

# Konstruktionen von Turingmaschinen und LOOP-Programme

Prof. Dr. David Sabel

LFE Theoretische Informatik



## Definition (Turingberechenbarkeit)

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

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

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

g.d.w.

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

Eine Funktion  $f : \Sigma^* \rightarrow \Sigma^*$  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

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

### Definition (Mehrband-Turingmaschine)

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

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

## Konstruktion von 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  $M(i, k)$

## 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  $i := \text{Band } i - 1$ “:  
 $k$ -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  $i := \text{Band } j$ “:  
 $k$ -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

$$\delta(z, (a_1, \dots, a_k)) = \begin{cases} \delta_1(z, (a_1, \dots, a_k)) & \text{falls } z \in Z_1 \setminus E_1, (a_1, \dots, a_k) \in \Gamma_1^k \\ \delta_2(z, (a_1, \dots, a_k)) & \text{falls } z \in Z_2, (a_1, \dots, a_k) \in \Gamma_2^k \\ (z_2, (a_1, \dots, a_k), N^k) & \text{falls } z \in E_1, (a_1, \dots, a_k) \in \Gamma_1^k \end{cases}$$

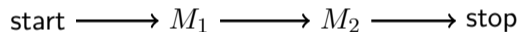
Die TM  $M_1; M_2$

- 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)

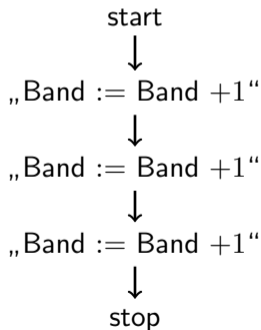
---

Beispiel:

„Band := Band+3“ wird konstruiert durch

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

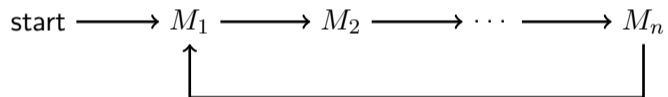
Flussdiagramm dazu



## Hintereinanderschaltung von TMs (4)

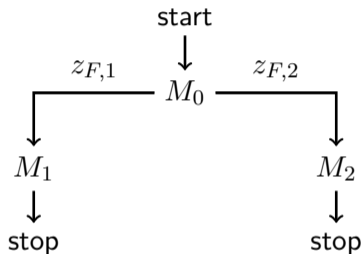
---

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



# 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

$$\delta(z_{F,1}, (a_1, \dots, a_k)) = (z_{0,M_1}, (a_1, \dots, a_k), N) \text{ und}$$

$$\delta(z_{F,2}, (a_1, \dots, a_k)) = (z_{0,M_2}, (a_1, \dots, a_k), N)$$

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

$$\delta(z_0, a) = (nein, a, N) \quad \text{für } a \neq 0$$

$$\delta(z_0, 0) = (z_1, 0, R)$$

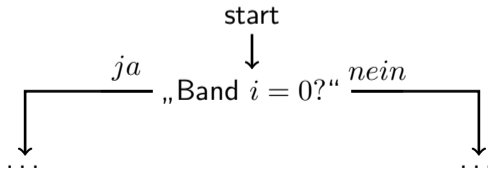
$$\delta(z_1, a) = (nein, a, L) \quad \text{für } a \neq \square$$

$$\delta(z_1, \square) = (ja, \square, L)$$

mit  $ja$  und  $nein$  Endzustände und  $z_0$  Startzustand

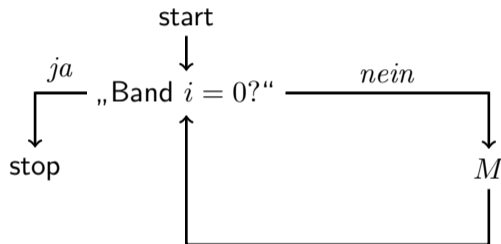
TM  $M_0$  prüft, ob das Band eine 0 enthält oder nicht.

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  $(V, \Sigma, P, Prg)$  erzeugt, wobei:

$$\begin{aligned} V &= \{Prg, Var, Id, Const\} \\ \Sigma &= \{\mathbf{LOOP}, \mathbf{DO}, \mathbf{END}, x, 0, \dots, 9, ;, :=, +, -\} \\ P &= \{Prg \rightarrow \mathbf{LOOP} Var \mathbf{DO} Prg \mathbf{END} \\ &\quad | Prg; Prg \\ &\quad | Var := Var + Const \\ &\quad | Var := Var - Const \\ \\ Var &\rightarrow x Id \\ Const &\rightarrow Id \\ Id &\rightarrow 0 | 1 | \dots | 9 | 1Const | 2Const | \dots | 9Const\} \end{aligned}$$

Beachte:

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



## Definition (Variablenbelegung)

Eine **Variablenbelegung**  $\rho$  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  $\rho, \rho'$  Variablenbelegungen und  $P, P'$  LOOP-Programme oder  $\varepsilon$  (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  $(\rho, P_1) \xrightarrow{\text{LOOP}} (\rho', \varepsilon)$
- $(\rho, \text{LOOP } x_i \text{ DO } P \text{ END}) \xrightarrow{\text{LOOP}} (\rho, \underbrace{P; \dots; P}_{\rho(x_i)\text{-mal}})$

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:  $x_2 := x_1 + 1;$

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

**LOOP**  $x_2$  **DO**  $x_3 := x_3 + 1$  **END**

Ausführung:

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

# LOOP-Programme: Beispiel für die Semantik

Programm:  $x_2 := x_1 + 1;$

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

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

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

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

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

# LOOP-Programme: Beispiel für die Semantik

Programm:  $x_2 := x_1 + 1;$

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

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

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

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

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

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



## Definition (LOOP-berechenbare Funktion)

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

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

D.h. das LOOP-Programm

- empfängt Eingaben über die Variablen  $x_1, \dots, x_k$
- 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  $n_1 \in \mathbb{N}$ :

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

## Satz

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

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

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  $\rho$  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  $j_{P,\rho_i}$  und  $\rho_i$  mit  $(\rho_1, \mathbf{LOOP} \ x_i \ \mathbf{DO} \ P \ \mathbf{END}) \xrightarrow[\text{LOOP}]{} (\rho_1, P; \dots; P) \xrightarrow[\text{LOOP}]{}^{j_{P,\rho_1}} (\rho_2, P; \dots; P) \xrightarrow[\text{LOOP}]{}^{j_{P,\rho_2}} \dots \xrightarrow[\text{LOOP}]{}^{j_{P,\rho_n}} (\rho_{n+1}, \varepsilon)$  mit  $n = \rho_1(x_i)$

- 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  $\rho(x_n) = 0$ )

## 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:  $x_i := x_j + 0$

## 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