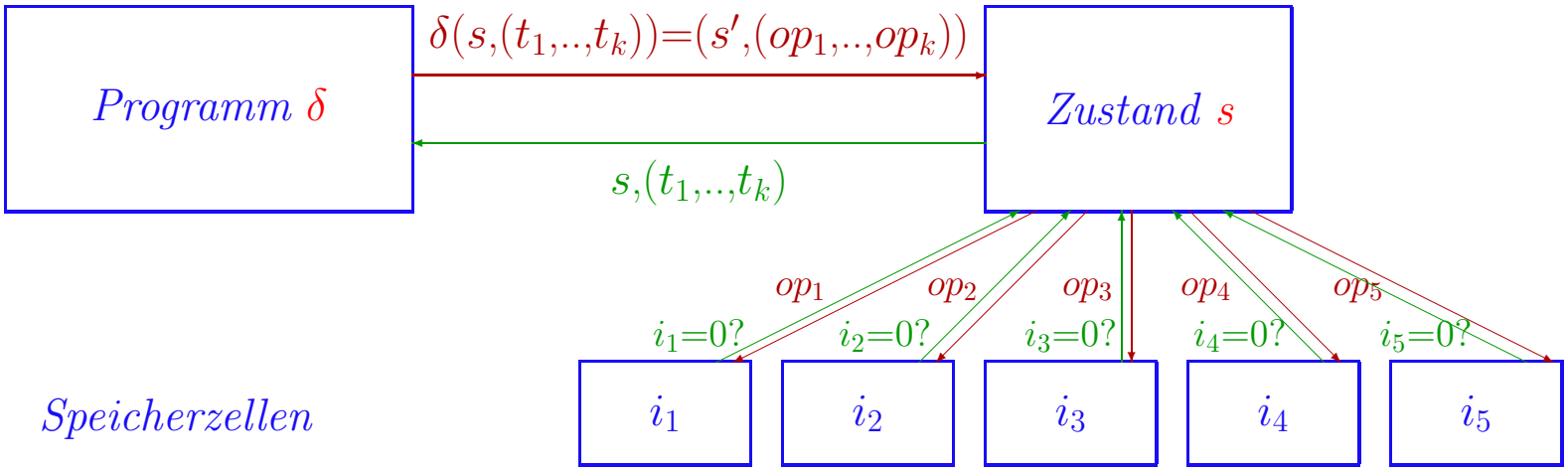
Theoretische Informatik II

Einheit 6.2

Registermaschinen

1. Arbeitsweise
2. Formale Semantik
3. Register-Berechenbarkeit
4. Programmiermethoden
5. Äquivalenz zu Turingmaschinen

Registermaschinen

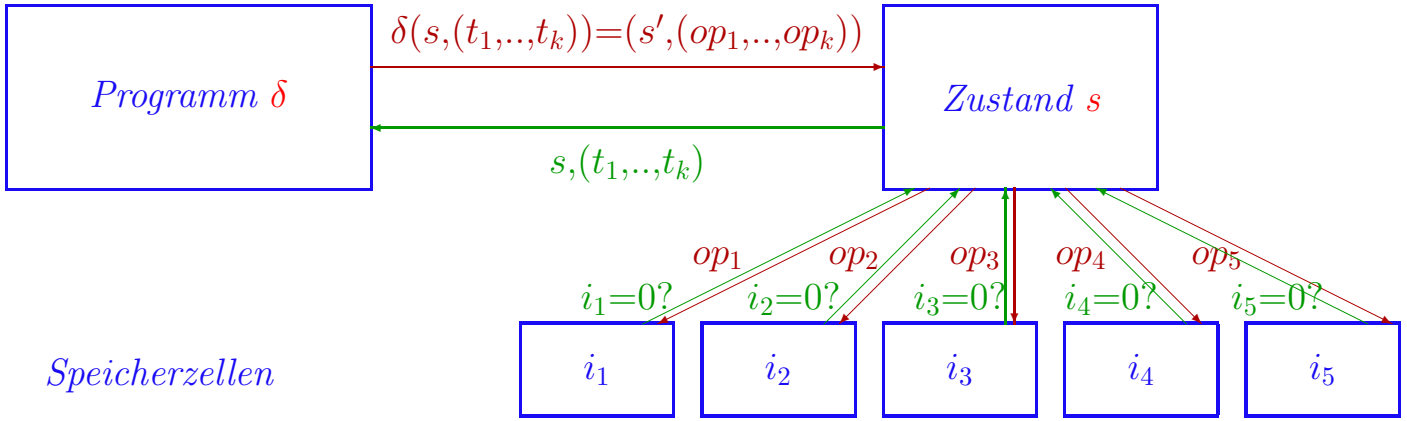


- **Standardarchitektur von Einprozessorsystemen**
  - Direkter und simultaner Speicherzugriff
  - Speicherzellen enthalten natürliche Zahlen
  - Keine Ein/Ausgabe, sehr einfacher Befehlssatz
- **Unterschiede zur Turingmaschine**
  - Endlicher Speicher, aber unendlicher Bereich für Werte von Zellen

Achtung! Modelle in Literatur oft flexibler

Registermaschinen – mathematisch

Definition F



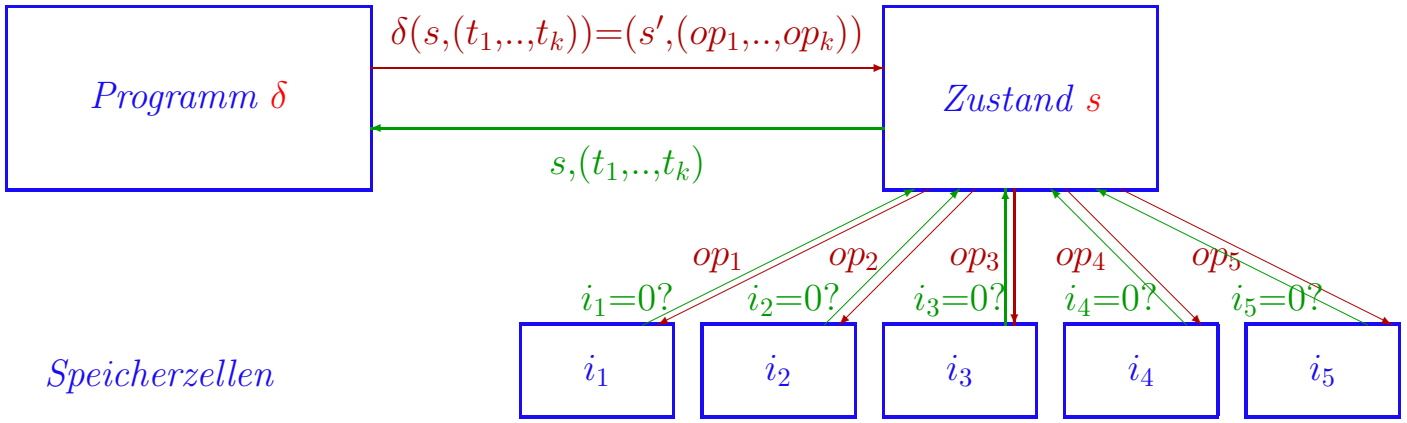
Eine **Registermaschine** ist ein 5-Tupel $\boldsymbol{\rho} = (S,\, k,\, \delta,\, s_0,\, F)$

- S nichtleere endliche Zustandsmenge
- $s_0 \in S$ Anfangszustand
- $k \in \mathbb{N}$ Anzahl der Register
- $F \subseteq S$ Menge der Endzustände
- $\delta : (S \setminus F) \times \{0,1\}^k \to S \times \{-1,0,1\}^k$ Zustandsüberführungsfunktion

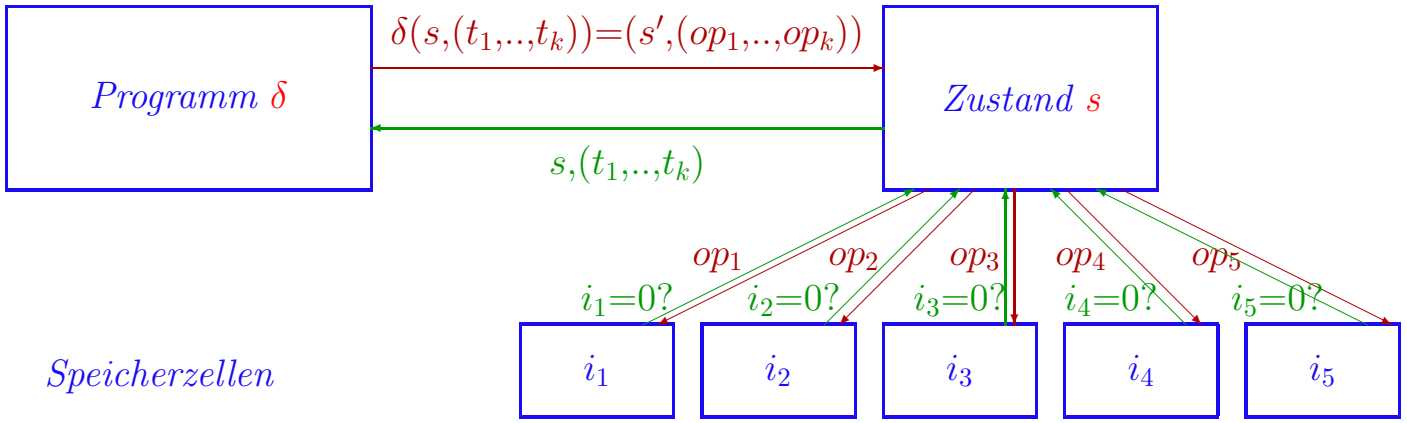
  Eingabe: Zustand + Testergebnisse: $t_j = \mathrm{sign}(i_j) = \begin{cases} 0 & \text{falls } i_j{=}0, \\ 1 & \text{falls } i_j{>}0 \end{cases}$

  Ausgabe: Zustand + Registeroperationen: op_j (Subtraktion, Identität, Addition)

Arbeitsweise von Registermaschinen

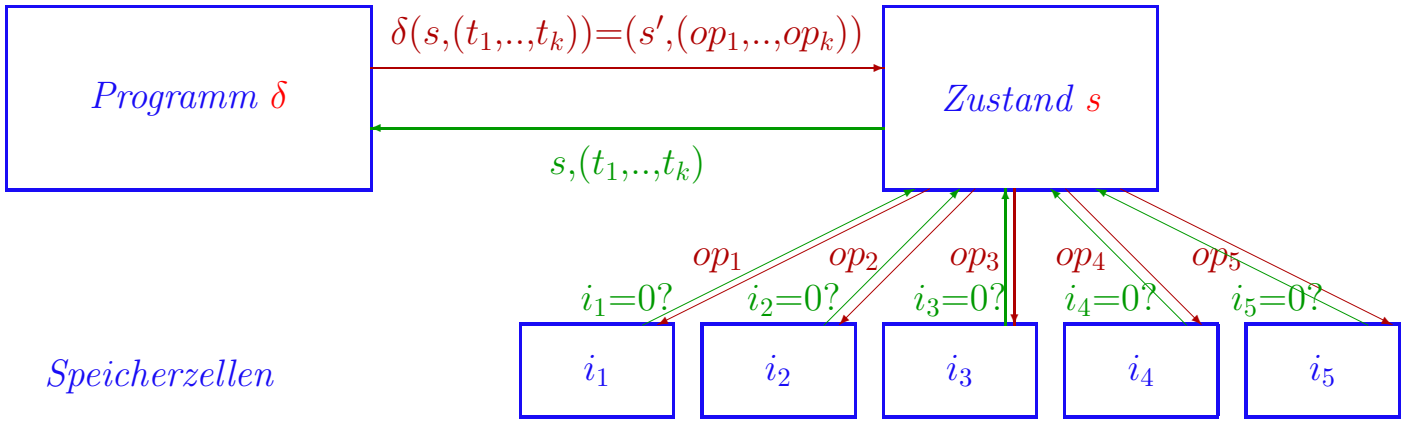


Arbeitsweise von Registermaschinen



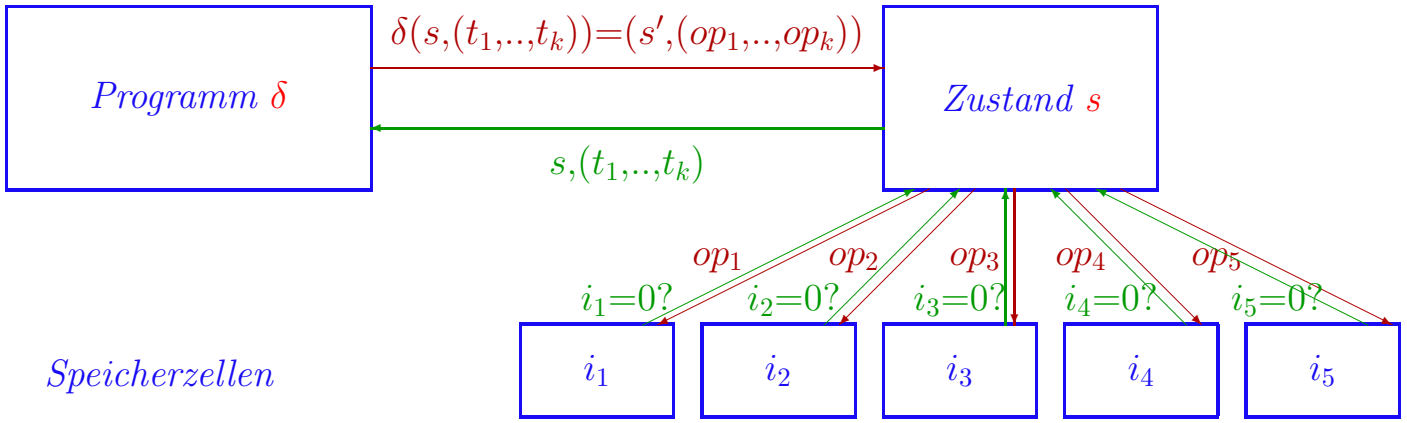
- **Anfangssituation**
  - Eingabezahl n steht im ersten Register

Arbeitsweise von Registermaschinen



- **Anfangssituation**
  - Eingabezahl n steht im ersten Register
- **Arbeitschritt**
  - Inhalte der Register $i_1,..,i_k$ lesen und mit sign(i_j) auf Null testen
  - Zustand s und Testergebnisse $t_1,..,t_k$ als Argumente an δ geben
  - $\delta(s,(t_1,..,t_k))=(s',(op_1,..,op_k))$ bestimmen
  - Neuer Zustand s', Register j gemäß Operation op_j modifizieren
  - Stop wenn s' Endzustand ist

Arbeitsweise von Registermaschinen



- **Anfangssituation**
  - Eingabezahl n steht im ersten Register
- **Arbeitschritt**
  - Inhalte der Register $i_1,..,i_k$ lesen und mit sign(i_j) auf Null testen
  - Zustand s und Testergebnisse $t_1,..,t_k$ als Argumente an δ geben
  - $\delta(s,(t_1,..,t_k))=(s',(op_1,..,op_k))$ bestimmen
  - Neuer Zustand s', Register j gemäß Operation op_j modifizieren
  - Stop wenn s' Endzustand ist
- **Ergebnis**
  - Inhalt des ersten Registers

Semantik von Registermaschinen

Definition G

- $\boldsymbol{K_\rho}$ $\equiv$ **Menge aller Konfigurationen** $\boldsymbol{\kappa} = (s,(i_1,..,i_k))$ **von** ρ **mit**
  - $s \in S$ aktueller Zustand
  - $i_j \in \mathbb{N}$ Inhalt des Registers j

Semantik von Registermaschinen

Definition G

- $\boldsymbol{K_\rho} \equiv$ **Menge aller Konfigurationen** $\boldsymbol{\kappa} = (s,(i_1,..,i_k))$ **von** ρ **mit**
  - $s \in S$ aktueller Zustand
  - $i_j \in \mathbb{N}$ Inhalt des Registers j
- **Anfangskonfiguration** $\boldsymbol{\alpha}$**:**$\mathbb{N} \to K_\rho$**:** $\alpha(n) = (s_0,(n,0,...,0))$

Semantik von Registermaschinen

Definition G

- $\boldsymbol{K_\rho}$ $\equiv$ **Menge aller Konfigurationen** $\boldsymbol{\kappa} = (s,(i_1,..,i_k))$ **von** ρ **mit**
  - $s \in S$ aktueller Zustand
  - $i_j \in \mathbb{N}$ Inhalt des Registers j
- **Anfangskonfiguration** $\boldsymbol{\alpha}$**:**$\mathbb{N} \to K_\rho$**:** $\alpha(n) = (s_0,(n,0,...,0))$
- **Nachfolgekonfiguration:** $\hat{\boldsymbol{\delta}}$**:**$K_\rho \to K_\rho$
  - Für $\kappa=(s,(i_1,..,i_k))$ mit $\delta(s,(\text{sign}(i_1),..,\text{sign}(i_k))) = (s',(op_1,..,op_k))$ ist

$$\hat{\delta}(\kappa)=(s',(i'_1,..,i'_k)), \text{ wobei } i'_j = \begin{cases} 0 & \text{falls } i_j=\texttt{0} \text{ und } op_j=\texttt{-1}, \\ i_j+op_j & \text{sonst} \end{cases}$$

Semantik von Registermaschinen

Definition G

- $\boldsymbol{K_\rho}$ $\equiv$ **Menge aller Konfigurationen** $\boldsymbol{\kappa} = (s,(i_1,..,i_k))$ **von** ρ **mit**
  - $s \in S$ aktueller Zustand
  - $i_j \in \mathbb{N}$ Inhalt des Registers j
- **Anfangskonfiguration** $\boldsymbol{\alpha}$**:** $\mathbb{N} \to K_\rho$**:** $\alpha(n) = (s_0,(n,0,...,0))$
- **Nachfolgekonfiguration:** $\hat{\boldsymbol{\delta}}$**:** $K_\rho \to K_\rho$
  - Für $\kappa=(s,(i_1,..,i_k))$ mit $\delta(s,(\text{sign}(i_1),..,\text{sign}(i_k))) = (s',(op_1,..,op_k))$ ist

$$\hat{\delta}(\kappa)=(s',(i'_1,..,i'_k)), \text{ wobei } i'_j = \begin{cases} 0 & \text{falls } i_j=\texttt{0} \text{ und } op_j=\texttt{-1}, \\ i_j+op_j & \text{sonst} \end{cases}$$

- **Ausgabefunktion** $\boldsymbol{\omega}$**:** $K_\rho \to \mathbb{N}$**:** $\omega(s,(i_1,..,i_k)) = i_1$

Semantik von Registermaschinen

Definition G

- **$K_\rho \equiv$ Menge aller Konfigurationen $\kappa = (s,(i_1,..,i_k))$ von ρ mit**
  - $s \in S$ aktueller Zustand
  - $i_j \in \mathbb{N}$ Inhalt des Registers j
- **Anfangskonfiguration α:$\mathbb{N} \to K_\rho$:** $\alpha(n) = (s_0,(n,0,...,0))$
- **Nachfolgekonfiguration: $\hat{\delta}$:$K_\rho \to K_\rho$**
  - Für $\kappa=(s,(i_1,..,i_k))$ mit $\delta(s,(\text{sign}(i_1),..,\text{sign}(i_k))) = (s',(op_1,..,op_k))$ ist

$$\hat{\delta}(\kappa)=(s',(i'_1,..,i'_k)), \text{ wobei } i'_j = \begin{cases} 0 & \text{falls } i_j=\texttt{0} \text{ und } op_j=\texttt{-1}, \\ i_j+op_j & \text{sonst} \end{cases}$$

- **Ausgabefunktion ω:$K_\rho \to \mathbb{N}$:** $\omega(s,(i_1,..,i_k)) = i_1$
- **Die von ρ berechnete Funktion h_ρ:$\mathbb{N} \to \mathbb{N}$ ist definiert durch**

$$h_\rho(n) = \begin{cases} \omega(\hat{\delta}^m(\alpha(n))) & \text{falls } m=\min\{j \mid \exists s \in F.\ \hat{\delta}^j(\alpha(n))=(s,\_)\} \text{ existiert} \\ \perp & \text{sonst} \end{cases}$$

Semantik von Registermaschinen

Definition G

- $K_\rho \equiv$ **Menge aller Konfigurationen** $\kappa = (s,(i_1,..,i_k))$ **von** ρ **mit**
  - $s \in S$ aktueller Zustand
  - $i_j \in \mathbb{N}$ Inhalt des Registers j
- **Anfangskonfiguration** α:$\mathbb{N} \to K_\rho$: $\alpha(n) = (s_0,(n,0,...,0))$
- **Nachfolgekonfiguration:** $\hat{\delta}$:$K_\rho \to K_\rho$
  - Für $\kappa=(s,(i_1,..,i_k))$ mit $\delta(s,(\text{sign}(i_1),..,\text{sign}(i_k))) = (s',(op_1,..,op_k))$ ist

$$\hat{\delta}(\kappa)=(s',(i'_1,..,i'_k)), \text{ wobei } i'_j = \begin{cases} 0 & \text{falls } i_j=\texttt{0} \text{ und } op_j=\texttt{-1}, \\ i_j+op_j & \text{sonst} \end{cases}$$

- **Ausgabefunktion** ω:$K_\rho \to \mathbb{N}$: $\omega(s,(i_1,..,i_k)) = i_1$
- **Die von** ρ **berechnete Funktion** h_ρ:$\mathbb{N} \to \mathbb{N}$ **ist definiert durch**

$$h_\rho(n) = \begin{cases} \omega(\hat{\delta}^m(\alpha(n))) & \text{falls } m=\min\{j \mid \exists s \in F.\ \hat{\delta}^j(\alpha(n))=(s,\_)\} \text{ existiert} \\ \perp & \text{sonst} \end{cases}$$

Definition H

Definitionsbereich von ρ ist $\{n \in \mathbb{N} \mid h_\rho(n) \neq \perp\}$,
Wertebereich von ρ ist $\{m \in \mathbb{N} \mid \exists n \in \mathbb{N}\ h_\rho(n)=m\}$

Beispiele für Registermaschinen

- $\boldsymbol{\rho_1} = (\{\mathtt{s}_0, \mathtt{s}_1\}, \mathtt{1}, \delta_1, \mathtt{s}_0, \{\mathtt{s}_1\})$ mit $\delta_1 =$

s	t_1	s'	op_1
$\mathtt{s}_0$	0	$\mathtt{s}_1$	+1
$\mathtt{s}_0$	1	$\mathtt{s}_0$	-1

Beispiele für Registermaschinen

- $\boldsymbol{\rho_1} = (\{\mathtt{s}_0, \mathtt{s}_1\}, \mathtt{1}, \delta_1, \mathtt{s}_0, \{\mathtt{s}_1\})$ mit $\delta_1 =$

s	t_1	s'	op_1
$\mathtt{s}_0$	0	$\mathtt{s}_1$	+1
$\mathtt{s}_0$	1	$\mathtt{s}_0$	-1

Zähle Eingabewert n auf Null herunter und addiere Eins

Beispiele für Registermaschinen

- $\boldsymbol{\rho_1} = (\{\mathtt{s}_0, \mathtt{s}_1\}, \mathtt{1}, \delta_1, \mathtt{s}_0, \{\mathtt{s}_1\})$ mit $\delta_1 =$

s	t_1	s'	op_1
$\mathtt{s}_0$	0	$\mathtt{s}_1$	+1
$\mathtt{s}_0$	1	$\mathtt{s}_0$	-1

  Zähle Eingabewert n auf Null herunter und addiere Eins

- **Mathematische Analyse**:

Beispiele für Registermaschinen

- $\boldsymbol{\rho_1} = (\{\mathtt{s}_0, \mathtt{s}_1\}, \mathtt{1}, \delta_1, \mathtt{s}_0, \{\mathtt{s}_1\})$ mit $\delta_1 =$

s	t_1	s'	op_1
$\mathtt{s}_0$	0	$\mathtt{s}_1$	+1
$\mathtt{s}_0$	1	$\mathtt{s}_0$	-1

  Zähle Eingabewert n auf Null herunter und addiere Eins

- **Mathematische Analyse**:
  - Anfangskonfiguration: $\alpha(n) = (\mathtt{s}_0, n)$

Beispiele für Registermaschinen

- $\boldsymbol{\rho_1} = (\{\mathtt{s}_0, \mathtt{s}_1\}, \mathtt{1}, \delta_1, \mathtt{s}_0, \{\mathtt{s}_1\})$ mit $\delta_1 =$

s	t_1	s'	op_1
$\mathtt{s}_0$	0	$\mathtt{s}_1$	+1
$\mathtt{s}_0$	1	$\mathtt{s}_0$	-1

  Zähle Eingabewert n auf Null herunter und addiere Eins

- **Mathematische Analyse**:
  - Anfangskonfiguration: $\alpha(n) = (\mathtt{s}_0, n)$
  - Nachfolgekonfigurationen: $\hat{\delta}(\mathtt{s}_0, n) = \begin{cases} (\mathtt{s}_0, n-1) & \text{falls } n>0, \\ (\mathtt{s}_1, \mathtt{1}) & \text{falls } n=0 \end{cases}$

Beispiele für Registermaschinen

- $\boldsymbol{\rho_1} = (\{\mathtt{s}_0, \mathtt{s}_1\},\ \mathtt{1},\ \delta_1,\ \mathtt{s}_0,\ \{\mathtt{s}_1\})$ mit $\delta_1 =$

s	t_1	s'	op_1
$\mathtt{s}_0$	$\mathtt{0}$	$\mathtt{s}_1$	$\mathtt{+1}$
$\mathtt{s}_0$	$\mathtt{1}$	$\mathtt{s}_0$	$\mathtt{-1}$

  Zähle Eingabewert n auf Null herunter und addiere Eins

- **Mathematische Analyse**:
  - Anfangskonfiguration: $\alpha(n) = (\mathtt{s}_0, n)$
  - Nachfolgekonfigurationen: $\hat{\delta}(\mathtt{s}_0, n) = \begin{cases} (\mathtt{s}_0, n-1) & \text{falls } n>0, \\ (\mathtt{s}_1, \mathtt{1}) & \text{falls } n=0 \end{cases}$
  - Terminierung: $\min\{j \mid \hat{\delta}^j(\mathtt{s}_0, n) = (\mathtt{s}_1, \_)\} = n+1$

Beispiele für Registermaschinen

- $\boldsymbol{\rho_1} = (\{\mathtt{s}_0, \mathtt{s}_1\},\ \mathtt{1},\ \delta_1,\ \mathtt{s}_0,\ \{\mathtt{s}_1\})$ mit $\delta_1 =$

s	t_1	s'	op_1
$\mathtt{s}_0$	0	$\mathtt{s}_1$	+1
$\mathtt{s}_0$	1	$\mathtt{s}_0$	-1

  Zähle Eingabewert n auf Null herunter und addiere Eins

- **Mathematische Analyse**:
  - Anfangskonfiguration: $\alpha(n) = (\mathtt{s}_0, n)$
  - Nachfolgekonfigurationen: $\hat{\delta}(\mathtt{s}_0, n) = \begin{cases} (\mathtt{s}_0, n-1) & \text{falls } n>0, \\ (\mathtt{s}_1, \mathtt{1}) & \text{falls } n=0 \end{cases}$
  - Terminierung: $\min\{j \mid \hat{\delta}^j(\mathtt{s}_0, n) = (\mathtt{s}_1, \_)\} = n+1$
  - Ergebnis: $\hat{\delta}^{n+1}(\mathtt{s}_0, n) = (\mathtt{s}_1, \mathtt{1})$

Beispiele für Registermaschinen

- $\boldsymbol{\rho_1} = (\{\mathtt{s}_0, \mathtt{s}_1\},\ \mathtt{1},\ \delta_1,\ \mathtt{s}_0,\ \{\mathtt{s}_1\})$ mit $\delta_1 =$

s	t_1	s'	op_1
$\mathtt{s}_0$	$\mathtt{0}$	$\mathtt{s}_1$	$\mathtt{+1}$
$\mathtt{s}_0$	$\mathtt{1}$	$\mathtt{s}_0$	$\mathtt{-1}$

  Zähle Eingabewert n auf Null herunter und addiere Eins

- **Mathematische Analyse**:
  - Anfangskonfiguration: $\alpha(n) = (\mathtt{s}_0, n)$
  - Nachfolgekonfigurationen: $\hat{\delta}(\mathtt{s}_0, n) = \begin{cases} (\mathtt{s}_0, n-1) & \text{falls } n>0, \\ (\mathtt{s}_1, \mathtt{1}) & \text{falls } n=0 \end{cases}$
  - Terminierung: $\min\{j \mid \hat{\delta}^j(\mathtt{s}_0, n) = (\mathtt{s}_1, \_)\} = n+1$
  - Ergebnis: $\hat{\delta}^{n+1}(\mathtt{s}_0, n) = (\mathtt{s}_1, \mathtt{1})$
  - Ausgabefunktion: $\omega(\mathtt{s}_1, \mathtt{1}) = \mathtt{1}$

Beispiele für Registermaschinen

- $\boldsymbol{\rho_1} = (\{\mathtt{s}_0, \mathtt{s}_1\},\ \mathtt{1},\ \delta_1,\ \mathtt{s}_0,\ \{\mathtt{s}_1\})$ mit $\delta_1 =$

s	t_1	s'	op_1
$\mathtt{s}_0$	0	$\mathtt{s}_1$	+1
$\mathtt{s}_0$	1	$\mathtt{s}_0$	-1

  Zähle Eingabewert n auf Null herunter und addiere Eins

- **Mathematische Analyse**:
  - Anfangskonfiguration: $\alpha(n) = (\mathtt{s}_0, n)$
  - Nachfolgekonfigurationen: $\hat{\delta}(\mathtt{s}_0, n) = \begin{cases} (\mathtt{s}_0, n-1) & \text{falls } n>0, \\ (\mathtt{s}_1, \mathtt{1}) & \text{falls } n=0 \end{cases}$
  - Terminierung: $\min\{j \mid \hat{\delta}^j(\mathtt{s}_0, n) = (\mathtt{s}_1, \_)\} = n+1$
  - Ergebnis: $\hat{\delta}^{n+1}(\mathtt{s}_0, n) = (\mathtt{s}_1, \mathtt{1})$
  - Ausgabefunktion: $\omega(\mathtt{s}_1, \mathtt{1}) = \mathtt{1}$

$\Downarrow$

$h_{\rho_1}(n) = \mathtt{1}$ für alle n, Definitionsbereich $\mathbb{N}$, Wertebereich $\{\mathtt{1}\}$

Beispiele für Registermaschinen II

- $\boldsymbol{\rho_2} = (\{\mathtt{s}_0, \mathtt{s}_1, \mathtt{s}_2, \mathtt{s}_3, \mathtt{s}_4\}, 2, \delta_2, \mathtt{s}_0, \{\mathtt{s}_4\})$ mit $\delta_2 =$

s	t_1	t_2	s'	op_1	op_2
$\mathtt{s}_0$	0	0	$\mathtt{s}_4$	0	0
$\mathtt{s}_0$	0	1	$\mathtt{s}_2$	0	0
$\mathtt{s}_0$	1	*	$\mathtt{s}_1$	-1	+1
$\mathtt{s}_1$	*	*	$\mathtt{s}_0$	0	+1
$\mathtt{s}_2$	*	0	$\mathtt{s}_4$	0	0
$\mathtt{s}_2$	*	1	$\mathtt{s}_3$	+1	-1
$\mathtt{s}_3$	*	*	$\mathtt{s}_2$	+1	0

Beispiele für Registermaschinen II

- $\boldsymbol{\rho_2} = (\{\mathtt{s}_0, \mathtt{s}_1, \mathtt{s}_2, \mathtt{s}_3, \mathtt{s}_4\}, 2, \delta_2, \mathtt{s}_0, \{\mathtt{s}_4\})$ mit $\delta_2 =$

s	t_1	t_2	s'	op_1	op_2
$\mathtt{s}_0$	0	0	$\mathtt{s}_4$	0	0
$\mathtt{s}_0$	0	1	$\mathtt{s}_2$	0	0
$\mathtt{s}_0$	1	*	$\mathtt{s}_1$	-1	+1
$\mathtt{s}_1$	*	*	$\mathtt{s}_0$	0	+1
$\mathtt{s}_2$	*	0	$\mathtt{s}_4$	0	0
$\mathtt{s}_2$	*	1	$\mathtt{s}_3$	+1	-1
$\mathtt{s}_3$	*	*	$\mathtt{s}_2$	+1	0

- **Analyse**

Beispiele für Registermaschinen II

- $\boldsymbol{\rho_2} = (\{\mathtt{s}_0, \mathtt{s}_1, \mathtt{s}_2, \mathtt{s}_3, \mathtt{s}_4\}, 2, \delta_2, \mathtt{s}_0, \{\mathtt{s}_4\})$ mit $\delta_2 =$

s	t_1	t_2	s'	op_1	op_2
$\mathtt{s}_0$	0	0	$\mathtt{s}_4$	0	0
$\mathtt{s}_0$	0	1	$\mathtt{s}_2$	0	0
$\mathtt{s}_0$	1	*	$\mathtt{s}_1$	-1	+1
$\mathtt{s}_1$	*	*	$\mathtt{s}_0$	0	+1
$\mathtt{s}_2$	*	0	$\mathtt{s}_4$	0	0
$\mathtt{s}_2$	*	1	$\mathtt{s}_3$	+1	-1
$\mathtt{s}_3$	*	*	$\mathtt{s}_2$	+1	0

- **Analyse**

  $n \xrightarrow{\alpha} (\mathtt{s}_0, n, \mathtt{0})$

Beispiele für Registermaschinen II

- $\boldsymbol{\rho_2} = (\{\mathtt{s}_0, \mathtt{s}_1, \mathtt{s}_2, \mathtt{s}_3, \mathtt{s}_4\}, 2, \delta_2, \mathtt{s}_0, \{\mathtt{s}_4\})$ mit $\delta_2 =$

s	t_1	t_2	s'	op_1	op_2
$\mathtt{s}_0$	0	0	$\mathtt{s}_4$	0	0
$\mathtt{s}_0$	0	1	$\mathtt{s}_2$	0	0
$\mathtt{s}_0$	1	*	$\mathtt{s}_1$	-1	+1
$\mathtt{s}_1$	*	*	$\mathtt{s}_0$	0	+1
$\mathtt{s}_2$	*	0	$\mathtt{s}_4$	0	0
$\mathtt{s}_2$	*	1	$\mathtt{s}_3$	+1	-1
$\mathtt{s}_3$	*	*	$\mathtt{s}_2$	+1	0

- **Analyse**

$$n \xrightarrow{\alpha} (\mathtt{s}_0, n, \mathtt{0})$$
$$\xrightarrow{\delta} (\mathtt{s}_1, n-1, \mathtt{1})$$

Beispiele für Registermaschinen II

- $\boldsymbol{\rho_2} = (\{\mathtt{s}_0, \mathtt{s}_1, \mathtt{s}_2, \mathtt{s}_3, \mathtt{s}_4\}, 2, \delta_2, \mathtt{s}_0, \{\mathtt{s}_4\})$ mit $\delta_2 =$

s	t_1	t_2	s'	op_1	op_2
$\mathtt{s}_0$	0	0	$\mathtt{s}_4$	0	0
$\mathtt{s}_0$	0	1	$\mathtt{s}_2$	0	0
$\mathtt{s}_0$	1	*	$\mathtt{s}_1$	-1	+1
$\mathtt{s}_1$	*	*	$\mathtt{s}_0$	0	+1
$\mathtt{s}_2$	*	0	$\mathtt{s}_4$	0	0
$\mathtt{s}_2$	*	1	$\mathtt{s}_3$	+1	-1
$\mathtt{s}_3$	*	*	$\mathtt{s}_2$	+1	0

- **Analyse**

$n \xrightarrow{\alpha} (\mathtt{s}_0, n, 0)$

$\xrightarrow{\delta} (\mathtt{s}_1, n-1, 1) \qquad \xrightarrow{\delta} (\mathtt{s}_0, n-1, 2)$

Beispiele für Registermaschinen II

- $\boldsymbol{\rho_2} = (\{\mathtt{s}_0, \mathtt{s}_1, \mathtt{s}_2, \mathtt{s}_3, \mathtt{s}_4\}, 2, \delta_2, \mathtt{s}_0, \{\mathtt{s}_4\})$ mit $\delta_2 =$

s	t_1	t_2	s'	op_1	op_2
$\mathtt{s}_0$	0	0	$\mathtt{s}_4$	0	0
$\mathtt{s}_0$	0	1	$\mathtt{s}_2$	0	0
$\mathtt{s}_0$	1	*	$\mathtt{s}_1$	−1	+1
$\mathtt{s}_1$	*	*	$\mathtt{s}_0$	0	+1
$\mathtt{s}_2$	*	0	$\mathtt{s}_4$	0	0
$\mathtt{s}_2$	*	1	$\mathtt{s}_3$	+1	−1
$\mathtt{s}_3$	*	*	$\mathtt{s}_2$	+1	0

- **Analyse**

$$n \xrightarrow{\alpha} (\mathtt{s}_0, n, 0) \xrightarrow{\delta} (\mathtt{s}_1, n-1, 1) \xrightarrow{\delta} (\mathtt{s}_0, n-1, 2) \xrightarrow{\delta} \ldots \xrightarrow{\delta} (\mathtt{s}_2, 0, 2n)$$

Beispiele für Registermaschinen II

- $\boldsymbol{\rho_2} = (\{\mathtt{s}_0, \mathtt{s}_1, \mathtt{s}_2, \mathtt{s}_3, \mathtt{s}_4\}, 2, \delta_2, \mathtt{s}_0, \{\mathtt{s}_4\})$ mit $\delta_2 =$

s	t_1	t_2	s'	op_1	op_2
$\mathtt{s}_0$	0	0	$\mathtt{s}_4$	0	0
$\mathtt{s}_0$	0	1	$\mathtt{s}_2$	0	0
$\mathtt{s}_0$	1	*	$\mathtt{s}_1$	-1	+1
$\mathtt{s}_1$	*	*	$\mathtt{s}_0$	0	+1
$\mathtt{s}_2$	*	0	$\mathtt{s}_4$	0	0
$\mathtt{s}_2$	*	1	$\mathtt{s}_3$	+1	-1
$\mathtt{s}_3$	*	*	$\mathtt{s}_2$	+1	0

- **Analyse**

$$n \xrightarrow{\alpha} (\mathtt{s}_0, n, 0)$$
$$\xrightarrow{\delta} (\mathtt{s}_1, n-1, 1) \xrightarrow{\delta} (\mathtt{s}_0, n-1, 2) \xrightarrow{\delta} \ldots \xrightarrow{\delta} (\mathtt{s}_2, 0, 2n)$$
$$\xrightarrow{\delta} (\mathtt{s}_3, 1, 2n-1)$$

Beispiele für Registermaschinen II

- $\boldsymbol{\rho_2} = (\{\mathtt{s}_0, \mathtt{s}_1, \mathtt{s}_2, \mathtt{s}_3, \mathtt{s}_4\}, 2, \delta_2, \mathtt{s}_0, \{\mathtt{s}_4\})$ mit $\delta_2 =$

s	t_1	t_2	s'	op_1	op_2
$\mathtt{s}_0$	0	0	$\mathtt{s}_4$	0	0
$\mathtt{s}_0$	0	1	$\mathtt{s}_2$	0	0
$\mathtt{s}_0$	1	*	$\mathtt{s}_1$	−1	+1
$\mathtt{s}_1$	*	*	$\mathtt{s}_0$	0	+1
$\mathtt{s}_2$	*	0	$\mathtt{s}_4$	0	0
$\mathtt{s}_2$	*	1	$\mathtt{s}_3$	+1	−1
$\mathtt{s}_3$	*	*	$\mathtt{s}_2$	+1	0

- **Analyse**

$$n \xrightarrow{\alpha} (\mathtt{s}_0, n, 0)$$
$$\xrightarrow{\delta} (\mathtt{s}_1, n-1, 1) \xrightarrow{\delta} (\mathtt{s}_0, n-1, 2) \xrightarrow{\delta} \ldots \xrightarrow{\delta} (\mathtt{s}_2, 0, 2n)$$
$$\xrightarrow{\delta} (\mathtt{s}_3, 1, 2n-1) \xrightarrow{\delta} (\mathtt{s}_2, 2, 2n-1)$$

Beispiele für Registermaschinen II

- $\boldsymbol{\rho_2} = (\{\mathtt{s}_0, \mathtt{s}_1, \mathtt{s}_2, \mathtt{s}_3, \mathtt{s}_4\}, 2, \delta_2, \mathtt{s}_0, \{\mathtt{s}_4\})$ mit $\delta_2 =$

s	t_1	t_2	s'	op_1	op_2
$\mathtt{s}_0$	0	0	$\mathtt{s}_4$	0	0
$\mathtt{s}_0$	0	1	$\mathtt{s}_2$	0	0
$\mathtt{s}_0$	1	*	$\mathtt{s}_1$	-1	+1
$\mathtt{s}_1$	*	*	$\mathtt{s}_0$	0	+1
$\mathtt{s}_2$	*	0	$\mathtt{s}_4$	0	0
$\mathtt{s}_2$	*	1	$\mathtt{s}_3$	+1	-1
$\mathtt{s}_3$	*	*	$\mathtt{s}_2$	+1	0

- **Analyse**

$$
\begin{aligned}
n &\xrightarrow{\alpha} (\mathtt{s}_0, n, 0) \\
&\xrightarrow{\delta} (\mathtt{s}_1, n-1, 1) \xrightarrow{\delta} (\mathtt{s}_0, n-1, 2) \xrightarrow{\delta} \ldots \xrightarrow{\delta} (\mathtt{s}_2, 0, 2n) \\
&\xrightarrow{\delta} (\mathtt{s}_3, 1, 2n-1) \xrightarrow{\delta} (\mathtt{s}_2, 2, 2n-1) \xrightarrow{\delta} \ldots \xrightarrow{\delta} (\mathtt{s}_4, 4n, 0)
\end{aligned}
$$

Beispiele für Registermaschinen II

- $\boldsymbol{\rho_2} = (\{\mathtt{s}_0, \mathtt{s}_1, \mathtt{s}_2, \mathtt{s}_3, \mathtt{s}_4\}, 2, \delta_2, \mathtt{s}_0, \{\mathtt{s}_4\})$ mit $\delta_2 =$

s	t_1	t_2	s'	op_1	op_2
$\mathtt{s}_0$	0	0	$\mathtt{s}_4$	0	0
$\mathtt{s}_0$	0	1	$\mathtt{s}_2$	0	0
$\mathtt{s}_0$	1	*	$\mathtt{s}_1$	-1	+1
$\mathtt{s}_1$	*	*	$\mathtt{s}_0$	0	+1
$\mathtt{s}_2$	*	0	$\mathtt{s}_4$	0	0
$\mathtt{s}_2$	*	1	$\mathtt{s}_3$	+1	-1
$\mathtt{s}_3$	*	*	$\mathtt{s}_2$	+1	0

- **Analyse**

$$
\begin{aligned}
n &\xrightarrow{\alpha} (\mathtt{s}_0, n, 0) \\
&\xrightarrow{\delta} (\mathtt{s}_1, n-1, 1) \xrightarrow{\delta} (\mathtt{s}_0, n-1, 2) \xrightarrow{\delta} \ldots \xrightarrow{\delta} (\mathtt{s}_2, 0, 2n) \\
&\xrightarrow{\delta} (\mathtt{s}_3, 1, 2n-1) \xrightarrow{\delta} (\mathtt{s}_2, 2, 2n-1) \xrightarrow{\delta} \ldots \xrightarrow{\delta} (\mathtt{s}_4, 4n, 0) \\
&\xrightarrow{\omega} 4n
\end{aligned}
$$

Beispiele für Registermaschinen II

- $\boldsymbol{\rho_2} = (\{\mathtt{s}_0, \mathtt{s}_1, \mathtt{s}_2, \mathtt{s}_3, \mathtt{s}_4\}, 2, \delta_2, \mathtt{s}_0, \{\mathtt{s}_4\})$ mit $\delta_2 =$

s	t_1	t_2	s'	op_1	op_2
$\mathtt{s}_0$	0	0	$\mathtt{s}_4$	0	0
$\mathtt{s}_0$	0	1	$\mathtt{s}_2$	0	0
$\mathtt{s}_0$	1	*	$\mathtt{s}_1$	−1	+1
$\mathtt{s}_1$	*	*	$\mathtt{s}_0$	0	+1
$\mathtt{s}_2$	*	0	$\mathtt{s}_4$	0	0
$\mathtt{s}_2$	*	1	$\mathtt{s}_3$	+1	−1
$\mathtt{s}_3$	*	*	$\mathtt{s}_2$	+1	0

- **Analyse**

$$
\begin{aligned}
n &\xrightarrow{\alpha} (\mathtt{s}_0, n, 0) \\
&\xrightarrow{\delta} (\mathtt{s}_1, n-1, 1) \xrightarrow{\delta} (\mathtt{s}_0, n-1, 2) \xrightarrow{\delta} \ldots \xrightarrow{\delta} (\mathtt{s}_2, 0, 2n) \\
&\xrightarrow{\delta} (\mathtt{s}_3, 1, 2n-1) \xrightarrow{\delta} (\mathtt{s}_2, 2, 2n-1) \xrightarrow{\delta} \ldots \xrightarrow{\delta} (\mathtt{s}_4, 4n, 0) \\
&\xrightarrow{\omega} 4n
\end{aligned}
$$

$\Downarrow$

$h_{\rho_2}(n) = 4n$ für alle n, Definitionsbereich $\mathbb{N}$, Wertebereich $\{4n | n \in \mathbb{N}\}$

Register-Berechenbarkeit

Definition H

- f:$\mathbb{N}\to\mathbb{N}$ **RM_k-berechenbar**
  - Es gibt eine Registermaschine $\rho = (S, k, \delta, s_0, F)$ mit $h_\rho = f$

- f:$\mathbb{N}^m\to\mathbb{N}^n$ **RM_k-berechenbar** $(k \geq \max(m,n))$
  - Es gibt eine Registermaschine $\rho = (S, k, \delta, s_0, F)$ mit $h_\rho = f$ und
    - Anfangskonfiguration α^m:$\mathbb{N}^m\to K_\rho$: $\alpha^m(n_1,..,n_m) = (s_0,(n_1,..,n_m,0,...,0))$
    - Ausgabefunktion ω^n:$K_\rho\to\mathbb{N}^n$: $\omega^n(s,(i_1,..,i_k)) = i_1,..,i_n$

- **$\mathcal{RM}$: Menge der Register-berechenbaren Funktionen**
  - $\mathcal{RM}_k = \{f$:$\mathbb{N}\to\mathbb{N} \mid f$ ist RM_k-berechenbar$\}$
  - $\mathcal{RM} = \bigcup\{\mathcal{RM}_k \mid k \in \mathbb{N}\}$

Beispiele Register-Berechenbarer Funktionen

- **Konstante Funktion** $f_3(n) = c$

Beispiele Register-Berechenbarer Funktionen

- **Konstante Funktion** $f_3(n) = c$
  - ρ_3 muß Register s_0 auf Null herunterzählen und dann c mal 1 addieren

Beispiele Register-Berechenbarer Funktionen

- **Konstante Funktion** $f_3(n) = c$
  - ρ_3 muß Register $\mathtt{s}_0$ auf Null herunterzählen und dann c mal 1 addieren

$\rho_3 = (\{\mathtt{s}_0, \ldots, \mathtt{s}_c\}, \mathtt{1}, \delta_3, \mathtt{s}_0, \{\mathtt{s}_c\})$ mit $\delta_3 =$

s	t_1	s'	op_1
$\mathtt{s}_0$	0	$\mathtt{s}_1$	+1
$\mathtt{s}_0$	1	$\mathtt{s}_0$	-1
$\mathtt{s}_1$	*	$\mathtt{s}_2$	+1
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\mathtt{s}_{c-1}$	*	$\mathtt{s}_c$	+1

Beispiele Register-Berechenbarer Funktionen

- **Konstante Funktion** $f_3(n) = c$
  - ρ_3 muß Register $\mathtt{s}_0$ auf Null herunterzählen und dann c mal 1 addieren

$\rho_3 = (\{\mathtt{s}_0, \ldots, \mathtt{s}_c\}, \mathtt{1}, \delta_3, \mathtt{s}_0, \{\mathtt{s}_c\})$ mit $\delta_3 =$

s	t_1	s'	op_1
$\mathtt{s}_0$	0	$\mathtt{s}_1$	+1
$\mathtt{s}_0$	1	$\mathtt{s}_0$	-1
$\mathtt{s}_1$	*	$\mathtt{s}_2$	+1
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\mathtt{s}_{c-1}$	*	$\mathtt{s}_c$	+1

- **Addition** $f_4(n, m) = n+m$

Beispiele Register-Berechenbarer Funktionen

- **Konstante Funktion** $f_3(n) = c$
  - ρ_3 muß Register $\mathtt{s}_0$ auf Null herunterzählen und dann c mal 1 addieren

$\rho_3 = (\{\mathtt{s}_0, \ldots, \mathtt{s}_c\}, \mathtt{1}, \delta_3, \mathtt{s}_0, \{\mathtt{s}_c\})$ mit $\delta_3 =$

s	t_1	s'	op_1
$\mathtt{s}_0$	0	$\mathtt{s}_1$	+1
$\mathtt{s}_0$	1	$\mathtt{s}_0$	-1
$\mathtt{s}_1$	*	$\mathtt{s}_2$	+1
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\mathtt{s}_{c-1}$	*	$\mathtt{s}_c$	+1

- **Addition** $f_4(n, m) = n+m$
  - ρ_4 muß Register $\mathtt{s}_1$ auf Null herunterzählen und dabei $\mathtt{s}_0$ hochzählen

Beispiele Register-Berechenbarer Funktionen

- **Konstante Funktion** $f_3(n) = c$
  - ρ_3 muß Register $\mathtt{s}_0$ auf Null herunterzählen und dann c mal 1 addieren

$\rho_3 = (\{\mathtt{s}_0, \ldots, \mathtt{s}_c\}, \mathtt{1}, \delta_3, \mathtt{s}_0, \{\mathtt{s}_c\})$ mit $\delta_3 =$

s	t_1	s'	op_1
$\mathtt{s}_0$	0	$\mathtt{s}_1$	+1
$\mathtt{s}_0$	1	$\mathtt{s}_0$	-1
$\mathtt{s}_1$	*	$\mathtt{s}_2$	+1
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\mathtt{s}_{c-1}$	*	$\mathtt{s}_c$	+1

- **Addition** $f_4(n, m) = n+m$
  - ρ_4 muß Register $\mathtt{s}_1$ auf Null herunterzählen und dabei $\mathtt{s}_0$ hochzählen

$\rho_4 = (\{\mathtt{s}_0, \mathtt{s}_1\}, \mathtt{1}, \delta_4, \mathtt{s}_0, \{\mathtt{s}_1\})$ mit $\delta_4 =$

s	t_1	t_2	s'	op_1	op_2
$\mathtt{s}_0$	*	0	$\mathtt{s}_1$	0	0
$\mathtt{s}_0$	*	1	$\mathtt{s}_0$	+1	-1

Beispiele Register-Berechenbarer Funktionen

- **Konstante Funktion** $f_3(n) = c$
  - ρ_3 muß Register $\mathtt{s}_0$ auf Null herunterzählen und dann c mal 1 addieren

  $\rho_3 = (\{\mathtt{s}_0, \ldots, \mathtt{s}_c\}, \mathtt{1}, \delta_3, \mathtt{s}_0, \{\mathtt{s}_c\})$ mit $\delta_3 =$

s	t_1	s'	op_1
$\mathtt{s}_0$	0	$\mathtt{s}_1$	+1
$\mathtt{s}_0$	1	$\mathtt{s}_0$	-1
$\mathtt{s}_1$	*	$\mathtt{s}_2$	+1
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\mathtt{s}_{c-1}$	*	$\mathtt{s}_c$	+1

- **Addition** $f_4(n, m) = n+m$
  - ρ_4 muß Register $\mathtt{s}_1$ auf Null herunterzählen und dabei $\mathtt{s}_0$ hochzählen

  $\rho_4 = (\{\mathtt{s}_0, \mathtt{s}_1\}, \mathtt{1}, \delta_4, \mathtt{s}_0, \{\mathtt{s}_1\})$ mit $\delta_4 =$

s	t_1	t_2	s'	op_1	op_2
$\mathtt{s}_0$	*	0	$\mathtt{s}_1$	0	0
$\mathtt{s}_0$	*	1	$\mathtt{s}_0$	+1	-1

- **Multiplikation** $f_5(n, m) = n*m$

Beispiele Register-Berechenbarer Funktionen

- **Konstante Funktion** $f_3(n) = c$
  - ρ_3 muß Register $\mathtt{s}_0$ auf Null herunterzählen und dann c mal 1 addieren

  $\rho_3 = (\{\mathtt{s}_0, \ldots, \mathtt{s}_c\}, \mathtt{1}, \delta_3, \mathtt{s}_0, \{\mathtt{s}_c\})$ mit $\delta_3 =$

s	t_1	s'	op_1
$\mathtt{s}_0$	0	$\mathtt{s}_1$	+1
$\mathtt{s}_0$	1	$\mathtt{s}_0$	-1
$\mathtt{s}_1$	*	$\mathtt{s}_2$	+1
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\mathtt{s}_{c-1}$	*	$\mathtt{s}_c$	+1

- **Addition** $f_4(n, m) = n+m$
  - ρ_4 muß Register $\mathtt{s}_1$ auf Null herunterzählen und dabei $\mathtt{s}_0$ hochzählen

  $\rho_4 = (\{\mathtt{s}_0, \mathtt{s}_1\}, \mathtt{1}, \delta_4, \mathtt{s}_0, \{\mathtt{s}_1\})$ mit $\delta_4 =$

s	t_1	t_2	s'	op_1	op_2
$\mathtt{s}_0$	*	0	$\mathtt{s}_1$	0	0
$\mathtt{s}_0$	*	1	$\mathtt{s}_0$	+1	-1

- **Multiplikation** $f_5(n, m) = n*m$
  - ρ_5 muß $\mathtt{s}_1$ auf Null herunterzählen und dabei jeweils $\mathtt{s}_0$+n berechnen
  - n muß zuvor in Hilfsregister kopiert werden

Erweiterung des Modells für Programmierzwecke

- **Unterprogramme**
  - Umbenennung: Separate Zustände $s'_0,..s'_n$ und Register $r'_1,..,r'_k$
  - Aufruf: speichere Argumente in r'_1, springe nach s'_0
  - Rückgabe: kopiere Werte von r'_1 ins gewünschte Register, springe zum Folgezustand des Aufrufs

Erweiterung des Modells für Programmierzwecke

- **Unterprogramme**
  - Umbenennung: Separate Zustände $s'_0,..s'_n$ und Register $r'_1,..,r'_k$
  - Aufruf: speichere Argumente in r'_1, springe nach s'_0
  - Rückgabe: kopiere Werte von r'_1 ins gewünschte Register, springe zum Folgezustand des Aufrufs

- **RM-Programmiersprache**

`r`$_j$ `:= r`$_j$`+1`	
`r`$_j$ `:= r`$_j$$\dot{-}$`1`	$i \dot{-} j = \max(i-j,0)$
`r`$_j$ `:=` c	($c \in \mathbb{N}$)
`while r`$_j$`=0 do` op `od`	(op beliebiger Befehl)

  - Jeder Befehl kann durch RM-Unterprogramme simuliert werden

Erweiterung des Modells für Programmierzwecke

- **Unterprogramme**
  - Umbenennung: Separate Zustände $s'_0,..s'_n$ und Register $r'_1,..,r'_k$
  - Aufruf: speichere Argumente in r'_1, springe nach s'_0
  - Rückgabe: kopiere Werte von r'_1 ins gewünschte Register, springe zum Folgezustand des Aufrufs
- **RM-Programmiersprache**

  `r`$_j$ `:= r`$_j$`+1`

  `r`$_j$ `:= r`$_j \dot{-} $`1` $\quad i \dot{-} j = \max(i-j,0)$

  `r`$_j$ `:= `c $\quad (c \in \mathbb{N})$

  `while r`$_j$`=0 do `op` od` (op beliebiger Befehl)
  - Jeder Befehl kann durch RM-Unterprogramme simuliert werden
- **Befehlsmacros**
  - Abkürzungen für Programmfragmente in RM-Programmiersprache

Simulation der RM-Programmiersprache

- $\texttt{r}_j$ `:=` $\texttt{r}_j$`+1`

Simulation der RM-Programmiersprache

- `r`$_j$ `:= r`$_j$`+1`
  - Direkter RM Befehl

Simulation der RM-Programmiersprache

- $\texttt{r}_j$ `:=` $\texttt{r}_j$`+1`
  - Direkter RM Befehl
- $\texttt{r}_j$ `:=` $\texttt{r}_j \dot{-} \texttt{1}$

Simulation der RM-Programmiersprache

- $\texttt{r}_j$ `:=` $\texttt{r}_j$`+1`
  - Direkter RM Befehl
- $\texttt{r}_j$ `:=` $\texttt{r}_j \dot{-} \texttt{1}$
  - Direkter RM Befehl

Simulation der RM-Programmiersprache

- $\texttt{r}_j$ `:=` $\texttt{r}_j$`+1`
  - Direkter RM Befehl
- $\texttt{r}_j$ `:=` $\texttt{r}_j \dot{-} \texttt{1}$
  - Direkter RM Befehl
- $\texttt{r}_j$ `:=` c

Simulation der RM-Programmiersprache

- $\texttt{r}_j$ `:=` $\texttt{r}_j$`+1`
  - Direkter RM Befehl
- $\texttt{r}_j$ `:=` $\texttt{r}_j \dot{-} \texttt{1}$
  - Direkter RM Befehl
- $\texttt{r}_j$ `:=` c
  - Unterprogramm ähnlich zu ρ_3

Simulation der RM-Programmiersprache

- $\texttt{r}_j$ `:=` $\texttt{r}_j$`+1`
  - Direkter RM Befehl
- $\texttt{r}_j$ `:=` $\texttt{r}_j \dot{-} \texttt{1}$
  - Direkter RM Befehl
- $\texttt{r}_j$ `:=` c
  - Unterprogramm ähnlich zu ρ_3
- `while` $\texttt{r}_j$`>0 do` op `od`

Simulation der RM-Programmiersprache

- $\mathtt{r}_j$ `:=` $\mathtt{r}_j$`+1`
  - Direkter RM Befehl
- $\mathtt{r}_j$ `:=` $\mathtt{r}_j \dot{-} \mathtt{1}$
  - Direkter RM Befehl
- $\mathtt{r}_j$ `:=` c
  - Unterprogramm ähnlich zu ρ_3
- `while` $\mathtt{r}_j$`>0 do` op `od`
  - Solange $\mathtt{r}_j$>0 springe zu Startzustand von op
  - Am Endzustand von op springe zurück zum Anfang
  - Wenn $\mathtt{r}_j$=0 gehe zum Endzustand des Befehls

$\delta =$

s	...	t_j	...	s'	...$_1$	op_j	...
s	*	0	*	$\mathtt{s}_e$	0 ...	0	...0
s	*	1	*	$\mathtt{s}_{op_0}$	0 ...	0	...0
$\mathtt{s}_{op_0}$					— op —		
$\mathtt{s}_{op_e}$	*	*	*	s	0 ...	0	...0

Befehlsmacros I: Arithmetik

- $\texttt{r}_j$ `:=` $\texttt{r}_i$

Befehlsmacros I: Arithmetik

- r_j := r_i
  - Verschiebe r_i in Hilfsregister r' und kopiere r' simultan nach r_j und r_i

```
rj:=0; r':=0;  while ri>0 do ri:=ri−1; r':=r'+1 od;
       while r'>0 do r':=r'−1; rj:=rj+1; ri:=ri+1 od
```

Befehlsmacros I: Arithmetik

- r_j := r_i
  - Verschiebe r_i in Hilfsregister r' und kopiere r' simultan nach r_j und r_i

```
rj:=0; r':=0;  while ri>0 do ri:=ri−1; r':=r'+1 od;
       while r'>0 do r':=r'−1; rj:=rj+1; ri:=ri+1 od
```

- r_j := r_j+r_i

Befehlsmacros I: Arithmetik

- r_j := r_i
  - Verschiebe r_i in Hilfsregister r' und kopiere r' simultan nach r_j und r_i

```
rj:=0; r':=0;  while ri>0 do ri:=ri−1; r':=r'+1 od;
       while r'>0 do r':=r'−1; rj:=rj+1; ri:=ri+1 od
```

- r_j := r_j+r_i
  - Kopiere r_i in nach r' und zähle simultan r' herunter und r_j hoch

```
r' := ri;  while r'>0 do r':=r'−1; rj:=rj+1 od
```

Befehlsmacros I: Arithmetik

- r_j := r_i
  - Verschiebe r_i in Hilfsregister r' und kopiere r' simultan nach r_j und r_i

```
rj:=0; r':=0;  while ri>0 do ri:=ri−1; r':=r'+1 od;
     while r'>0 do r':=r'−1; rj:=rj+1; ri:=ri+1 od
```

- r_j := r_j+r_i
  - Kopiere r_i in nach r' und zähle simultan r' herunter und r_j hoch

```
r' := ri;  while r'>0 do r':=r'−1; rj:=rj+1 od
```

- r_j := $r_j \dot{-} r_i$

Befehlsmacros I: Arithmetik

- r_j := r_i
  - Verschiebe r_i in Hilfsregister r' und kopiere r' simultan nach r_j und r_i

```
rj:=0; r':=0;  while ri>0 do ri:=ri−1; r':=r'+1 od;
     while r'>0 do r':=r'−1; rj:=rj+1; ri:=ri+1 od
```

- r_j := r_j+r_i
  - Kopiere r_i in nach r' und zähle simultan r' herunter und r_j hoch

```
r' := ri;  while r'>0 do r':=r'−1; rj:=rj+1 od
```

- r_j := $r_j \dot{-} r_i$
  - Kopiere r_i nach r' und reduziere r' und r_j simultan

```
r':=ri;  while r'>0 do r':=r'−1; rj:=rj−1 od
```

Befehlsmacros I: Arithmetik

- r_j := r_i
  - Verschiebe r_i in Hilfsregister r' und kopiere r' simultan nach r_j und r_i

```
rj:=0; r':=0;  while ri>0 do ri:=ri−1; r':=r'+1 od;
       while r'>0 do r':=r'−1; rj:=rj+1; ri:=ri+1 od
```

- r_j := r_j+r_i
  - Kopiere r_i in nach r' und zähle simultan r' herunter und r_j hoch

```
r' := ri;  while r'>0 do r':=r'−1; rj:=rj+1 od
```

- r_j := r_j−r_i
  - Kopiere r_i nach r' und reduziere r' und r_j simultan

```
r':=ri;  while r'>0 do r':=r'−1; rj:=rj−1 od
```

- r_j := r_j*r_i

Befehlsmacros I: Arithmetik

- r_j := r_i
  - Verschiebe r_i in Hilfsregister r' und kopiere r' simultan nach r_j und r_i

```
rj:=0; r':=0;  while ri>0 do ri:=ri−1; r':=r'+1 od;
     while r'>0 do r':=r'−1; rj:=rj+1; ri:=ri+1 od
```

- r_j := r_j+r_i
  - Kopiere r_i in nach r' und zähle simultan r' herunter und r_j hoch

```
r' := ri;  while r'>0 do r':=r'−1; rj:=rj+1 od
```

- r_j := r_j−r_i
  - Kopiere r_i nach r' und reduziere r' und r_j simultan

```
r':=ri;  while r'>0 do r':=r'−1; rj:=rj−1 od
```

- r_j := r_j*r_i
  - r_i-faches Aufaddieren von r_j unter Verwendung von Hilfsregistern

```
r':=ri; r'':=rj;  while r'>0 do r':=r'−1; rj:=rj+r'' od
```

Befehlsmacros I: Arithmetik

- r_j := r_i
  - Verschiebe r_i in Hilfsregister r' und kopiere r' simultan nach r_j und r_i

    r_j:=0; r':=0; while r_i>0 do r_i:=$r_i \dot{-} 1$; r':=r'+1 od;
    while r'>0 do r':=r'$\dot{-}$1; r_j:=r_j+1; r_i:=r_i+1 od

- r_j := r_j+r_i
  - Kopiere r_i in nach r' und zähle simultan r' herunter und r_j hoch

    r' := r_i; while r'>0 do r':=r'$\dot{-}$1; r_j:=r_j+1 od

- r_j := $r_j \dot{-} r_i$
  - Kopiere r_i nach r' und reduziere r' und r_j simultan

    r':=r_i; while r'>0 do r':=r'$\dot{-}$1; r_j:=$r_j \dot{-} 1$ od

- r_j := r_j*r_i
  - r_i-faches Aufaddieren von r_j unter Verwendung von Hilfsregistern

    r':=r_i; r'':=r_j; while r'>0 do r':=r'$\dot{-}$1; r_j:=r_j+r'' od

- r_j := $r_j^{r_i}$

Befehlsmacros I: Arithmetik

- r_j := r_i
  - Verschiebe r_i in Hilfsregister r' und kopiere r' simultan nach r_j und r_i

```
rj:=0; r':=0;  while ri>0 do ri:=ri−1; r':=r'+1 od;
     while r'>0 do r':=r'−1; rj:=rj+1; ri:=ri+1 od
```

- r_j := r_j+r_i
  - Kopiere r_i in nach r' und zähle simultan r' herunter und r_j hoch

```
r' := ri;  while r'>0 do r':=r'−1; rj:=rj+1 od
```

- r_j := r_j − r_i
  - Kopiere r_i nach r' und reduziere r' und r_j simultan

```
r':=ri;  while r'>0 do r':=r'−1; rj:=rj−1 od
```

- r_j := r_j*r_i
  - r_i-faches Aufaddieren von r_j unter Verwendung von Hilfsregistern

```
r':=ri; r'':=rj;  while r'>0 do r':=r'−1; rj:=rj+r'' od
```

- r_j := $r_j^{r_i}$

```
r':=ri; r'':=rj; rj:=1;
while r'>0 do r':=r'−1; rj:=rj*r'' od
```

Befehlsmacros II: Programmstrukturen

- `while` exp`(r`$_j$`)>0 do` op `od` ($\mathtt{r}_j$ `:=` exp`(r`$_j$`)` programmierbar)

Befehlsmacros II: Programmstrukturen

- `while` exp`(r`$_j$`)>0 do` op `od` ($\texttt{r}_j$:= $exp(\texttt{r}_j)$ programmierbar)

  `r':=r`$_j$`; r':=`exp`(r');`

  `while r'>0 do` op`; r':=r`$_j$`; r':=`exp`(r') od`

Befehlsmacros II: Programmstrukturen

- `while` exp`(r`$_j$`)>0 do` op `od` ($\mathtt{r}_j$:= $exp(\mathtt{r}_j)$ programmierbar)

  `r':=r`$_j$`; r':=`exp`(r');`

  `while r'>0 do` op`; r':=r`$_j$`; r':=`exp`(r') od`

- `while` exp`(r`$_j$`)=0 do` op `od`

Befehlsmacros II: Programmstrukturen

- `while` exp`(r`$_j$`)>0 do` op `od` ($\mathtt{r}_j$:= $exp(\mathtt{r}_j)$ programmierbar)

  `r':=r`$_j$`; r':=`exp`(r');`

  `while r'>0 do` op`; r':=r`$_j$`; r':=`exp`(r') od`

- `while` exp`(r`$_j$`)=0 do` op `od`

  `while 1`$\dot{-}$$exp$`(r`$_j$`)>0 do` op `od`

Befehlsmacros II: Programmstrukturen

- `while` $exp(\mathtt{r}_j)$`>0 do` op `od` ($\mathtt{r}_j$ `:=` $exp(\mathtt{r}_j)$ programmierbar)

  `r':=`$\mathtt{r}_j$`; r':=`exp`(r');`

  `while r'>0 do` op`; r':=`$\mathtt{r}_j$`; r':=`exp`(r') od`

- `while` $exp(\mathtt{r}_j)$`=0 do` op `od`

  `while 1`$\dot{-}$$exp(\mathtt{r}_j)$`>0 do` op `od`

- `if` $\mathtt{r}_j$`=0 then` op `fi`

Befehlsmacros II: Programmstrukturen

- `while` $exp(\mathtt{r}_j)$`>0 do` op `od` ($\mathtt{r}_j$ `:=` $exp(\mathtt{r}_j)$ programmierbar)

  `r':=`$\mathtt{r}_j$`;` exp`(r');`

  `while r'>0 do` op`; r':=`$\mathtt{r}_j$`;` exp`(r') od`

- `while` $exp(\mathtt{r}_j)$`=0 do` op `od`

  `while 1`$\dot{-}$$exp(\mathtt{r}_j)$`>0 do` op `od`

- `if` $\mathtt{r}_j$`=0 then` op `fi`

  `r':=`$\mathtt{r}_j$`;  while r'=0 do` op`; r':=1 od`

Befehlsmacros II: Programmstrukturen

- while $exp(r_j)>0$ do op od (r_j := $exp(r_j)$ programmierbar)

  r':=r_j; r':=exp(r');

  while r'>0 do op; r':=r_j; r':=exp(r') od

- while $exp(r_j)=0$ do op od

  while $1 \dot{-} exp(r_j)>0$ do op od

- if r_j=0 then op fi

  r':=r_j; while r'=0 do op; r':=1 od

- if $r_j \leq r_i$ then op fi

Befehlsmacros II: Programmstrukturen

- `while` exp`(r`$_j$`)>0 do` op `od` ($\texttt{r}_j$:= $exp(\texttt{r}_j)$ programmierbar)

  `r':=r`$_j$`; r':=`exp`(r');`

  `while r'>0 do` op`; r':=r`$_j$`; r':=`exp`(r') od`

- `while` exp`(r`$_j$`)=0 do` op `od`

  `while 1`$\dot{-}$$exp$`(r`$_j$`)>0 do` op `od`

- `if r`$_j$`=0 then` op `fi`

  `r':=r`$_j$`; while r'=0 do` op`; r':=1 od`

- `if r`$_j$$\leq$`r`$_i$ `then` op `fi`

  `r':=r`$_i$$\dot{-}$`r`$_j$`; if r'=0 then` op `fi`

  – Vergleichsoperation $\geq, <, >, =, \neq$ analog

Befehlsmacros III: Division und Produkte

- $\mathtt{r}_j$ `:=` $\mathtt{r}_j \mathbin{\dot{-}} \mathtt{r}_i$ (Integerdivision)

Befehlsmacros III: Division und Produkte

- r_j := $r_j \dot{-} r_i$ (Integerdivision)
  - Subtrahiere r_i wiederholt von Kopie von r_j und zähle dabei r_j hoch
  - Korrektur um $\dot{-}1$, wenn Divisionsrest bleibt

```
r':=r_j; r_j:=0;
while r'>0 do r'':=r'; r':=r' -. r_i; r_j:=r_j+1 od
if r''<r_i then r_j:=r_j -. 1
```

Befehlsmacros III: Division und Produkte

- r_j := $r_j \dot{-} r_i$ (Integerdivision)
  - Subtrahiere r_i wiederholt von Kopie von r_j und zähle dabei r_j hoch
  - Korrektur um $\dot{-}1$, wenn Divisionsrest bleibt

```
r':=r_j; r_j:=0;
while r'>0 do r'':=r'; r':=r' -. r_i; r_j:=r_j+1 od
if r''<r_i then r_j:=r_j -. 1
```

- r_j := r_j mod r_i

Befehlsmacros III: Division und Produkte

- r_j := $r_j \dot{\div} r_i$ (Integerdivision)
  - Subtrahiere r_i wiederholt von Kopie von r_j und zähle dabei r_j hoch
  - Korrektur um $\dot{-}1$, wenn Divisionsrest bleibt

```
r':=r_j; r_j:=0;
while r'>0 do r'':=r'; r':=r' -. r_i; r_j:=r_j+1 od
if r''<r_i then r_j:=r_j -. 1
```

- r_j := r_j mod r_i
  - Subtrahiere r_i wiederholt von r_j und speichere letzten "Divisionsrest"
  - Korrektur: Ergebnis bleibt 0, wenn Divisionsrest gleich r_i ist

```
while r_j>0 do r':= r_j; r_j:=r_j -. r_i od;
if r'<r_i then r_j:= r'
```

Befehlsmacros III: Division und Produkte

- $\mathtt{r}_j$ `:=` $\mathtt{r}_j \mathbin{\dot{\div}} \mathtt{r}_i$ (Integerdivision)
  - Subtrahiere r_i wiederholt von Kopie von r_j und zähle dabei r_j hoch
  - Korrektur um $\dot{-}1$, wenn Divisionsrest bleibt

```
r':=r_j; r_j:=0;
while r'>0 do r'':=r'; r':=r' -. r_i; r_j:=r_j+1 od
if r''<r_i then r_j:=r_j -. 1
```

- $\mathtt{r}_j$ `:=` $\mathtt{r}_j$ `mod` $\mathtt{r}_i$
  - Subtrahiere r_i wiederholt von r_j und speichere letzten "Divisionsrest"
  - Korrektur: Ergebnis bleibt 0, wenn Divisionsrest gleich r_i ist

```
while r_j>0 do r':= r_j; r_j:=r_j -. r_i od;
if r'<r_i then r_j:= r'
```

- $\mathtt{r}_j$ `:=` $\prod_{j=1}^{\mathtt{r}_i} exp(j)$

Befehlsmacros III: Division und Produkte

- $\mathtt{r}_j$ `:=` $\mathtt{r}_j \dot{\div} \mathtt{r}_i$ (Integerdivision)
  - Subtrahiere r_i wiederholt von Kopie von r_j und zähle dabei r_j hoch
  - Korrektur um $\dot{-}1$, wenn Divisionsrest bleibt

```
r':=r_j; r_j:=0;
while r'>0 do r'':=r'; r':=r' -. r_i; r_j:=r_j+1 od
if r''<r_i then r_j:=r_j -. 1
```

- $\mathtt{r}_j$ `:=` $\mathtt{r}_j$ `mod` $\mathtt{r}_i$
  - Subtrahiere r_i wiederholt von r_j und speichere letzten "Divisionsrest"
  - Korrektur: Ergebnis bleibt 0, wenn Divisionsrest gleich r_i ist

```
while r_j>0 do r':= r_j; r_j:=r_j -. r_i od;
if r'<r_i then r_j:= r'
```

- $\mathtt{r}_j$ `:=` $\prod_{j=1}^{\mathtt{r}_i} exp(j)$

```
r':=r_i; r_j:=1;  while r'>0 do r_j:=r_j*exp(r'); r':=r' -. 1 od
```

Befehlsmacros IV: Primfaktoren

- `r`$_j$ `:= prime(r`$_i$`)` (`prime(`x`)` $= 0$, wenn x Primzahl ist)

Befehlsmacros IV: Primfaktoren

- r_j := prime(r_i)

  ($\texttt{prime}(x) = 0$, wenn x Primzahl ist)

```
r_j:=1; r':=2;
while r_i-r'>0 do
   if r_i mod r' = 0 then r_j:=0 fi;
   r':=r'+1
od
```

Befehlsmacros IV: Primfaktoren

- r_j := prime(r_i) (prime(x) $= 0$, wenn x Primzahl ist)

```
r_j:=1; r':=2;
while r_i-r'>0 do
   if r_i mod r' = 0 then r_j:=0 fi;
   r':=r'+1
od
```

- r_j := nth-prime(r_i) (nth-prime(n) ist die n-te Primzahl)

Befehlsmacros IV: Primfaktoren

- r_j := prime(r_i) (prime(x) = 0, wenn x Primzahl ist)

```
r_j:=1; r':=2;
while r_i-r'>0 do
   if r_i mod r' = 0 then r_j:=0 fi;
   r':=r'+1
od
```

- r_j := nth-prime(r_i) (nth-prime(n) ist die n-te Primzahl)

```
r':=r_i; r_j:=1;
while r'>0 do
   r_j:=r_j+1;
   if prime(r_j) = 0 then r':=r'-1 fi
od
```

Befehlsmacros IV: Primfaktoren

- r_j := prime(r_i) (prime(x) $= 0$, wenn x Primzahl ist)

```
r_j:=1; r':=2;
while r_i-r'>0 do
   if r_i mod r' = 0 then r_j:=0 fi;
   r':=r'+1
od
```

- r_j := nth-prime(r_i) (nth-prime(n) ist die n-te Primzahl)

```
r':=r_i; r_j:=1;
while r'>0 do
   r_j:=r_j+1;
   if prime(r_j) = 0 then r':=r'-1 fi
od
```

- r_k := prim-exp(r_j,r_i) (prim-exp(n,p) $= \max\{k | p^k$ teilt $n\}$)

Befehlsmacros IV: Primfaktoren

- `r`$_j$ `:= prime(r`$_i$`)` (`prime`$(x) = 0$, wenn x Primzahl ist)

```
r_j:=1; r':=2;
while r_i-r'>0 do
   if r_i mod r' = 0 then r_j:=0 fi;
   r':=r'+1
od
```

- `r`$_j$ `:= nth-prime(r`$_i$`)` (`nth-prime`(n) ist die n-te Primzahl)

```
r':=r_i; r_j:=1;
while r'>0 do
   r_j:=r_j+1;
   if prime(r_j) = 0 then r':=r'-1 fi
od
```

- `r`$_k$ `:= prim-exp(r`$_j$`,r`$_i$`)` (`prim-exp`$(n,p) = \max\{k | p^k$ teilt $n\}$)

```
r_k:=0; r':=r_j;
while r' mod r_i = 0 do r':=r' div r_i; r_k:=r_k+1 od
```

Register-Berechenbarkeit $\hat{=}$ Turing-Berechenbarkeit

$$\mathcal{RM} = \mathcal{T}_{\{1\},\{1\}}$$ Satz I

Beweis durch gegenseitige Simulation

Register-Berechenbarkeit $\hat{=}$ Turing-Berechenbarkeit

$$\mathcal{RM} = \mathcal{T}_{\{1\},\{1\}}$$ Satz I

Beweis durch gegenseitige Simulation

- $\mathcal{RM} \subseteq \mathcal{T}_{\{1\},\{1\}}$
  - Simuliere jedes Register durch separates (einseitiges) Turingband
  - Simuliere Registeroperationen durch Hinzufügen bzw. Löschen von Einsen
  - k-Band-Maschine durch Einbandmaschine simulierbar

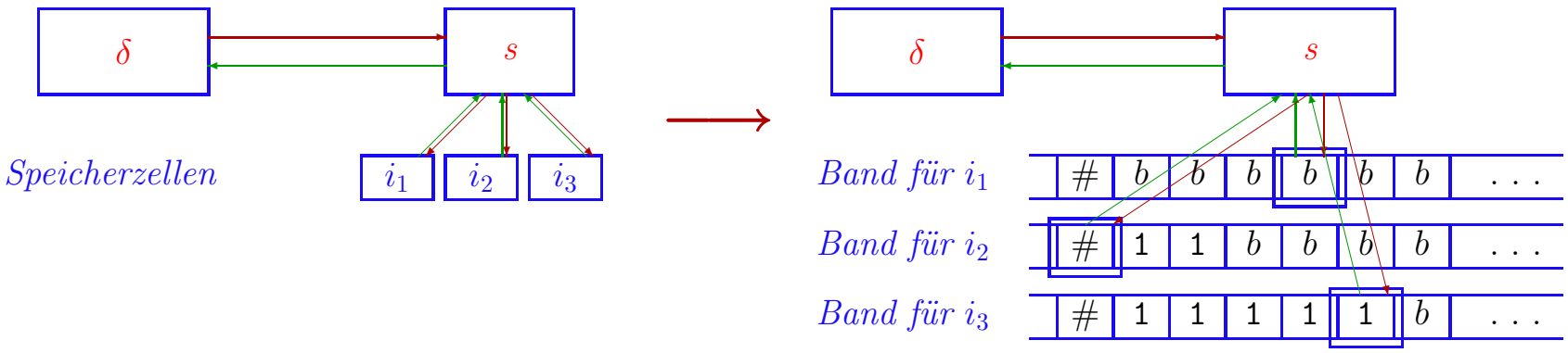
Register-Berechenbarkeit $\hat{=}$ Turing-Berechenbarkeit

$$\mathcal{RM} = \mathcal{T}_{\{1\},\{1\}}$$ Satz I

Beweis durch gegenseitige Simulation

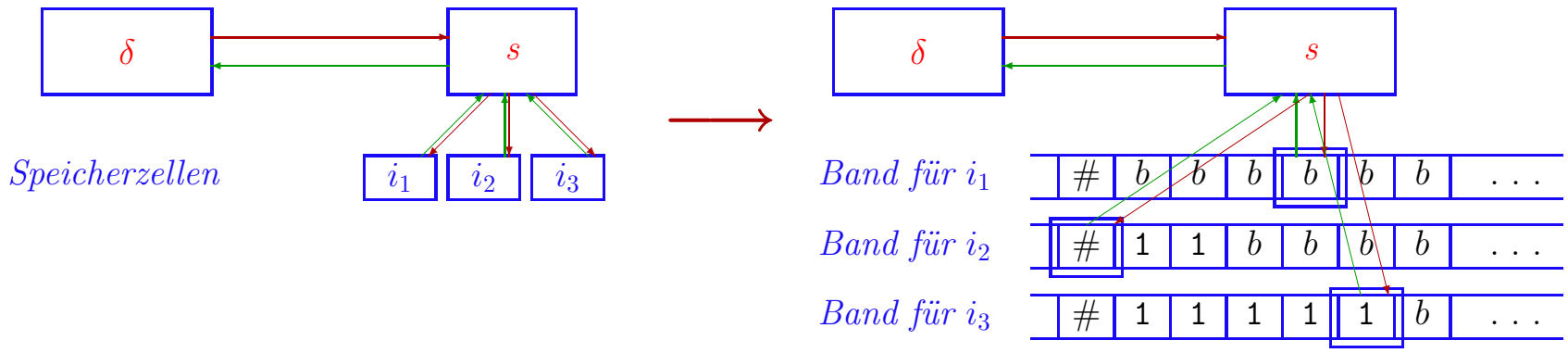
- $\mathcal{RM} \subseteq \mathcal{T}_{\{1\},\{1\}}$
  - Simuliere jedes Register durch separates (einseitiges) Turingband
  - Simuliere Registeroperationen durch Hinzufügen bzw. Löschen von Einsen
  - k-Band-Maschine durch Einbandmaschine simulierbar
- $\mathcal{RM} \supseteq \mathcal{T}_{\{1\},\{1\}}$
  - Direkte Simulation nicht möglich da Anzahl der Register endlich
  - Codiere Bandinhalt und Kopfsymbol als (beliebig große) Zahlen
  - Simuliere Einzelschritte durch entsprechende arithmetische Operationen
  - Umfangreiche Details

Simulation einer RM durch eine TM



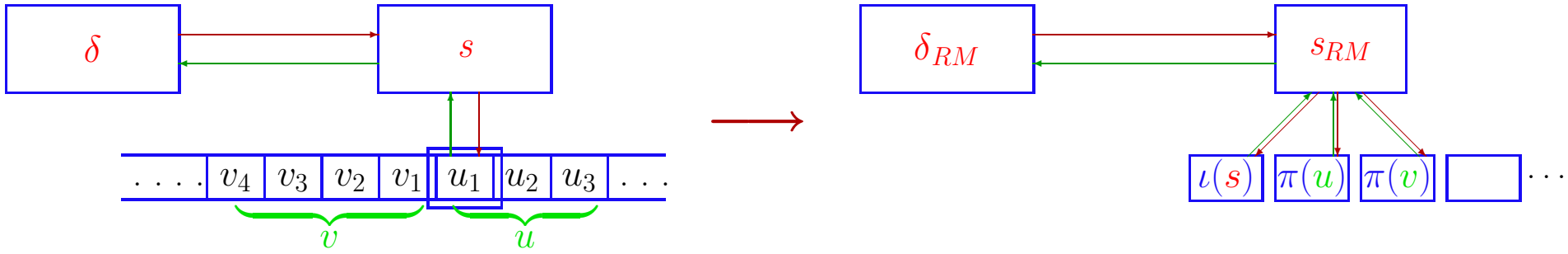
- **Band für Register r wird kellerartig verarbeitet**
  - Kopf am rechten Ende der unären Codierung des Registerinhalts

Simulation einer RM durch eine TM



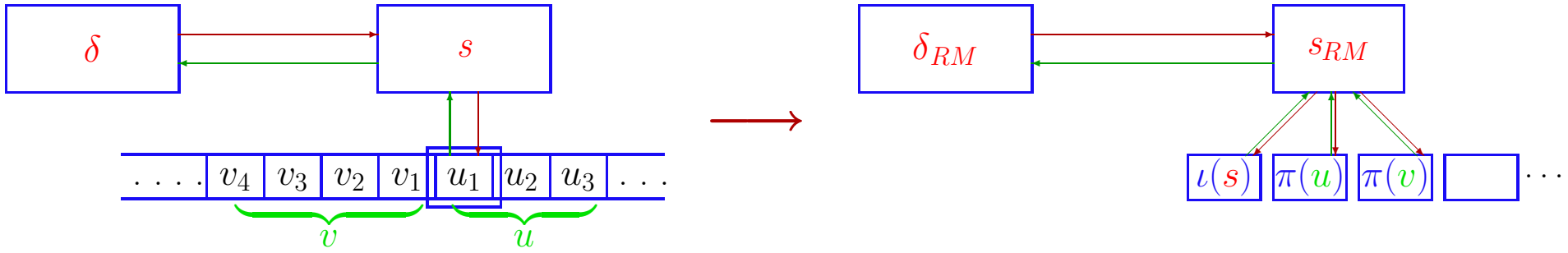
- **Band für Register r wird kellerartig verarbeitet**
  - Kopf am rechten Ende der unären Codierung des Registerinhalts
- **Überführungsfunktion direkt simulierbar**
  - Registerinhaltstest 0: Lesen des Bandanfangsmarkers #
  - Registerinhaltstest 1: Lesen einer 1
  - Registerinhalt vergrößern: nach rechts gehen und eine 1 schreiben
  - Registerinhalt verringern: 1 löschen und nach links gehen (wenn möglich)

Simulation einer TM durch eine RM



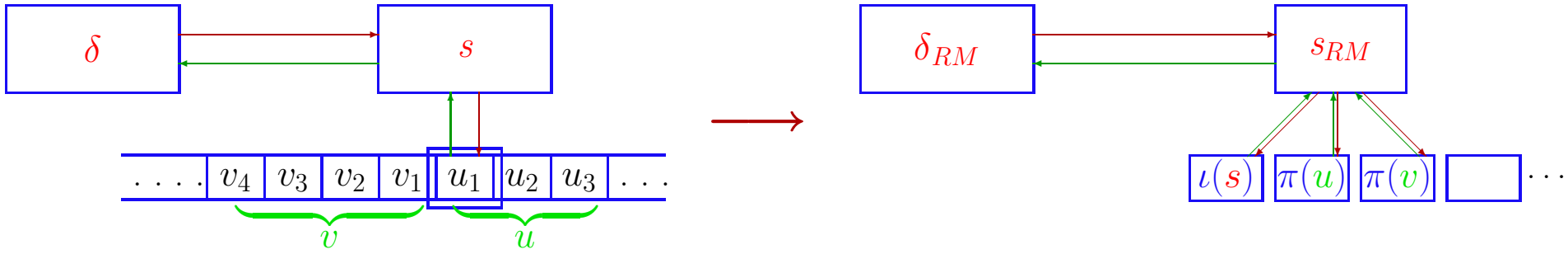
- **Repräsentiere TM-Konfigurationen in Registern**
  - 3 Register für linke Hälfte, rechte Häfte, Zustand
  - Braucht eindeutige Codierung π von Strings als Zahlen

Simulation einer TM durch eine RM



- **Repräsentiere TM-Konfigurationen in Registern**
  - 3 Register für linke Hälfte, rechte Häfte, Zustand
  - Braucht eindeutige Codierung π von Strings als Zahlen
- **Repräsentiere Überführungstabelle in Zuständen**
  - Je ein separater Zustand pro Eintrag in der Tabelle
  - Unterprogramm schreibt (codiertes) Ergebnis $\delta(s,a)=(s',a',P)$ in Register

Simulation einer TM durch eine RM



- **Repräsentiere TM-Konfigurationen in Registern**
  - 3 Register für linke Hälfte, rechte Häfte, Zustand
  - Braucht eindeutige Codierung π von Strings als Zahlen
- **Repräsentiere Überführungstabelle in Zuständen**
  - Je ein separater Zustand pro Eintrag in der Tabelle
  - Unterprogramm schreibt (codiertes) Ergebnis $\delta(s,a)=(s',a',P)$ in Register
- **Simuliere Ausführung der Turingmaschine**
  - Erzeuge Codierung der TM-Anfangskonfiguration aus RM-Eingabe
  - Simuliere Berechnung der TM-Nachfolgekonfiguration
  - Decodiere TM-Endkonfiguration in Ausgabe der Registermaschine

- **Primzahlcodierung π von Strings über Γ** Definition J
  - Für $\Gamma = \{a_1, \ldots, a_m\}$ sei $\iota(a_i) = i$
  - Für $w = w_1 \ldots w_r$ sei $\pi(w) = \Pi_{j=1}^{r} p_j^{\iota(w_j)}$ $(p_j = \texttt{nth-prime}(j))$

    z.B. $\pi(\epsilon) = 1, \quad \pi(a_4 a_2 a_1) = 2^4 \cdot 3^2 \cdot 5^1 = 720$
  - π injektiv wegen Eindeutigkeit der Primzahlzerlegung

- **Primzahlcodierung π von Strings über Γ** Definition J
  - Für $\Gamma = \{a_1,\ldots,a_m\}$ sei $\iota(a_i)=i$
  - Für $w = w_1 \ldots w_r$ sei $\pi(w) = \Pi_{j=1}^{r}\, p_j^{\iota(w_j)}$ $\quad (p_j = \texttt{nth-prime}(j))$

    z.B. $\pi(\epsilon) = 1, \quad \pi(a_4 a_2 a_1) = 2^4 \cdot 3^2 \cdot 5^1 = 720$
  - π injektiv wegen Eindeutigkeit der Primzahlzerlegung

- **Repräsentation von TM-Konfigurationen**
  - Für $S = \{s_0,\ldots,s_k\}$ sei $\iota(s_i)=i$
  - Verwende alternative Repräsentation (s,u,v) von Konfigurationen
  - Codierung durch Registerinhalt $(\mathtt{r}_1,\mathtt{r}_2,\mathtt{r}_3,\ldots) = (\iota(s),\pi(u),\pi(v),\ldots)$

- **Primzahlcodierung π von Strings über Γ** Definition J
  - Für $\Gamma = \{a_1,\ldots,a_m\}$ sei $\iota(a_i)=i$
  - Für $w = w_1 \ldots w_r$ sei $\pi(w) = \Pi_{j=1}^{r}\, p_j^{\iota(w_j)}$ $\quad (p_j = \texttt{nth-prime}(j))$

    z.B. $\pi(\epsilon) = 1, \quad \pi(a_4a_2a_1) = 2^4 \cdot 3^2 \cdot 5^1 = 720$
  - π injektiv wegen Eindeutigkeit der Primzahlzerlegung

- **Repräsentation von TM-Konfigurationen**
  - Für $S = \{s_0,\ldots,s_k\}$ sei $\iota(s_i)=i$
  - Verwende alternative Repräsentation (s,u,v) von Konfigurationen
  - Codierung durch Registerinhalt $(\mathtt{r}_1,\mathtt{r}_2,\mathtt{r}_3,\ldots) = (\iota(s),\pi(u),\pi(v),\ldots)$

- **Repräsentation der Überführungstabelle**
  - Für $S=\{s_0,\ldots,s_k\}$, $\Gamma=\{a_1,\ldots,a_m\}$ verwende Zustände $\bar{s}_1,\ldots,\bar{s}_{(k+1)*m}$
  - Zustand $\bar{s}_{j*m+i}$ enthält Unterprogramm für $\delta(s_j,a_i)=(s_{j'},a_{i'},P)$

    $\mathtt{r}_1\texttt{:=}j'\texttt{;}\ \ \mathtt{r}_4\texttt{:=}i'\texttt{;}\ \ \mathtt{r}_5\texttt{:=}\iota(P)$ $\qquad (\iota(l)=0,\ \iota(h)=1,\ \iota(r)=2)$
  - Register $\mathtt{r}_6$, $\mathtt{r}_7$, ... werden für Zwischenergebnisse der Simulation benutzt

Simulation der TM Ausführung im Detail

- **Erzeugung der Anfangskonfiguration**
  - Anfangskonfiguration der RM ist $\alpha(n) = (s_0,(n,0,...,0))$
  - Anfangskonfiguration der TM ist $(s_0,\epsilon,\underbrace{1...1}_{n})$
  - Der Einfachheit halber hat rechter String mindestens ein (Kopf-)symbol

    `if` r_1`=0 then` r_3`:=`$2^{\iota(b)}$ `else` r_3`:=`$\Pi_{j=1}^{r_1}$ `nth-prime(`j`)`$^{\iota(1)}$ `fi;`

    r_1`:=0;` r_2`:=1`

Simulation der TM Ausführung im Detail

- **Erzeugung der Anfangskonfiguration**
  - Anfangskonfiguration der RM ist $\alpha(n) = (s_0,(n,0,...,0))$
  - Anfangskonfiguration der TM ist $(s_0,\epsilon,\underbrace{1...1}_{n})$
  - Der Einfachheit halber hat rechter String mindestens ein (Kopf-)symbol

    `if r`$_1$`=0 then r`$_3$`:=`$2^{\iota(b)}$ `else r`$_3$`:=`$\Pi_{j=1}^{r_1}$ `nth-prime(`j`)`$^{\iota(1)}$ `fi;`

    `r`$_1$`:=0; r`$_2$`:=1`
- **Lesen des Symbols unter dem Kopf**
  - Codierung des Strings $v_1...v_r$ in $\mathtt{r}_3$ ist $2^{\iota(v_1)}*\Pi_{j=2}^{r} p_j^{\iota(v_j)}$
  - Berechne $\iota(v_1)$ und speichere nach $\mathtt{r}_4$, "lösche" v_1 aus $\mathtt{r}_3$

    `r`$_4$`:=prim-exp(r`$_3$`,2); r`$_3$`:= r`$_3$ `div 2`$^{\mathtt{r}_4}$

Simulation der TM Ausführung im Detail

- **Erzeugung der Anfangskonfiguration**
  - Anfangskonfiguration der RM ist $\alpha(n) = (s_0,(n,0,...,0))$
  - Anfangskonfiguration der TM ist $(s_0,\epsilon,\underbrace{1...1}_{n})$
  - Der Einfachheit halber hat rechter String mindestens ein (Kopf-)symbol

    `if r1=0 then r3:=2^ι(b) else r3:=Π_{j=1}^{r1} nth-prime(j)^ι(1) fi;`

    `r1:=0; r2:=1`
- **Lesen des Symbols unter dem Kopf**
  - Codierung des Strings $v_1...v_r$ in $\mathtt{r}_3$ ist $2^{\iota(v_1)}*\Pi_{j=2}^{r}\ p_j^{\iota(v_j)}$
  - Berechne $\iota(v_1)$ und speichere nach $\mathtt{r}_4$, "lösche" v_1 aus $\mathtt{r}_3$

    `r4:=prim-exp(r3,2); r3:=r3 div 2^r4`
- **Bestimmung des auszuführenden Befehls**
  - Durch schrittweises Herunterzählen gehe zum Zustand $\bar{s}_{\mathtt{r}_1*m+\mathtt{r}_4}$
  - Führe entsprechendes Unterprogramm aus und gehe zum Zustand $\bar{s}$
  - $\mathtt{r}_1$ enthät Folgezustand, $\mathtt{r}_4$ neues Kopfsymbol, $\mathtt{r}_5$ Kopfbewegung

Simuliere Schreiben ($\mathtt{r}_4$) und Kopfbewegung ($\mathtt{r}_5$)

Kopfbewegung erfordert Umcodierung der Strings

Simuliere Schreiben (r_4) und Kopfbewegung (r_5)

Kopfbewegung erfordert Umcodierung der Strings

`if` r_5`=1 then` r_3`:=` r_3 `*` 2^{r_4}`;` `goto` s_e `fi`

$P = h$

Simuliere Schreiben (r_4) und Kopfbewegung (r_5)

Kopfbewegung erfordert Umcodierung der Strings

```
if r5=1 then r3:=r3 * 2^r4; goto se fi                    P = h
if r5=2 then                                              P = r
   r6:=r3; r3:=1; r7:=2; r8:=2          r7=n, r8 = (n-1)te Primzahl
   while r6>1 do          Umverschlüsseln r3: von nte auf (n-1)te Primzahl)
      r9:= nth-prime(r7); r10:=prim-exp(r6,r9)
      r6:=r6 div r9^r10; r3:=r3*r8^r10; r8:=r9; r7:=r7+1
   od
```

Simuliere Schreiben (r_4) und Kopfbewegung (r_5)

Kopfbewegung erfordert Umcodierung der Strings

```
if r5=1 then r3:=r3 * 2^r4; goto se fi                    P = h
if r5=2 then                                              P = r
   r6:=r3; r3:=1; r7:=2; r8:=2          r7=n, r8 = (n-1)te Primzahl
   while r6>1 do          Umverschlüsseln r3: von nte auf (n-1)te Primzahl)
      r9:= nth-prime(r7); r10:=prim-exp(r6,r9)
      r6:=r6 div r9^r10; r3:=r3*r8^r10; r8:=r9; r7:=r7+1
   od
   if r3=1 then r3:=2^ι(b) fi     Sonderfall Bandende: rechts ein b anhängen
```

Simuliere Schreiben (r_4) und Kopfbewegung (r_5)

Kopfbewegung erfordert Umcodierung der Strings

```
if r5=1 then r3:=r3 * 2^r4; goto se fi                                   P = h
if r5=2 then                                                             P = r
   r6:=r3; r3:=1; r7:=2; r8:=2                  r7=n, r8 = (n-1)te Primzahl
   while r6>1 do                  Umverschlüsseln r3: von nte auf (n-1)te Primzahl)
      r9:= nth-prime(r7); r10:=prim-exp(r6,r9)
      r6:=r6 div r9^r10; r3:=r3*r8^r10; r8:=r9; r7:=r7+1
   od
   if r3=1 then r3:=2^ι(b) fi     Sonderfall Bandende: rechts ein b anhängen
      r6:=r2; r2:=2^r4; r7:=2; r8:=2
   while r6>1 do                  Umverschlüsseln r2: von (n-1)te auf nte Primzahl)
      r9:= nth-prime(r7); r10:=prim-exp(r6,r8)
      r6:=r6 div r8^r10; r2:=r2*r9^r10; r8:=r9; r7:=r7+1
   od
fi
```

Simuliere Schreiben (r_4) und Kopfbewegung (r_5)

Kopfbewegung erfordert Umcodierung der Strings

```
if r5=1 then r3:=r3 * 2^r4; goto se fi                                  P = h
if r5=2 then                                                            P = r
   r6:=r3; r3:=1; r7:=2; r8:=2                  r7=n, r8 = (n-1)te Primzahl
   while r6>1 do                  Umverschlüsseln r3: von nte auf (n-1)te Primzahl)
      r9:= nth-prime(r7); r10:=prim-exp(r6,r9)
      r6:=r6 div r9^r10; r3:=r3*r8^r10; r8:=r9; r7:=r7+1
   od
   if r3=1 then r3:=2^ι(b) fi     Sonderfall Bandende: rechts ein b anhängen
      r6:=r2; r2:=2^r4; r7:=2; r8:=2
   while r6>1 do                  Umverschlüsseln r2: von (n-1)te auf nte Primzahl)
      r9:= nth-prime(r7); r10:=prim-exp(r6,r8)
      r6:=r6 div r8^r10; r2:=r2*r9^r10; r8:=r9; r7:=r7+1
   od
fi
if r5=0 then ...                                                        P = l, analog
```

Simulation der TM Ausführung: Ausgabe

- **Decodierung der Ausgabekonfiguration**
  - Endkonfiguration der TM ist (s,u,v) mit $u{=}1^k b^*$, $v{=}1^j b^*$
  - Endkonfiguration der RM muß $k+j$ in Register $\mathtt{r}_1$ enthalten
  - Schrittweise decodiere Primexponenten bis zum ersten Leerzeichen

Simulation der TM Ausführung: Ausgabe

- **Decodierung der Ausgabekonfiguration**
  - Endkonfiguration der TM ist (s,u,v) mit $u{=}1^k b^*$, $v{=}1^j b^*$
  - Endkonfiguration der RM muß $k+j$ in Register $\mathtt{r}_1$ enthalten
  - Schrittweise decodiere Primexponenten bis zum ersten Leerzeichen

```
r1:=0; r4:=1; r5:=nth-prime(r4); r6:=prim-exp(r2,r5)
while r6=ι(1) do
   r2:= r2 div r5^r6; r1:=r1+1;
   r4:=r4+1; r5:=nth-prime(r4); r6:=prim-exp(r2,r5)
od;
r4:=1; r5:=nth-prime(r4); r6:=prim-exp(r2,r5)
while r6=ι(1) do
   r3:= r3 div r5^r6; r1:=r1+1;
   r4:=r4+1; r5:=nth-prime(r4); r6:=prim-exp(r2,r5)
od
```

Simulation der TM Ausführung: Ausgabe

- **Decodierung der Ausgabekonfiguration**
  - Endkonfiguration der TM ist (s,u,v) mit $u{=}1^k b^*$, $v{=}1^j b^*$
  - Endkonfiguration der RM muß $k+j$ in Register $\mathtt{r}_1$ enthalten
  - Schrittweise decodiere Primexponenten bis zum ersten Leerzeichen

```
r1:=0; r4:=1; r5:=nth-prime(r4); r6:=prim-exp(r2,r5)
while r6=ι(1) do
   r2:= r2 div r5^r6; r1:=r1+1;
   r4:=r4+1; r5:=nth-prime(r4); r6:=prim-exp(r2,r5)
od;
r4:=1; r5:=nth-prime(r4); r6:=prim-exp(r2,r5)
while r6=ι(1) do
   r3:= r3 div r5^r6; r1:=r1+1;
   r4:=r4+1; r5:=nth-prime(r4); r6:=prim-exp(r2,r5)
od
```

q.e.d