

# WHILE- und GOTO-Programme

Prof. Dr. Jasmin Blanchette

Lehr- und Forschungseinheit für  
Theoretische Informatik

Stand: 20. Juni 2023

Folien ursprünglich von PD Dr. David Sabel



# WHILE-Programme: Syntax

WHILE-Programme werden durch die CFG  $(V, \Sigma, P, Prg)$  erzeugt:

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

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  $\rho, \rho'$  Variablenbelegungen und  $P, P'$  WHILE-Programme oder  $\varepsilon$  (leeres Programm)

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

## Definition (WHILE-berechenbare Funktion)

Eine Funktion  $f : \mathbb{N}^k \rightarrow \mathbb{N}$  heißt **WHILE-berechenbar**, wenn es ein WHILE-Programm  $P$  gibt, das die folgenden Bedingungen erfüllt:

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

# LOOP-berechenbar $\implies$ WHILE-berechenbar

---

## Satz

Jede LOOP-berechenbare Funktion ist auch WHILE-berechenbar.

# LOOP-berechenbar $\implies$ WHILE-berechenbar

---

## Satz

Jede LOOP-berechenbare Funktion ist auch WHILE-berechenbar.

Beweis:

Offensichtlich, da jedes LOOP-Programm auch ein WHILE-Programm mit derselben Semantik 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 $\implies$ Turingberechenbar

---

## **Theorem 10.2.4**

Jede WHILE-berechenbare Funktion ist auch Turingberechenbar.

# WHILE-berechenbar $\implies$ Turingberechenbar

---

## Theorem 10.2.4

Jede WHILE-berechenbare Funktion ist auch Turingberechenbar.

Beweis:

- ▶ Wir haben bereits gezeigt, dass Zuweisung, Hintereinanderschaltung 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  $(V, \Sigma, P, Prg)$  erzeugt:

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

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

# GOTO-Programme: Semantik (Berechnungsschritte)

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

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

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

# GOTO-Programme: Semantik (Berechnungsschritte, Fortsetzung)

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

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  $P_0$ ).

## Definition (GOTO-berechenbare Funktion)

Eine Funktion  $f : \mathbb{N}^k \rightarrow \mathbb{N}$  heißt **GOTO-berechenbar**, wenn es ein normalisiertes GOTO-Programm  $P$  gibt, das die folgenden Bedingungen erfüllt:

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

# WHILE-berechenbar $\implies$ GOTO-berechenbar (1)

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**  $P_W$  **END** durch

```
M1:    IF  $x_i = 0$  THEN GOTO  $M_j$ ;  
...     $P_G$ ;  
 $M_{j-1}$ : GOTO  $M_1$ ;  
 $M_j$ :    ...
```

simulieren, wobei  $P_G$  das GOTO-Programm zu  $P_W$  ist

- ▶ **HALT** am Ende einfügen

## WHILE-berechenbar $\implies$ GOTO-berechenbar (2)

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

Dann gilt:

- ▶ Falls  $(\rho, P_W) \xrightarrow[\text{WHILE}]{}^* (\rho', \varepsilon)$ ,  
dann  $(\rho, P_G) \xrightarrow[\text{GOTO}]{}^* (\rho'', \mathbf{HALT})$   
mit  $\rho'(x_0) = \rho''(x_0)$ ,  
und umgekehrt.

Damit folgt:

Jede WHILE-berechenbare Funktion ist auch GOTO-berechenbar.

## GOTO-berechenbar $\implies$ WHILE-berechenbar (1)

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

```
 $x_M := 1;$   
WHILE  $x_M \neq 0$  DO  
  IF  $x_M = 1$  THEN  $P_1$  END;  
  ...  
  IF  $x_M = n$  THEN  $P_n$  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  $A_i = \mathbf{GOTO} M_j$
- ▶ **IF**  $x_k = 0$  **THEN**  $x_M := j$  **ELSE**  $x_M := x_M + 1$  **END**,  
falls  $A_i = \mathbf{IF} x_k = 0 \mathbf{THEN GOTO} M_j$
- ▶  $x_M := 0$ , falls  $A_i = \mathbf{HALT}$

## GOTO-berechenbar $\implies$ WHILE-berechenbar (2)

Es gilt:

- ▶ Falls  $(\rho, M_1: A_1; \dots; M_n: A_n) \xrightarrow[\text{GOTO}]{M_1: A_1; \dots; M_n: A_n}^* (\rho', \mathbf{HALT})$ ,  
dann  $(\rho, P) \xrightarrow[\text{WHILE}]{}^* (\rho'', \varepsilon)$   
mit  $\rho'(x_0) = \rho''(x_0)$ ,  
und umgekehrt.

Damit folgt:

Jede GOTO-berechenbare Funktion ist auch 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.

## 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 Übersetzung von GOTO-Programmen in WHILE-Programme zeigen auch:

## Satz

WHILE-Programme benötigen nur eine WHILE-Schleife.

# Turingberechenbar $\implies$ GOTO-berechenbar (1)

---

Ziel:

- ▶ GOTO-Programm simuliert die (deterministische) Turingmaschine.

Idee:

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

# Darstellung von Wörtern als natürliche Zahlen

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

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

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

- ▶ Teile wiederholt mit Rest durch Basis  $b$ .
- ▶ Liste der Reste:  $r_0, \dots, r_m$ .
- ▶ Ergibt  $(r_m \cdots r_0)$  zur Basis  $b$  und repräsentiert 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

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

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 $\implies$ GOTO-berechenbar (2)

### TM-Konfiguration im GOTO-Programm:

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

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

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

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

Beachte:  $x_s$  verwendet umgekehrte Reihenfolge der Ziffern.

## Turingberechenbar $\implies$ GOTO-berechenbar (3)

---

### GOTO-Programme für Übergänge:

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

## Turingberechenbar $\implies$ GOTO-berechenbar (4)

**Fall**  $\delta(z_k, a_{j_1}) = (z_\ell, a_{j'}, N)$

Übergang ist

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

- ▶  $x_z$  muss auf den Wert  $\ell$  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  $DELTA(k, j_1)$  sei dann

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

## Turingberechenbar $\implies$ GOTO-berechenbar (5)

**Fall**  $\delta(z_k, a_{j_1}) = (z_\ell, a_{j'}, R)$

Übergang ist

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

falls  $m > 1$ .

- ▶  $x_z$  muss auf den Wert  $\ell$  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  $DELTA(k, j_1)$  sei dann

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

## Turingberechenbar $\implies$ GOTO-berechenbar (6)

Für den Fall  $m = 1$  gilt

$$a_{i_1} \cdots a_{i_n} z_k a_{j_1} \vdash a_{i_1} \cdots a_{i_n} a_{j'} z_\ell \square$$

In diesem Fall sei  $DELTA(k, j_1)$  das Programm

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

wobei  $\square = a_r$ .

## Turingberechenbar $\implies$ GOTO-berechenbar (7)

**Fall**  $\delta(z_k, a_{j_1}) = (z_\ell, a_{j'}, L)$

Übergang ist

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

falls  $n > 0$ .

- ▶  $x_z$  muss auf den Wert  $\ell$  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  $DELTA(k, j_1)$  dazu:

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

## Turingberechenbar $\implies$ GOTO-berechenbar (8)

---

Für den Grenzfall

$$z_k a_{j_1} \cdots a_{j_m} \vdash z_\ell \square a_{j_1} \cdots a_{j_m}$$

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

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

## Turingberechenbar $\implies$ GOTO-berechenbar (9)

---

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

$M_1 : P_1;$

$M_2 : P_2;$

$M_3 : P_3$

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

## Turingberechenbar $\implies$ GOTO-berechenbar (10)

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  $DELTA(i,j)$  sonst.

# Turingberechenbar $\implies$ GOTO-berechenbar (11)

---

Insgesamt zeigt das GOTO-Programm:

## **Satz 10.5.2**

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

## Theorem

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

