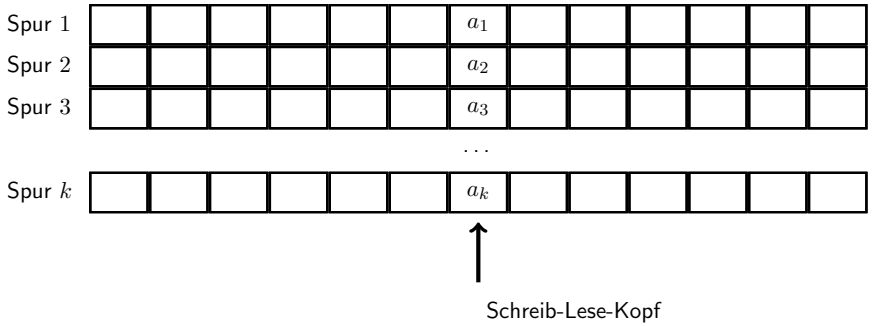
Die Funktion $f(x) = x$ für alle $x \in \mathbb{N}$ ist Turingberechenbar:
Für die Turingmaschine $M = (\{z_0\}, \{0, 1, \#\}, \{0, 1, \#\square\}, \delta, z_0, \square, \{z_0\})$ mit $\delta(z_0, a) = (z_0, a, N)$ für alle $a \in \{0, 1, \#, \square\}$ gilt:
$z_0 bin(n) \vdash^* z_0 bin(n)$ für alle $n \in \mathbb{N}$.

Überall undefinierte Funktion

Die Funktion $f(x) = \perp$ für alle x ist Turingberechenbar, da die TM $M = (\{z_0\}, \{0, 1, \#\}, \{0, 1, \#, \square\}, \delta, z_0, \square, \emptyset)$ mit $\delta(z_0, a) = (z_0, a, N)$ für keine Eingabe akzeptiert (sondern stets in eine Endlosschleife geht).

Mehrspuren-Turingmaschinen (1)

Erweiterung: Das Band hat k-Spuren:



Mehrspuren-Turingmaschinen (2)

Definition (Mehrspuren-Turingmaschine)

Für $k \in \mathbb{N}_{>0}$ ist eine **k-Spuren-Turingmaschine** ein 7-Tupel $M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$ wobei

- Z ist eine endliche Menge von Zuständen,
- Σ ist das (endliche) Eingabealphabet,
- $\Gamma \supset \Sigma$ ist das (endliche) Bandalphabet,
- δ ist die Zustandsüberführungsfunktion:
  - für eine DTM: $\delta : (Z \times \Gamma^k) \to Z \times \Gamma^k \times \{L, R, N\}$
  - für eine NTM: $\delta : (Z \times \Gamma^k) \to \mathcal{P}(Z \times \Gamma^k \times \{L, R, N\})$
- $z_0 \in Z$ ist der Startzustand,
- $\square \in \Gamma \setminus \Sigma$ ist das Blank-Symbol und
- $E \subseteq Z$ ist die Menge der Endzustände.

Mehrspuren-Turingmaschinen (3)

Berechenbarkeit: Ein- und Ausgabe auf der ersten Spur.

Satz

Jede Mehrspuren-Turingmaschine kann auf einer 1-Band-Turingmaschine simuliert werden.

Beweis: Konstruktion der 1-Band-TM mit einer Spur:

- Verwende als Bandalphabet $\Gamma \cup \Gamma^k$
- Eingabealphabet Σ und Blank-Symbol $\square$.
- Aus Eingabe w aus Σ^* erzeugt die TM die Mehrspurendarstellung:
  Ersetze $a \in \Sigma$ durch k-Tupel $(a, \square, \ldots, \square)$.
- Anschließend wir die Mehrspurenmaschine simuliert
- Nach Akzeptanz der Mehrspurenmaschine:
  Erzeuge 1-Spuren-Darstellung, d.h. ersetze alle $(a_1, \ldots, a_k)$ durch a_1. □

Beispiel einer Mehrspurenmaschine (1)

- TM, die $\{wcw \mid w \in \{a, b\}^+\}$ erkennt.
- Wir konzentrieren uns auf die 2-Spurendarstellung, daher: TM die $\{w(c, \square)w \mid w \in \{(a, \square), (b, \square)\}^+\}$ erkennt

Ideen:

- Eingabe wcw auf Spur 1 wird nur gelesen nicht verändert.
- Auf Spur 2 wird nur $\square$ und ✓ zum Markieren von Zeichen auf Spur 1 verwendet.
- Markieren: Erst ein Zeichen im linken w, dann selbes Zeichen im rechten w
- Zustände $Z = \{z_0, z_{1a}, z_{1b}, z_{2a}, z_{2b}, z_3, \ldots, z_9\}$
  $z_{1a}, z_{1b}, z_{2a}, z_{2b}$ „speichern“ das gelesene Zeichen (a oder b).
  - z_0 ist der Startzustand.
  - z_8 ist der einzige akzeptierende Zustand.
  - z_9 ist Müllzustand (zum Verwerfen)

Beispiel einer Mehrspurenmaschine (2)

Übergangsfunktion δ

- $\delta(z_0,(s,\square)) = (z_{1s},(s,\checkmark),R)$ für $s \in \{a,b\}$: markiere linkestes unmarkiertes Zeichen s im linken w, speichere s in z_{1s}, Suche nach s im rechten w beginnt.
- $\delta(z_{1s},(s',\square)) = (z_{1s},(s',\square),R)$ für $s,s' \in \{a,b\}$: laufe nach rechts zum c
- $\delta(z_{1s},(c,\square)) = (z_{2s},(c,\square),R)$ mit $s \in \{a,b\}$: c wurde erreicht
- $\delta(z_{2s},(s',\checkmark)) = (z_{2s},(s',\checkmark),R)$ mit $s,s' \in \{a,b\}$: suche erstes unmarkiertes Symbol im rechten w.
- $\delta(z_{2s},(s,\square)) = (z_3,(s,\checkmark),L)$ mit $s \in \{a,b\}$: richtiges unmarkiertes Zeichen im rechten w .
- $\delta(z_3,(s,\checkmark)) = (z_3,(s,\checkmark),L)$ mit $s \in \{a,b\}$: nach links laufen zum c.
- $\delta(z_3,(c,\square)) = (z_4,(c,\square,L))$ c von rechts wieder erreicht.
- ...

Beispiel einer Mehrspurenmaschine (3)

- ...
- $\delta(z_4,(s,\square)) = (z_5,(s,\square),L)$ mit $s \in \{a,b\}$: links vom c kein markiertes Zeichen: zu z_5
- $\delta(z_4,(s,\checkmark)) = (z_6,(s,\square),R)$ mit $s \in \{a,b\}$: links vom c ist markiertes Zeichen: Prüfe, dass alle Zeichen im rechten w markiert sind (mit z_6).
- $\delta(z_5,(s,\square)) = (z_5,(s,\square),L)$ mit $s \in \{a,b\}$: suche erstes markiertes Zeichen im linken w
- $\delta(z_5,(s,\checkmark)) = (z_0,(s,\checkmark),R)$ mit $s \in \{a,b\}$: erstes markiertes Zeichen im linken w gefunden , gehe zu z_0
- $\delta(z_6,(c,\square)) = (z_7,(c,\square),R)$: suche im rechten w nach unmarkierten Zeichen
- $\delta(z_7,(s,\checkmark)) = (z_7,(s,\checkmark),R)$ für $s \in \{a,b\}$: suche im rechten w nach unmarkierten Zeichen
- $\delta(z_7,(\square,\square)) = (z_8,(\square,\square),N)$: alle Zeichen im rechten w markiert, akzeptiere in z_8
- $\delta(z_8,(s,t)) = (z_8,(s,t),N)$ für $s \in \{a,b,c\}$, $t \in \{\square,\checkmark\}$ verbleibe akzeptierend.
- $\delta(z_i,(s,t)) = (z_9,(s,t),N)$ für alle anderen Fälle verbleibe oder wechsle in den verwerfenden Zustand.

Beispiel einer Mehrspurenmaschine (4)

Ein Lauf der Turingmaschine für das Wort $abcab$ ist:

$$z_0 \begin{matrix}a&b&c&a&b\\ \square&\square&\square&\square&\square\end{matrix} \vdash \begin{matrix}a\\ \checkmark\end{matrix} z_{1a} \begin{matrix}b&c&a&b\\ \square&\square&\square&\square\end{matrix} \vdash \begin{matrix}a&b\\ \checkmark&\square\end{matrix} z_{1a} \begin{matrix}c&a&b\\ \square&\square&\square\end{matrix} \vdash \begin{matrix}a&b&c\\ \checkmark&\square&\square\end{matrix} z_{2a} \begin{matrix}a&b\\ \square&\square\end{matrix}$$

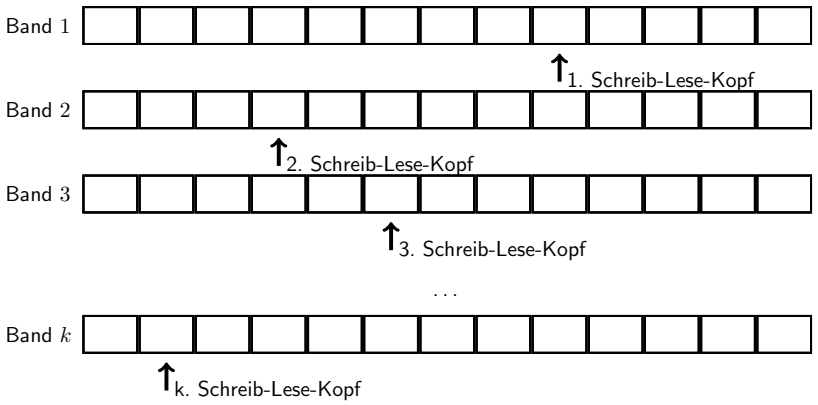
$$\vdash \begin{matrix}a&b\\ \checkmark&\square\end{matrix} z_3 \begin{matrix}c&a&b\\ \square&\checkmark&\square\end{matrix} \vdash \begin{matrix}a\\ \checkmark\end{matrix} z_4 \begin{matrix}b&c&a&b\\ \square&\square&\checkmark&\square\end{matrix} \vdash z_5 \begin{matrix}a&b&c&a&b\\ \checkmark&\square&\square&\checkmark&\square\end{matrix} \vdash \begin{matrix}a\\ \checkmark\end{matrix} z_0 \begin{matrix}b&c&a&b\\ \square&\square&\checkmark&\square\end{matrix}$$

$$\vdash \begin{matrix}a&b\\ \checkmark&\checkmark\end{matrix} z_{1b} \begin{matrix}c&a&b\\ \square&\checkmark&\square\end{matrix} \vdash \begin{matrix}a&b&c\\ \checkmark&\checkmark&\square\end{matrix} z_{2b} \begin{matrix}a&b\\ \checkmark&\square\end{matrix} \vdash \begin{matrix}a&b&c&a\\ \checkmark&\checkmark&\square&\checkmark\end{matrix} z_{2b} \begin{matrix}b\\ \square\end{matrix} \vdash \begin{matrix}a&b&c\\ \checkmark&\checkmark&\square\end{matrix} z_3 \begin{matrix}a&b\\ \checkmark&\checkmark\end{matrix}$$

$$\vdash \begin{matrix}a&b\\ \checkmark&\checkmark\end{matrix} z_3 \begin{matrix}c&a&b\\ \square&\checkmark&\checkmark\end{matrix} \vdash \begin{matrix}a\\ \checkmark\end{matrix} z_4 \begin{matrix}b&c&a&b\\ \checkmark&\square&\checkmark&\checkmark\end{matrix} \vdash \begin{matrix}a&b\\ \checkmark&\checkmark\end{matrix} z_6 \begin{matrix}c&a&b\\ \square&\checkmark&\checkmark\end{matrix} \vdash \begin{matrix}a&b&c\\ \checkmark&\checkmark&\square\end{matrix} z_7 \begin{matrix}a&b\\ \checkmark&\checkmark\end{matrix}$$

$$\vdash \begin{matrix}a&b&c&a\\ \checkmark&\checkmark&\square&\checkmark\end{matrix} z_7 \begin{matrix}b\\ \checkmark\end{matrix} \vdash \begin{matrix}a&b&c&a&b\\ \checkmark&\checkmark&\square&\checkmark&\checkmark\end{matrix} z_7 \begin{matrix}\square\\ \square\end{matrix} \vdash \begin{matrix}a&b&c&a&b\\ \checkmark&\checkmark&\square&\checkmark&\checkmark\end{matrix} z_8 \begin{matrix}\square\\ \square\end{matrix}$$

Mehrbandmaschinen



Schreib-Lese-Köpfe bewegen sich unabhängig!

Mehrbandmaschinen (2)

Definition (Mehrband-Turingmaschine)

Für $k \in \mathbb{N}_{>0}$ ist eine k-Band-Turingmaschine ein 7-Tupel $M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$ wobei

- Z ist eine endliche Menge von Zuständen,
- Σ ist das (endliche) Eingabealphabet,
- $\Gamma \supset \Sigma$ ist das (endliche) Bandalphabet,
- δ ist die Zustandsüberführungsfunktion
  - für eine DTM: $\delta : (Z \times \Gamma^k) \to (Z \times \Gamma^k \times \{L, R, N\}^k)$
  - für eine NTM: $\delta : (Z \times \Gamma^k) \to \mathcal{P}(Z \times \Gamma^k \times \{L, R, N\}^k)$
- $z_0 \in Z$ ist der Startzustand,
- $\square \in \Gamma \setminus \Sigma$ ist das Blank-Symbol
- $E \subseteq Z$ ist die Menge der Endzustände.

Berechnung: Eingabe auf dem ersten Band, alle anderen leer
Ausgabe auf dem ersten Band

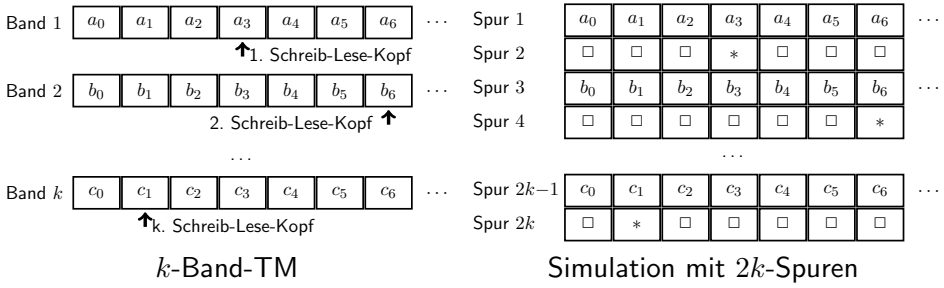
Mehrbandmaschinen (3)

Theorem

Jede Mehrband-TM kann von einer 1-Band TM simuliert werden.

Beweis:

- Sei M eine k-Band-TM. Wenn $k = 1$, dann nichts zu zeigen.
- Für $k > 1$ verwenden wir eine $2 \cdot k$-Spuren-TM um M zu simulieren.



k-Band-TM | Simulation mit $2k$-Spuren

Mehrbandmaschinen (4)

- Eingabe $w \in \Sigma^*$ auf dem Eingabeband der Mehrband-Maschine
- 1-Band-Maschine erzeugt Darstellung in $2k$-Spuren
- 1-Band-Maschine simuliert anschließend Berechnungsschritte.
- Bei Akzeptanz der Mehrband-TM transformiert die 1-Band-TM die Spurendarstellung in die Darstellung der Ausgabe.
- Simulation eines Berechnungsschrittes der Mehrband-TM:
  1) lese den verwendeten Bandbereich von links nach rechts
  2) speichere alle k Kopfpositionen (durch Zustände)
  3) führe Schritt der Mehrband-TM aus, durch Anpassen der Bandinhalte und der Kopfpositionen □

Konstruktion von TMs und Notation

- Wenn M eine 1-Band-Turingmaschine ist, dann schreiben wir $M(i,k)$ für die k-Band-Turingmaschine (mit $i \leq k$), die die Operationen von M auf dem i. Band durchführt und alle anderen Bänder unverändert lässt.
- Wenn k nicht von Bedeutung (und groß genug gewählt werden kann), schreiben wir $M(i)$ statt $M(i,k)$

- TM die 1 addiert (bereits gesehen) nennen wir

  „Band := Band+1“

- mit obiger Notation „Band := Band+1“(i)
- andere Notation „Band i := Band $i+1$“

Konstruktion von TMs (2)

Weitere TMs (Konstruktionen sind einfach)

- „Band $i :=$ Band $i-1$“:
  k-Band-TM ($k \geq i$), die eine angepasste Subtraktion von 1 auf Band i durchführt. Anpassung: $0 - 1 = 0$
- „Band $i := 0$“:
  k-Band-TM ($k \geq i$), die Band i mit 0 überschreibt.
- „Band $i :=$ Band j“:
  k-Band-TM ($k \geq i$ und $k \geq j$), welche Zahl von Band j auf Band i kopiert

Hintereinanderschaltung

Seien $M_i = (Z_i, \Sigma, \Gamma_i, \delta_i, z_i, \square, E_i)$, für $i = 1, 2$, k-Band-TMs.

Die TM $M_1; M_2$ führt M_1 und M_2 hintereinandergeschaltet aus:

- O.B.d.A. $Z_1 \cap Z_2 = \emptyset$
- $M_1; M_2 = ((Z_1 \cup Z_2), \Sigma, \Gamma_1 \cup \Gamma_2, \delta, z_1, \square, E_2)$ mit

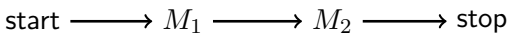
$$\delta(z,(a_1,\ldots,a_k)) = \begin{cases} \delta(z,(a_1,\ldots,a_k)) & \text{falls } z \in Z_1 \backslash E_1, (a_1,\ldots,a_k) \in \Gamma_1^k \\ \delta'(z,(a_1,\ldots,a_k)) & \text{falls } z \in Z_2,\ (a_1,\ldots,a_k) \in \Gamma_2^k \\ (z_2,(a_1,\ldots,a_k),N) & \text{falls } z \in E_1,\ (a_1,\ldots,a_k) \in \Gamma_1^k \end{cases}$$

Die TM $M_1; M_2$

- führt erst M_1 aus,
- wechselt im Endzustand $z \in E_1$ in Startzustand z_2 von M_2
- führt anschließend M_2 aus.

Hintereinanderschaltung von TMs (2)

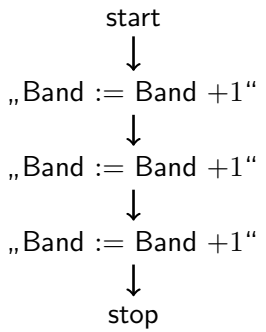
Flussdiagramm für $M_1; M_2$



Hintereinanderschaltung von TMs (3)

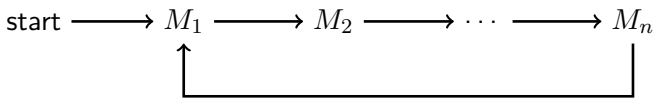
Beispiel:
„Band := Band+3“ wird konstruiert durch
„Band := Band+1“;„Band := Band+1“;„Band := Band+1“

Flussdiagramm dazu



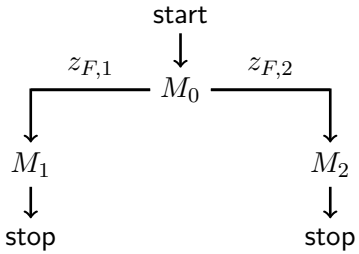
Hintereinanderschaltung von TMs (4)

Zyklische Verkettung von $M_1, \ldots, M_n$:



Verzweigende Fortsetzung

Seien M_0, M_1, M_2 TMs, $z_{F,1}$ und $z_{F,2}$ die Endzustände von M_0



Konstruktion fügt Übergänge

$\delta(z_{F,1}, (a_1, \ldots, a_k)) = (z_{0,M_1}, (a_1, \ldots, a_k), N)$ und
$\delta(z_{F,2}, (a_1, \ldots, a_k)) = (z_{0,M_2}, (a_1, \ldots, a_k), N)$

ein, wobei z_{0,M_i} Startzustand von M_i (für $i = 1, 2$).

TM: Test auf 0

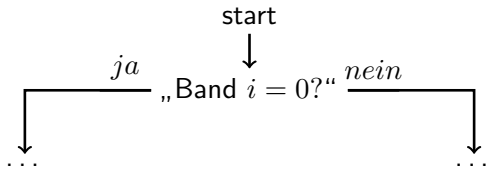
Beispiel: M_0 als TM, die Zustände $\{z_0, z_1, ja, nein\}$ hat und

$$\begin{array}{lcll} \delta(z_0, a) & = & (nein, a, N) & \text{für } a \neq 0 \\ \delta(z_0, 0) & = & (z_1, 0, R) & \\ \delta(z_1, a) & = & (nein, a, L) & \text{für } a \neq \square \\ \delta(z_1, \square) & = & (ja, \square, L) & \end{array}$$

mit ja und $nein$ Endzustände und z_0 Startzustand

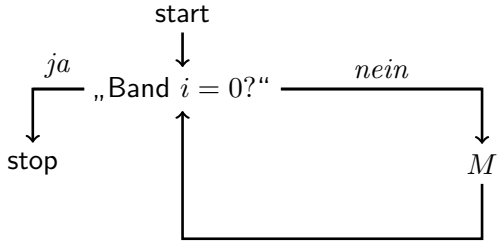
TM M_0 prüft, ob das Band eine 0 enthält oder nicht.

Notation: „Band=0?“ und „Band $i = 0$?“ (statt „Band= 0?“(i))



TM: Schleife

Mit Verzweigung, „Band i=0?“, (zyklischer) Hintereinanderschaltung und einer TM M erstellen wir



Die TM M wird solange wieder aufgerufen,
bis das i. Band die Zahl 0 enthält.

Die Maschine nennen wir „WHILE Band $i \neq 0$ DO M“.

Fazit zu den Konstruktionen

Programme einer einfachen imperativen Programmiersprache mit Zuweisungen, Verzweigungen und Schleifen können durch TMs simuliert werden.

LOOP-, WHILE-, GOTO-Programme

Ziel

- Betrachte drei einfache imperative Programmiersprachen:
  - LOOP-Programme
  - WHILE-Programme
  - GOTO-Programme
- und die dazugehörigen Berechenbarkeitsbegriffe
- Welche Berechenbarkeitsbegriffe sind gleich / verschieden (untereinander aber auch bezüglich Turingberechenbarkeit)?

LOOP-Programme: Syntax

LOOP-Programme werden durch CFG (V, Σ, P, Prg) erzeugt, wobei:

$$
\begin{array}{rcl}
V & = & \{Prg, Var, Id, Cons\} \\
\Sigma & = & \{\textbf{LOOP}, \textbf{DO}, \textbf{END}, x, 0, \ldots, 9, ;, :=, +, -\} \\
P & = & \{Prg \quad \to \quad \textbf{LOOP}\ Var\ \textbf{DO}\ Prg\ \textbf{END} \\
 & & \qquad\quad \mid Prg; Prg \\
 & & \qquad\quad \mid Var := Var + Const \\
 & & \qquad\quad \mid Var := Var - Const \\
 & & Var \quad \to \quad x_{Id} \\
 & & Const \quad \to \quad Id \\
 & & Id \quad \to \quad 0 \mid 1 \mid \ldots \mid 9 \mid 1Const \mid 2Const \mid \ldots\ 9Const\}
\end{array}
$$

Beachte:

- Var erzeugt Variablen $x_0, x_1, x_2, \ldots$
- $Cons$ erzeugt alle natürlichen Zahlen

LOOP-Programme: Semantik (Modellierung des Speichers)

Definition (Variablenbelegung)

Eine Variablenbelegung ρ ist eine endliche Abbildung mit Einträgen $x_i \mapsto n$ mit x_i ist Variable und $n \in \mathbb{N}$.

$$\text{Wir definieren } \rho(x_i) := \begin{cases} n, & \text{wenn } x_i \mapsto n \in \rho \\ 0, & \text{sonst} \end{cases}$$

Wir schreiben $\rho\{x_i \mapsto m\}$ für

$$\rho\{x_i \mapsto m\}(x_j) = \begin{cases} \rho(x_j), & \text{wenn } x_j \neq x_i \\ m, & \text{wenn } x_j = x_i \end{cases}$$

LOOP-Programme: Semantik (Berechnungsschritte)

Definition (Berechnungsschritt $\xrightarrow[\text{LOOP}]{}$)

Berechnungsschritt $(\rho, P) \xrightarrow[\text{LOOP}]{} (\rho', P')$, wobei ρ, ρ' Variablenbelegungen und P, P' LOOP-Programme oder ε (leeres Programm)

- $(\rho, x_i := x_j + c) \xrightarrow[\text{LOOP}]{} (\rho', \varepsilon)$ wobei $\rho' = \rho\{x_i \mapsto n\}$ und $n = \rho(x_j) + c$
- $(\rho, x_i := x_j - c) \xrightarrow[\text{LOOP}]{} (\rho', \varepsilon)$ wobei $\rho' = \rho\{x_i \mapsto n\}$ und $n = \max(0, \rho(x_j) - c)$
- $(\rho, P_1; P_2) \xrightarrow[\text{LOOP}]{} (\rho', P_1'; P_2))$ wenn $(\rho, P_1) \xrightarrow[\text{LOOP}]{} (\rho', P_1')$ und $P_1' \neq \varepsilon$
- $(\rho, P_1; P_2) \xrightarrow[\text{LOOP}]{} (\rho', P_2))$ wenn $(\rho, P_1) \xrightarrow[\text{LOOP}]{} (\rho', \varepsilon)$
- $(\rho, \textbf{LOOP}\ x_i\ \textbf{DO}\ P\ \textbf{END}) \xrightarrow[\text{LOOP}]{} (\rho, \underbrace{P; \ldots; P}_{\rho(x_i)\text{-mal}})$

Wir schreiben $\xrightarrow[\text{LOOP}]{}^i$ für i Schritte und $\xrightarrow[\text{LOOP}]{}^*$ für 0 oder beliebig viele Schritte

LOOP-Programme: Beispiel für die Semantik

Wir betrachten das Programm

$$x_2 := x_1 + 1;$$
$$\textbf{LOOP}\ x_2\ \textbf{DO}\ x_3 := x_3 + 1\ \textbf{END}$$

und die Variablenbelegung $\{x_1 \mapsto 2\}$.

$$(\{x_1 \mapsto 2\}, x_2 := x_1 + 1; \textbf{LOOP}\ x_2\ \textbf{DO}\ x_3 := x_3 + 1)$$
$$\xrightarrow[\text{LOOP}]{} (\{x_1 \mapsto 2, x_2 \mapsto 3\}, \textbf{LOOP}\ x_2\ \textbf{DO}\ x_3 := x_3 + 1)$$

da $(\{x_1 \mapsto 2\}, x_2 := x_1 + 1) \xrightarrow[\text{LOOP}]{} (\{x_1 \mapsto 2, x_2 \mapsto 3\}, \varepsilon)$

$$\xrightarrow[\text{LOOP}]{} (\{x_1 \mapsto 2, x_2 \mapsto 3\}, x_3 := x_3 + 1; x_3 := x_3 + 1; x_3 := x_3 + 1)$$
$$\xrightarrow[\text{LOOP}]{} (\{x_1 \mapsto 2, x_2 \mapsto 3, x_3 \mapsto 1\}, x_3 := x_3 + 1; x_3 := x_3 + 1)$$

da $(\{x_1 \mapsto 2, x_2 \mapsto 3\}, x_3 := x_3 + 1) \xrightarrow[\text{LOOP}]{} (\{x_1 \mapsto 2, x_2 \mapsto 3, x_3 \mapsto 1\}, \varepsilon)$

$$\xrightarrow[\text{LOOP}]{} (\{x_1 \mapsto 2, x_2 \mapsto 3, x_3 \mapsto 2\}, x_3 := x_3 + 1)$$

da $(\{x_1 \mapsto 2, x_2 \mapsto 3, x_3 \mapsto 1\}, x_3 := x_3 + 1) \xrightarrow[\text{LOOP}]{} (\{x_1 \mapsto 2, x_2 \mapsto 3, x_3 \mapsto 2\}, \varepsilon)$

$$\xrightarrow[\text{LOOP}]{} (\{x_1 \mapsto 2, x_2 \mapsto 3, x_3 \mapsto 3\}, \varepsilon)$$

LOOP-Berechenbarkeit

Definition (LOOP-berechenbare Funktion)

Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ heißt LOOP-berechenbar, wenn es ein LOOP-Programm P gibt, sodass für alle $n_1, \ldots, n_k \in \mathbb{N}$ und Variablenbelegungen $\rho = \{x_1 \mapsto n_1, \ldots, x_k \mapsto n_k\}$ gilt:

$$(\rho, P) \xrightarrow[\text{LOOP}]{}^* (\rho', \varepsilon) \text{ und } \rho'(x_0) = f(n_1, \ldots, n_k)$$

D.h. das LOOP-Programm

- empfängt Eingaben über die Variablen $x_1, \ldots, x_k$
- liefert Ergebnis in Variable x_0

LOOP-Berechenbarkeit: Beispiel

Die Funktion $f(n_1) = n_1 + c$ ist LOOP-berechenbar.

Das Programm $x_0 := x_1 + c$ belegt dies, denn für alle $n_1 \in \mathbb{N}$:

$$(\{x_1 = n_1\}, x_0 := x_1 + c) \xrightarrow[\text{LOOP}]{} (\{x_0 = n_1 + c, x_1 = n_1\}, \varepsilon)$$

LOOP-Programme terminieren stets

Satz

Alle LOOP-Programme terminieren. Daher sind alle LOOP-berechenbaren Funktionen total.

Beweis: Zeige Für alle (ρ, P): $\exists j_{P,\rho} \in \mathbb{N}, \rho'$: $(\rho, P) \xrightarrow[\text{LOOP}]{}^{j_{P,\rho}} (\rho', \varepsilon)$.

Beweis mit Induktion über die Struktur von P.

- Basis: Für $(\rho, x_i := x_j \pm c)$ wird genau 1 Schritt benötigt.
- Für Sequenzen $P_1; P_2$ und ρ liefert die Induktionshypothese $j_{P_1,\rho}$ und $j_{P_2,\rho'}$ mit $(\rho, P_1; P_2) \xrightarrow[\text{LOOP}]{}^{j_{P_1,\rho}} (\rho', P_2) \xrightarrow[\text{LOOP}]{}^{j_{P_2,\rho'}} (\rho'', \varepsilon)$.
- Für **LOOP** x_i **DO** P **END** liefert die Induktionshypothese j_{P,ρ_i} und ρ_i mit $(\rho_1, \textbf{LOOP } x_i \textbf{ DO } P \textbf{ END}) \xrightarrow[\text{LOOP}]{} (\rho_1, P; \ldots; P) \xrightarrow[\text{LOOP}]{}^{j_{P,\rho_1}}$ $(\rho_2, P; \ldots; P) \xrightarrow[\text{LOOP}]{}^{j_{P,\rho_2}} \ldots \xrightarrow[\text{LOOP}]{}^{i_{P,\rho_n}} (\rho_{n+1}, \varepsilon)$ mit $n = \rho_1(x_i)$

LOOP-Berechenbarkeit

- Da es partielle Turingberechenbare Funktionen gilt:

  Es gibt Turingberechenbare Funktionen,
  die **nicht** LOOP-berechenbar sind.

  Z.B. die überall undefinierte Funktion.
- Es gilt sogar:

  Es gibt intuitiv berechenbare Funktionen,
  die total sind,
  aber trotzdem nicht LOOP-berechenbar sind

  (ein Beispiel ist die Ackermannfunktion, siehe später).

Kodierung weiterer Befehle mit LOOP-Programmen

Befehl: $x_i := c$

Kodierung: $x_i := x_n + c$
wobei x_n keine der Eingabevariablen ist
(dann gilt $\rho(x_n) = 0$)

Befehl: $x_i := x_j$

Kodierung: $x_i := x_j + 0$

Befehl: **IF** $x_i = 0$ **THEN** P **END**

Kodierung: $x_n := 1$;
LOOP x_i **DO** $x_n := 0$ **END**;
LOOP x_n **DO** P **END**
wobei x_n eine Variable ist, die nicht in P und nicht in der Eingabe vorkommt.

Kodierung weiterer Befehle mit LOOP-Programmen (2)

Befehl: **IF** $x_i = 0$ **THEN** P_1 **ELSE** P_2 **END**

Kodierung: $x_n := 1$;
$x_m := 1$;
LOOP x_i **DO** $x_n := 0$ **END**;
LOOP x_n **DO** $x_m := 0; P_1$ **END**;
LOOP x_m **DO** P_2 **END**
wobei x_n, x_m nicht in der Eingabe
und nicht sonst irgendwo im Programm vorkommen

Kompliziertere if-Bedingungen gehen analog.

Kodierung weiterer Befehle mit LOOP-Programmen (3)

Befehl: $x_i := x_j + x_l$

Kodierung: $x_i := x_j$;
LOOP x_l **DO** $x_i := x_i + 1$ **END**

- zeigt auch, dass die Additionsfunktion $f(x_1, x_2) = x_1 + x_2$ LOOP-berechenbar ist.
- andere Rechenoperationen (wie *, mod div) gehen analog

WHILE-Programme: Syntax

WHILE-Programme werden durch die CFG (V, Σ, P, Prg) erzeugt:

$$
\begin{aligned}
V &= \{Prg, Var, Id, Cons\} \\
\Sigma &= \{\textbf{WHILE}, \textbf{LOOP}, \textbf{DO}, \textbf{END}, x, 0, \ldots, 9, ;, :=, +, -\} \\
P &= \{Prg \to \textbf{WHILE}\ Var\ \neq 0\ \textbf{DO}\ Prg\ \textbf{END} \\
&\quad \mid \textbf{LOOP}\ Var\ \textbf{DO}\ Prg\ \textbf{END} \\
&\quad \mid Prg; Prg \\
&\quad \mid Var := Var + Const \\
&\quad \mid Var := Var - Const \\
&\quad Var \to x_{Id} \\
&\quad Const \to Id \\
&\quad Id \to 0 \mid 1 \mid \ldots \mid 9 \mid 1Const \mid 2Const \mid \ldots\ 9Const\}
\end{aligned}
$$

Beachte: WHILE-Programme **erweitern** LOOP-Programme um das **WHILE**-Konstrukt

WHILE-Programme: Semantik (Berechnungsschritte)

Definition (Berechnungsschritt $\xrightarrow[\text{WHILE}]{}$)

Berechnungsschritt $(\rho, P) \xrightarrow[\text{WHILE}]{} (\rho', P')$, wobei ρ, ρ' Variablenbelegungen und P, P' WHILE-Programme oder ε (leeres Programm)

- $(\rho, x_i := x_j + c) \xrightarrow[\text{WHILE}]{} (\rho', \varepsilon)$ wobei $\rho' = \rho\{x_i \mapsto n\}$ und $n = \rho(x_j) + c$
- $(\rho, x_i := x_j - c) \xrightarrow[\text{WHILE}]{} (\rho', \varepsilon)$ wobei $\rho' = \rho\{x_i \mapsto n\}$ und $n = \max(0, \rho(x_j) - c)$
- $(\rho, P_1; P_2) \xrightarrow[\text{WHILE}]{} (\rho', P_1'; P_2))$ wenn $(\rho, P_1) \xrightarrow[\text{WHILE}]{} (\rho', P_1')$ und $P_1' \neq \varepsilon$
- $(\rho, P_1; P_2) \xrightarrow[\text{WHILE}]{} (\rho', P_2))$ wenn $(\rho, P_1) \xrightarrow[\text{WHILE}]{} (\rho', \varepsilon)$
- $(\rho, \textbf{LOOP}\ x_i\ \textbf{DO}\ P\ \textbf{END}) \xrightarrow[\text{WHILE}]{} (\rho, \underbrace{P; \ldots; P}_{\rho(x_i)\text{-mal}})$
- $(\rho, \textbf{WHILE}\ x_i \neq 0\ \textbf{DO}\ P\ \textbf{END}) \xrightarrow[\text{WHILE}]{} (\rho, \varepsilon)$ wenn $\rho(x_i) = 0$
- $(\rho, \textbf{WHILE}\ x_i \neq 0\ \textbf{DO}\ P\ \textbf{END}) \xrightarrow[\text{WHILE}]{} (\rho, P; \textbf{WHILE}\ x_i \neq 0\ \textbf{DO}\ P\ \textbf{END})$ wenn $\rho(x_i) \neq 0$

Mit $\xrightarrow[\text{WHILE}]{}^i$ bezeichnen wir i-Schritte und mit $\xrightarrow[\text{WHILE}]{}^*$ 0 oder beliebig viele Schritte

WHILE-Berechenbarkeit

Definition (WHILE-berechenbare Funktion)

Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ heißt WHILE-berechenbar, wenn es ein WHILE-Programm P gibt, sodass

- für alle $n_1, \ldots, n_k \in \mathbb{N}$, sodass $f(n_1, \ldots, n_k)$ definiert ist: Für Variablenbelegung $\rho = \{x_1 \mapsto n_1, \ldots, x_k \mapsto n_k\}$ gilt $(\rho, P) \xrightarrow[\text{WHILE}]{}^* (\rho', \varepsilon)$ und $\rho'(x_0) = f(n_1, \ldots, n_k)$.
- Falls $f(n_1, \ldots, n_k)$ nicht definiert ist, so stoppt das Programm P nicht, d. h. für alle $i \in \mathbb{N}$ gibt es ein ρ', sodass $(\rho, P) \xrightarrow[\text{WHILE}]{}^i (\rho', P')$

LOOP-berechenbar $\Longrightarrow$ WHILE-berechenbar

Satz

Jede LOOP-berechenbare Funktion ist auch WHILE-berechenbar.

Beweis: Offensichtlich, da jedes LOOP-Programm auch ein WHILE-Programm ist, und die Semantik identisch dafür ist

LOOP durch WHILE

Das **LOOP**-Konstrukt ist überflüssig in WHILE:

Ersetze **LOOP** x_i **DO** P **END**

durch $x_m := x_i;$
WHILE $x_m \neq 0$ **DO** $x_m := x_m - 1; P$ **END**
wobei x_m nirgends sonst vorkommt

WHILE-berechenbar $\Longrightarrow$ Turingberechenbar

Theorem 10.2.4

Jede WHILE-berechenbare Funktion ist auch Turingberechenbar.

Beweis:

- Wir haben bereits gezeigt, dass Zuweisung, Sequenz und **WHILE** durch Turingmaschinen darstellbar sind.
- Dabei liegen die Variablen x_i jeweils auf Band i einer Mehrband-TM
- **LOOP** wird vorher in **LOOP**-freies WHILE-Programm transformiert.

GOTO-Programme: Syntax

GOTO-Programme werden durch die CFG (V, Σ, P, Prg) erzeugt:

$$
\begin{aligned}
V &= \{Prg, Var, Id, Cons\} \\
\Sigma &= \{\textbf{GOTO}, \textbf{HALT}, \textbf{IF}, x, 0, \ldots, 9, ;, :=, +, -\} \\
P &= \{Prg \rightarrow M_{Id} : Var := Var + Const \\
&\quad \mid M_{Id} : Var := Var - Const \\
&\quad \mid M_{Id} : \textbf{GOTO}\ M_{Id} \\
&\quad \mid M_{Id} : \textbf{IF}\ x_i = 0\ \textbf{GOTO}\ M_{Id} \\
&\quad \mid M_{Id} : \textbf{HALT} \\
&\quad \mid Prg; Prg \\
&\quad Var \rightarrow x_{Id} \\
&\quad Const \rightarrow Id \\
&\quad Id \rightarrow 0 \mid 1 \mid \ldots \mid 9 \mid 1Const \mid 2Const \mid \ldots 9Const\}
\end{aligned}
$$

- Befehle sind mit M_i : markiert. Wenn unwichtig, lassen wir sie weg.
- Nebenbedingung: Alle Markierungen verschieden.
- Normalisiertes GOTO-Programm: $M_1 : A_1; \ldots; M_n : A_n$
  durch konsistentes Umbenennen der Markierungen herstellbar

GOTO-Programme: Semantik (Berechnungsschritte)

Definition (Berechnungsschritt $\xrightarrow[\text{GOTO}]{P_0}$)

Sei P_0 ein GOTO-Programm, ρ eine Variablenbelegung und P ein GOTO-Programm. Dann ist ein Berechnungsschritt $\xrightarrow[\text{GOTO}]{P_0}$ durch die folgenden Fälle definiert:

- $(\rho, M_i : x_j := x_k + c; P) \xrightarrow[\text{GOTO}]{P_0} (\rho[x_j \mapsto n], P)$, wobei $n = \rho(x_k) + c$
- $(\rho, M_i : x_j := x_k + c) \xrightarrow[\text{GOTO}]{P_0} (\rho[x_j \mapsto n], \varepsilon)$, wobei $n = \rho(x_k) + c$
- $(\rho, M_i : x_j := x_k - c; P) \xrightarrow[\text{GOTO}]{P_0} (\rho[x_j \mapsto n], P)$, wobei $n = \max(0, \rho(x_k) - c)$
- $(\rho, M_i : x_j := x_k - c) \xrightarrow[\text{GOTO}]{P_0} (\rho[x_j \mapsto n], \varepsilon)$, wobei $n = \max(0, \rho(x_k) - c)$
- $(\rho, M_i : \textbf{GOTO } M_k) \xrightarrow[\text{GOTO}]{M_1:A_1;\ldots;M_n:A_n} (\rho, M_k : A_k; \ldots; M_n : A_n)$ wenn $1 \leq i \leq n$
- $(\rho, M_i : \textbf{GOTO } M_k; P) \xrightarrow[\text{GOTO}]{M_1:A_1;\ldots;M_n:A_n} (\rho, M_k : A_k; \ldots; M_n : A_n)$ wenn $1 \leq i \leq n$
- $\ldots$

GOTO-Programme: Semantik (Berechnungsschritte, Forts.)

- $\ldots$
- $(\rho, M_i : \textbf{IF } x_r = 0 \textbf{ THEN GOTO } M_k; P) \xrightarrow[\text{GOTO}]{M_1:A_1;\ldots;M_n:A_n} (\rho, M_k : A_k; \ldots; M_n : A_n)$ wenn $1 \leq i \leq n$ und $\rho(x_r) = 0$
- $(\rho, M_i : \textbf{IF } x_r = 0 \textbf{ THEN GOTO } M_k) \xrightarrow[\text{GOTO}]{M_1:A_1;\ldots;M_n:A_n} (\rho, M_k : A_k; \ldots; M_n : A_n)$ wenn $1 \leq i \leq n$ und $\rho(x_r) = 0$
- $(\rho, M_i : \textbf{IF } x_r = 0 \textbf{ THEN GOTO } M_k; P) \xrightarrow[\text{GOTO}]{P_0} (\rho, P)$ wenn $\rho(x_r) \neq 0$
- $(\rho, M_i : \textbf{IF } x_r = 0 \textbf{ THEN GOTO } M_k) \xrightarrow[\text{GOTO}]{P_0} (\rho, \varepsilon)$ wenn $\rho(x_r) \neq 0$
- $(\rho, M_i : \textbf{HALT}; P) \xrightarrow[\text{GOTO}]{P_0} (\rho, M_i : \textbf{HALT})$

Wir schreiben $\xrightarrow[\text{GOTO}]{P_0}^*$ für 0 oder beliebig viele Berechnungsschritte und $\xrightarrow[\text{GOTO}]{P_0}^i$ für genau i Berechnungsschritte (mit dem Programm P_0).

GOTO-Berechenbarkeit

Definition (GOTO-berechenbare Funktion)

Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ heißt GOTO-berechenbar, wenn es ein normalisiertes GOTO-Programm P gibt, sodass

- für alle $n_1, \ldots, n_k \in \mathbb{N}$, sodass $f(n_1, \ldots, n_k)$ definiert ist: Für Variablenbelegung $\rho = \{x_1 \mapsto n_1, \ldots, x_k \mapsto n_k\}$ gilt: $(\rho, P) \xrightarrow[\text{GOTO}]{P}^* (\rho', \textbf{HALT})$ und $\rho'(x_0) = f(n_1, \ldots, n_k)$.
- Falls $f(n_1, \ldots, n_k)$ nicht definiert ist, so stoppt das Programm P nicht, d.h. für alle $i \in \mathbb{N}$ gibt es ρ' und P' sodass: $(\rho, P) \xrightarrow[\text{GOTO}]{P}^i (\rho', P')$

WHILE-berechenbar $\Longrightarrow$ GOTO-berechenbar

Jedes WHILE-Programm ist durch ein GOTO-Programm simulierbar:

- Klar für $x_j := x_k \pm c$ und $P_1; P_2$
- **LOOP**-Schleife durch **WHILE**-Schleifen ersetzen
- **WHILE** $x_i \neq 0$ **DO** P **END** kann durch

$$\begin{array}{ll} M_1: & \textbf{IF } x_i = 0 \textbf{ THEN GOTO } M_j \\ \ldots & P'; \\ M_{j-1}: & \textbf{GOTO } M_1; \\ M_j: & \ldots \end{array}$$

  simuliert werden, wobei P' das GOTO-Programm zu P ist.
- **HALT** am Ende einfügen

WHILE-berechenbar $\Longrightarrow$ GOTO-berechenbar (2)

Sei P_W WHILE-Programm und P_G das beschriebene GOTO-Programm.

Dann gilt:

- Falls $(\rho, P_W) \xrightarrow[\text{WHILE}]{}^* (\rho', \varepsilon)$, dann $(\rho, P_G) \xrightarrow[\text{GOTO}]{P_G}^* (\rho'', \textbf{HALT})$ mit $\rho'(x_0) = \rho''(x_0)$.
- Falls $\forall i \in \mathbb{N} : (\rho, P_W) \xrightarrow[\text{WHILE}]{}^i (\rho_i, P_{W,i})$ dann auch $\forall i \in \mathbb{N} : (\rho, P_G) \xrightarrow[\text{GOTO}]{P_G}^i (\rho'_i, P_{G,i})$.

Beweisbar durch Induktion über die gegebenen Sequenzen.

Damit folgt:

Jede WHILE-berechenbare Funktion ist auch GOTO-berechenbar

GOTO-berechenbar $\Longrightarrow$ WHILE-berechenbar

Sei $M_1 : A_1; \ldots; M_n : A_n$ ein normalisiertes GOTO-Programm, sodass x_M darin vorkommt. Passendes WHILE-Programm:

$x_M := 1;$
WHILE $x_M \neq 0$ **DO**
  IF $x_M = 1$ **THEN** P_1 **END**;
  $\ldots$
  IF $x_M = n$ **THEN** P_n **END**;
  IF $x_M = n+1$ **THEN** $x_M := 0$ **END**
END

wobei P_i das folgende Programm ist:

- $x_j := x_k \pm c; x_M := x_M + 1$; falls $A_i = x_j := x_k \pm c$
- $x_M := j$ falls $A_i =$ **GOTO** M_j
- **IF** $x_k = 0$ **THEN** $x_M := j$ **ELSE** $x_M := x_M + 1$ **END**, falls $A_i =$ **IF** $x_k = 0$ **THEN GOTO** M_j
- $x_M := 0$ falls $A_i =$ **HALT**

GOTO-berechenbar $\Longrightarrow$ WHILE-berechenbar (2)

Es gilt:

- $(\rho, M_1 : A_1; \ldots; M_n : A_n) \xrightarrow[\text{GOTO}]{M_1:A_1;\ldots;M_n:A_n}^* (\rho', \textbf{HALT})$ g.d.w. $(\rho, P) \xrightarrow[\text{WHILE}]{}^* (\rho'', \varepsilon)$ wobei $\rho'(x_0) = \rho''(x_0)$
- Falls $\forall i \in \mathbb{N} : (\rho, M_1 : A_1; \ldots; M_n : A_n) \xrightarrow[\text{GOTO}]{M_1:A_1;\ldots;M_n:A_n}^i (\rho_i, P_i)$, dann auch $\forall i \in \mathbb{N} : (\rho, P) \xrightarrow[\text{WHILE}]{}^i (\rho'_i, P'_i)$

Damit folgt:

Jede GOTO-berechenbare Funktion ist auch WHILE-berechenbar.

GOTO-berechenbar $\Longleftrightarrow$ WHILE-berechenbar

Theorem 10.4.1

WHILE- und GOTO-Berechenbarkeit sind äquivalente Begriffe. D.h. jede WHILE-berechenbare Funktion ist auch GOTO-berechenbar und umgekehrt ist jede GOTO-berechenbare Funktion auch WHILE-berechenbar.

Die Übersetzungen zeigen auch:

Satz

WHILE-Programme benötigen nur eine WHILE-Schleife.

Turingberechenbar $\Longrightarrow$ GOTO-berechenbar (1)

Ziel: GOTO-Programm simuliert TM
Idee:

- Stelle TM-Konfiguration durch Zahlen dar
- Zahlen werden in Programmvariablen abgelegt.
- Es werden am Ende nur 3 Programmvariablen benutzt!

Darstellung von Worten als natürliche Zahlen

- Sei Bandalphabet $\Gamma = \{a_1, \ldots, a_m\}$
- Zeichen a_i wird mit Index i identifiziert
- D.h. $a_{i_1} \cdots a_{i_n} \in \Gamma^*$ äquivalent zur Folge $(i_1 \ldots i_n)$
- Sei $b > m = |\Gamma|$
- $a_{i_1} \ldots a_{i_n}$: Zahl im Stellenwertsystem zur Basis b (beachte: 0-wertige Ziffer nicht in $a_1, \ldots, a_m$)

  Dezimalwert von $a_{i_1} \cdots a_{i_n}$: $(i_1 \cdots i_n)_b = \sum_{j=1}^{n} i_j \cdot b^{n-j}$

Umkehrung: Berechne $(i_1 \cdots i_n)$ aus einer natürlichen Zahl:

- Teile wiederholt mit Rest durch Basis b
- Liste der Reste: $r_0, \ldots, r_m$.
- Ergibt $(r_m \cdots r_0)$ zur Basis b
  repräsentieren daher das Wort $a_{r_m} \cdots a_{r_0}$.

Beispiel

Sei $\Gamma = \{\square, 0, 1\}$, sodass $a_1 = \square, a_2 = 0, a_3 = 1$.

Betrachte $\square 110 \square$, d.h. $a_1 a_3 a_3 a_2 a_1$, sei $b = 4$

Identifiziere $\square 110 \square$ mit (13321) und
$(13321)_b = (13321)_4 = 1 * 4^4 + 3 * 4^3 + 3 * 4^2 + 2 * 4^1 + 1 * 4^0 =$
$256 + 3 * 64 + 3 * 16 + 2 * 4 + 1 * 1 = 505$

Umgekehrt ergibt

$$\begin{array}{rcll} 505/4 & = & 126 & \text{Rest } 1 \\ 126/4 & = & 31 & \text{Rest } 2 \\ 31/4 & = & 7 & \text{Rest } 3 \\ 7/4 & = & 1 & \text{Rest } 3 \\ 1/4 & = & 0 & \text{Rest } 1 \end{array}$$

und daher ergibt dies die Zahl (13321) zur Basis b, welche das Wort $a_1 a_3 a_3 a_2 a_1 = \square 110 \square$ repräsentiert.

Turingberechenbar $\Longrightarrow$ GOTO-berechenbar (2)

TM-Konfiguration im GOTO-Programm:

Konfiguration der TM mit $\Gamma = \{a_1, \ldots, a_m\}$:

$$a_{i_1} \cdots a_{i_n} z_k a_{j_1} \cdots a_{j_m}$$

Im GOTO-Programm durch 3 Variablen x_p, x_z, x_s dargestellt:

$$\begin{array}{rcl} x_p & = & (i_1 \cdots i_n)_b \\ x_z & = & k \\ x_s & = & (j_m \cdots j_1)_b \end{array}$$

Beachte: x_s verwendet umgekehrte Reihenfolge der Ziffern

GOTO-Programme für Transitionen

Für $\delta(z_k, a_{j_1}) = (z_l, a_{j'}, Q)$ mit $Q \in \{N, L, R\}$ sei $P(k, j_1)$ das zugehörige GOTO-Programm.

Turingberechenbar $\Longrightarrow$ GOTO-berechenbar (3)

Fall $\delta(z_k, a_{j_1}) = (z_l, a_{j'}, L)$
Transition ist

$$a_{i_1} \cdots a_{i_n} z_k a_{j_1} \cdots a_{j_m} \vdash a_{i_1} \cdots a_{i_{n-1}} z_l a_{i_n} a_{j'} a_{j_2} \cdots a_{j_m},$$

falls $n > 0$. Bezüglich der Zahlendarstellung gilt daher:

- x_z muss auf den Wert l aktualisiert werden
- x_p muss von $(i_1 \cdots i_n)_b$ zu $(i_1 \cdots i_{n-1})_b$ aktualisiert werden
- x_s muss von $(j_m \cdots j_1)_b$ zu $(j_m \cdots j_2 j' i_n)_b$ aktualisiert werden

Das Programm $P(k, j_1)$ dazu

$x_z := l$; // neuer Zustand z_l
$x_s := x_s \text{ div } b$; // Abschneiden der letzten Stelle $(j_m \cdots j_1) \to (j_m \cdots j_2)$
$x_s := b * x_s + j'$; // j' als letzte Stelle hinzufügen $(j_m \cdots j_2) \to (j_m \cdots j_2 j')$
$x_s := b * x_s + (x_p \bmod b)$; // $i_n = x_p \bmod b$ am Endes hinzufügen $(j_m \cdots j_2 j') \to (j_m \cdots j_2 j' i_n)$
$x_p := x_p \text{ div } b$// Abschneiden der letzten Stelle $(i_1 \cdots i_n) \to (i_1 \cdots i_{n-1})$

Turingberechenbar $\Longrightarrow$ GOTO-berechenbar (4)

Fall $\delta(z_k, a_{j_1}) = (z_l, a_{j'}, L)$
Für den Grenzfall

$$z_k a_{j_1} \cdots a_{j_m} \vdash z_l \square a_{j_1} \cdots a_{j_m}$$

setze zuerst $x_p := r$; wobei $\square = a_r$ und führe dann das Programm für den allgemein Fall aus.

Turingberechenbar $\Longrightarrow$ GOTO-berechenbar (5)

Fall $\delta(z_k, a_{j_1}) = (z_l, a_{j'}, N)$
Transition ist

$$a_{i_1} \cdots a_{i_n} z_k a_{j_1} \cdots a_{j_m} \vdash a_{i_1} \cdots a_{i_n} z_l a_{j'} a_{j_2} \cdots a_{j_m}$$

- x_z muss auf den Wert l aktualisiert werden
- x_p bleibt unverändert.
- x_s muss von $(j_m \cdots j_1)_b$ zu $(j_m \cdots j_2 j')_b$ aktualisiert werden

Das Programm $P(k, j_1)$ sei dann:

$$\begin{aligned} x_z &:= l; \\ x_s &:= x_s \text{ div } b; \\ x_s &:= b * x_s + j' \end{aligned}$$

Turingberechenbar $\Longrightarrow$ GOTO-berechenbar (6)

Fall $\delta(z_k, a_{j_1}) = (z_l, a_{j'}, R)$
Transition ist

$$a_{i_1} \cdots a_{i_n} z_k a_{j_1} \cdots a_{j_m} \vdash a_{i_1} \cdots a_{i_n} a_{j'} z_l a_{j_2} \cdots a_{j_m},$$

falls $m > 1$. Bezüglich der Zahlendarstellung gilt daher:

- x_z muss auf den Wert l aktualisiert werden
- x_p muss von $(i_1 \cdots i_n)_b$ zu $(i_1 \cdots i_n j')_b$ aktualisiert werden
- x_s muss von $(j_m \cdots j_1)_b$ zu $(j_m \cdots j_2)_b$ aktualisiert werden

Das Programm $P(k, j_1)$ sei dann:

$$\begin{aligned} x_z &:= l; \\ x_p &:= b * x_p + j'; \\ x_s &:= x_s \text{ div } b \end{aligned}$$

Turingberechenbar $\Longrightarrow$ GOTO-berechenbar (7)

Fall $\delta(z_k, a_{j_1}) = (z_l, a_{j'}, R)$
Für den Fall $m = 1$ gilt

$$a_{i_1} \cdots a_{i_n} z_k a_{j_1} \vdash a_{i_1} \cdots a_{i_n} a'_j z_l \square$$

In diesem Fall sei P_{k,j_1} das Programm

$$\begin{aligned} x_z &:= l; \\ x_p &:= b * x_p + j'; \\ x_s &:= r \end{aligned}$$

wobei $\square = a_r$.

Beachte, das wir die Sonderfälle abfragen können, durch testen, ob x_p bzw. x_s gleich zu 0 ist.

Turingberechenbar $\Longrightarrow$ GOTO-berechenbar (8)

Simulation der TM durch das folgende GOTO-Program:
(Eingaben in $x_1, \ldots, x_k$, Ausgabe in x_0)

$$\begin{aligned} M_1 &: P_1; \\ M_2 &: P_2; \\ M_3 &: P_3 \end{aligned}$$

- P_1 erzeugt für die Eingabe in $x_1, \ldots, x_k$ die Binärdarstellung und dann die Darstellung der TM-Konfiguration in x_p, x_z, x_s
- P_3 verwendet die Darstellung der Endkonfiguration, um die Ausgabe in der Ausgabevariablen x_0 zu erzeugen.
- P_2: Nächste Folie

Turingberechenbar $\Longrightarrow$ GOTO-berechenbar (9)

Programmteil P_2 hat die Form:

M_2 : $x_a := x_s \bmod b$;
IF $(x_z = 1)$ **and** $(x_a = 1)$ **THEN GOTO** $M_{1,1}$;
IF $(x_z = 1)$ **and** $(x_a = 2)$ **THEN GOTO** $M_{1,2}$;
...
IF $(x_z = q)$ **and** $(x_a = m)$ **THEN GOTO** $M_{q,m}$

$M_{1,1}$: $P'_{1,1}$;
GOTO M_2;

$M_{1,2}$: $P'_{1,2}$;
GOTO M_2;

...

$M_{q,m}$: $P'_{q,m}$
GOTO M_2

Programm $P'_{i,j}$ ist **GOTO** M_3, wenn z_i Endzustand ist und $P_{i,j}$ sonst.

Turingberechenbar $\Longrightarrow$ GOTO-berechenbar (10)

Insgesamt zeigt das GOTO-Programm:

Satz 10.5.2

GOTO-Programme können Turingmaschinen simulieren. Jede Turingberechenbare Funktion ist auch GOTO-berechenbar.

Fazit

Theorem

Turingberechenbarkeit, WHILE-Berechenbarkeit und GOTO-Berechenbarkeit sind äquivalente Begriffe.