9a

Konstruktionen von Turingmaschinen und LOOP-Programme

Prof. Dr. Jasmin Blanchette

Lehr- und Forschungseinheit für
Theoretische Informatik und Theorembeweisen

Stand: 18. Juni 2024
Basierend auf Folien von PD Dr. David Sabel

Wiederholung: Turingberechenbarkeit

Definition

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:

$$f(u) = v$$
g.d.w.
es gibt $z \in E$, sodass $Start_M(u) \vdash^* \square \cdots \square z v \square \cdots \square$

Wiederholung: Turingberechenbarkeit

Definition

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.

es gibt $z \in E$, sodass $z_0 bin(n_1) \# \cdots \# bin(n_k) \vdash^* \square \cdots \square z bin(m) \square \cdots \square$

wobei $bin(n)$ die Binärzahldarstellung von $n \in \mathbb{N}$ ist.

Wiederholung: Mehrband-Turingmaschinen

Definition

Eine k-Band-Turingmaschine (für $k \in \mathbb{N}_{>0}$) ist ein 7-Tupel $(Z, \Sigma, \Gamma, \delta, z_0, \square, E)$ mit

- Z ist eine endliche Menge von Zuständen
- Σ ist das (endliche) Eingabealphabet
- $\Gamma \supset \Sigma$ ist das (endliche) Bandalphabet
- δ ist die Überführungsfunktion
  - für DTM: $\delta : Z \times \Gamma^k \to Z \times \Gamma^k \times \{L, R, N\}^k$
  - für 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.

Konstruktion von Turingmaschinen und Notationen

DTMs können Programme einer einfachen imperativen Programmiersprache mit Zuweisungen, Verzweigungen und Schleifen simulieren.

Notationen:

- 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, schreiben wir $M(i)$ statt $M(i, k)$.

Konstruktion von Turingmaschinen

Beispiel:

- Die TM, die 1 addiert nennen wir „Band := Band + 1“.
- Die k-Band-TM, die 1 auf dem i-ten Band addiert nennen wir „Band := Band + 1“(i, k), „Band := Band + 1“(i) oder sogar „Band i := Band i + 1“.

Konstruktion von Turingmaschinen

Beispiel:

- Die TM, die 1 addiert nennen wir „Band := Band + 1“.
- Die k-Band-TM, die 1 auf dem i-ten Band addiert nennen wir „Band := Band + 1“(i, k), „Band := Band + 1“(i) oder sogar „Band i := Band i + 1“.

Weitere Turingmaschinen folgen. Die Konstruktionen sind relativ einfach.

- „Band i := (Band i) − 1“:
  k-Band-TM ($k \geq i$), die eine angepasste Subtraktion von 1 auf Band i durchführt. Beispiel für die 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 Turingmaschinen

Seien $M_i = (Z_i, \Sigma, \Gamma_i, \delta_i, z_{0i}, \square, E_i)$ für $i \in \{1, 2\}$ k-Band-TMs.
O.B.d.A. $Z_1 \cap Z_2 = \emptyset$.

Hintereinanderschaltung von Turingmaschinen

Seien $M_i = (Z_i, \Sigma, \Gamma_i, \delta_i, z_{0i}, \square, E_i)$ für $i \in \{1, 2\}$ k-Band-TMs.
O.B.d.A. $Z_1 \cap Z_2 = \emptyset$.

Die TM $M_1; M_2$ führt M_1 und M_2 hintereinandergeschaltet aus:
$M_1; M_2 = (Z_1 \cup Z_2, \Sigma, \Gamma_1 \cup \Gamma_2, \delta, z_{01}, \square, E_2)$ mit

$$\delta(z, (a_1, \ldots, a_k)) = \begin{cases} \delta_1(z, (a_1, \ldots, a_k)) & \text{falls } z \in Z_1 \setminus E_1 \\ (z_{02}, (a_1, \ldots, a_k), N^k) & \text{falls } z \in E_1 \\ \delta_2(z, (a_1, \ldots, a_k)) & \text{falls } z \in Z_2 \setminus E_2 \end{cases}$$

Hintereinanderschaltung von Turingmaschinen

Seien $M_i = (Z_i, \Sigma, \Gamma_i, \delta_i, z_{0i}, \square, E_i)$ für $i \in \{1, 2\}$ k-Band-TMs.
O.B.d.A. $Z_1 \cap Z_2 = \emptyset$.

Die TM $M_1; M_2$ führt M_1 und M_2 hintereinandergeschaltet aus:
$M_1; M_2 = (Z_1 \cup Z_2, \Sigma, \Gamma_1 \cup \Gamma_2, \delta, z_{01}, \square, E_2)$ mit

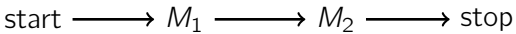
$$\delta(z, (a_1, \ldots, a_k)) = \begin{cases} \delta_1(z, (a_1, \ldots, a_k)) & \text{falls } z \in Z_1 \setminus E_1 \\ (z_{02}, (a_1, \ldots, a_k), N^k) & \text{falls } z \in E_1 \\ \delta_2(z, (a_1, \ldots, a_k)) & \text{falls } z \in Z_2 \setminus E_2 \end{cases}$$

Die TM $M_1; M_2$

- führt erst M_1 aus
- wechselt im Endzustand $z \in E_1$ in Startzustand z_{02} von M_2
- führt anschließend M_2 aus.

Hintereinanderschaltung von Turingmaschinen

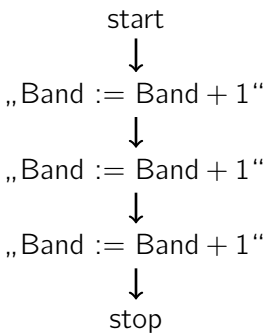
Flussdiagramm für $M_1; M_2$:



Hintereinanderschaltung von Turingmaschinen

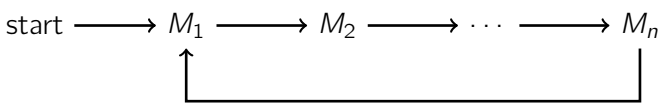
Beispiel: „Band := Band + 3“ wird konstruiert durch
„Band := Band + 1“;„Band := Band + 1“;„Band := Band + 1“

Flussdiagramm dazu:



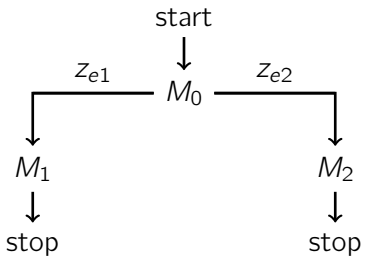
Hintereinanderschaltung von Turingmaschinen

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



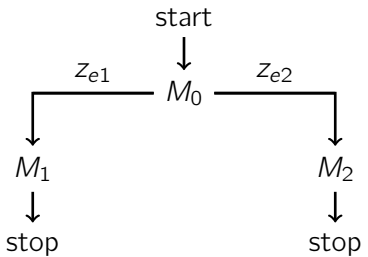
Verzweigende Fortsetzung

Seien M_0, M_1, M_2 TMs und seien z_{e1} und z_{e2} die Endzustände von M_0.
Verzweigende Fortsetzung von M_0:



Verzweigende Fortsetzung

Seien M_0, M_1, M_2 TMs und seien z_{e1} und z_{e2} die Endzustände von M_0.
Verzweigende Fortsetzung von M_0:



Die Konstruktion fügt Übergänge

$$\delta(z_{e1}, (a_1, \ldots, a_k)) = (z_{01}, (a_1, \ldots, a_k), N^k)$$
$$\delta(z_{e2}, (a_1, \ldots, a_k)) = (z_{02}, (a_1, \ldots, a_k), N^k)$$

ein, wobei z_{0i} der Startzustand von M_i ist (für $i \in \{1, 2\}$).

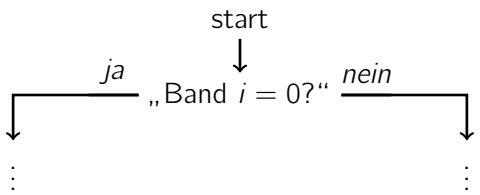
Beispiel für Test auf 0

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

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

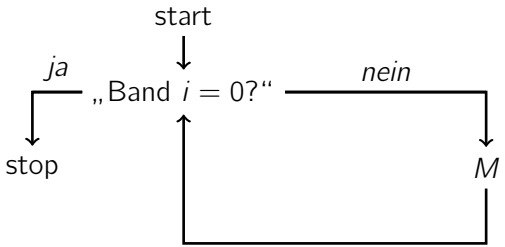
mit z_0 Startzustand und *ja* und *nein* Endzustände.

Notationen: „Band $= 0$?“ und „Band $i = 0$?“.



Schleife

Mit Verzweigung, „Band $i = 0$?“, zyklischer Hintereinanderschaltung und einer TM M erstellen wir die Schleife



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“.

LOOP-, WHILE-, GOTO-Programme

Ziel:

- Betrachte drei einfache imperative Programmiersprachen
  - LOOP-Programme
  - WHILE-Programme
  - GOTO-Programme

  und die dazugehörigen Berechenbarkeitsbegriffe.
- Welche Berechenbarkeitsbegriffe sind gleich bzw. verschieden (untereinander aber auch bezüglich Turingberechenbarkeit)?

Syntax von LOOP-Programmen

LOOP-Programme werden durch die kontextfreie Grammatik (V, Σ, P, Prg) erzeugt, wobei:

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

Beachte:

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

Semantik von LOOP-Programmen

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

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

Semantik von LOOP-Programmen

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 definieren auch

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

Semantik von LOOP-Programmen

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 definieren auch

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

Die Notation $\rho\{x_i \mapsto m\}$ steht also für die Variablenbelegung, die überall mit ρ übereinstimmt außer bei x_i, wofür m ausgegeben wird.

Semantik von LOOP-Programmen

Definition

Die Berechnungsschritte $(\rho, P) \xrightarrow[\text{LOOP}]{} (\rho', P')$, wobei ρ, ρ' Variablenbelegungen und P, P' LOOP-Programme oder das leere Programm ε sind, sind durch folgende Regeln definiert:

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

Semantik von LOOP-Programmen

Definition

Die Berechnungsschritte $(\rho, P) \xrightarrow[\text{LOOP}]{} (\rho', P')$, wobei ρ, ρ' Variablenbelegungen und P, P' LOOP-Programme oder das leere Programm ε sind, sind durch folgende Regeln definiert:

- $(\rho, x_i := x_j + c) \xrightarrow[\text{LOOP}]{} (\rho', \varepsilon)$, 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.

Beispiel für die Semantik von LOOP-Programmen

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

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

Beispiel für die Semantik von LOOP-Programmen

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

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

Ausführung:

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

Beispiel für die Semantik von LOOP-Programmen

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

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

Ausführung:

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

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

Beispiel für die Semantik von LOOP-Programmen

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

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

Ausführung:

$$
\begin{aligned}
& (\{x_1 \mapsto 2\}, x_2 := x_1 + 1; \textbf{LOOP } x_2 \textbf{ DO } x_3 := x_3 + 1 \textbf{ END}) \\
\xrightarrow[\text{LOOP}]{} & (\{x_1 \mapsto 2, x_2 \mapsto 3\}, \textbf{LOOP } x_2 \textbf{ DO } x_3 := x_3 + 1 \textbf{ END}) \\
& \text{da } (\{x_1 \mapsto 2\}, x_2 := x_1 + 1) \xrightarrow[\text{LOOP}]{} (\{x_1 \mapsto 2, x_2 \mapsto 3\}, \varepsilon) \\
\xrightarrow[\text{LOOP}]{} & (\{x_1 \mapsto 2, x_2 \mapsto 3\}, x_3 := x_3 + 1; x_3 := x_3 + 1; x_3 := x_3 + 1)
\end{aligned}
$$

Beispiel für die Semantik von LOOP-Programmen

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

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

Ausführung:

$$
\begin{aligned}
& (\{x_1 \mapsto 2\}, x_2 := x_1 + 1; \textbf{LOOP } x_2 \textbf{ DO } x_3 := x_3 + 1 \textbf{ END}) \\
\xrightarrow[\text{LOOP}]{} & (\{x_1 \mapsto 2, x_2 \mapsto 3\}, \textbf{LOOP } x_2 \textbf{ DO } x_3 := x_3 + 1 \textbf{ END}) \\
& \text{da } (\{x_1 \mapsto 2\}, x_2 := x_1 + 1) \xrightarrow[\text{LOOP}]{} (\{x_1 \mapsto 2, x_2 \mapsto 3\}, \varepsilon) \\
\xrightarrow[\text{LOOP}]{} & (\{x_1 \mapsto 2, x_2 \mapsto 3\}, x_3 := x_3 + 1; x_3 := x_3 + 1; x_3 := x_3 + 1) \\
\xrightarrow[\text{LOOP}]{} & (\{x_1 \mapsto 2, x_2 \mapsto 3, x_3 \mapsto 1\}, x_3 := x_3 + 1; x_3 := x_3 + 1) \\
& \text{da } (\{x_1 \mapsto 2, x_2 \mapsto 3\}, x_3 := x_3 + 1) \xrightarrow[\text{LOOP}]{} (\{x_1 \mapsto 2, x_2 \mapsto 3, x_3 \mapsto 1\}, \varepsilon)
\end{aligned}
$$

Beispiel für die Semantik von LOOP-Programmen

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

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

Ausführung:

$$
\begin{aligned}
& (\{x_1 \mapsto 2\}, x_2 := x_1 + 1; \textbf{LOOP } x_2 \textbf{ DO } x_3 := x_3 + 1 \textbf{ END}) \\
\xrightarrow[\text{LOOP}]{} & (\{x_1 \mapsto 2, x_2 \mapsto 3\}, \textbf{LOOP } x_2 \textbf{ DO } x_3 := x_3 + 1 \textbf{ END}) \\
& \text{da } (\{x_1 \mapsto 2\}, x_2 := x_1 + 1) \xrightarrow[\text{LOOP}]{} (\{x_1 \mapsto 2, x_2 \mapsto 3\}, \varepsilon) \\
\xrightarrow[\text{LOOP}]{} & (\{x_1 \mapsto 2, x_2 \mapsto 3\}, x_3 := x_3 + 1; x_3 := x_3 + 1; x_3 := x_3 + 1) \\
\xrightarrow[\text{LOOP}]{} & (\{x_1 \mapsto 2, x_2 \mapsto 3, x_3 \mapsto 1\}, x_3 := x_3 + 1; x_3 := x_3 + 1) \\
& \text{da } (\{x_1 \mapsto 2, x_2 \mapsto 3\}, x_3 := x_3 + 1) \xrightarrow[\text{LOOP}]{} (\{x_1 \mapsto 2, x_2 \mapsto 3, x_3 \mapsto 1\}, \varepsilon) \\
\xrightarrow[\text{LOOP}]{} & (\{x_1 \mapsto 2, x_2 \mapsto 3, x_3 \mapsto 2\}, x_3 := x_3 + 1) \\
& \text{da } (\{x_1 \mapsto 2, x_2 \mapsto 3, x_3 \mapsto 1\}, x_3 := x_3 + 1) \xrightarrow[\text{LOOP}]{} (\{x_1 \mapsto 2, x_2 \mapsto 3, x_3 \mapsto 2\}, \varepsilon)
\end{aligned}
$$

Beispiel für die Semantik von LOOP-Programmen

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

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

Ausführung:

$$
\begin{aligned}
& (\{x_1 \mapsto 2\}, x_2 := x_1 + 1; \textbf{LOOP } x_2 \textbf{ DO } x_3 := x_3 + 1 \textbf{ END}) \\
\xrightarrow[\text{LOOP}]{} & (\{x_1 \mapsto 2, x_2 \mapsto 3\}, \textbf{LOOP } x_2 \textbf{ DO } x_3 := x_3 + 1 \textbf{ END}) \\
& \text{da } (\{x_1 \mapsto 2\}, x_2 := x_1 + 1) \xrightarrow[\text{LOOP}]{} (\{x_1 \mapsto 2, x_2 \mapsto 3\}, \varepsilon) \\
\xrightarrow[\text{LOOP}]{} & (\{x_1 \mapsto 2, x_2 \mapsto 3\}, x_3 := x_3 + 1; x_3 := x_3 + 1; x_3 := x_3 + 1) \\
\xrightarrow[\text{LOOP}]{} & (\{x_1 \mapsto 2, x_2 \mapsto 3, x_3 \mapsto 1\}, x_3 := x_3 + 1; x_3 := x_3 + 1) \\
& \text{da } (\{x_1 \mapsto 2, x_2 \mapsto 3\}, x_3 := x_3 + 1) \xrightarrow[\text{LOOP}]{} (\{x_1 \mapsto 2, x_2 \mapsto 3, x_3 \mapsto 1\}, \varepsilon) \\
\xrightarrow[\text{LOOP}]{} & (\{x_1 \mapsto 2, x_2 \mapsto 3, x_3 \mapsto 2\}, x_3 := x_3 + 1) \\
& \text{da } (\{x_1 \mapsto 2, x_2 \mapsto 3, x_3 \mapsto 1\}, x_3 := x_3 + 1) \xrightarrow[\text{LOOP}]{} (\{x_1 \mapsto 2, x_2 \mapsto 3, x_3 \mapsto 2\}, \varepsilon) \\
\xrightarrow[\text{LOOP}]{} & (\{x_1 \mapsto 2, x_2 \mapsto 3, x_3 \mapsto 3\}, \varepsilon)
\end{aligned}
$$

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 $\rho'(x_0) = f(n_1, \ldots, n_k)$.

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

Beispiel für die LOOP-Berechenbarkeit

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

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

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

Terminierung von LOOP-Programmen

Satz

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

Terminierung von LOOP-Programmen

Satz

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

Beweis Zeige für alle (ρ, P): Es gibt $j \in \mathbb{N}$ und ρ', sodass $(\rho, P) \xrightarrow[\text{LOOP}]{}^{j} (\rho', \varepsilon)$. Durch Induktion über die Größe von P.

Terminierung von LOOP-Programmen

Satz

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

Beweis Zeige für alle (ρ, P): Es gibt $j \in \mathbb{N}$ und ρ', sodass $(\rho, P) \xrightarrow[\text{LOOP}]{}^{j} (\rho', \varepsilon)$. Durch Induktion über die Größe von P.

- Fall $(\rho, x_i := x_j \pm c)$: Es wird genau 1 Schritt benötigt.

Terminierung von LOOP-Programmen

Satz

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

Beweis Zeige für alle (ρ, P): Es gibt $j \in \mathbb{N}$ und ρ', sodass $(\rho, P) \xrightarrow[\text{LOOP}]{}^{j} (\rho', \varepsilon)$.
Durch Induktion über die Größe von P.

- Fall $(\rho, x_i := x_j \pm c)$: Es wird genau 1 Schritt benötigt.
- Fall $P_1; P_2$: Die Induktionshypothese liefert 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.

Terminierung von LOOP-Programmen

Satz

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

Beweis Zeige für alle (ρ, P): Es gibt $j \in \mathbb{N}$ und ρ', sodass $(\rho, P) \xrightarrow[\text{LOOP}]{}^{j} (\rho', \varepsilon)$.
Durch Induktion über die Größe von P.

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

LOOP-Berechenbarkeit

- Da es partielle turingberechenbare Funktionen gibt, gilt:
  Es gibt turingberechenbare Funktionen, die nicht LOOP-berechenbar sind.

  Ein Beispiel ist 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 (später heute).

Kodierung weiterer Befehle mit LOOP-Programmen

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
(und daher $\rho(x_n) = 0$)

Kodierung weiterer Befehle mit LOOP-Programmen

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
(und daher $\rho(x_n) = 0$)

Befehl: $x_i := x_j$

Kodierung: $x_i := x_j + 0$

Kodierung weiterer Befehle mit LOOP-Programmen

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
(und daher $\rho(x_n) = 0$)

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

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

Kodierung weiterer Befehle mit LOOP-Programmen

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

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 vorkommt

Kodierung weiterer Befehle mit LOOP-Programmen

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 vorkommt

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.