Konstruktionen von Turingmaschinen und LOOP-Programme

Prof. Dr. Jasmin Blanchette

Lehr- und Forschungseinheit für
Theoretische Informatik

Stand: 13. Juni 2023
Folien ursprünglich von PD Dr. David Sabel

Wiederholung: 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, sodass für alle $n_1, \ldots, n_k, m \in \mathbb{N}$ gilt:

$$f(n_1, \ldots, n_k) = m$$

g.d.w.

$$z_0 bin(n_1)\# \cdots \# bin(n_k) \vdash^* \square \cdots \square z_e bin(m) \square \cdots \square \text{ mit } z_e \in E$$

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

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

Wiederholung: Mehrbandmaschinen

Definition (Mehrband-Turingmaschine)

Eine *k*-Band-Turingmaschine ($k \in \mathbb{N}_{>0}$) ein 7-Tupel $M = (Z, \Sigma, \Gamma, \delta, z_0, \Box, E)$ 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,
- $\Box \in \Gamma \setminus \Sigma$ ist das Blank-Symbol und
- $E \subseteq Z$ ist die Menge der Endzustände.

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-ten Band durchführt und alle anderen Bänder unverändert lässt.
- Wenn k nicht von Bedeutung (und groß genug gewählt werden kann), schreiben wir $M(i)$ statt $M(i, k)$.

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-ten Band durchführt und alle anderen Bänder unverändert lässt.
- Wenn k nicht von Bedeutung (und groß genug gewählt werden kann), schreiben wir $M(i)$ statt $M(i, k)$.
- Die TM, die 1 addiert 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 die Zahl von Band j auf Band i kopiert.

Hintereinanderschaltung von TMs (1)

Seien $M_i = (Z_i, \Sigma, \Gamma_i, \delta_i, z_{0,i}, \square, E_i)$, 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. $Z_1 \cap Z_2 = \emptyset$.
- $M_1; M_2 = (Z_1 \cup Z_2, \Sigma, \Gamma_1 \cup \Gamma_2, \delta, z_{0,1}, \square, E_2)$ mit

$$\delta(z,(a_1,\ldots,a_k)) = \begin{cases} \delta_1(z,(a_1,\ldots,a_k)), & \text{falls } z \in Z_1 \setminus E_1, (a_1,\ldots,a_k) \in \Gamma_1^k \\ (z_{0,2},(a_1,\ldots,a_k),N^k), & \text{falls } z \in E_1,\ (a_1,\ldots,a_k) \in \Gamma_1^k \\ \delta_2(z,(a_1,\ldots,a_k)), & \text{falls } z \in Z_2,\ (a_1,\ldots,a_k) \in \Gamma_2^k \end{cases}$$

Die TM $M_1; M_2$

- führt erst M_1 aus,
- wechselt im Endzustand $z \in E_1$ in Startzustand $z_{0,2}$ von M_2,
- führt anschließend M_2 aus.

Hintereinanderschaltung von TMs (2)

Flussdiagramm für $M_1; M_2$

$$\text{start} \longrightarrow M_1 \longrightarrow M_2 \longrightarrow \text{stop}$$

Hintereinanderschaltung von TMs (3)

Beispiel:

„Band := Band + 3“ wird konstruiert durch

„Band := Band + 1“;„Band := Band + 1“;„Band := Band + 1“

Flussdiagramm dazu:

$$\begin{array}{c} \text{start} \\ \downarrow \\ \text{„Band := Band + 1“} \\ \downarrow \\ \text{„Band := Band + 1“} \\ \downarrow \\ \text{„Band := Band + 1“} \\ \downarrow \\ \text{stop} \end{array}$$

Hintereinanderschaltung von TMs (4)

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

$$
\begin{tikzcd}
\text{start} \arrow[r] & M_1 \arrow[r] & M_2 \arrow[r] & \cdots \arrow[r] & M_n \arrow[llll, bend left, to path={-- ([yshift=-4ex]\tikztostart.south) -| (\tikztotarget)}]
\end{tikzcd}
$$

Verzweigende Fortsetzung

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

$$\begin{tikzcd}
& \text{start} \arrow[d] & \\
M_1 \arrow[d] & M_0 \arrow[l, "z_{E,1}"'] \arrow[r, "z_{E,2}"] & M_2 \arrow[d] \\
\text{stop} & & \text{stop}
\end{tikzcd}$$

Konstruktion fügt Übergänge

$$\delta(z_{E,1}, (a_1, \ldots, a_k)) = (z_{0,1}, (a_1, \ldots, a_k), N) \text{ und}$$
$$\delta(z_{E,2}, (a_1, \ldots, a_k)) = (z_{0,2}, (a_1, \ldots, a_k), N)$$

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

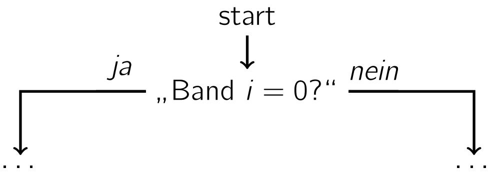
Beispiel: Test auf 0

Folgende TM M_0 prüft, ob das Band eine 0 enthält oder nicht.
M_0 hat die Zustände $\{z_0, z_1, ja, nein\}$ und

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

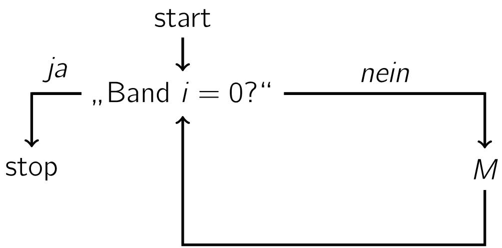
mit z_0 Startzustand und *ja* und *nein* Endzustände.

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-te 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:

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

Beachte:

- Var erzeugt Variablen $x_0, x_1, x_2, \ldots$
- $Const$ 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 $n \in \mathbb{N}$.

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

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

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

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

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

Variablenbelegung: $\{x_1 \mapsto 2\}$

Ausführung:

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

LOOP-Programme: Beispiel für die Semantik

Programm: $x_2 := x_1 + 1;$ Variablenbelegung: $\{x_1 \mapsto 2\}$

$\mathbf{LOOP}\ x_2\ \mathbf{DO}\ x_3 := x_3 + 1\ \mathbf{END}$

Ausführung:

$$(\{x_1 \mapsto 2\}, x_2 := x_1 + 1; \mathbf{LOOP}\ x_2\ \mathbf{DO}\ x_3 := x_3 + 1)$$

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

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

LOOP-Programme: Beispiel für die Semantik

Programm: $x_2 := x_1 + 1;$ $\quad$ Variablenbelegung: $\{x_1 \mapsto 2\}$

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

Ausführung:

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

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

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

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

LOOP-Programme: Beispiel für die Semantik

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

Variablenbelegung: $\{x_1 \mapsto 2\}$

Ausführung:

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

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

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

LOOP-Programme: Beispiel für die Semantik

Programm: $x_2 := x_1 + 1;$ **LOOP** x_2 **DO** $x_3 := x_3 + 1$ **END**

Variablenbelegung: $\{x_1 \mapsto 2\}$

Ausführung:

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

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

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

LOOP-Programme: Beispiel für die Semantik

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

Variablenbelegung: $\{x_1 \mapsto 2\}$

Ausführung:

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

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

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 $n_1, \ldots, n_k \in \mathbb{N}$ gilt $(\rho, P) \xrightarrow[\text{LOOP}]{}^* (\rho', \varepsilon)$, wobei $\rho = \{x_1 \mapsto n_1, \ldots, x_k \mapsto n_k\}$ 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 $n_1 \in \mathbb{N}$:

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

LOOP-Programme terminieren stets

Satz

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

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

Beweis mit Induktion über die Struktur von P.

- Basis: Für $(\rho, x_i := x_j \pm c)$ wird genau 1 Schritt benötigt.
- Für Sequenzen $P_1; P_2$ liefert die Induktionshypothese j_1 und j_2 mit $(\rho, P_1; P_2) \xrightarrow[\text{LOOP}]{}{}^{j_1} (\rho', P_2) \xrightarrow[\text{LOOP}]{}{}^{j_2} (\rho'', \varepsilon)$.
  Es werden genau $j_1 + j_2$ Schritte benötigt.
- Für **LOOP** x_i **DO** P **END** liefert die Induktionshypothese j_i's und ρ_i's mit $(\rho_1, \textbf{LOOP}\ x_i\ \textbf{DO}\ P\ \textbf{END}) \xrightarrow[\text{LOOP}]{} (\rho_1, P; \ldots; P) \xrightarrow[\text{LOOP}]{}{}^{j_1} (\rho_2, P; \ldots; P) \xrightarrow[\text{LOOP}]{}{}^{j_2}$ $\cdots \xrightarrow[\text{LOOP}]{}{}^{j_n} (\rho_{n+1}, \varepsilon)$ mit $n = \rho_1(x_i)$.
  Es werden genau $1 + j_1 + j_2 + \cdots + j_n$ Schritte benötigt.

LOOP-Berechenbarkeit

- Da es partielle Turingberechenbare Funktionen gibt, 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 (1)

Befehl: $x_i := c$

Kodierung: $x_i := x_n + c$

wobei x_n keine der Eingabevariablen ist und
an keiner anderen Stelle im Programm verwendet wird
(dann gilt $\rho(x_n) = 0$)

Kodierung weiterer Befehle mit LOOP-Programmen (1)

Befehl: $x_i := c$

Kodierung: $x_i := x_n + c$

wobei x_n keine der Eingabevariablen ist und
an keiner anderen Stelle im Programm verwendet wird
(dann gilt $\rho(x_n) = 0$)

Befehl: $x_i := x_j$

Kodierung: $x_i := x_j + 0$

Kodierung weiterer Befehle mit LOOP-Programmen (1)

Befehl: $x_i := c$

Kodierung: $x_i := x_n + c$

wobei x_n keine der Eingabevariablen ist und
an keiner anderen Stelle im Programm verwendet wird
(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 nicht in der Eingabe und nicht in P vorkommt

Kodierung weiterer Befehle mit LOOP-Programmen (2)

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

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

Kompliziertere if-Bedingungen gehen auch.

Kodierung weiterer Befehle mit LOOP-Programmen (3)

Befehl:	$x_i := x_j + x_k$
Kodierung:	$x_\ell := x_j$;
	LOOP x_k **DO** $x_\ell := x_\ell + 1$ **END**;
	$x_i := x_\ell$
	wobei x_ℓ nicht in der Eingabe
	und nicht sonst irgendwo im Programm vorkommen

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