

# Theoretische Informatik



## Einheit 2

### Berechenbarkeitsmodelle



1. Turingmaschinen
2. Registermaschinen
3.  $\mu$ -rekursive Funktionen
4. Weitere Berechenbarkeitsmodelle
5. Church'sche These

# BERECHENBARKEITSMODELLE – WOZU?

- **Es gibt mehr als nur die Standard PC Architektur**
  - Lisp Maschinen, Parallelrechner, Neuronale Netze
  - Nichtdeterministische Maschinen (Quantencomputer)

# BERECHENBARKEITSMODELLE – WOZU?

- **Es gibt mehr als nur die Standard PC Architektur**
  - Lisp Maschinen, Parallelrechner, Neuronale Netze
  - Nichtdeterministische Maschinen (Quantencomputer)
- **Abstrakte Modelle betrachten die wirklichen Fragen zuerst**
  - Was genau ist das Problem?
  - Was charakterisiert eine Lösung des Problems?
  - Wie kann man prinzipiell an das Problem herangehen?
  - Wie kann man über den Stand der Technik hinausgehen?

# BERECHENBARKEITSMODELLE – WOZU?

- **Es gibt mehr als nur die Standard PC Architektur**
  - Lisp Maschinen, Parallelrechner, Neuronale Netze
  - Nichtdeterministische Maschinen (Quantencomputer)
- **Abstrakte Modelle betrachten die wirklichen Fragen zuerst**
  - Was genau ist das Problem?
  - Was charakterisiert eine Lösung des Problems?
  - Wie kann man prinzipiell an das Problem herangehen?
  - Wie kann man über den Stand der Technik hinausgehen?
- **Berechenbarkeitsmodelle klären fundamentale Fragen**
  - Was ist überhaupt Berechenbarkeit?
  - Auf welche Arten kann man Berechnungen durchführen?
  - Sind bestimmte Berechnungsmodelle besser als andere?

# BERECHENBARKEITSMODELLE – WOZU?

- **Es gibt mehr als nur die Standard PC Architektur**
  - Lisp Maschinen, Parallelrechner, Neuronale Netze
  - Nichtdeterministische Maschinen (Quantencomputer)
- **Abstrakte Modelle betrachten die wirklichen Fragen zuerst**
  - Was genau ist das Problem?
  - Was charakterisiert eine Lösung des Problems?
  - Wie kann man prinzipiell an das Problem herangehen?
  - Wie kann man über den Stand der Technik hinausgehen?
- **Berechenbarkeitsmodelle klären fundamentale Fragen**
  - Was ist überhaupt Berechenbarkeit?
  - Auf welche Arten kann man Berechnungen durchführen?
  - Sind bestimmte Berechnungsmodelle besser als andere?

**Berechenbarkeitsmodelle gab es lange vor dem ersten Computer**

# DIE WICHTIGSTEN BERECHENBARKEITSMODELLE

- **Turingmaschine** (Rechnen mit Papier und Bleistift)
- **Abakus** (Das älteste mechanische Hilfsmittel)
- **Registermaschine** (Assembler / Maschinenprogrammierung)
- **PASCAL-reduziert** (Imperative höhere Sprachen)
- **Nichtdeterministische Turingmaschine** (Parallelismus/Quantenrechner)
- **$\mu$ -rekursive Funktionen** (Mathematisches Rechnen)
- **$\lambda$ -Kalkül** (Funktionale Sprachen, LISP)
- **Logische Repräsentierbarkeit** (Logikprogrammierung, PROLOG)
- **Typ-0 Grammatiken / Markov-Algorithmen** (Regelbasierte Sprachen)

# DIE WICHTIGSTEN BERECHENBARKEITSMODELLE

- **Turingmaschine** (Rechnen mit Papier und Bleistift)
- **Abakus** (Das älteste mechanische Hilfsmittel)
- **Registermaschine** (Assembler / Maschinenprogrammierung)
- **PASCAL-reduziert** (Imperative höhere Sprachen)
- **Nichtdeterministische Turingmaschine** (Parallelismus/Quantenrechner)
- **$\mu$ -rekursive Funktionen** (Mathematisches Rechnen)
- **$\lambda$ -Kalkül** (Funktionale Sprachen, LISP)
- **Logische Repräsentierbarkeit** (Logikprogrammierung, PROLOG)
- **Typ-0 Grammatiken / Markov-Algorithmen** (Regelbasierte Sprachen)

Alle Modelle führen zu demselben Berechenbarkeitsbegriff

# DIE WICHTIGSTEN BERECHENBARKEITSMODELLE

- **Turingmaschine** (Rechnen mit Papier und Bleistift)
- **Abakus** (Das älteste mechanische Hilfsmittel)
- **Registermaschine** (Assembler / Maschinenprogrammierung)
- **PASCAL-reduziert** (Imperative höhere Sprachen)
- **Nichtdeterministische Turingmaschine** (Parallelismus/Quantenrechner)
- **$\mu$ -rekursive Funktionen** (Mathematisches Rechnen)
- **$\lambda$ -Kalkül** (Funktionale Sprachen, LISP)
- **Logische Repräsentierbarkeit** (Logikprogrammierung, PROLOG)
- **Typ-0 Grammatiken / Markov-Algorithmen** (Regelbasierte Sprachen)

Alle Modelle führen zu demselben Berechenbarkeitsbegriff

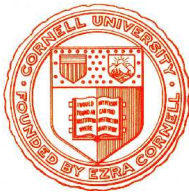


## Church'sche These:

Intuitive Berechenbarkeit wird durch diese Modelle exakt beschrieben



# Theoretische Informatik



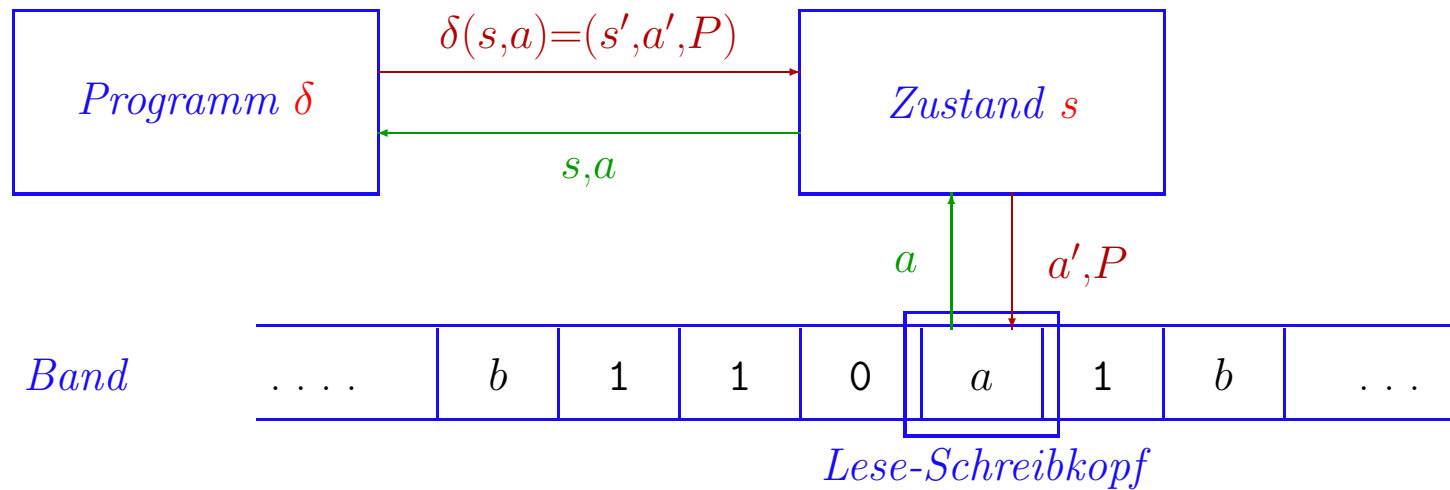
## Einheit 1.1

## Turingmaschinen



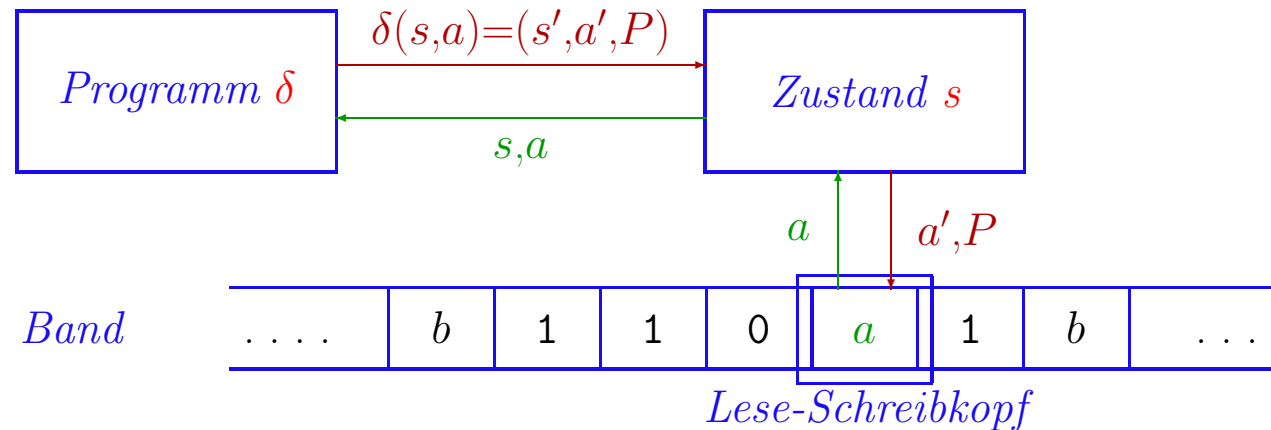
1. Arbeitsweise
2. Formale Semantik
3. Turing-Berechenbarkeit
4. Varianten von Turingmaschinen

# TURINGMASCHINEN



- **Automaten mit potentiell unendlichem Band**
  - Band fast überall unbeschrieben (Leersymbol  $b \equiv$  “Blank”)
  - Lese-Schreibkopf kann Symbole lesen, schreiben und bewegt werden

# TURINGMASCHINEN – MATHEMATISCH



Eine **Turingmaschine** ist ein 6-Tupel  $\tau = (S, X, \Gamma, \delta, s_0, b)$

- $S$  nichtleere endliche Zustandsmenge
- $s_0 \in S$  Anfangszustand
- $\Gamma$  nichtleeres endliches Bandalphabet
- $X \subseteq \Gamma$  Eingabealphabet
- $b \in \Gamma \setminus X$  Blanksymbol
- $\delta: S \times \Gamma \rightarrow S \times \Gamma \times \{r, l, h\}$  (partielle) Zustandsüberföhrungsfunktion

## Übergangstabelle für $\delta$

Zustand	gelesen	Folgezustand	zu schreiben	Kopfbewegung
$s_0$	0	$s_0$	0	r
$s_0$	1	$s_0$	1	r
$s_0$	b	$s_1$	b	l
$s_1$	1	$s_1$	0	l
$s_1$	0	$s_2$	1	l
$s_1$	b	$s_3$	1	h
$s_2$	0	$s_2$	0	l
$s_2$	1	$s_2$	1	l
$s_2$	b	$s_3$	b	h

## Übergangstabelle für $\delta$

Zustand	gelesen	Folgezustand	zu schreiben	Kopfbewegung
$s_0$	0	$s_0$	0	r
$s_0$	1	$s_0$	1	r
$s_0$	b	$s_1$	b	l
$s_1$	1	$s_1$	0	l
$s_1$	0	$s_2$	1	l
$s_1$	b	$s_3$	1	h
$s_2$	0	$s_2$	0	l
$s_2$	1	$s_2$	1	l
$s_2$	b	$s_3$	b	h

## Restliche Komponenten implizit bestimmt

Zustandsmenge  $S = \{s_0, s_1, s_2, s_3\}$

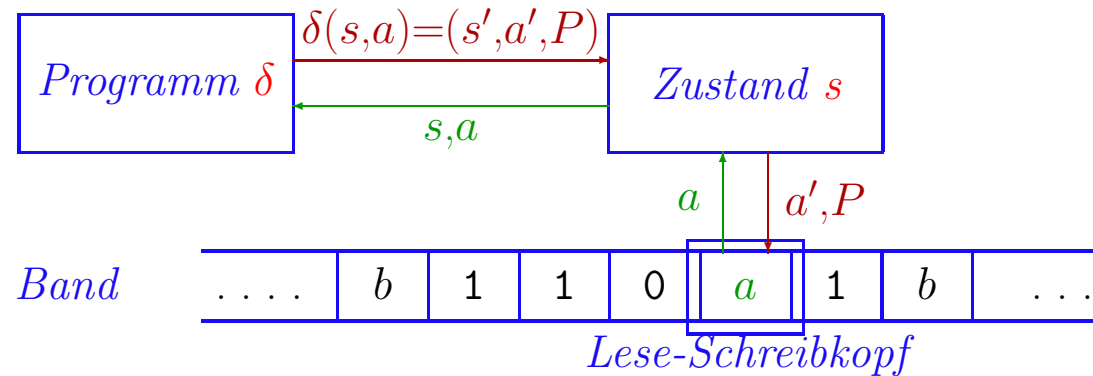
Anfangszustand  $s_0 = s_0$

Bandalphabet  $\Gamma = \{0, 1, b\}$

Eingabealphabet  $X = \{0, 1\}$

Blanksymbol  $b = b$

# ARBEITSWEISE VON TURINGMASCHINEN



## ● Anfangssituation

- Eingabewort  $w$  steht auf dem Band, umgeben von Leerzeichen
- Kopf über erstem Symbol, Zustand ist  $s_0$

## ● Arbeitsschritt

- Zeichen  $a$  lesen, Zustand  $s$  und  $\delta(s,a)=(s',a',P)$  bestimmen
- Neuer Zustand  $s'$ , Zeichen  $a'$  schreiben, Kopf gemäß  $P$  bewegen
- Stop wenn  $P=h$

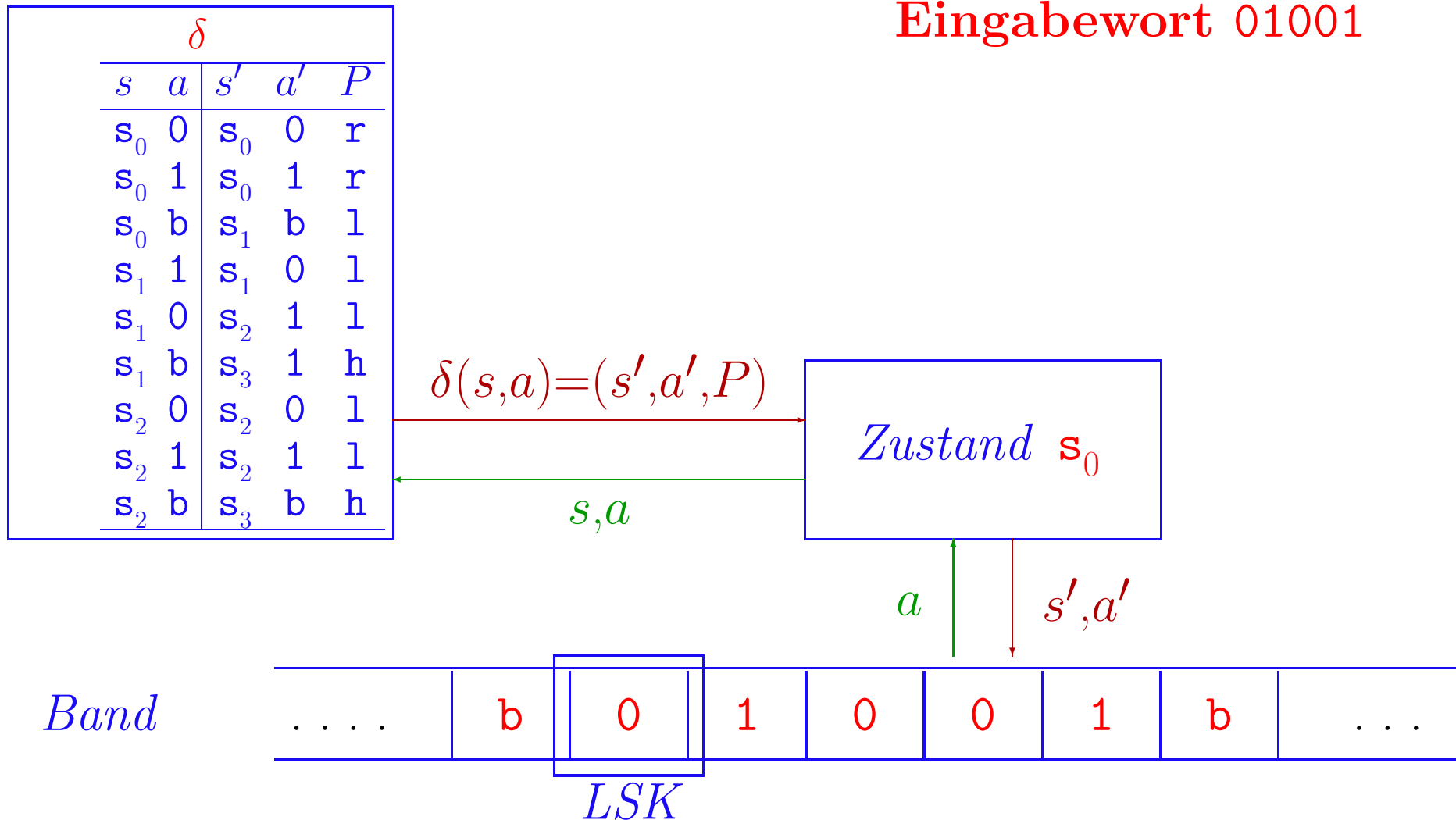
## ● Ergebnis

- Längstes Wort auf Band ohne Leerzeichen am Anfang und Ende

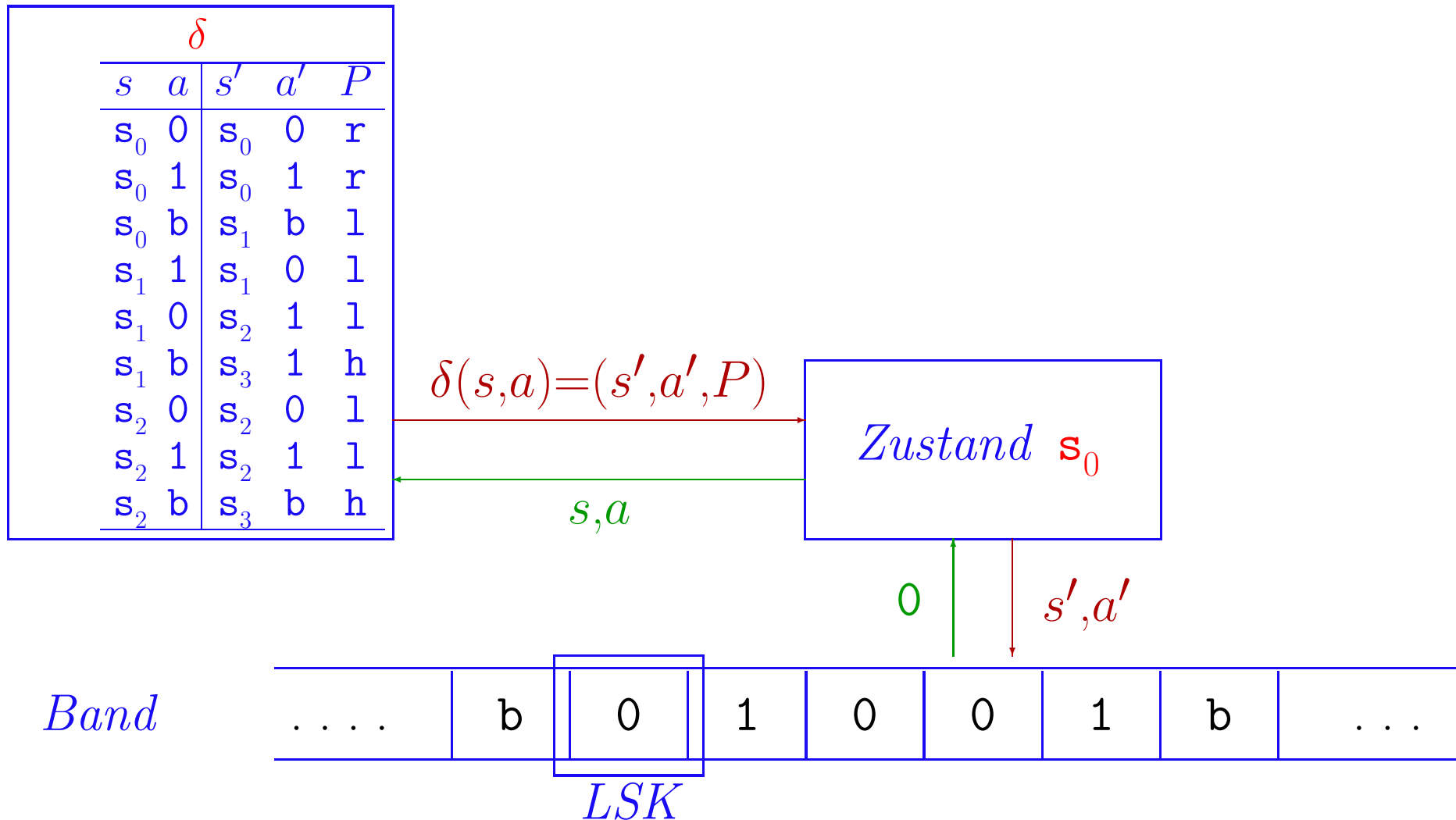
**Achtung! Details in Literatur unterschiedlich**

# ABARBEITUNG VON TURING-PROGRAMMEN

Eingabewort 01001

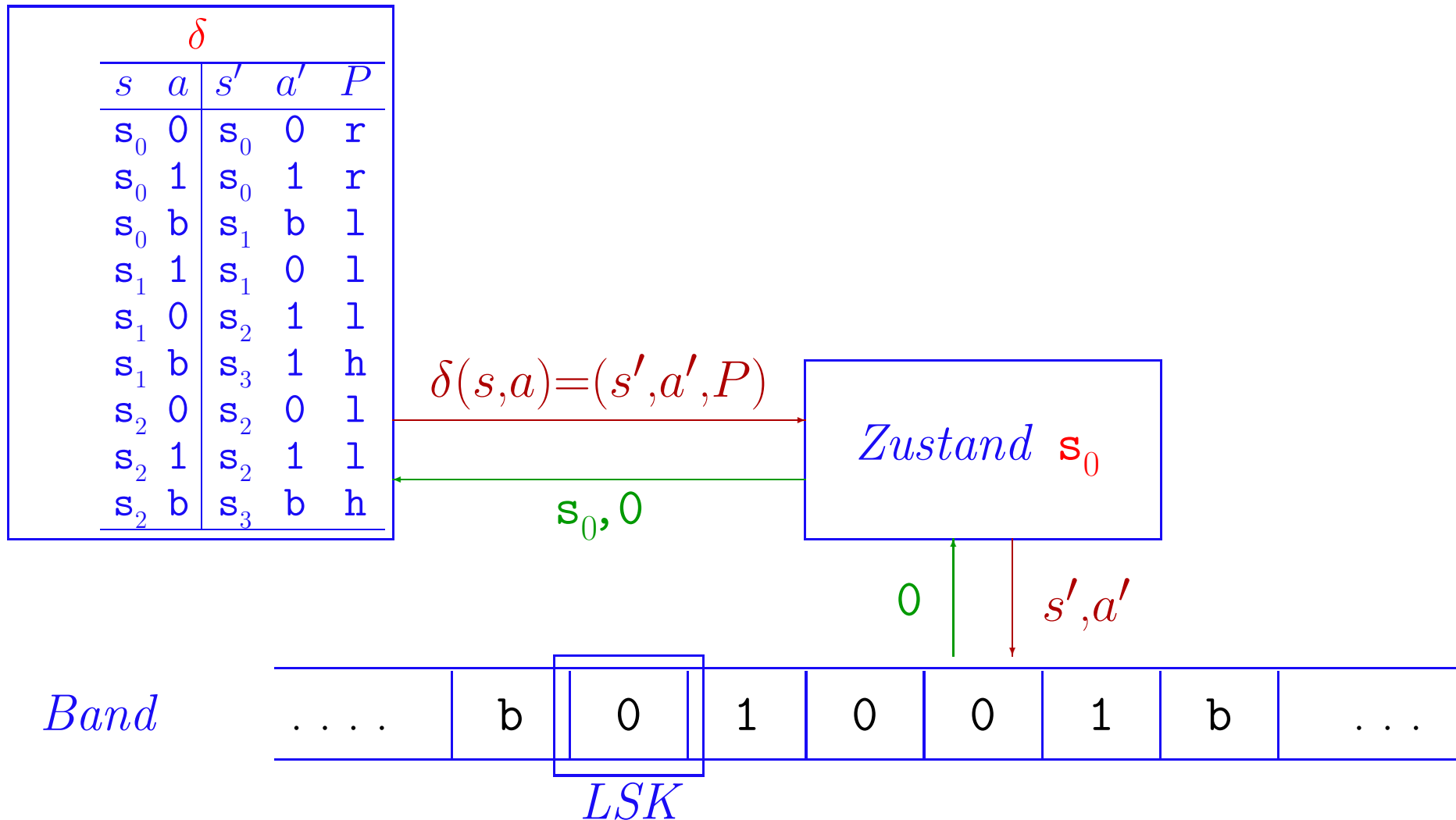


# ABARBEITUNG VON TURING-PROGRAMMEN

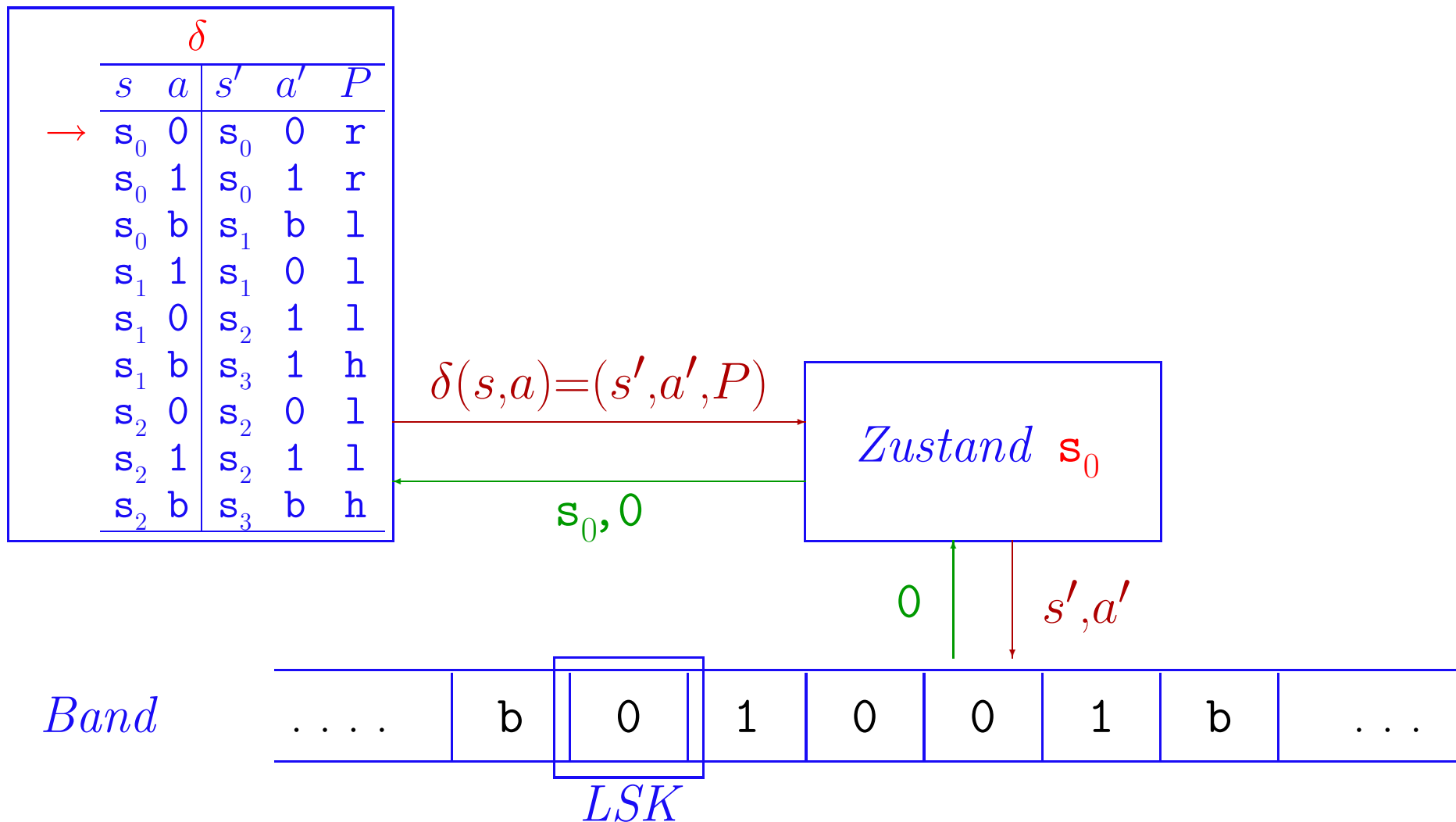




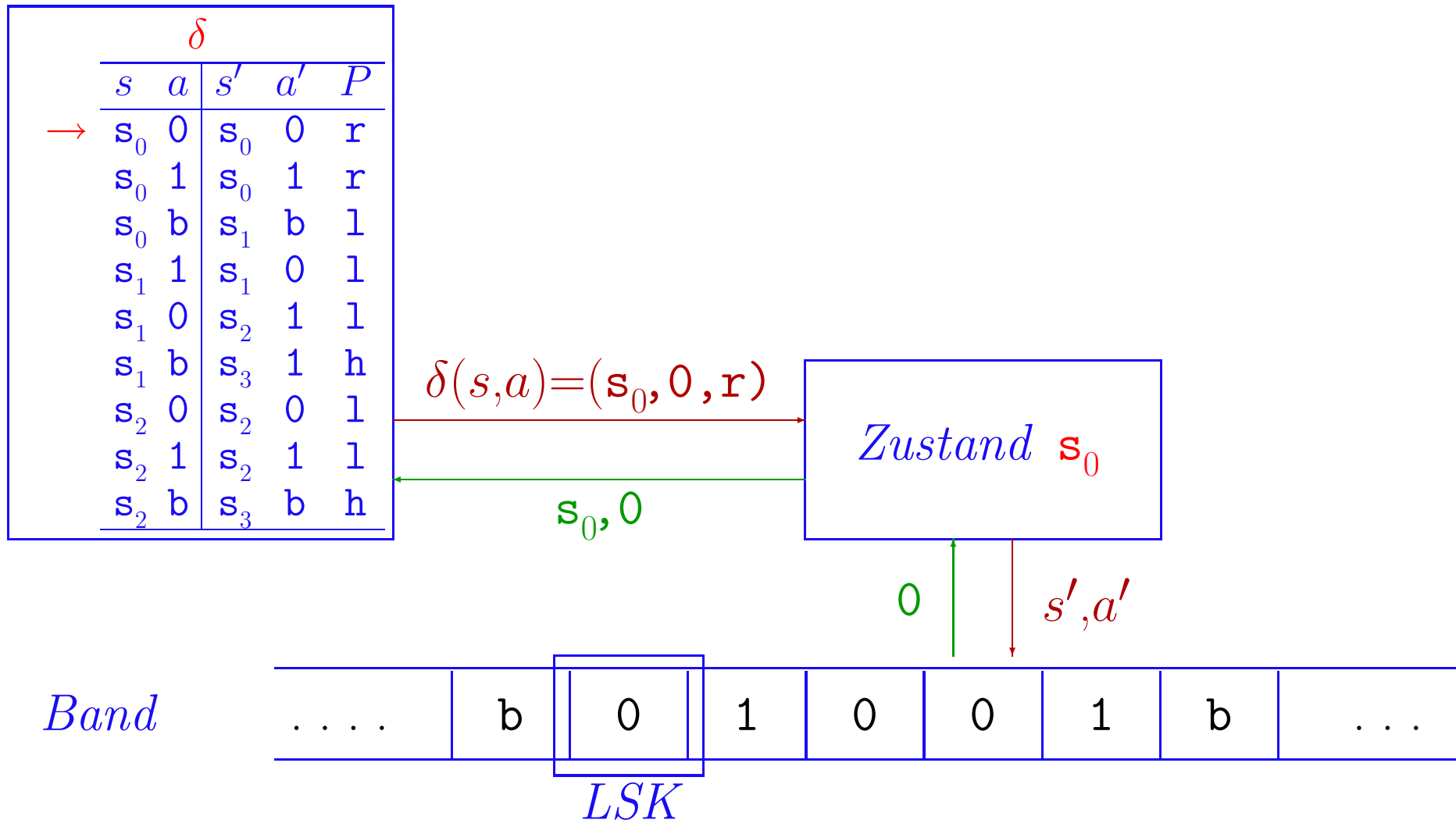
# ABARBEITUNG VON TURING-PROGRAMMEN



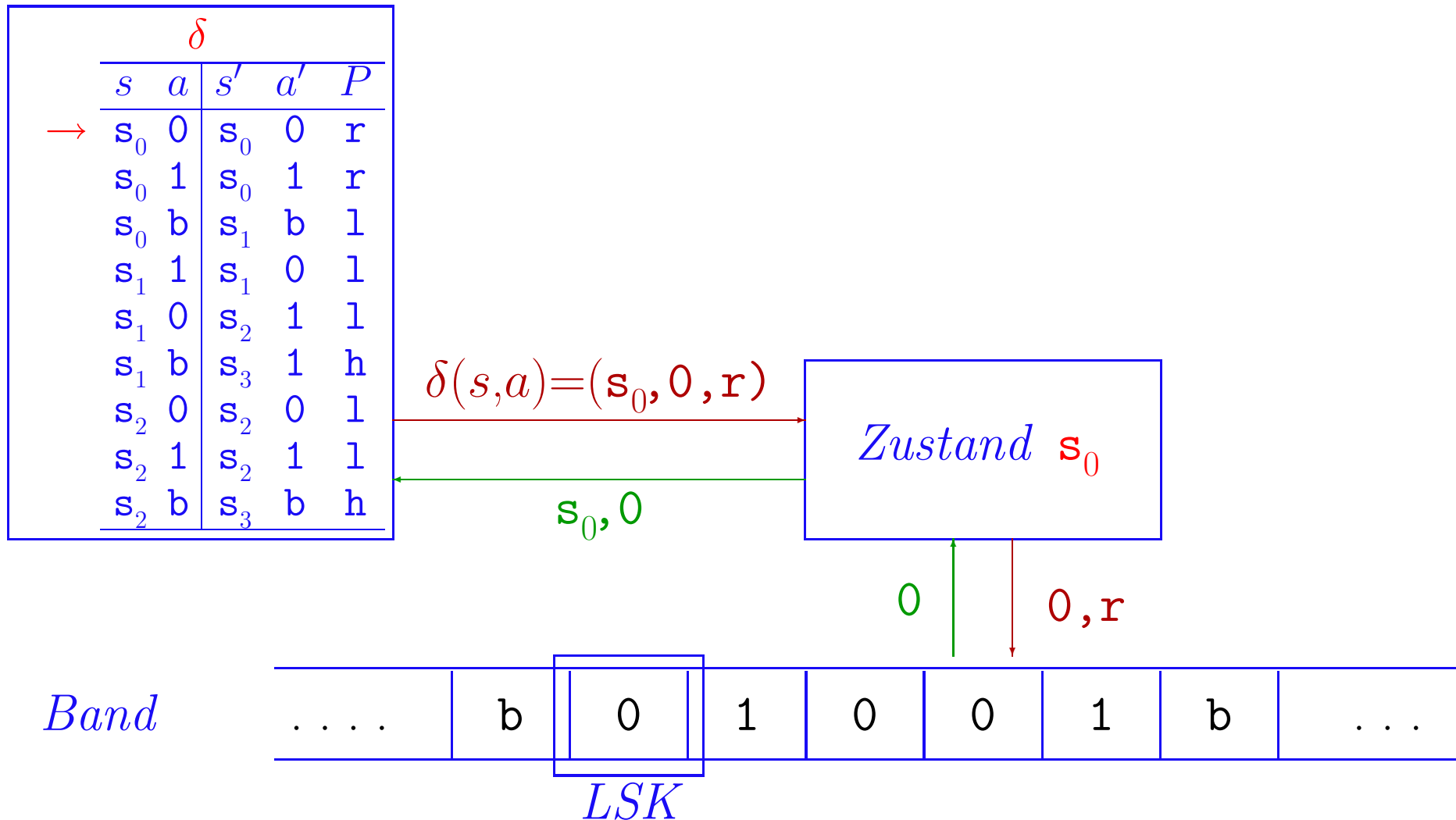
# ABARBEITUNG VON TURING-PROGRAMMEN



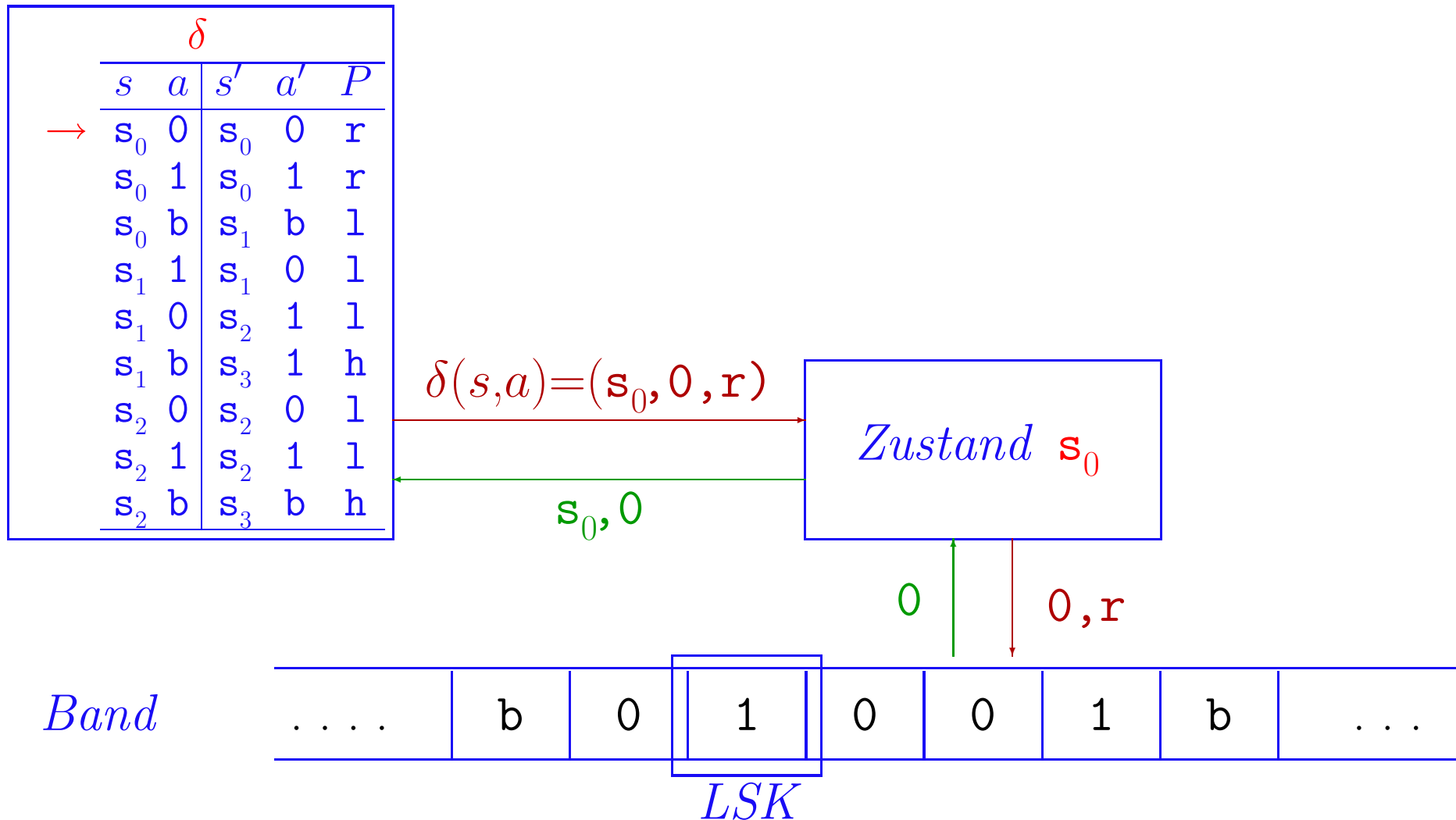
# ABARBEITUNG VON TURING-PROGRAMMEN



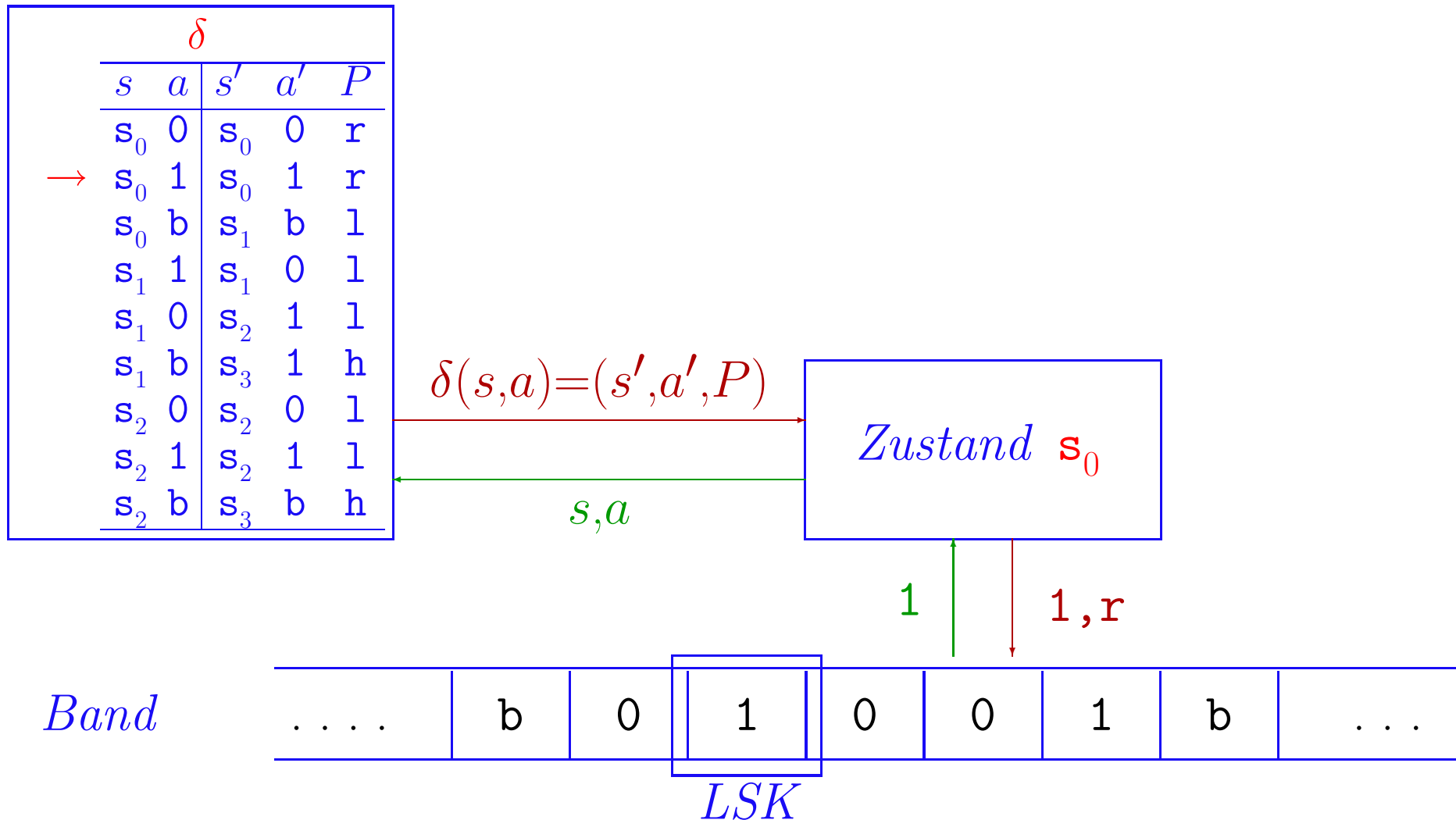
# ABARBEITUNG VON TURING-PROGRAMMEN



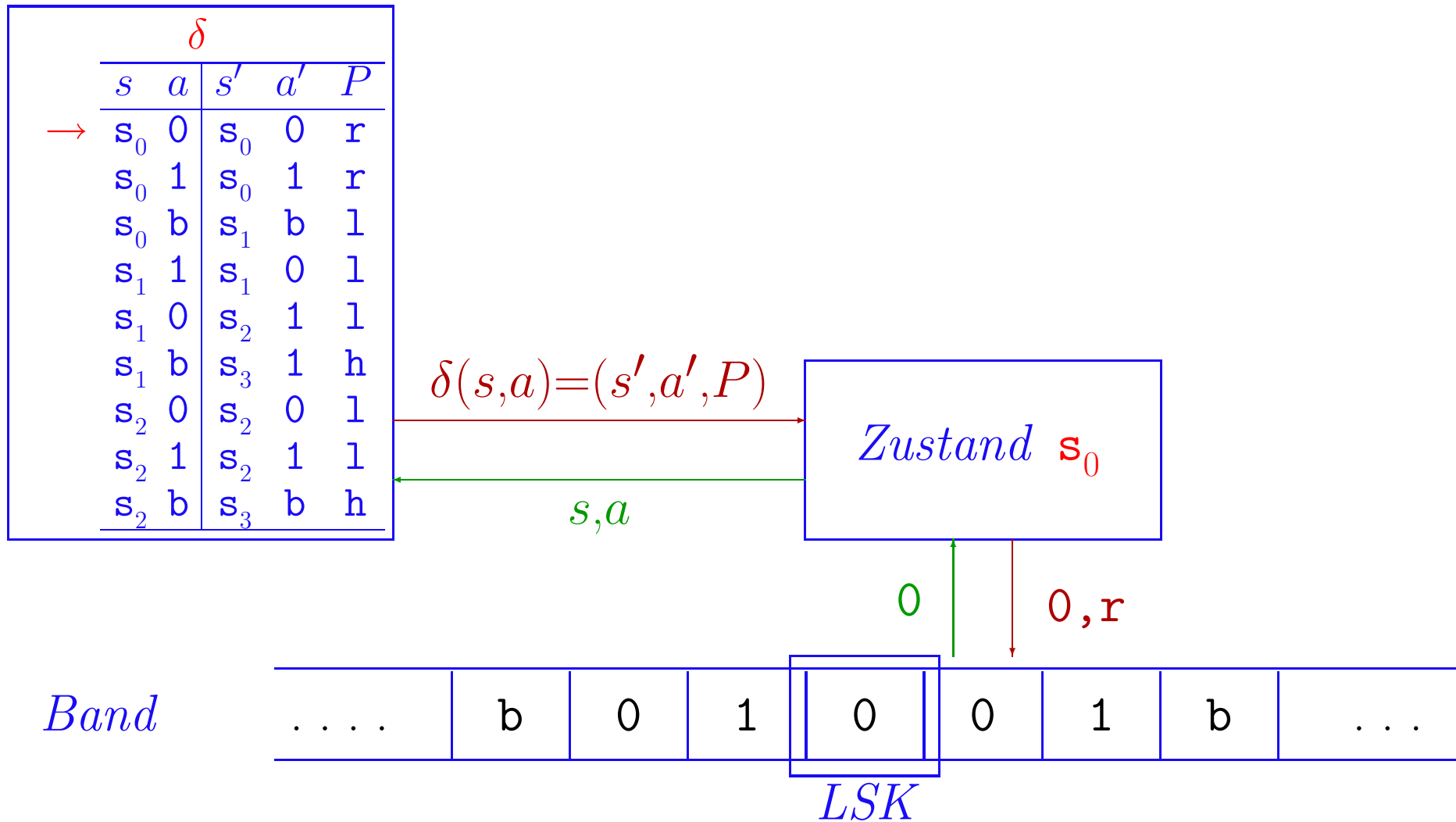
# ABARBEITUNG VON TURING-PROGRAMMEN



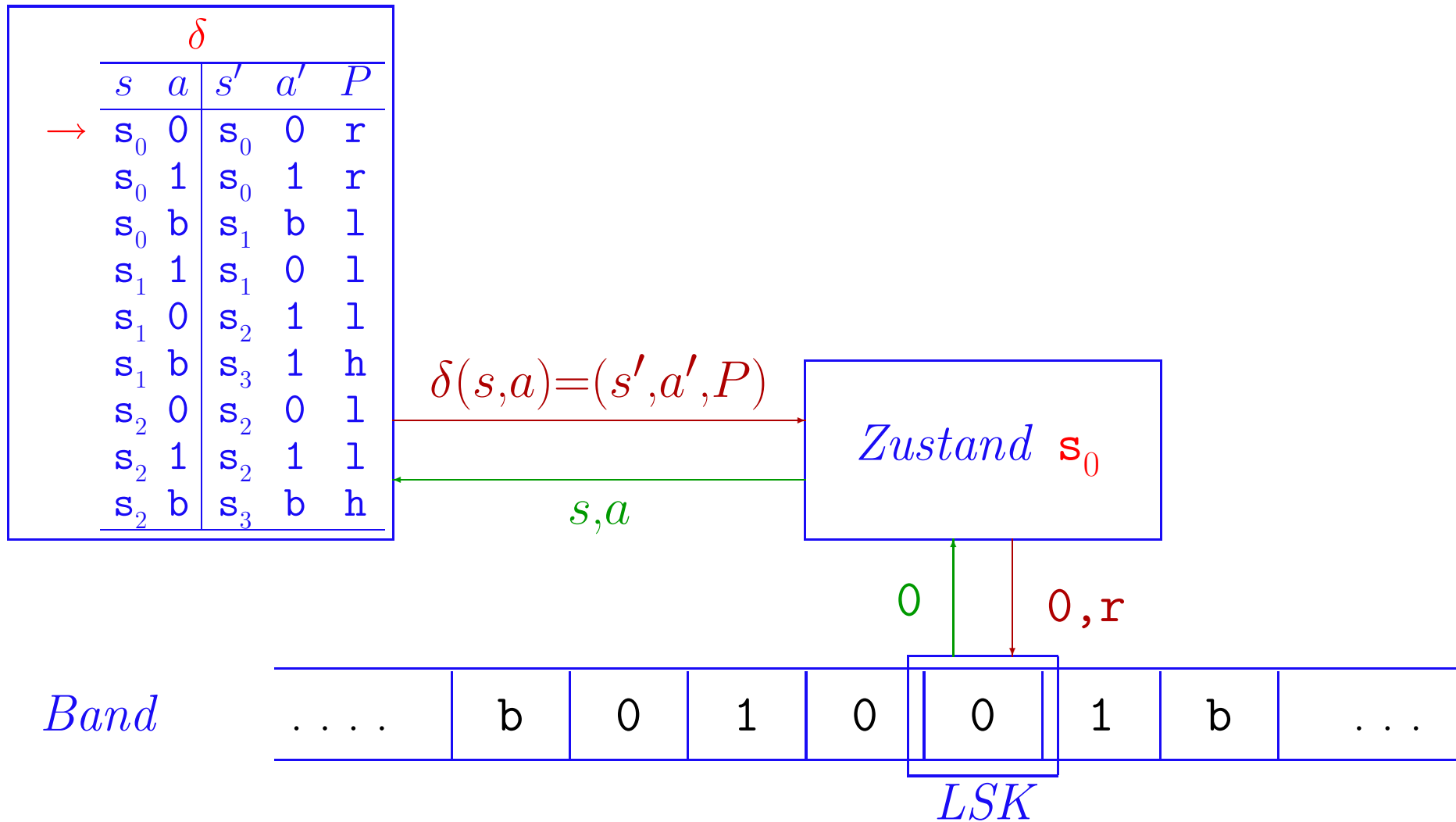
# ABARBEITUNG VON TURING-PROGRAMMEN



# ABARBEITUNG VON TURING-PROGRAMMEN

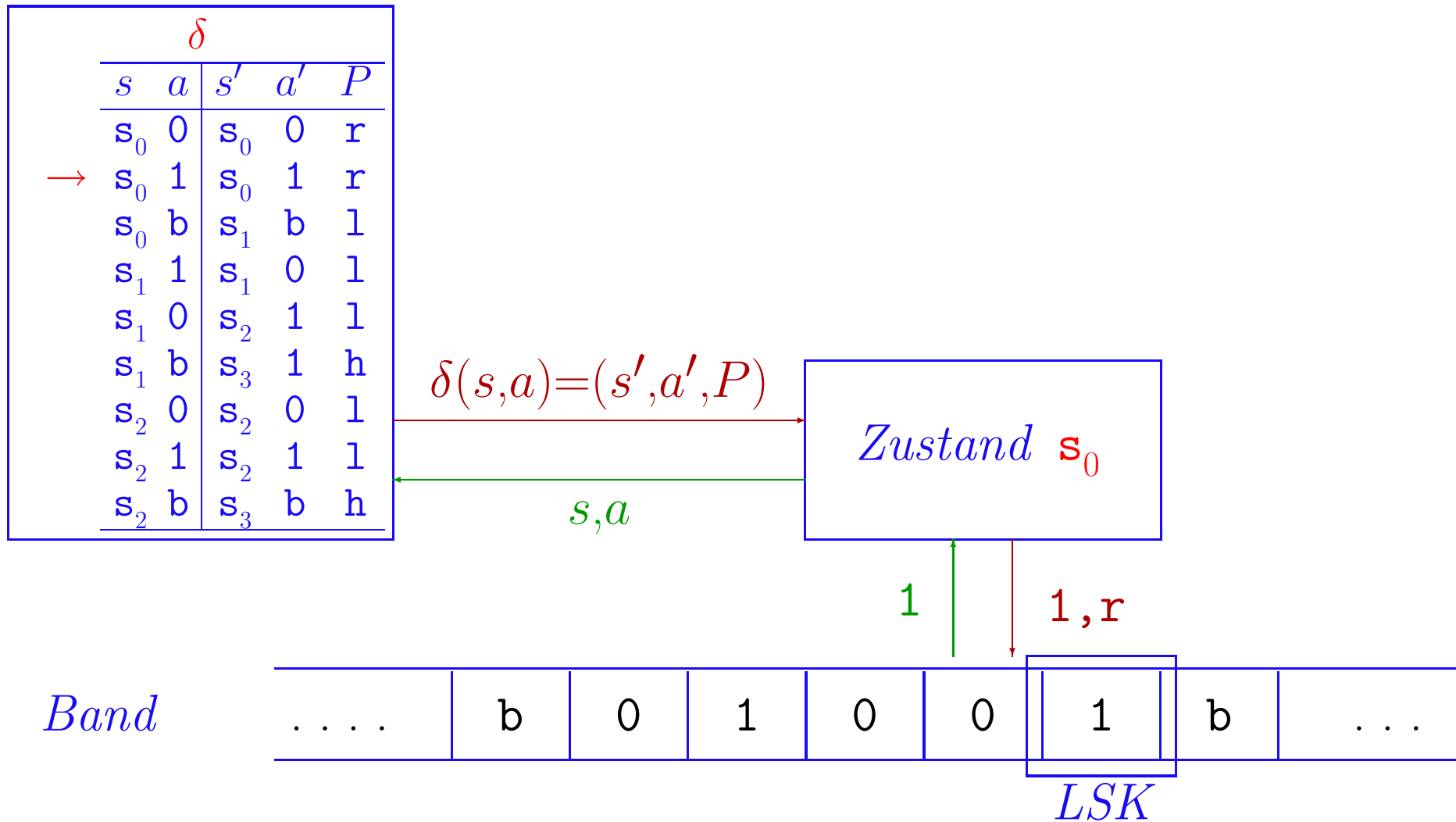


# ABARBEITUNG VON TURING-PROGRAMMEN

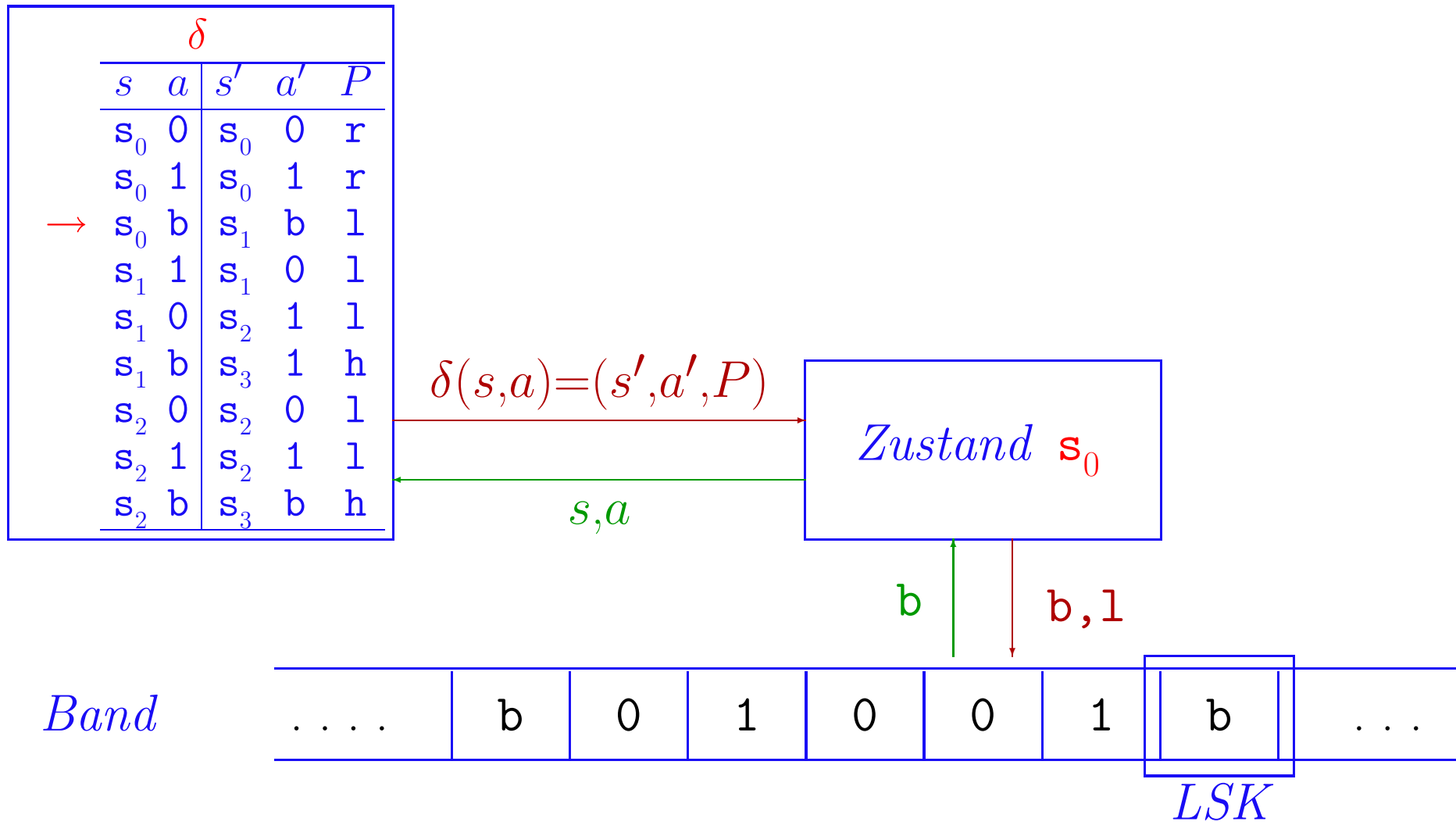




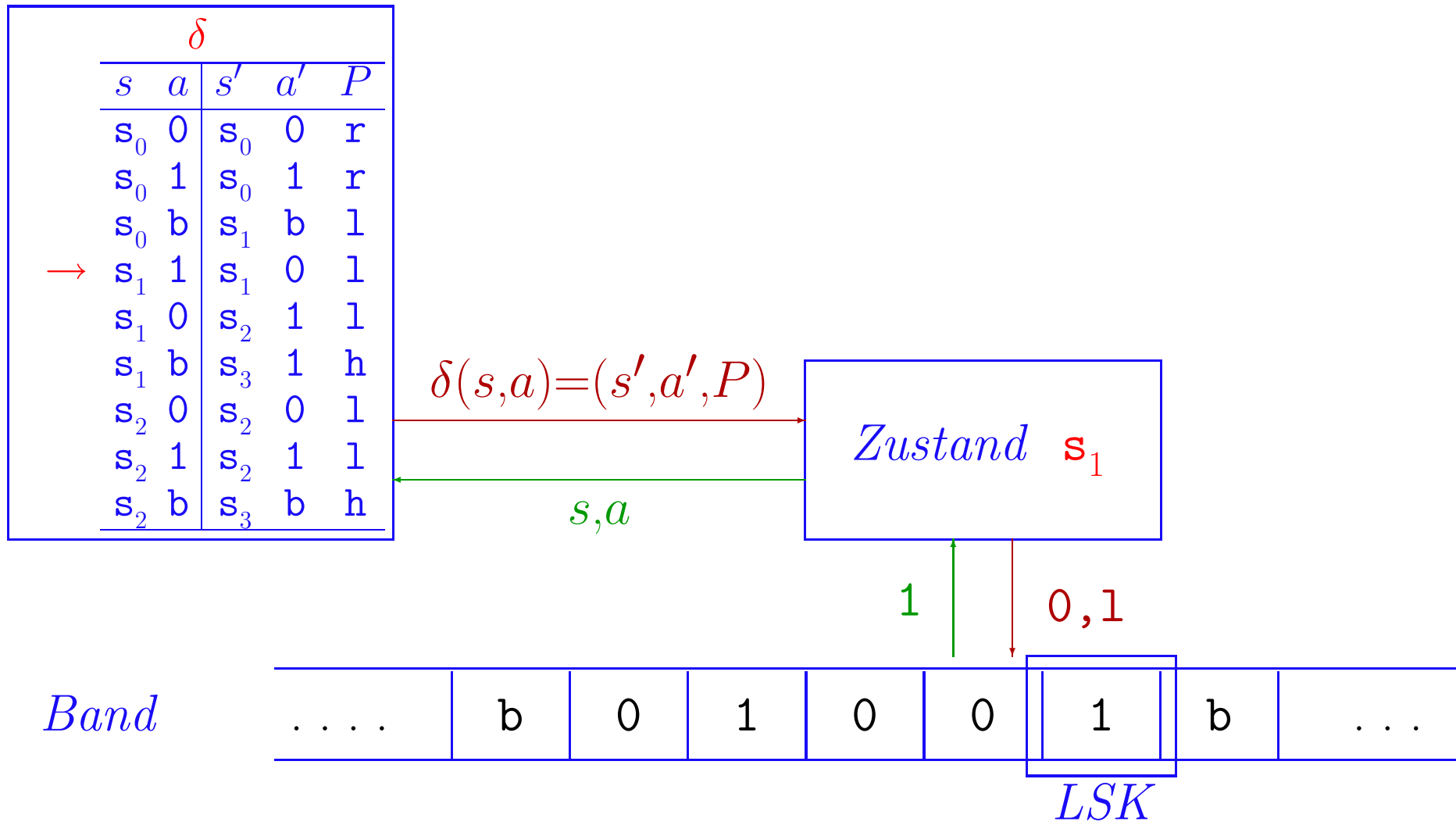
# ABARBEITUNG VON TURING-PROGRAMMEN



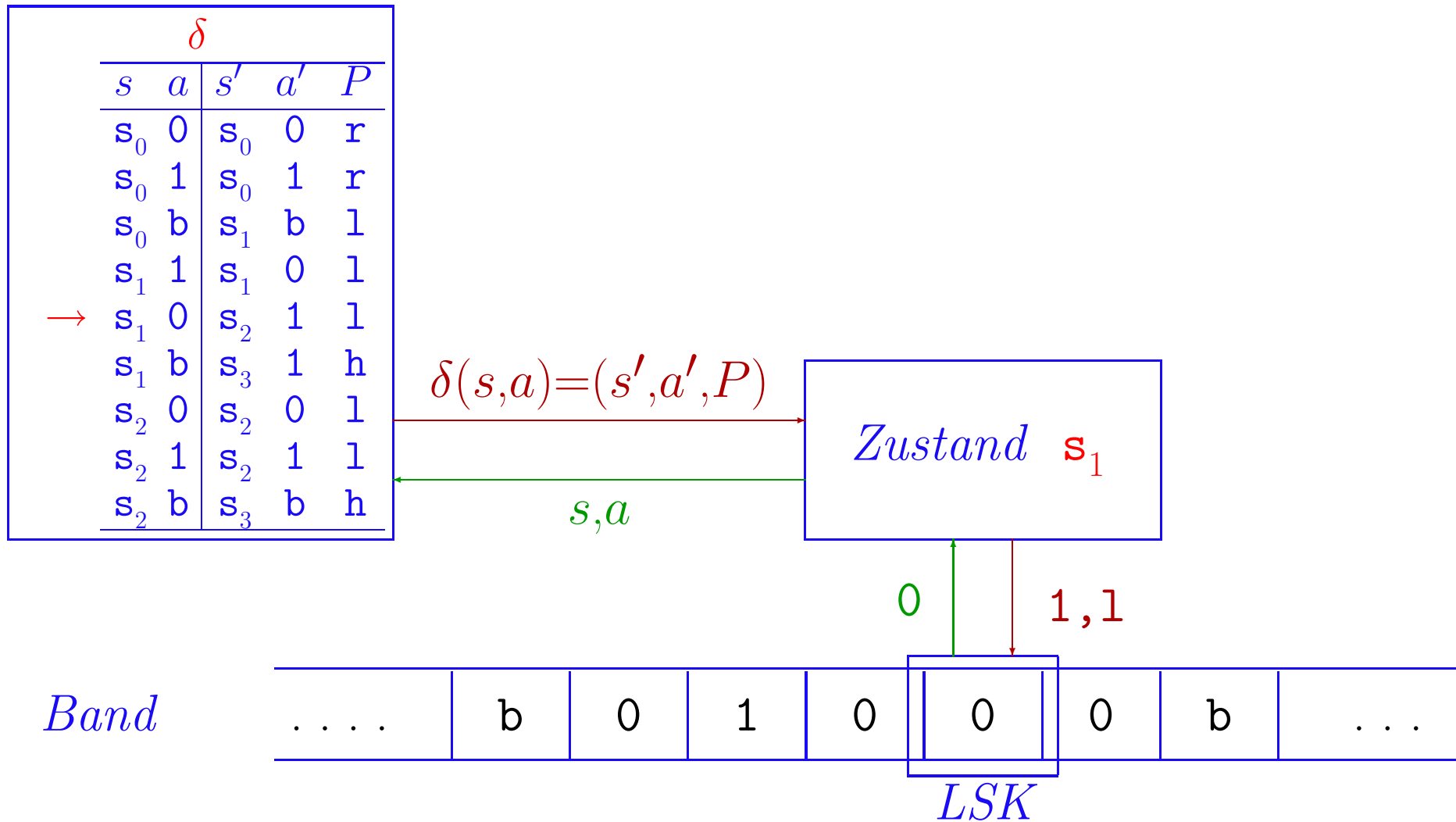
# ABARBEITUNG VON TURING-PROGRAMMEN



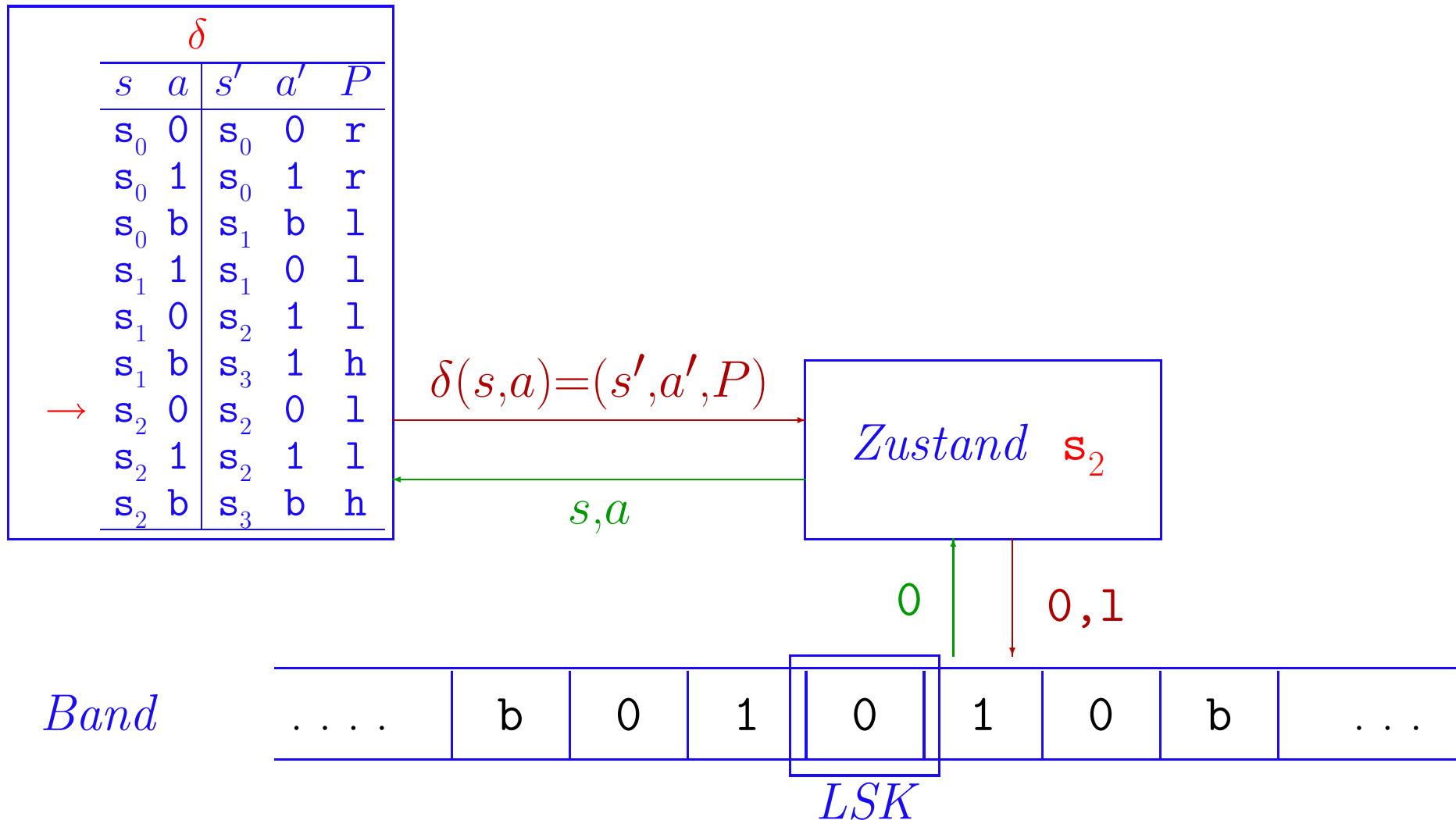
# ABARBEITUNG VON TURING-PROGRAMMEN



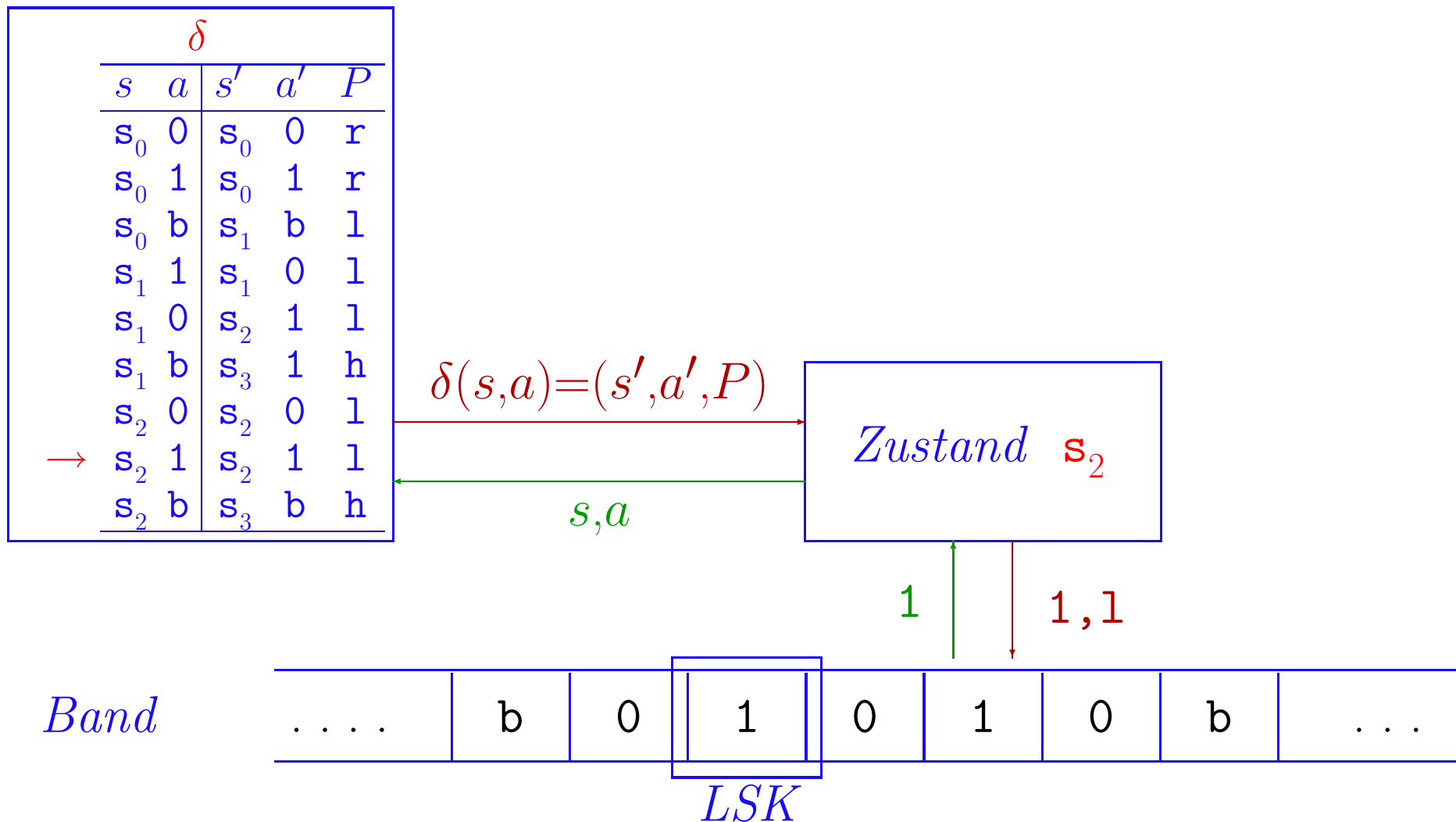
# ABARBEITUNG VON TURING-PROGRAMMEN



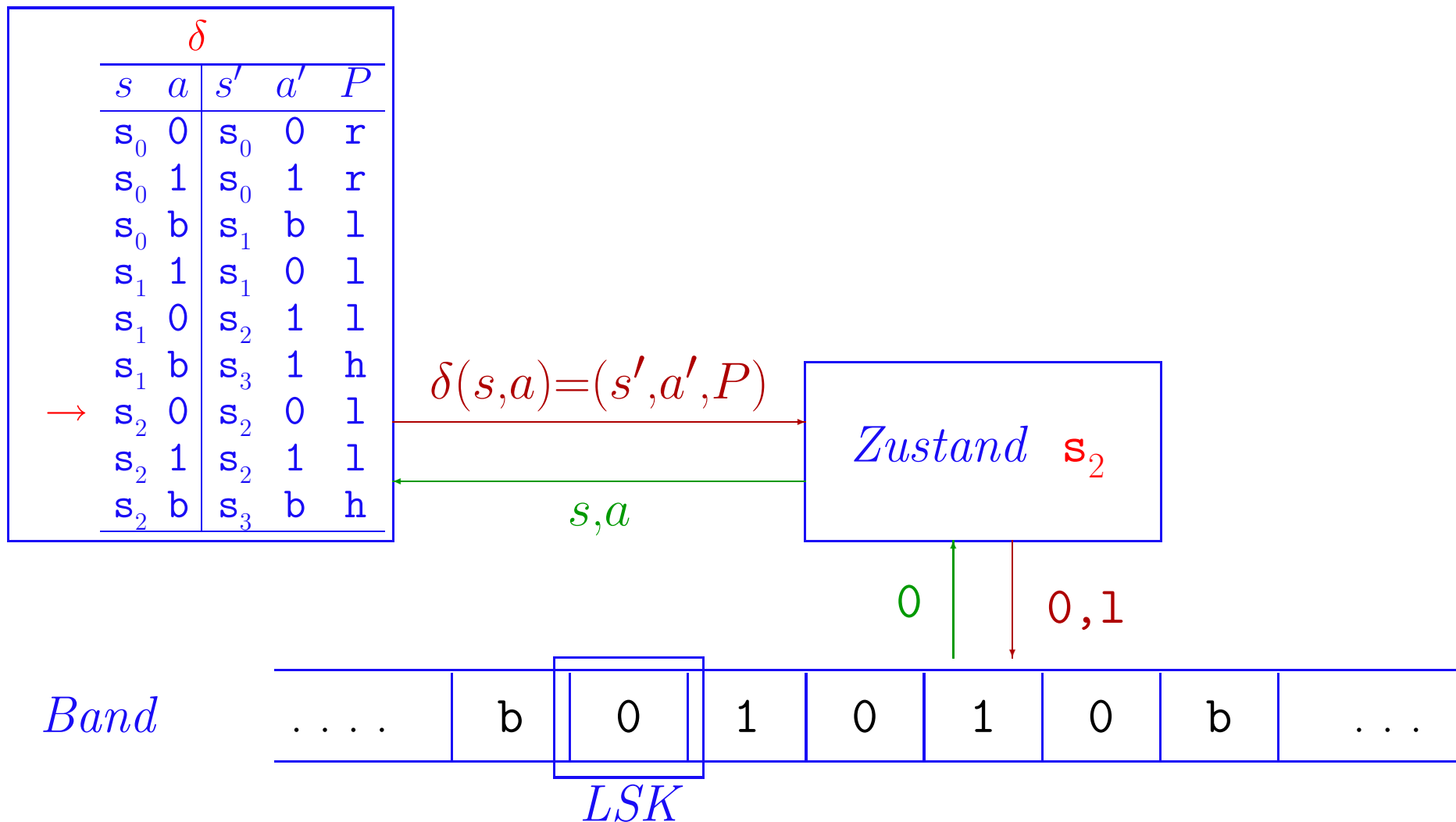
# ABARBEITUNG VON TURING-PROGRAMMEN



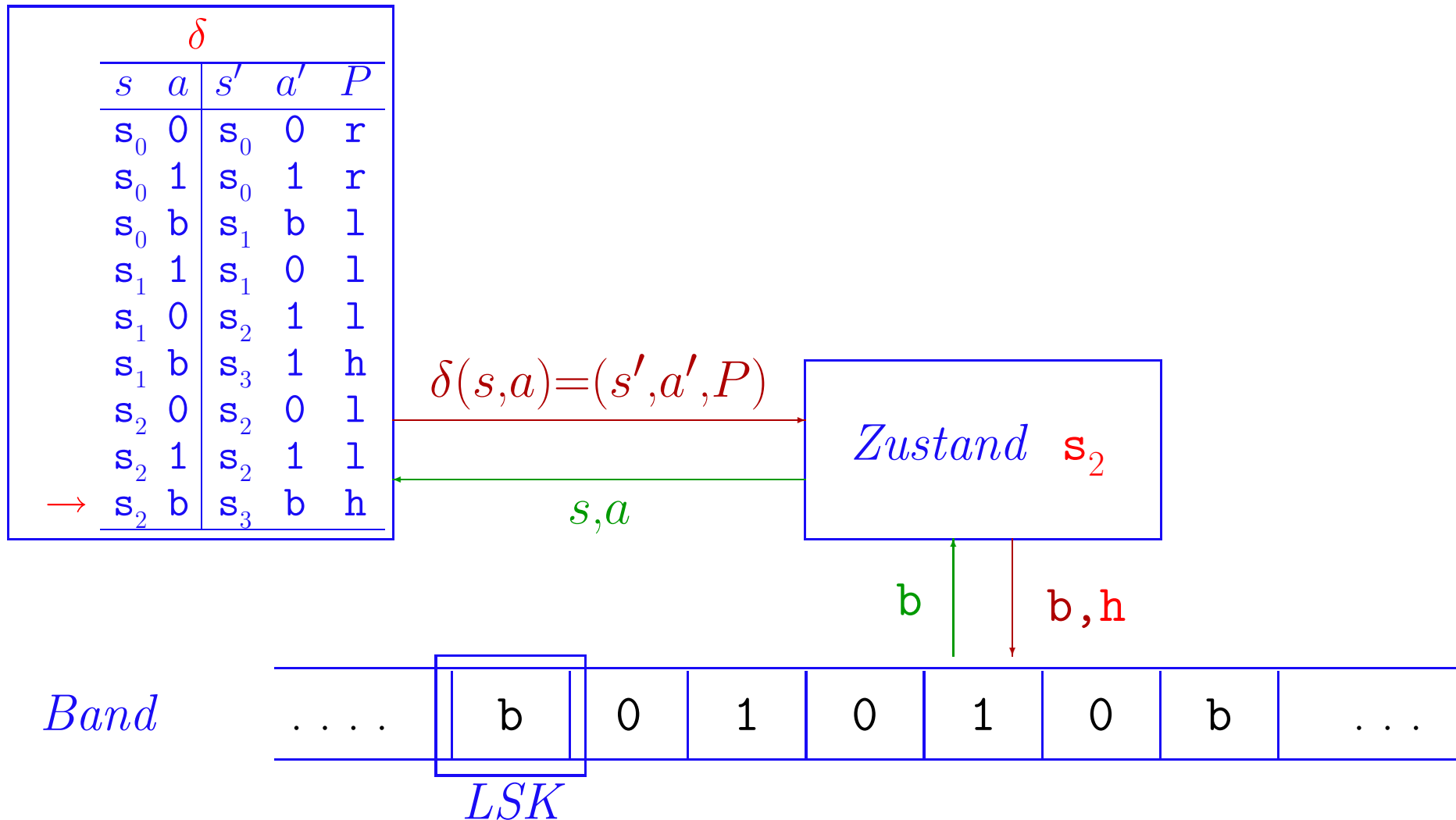
# ABARBEITUNG VON TURING-PROGRAMMEN



# ABARBEITUNG VON TURING-PROGRAMMEN



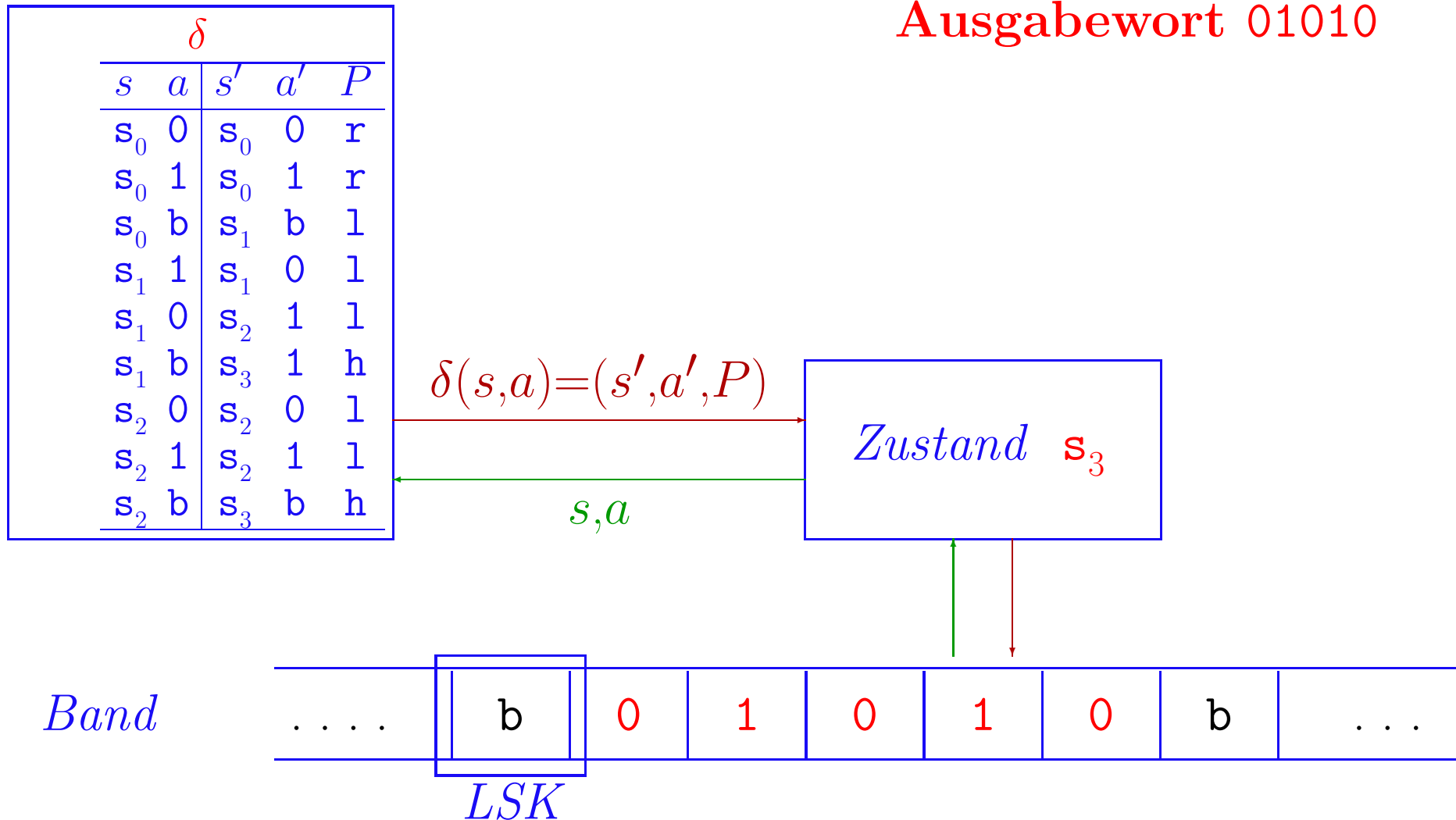
# ABARBEITUNG VON TURING-PROGRAMMEN





# ABARBEITUNG VON TURING-PROGRAMMEN

Ausgabewort 01010



- Definiere **Konfiguration** von  $\tau$ 
  - Schnapschuß der Turingmaschine  $\tau$  zu einem gegebenen Zeitpunkt
    - aktueller Zustand + Bandinhalt + Kopfposition
  - $K_\tau$ : Menge aller Konfigurationen von  $\tau$

- Definiere **Konfiguration** von  $\tau$ 
  - Schnapschuß der Turingmaschine  $\tau$  zu einem gegebenen Zeitpunkt
    - aktueller Zustand + Bandinhalt + Kopfposition
  - $K_\tau$ : Menge aller Konfigurationen von  $\tau$
- Definiere **Arbeitsweise** von  $\tau$ 
  - Anfangskonfiguration  $\alpha(w)$  für Eingabeworte  $w \in X^*$
  - Nachfolgekongfiguration (Arbeitsschritt)  $\hat{\delta}: K_\tau \rightarrow K_\tau$
  - Ausgabefunktion (Ergebnis)  $\omega: K_\tau \rightarrow \Gamma^*$

- Definiere **Konfiguration** von  $\tau$ 
  - Schnapschuß der Turingmaschine  $\tau$  zu einem gegebenen Zeitpunkt
    - aktueller Zustand + Bandinhalt + Kopfposition
  - $K_\tau$ : Menge aller Konfigurationen von  $\tau$
- Definiere **Arbeitsweise** von  $\tau$ 
  - Anfangskonfiguration  $\alpha(w)$  für Eingabeworte  $w \in X^*$
  - Nachfolgekongfiguration (Arbeitsschritt)  $\hat{\delta}: K_\tau \rightarrow K_\tau$
  - Ausgabefunktion (Ergebnis)  $\omega: K_\tau \rightarrow \Gamma^*$
- Definiere die von  $\tau$  **berechnete Funktion**  $h_\tau$

- Eine **Konfiguration** ist ein Tripel  $\kappa = (s, f, i)$  mit

- $s \in S$  aktueller Zustand

- $f: \mathbb{Z} \rightarrow \Gamma$  Bandinhaltsfunktion

$f(n) \equiv$  Inhalt der  $n$ -ten Bandzelle

$(f(j) = b \text{ für fast alle } j)$

- $i \in \mathbb{Z}$  Kopfposition

Alternative Repräsentation: Tripel  $(s, u, v)$  mit

- $s$  aktueller Zustand,

- $u$  String links vom Kopf (von rechts nach links),

- $v$  String rechts vom Kopf

- **$K_\tau$** : Menge aller Konfigurationen von  $\tau$

- **Anfangskonfiguration**  $\alpha: X^* \rightarrow K_\tau$ 
  - Für ein Eingabewort  $w = w_0 w_1 \dots w_k$  ist  $\alpha(w) = (s_0 f_w, 0)$ ,  
mit  $f_w(j) = \begin{cases} w_j & \text{falls } j \in \{0, \dots, k\}, \\ b & \text{sonst} \end{cases}$

- **Anfangskonfiguration**  $\alpha: X^* \rightarrow K_\tau$ 
  - Für ein Eingabewort  $w = w_0 w_1 \dots w_k$  ist  $\alpha(w) = (s_0 f_w, 0)$ ,  
mit  $f_w(j) = \begin{cases} w_j & \text{falls } j \in \{0, \dots, k\}, \\ b & \text{sonst} \end{cases}$
- **Nachfolgekonfiguration**  $\hat{\delta}: K_\tau \rightarrow K_\tau$ 
  - Für eine Konfiguration  $\kappa = (s, f, i)$  mit  $\delta(s, f(i)) = (s', a', P)$  ist  $\hat{\delta}(\kappa) = (s', f', i')$   
wobei  $f'(j) = \begin{cases} a' & \text{falls } j=i, \\ f(j) & \text{sonst} \end{cases}$  und  $i' = \begin{cases} i+1 & \text{falls } P=r, \\ i-1 & \text{falls } P=l, \\ i & \text{falls } P=h \end{cases}$

- **Anfangskonfiguration**  $\alpha: X^* \rightarrow K_\tau$ 
  - Für ein Eingabewort  $w = w_0 w_1 \dots w_k$  ist  $\alpha(w) = (s_0 f_w, 0)$ ,  
mit  $f_w(j) = \begin{cases} w_j & \text{falls } j \in \{0, \dots, k\}, \\ b & \text{sonst} \end{cases}$
- **Nachfolgekonfiguration**  $\hat{\delta}: K_\tau \rightarrow K_\tau$ 
  - Für eine Konfiguration  $\kappa = (s, f, i)$  mit  $\delta(s, f(i)) = (s', a', P)$  ist  $\hat{\delta}(\kappa) = (s', f', i')$   
wobei  $f'(j) = \begin{cases} a' & \text{falls } j=i, \\ f(j) & \text{sonst} \end{cases}$  und  $i' = \begin{cases} i+1 & \text{falls } P=r, \\ i-1 & \text{falls } P=l, \\ i & \text{falls } P=h \end{cases}$
- **Ausgabefunktion**  $\omega: K_\tau \rightarrow \Gamma^*$ 
  - Für eine Konfiguration  $\kappa = (s, f, i)$  ist  
 $\omega(\kappa) = \begin{cases} \epsilon & \text{falls } f(j)=b \text{ für alle } j, \\ f(k)f(k+1)\dots f(k+n) & \text{sonst} \end{cases}$   
wobei  $k = \max\{i \mid \forall j < i \ f(j)=b\}$  und  $n = \min\{i \mid \forall j > k+i \ f(j)=b\}$



## ● Intuitive Beschreibung

- Eingabe  $\alpha(w)$
- Wiederholte Anwendung von  $\hat{\delta}$
- Ausgabe  $\omega(\kappa)$ , wenn Stop-Konfiguration  $\kappa$  erreicht wird.
- **Undefiniert** (Endlosschleife), andernfalls

## ● Intuitive Beschreibung

- Eingabe  $\alpha(w)$
- Wiederholte Anwendung von  $\hat{\delta}$
- Ausgabe  $\omega(\kappa)$ , wenn Stop-Konfiguration  $\kappa$  erreicht wird.
- **Undefiniert** (Endlosschleife), andernfalls

## ● Mathematische Semantik von $\tau = (S, X, \Gamma, \delta, s_0, b)$

- Die von  $\tau =$  **berechnete Funktion**  $h_\tau: X^* \rightarrow \Gamma^*$  ist definiert durch

$$h_\tau(w) = \begin{cases} \omega(\hat{\delta}^{m+1}(\alpha(w))) & \text{falls } m = \min\{j \mid \exists s, f, i, s', a' \hat{\delta}^j(\alpha(w)) = (s, f, i) \\ & \text{und } \delta(s, f(i)) = (s', a', h)\} \\ \perp & \text{sonst} \end{cases}$$

## ● Intuitive Beschreibung

- Eingabe  $\alpha(w)$
- Wiederholte Anwendung von  $\hat{\delta}$
- Ausgabe  $\omega(\kappa)$ , wenn Stop-Konfiguration  $\kappa$  erreicht wird.
- **Undefiniert** (Endlosschleife), andernfalls

## ● Mathematische Semantik von $\tau = (S, X, \Gamma, \delta, s_0, b)$

- Die von  $\tau =$  **berechnete Funktion**  $h_\tau: X^* \rightarrow \Gamma^*$  ist definiert durch

$$h_\tau(w) = \begin{cases} \omega(\hat{\delta}^{m+1}(\alpha(w))) & \text{falls } m = \min\{j \mid \exists s, f, i, s', a' \hat{\delta}^j(\alpha(w)) = (s, f, i) \\ & \text{und } \delta(s, f(i)) = (s', a', h)\} \\ & \text{existiert,} \\ \perp & \text{sonst} \end{cases}$$

- **Definitionsbereich** von  $\tau$ :  $\{w \in X^* \mid h_\tau(w) \neq \perp\}$  (**Haltebereich**, domain)

## ● Intuitive Beschreibung

- Eingabe  $\alpha(w)$
- Wiederholte Anwendung von  $\hat{\delta}$
- Ausgabe  $\omega(\kappa)$ , wenn Stop-Konfiguration  $\kappa$  erreicht wird.
- **Undefiniert** (Endlosschleife), andernfalls

## ● Mathematische Semantik von $\tau = (S, X, \Gamma, \delta, s_0, b)$

- Die von  $\tau =$  **berechnete Funktion**  $h_\tau: X^* \rightarrow \Gamma^*$  ist definiert durch

$$h_\tau(w) = \begin{cases} \omega(\hat{\delta}^{m+1}(\alpha(w))) & \text{falls } m = \min\{j \mid \exists s, f, i, s', a' \quad \hat{\delta}^j(\alpha(w)) = (s, f, i) \\ & \text{und } \delta(s, f(i)) = (s', a', h)\} \\ & \text{existiert,} \\ \perp & \text{sonst} \end{cases}$$

- **Definitionsbereich** von  $\tau$ :  $\{w \in X^* \mid h_\tau(w) \neq \perp\}$  (**Haltebereich**, domain)
- **Wertebereich** von  $\tau$ :  $\{v \in \Gamma^* \mid \exists w \in X^* \quad h_\tau(w) = v\}$  (**Ergebnisbereich**, range)

# BEISPIELE FÜR TURING-MASCHINEN

•  $\tau_1 = (\{s_0\}, \{1\}, \{b, 1\}, \delta_1, s_0, b)$  mit  $\delta_1 =$

$s$	$a$	$s'$	$a'$	$P$
$s_0$	1	$s_0$	1	r
$s_0$	b	$s_0$	1	h

# BEISPIELE FÜR TURING-MASCHINEN

•  $\tau_1 = (\{s_0\}, \{1\}, \{b, 1\}, \delta_1, s_0, b)$  mit  $\delta_1 =$

$s$	$a$	$s'$	$a'$	$P$
$s_0$	1	$s_0$	1	r
$s_0$	b	$s_0$	1	h

Fügt am Ende eines Wortes  $w \in 1^*$  eine 1 an (“Bierdeckelmaschine”)

# BEISPIELE FÜR TURING-MASCHINEN

•  $\tau_1 = (\{s_0\}, \{1\}, \{b, 1\}, \delta_1, s_0, b)$  mit  $\delta_1 =$

$s$	$a$	$s'$	$a'$	$P$
$s_0$	1	$s_0$	1	r
$s_0$	b	$s_0$	1	h

Fügt am Ende eines Wortes  $w \in 1^*$  eine 1 an (“Bierdeckelmaschine”)

- **Mathematische Analyse:**

# BEISPIELE FÜR TURING-MASCHINEN

•  $\tau_1 = (\{s_0\}, \{1\}, \{b, 1\}, \delta_1, s_0, b)$  mit  $\delta_1 =$

$s$	$a$	$s'$	$a'$	$P$
$s_0$	$1$	$s_0$	$1$	$r$
$s_0$	$b$	$s_0$	$1$	$h$

Fügt am Ende eines Wortes  $w \in 1^*$  eine  $1$  an (“Bierdeckelmaschine”)

• **Mathematische Analyse:**

– Anfangskonfiguration:  $\alpha(1^n) = (s_0, f_n, 0)$ , wobei  $f_n(j) = \begin{cases} 1 & \text{falls } j \in \{0, \dots, n-1\}, \\ b & \text{sonst} \end{cases}$



# BEISPIELE FÜR TURING-MASCHINEN

•  $\tau_1 = (\{s_0\}, \{1\}, \{b, 1\}, \delta_1, s_0, b)$  mit  $\delta_1 =$

$s$	$a$	$s'$	$a'$	$P$
$s_0$	$1$	$s_0$	$1$	$r$
$s_0$	$b$	$s_0$	$1$	$h$

Fügt am Ende eines Wortes  $w \in 1^*$  eine  $1$  an (“Bierdeckelmaschine”)

• **Mathematische Analyse:**

- Anfangskonfiguration:  $\alpha(1^n) = (s_0, f_n, 0)$ , wobei  $f_n(j) = \begin{cases} 1 & \text{falls } j \in \{0, \dots, n-1\}, \\ b & \text{sonst} \end{cases}$
- Nachfolgekonfigurationen:  $\hat{\delta}(s_0, f_n, j) = \begin{cases} (s_0, f_n, j+1) & \text{falls } j \in \{0, \dots, n-1\}, \\ (s_0, f_{n+1}, n) & \text{falls } j=n \end{cases}$

# BEISPIELE FÜR TURING-MASCHINEN

•  $\tau_1 = (\{s_0\}, \{1\}, \{b, 1\}, \delta_1, s_0, b)$  mit  $\delta_1 =$

$s$	$a$	$s'$	$a'$	$P$
$s_0$	$1$	$s_0$	$1$	$r$
$s_0$	$b$	$s_0$	$1$	$h$

Fügt am Ende eines Wortes  $w \in 1^*$  eine  $1$  an (“Bierdeckelmaschine”)

• **Mathematische Analyse:**

- Anfangskonfiguration:  $\alpha(1^n) = (s_0, f_n, 0)$ , wobei  $f_n(j) = \begin{cases} 1 & \text{falls } j \in \{0, \dots, n-1\}, \\ b & \text{sonst} \end{cases}$
- Nachfolgekonfigurationen:  $\hat{\delta}(s_0, f_n, j) = \begin{cases} (s_0, f_n, j+1) & \text{falls } j \in \{0, \dots, n-1\}, \\ (s_0, f_{n+1}, n) & \text{falls } j=n \end{cases}$
- Terminierung:  $\min\{j \mid \hat{\delta}^j(s_0, f_n, 0) = (s_0, f_n, j) \wedge \delta(s_0, f_n(j)) = (s_0, b, h)\} = n$

# BEISPIELE FÜR TURING-MASCHINEN

•  $\tau_1 = (\{s_0\}, \{1\}, \{b, 1\}, \delta_1, s_0, b)$  mit  $\delta_1 =$

$s$	$a$	$s'$	$a'$	$P$
$s_0$	$1$	$s_0$	$1$	$r$
$s_0$	$b$	$s_0$	$1$	$h$

Fügt am Ende eines Wortes  $w \in 1^*$  eine  $1$  an (“Bierdeckelmaschine”)

## • Mathematische Analyse:

- Anfangskonfiguration:  $\alpha(1^n) = (s_0, f_n, 0)$ , wobei  $f_n(j) = \begin{cases} 1 & \text{falls } j \in \{0, \dots, n-1\}, \\ b & \text{sonst} \end{cases}$
- Nachfolgekonfigurationen:  $\hat{\delta}(s_0, f_n, j) = \begin{cases} (s_0, f_n, j+1) & \text{falls } j \in \{0, \dots, n-1\}, \\ (s_0, f_{n+1}, n) & \text{falls } j=n \end{cases}$
- Terminierung:  $\min\{j \mid \hat{\delta}^j(s_0, f_n, 0) = (s_0, f_n, j) \wedge \delta(s_0, f_n(j)) = (s_0, b, h)\} = n$
- Ergebnis:  $\hat{\delta}^{n+1}(s_0, f_n, 0) = (s_0, f_{n+1}, n)$

# BEISPIELE FÜR TURING-MASCHINEN

•  $\tau_1 = (\{s_0\}, \{1\}, \{b, 1\}, \delta_1, s_0, b)$  mit  $\delta_1 =$

$s$	$a$	$s'$	$a'$	$P$
$s_0$	$1$	$s_0$	$1$	$r$
$s_0$	$b$	$s_0$	$1$	$h$

Fügt am Ende eines Wortes  $w \in 1^*$  eine  $1$  an (“Bierdeckelmaschine”)

## • Mathematische Analyse:

- Anfangskonfiguration:  $\alpha(1^n) = (s_0, f_n, 0)$ , wobei  $f_n(j) = \begin{cases} 1 & \text{falls } j \in \{0, \dots, n-1\}, \\ b & \text{sonst} \end{cases}$
- Nachfolgekonfigurationen:  $\hat{\delta}(s_0, f_n, j) = \begin{cases} (s_0, f_n, j+1) & \text{falls } j \in \{0, \dots, n-1\}, \\ (s_0, f_{n+1}, n) & \text{falls } j=n \end{cases}$
- Terminierung:  $\min\{j \mid \hat{\delta}^j(s_0, f_n, 0) = (s_0, f_n, j) \wedge \delta(s_0, f_n(j)) = (s_0, b, h)\} = n$
- Ergebnis:  $\hat{\delta}^{n+1}(s_0, f_n, 0) = (s_0, f_{n+1}, n)$
- Ausgabefunktion:  $\omega(s_0, f_{n+1}, n) = 1^{n+1}$   
 $(\max\{i \mid \forall j < i \ f_{n+1}(j) = b\} = 0, \min\{i \mid \forall j > i \ f_{n+1}(j) = b\} = n+1)$

# BEISPIELE FÜR TURING-MASCHINEN

•  $\tau_1 = (\{s_0\}, \{1\}, \{b, 1\}, \delta_1, s_0, b)$  mit  $\delta_1 =$

$s$	$a$	$s'$	$a'$	$P$
$s_0$	$1$	$s_0$	$1$	$r$
$s_0$	$b$	$s_0$	$1$	$h$

Fügt am Ende eines Wortes  $w \in 1^*$  eine  $1$  an (“Bierdeckelmaschine”)

## • Mathematische Analyse:

- Anfangskonfiguration:  $\alpha(1^n) = (s_0, f_n, 0)$ , wobei  $f_n(j) = \begin{cases} 1 & \text{falls } j \in \{0, \dots, n-1\}, \\ b & \text{sonst} \end{cases}$
- Nachfolgekonfigurationen:  $\hat{\delta}(s_0, f_n, j) = \begin{cases} (s_0, f_n, j+1) & \text{falls } j \in \{0, \dots, n-1\}, \\ (s_0, f_{n+1}, n) & \text{falls } j=n \end{cases}$
- Terminierung:  $\min\{j \mid \hat{\delta}^j(s_0, f_n, 0) = (s_0, f_n, j) \wedge \delta(s_0, f_n(j)) = (s_0, b, h)\} = n$
- Ergebnis:  $\hat{\delta}^{n+1}(s_0, f_n, 0) = (s_0, f_{n+1}, n)$
- Ausgabefunktion:  $\omega(s_0, f_{n+1}, n) = 1^{n+1}$   
 $(\max\{i \mid \forall j < i \ f_{n+1}(j) = b\} = 0, \min\{i \mid \forall j > i \ f_{n+1}(j) = b\} = n+1)$



$h_{\tau_1}(1^n) = 1^{n+1} \text{ für alle } n, \text{ Definitionsbereich } \{1\}^*, \text{ Wertebereich } \{1\}^+$

## BEISPIELE FÜR TURING-MASCHINEN II

•  $\tau_2 = (\{s_0\}, \{1\}, \{b, 1\}, \delta_2, s_0, b)$  mit  $\delta_2 =$

$s$	$a$	$s'$	$a'$	$P$
$s_0$	1	$s_0$	b	r
$s_0$	b	$s_0$	b	h

## BEISPIELE FÜR TURING-MASCHINEN II

•  $\tau_2 = (\{s_0\}, \{1\}, \{b, 1\}, \delta_2, s_0, b)$  mit  $\delta_2 =$

$s$	$a$	$s'$	$a'$	$P$
$s_0$	1	$s_0$	b	r
$s_0$	b	$s_0$	b	h

Löscht ein Wort  $w \in 1^*$ :

$h_{\tau_2}(w) = \epsilon$  für alle  $w$ , Definitionsbereich  $\{1\}^*$ , Wertebereich  $\{\epsilon\}$

# BEISPIELE FÜR TURING-MASCHINEN II

- $\tau_2 = (\{s_0\}, \{1\}, \{b, 1\}, \delta_2, s_0, b)$  mit  $\delta_2 =$ 

$s$	$a$	$s'$	$a'$	$P$
$s_0$	1	$s_0$	b	r
$s_0$	b	$s_0$	b	h

Löscht ein Wort  $w \in 1^*$ :

$h_{\tau_2}(w) = \epsilon$  für alle  $w$ , Definitionsbereich  $\{1\}^*$ , Wertebereich  $\{\epsilon\}$

- $\tau_3 = (\{s_0, s_1\}, \{1\}, \{b, 1\}, \delta_3, s_0, b)$  mit  $\delta_3 =$ 

$s$	$a$	$s'$	$a'$	$P$
$s_0$	1	$s_1$	1	r
$s_0$	b	$s_1$	1	h
$s_1$	1	$s_0$	1	r
$s_1$	b	$s_1$	b	r



# BEISPIELE FÜR TURING-MASCHINEN II

- $\tau_2 = (\{s_0\}, \{1\}, \{b, 1\}, \delta_2, s_0, b)$  mit  $\delta_2 =$ 

$s$	$a$	$s'$	$a'$	$P$
$s_0$	1	$s_0$	b	r
$s_0$	b	$s_0$	b	h

Löscht ein Wort  $w \in 1^*$ :

$h_{\tau_2}(w) = \epsilon$  für alle  $w$ , Definitionsbereich  $\{1\}^*$ , Wertebereich  $\{\epsilon\}$

- $\tau_3 = (\{s_0, s_1\}, \{1\}, \{b, 1\}, \delta_3, s_0, b)$  mit  $\delta_3 =$ 

$s$	$a$	$s'$	$a'$	$P$
$s_0$	1	$s_1$	1	r
$s_0$	b	$s_1$	1	h
$s_1$	1	$s_0$	1	r
$s_1$	b	$s_1$	b	r

Testet Anzahl der Einsen in  $w \in 1^*$ :

$$h_{\tau_3}(1^n) = \begin{cases} 1^{n+1} & \text{falls } n \text{ gerade,} \\ \perp & \text{sonst} \end{cases}$$

# BEISPIELE FÜR TURING-MASCHINEN II

- $\tau_2 = (\{s_0\}, \{1\}, \{b, 1\}, \delta_2, s_0, b)$  mit  $\delta_2 =$ 

$s$	$a$	$s'$	$a'$	$P$
$s_0$	1	$s_0$	b	r
$s_0$	b	$s_0$	b	h

Löscht ein Wort  $w \in 1^*$ :

$h_{\tau_2}(w) = \epsilon$  für alle  $w$ , Definitionsbereich  $\{1\}^*$ , Wertebereich  $\{\epsilon\}$

- $\tau_3 = (\{s_0, s_1\}, \{1\}, \{b, 1\}, \delta_3, s_0, b)$  mit  $\delta_3 =$ 

$s$	$a$	$s'$	$a'$	$P$
$s_0$	1	$s_1$	1	r
$s_0$	b	$s_1$	1	h
$s_1$	1	$s_0$	1	r
$s_1$	b	$s_1$	b	r

Testet Anzahl der Einsen in  $w \in 1^*$ :

$$h_{\tau_3}(1^n) = \begin{cases} 1^{n+1} & \text{falls } n \text{ gerade,} \\ \perp & \text{sonst} \end{cases}$$

Definitionsbereich  $\{1^{2k} \mid k \in \mathbb{N}\}$ , Wertebereich  $\{1^{2k+1} \mid k \in \mathbb{N}\}$

# BEISPIELE FÜR TURING-MASCHINEN III

•  $\tau_4 = (\{s_0, s_1, s_2, s_3\}, \{1\}, \{b, 1, c\}, \delta_4, s_0, b)$  mit  $\delta_4 =$

$s$	$a$	$s'$	$a'$	$P$
$s_0$	1	$s_1$	b	r
$s_0$	c	$s_0$	c	h
$s_0$	b	$s_0$	b	h
$s_1$	1	$s_1$	1	r
$s_1$	c	$s_1$	c	r
$s_1$	b	$s_2$	c	r
$s_2$	1	$s_2$	1	h
$s_2$	c	$s_2$	c	h
$s_2$	b	$s_3$	c	l
$s_3$	1	$s_3$	1	l
$s_3$	c	$s_3$	c	l
$s_3$	b	$s_0$	b	r

# BEISPIELE FÜR TURING-MASCHINEN III

•  $\tau_4 = (\{s_0, s_1, s_2, s_3\}, \{1\}, \{b, 1, c\}, \delta_4, s_0, b)$  mit  $\delta_4 =$

$s$	$a$	$s'$	$a'$	$P$
$s_0$	1	$s_1$	b	r
$s_0$	c	$s_0$	c	h
$s_0$	b	$s_0$	b	h
$s_1$	1	$s_1$	1	r
$s_1$	c	$s_1$	c	r
$s_1$	b	$s_2$	c	r
$s_2$	1	$s_2$	1	h
$s_2$	c	$s_2$	c	h
$s_2$	b	$s_3$	c	l
$s_3$	1	$s_3$	1	l
$s_3$	c	$s_3$	c	l
$s_3$	b	$s_0$	b	r

Verdoppelt Anzahl der Einsen

$h_{\tau_4}(1^n) = c^{2n}$ , Definitionsbereich  $\{1\}^*$ , Wertebereich  $\{c^{2k} \mid k \in \mathbb{N}\}$

# BEISPIELE FÜR TURING-MASCHINEN III

•  $\tau_4 = (\{s_0, s_1, s_2, s_3\}, \{1\}, \{b, 1, c\}, \delta_4, s_0, b)$  mit  $\delta_4 =$

$s$	$a$	$s'$	$a'$	$P$
$s_0$	1	$s_1$	b	r
$s_0$	c	$s_0$	c	h
$s_0$	b	$s_0$	b	h
$s_1$	1	$s_1$	1	r
$s_1$	c	$s_1$	c	r
$s_1$	b	$s_2$	c	r
$s_2$	1	$s_2$	1	h
$s_2$	c	$s_2$	c	h
$s_2$	b	$s_3$	c	l
$s_3$	1	$s_3$	1	l
$s_3$	c	$s_3$	c	l
$s_3$	b	$s_0$	b	r

Verdoppelt Anzahl der Einsen

$h_{\tau_4}(1^n) = c^{2n}$ , Definitionsbereich  $\{1\}^*$ , Wertebereich  $\{c^{2k} \mid k \in \mathbb{N}\}$

Kombinierbar mit isomorpher Variante von  $\tau_3$ :  $h_{\tau'_3} \circ h_{\tau_4}(1^n) = c^{2n+1}$

- $f: X^* \rightarrow Y^*$  **Turing-berechenbar**

- Es gibt eine Turingmaschine  $\tau = (S, X, \Gamma, \delta, s_0, b)$  mit  $Y \subseteq \Gamma$  und  $h_\tau = f$

- **$\mathcal{T}$** : Menge der Turing-berechenbaren Funktionen

- $\mathcal{T}_{X,Y} = \{f: X^* \rightarrow Y^* \mid f \text{ ist Turing-berechenbar}\}$

- $\mathcal{T} = \bigcup \{\mathcal{T}_{X,Y} \mid X, Y \text{ endliches Alphabet}\}$

# ÜBERTRAGUNG DES BERECHENBARKEITBEGRIFFS

## ● Berechenbarkeit von Mengen $M \subseteq X^*$

- **Semi-Entscheidbarkeit**: Berechenbarkeit von  $\psi_M: X^* \rightarrow \{0,1\}^*$ ,
- **Entscheidbarkeit**: Berechenbarkeit von  $\chi_M: X^* \rightarrow \{0,1\}^*$ ,

$$\text{wobei } \psi_M(w) = \begin{cases} 1 & \text{falls } w \in M, \\ \perp & \text{sonst} \end{cases} \quad \chi_M(w) = \begin{cases} 1 & \text{falls } w \in M, \\ 0 & \text{sonst} \end{cases}$$

(partiell-)charakteristische Funktion

# ÜBERTRAGUNG DES BERECHENBARKEITBEGRIFFS

## ● Berechenbarkeit von **Mengen** $M \subseteq X^*$

- **Semi-Entscheidbarkeit**: Berechenbarkeit von  $\psi_M: X^* \rightarrow \{0,1\}^*$ ,
- **Entscheidbarkeit**: Berechenbarkeit von  $\chi_M: X^* \rightarrow \{0,1\}^*$ ,

$$\text{wobei } \psi_M(w) = \begin{cases} 1 & \text{falls } w \in M, \\ \perp & \text{sonst} \end{cases} \quad \chi_M(w) = \begin{cases} 1 & \text{falls } w \in M, \\ 0 & \text{sonst} \end{cases}$$

(partiell-)charakteristische Funktion

## ● Berechenbarkeit auf **Zahlen** $f: \mathbb{N} \rightarrow \mathbb{N}$

$\equiv$  Berechenbarkeit der **Repräsentation**  $f_r: X^* \rightarrow X^*$ ,

wobei  $r: \mathbb{N} \rightarrow X^*$  bijektiv und  $f_r(w) = r(f(r^{-1}(w)))$

Standardcodierungen von Zahlen

- **unäre** Darstellung  $r_u: \mathbb{N} \rightarrow \{1\}^*$  mit  $r_u(n) = 1^n$
- **binäre** Darstellung  $r_b: \mathbb{N} \rightarrow \{0,1\}^*$  (ohne führende Nullen)



# ÜBERTRAGUNG DES BERECHENBARKEITBEGRIFFS

## ● Berechenbarkeit von **Mengen** $M \subseteq X^*$

- **Semi-Entscheidbarkeit**: Berechenbarkeit von  $\psi_M: X^* \rightarrow \{0,1\}^*$ ,
- **Entscheidbarkeit**: Berechenbarkeit von  $\chi_M: X^* \rightarrow \{0,1\}^*$ ,

$$\text{wobei } \psi_M(w) = \begin{cases} 1 & \text{falls } w \in M, \\ \perp & \text{sonst} \end{cases} \quad \chi_M(w) = \begin{cases} 1 & \text{falls } w \in M, \\ 0 & \text{sonst} \end{cases}$$

(partiell-)charakteristische Funktion

## ● Berechenbarkeit auf **Zahlen** $f: \mathbb{N} \rightarrow \mathbb{N}$

$\equiv$  Berechenbarkeit der **Repräsentation**  $f_r: X^* \rightarrow X^*$ ,  
wobei  $r: \mathbb{N} \rightarrow X^*$  bijektiv und  $f_r(w) = r(f(r^{-1}(w)))$

Standardcodierungen von Zahlen

- **unäre** Darstellung  $r_u: \mathbb{N} \rightarrow \{1\}^*$  mit  $r_u(n) = 1^n$
- **binäre** Darstellung  $r_b: \mathbb{N} \rightarrow \{0,1\}^*$  (ohne führende Nullen)

## ● Berechenbarkeit auf **Tupeln** $f: X^* \times X^* \rightarrow Y^*$

$\equiv$  Berechenbarkeit von  $f': (X \cup \{\#\})^* \rightarrow Y^*$  mit  $f'(v\#w) = f(v,w)$

# BERECHENBARKEIT DER NACHFOLGERFUNKTION

Ist  $s:\mathbb{N}\rightarrow\mathbb{N}$  mit  $s(n) = n+1$  Turing-berechenbar?

# BERECHENBARKEIT DER NACHFOLGERFUNKTION

Ist  $s:\mathbb{N}\rightarrow\mathbb{N}$  mit  $s(n) = n+1$  Turing-berechenbar?

- Bei unärer Codierung:
  - Ist  $s_u:\{1\}^*\rightarrow\{1\}^*$  mit  $s_u(1^n) = 1^{n+1}$  Turing-berechenbar?

# BERECHENBARKEIT DER NACHFOLGERFUNKTION

Ist  $s:\mathbb{N}\rightarrow\mathbb{N}$  mit  $s(n) = n+1$  Turing-berechenbar?

- Bei unärer Codierung:
  - Ist  $s_u:\{1\}^*\rightarrow\{1\}^*$  mit  $s_u(1^n) = 1^{n+1}$  Turing-berechenbar?
  - Turingmaschine muß eine 1 anhängen:  $s_u = h_{\tau_1}$

Ist  $s:\mathbb{N}\rightarrow\mathbb{N}$  mit  $s(n) = n+1$  Turing-berechenbar?

- Bei unärer Codierung:
  - Ist  $s_u:\{1\}^*\rightarrow\{1\}^*$  mit  $s_u(1^n) = 1^{n+1}$  Turing-berechenbar?
  - Turingmaschine muß eine 1 anhängen:  $s_u = h_{\tau_1}$
- Bei binärer Codierung
  - Ist  $s_b:\{0,1\}^*\rightarrow\{0,1\}^*$  mit  $s_b(r_b(n)) = r_b(n+1)$  Turing-berechenbar?

Ist  $s:\mathbb{N}\rightarrow\mathbb{N}$  mit  $s(n) = n+1$  Turing-berechenbar?

- Bei unärer Codierung:
  - Ist  $s_u:\{1\}^*\rightarrow\{1\}^*$  mit  $s_u(1^n) = 1^{n+1}$  Turing-berechenbar?
  - Turingmaschine muß eine 1 anhängen:  $s_u = h_{\tau_1}$
- Bei binärer Codierung
  - Ist  $s_b:\{0,1\}^*\rightarrow\{0,1\}^*$  mit  $s_b(r_b(n)) = r_b(n+1)$  Turing-berechenbar?
  - $\tau_s$  muß Ziffern von rechts beginnend umwandeln, ggf. mit Übertrag

# BERECHENBARKEIT DER NACHFOLGERFUNKTION

Ist  $s:\mathbb{N}\rightarrow\mathbb{N}$  mit  $s(n) = n+1$  Turing-berechenbar?

- Bei unärer Codierung:
  - Ist  $s_u:\{1\}^*\rightarrow\{1\}^*$  mit  $s_u(1^n) = 1^{n+1}$  Turing-berechenbar?
  - Turingmaschine muß eine 1 anhängen:  $s_u = h_{\tau_1}$
- Bei binärer Codierung
  - Ist  $s_b:\{0,1\}^*\rightarrow\{0,1\}^*$  mit  $s_b(r_b(n)) = r_b(n+1)$  Turing-berechenbar?
  - $\tau_s$  muß Ziffern von rechts beginnend umwandeln, ggf. mit Übertrag

$s$	$a$	$s'$	$a'$	$P$	
$s_0$	0	$s_0$	0	r	<i>rechtes Ende suchen</i>
$s_0$	1	$s_0$	1	r	<i>rechtes Ende suchen</i>
$s_0$	b	$s_1$	b	l	<i>rechtes Ende gefunden</i>

# BERECHENBARKEIT DER NACHFOLGERFUNKTION

Ist  $s:\mathbb{N}\rightarrow\mathbb{N}$  mit  $s(n) = n+1$  Turing-berechenbar?

- Bei unärer Codierung:
  - Ist  $s_u:\{1\}^*\rightarrow\{1\}^*$  mit  $s_u(1^n) = 1^{n+1}$  Turing-berechenbar?
  - Turingmaschine muß eine 1 anhängen:  $s_u = h_{\tau_1}$
- Bei binärer Codierung
  - Ist  $s_b:\{0,1\}^*\rightarrow\{0,1\}^*$  mit  $s_b(r_b(n)) = r_b(n+1)$  Turing-berechenbar?
  - $\tau_s$  muß Ziffern von rechts beginnend umwandeln, ggf. mit Übertrag

$s$	$a$	$s'$	$a'$	$P$	
$s_0$	0	$s_0$	0	r	<i>rechtes Ende suchen</i>
$s_0$	1	$s_0$	1	r	<i>rechtes Ende suchen</i>
$s_0$	b	$s_1$	b	l	<i>rechtes Ende gefunden</i>
$s_1$	1	$s_1$	0	l	<i>Addieren mit Übertrag</i>



# BERECHENBARKEIT DER NACHFOLGERFUNKTION

Ist  $s:\mathbb{N}\rightarrow\mathbb{N}$  mit  $s(n) = n+1$  Turing-berechenbar?

- Bei unärer Codierung:
  - Ist  $s_u:\{1\}^*\rightarrow\{1\}^*$  mit  $s_u(1^n) = 1^{n+1}$  Turing-berechenbar?
  - Turingmaschine muß eine 1 anhängen:  $s_u = h_{\tau_1}$
- Bei binärer Codierung
  - Ist  $s_b:\{0,1\}^*\rightarrow\{0,1\}^*$  mit  $s_b(r_b(n)) = r_b(n+1)$  Turing-berechenbar?
  - $\tau_s$  muß Ziffern von rechts beginnend umwandeln, ggf. mit Übertrag

$s$	$a$	$s'$	$a'$	$P$	
$s_0$	0	$s_0$	0	r	<i>rechtes Ende suchen</i>
$s_0$	1	$s_0$	1	r	<i>rechtes Ende suchen</i>
$s_0$	b	$s_1$	b	l	<i>rechtes Ende gefunden</i>
$s_1$	1	$s_1$	0	l	<i>Addieren mit Übertrag</i>
$s_1$	0	$s_2$	1	h	<i>Addieren ohne Übertrag</i>

# BERECHENBARKEIT DER NACHFOLGERFUNKTION

Ist  $s:\mathbb{N}\rightarrow\mathbb{N}$  mit  $s(n) = n+1$  Turing-berechenbar?

- Bei unärer Codierung:
  - Ist  $s_u:\{1\}^*\rightarrow\{1\}^*$  mit  $s_u(1^n) = 1^{n+1}$  Turing-berechenbar?
  - Turingmaschine muß eine 1 anhängen:  $s_u = h_{\tau_1}$
- Bei binärer Codierung
  - Ist  $s_b:\{0,1\}^*\rightarrow\{0,1\}^*$  mit  $s_b(r_b(n)) = r_b(n+1)$  Turing-berechenbar?
  - $\tau_s$  muß Ziffern von rechts beginnend umwandeln, ggf. mit Übertrag

$s$	$a$	$s'$	$a'$	$P$	
$s_0$	0	$s_0$	0	r	<i>rechtes Ende suchen</i>
$s_0$	1	$s_0$	1	r	<i>rechtes Ende suchen</i>
$s_0$	b	$s_1$	b	l	<i>rechtes Ende gefunden</i>
$s_1$	1	$s_1$	0	l	<i>Addieren mit Übertrag</i>
$s_1$	0	$s_2$	1	h	<i>Addieren ohne Übertrag</i>
$s_1$	b	$s_2$	1	h	<i>Übertrag am linken Ende</i>

## BERECHENBARKEIT DER DIVISION DURCH 2

Ist  $div_2: \mathbb{N} \rightarrow \mathbb{N}$  mit  $div_2(n) = \lfloor n/2 \rfloor$  Turing-berechenbar?

## BERECHENBARKEIT DER DIVISION DURCH 2

Ist  $div_2: \mathbb{N} \rightarrow \mathbb{N}$  mit  $div_2(n) = \lfloor n/2 \rfloor$  Turing-berechenbar?

- Bei unärer Codierung muß  $\tau$  je zwei Einsen löschen und eine neue hinter dem Ende des Wortes schreiben

# BERECHENBARKEIT DER DIVISION DURCH 2

Ist  $div_2: \mathbb{N} \rightarrow \mathbb{N}$  mit  $div_2(n) = \lfloor n/2 \rfloor$  Turing-berechenbar?

- Bei unärer Codierung muß  $\tau$  je zwei Einsen löschen und eine neue hinter dem Ende des Wortes schreiben

$s$	$a$	$s'$	$a'$	$P$	
$s_0$	1	$s_1$	b	r	<i>Erste 1</i>
$s_0$	b	$s_6$	b	h	<i>keine erste 1</i>
$s_1$	1	$s_2$	b	r	<i>Zweite 1</i>
$s_1$	b	$s_6$	b	h	<i>keine zweite 1</i>
$s_2$	1	$s_2$	1	r	<i>Nach rechts zum Eingabeende</i>
$s_2$	b	$s_3$	b	r	<i>Ende der Eingabe</i>
$s_3$	1	$s_3$	1	r	<i>Nach rechts zum Ausgabeende</i>
$s_3$	b	$s_4$	1	l	<i>Ende der Ausgabe, 1 schreiben</i>
$s_4$	1	$s_4$	1	l	<i>Nach links über Ausgabe</i>
$s_4$	b	$s_5$	b	l	
$s_5$	1	$s_5$	1	l	<i>Nach links über Eingabe</i>
$s_5$	b	$s_0$	b	r	

# BERECHENBARKEIT DER DIVISION DURCH 2

Ist  $div_2: \mathbb{N} \rightarrow \mathbb{N}$  mit  $div_2(n) = \lfloor n/2 \rfloor$  Turing-berechenbar?

- Bei unärer Codierung muß  $\tau$  je zwei Einsen löschen und eine neue hinter dem Ende des Wortes schreiben

$s$	$a$	$s'$	$a'$	$P$	
$s_0$	1	$s_1$	b	r	<i>Erste 1</i>
$s_0$	b	$s_6$	b	h	<i>keine erste 1</i>
$s_1$	1	$s_2$	b	r	<i>Zweite 1</i>
$s_1$	b	$s_6$	b	h	<i>keine zweite 1</i>
$s_2$	1	$s_2$	1	r	<i>Nach rechts zum Eingabeende</i>
$s_2$	b	$s_3$	b	r	<i>Ende der Eingabe</i>
$s_3$	1	$s_3$	1	r	<i>Nach rechts zum Ausgabeende</i>
$s_3$	b	$s_4$	1	l	<i>Ende der Ausgabe, 1 schreiben</i>
$s_4$	1	$s_4$	1	l	<i>Nach links über Ausgabe</i>
$s_4$	b	$s_5$	b	l	
$s_5$	1	$s_5$	1	l	<i>Nach links über Eingabe</i>
$s_5$	b	$s_0$	b	r	

- Bei binärer Codierung muß  $\tau$  die letzte Ziffer löschen

- Vereinfachung für theoretische Analysen

- Binäres Bandalphabet  $\Gamma = \{1, b\}$
- Halbseitig unendliches Band
- Restriktivere Ausgabekonvention
- Endzustand statt Halteinstruktion

- Erweiterung des Modells für Programmierzwecke

- Unvollständige Tabellen für  $\delta$
- Mehrspurmaschinen
- Mehrkopfmachines
- Mehrbandmaschinen
- Mehrdimensionale Maschinen
- Unterprogramme

Alle Varianten führen zum gleichen Berechenbarkeitsbegriff

# EINFACHERE TURINGMASCHINENMODELLE

Kein Verlust der Ausdruckskraft  
Simulation normaler Turingmaschinen möglich



Kein Verlust der Ausdruckskraft

Simulation normaler Turingmaschinen möglich

- Halbseitig unendliches Band

Kein Verlust der Ausdruckskraft

Simulation normaler Turingmaschinen möglich

- **Halbseitig unendliches Band**

- **Simulation** eines beidseitig unendlichen Bands **durch Tupelalphabet**  $(a_l, a_r)$   
 $a_l$  repräsentiert die linke,  $a_r$  die rechte Bandhälfte

## Kein Verlust der Ausdruckskraft

### Simulation normaler Turingmaschinen möglich

- **Halbseitig unendliches Band**
  - Simulation eines beidseitig unendlichen Bands durch Tupelalphabet  $(a_l, a_r)$   
 $a_l$  repräsentiert die linke,  $a_r$  die rechte Bandhälfte
- **Binäres Bandalphabet**  $\Gamma = \{1, b\}$

## Kein Verlust der Ausdruckskraft

### Simulation normaler Turingmaschinen möglich

- **Halbseitig unendliches Band**

- Simulation eines beidseitig unendlichen Bands durch Tupelalphabet  $(a_l, a_r)$   
 $a_l$  repräsentiert die linke,  $a_r$  die rechte Bandhälfte

- **Binäres Bandalphabet  $\Gamma = \{1, b\}$**

- Binärcodierung beliebiger Alphabete als Strings über  $\{1b, 11\}$

## Kein Verlust der Ausdruckskraft

### Simulation normaler Turingmaschinen möglich

- **Halbseitig unendliches Band**

- Simulation eines beidseitig unendlichen Bands durch Tupelalphabet  $(a_l, a_r)$   
 $a_l$  repräsentiert die linke,  $a_r$  die rechte Bandhälfte

- **Binäres Bandalphabet  $\Gamma = \{1, b\}$**

- Binärcodierung beliebiger Alphabete als Strings über  $\{1b, 11\}$

- **Ausgabewort muß unter dem Kopf beginnen**

- Ausgabefunktion ist Bandinhalt vom Kopfsymbol bis zum ersten Blank.

## Kein Verlust der Ausdruckskraft

### Simulation normaler Turingmaschinen möglich

- **Halbseitig unendliches Band**

- Simulation eines beidseitig unendlichen Bands durch Tupelalphabet  $(a_l, a_r)$   
 $a_l$  repräsentiert die linke,  $a_r$  die rechte Bandhälfte

- **Binäres Bandalphabet  $\Gamma = \{1, b\}$**

- Binärcodierung beliebiger Alphabete als Strings über  $\{1b, 11\}$

- **Ausgabewort muß unter dem Kopf beginnen**

- Ausgabefunktion ist Bandinhalt vom Kopfsymbol bis zum ersten Blank.
- Ergänze Programm für  $\delta$  um Kopfbewegung zum Wortanfang.

## Kein Verlust der Ausdruckskraft

### Simulation normaler Turingmaschinen möglich

- **Halbseitig unendliches Band**
  - Simulation eines beidseitig unendlichen Bands durch Tupelalphabet  $(a_l, a_r)$   
 $a_l$  repräsentiert die linke,  $a_r$  die rechte Bandhälfte
- **Binäres Bandalphabet  $\Gamma = \{1, b\}$** 
  - Binärcodierung beliebiger Alphabete als Strings über  $\{1b, 11\}$
- **Ausgabewort muß unter dem Kopf beginnen**
  - Ausgabefunktion ist Bandinhalt vom Kopfsymbol bis zum ersten Blank.
  - Ergänze Programm für  $\delta$  um Kopfbewegung zum Wortanfang.
- **Fester Endzustand  $s_e$  statt Halteinstruktion**

## Kein Verlust der Ausdruckskraft

### Simulation normaler Turingmaschinen möglich

- **Halbseitig unendliches Band**

- Simulation eines beidseitig unendlichen Bands durch Tupelalphabet  $(a_l, a_r)$   
 $a_l$  repräsentiert die linke,  $a_r$  die rechte Bandhälfte

- **Binäres Bandalphabet  $\Gamma = \{1, b\}$**

- Binärcodierung beliebiger Alphabete als Strings über  $\{1b, 11\}$

- **Ausgabewort muß unter dem Kopf beginnen**

- Ausgabefunktion ist Bandinhalt vom Kopfsymbol bis zum ersten Blank.
- Ergänze Programm für  $\delta$  um Kopfbewegung zum Wortanfang.

- **Fester Endzustand  $s_e$  statt Halteinstruktion**

- Ändere  $\delta(s, a) = (s', a', h)$  in  $\delta(s, a) = (s_e, a', l)$



**Keine Erweiterung der Ausdruckskraft**  
**Simulation durch normale Turingmaschinen möglich**

Keine Erweiterung der Ausdruckskraft  
Simulation durch normale Turingmaschinen möglich

- Unvollständige Tabellen für  $\delta$

## Keine Erweiterung der Ausdruckskraft Simulation durch normale Turingmaschinen möglich

- Unvollständige Tabellen für  $\delta$ 
  - Ergänze nichtgenannte Einträge als  $\delta(s,a) = (s,a,h)$

## Keine Erweiterung der Ausdruckskraft Simulation durch normale Turingmaschinen möglich

- **Unvollständige Tabellen für  $\delta$** 
  - Ergänze nichtgenannte Einträge als  $\delta(s,a) = (s,a,h)$
- **Mehrspurmaschinen**

## Keine Erweiterung der Ausdruckskraft Simulation durch normale Turingmaschinen möglich

- **Unvollständige Tabellen für  $\delta$** 
  - Ergänze nichtgenannte Einträge als  $\delta(s,a) = (s,a,h)$
- **Mehrspurmaschinen**
  - Simulation von  $k$  Spuren durch **Tupelalphabet**  $(a_1, \dots, a_k)$   
 $a_i$  repräsentiert Spur  $i$

## Keine Erweiterung der Ausdruckskraft

## Simulation durch normale Turingmaschinen möglich

- **Unvollständige Tabellen für  $\delta$** 
  - Ergänze nichtgenannte Einträge als  $\delta(s,a) = (s,a,h)$
- **Mehrspurmaschinen**
  - Simulation von  $k$  Spuren durch **Tupelalphabet**  $(a_1, \dots, a_k)$   
 $a_i$  repräsentiert Spur  $i$
- **Mehrbandmaschinen**

## Keine Erweiterung der Ausdruckskraft Simulation durch normale Turingmaschinen möglich

- **Unvollständige Tabellen für  $\delta$** 
  - Ergänze nichtgenannte Einträge als  $\delta(s,a) = (s,a,h)$
- **Mehrspurmaschinen**
  - Simulation von  $k$  Spuren durch **Tupelalphabet**  $(a_1, \dots, a_k)$   
 $a_i$  repräsentiert Spur  $i$
- **Mehrbandmaschinen**
  - Simulation durch **Mehrspurmaschine und Marker** für Kopfpositionen

## Keine Erweiterung der Ausdruckskraft Simulation durch normale Turingmaschinen möglich

- **Unvollständige Tabellen für  $\delta$** 
  - Ergänze nichtgenannte Einträge als  $\delta(s,a) = (s,a,h)$
- **Mehrspurmaschinen**
  - Simulation von  $k$  Spuren durch **Tupelalphabet**  $(a_1, \dots, a_k)$   
 $a_i$  repräsentiert Spur  $i$
- **Mehrbandmaschinen**
  - Simulation durch **Mehrspurmaschine und Marker** für Kopfpositionen
- **Mehrkopfmaschinen**



## Keine Erweiterung der Ausdruckskraft Simulation durch normale Turingmaschinen möglich

- **Unvollständige Tabellen für  $\delta$** 
  - Ergänze nichtgenannte Einträge als  $\delta(s,a) = (s,a,h)$
- **Mehrspurmaschinen**
  - Simulation von  $k$  Spuren durch **Tupelalphabet**  $(a_1, \dots, a_k)$   
 $a_i$  repräsentiert Spur  $i$
- **Mehrbandmaschinen**
  - Simulation durch **Mehrspurmaschine und Marker** für Kopfpositionen
- **Mehrkopfmaschinen**
  - Speichere **Kopfpositionen** auf separatem Band and verarbeite sequentiell

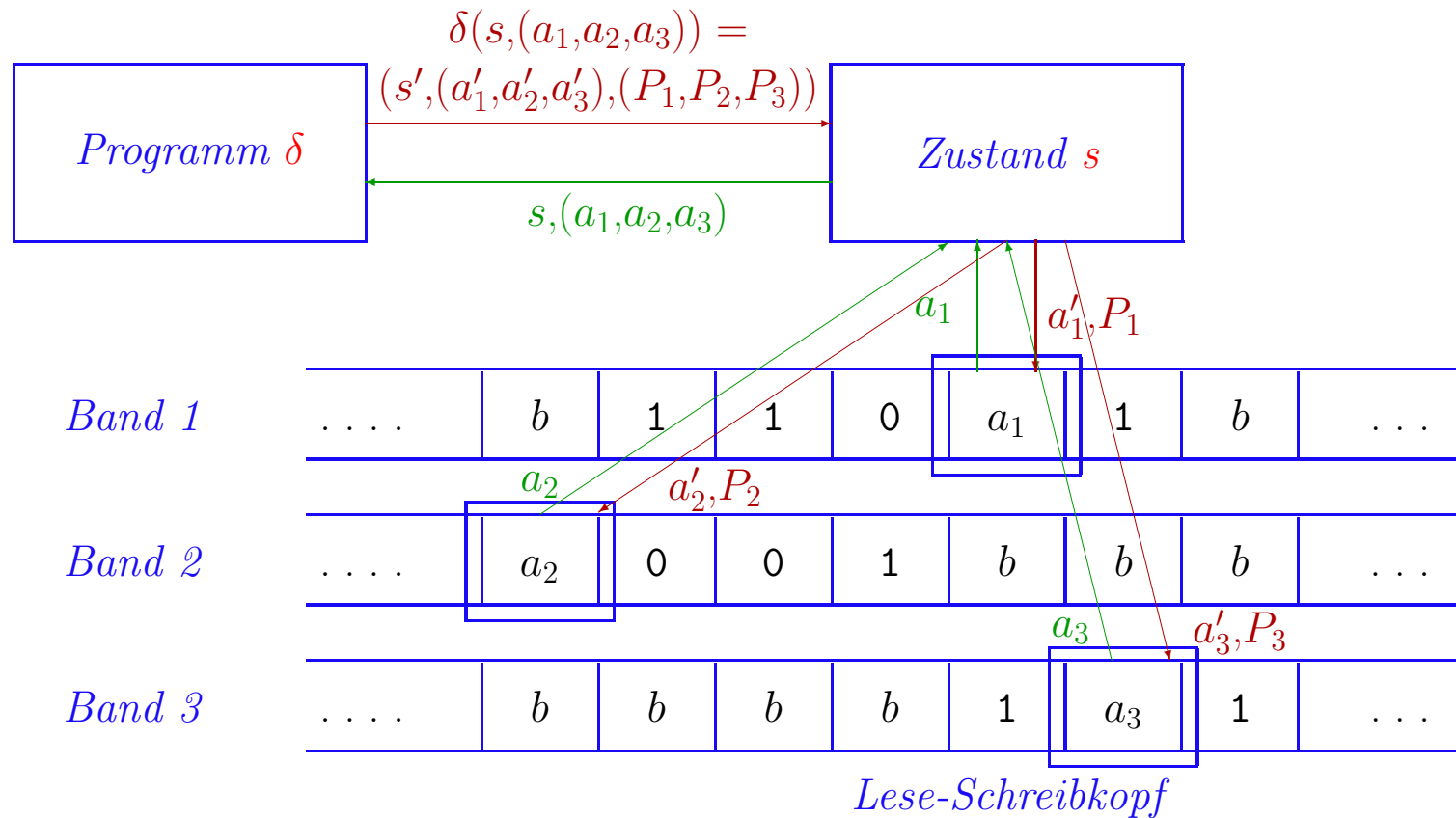
## Keine Erweiterung der Ausdruckskraft Simulation durch normale Turingmaschinen möglich

- **Unvollständige Tabellen für  $\delta$** 
  - Ergänze nichtgenannte Einträge als  $\delta(s,a) = (s,a,h)$
- **Mehrspurmaschinen**
  - Simulation von  $k$  Spuren durch **Tupelalphabet**  $(a_1, \dots, a_k)$   
 $a_i$  repräsentiert Spur  $i$
- **Mehrbandmaschinen**
  - Simulation durch **Mehrspurmaschine und Marker** für Kopfpositionen
- **Mehrkopfmaschinen**
  - Speichere **Kopfpositionen** auf separatem Band and verarbeite sequentiell
- **Unterprogramme**

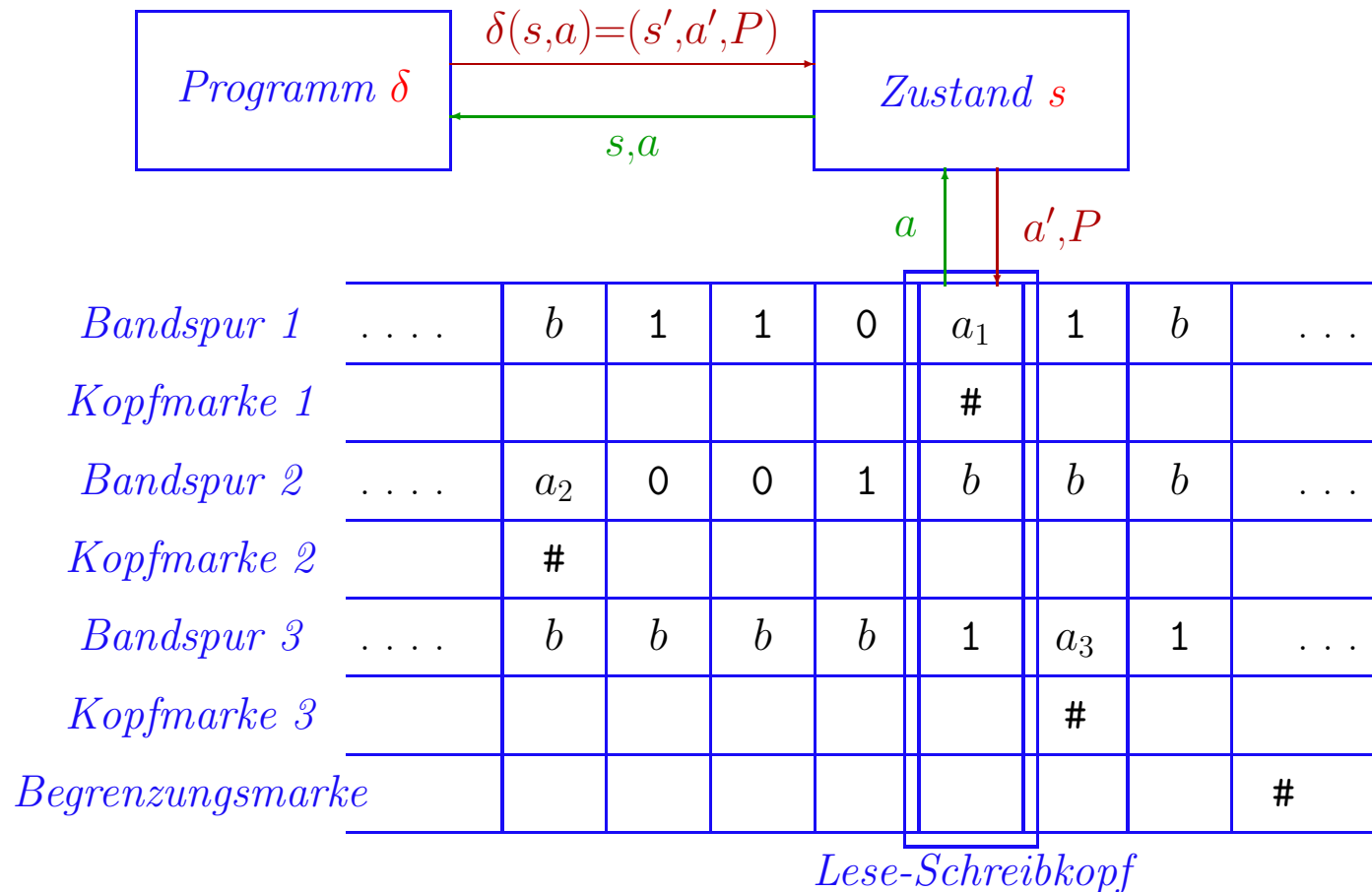
## Keine Erweiterung der Ausdruckskraft Simulation durch normale Turingmaschinen möglich

- **Unvollständige Tabellen für  $\delta$** 
  - Ergänze nichtgenannte Einträge als  $\delta(s,a) = (s,a,h)$
- **Mehrspurmaschinen**
  - Simulation von  $k$  Spuren durch **Tupelalphabet**  $(a_1, \dots, a_k)$   
 $a_i$  repräsentiert Spur  $i$
- **Mehrbandmaschinen**
  - Simulation durch **Mehrspurmaschine und Marker** für Kopfpositionen
- **Mehrkopfmaschinen**
  - Speichere **Kopfpositionen** auf separatem Band and verarbeite sequentiell
- **Unterprogramme**
  - Speichere **Argumente und Rückgabewerte** auf separatem Band.

# MEHRBANDMASCHINE



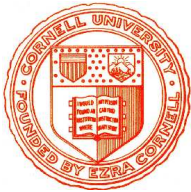
# SIMULATION EINER MEHRBANDMASCHINE



## • Verarbeite Bänder sequentiell

- Lesen: Suche Begrenzungs-marke, laufe zurück bis zu Kopfmarken
- Schreiben und Kopfbewegung analog
- Codiere Symbole und Kopfinstruktionen im Zustand
- Simulation benötigt **quadratische Zeit**

# Theoretische Informatik



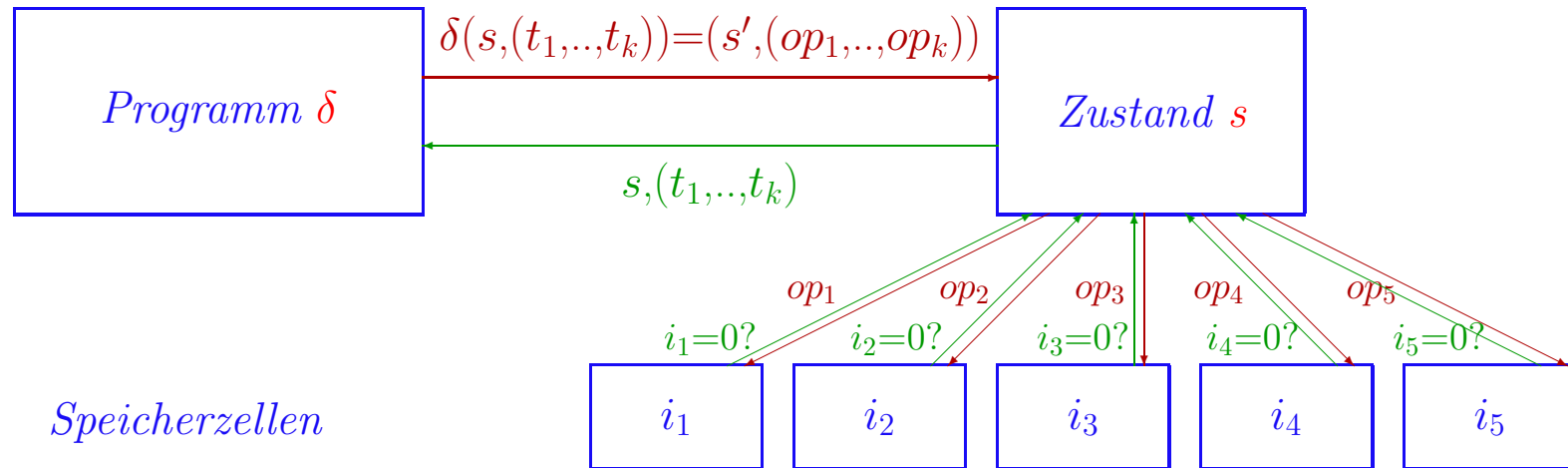
## Einheit 1.2

### Registermaschinen

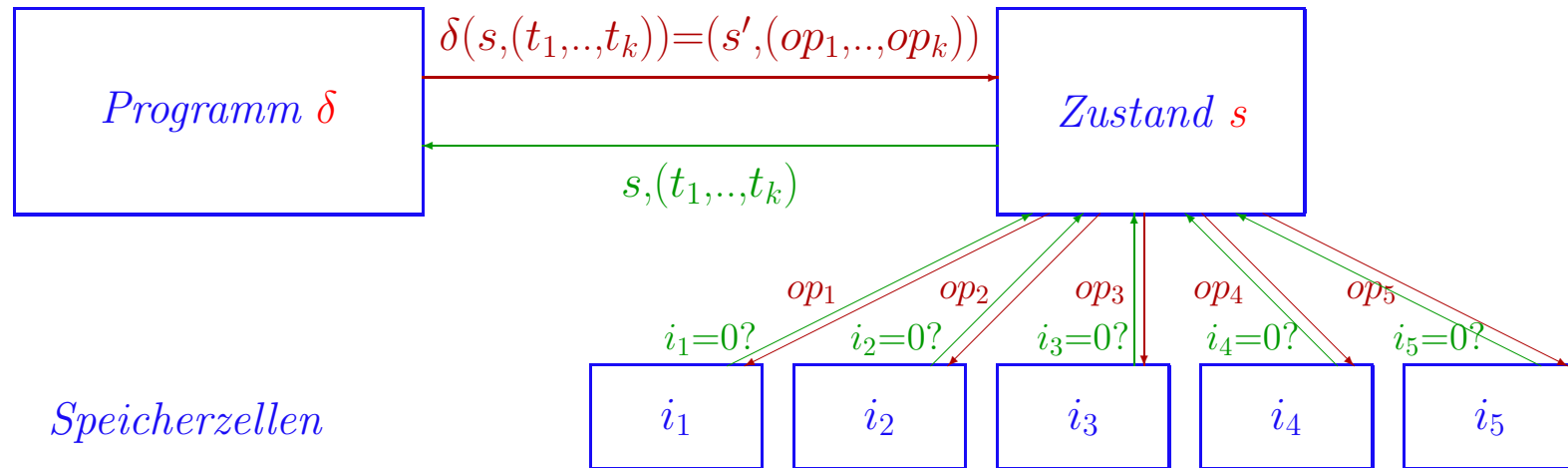


1. Arbeitsweise
2. Formale Semantik
3. Register-Berechenbarkeit
4. Äquivalenz zu Turingmaschinen

# REGISTERMASCHINEN



# REGISTERMASCHINEN

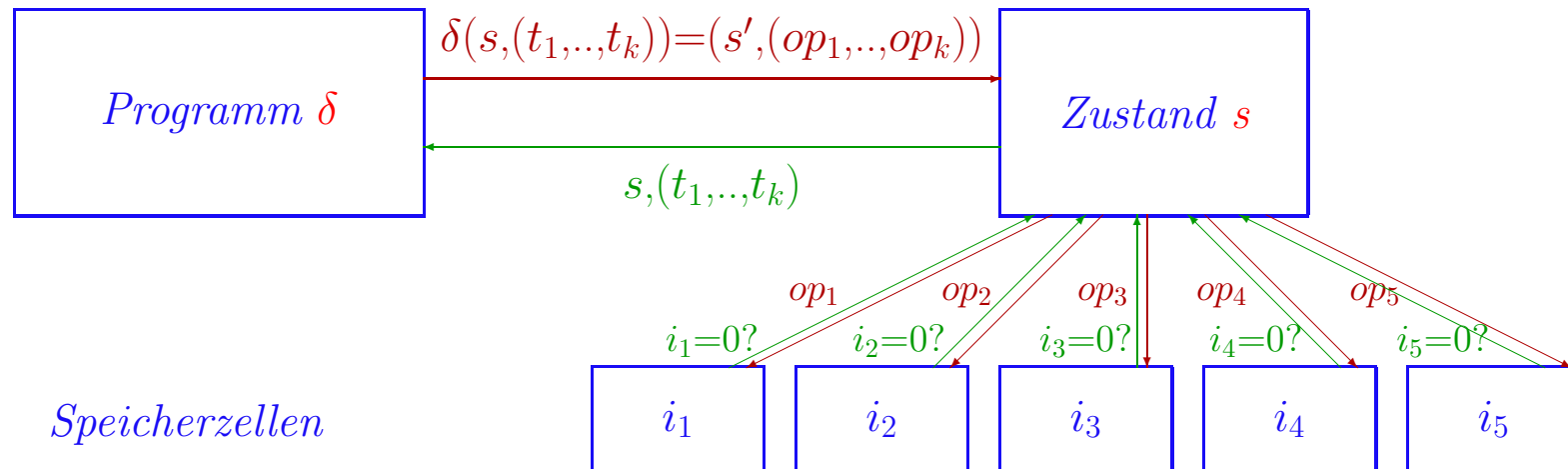


## ● Standardarchitektur von Einprozessorsystemen

- Direkter und simultaner Speicherzugriff
- Speicherzellen enthalten natürliche Zahlen
- Keine Ein/Ausgabe, sehr einfacher Befehlssatz



# REGISTERMASCHINEN



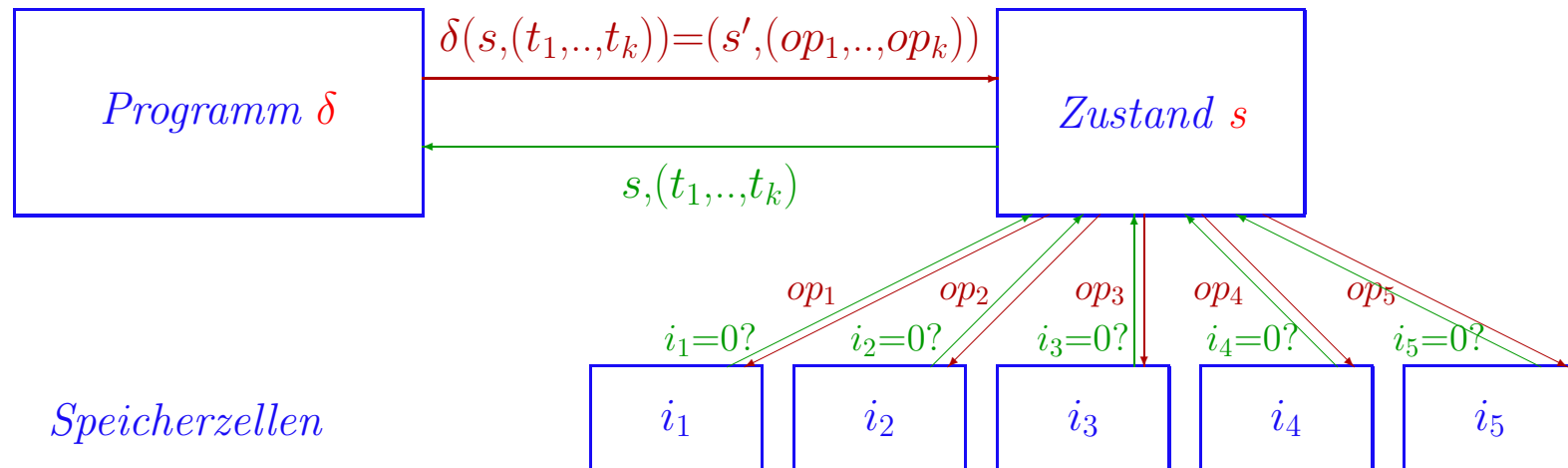
## ● Standardarchitektur von Einprozessorsystemen

- 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

# REGISTERMASCHINEN



## ● Standardarchitektur von Einprozessorsystemen

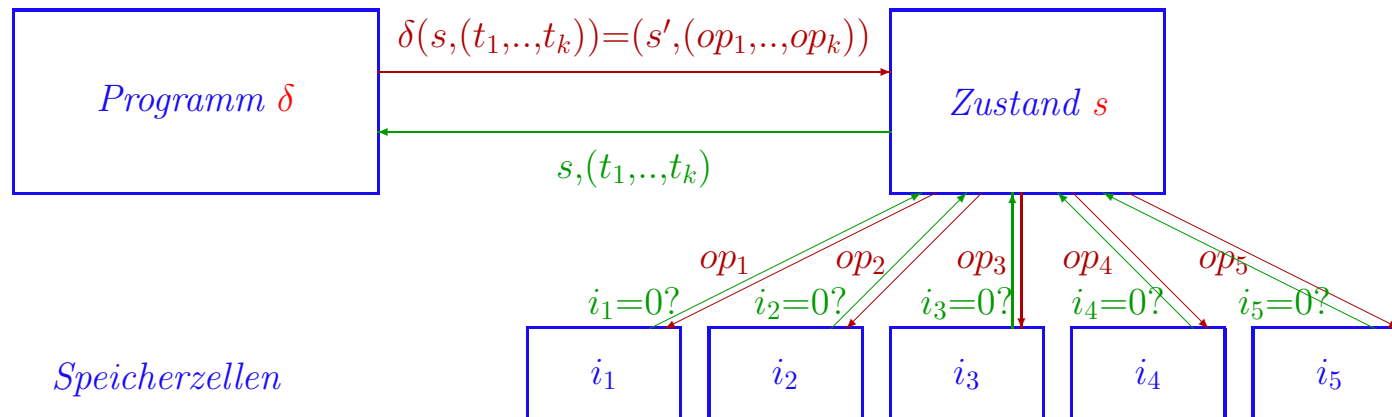
- Direkter und simultaner Speicherzugriff
- Speicherzellen enthalten natürliche Zahlen
- Keine Ein/Ausgabe, sehr einfacher Befehlssatz

## ● Unterschiede zur Turingmaschine

- Endlicher Speicher, aber unendlicher Bereich für Werte von Zellen

**Achtung! Modelle in Literatur oft flexibler**

# REGISTERMASCHINEN – MATHEMATISCH



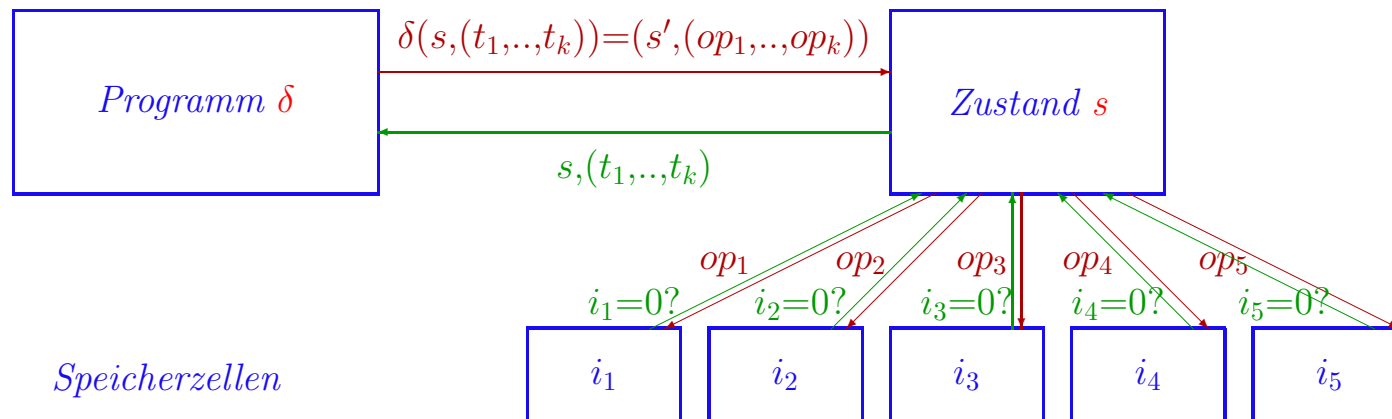
Eine **Registermaschine** ist ein 5-Tupel  $\rho = (S, k, \delta, s_0, F)$

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

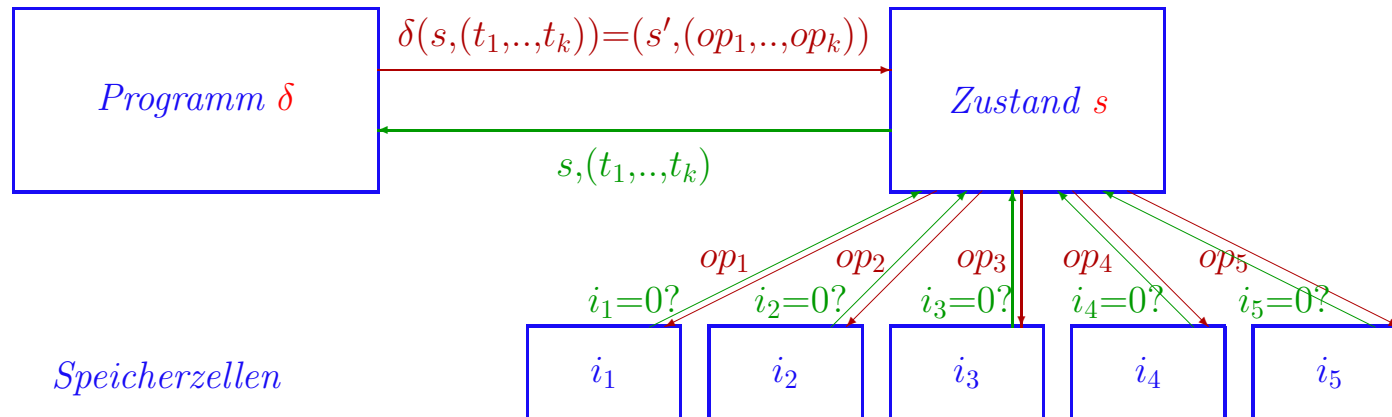
Eingabe: Zustand + Testergebnisse:  $t_j = \text{sign}(i_j) = \begin{cases} 0 & \text{falls } i_j=0, \\ 1 & \text{falls } i_j>0 \end{cases}$

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

# ARBEITSWEISE VON REGISTERMASCHINEN



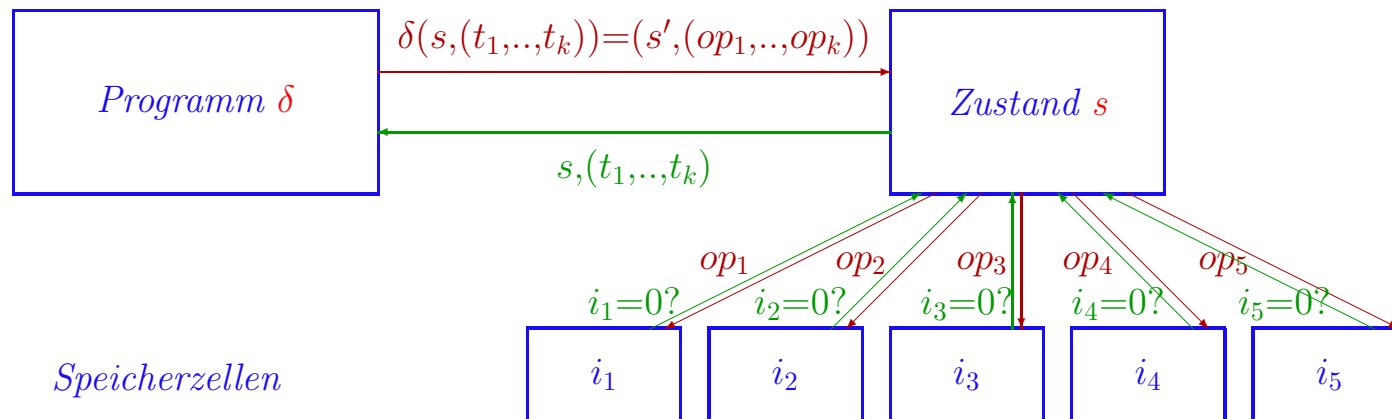
# ARBEITSWEISE VON REGISTERMASCHINEN



## ● Anfangssituation

- Eingabezahl  $n$  steht im ersten Register

# ARBEITSWEISE VON REGISTERMASCHINEN



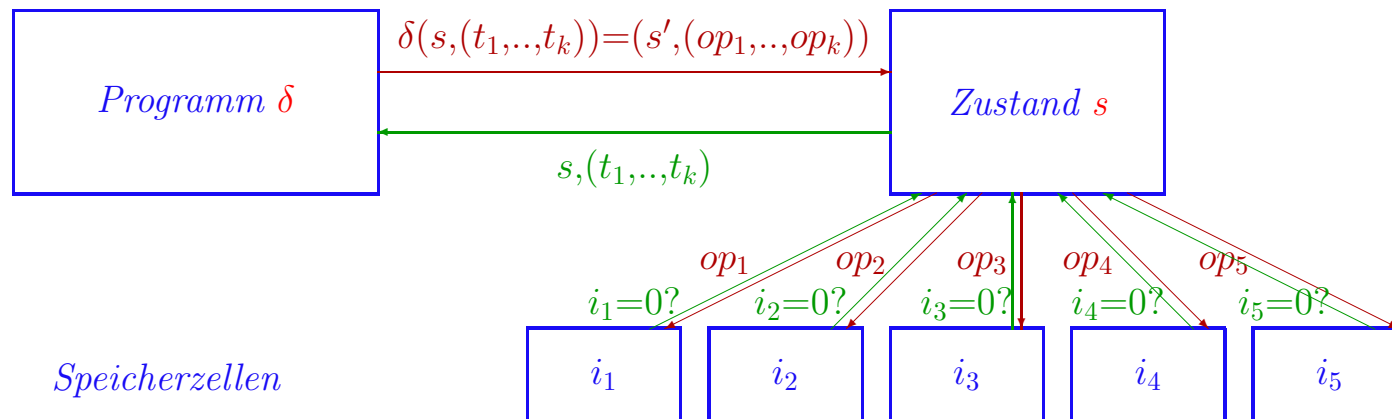
## ● Anfangssituation

- Eingabezahl  $n$  steht im ersten Register

## ● Arbeitsschritt

- Inhalte der Register  $i_1, \dots, i_k$  lesen und mit  $\text{sign}(i_j)$  auf Null testen
- Zustand  $s$  und Testergebnisse  $t_1, \dots, t_k$  als Argumente an  $\delta$  geben
- $\delta(s, (t_1, \dots, t_k)) = (s', (op_1, \dots, op_k))$  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

## ● Arbeitsschritt

- Inhalte der Register  $i_1, \dots, i_k$  lesen und mit  $\text{sign}(i_j)$  auf Null testen
- Zustand  $s$  und Testergebnisse  $t_1, \dots, t_k$  als Argumente an  $\delta$  geben
- $\delta(s, (t_1, \dots, t_k)) = (s', (op_1, \dots, op_k))$  bestimmen
- Neuer Zustand  $s'$ , Register  $j$  gemäß Operation  $op_j$  modifizieren
- Stop wenn  $s'$  Endzustand ist

## ● Ergebnis

- Inhalt des ersten Registers

# SEMANTIK VON REGISTERMASCHINEN

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



# SEMANTIK VON REGISTERMASCHINEN

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

# SEMANTIK VON REGISTERMASCHINEN

- $K_\rho \equiv$  Menge aller **Konfigurationen**  $\kappa = (s, (i_1, \dots, i_k))$  von  $\rho$  mit
  - $s \in S$  aktueller Zustand
  - $i_j \in \mathbb{N}$  Inhalt des Registers  $j$
- Anfangskonfiguration  $\alpha: \mathbb{N} \rightarrow K_\rho$ :  $\alpha(n) = (s_0, (n, 0, \dots, 0))$
- Nachfolgekonfiguration:  $\hat{\delta}: K_\rho \rightarrow K_\rho$ 
  - Für  $\kappa = (s, (i_1, \dots, i_k))$  mit  $\delta(s, (\text{sign}(i_1), \dots, \text{sign}(i_k))) = (s', (op_1, \dots, op_k))$  ist
$$\hat{\delta}(\kappa) = (s', (i'_1, \dots, i'_k)), \text{ wobei } i'_j = \begin{cases} 0 & \text{falls } i_j = 0 \text{ und } op_j = -1, \\ i_j + op_j & \text{sonst} \end{cases}$$

# SEMANTIK VON REGISTERMASCHINEN

- $K_\rho \equiv$  Menge aller **Konfigurationen**  $\kappa = (s, (i_1, \dots, i_k))$  von  $\rho$  mit
  - $s \in S$  aktueller Zustand
  - $i_j \in \mathbb{N}$  Inhalt des Registers  $j$
- Anfangskonfiguration  $\alpha: \mathbb{N} \rightarrow K_\rho$ :  $\alpha(n) = (s_0, (n, 0, \dots, 0))$
- Nachfolgekonfiguration:  $\hat{\delta}: K_\rho \rightarrow K_\rho$ 
  - Für  $\kappa = (s, (i_1, \dots, i_k))$  mit  $\delta(s, (\text{sign}(i_1), \dots, \text{sign}(i_k))) = (s', (op_1, \dots, op_k))$  ist
$$\hat{\delta}(\kappa) = (s', (i'_1, \dots, i'_k)), \text{ wobei } i'_j = \begin{cases} 0 & \text{falls } i_j = 0 \text{ und } op_j = -1, \\ i_j + op_j & \text{sonst} \end{cases}$$
- Ausgabefunktion  $\omega: K_\rho \rightarrow \mathbb{N}$ :  $\omega(s, (i_1, \dots, i_k)) = i_1$

# SEMANTIK VON REGISTERMASCHINEN

- $K_\rho \equiv$  Menge aller **Konfigurationen**  $\kappa = (s, (i_1, \dots, i_k))$  von  $\rho$  mit
  - $s \in S$  aktueller Zustand
  - $i_j \in \mathbb{N}$  Inhalt des Registers  $j$
- Anfangskonfiguration  $\alpha: \mathbb{N} \rightarrow K_\rho$ :  $\alpha(n) = (s_0, (n, 0, \dots, 0))$
- Nachfolgekonfiguration:  $\hat{\delta}: K_\rho \rightarrow K_\rho$ 
  - Für  $\kappa = (s, (i_1, \dots, i_k))$  mit  $\delta(s, (\text{sign}(i_1), \dots, \text{sign}(i_k))) = (s', (op_1, \dots, op_k))$  ist
$$\hat{\delta}(\kappa) = (s', (i'_1, \dots, i'_k)), \text{ wobei } i'_j = \begin{cases} 0 & \text{falls } i_j = 0 \text{ und } op_j = -1, \\ i_j + op_j & \text{sonst} \end{cases}$$
- Ausgabefunktion  $\omega: K_\rho \rightarrow \mathbb{N}$ :  $\omega(s, (i_1, \dots, i_k)) = i_1$
- Die von  $\rho$  **berechnete Funktion**  $h_\rho: \mathbb{N} \rightarrow \mathbb{N}$  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

- $K_\rho \equiv$  Menge aller **Konfigurationen**  $\kappa = (s, (i_1, \dots, i_k))$  von  $\rho$  mit
  - $s \in S$  aktueller Zustand
  - $i_j \in \mathbb{N}$  Inhalt des Registers  $j$
- Anfangskonfiguration  $\alpha: \mathbb{N} \rightarrow K_\rho$ :  $\alpha(n) = (s_0, (n, 0, \dots, 0))$
- Nachfolgekonfiguration:  $\hat{\delta}: K_\rho \rightarrow K_\rho$ 
  - Für  $\kappa = (s, (i_1, \dots, i_k))$  mit  $\delta(s, (\text{sign}(i_1), \dots, \text{sign}(i_k))) = (s', (op_1, \dots, op_k))$  ist
 
$$\hat{\delta}(\kappa) = (s', (i'_1, \dots, i'_k)), \text{ wobei } i'_j = \begin{cases} 0 & \text{falls } i_j = 0 \text{ und } op_j = -1, \\ i_j + op_j & \text{sonst} \end{cases}$$
- Ausgabefunktion  $\omega: K_\rho \rightarrow \mathbb{N}$ :  $\omega(s, (i_1, \dots, i_k)) = i_1$
- Die von  $\rho$  **berechnete Funktion**  $h_\rho: \mathbb{N} \rightarrow \mathbb{N}$  ist definiert durch

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

Definitionsbereich von  $\rho$  ist  $\{n \in \mathbb{N} \mid h_\rho(n) \neq \perp\}$ ,

Wertebereich von  $\rho$  ist  $\{m \in \mathbb{N} \mid \exists n \in \mathbb{N} \ h_\rho(n) = m\}$

# BEISPIELE FÜR REGISTERMASCHINEN

•  $\rho_1 = (\{s_0, s_1\}, 1, \delta_1, s_0, \{s_1\})$  mit  $\delta_1 =$

$s$	$t_1$	$s'$	$op_1$
$s_0$	0	$s_1$	+1
$s_0$	1	$s_0$	-1

# BEISPIELE FÜR REGISTERMASCHINEN

•  $\rho_1 = (\{s_0, s_1\}, 1, \delta_1, s_0, \{s_1\})$  mit  $\delta_1 =$

$s$	$t_1$	$s'$	$op_1$
$s_0$	0	$s_1$	+1
$s_0$	1	$s_0$	-1

Zähle Eingabewert  $n$  auf Null herunter und addiere Eins

# BEISPIELE FÜR REGISTERMASCHINEN

- $\rho_1 = (\{s_0, s_1\}, 1, \delta_1, s_0, \{s_1\})$  mit  $\delta_1 =$

$s$	$t_1$	$s'$	$op_1$
$s_0$	0	$s_1$	+1
$s_0$	1	$s_0$	-1

Zähle Eingabewert  $n$  auf Null herunter und addiere Eins

- **Mathematische Analyse:**



# BEISPIELE FÜR REGISTERMASCHINEN

•  $\rho_1 = (\{\mathbf{s}_0, \mathbf{s}_1\}, 1, \delta_1, \mathbf{s}_0, \{\mathbf{s}_1\})$  mit  $\delta_1 =$

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

Zähle Eingabewert  $n$  auf Null herunter und addiere Eins

• **Mathematische Analyse:**

– Anfangskonfiguration:  $\alpha(n) = (\mathbf{s}_0, n)$

# BEISPIELE FÜR REGISTERMASCHINEN

•  $\rho_1 = (\{\mathbf{s}_0, \mathbf{s}_1\}, 1, \delta_1, \mathbf{s}_0, \{\mathbf{s}_1\})$  mit  $\delta_1 =$

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

Zähle Eingabewert  $n$  auf Null herunter und addiere Eins

• **Mathematische Analyse:**

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

# BEISPIELE FÜR REGISTERMASCHINEN

•  $\rho_1 = (\{\mathbf{s}_0, \mathbf{s}_1\}, 1, \delta_1, \mathbf{s}_0, \{\mathbf{s}_1\})$  mit  $\delta_1 =$

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

Zähle Eingabewert  $n$  auf Null herunter und addiere Eins

• **Mathematische Analyse:**

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

# BEISPIELE FÜR REGISTERMASCHINEN

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

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

Zähle Eingabewert  $n$  auf Null herunter und addiere Eins

- **Mathematische Analyse:**

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

# BEISPIELE FÜR REGISTERMASCHINEN

•  $\rho_1 = (\{\mathbf{s}_0, \mathbf{s}_1\}, 1, \delta_1, \mathbf{s}_0, \{\mathbf{s}_1\})$  mit  $\delta_1 =$

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

Zähle Eingabewert  $n$  auf Null herunter und addiere Eins

• **Mathematische Analyse:**

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

# BEISPIELE FÜR REGISTERMASCHINEN

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

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

Zähle Eingabewert  $n$  auf Null herunter und addiere Eins

- **Mathematische Analyse:**

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



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

# BEISPIELE FÜR REGISTERMASCHINEN II

•  $\rho_2 = (\{s_0, s_1, s_2, s_3, s_4\}, 2, \delta_2, s_0, \{s_4\})$  mit  $\delta_2 =$

$s$	$t_1$	$t_2$	$s'$	$op_1$	$op_2$
$s_0$	0	0	$s_4$	0	0
$s_0$	0	1	$s_2$	0	0
$s_0$	1	*	$s_1$	-1	+1
$s_1$	*	*	$s_0$	0	+1
$s_2$	*	0	$s_4$	0	0
$s_2$	*	1	$s_3$	+1	-1
$s_3$	*	*	$s_2$	+1	0

# BEISPIELE FÜR REGISTERMASCHINEN II

•  $\rho_2 = (\{s_0, s_1, s_2, s_3, s_4\}, 2, \delta_2, s_0, \{s_4\})$  mit  $\delta_2 =$

$s$	$t_1$	$t_2$	$s'$	$op_1$	$op_2$
$s_0$	0	0	$s_4$	0	0
$s_0$	0	1	$s_2$	0	0
$s_0$	1	*	$s_1$	-1	+1
$s_1$	*	*	$s_0$	0	+1
$s_2$	*	0	$s_4$	0	0
$s_2$	*	1	$s_3$	+1	-1
$s_3$	*	*	$s_2$	+1	0

• Analyse



# BEISPIELE FÜR REGISTERMASCHINEN II

•  $\rho_2 = (\{s_0, s_1, s_2, s_3, s_4\}, 2, \delta_2, s_0, \{s_4\})$  mit  $\delta_2 =$

$s$	$t_1$	$t_2$	$s'$	$op_1$	$op_2$
$s_0$	0	0	$s_4$	0	0
$s_0$	0	1	$s_2$	0	0
$s_0$	1	*	$s_1$	-1	+1
$s_1$	*	*	$s_0$	0	+1
$s_2$	*	0	$s_4$	0	0
$s_2$	*	1	$s_3$	+1	-1
$s_3$	*	*	$s_2$	+1	0

• Analyse

$$n \xrightarrow{\alpha} (s_0, n, 0)$$

# BEISPIELE FÜR REGISTERMASCHINEN II

•  $\rho_2 = (\{s_0, s_1, s_2, s_3, s_4\}, 2, \delta_2, s_0, \{s_4\})$  mit  $\delta_2 =$

$s$	$t_1$	$t_2$	$s'$	$op_1$	$op_2$
$s_0$	0	0	$s_4$	0	0
$s_0$	0	1	$s_2$	0	0
$s_0$	1	*	$s_1$	-1	+1
$s_1$	*	*	$s_0$	0	+1
$s_2$	*	0	$s_4$	0	0
$s_2$	*	1	$s_3$	+1	-1
$s_3$	*	*	$s_2$	+1	0

## • Analyse

$$\begin{aligned}
 n &\xrightarrow{\alpha} (s_0, n, 0) \\
 &\xrightarrow{\delta} (s_1, n-1, 1)
 \end{aligned}$$

# BEISPIELE FÜR REGISTERMASCHINEN II

•  $\rho_2 = (\{s_0, s_1, s_2, s_3, s_4\}, 2, \delta_2, s_0, \{s_4\})$  mit  $\delta_2 =$

$s$	$t_1$	$t_2$	$s'$	$op_1$	$op_2$
$s_0$	0	0	$s_4$	0	0
$s_0$	0	1	$s_2$	0	0
$s_0$	1	*	$s_1$	-1	+1
$s_1$	*	*	$s_0$	0	+1
$s_2$	*	0	$s_4$	0	0
$s_2$	*	1	$s_3$	+1	-1
$s_3$	*	*	$s_2$	+1	0

## • Analyse

$$\begin{array}{lcl}
 n & \xrightarrow{\alpha} & (s_0, n, 0) \\
 & \xrightarrow{\delta} & (s_1, n-1, 1) \quad \xrightarrow{\delta} \quad (s_0, n-1, 2)
 \end{array}$$

# BEISPIELE FÜR REGISTERMASCHINEN II

•  $\rho_2 = (\{s_0, s_1, s_2, s_3, s_4\}, 2, \delta_2, s_0, \{s_4\})$  mit  $\delta_2 =$

$s$	$t_1$	$t_2$	$s'$	$op_1$	$op_2$
$s_0$	0	0	$s_4$	0	0
$s_0$	0	1	$s_2$	0	0
$s_0$	1	*	$s_1$	-1	+1
$s_1$	*	*	$s_0$	0	+1
$s_2$	*	0	$s_4$	0	0
$s_2$	*	1	$s_3$	+1	-1
$s_3$	*	*	$s_2$	+1	0

## • Analyse

$$\begin{array}{l}
 n \xrightarrow{\alpha} (s_0, n, 0) \\
 \xrightarrow{\delta} (s_1, n-1, 1) \quad \xrightarrow{\delta} (s_0, n-1, 2) \quad \xrightarrow{\delta} \dots \xrightarrow{\delta} (s_2, 0, 2n)
 \end{array}$$

# BEISPIELE FÜR REGISTERMASCHINEN II

•  $\rho_2 = (\{s_0, s_1, s_2, s_3, s_4\}, 2, \delta_2, s_0, \{s_4\})$  mit  $\delta_2 =$

$s$	$t_1$	$t_2$	$s'$	$op_1$	$op_2$
$s_0$	0	0	$s_4$	0	0
$s_0$	0	1	$s_2$	0	0
$s_0$	1	*	$s_1$	-1	+1
$s_1$	*	*	$s_0$	0	+1
$s_2$	*	0	$s_4$	0	0
$s_2$	*	1	$s_3$	+1	-1
$s_3$	*	*	$s_2$	+1	0

## • Analyse

$$\begin{array}{l}
 n \xrightarrow{\alpha} (s_0, n, 0) \\
 \xrightarrow{\delta} (s_1, n-1, 1) \quad \xrightarrow{\delta} (s_0, n-1, 2) \quad \xrightarrow{\delta} \dots \xrightarrow{\delta} (s_2, 0, 2n) \\
 \xrightarrow{\delta} (s_3, 1, 2n-1)
 \end{array}$$

# BEISPIELE FÜR REGISTERMASCHINEN II

•  $\rho_2 = (\{s_0, s_1, s_2, s_3, s_4\}, 2, \delta_2, s_0, \{s_4\})$  mit  $\delta_2 =$

$s$	$t_1$	$t_2$	$s'$	$op_1$	$op_2$
$s_0$	0	0	$s_4$	0	0
$s_0$	0	1	$s_2$	0	0
$s_0$	1	*	$s_1$	-1	+1
$s_1$	*	*	$s_0$	0	+1
$s_2$	*	0	$s_4$	0	0
$s_2$	*	1	$s_3$	+1	-1
$s_3$	*	*	$s_2$	+1	0

## • Analyse

$$\begin{array}{lcl}
 n & \xrightarrow{\alpha} & (s_0, n, 0) \\
 & \xrightarrow{\delta} & (s_1, n-1, 1) \quad \xrightarrow{\delta} (s_0, n-1, 2) \quad \xrightarrow{\delta} \dots \xrightarrow{\delta} (s_2, 0, 2n) \\
 & \xrightarrow{\delta} & (s_3, 1, 2n-1) \quad \xrightarrow{\delta} (s_2, 2, 2n-1)
 \end{array}$$

# BEISPIELE FÜR REGISTERMASCHINEN II

•  $\rho_2 = (\{s_0, s_1, s_2, s_3, s_4\}, 2, \delta_2, s_0, \{s_4\})$  mit  $\delta_2 =$

$s$	$t_1$	$t_2$	$s'$	$op_1$	$op_2$
$s_0$	0	0	$s_4$	0	0
$s_0$	0	1	$s_2$	0	0
$s_0$	1	*	$s_1$	-1	+1
$s_1$	*	*	$s_0$	0	+1
$s_2$	*	0	$s_4$	0	0
$s_2$	*	1	$s_3$	+1	-1
$s_3$	*	*	$s_2$	+1	0

## • Analyse

$$\begin{array}{lcl}
 n & \xrightarrow{\alpha} & (s_0, n, 0) \\
 & \xrightarrow{\delta} & (s_1, n-1, 1) \quad \xrightarrow{\delta} (s_0, n-1, 2) \quad \xrightarrow{\delta} \dots \xrightarrow{\delta} (s_2, 0, 2n) \\
 & \xrightarrow{\delta} & (s_3, 1, 2n-1) \quad \xrightarrow{\delta} (s_2, 2, 2n-1) \quad \xrightarrow{\delta} \dots \xrightarrow{\delta} (s_4, 4n, 0)
 \end{array}$$

# BEISPIELE FÜR REGISTERMASCHINEN II

•  $\rho_2 = (\{s_0, s_1, s_2, s_3, s_4\}, 2, \delta_2, s_0, \{s_4\})$  mit  $\delta_2 =$

$s$	$t_1$	$t_2$	$s'$	$op_1$	$op_2$
$s_0$	0	0	$s_4$	0	0
$s_0$	0	1	$s_2$	0	0
$s_0$	1	*	$s_1$	-1	+1
$s_1$	*	*	$s_0$	0	+1
$s_2$	*	0	$s_4$	0	0
$s_2$	*	1	$s_3$	+1	-1
$s_3$	*	*	$s_2$	+1	0

## • Analyse

$$\begin{array}{l}
 n \xrightarrow{\alpha} (s_0, n, 0) \\
 \xrightarrow{\delta} (s_1, n-1, 1) \quad \xrightarrow{\delta} (s_0, n-1, 2) \quad \xrightarrow{\delta} \dots \xrightarrow{\delta} (s_2, 0, 2n) \\
 \xrightarrow{\delta} (s_3, 1, 2n-1) \quad \xrightarrow{\delta} (s_2, 2, 2n-1) \quad \xrightarrow{\delta} \dots \xrightarrow{\delta} (s_4, 4n, 0) \\
 \xrightarrow{\omega} 4n
 \end{array}$$



# BEISPIELE FÜR REGISTERMASCHINEN II

- $\rho_2 = (\{s_0, s_1, s_2, s_3, s_4\}, 2, \delta_2, s_0, \{s_4\})$  mit  $\delta_2 =$ 

$s$	$t_1$	$t_2$	$s'$	$op_1$	$op_2$
$s_0$	0	0	$s_4$	0	0
$s_0$	0	1	$s_2$	0	0
$s_0$	1	*	$s_1$	-1	+1
$s_1$	*	*	$s_0$	0	+1
$s_2$	*	0	$s_4$	0	0
$s_2$	*	1	$s_3$	+1	-1
$s_3$	*	*	$s_2$	+1	0

## • Analyse

$$\begin{array}{l}
 n \xrightarrow{\alpha} (s_0, n, 0) \\
 \xrightarrow{\delta} (s_1, n-1, 1) \quad \xrightarrow{\delta} (s_0, n-1, 2) \quad \xrightarrow{\delta} \dots \xrightarrow{\delta} (s_2, 0, 2n) \\
 \xrightarrow{\delta} (s_3, 1, 2n-1) \quad \xrightarrow{\delta} (s_2, 2, 2n-1) \quad \xrightarrow{\delta} \dots \xrightarrow{\delta} (s_4, 4n, 0) \\
 \xrightarrow{\omega} 4n
 \end{array}$$



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

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

# REGISTER-BERECHENBARKEIT

- $f: \mathbb{N} \rightarrow \mathbb{N}$   **$\text{RM}_k$ -berechenbar**

- Es gibt eine Registermaschine  $\rho = (S, k, \delta, s_0, F)$  mit  $h_\rho = f$

- $f: \mathbb{N}^m \rightarrow \mathbb{N}^n$   **$\text{RM}_k$ -berechenbar** ( $k \geq \max(m, n)$ )

- Es gibt eine Registermaschine  $\rho = (S, k, \delta, s_0, F)$  mit  $h_\rho = f$  und

- Anfangskonfiguration  $\alpha^m: \mathbb{N}^m \rightarrow K_\rho$ :  $\alpha^m(n_1, \dots, n_m) = (s_0, (n_1, \dots, n_m, 0, \dots, 0))$

- Ausgabefunktion  $\omega^n: K_\rho \rightarrow \mathbb{N}^n$ :  $\omega^n(s, (i_1, \dots, i_k)) = i_1, \dots, i_n$

# REGISTER-BERECHENBARKEIT

- $f: \mathbb{N} \rightarrow \mathbb{N}$   **$\mathcal{RM}_k$ -berechenbar**
  - Es gibt eine Registermaschine  $\rho = (S, k, \delta, s_0, F)$  mit  $h_\rho = f$
- $f: \mathbb{N}^m \rightarrow \mathbb{N}^n$   **$\mathcal{RM}_k$ -berechenbar** ( $k \geq \max(m, n)$ )
  - Es gibt eine Registermaschine  $\rho = (S, k, \delta, s_0, F)$  mit  $h_\rho = f$  und
    - Anfangskonfiguration  $\alpha^m: \mathbb{N}^m \rightarrow K_\rho$ :  $\alpha^m(n_1, \dots, n_m) = (s_0, (n_1, \dots, n_m, 0, \dots, 0))$
    - Ausgabefunktion  $\omega^n: K_\rho \rightarrow \mathbb{N}^n$ :  $\omega^n(s, (i_1, \dots, i_k)) = i_1, \dots, i_n$
- **$\mathcal{RM}$** : Menge der Register-berechenbaren Funktionen
  - $\mathcal{RM}_k = \{f: \mathbb{N} \rightarrow \mathbb{N} \mid f \text{ ist } \mathcal{RM}_k\text{-berechenbar}\}$
  - $\mathcal{RM} = \bigcup \{\mathcal{RM}_k \mid k \in \mathbb{N}\}$

# BEISPIELE REGISTER-BERECHENBARER FUNKTIONEN

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

# BEISPIELE REGISTER-BERECHENBARER FUNKTIONEN

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

- $\rho_3$  muß Register  $s_0$  auf Null herunterzählen und dann  $c$  mal 1 addieren

# BEISPIELE REGISTER-BERECHENBARER FUNKTIONEN

## ● Konstante Funktion $f_3(n) = c$

–  $\rho_3$  muß Register  $s_0$  auf Null herunterzählen und dann  $c$  mal 1 addieren

$$\rho_3 = (\{s_0, \dots, s_c\}, 1, \delta_3, s_0, \{s_c\}) \quad \text{mit} \quad \delta_3 =$$

$s$	$t_1$	$s'$	$op_1$
$s_0$	0	$s_1$	+1
$s_0$	1	$s_0$	-1
$s_1$	*	$s_2$	+1
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$s_{c-1}$	*	$s_c$	+1

# BEISPIELE REGISTER-BERECHENBARER FUNKTIONEN

## ● Konstante Funktion $f_3(n) = c$

–  $\rho_3$  muß Register  $s_0$  auf Null herunterzählen und dann  $c$  mal 1 addieren

$$\rho_3 = (\{s_0, \dots, s_c\}, 1, \delta_3, s_0, \{s_c\}) \quad \text{mit} \quad \delta_3 =$$

$s$	$t_1$	$s'$	$op_1$
$s_0$	0	$s_1$	+1
$s_0$	1	$s_0$	-1
$s_1$	*	$s_2$	+1
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$s_{c-1}$	*	$s_c$	+1

## ● Addition $f_4(n, m) = n+m$



# BEISPIELE REGISTER-BERECHENBARER FUNKTIONEN

## ● Konstante Funktion $f_3(n) = c$

–  $\rho_3$  muß Register  $s_0$  auf Null herunterzählen und dann  $c$  mal 1 addieren

$$\rho_3 = (\{s_0, \dots, s_c\}, 1, \delta_3, s_0, \{s_c\}) \quad \text{mit} \quad \delta_3 =$$

$s$	$t_1$	$s'$	$op_1$
$s_0$	0	$s_1$	+1
$s_0$	1	$s_0$	-1
$s_1$	*	$s_2$	+1
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$s_{c-1}$	*	$s_c$	+1

## ● Addition $f_4(n, m) = n+m$

–  $\rho_4$  muß Register  $r_1$  auf Null herunterzählen und dabei  $r_2$  hochzählen

# BEISPIELE REGISTER-BERECHENBARER FUNKTIONEN

## ● Konstante Funktion $f_3(n) = c$

–  $\rho_3$  muß Register  $s_0$  auf Null herunterzählen und dann  $c$  mal 1 addieren

$$\rho_3 = (\{s_0, \dots, s_c\}, 1, \delta_3, s_0, \{s_c\}) \quad \text{mit} \quad \delta_3 =$$

$s$	$t_1$	$s'$	$op_1$
$s_0$	0	$s_1$	+1
$s_0$	1	$s_0$	-1
$s_1$	*	$s_2$	+1
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$s_{c-1}$	*	$s_c$	+1

## ● Addition $f_4(n, m) = n+m$

–  $\rho_4$  muß Register  $r_1$  auf Null herunterzählen und dabei  $r_2$  hochzählen

$$\rho_4 = (\{s_0, s_1\}, 2, \delta_4, s_0, \{s_1\}) \quad \text{mit} \quad \delta_4 =$$

$s$	$t_1$	$t_2$	$s'$	$op_1$	$op_2$
$s_0$	*	0	$s_1$	0	0
$s_0$	*	1	$s_0$	+1	-1

# BEISPIELE REGISTER-BERECHENBARER FUNKTIONEN

## ● Konstante Funktion $f_3(n) = c$

–  $\rho_3$  muß Register  $s_0$  auf Null herunterzählen und dann  $c$  mal 1 addieren

$$\rho_3 = (\{s_0, \dots, s_c\}, 1, \delta_3, s_0, \{s_c\}) \quad \text{mit} \quad \delta_3 =$$

$s$	$t_1$	$s'$	$op_1$
$s_0$	0	$s_1$	+1
$s_0$	1	$s_0$	-1
$s_1$	*	$s_2$	+1
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$s_{c-1}$	*	$s_c$	+1

## ● Addition $f_4(n, m) = n+m$

–  $\rho_4$  muß Register  $r_1$  auf Null herunterzählen und dabei  $r_2$  hochzählen

$$\rho_4 = (\{s_0, s_1\}, 2, \delta_4, s_0, \{s_1\}) \quad \text{mit} \quad \delta_4 =$$

$s$	$t_1$	$t_2$	$s'$	$op_1$	$op_2$
$s_0$	*	0	$s_1$	0	0
$s_0$	*	1	$s_0$	+1	-1

## ● Multiplikation $f_5(n, m) = n*m$

# BEISPIELE REGISTER-BERECHENBARER FUNKTIONEN

## ● Konstante Funktion $f_3(n) = c$

- $\rho_3$  muß Register  $s_0$  auf Null herunterzählen und dann  $c$  mal 1 addieren

$$\rho_3 = (\{s_0, \dots, s_c\}, 1, \delta_3, s_0, \{s_c\}) \quad \text{mit} \quad \delta_3 =$$

$s$	$t_1$	$s'$	$op_1$
$s_0$	0	$s_1$	+1
$s_0$	1	$s_0$	-1
$s_1$	*	$s_2$	+1
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$s_{c-1}$	*	$s_c$	+1

## ● Addition $f_4(n, m) = n+m$

- $\rho_4$  muß Register  $r_1$  auf Null herunterzählen und dabei  $r_2$  hochzählen

$$\rho_4 = (\{s_0, s_1\}, 2, \delta_4, s_0, \{s_1\}) \quad \text{mit} \quad \delta_4 =$$

$s$	$t_1$	$t_2$	$s'$	$op_1$	$op_2$
$s_0$	*	0	$s_1$	0	0
$s_0$	*	1	$s_0$	+1	-1

## ● Multiplikation $f_5(n, m) = n*m$

- $\rho_5$  muß  $r_1$  auf Null herunterzählen und dabei jeweils  $r_2+n$  berechnen
- $n$  muß zuvor in Hilfsregister kopiert werden

## ● Unterprogramme

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

## ● Unterprogramme

- Umbenennung: Separate Zustände  $s'_0, \dots, s'_n$  und Register  $r'_1, \dots, r'_k$
- 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

$r_j := r_j + 1$

$r_j := r_j - 1$

$i - j = \max(i - j, 0)$

$r_j := c$

$(c \in \mathbb{N})$

**while**  $r_j > 0$  **do**  $op$  **od** ( $op$  beliebiger Befehl)

- Jeder Befehl kann durch RM-Unterprogramme simuliert werden

## ● Unterprogramme

- Umbenennung: Separate Zustände  $s'_0, \dots, s'_n$  und Register  $r'_1, \dots, r'_k$
- 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

$r_j := r_j + 1$   
 $r_j := r_j - 1$   $i - j = \max(i - j, 0)$   
 $r_j := c$   $(c \in \mathbb{N})$   
**while**  $r_j > 0$  **do**  $op$  **od** ( $op$  beliebiger Befehl)

- Jeder Befehl kann durch RM-Unterprogramme simuliert werden

## ● Befehlsmacros

- Abkürzungen für Programmfragmente in RM-Programmiersprache

# BEFEHLSMACROS (AUSWAHL)

- $r_j := r_i$



## BEFEHLSMACROS (AUSWAHL)

- $r_j := r_i$ 
  - Verschiebe  $r_i$  in Hilfsregister  $r'$  und kopiere  $r'$  simultan nach  $r_j$  und  $r_i$   
 $r_j := 0; r' := 0; \text{ while } r_i > 0 \text{ do } r_i := r_i - 1; r' := r' + 1 \text{ od};$   
 $\text{ while } r' > 0 \text{ do } r' := r' - 1; r_j := r_j + 1; r_i := r_i + 1 \text{ od}$

## BEFEHLSMACROS (AUSWAHL)

- $r_j := r_i$ 
  - Verschiebe  $r_i$  in Hilfsregister  $r'$  und kopiere  $r'$  simultan nach  $r_j$  und  $r_i$   
 $r_j := 0; r' := 0; \text{ while } r_i > 0 \text{ do } r_i := r_i - 1; r' := r' + 1 \text{ od};$   
 $\text{ while } r' > 0 \text{ do } r' := r' - 1; r_j := r_j + 1; r_i := r_i + 1 \text{ od}$
- $r_j := r_j + r_i$

## BEFEHLSMACROS (AUSWAHL)

- $r_j := r_i$ 
  - Verschiebe  $r_i$  in Hilfsregister  $r'$  und kopiere  $r'$  simultan nach  $r_j$  und  $r_i$   
 $r_j := 0; r' := 0; \text{ while } r_i > 0 \text{ do } r_i := r_i - 1; r' := r' + 1 \text{ od};$   
 $\text{ while } r' > 0 \text{ do } r' := r' - 1; r_j := r_j + 1; r_i := r_i + 1 \text{ od}$
- $r_j := r_j + r_i$ 
  - Kopiere  $r_i$  in  $r'$  und zähle simultan  $r'$  herunter und  $r_j$  hoch  
 $r' := r_i; \text{ while } r' > 0 \text{ do } r' := r' - 1; r_j := r_j + 1 \text{ od}$

## BEFEHLSMACROS (AUSWAHL)

- $r_j := r_i$ 
  - Verschiebe  $r_i$  in Hilfsregister  $r'$  und kopiere  $r'$  simultan nach  $r_j$  und  $r_i$   
 $r_j := 0; r' := 0; \text{ while } r_i > 0 \text{ do } r_i := r_i - 1; r' := r' + 1 \text{ od};$   
 $\text{ while } r' > 0 \text{ do } r' := r' - 1; r_j := r_j + 1; r_i := r_i + 1 \text{ od}$
- $r_j := r_j + r_i$ 
  - Kopiere  $r_i$  in  $r'$  und zähle simultan  $r'$  herunter und  $r_j$  hoch  
 $r' := r_i; \text{ while } r' > 0 \text{ do } r' := r' - 1; r_j := r_j + 1 \text{ od}$
- $r_j := r_j - r_i$   
 $r_j := r_j * r_i$   
 $r_j := r_j \div r_i$   
 $r_j := r_j^{r_i}$   
 $r_j := r_j \bmod r_i$

## BEFEHLSMACROS (AUSWAHL)

- $r_j := r_i$ 
  - Verschiebe  $r_i$  in Hilfsregister  $r'$  und kopiere  $r'$  simultan nach  $r_j$  und  $r_i$   
 $r_j := 0; r' := 0; \text{ while } r_i > 0 \text{ do } r_i := r_i - 1; r' := r' + 1 \text{ od};$   
 $\text{ while } r' > 0 \text{ do } r' := r' - 1; r_j := r_j + 1; r_i := r_i + 1 \text{ od}$
- $r_j := r_j + r_i$ 
  - Kopiere  $r_i$  in  $r'$  und zähle simultan  $r'$  herunter und  $r_j$  hoch  
 $r' := r_i; \text{ while } r' > 0 \text{ do } r' := r' - 1; r_j := r_j + 1 \text{ od}$
- $r_j := r_j - r_i$   
 $r_j := r_j * r_i$   
 $r_j := r_j \div r_i$   
 $r_j := r_j^{r_i}$   
 $r_j := r_j \bmod r_i$
- $\text{while } exp(r_j) > 0 \text{ do } op \text{ od}$   
 $\text{while } exp(r_j) = 0 \text{ do } op \text{ od}$  ( $r_j := exp(r_j)$  programmierbar)

## BEFEHLSMACROS (AUSWAHL)

- $r_j := r_i$ 
  - Verschiebe  $r_i$  in Hilfsregister  $r'$  und kopiere  $r'$  simultan nach  $r_j$  und  $r_i$   
 $r_j := 0; r' := 0; \text{ while } r_i > 0 \text{ do } r_i := r_i - 1; r' := r' + 1 \text{ od};$   
 $\text{ while } r' > 0 \text{ do } r' := r' - 1; r_j := r_j + 1; r_i := r_i + 1 \text{ od}$
- $r_j := r_j + r_i$ 
  - Kopiere  $r_i$  in  $r'$  und zähle simultan  $r'$  herunter und  $r_j$  hoch  
 $r' := r_i; \text{ while } r' > 0 \text{ do } r' := r' - 1; r_j := r_j + 1 \text{ od}$
- $r_j := r_j - r_i$   
 $r_j := r_j * r_i$   
 $r_j := r_j \div r_i$   
 $r_j := r_j^{r_i}$   
 $r_j := r_j \bmod r_i$
- $\text{while } exp(r_j) > 0 \text{ do } op \text{ od}$   
 $\text{while } exp(r_j) = 0 \text{ do } op \text{ od}$   
( $r_j := exp(r_j)$  programmierbar)
- $\text{if } r_j = 0 \text{ then } op \text{ fi}$   
 $\text{if } r_j \leq r_i \text{ then } op \text{ fi}$

# REGISTER-BERECHENBARKEIT $\hat{=}$ TURING-BERECHENBARKEIT

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

Beweis durch gegenseitige Simulation

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

## 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

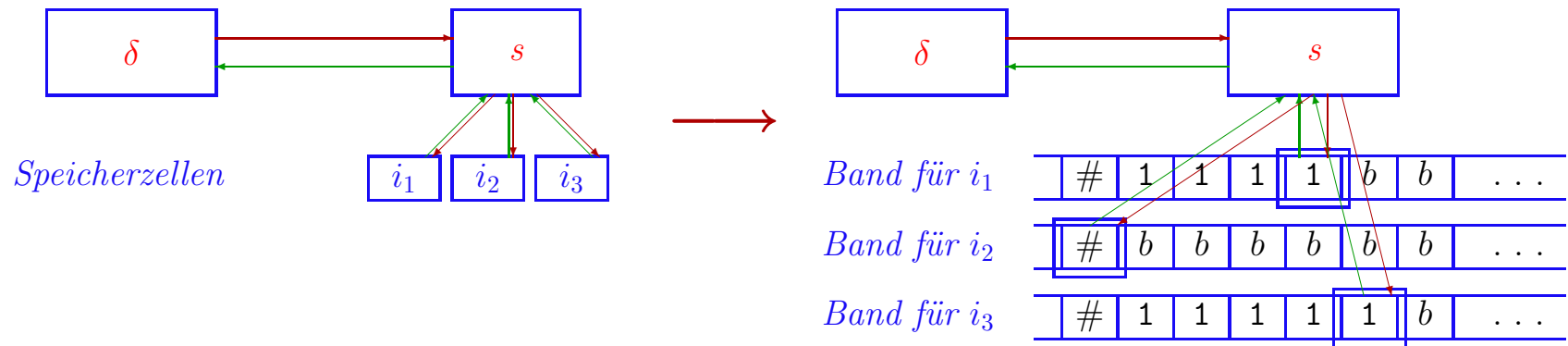


$$\mathcal{RM} = \mathcal{T}_{\{1\},\{1\}}$$

## Beweis durch gegenseitige Simulation

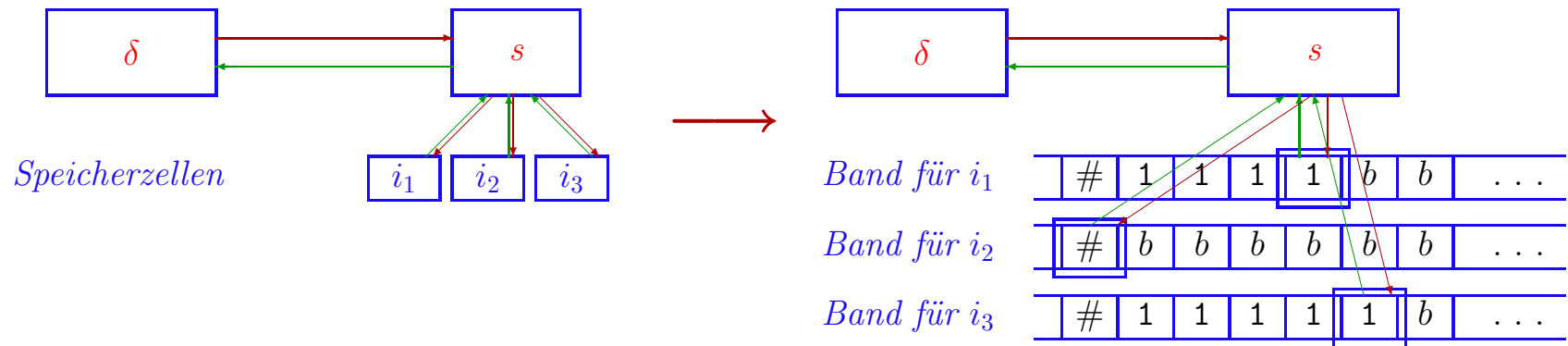
- $\mathcal{RM} \subseteq \mathcal{T}_{\{1\},\{1\}}$ 
  - Simuliere jedes Register durch separates (einseitiges) Turingband
  - Simuliere Registeroperationen durch Hinzufügen bzw. Löschen von Einsen
  - $k$ -Band-Maschine durch Einbandmaschine simulierbar
- $\mathcal{RM} \supseteq \mathcal{T}_{\{1\},\{1\}}$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



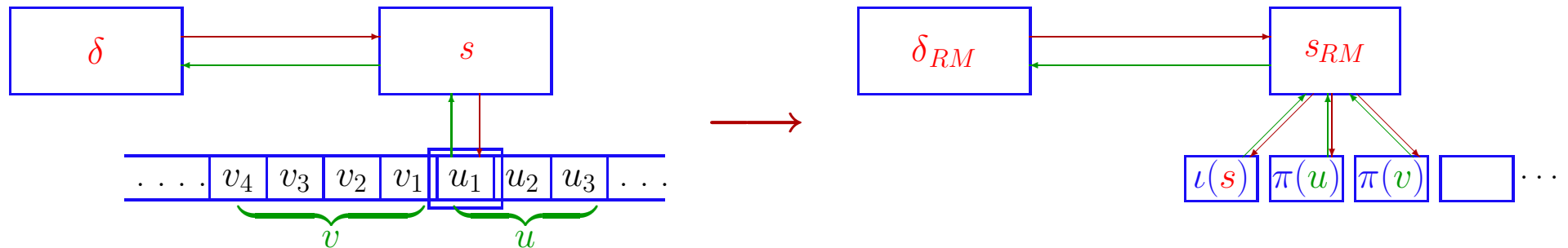
- Band für Register  $r$  wird **kellerartig** verarbeitet
  - Kopf am rechten Ende der unären Codierung des Registerinhalts

# SIMULATION EINER RM DURCH EINE TM



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

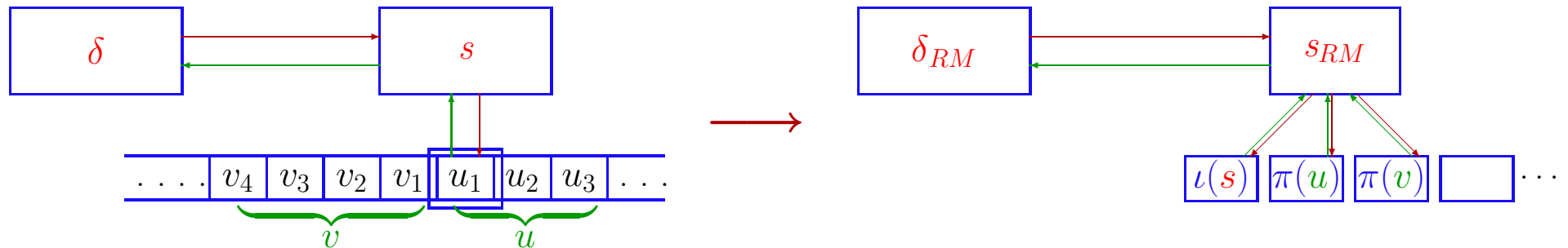
# SIMULATION EINER TM DURCH EINE RM



## ● Repräsentiere **TM-Konfigurationen** in Registern

- 3 Register für linke Hälfte, rechte Hälfte, Zustand
- Braucht eindeutige Codierung  $\pi$  von Strings als Zahlen

# SIMULATION EINER TM DURCH EINE RM



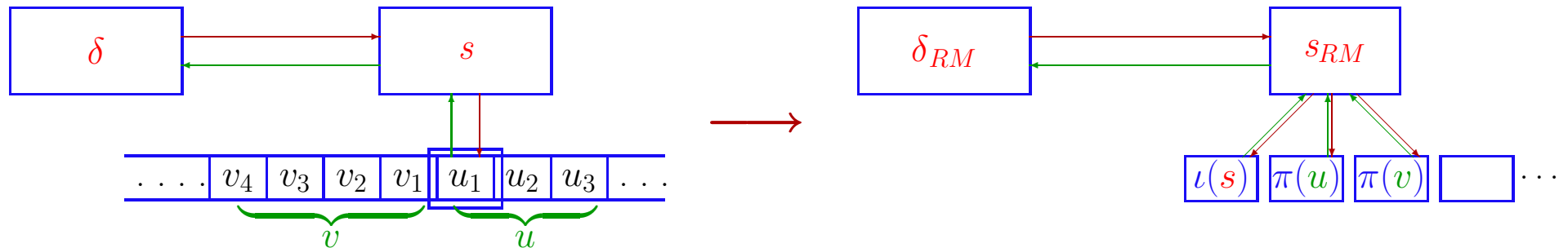
## ● Repräsentiere **TM-Konfigurationen** in Registern

- 3 Register für linke Hälfte, rechte Hälfte, Zustand
- Braucht eindeutige Codierung  $\pi$  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 Hälfte, Zustand
- Braucht eindeutige Codierung  $\pi$  von Strings als Zahlen

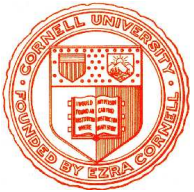
## ● Repräsentiere **Überführungstabelle** in Zuständen

- 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

# Theoretische Informatik



## Einheit 1.3

### Andere Berechenbarkeitsmodelle



1.  $\mu$ -rekursive Funktionen
2.  $\lambda$ -Kalkül
3. Church'sche These

## Mathematischer Funktionenkalkül auf $\mathbb{N}$

- Funktionen entstehen durch **Anwendung von Operationen auf Grundfunktionen**
  - Funktionsdefinition benötigt **keine Funktionsargumente**
  - Das informatiktypische “Baukastensystem” entspricht dieser Idee



## Mathematischer Funktionenkalkül auf $\mathbb{N}$

- Funktionen entstehen durch **Anwendung von Operationen auf Grundfunktionen**
  - Funktionsdefinition benötigt **keine Funktionsargumente**
  - Das informatiktypische “Baukastensystem” entspricht dieser Idee
- Bausteine gelten als **intuitiv berechenbar**
  - Grundfunktionen: **Konstante, Projektion, Nachfolgerzahl**
  - Operationen: **Komposition, einfache Rekursion, Suchschleife**

## Mathematischer Funktionenkalkül auf $\mathbb{N}$

- Funktionen entstehen durch **Anwendung von Operationen auf Grundfunktionen**
  - Funktionsdefinition benötigt **keine Funktionsargumente**
  - Das informatiktypische “Baukastensystem” entspricht dieser Idee
- Bausteine gelten als **intuitiv berechenbar**
  - Grundfunktionen: **Konstante, Projektion, Nachfolgerzahl**
  - Operationen: **Komposition, einfache Rekursion, Suchschleife**
- Berechnung durch **schrittweise Auswertung**
  - Direkte **Auswertung von Argumenten** bei Grundfunktionen
  - **Einsetzen des Definitionsschemas** bei Operationen

# BAUSTEINE $\mu$ -REKURSIVER FUNKTIONEN

1. **Nachfolgerfunktion**  $s:\mathbb{N}\rightarrow\mathbb{N}$  mit  $s(x) = x + 1$  für alle  $x \in \mathbb{N}$

# BAUSTEINE $\mu$ -REKURSIVER FUNKTIONEN

1. **Nachfolgerfunktion**  $s:\mathbb{N}\rightarrow\mathbb{N}$  mit  $s(x) = x + 1$  für alle  $x \in \mathbb{N}$
2. **Projektionsfunktionen**  $pr_k^n:\mathbb{N}^n\rightarrow\mathbb{N}$  mit  $pr_k^n(x_1, \dots, x_n) = x_k$  ( $1 \leq k \leq n$ )

# BAUSTEINE $\mu$ -REKURSIVER FUNKTIONEN

1. **Nachfolgerfunktion**  $s: \mathbb{N} \rightarrow \mathbb{N}$  mit  $s(x) = x + 1$  für alle  $x \in \mathbb{N}$
2. **Projektionsfunktionen**  $pr_k^n: \mathbb{N}^n \rightarrow \mathbb{N}$  mit  $pr_k^n(x_1, \dots, x_n) = x_k$  ( $1 \leq k \leq n$ )
3. **Konstantenfunktion**  $c_k^n: \mathbb{N}^n \rightarrow \mathbb{N}$  mit  $c_k^n(x_1, \dots, x_n) = k$  ( $0 \leq n$ )

# BAUSTEINE $\mu$ -REKURSIVER FUNKTIONEN

1. **Nachfolgerfunktion**  $s:\mathbb{N}\rightarrow\mathbb{N}$  mit  $s(x) = x + 1$  für alle  $x \in \mathbb{N}$
2. **Projektionsfunktionen**  $pr_k^n:\mathbb{N}^n\rightarrow\mathbb{N}$  mit  $pr_k^n(x_1, \dots, x_n) = x_k$  ( $1 \leq k \leq n$ )
3. **Konstantenfunktion**  $c_k^n:\mathbb{N}^n\rightarrow\mathbb{N}$  mit  $c_k^n(x_1, \dots, x_n) = k$  ( $0 \leq n$ )
4. **Komposition**  $f \circ (g_1 \dots g_n):\mathbb{N}^k\rightarrow\mathbb{N}$   
 $h = f \circ (g_1 \dots g_n)$ , wenn  $h(\vec{x}) = f(g_1(\vec{x}), \dots, g_n(\vec{x}))$   
 $h$  entsteht aus  $f:\mathbb{N}^n\rightarrow\mathbb{N}$  und  $g_1 \dots g_n:\mathbb{N}^k\rightarrow\mathbb{N}$  durch **simultane Einsetzung**

# BAUSTEINE $\mu$ -REKURSIVER FUNKTIONEN

1. **Nachfolgerfunktion**  $s : \mathbb{N} \rightarrow \mathbb{N}$  mit  $s(x) = x + 1$  für alle  $x \in \mathbb{N}$
2. **Projektionsfunktionen**  $pr_k^n : \mathbb{N}^n \rightarrow \mathbb{N}$  mit  $pr_k^n(x_1, \dots, x_n) = x_k$  ( $1 \leq k \leq n$ )
3. **Konstantenfunktion**  $c_k^n : \mathbb{N}^n \rightarrow \mathbb{N}$  mit  $c_k^n(x_1, \dots, x_n) = k$  ( $0 \leq n$ )
4. **Komposition**  $f \circ (g_1 \dots g_n) : \mathbb{N}^k \rightarrow \mathbb{N}$   
 $h = f \circ (g_1 \dots g_n)$ , wenn  $h(\vec{x}) = f(g_1(\vec{x}), \dots, g_n(\vec{x}))$   
 $h$  entsteht aus  $f : \mathbb{N}^n \rightarrow \mathbb{N}$  und  $g_1 \dots g_n : \mathbb{N}^k \rightarrow \mathbb{N}$  durch **simultane Einsetzung**
5. **Primitive Rekursion**  $Pr[f, g] : \mathbb{N}^k \rightarrow \mathbb{N}$   
 $h = Pr[f, g]$ , wenn  $h(\vec{x}, 0) = f(\vec{x})$ ,  $h(\vec{x}, y + 1) = g(\vec{x}, y, h(\vec{x}, y))$   
 $h$  entsteht aus  $f : \mathbb{N}^{k-1} \rightarrow \mathbb{N}$  und  $g : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$  durch **primitive Rekursion**

# BAUSTEINE $\mu$ -REKURSIVER FUNKTIONEN

1. **Nachfolgerfunktion**  $s:\mathbb{N}\rightarrow\mathbb{N}$  mit  $s(x) = x + 1$  für alle  $x \in \mathbb{N}$
2. **Projektionsfunktionen**  $pr_k^n:\mathbb{N}^n\rightarrow\mathbb{N}$  mit  $pr_k^n(x_1, \dots, x_n) = x_k$  ( $1 \leq k \leq n$ )
3. **Konstantenfunktion**  $c_k^n:\mathbb{N}^n\rightarrow\mathbb{N}$  mit  $c_k^n(x_1, \dots, x_n) = k$  ( $0 \leq n$ )
4. **Komposition**  $f \circ (g_1 \dots g_n):\mathbb{N}^k\rightarrow\mathbb{N}$   
 $h = f \circ (g_1 \dots g_n)$ , wenn  $h(\vec{x}) = f(g_1(\vec{x}), \dots, g_n(\vec{x}))$   
 $h$  entsteht aus  $f:\mathbb{N}^n\rightarrow\mathbb{N}$  und  $g_1 \dots g_n:\mathbb{N}^k\rightarrow\mathbb{N}$  durch **simultane Einsetzung**
5. **Primitive Rekursion**  $Pr[f, g]:\mathbb{N}^k\rightarrow\mathbb{N}$   
 $h = Pr[f, g]$ , wenn  $h(\vec{x}, 0) = f(\vec{x})$ ,  $h(\vec{x}, y + 1) = g(\vec{x}, y, h(\vec{x}, y))$   
 $h$  entsteht aus  $f:\mathbb{N}^{k-1}\rightarrow\mathbb{N}$  und  $g:\mathbb{N}^{k+1}\rightarrow\mathbb{N}$  durch **primitive Rekursion**
6.  **$\mu$ -Operator**  $\mu f:\mathbb{N}^k\rightarrow\mathbb{N}$   
 $h = \mu f$ , wenn  $h(\vec{x}) = \begin{cases} \min\{y \mid f(\vec{x}, y) = 0\} & \text{falls dies existiert und} \\ & \text{alle } f(\vec{x}, i), i < y \text{ definiert} \\ \perp & \text{sonst} \end{cases}$   
 $h$  entsteht aus  $f:\mathbb{N}^{k+1}\rightarrow\mathbb{N}$  durch **Minimierung**



- **Komposition**  $\hat{=}$  **Folge von Anweisungen**

$y_1 := g_1(x_1, \dots, x_m);$

$\vdots$

$y_n := g_n(x_1, \dots, x_m);$

$h := f(y_1, \dots, y_n)$

$(h = h(x_1, \dots, x_m))$

- **Komposition**  $\hat{=}$  **Folge von Anweisungen**

$y_1 := g_1(x_1, \dots, x_m);$

$\vdots$

$y_n := g_n(x_1, \dots, x_m);$

$h := f(y_1, \dots, y_n)$

$(h = h(x_1, \dots, x_m))$

- **Primitive Rekursion**  $\hat{=}$  **Zählschleife**

$h := f(x_1, \dots, x_n);$

for  $i:=1$  to  $y$  do  $h := g(x_1, \dots, x_n, i-1, h)$  od  $(h = h(x_1, \dots, x_n, y))$

– Primitive Rekursion arbeitet in umgekehrter Reihenfolge

- **Komposition**  $\hat{=}$  **Folge von Anweisungen**

$y_1 := g_1(x_1, \dots, x_m);$   
 $\vdots$   
 $y_n := g_n(x_1, \dots, x_m);$   
 $h := f(y_1, \dots, y_n)$

$(h = h(x_1, \dots, x_m))$

- **Primitive Rekursion**  $\hat{=}$  **Zählschleife**

$h := f(x_1, \dots, x_n);$

for  $i:=1$  to  $y$  do  $h := g(x_1, \dots, x_n, i-1, h)$  od  $(h = h(x_1, \dots, x_n, y))$

– Primitive Rekursion arbeitet in umgekehrter Reihenfolge

- **Minimierung**  $\hat{=}$  **While-schleife** (unbegrenzte Suche)

$y := 0;$

while  $f(x_1 \dots x_n, y) \neq 0$  do  $y:=y+1$  od;

$h := y$

$(h = h(x_1, \dots, x_n))$

– Ergebnis ist Anzahl der Schleifendurchläufe bis zum Erfolg

# PRIMITIV- UND $\mu$ -REKURSIVE FUNKTIONEN

- $f: \mathbb{N}^k \rightarrow \mathbb{N}$  **primitiv-rekursiv**
  - $f$  ist Nachfolger-, Projektions- oder Konstantenfunktion
  - $f$  entsteht aus primitiv-rekursiven Funktionen durch Komposition oder primitive Rekursion

# PRIMITIV- UND $\mu$ -REKURSIVE FUNKTIONEN

- $f: \mathbb{N}^k \rightarrow \mathbb{N}$  **primitiv-rekursiv**
  - $f$  ist Nachfolger-, Projektions- oder Konstantenfunktion
  - $f$  entsteht aus primitiv-rekursiven Funktionen durch Komposition oder primitive Rekursion

**$\mathcal{I}_{prim}$** : Menge der primitiv-rekursiven Funktionen

# PRIMITIV- UND $\mu$ -REKURSIVE FUNKTIONEN

- $f:\mathbb{N}^k \rightarrow \mathbb{N}$  **primitiv-rekursiv**
  - $f$  ist Nachfolger-, Projektions- oder Konstantenfunktion
  - $f$  entsteht aus primitiv-rekursiven Funktionen durch Komposition oder primitive Rekursion

**$\mathcal{I}_{prim}$** : Menge der primitiv-rekursiven Funktionen

- $f:\mathbb{N}^k \rightarrow \mathbb{N}$   **$\mu$ -rekursiv**
  - $f$  ist Nachfolger-, Projektions- oder Konstantenfunktion
  - $f$  entsteht aus  $\mu$ -rekursiven Funktionen durch Komposition, primitive Rekursion oder Minimierung

# PRIMITIV- UND $\mu$ -REKURSIVE FUNKTIONEN

- $f:\mathbb{N}^k \rightarrow \mathbb{N}$  **primitiv-rekursiv**
  - $f$  ist Nachfolger-, Projektions- oder Konstantenfunktion
  - $f$  entsteht aus primitiv-rekursiven Funktionen durch Komposition oder primitive Rekursion

**$\mathcal{T}_{prim}$** : Menge der primitiv-rekursiven Funktionen

- $f:\mathbb{N}^k \rightarrow \mathbb{N}$   **$\mu$ -rekursiv**
  - $f$  ist Nachfolger-, Projektions- oder Konstantenfunktion
  - $f$  entsteht aus  $\mu$ -rekursiven Funktionen durch Komposition, primitive Rekursion oder Minimierung

**$\mathcal{T}_\mu$** : Menge der  $\mu$ -rekursiven Funktionen

**$\mathcal{R}_\mu$** : Menge der **totalen**  $\mu$ -rekursiven Funktionen

## BEISPIEL EINER PRIMITIV-REKURSIVEN FUNKTION

- $f_1 = Pr[pr_1^1, s \circ pr_3^3]$

Was macht  $f_1$ ?



## BEISPIEL EINER PRIMITIV-REKURSIVEN FUNKTION

- $f_1 = Pr[pr_1^1, s \circ pr_3^3]$

Was macht  $f_1$ ?

- **Stelligeitsanalyse:**

$$pr_1^1: \mathbb{N} \rightarrow \mathbb{N}, pr_3^3: \mathbb{N}^3 \rightarrow \mathbb{N}, s \circ pr_3^3: \mathbb{N}^3 \rightarrow \mathbb{N}$$

$$\mapsto f_1: \mathbb{N}^2 \rightarrow \mathbb{N}$$

# BEISPIEL EINER PRIMITIV-REKURSIVEN FUNKTION

- $f_1 = Pr[pr_1^1, s \circ pr_3^3]$

Was macht  $f_1$ ?

- **Stelligeitsanalyse:**

$$pr_1^1: \mathbb{N} \rightarrow \mathbb{N}, pr_3^3: \mathbb{N}^3 \rightarrow \mathbb{N}, s \circ pr_3^3: \mathbb{N}^3 \rightarrow \mathbb{N}$$

$$\mapsto f_1: \mathbb{N}^2 \rightarrow \mathbb{N}$$

- **Abarbeitungsbeispiel:**  $f_1(2, 2)$

## BEISPIEL EINER PRIMITIV-REKURSIVEN FUNKTION

- $f_1 = Pr[pr_1^1, s \circ pr_3^3]$

Was macht  $f_1$ ?

- **Stelligeitsanalyse:**

$$pr_1^1: \mathbb{N} \rightarrow \mathbb{N}, pr_3^3: \mathbb{N}^3 \rightarrow \mathbb{N}, s \circ pr_3^3: \mathbb{N}^3 \rightarrow \mathbb{N} \quad \mapsto \quad f_1: \mathbb{N}^2 \rightarrow \mathbb{N}$$

- **Abarbeitungsbeispiel:**  $f_1(2, 2) = (s \circ pr_3^3(2, 1, f_1(2, 1)))$

## BEISPIEL EINER PRIMITIV-REKURSIVEN FUNKTION

- $f_1 = Pr[pr_1^1, s \circ pr_3^3]$

Was macht  $f_1$ ?

- **Stelligeitsanalyse:**

$$pr_1^1: \mathbb{N} \rightarrow \mathbb{N}, pr_3^3: \mathbb{N}^3 \rightarrow \mathbb{N}, s \circ pr_3^3: \mathbb{N}^3 \rightarrow \mathbb{N} \quad \mapsto \quad f_1: \mathbb{N}^2 \rightarrow \mathbb{N}$$

- **Abarbeitungsbeispiel:**  $f_1(2, 2) = (s \circ pr_3^3(2, 1, (s \circ pr_3^3(2, 0, f_1(2, 0))))$

# BEISPIEL EINER PRIMITIV-REKURSIVEN FUNKTION

- $f_1 = Pr[pr_1^1, s \circ pr_3^3]$

Was macht  $f_1$ ?

- **Stelligeitsanalyse:**

$$pr_1^1: \mathbb{N} \rightarrow \mathbb{N}, pr_3^3: \mathbb{N}^3 \rightarrow \mathbb{N}, s \circ pr_3^3: \mathbb{N}^3 \rightarrow \mathbb{N} \quad \mapsto \quad f_1: \mathbb{N}^2 \rightarrow \mathbb{N}$$

- **Abarbeitungsbeispiel:**  $f_1(2, 2) = (s \circ pr_3^3(2, 1, (s \circ pr_3^3(2, 0, pr_1^1(2))))$

## BEISPIEL EINER PRIMITIV-REKURSIVEN FUNKTION

- $f_1 = Pr[pr_1^1, s \circ pr_3^3]$

Was macht  $f_1$ ?

- **Stelligeitsanalyse:**

$$pr_1^1: \mathbb{N} \rightarrow \mathbb{N}, pr_3^3: \mathbb{N}^3 \rightarrow \mathbb{N}, s \circ pr_3^3: \mathbb{N}^3 \rightarrow \mathbb{N} \quad \mapsto \quad f_1: \mathbb{N}^2 \rightarrow \mathbb{N}$$

- **Abarbeitungsbeispiel:**  $f_1(2, 2) = (s \circ pr_3^3(2, 1, s \circ pr_3^3(2, 0, 2)))$

## BEISPIEL EINER PRIMITIV-REKURSIVEN FUNKTION

- $f_1 = Pr[pr_1^1, s \circ pr_3^3]$

Was macht  $f_1$ ?

- **Stelligeitsanalyse:**

$$pr_1^1: \mathbb{N} \rightarrow \mathbb{N}, pr_3^3: \mathbb{N}^3 \rightarrow \mathbb{N}, s \circ pr_3^3: \mathbb{N}^3 \rightarrow \mathbb{N} \quad \mapsto \quad f_1: \mathbb{N}^2 \rightarrow \mathbb{N}$$

- **Abarbeitungsbeispiel:**  $f_1(2, 2) = (s \circ pr_3^3(2, 1, s(2)))$

## BEISPIEL EINER PRIMITIV-REKURSIVEN FUNKTION

- $f_1 = Pr[pr_1^1, s \circ pr_3^3]$

Was macht  $f_1$ ?

- **Stelligeitsanalyse:**

$$pr_1^1: \mathbb{N} \rightarrow \mathbb{N}, pr_3^3: \mathbb{N}^3 \rightarrow \mathbb{N}, s \circ pr_3^3: \mathbb{N}^3 \rightarrow \mathbb{N} \quad \mapsto \quad f_1: \mathbb{N}^2 \rightarrow \mathbb{N}$$

- **Abarbeitungsbeispiel:**  $f_1(2, 2) = s \circ pr_3^3(2, 1, 3)$



## BEISPIEL EINER PRIMITIV-REKURSIVEN FUNKTION

- $f_1 = Pr[pr_1^1, s \circ pr_3^3]$

Was macht  $f_1$ ?

- **Stelligeitsanalyse:**

$$pr_1^1: \mathbb{N} \rightarrow \mathbb{N}, pr_3^3: \mathbb{N}^3 \rightarrow \mathbb{N}, s \circ pr_3^3: \mathbb{N}^3 \rightarrow \mathbb{N}$$

$$\mapsto f_1: \mathbb{N}^2 \rightarrow \mathbb{N}$$

- **Abarbeitungsbeispiel:**  $f_1(2, 2) = s(3)$

## BEISPIEL EINER PRIMITIV-REKURSIVEN FUNKTION

- $f_1 = Pr[pr_1^1, s \circ pr_3^3]$

Was macht  $f_1$ ?

- **Stelligeitsanalyse:**

$$pr_1^1: \mathbb{N} \rightarrow \mathbb{N}, pr_3^3: \mathbb{N}^3 \rightarrow \mathbb{N}, s \circ pr_3^3: \mathbb{N}^3 \rightarrow \mathbb{N}$$

$$\mapsto f_1: \mathbb{N}^2 \rightarrow \mathbb{N}$$

- **Abarbeitungsbeispiel:**  $f_1(2, 2) = 4$

# BEISPIEL EINER PRIMITIV-REKURSIVEN FUNKTION

- $f_1 = Pr[pr_1^1, s \circ pr_3^3]$

Was macht  $f_1$ ?

- **Stelligeitsanalyse:**

$$pr_1^1: \mathbb{N} \rightarrow \mathbb{N}, pr_3^3: \mathbb{N}^3 \rightarrow \mathbb{N}, s \circ pr_3^3: \mathbb{N}^3 \rightarrow \mathbb{N} \quad \mapsto \quad f_1: \mathbb{N}^2 \rightarrow \mathbb{N}$$

- **Abarbeitungsbeispiel:**  $f_1(2, 2) = 4$

- **Rekursives Verhalten:**

$$f_1(x, 0) = pr_1^1(x) = x$$

$$f_1(x, y+1) = (s \circ pr_3^3)(x, y, f_1(x, y)) = s(f_1(x, y)) = f_1(x, y) + 1$$

# BEISPIEL EINER PRIMITIV-REKURSIVEN FUNKTION

- $f_1 = Pr[pr_1^1, s \circ pr_3^3]$

Was macht  $f_1$ ?

- **Stelligeitsanalyse:**

$$pr_1^1: \mathbb{N} \rightarrow \mathbb{N}, pr_3^3: \mathbb{N}^3 \rightarrow \mathbb{N}, s \circ pr_3^3: \mathbb{N}^3 \rightarrow \mathbb{N} \quad \mapsto \quad f_1: \mathbb{N}^2 \rightarrow \mathbb{N}$$

- **Abarbeitungsbeispiel:**  $f_1(2, 2) = 4$

- **Rekursives Verhalten:**

$$f_1(x, 0) = pr_1^1(x) = x$$

$$f_1(x, y+1) = (s \circ pr_3^3)(x, y, f_1(x, y)) = s(f_1(x, y)) = f_1(x, y) + 1$$

**Das ist die Rekursionsgleichung der Addition**

$$\left. \begin{array}{l} x+0 = x \\ x+(y+1) = (x+y)+1 \end{array} \right\} \mapsto f_1 = \mathbf{add}: \mathbb{N}^2 \rightarrow \mathbb{N} \text{ mit } add(n, m) = n+m$$

# ANALYSE $\mu$ -REKURSIVER FUNKTIONEN

- $f_2 = \mu c_1^2$

# ANALYSE $\mu$ -REKURSIVER FUNKTIONEN

- $f_2 = \mu c_1^2$        $f_2(x) = \begin{cases} \min\{y \mid c_1^2(x, y) = 0\} & \text{falls } y \text{ existiert und alle} \\ & c_1^2(x, i), i < y \text{ definiert} \\ \perp & \text{sonst} \end{cases}$

# ANALYSE $\mu$ -REKURSIVER FUNKTIONEN

- $f_2 = \mu c_1^2$

$$f_2(x) = \begin{cases} \min\{y \mid c_1^2(x, y) = 0\} & \text{falls } y \text{ existiert und alle} \\ & c_1^2(x, i), i < y \text{ definiert} \\ \perp & \text{sonst} \end{cases}$$
$$= \begin{cases} \min\{y \mid 1 = 0\} & \text{falls dies existiert} \\ \perp & \text{sonst} \end{cases}$$

# ANALYSE $\mu$ -REKURSIVER FUNKTIONEN

- $f_2 = \mu c_1^2$

$$f_2(x) = \begin{cases} \min\{y \mid c_1^2(x, y) = 0\} & \text{falls } y \text{ existiert und alle} \\ & c_1^2(x, i), i < y \text{ definiert} \\ \perp & \text{sonst} \end{cases}$$
$$= \begin{cases} \min\{y \mid 1 = 0\} & \text{falls dies existiert} \\ \perp & \text{sonst} \end{cases}$$
$$= \perp$$



# ANALYSE $\mu$ -REKURSIVER FUNKTIONEN

- $f_2 = \mu c_1^2$

$$f_2(x) = \begin{cases} \min\{y \mid c_1^2(x, y) = 0\} & \text{falls } y \text{ existiert und alle} \\ & c_1^2(x, i), i < y \text{ definiert} \\ \perp & \text{sonst} \end{cases}$$
$$= \begin{cases} \min\{y \mid 1 = 0\} & \text{falls dies existiert} \\ \perp & \text{sonst} \end{cases}$$
$$= \perp$$

- $f_3 = \mu f_1$

# ANALYSE $\mu$ -REKURSIVER FUNKTIONEN

- $f_2 = \mu c_1^2$   $f_2(x) = \begin{cases} \min\{y \mid c_1^2(x, y) = 0\} & \text{falls } y \text{ existiert und alle} \\ & c_1^2(x, i), i < y \text{ definiert} \\ \perp & \text{sonst} \end{cases}$   
 $= \begin{cases} \min\{y \mid 1 = 0\} & \text{falls dies existiert} \\ \perp & \text{sonst} \end{cases}$   
 $= \perp$
- $f_3 = \mu f_1$   $f_3(x) = \begin{cases} 0 & \text{falls } x = 0 \\ \perp & \text{sonst} \end{cases}$

# ANALYSE $\mu$ -REKURSIVER FUNKTIONEN

- $f_2 = \mu c_1^2$ 

$$f_2(x) = \begin{cases} \min\{y \mid c_1^2(x, y) = 0\} & \text{falls } y \text{ existiert und alle} \\ & c_1^2(x, i), i < y \text{ definiert} \\ \perp & \text{sonst} \end{cases}$$

$$= \begin{cases} \min\{y \mid 1 = 0\} & \text{falls dies existiert} \\ \perp & \text{sonst} \end{cases}$$

$$= \perp$$
- $f_3 = \mu f_1$ 

$$f_3(x) = \begin{cases} 0 & \text{falls } x = 0 \\ \perp & \text{sonst} \end{cases}$$
- $f_4 = \mu h$  mit  $h(x, y) = \begin{cases} 0 & \text{falls } x = y \\ \perp & \text{sonst} \end{cases}$

# ANALYSE $\mu$ -REKURSIVER FUNKTIONEN

$$\begin{aligned}
 \bullet \quad f_2 &= \mu c_1^2 & f_2(x) &= \begin{cases} \min\{y \mid c_1^2(x, y) = 0\} & \text{falls } y \text{ existiert und alle} \\ & c_1^2(x, i), i < y \text{ definiert} \\ \perp & \text{sonst} \end{cases} \\
 & & &= \begin{cases} \min\{y \mid 1 = 0\} & \text{falls dies existiert} \\ \perp & \text{sonst} \end{cases} \\
 & & &= \perp
 \end{aligned}$$

$$\bullet \quad f_3 = \mu f_1 \quad f_3(x) = \begin{cases} 0 & \text{falls } x = 0 \\ \perp & \text{sonst} \end{cases}$$

$$\bullet \quad f_4 = \mu h \text{ mit } h(x, y) = \begin{cases} 0 & \text{falls } x = y \\ \perp & \text{sonst} \end{cases}$$

$$f_4(x) = \begin{cases} 0 & \text{falls } x = 0 \\ \perp & \text{sonst} \end{cases}$$

$h(x, y) = 0$  für  $x = y$  aber ist  $h$  für  $x > 0$  und  $y < x$  nicht definiert

# “PROGRAMMIERUNG” $\mu$ -REKURSIVER FUNKTIONEN

- **Vorgängerfunktion**  $p: \mathbb{N} \rightarrow \mathbb{N}$   $p(n) = n - 1$

# “PROGRAMMIERUNG” $\mu$ -REKURSIVER FUNKTIONEN

- **Vorgängerfunktion**  $p: \mathbb{N} \rightarrow \mathbb{N}$   $p(n) = n \dot{-} 1$
- **Rekursives Verhalten:**
  - $p(0) = 0 \dot{-} 1 = 0$
  - $p(y+1) = (y+1) \dot{-} 1 = y$

# “PROGRAMMIERUNG” $\mu$ -REKURSIVER FUNKTIONEN

- **Vorgängerfunktion**  $p: \mathbb{N} \rightarrow \mathbb{N}$   $p(n) = n \dot{-} 1$
- **Rekursives Verhalten:**
  - $p(0) = 0 \dot{-} 1 = 0$
  - $p(y+1) = (y+1) \dot{-} 1 = y$
- **Beschreibung durch Primitive Rekursion:**
  - Benötigt:  $f: \mathbb{N}^0 \rightarrow \mathbb{N}$  mit  $f() = 0$   $\mapsto f = c_0^0$
  - und  $g: \mathbb{N}^2 \rightarrow \mathbb{N}$  mit  $g(y, p(y)) = p(y+1) = y$   $\mapsto g = pr_1^2$

# “PROGRAMMIERUNG” $\mu$ -REKURSIVER FUNKTIONEN

- **Vorgängerfunktion**  $p: \mathbb{N} \rightarrow \mathbb{N}$   $p(n) = n \dot{-} 1$
  - **Rekursives Verhalten:**
    - $p(0) = 0 \dot{-} 1 = 0$
    - $p(y+1) = (y+1) \dot{-} 1 = y$
  - **Beschreibung durch Primitive Rekursion:**
    - Benötigt:  $f: \mathbb{N}^0 \rightarrow \mathbb{N}$  mit  $f() = 0$   $\mapsto f = c_0^0$   
und  $g: \mathbb{N}^2 \rightarrow \mathbb{N}$  mit  $g(y, p(y)) = p(y+1) = y$   $\mapsto g = pr_1^2$
- $\mapsto p = Pr[c_0^0, pr_1^2]$



# BEISPIELE PRIMITIV-REKURSIVER FUNKTIONEN

- **Subtraktion**  $sub : \mathbb{N}^2 \rightarrow \mathbb{N}$

$$sub(n, m) = n \dot{-} m$$

# BEISPIELE PRIMITIV-REKURSIVER FUNKTIONEN

- **Subtraktion  $sub$** :  $\mathbb{N}^2 \rightarrow \mathbb{N}$   $sub(n, m) = n \dot{-} m$ 
  - $sub(x, 0) = x = pr_1^1(x)$
  - $sub(x, y+1) = x \dot{-} (y+1) = (x \dot{-} y) \dot{-} 1 = p(x \dot{-} y) = (p \circ pr_3^3)(x, y, sub(x, y))$
  - $\mapsto sub = Pr[pr_1^1, p \circ pr_3^3]$

# BEISPIELE PRIMITIV-REKURSIVER FUNKTIONEN

- **Subtraktion  $sub$** :  $\mathbb{N}^2 \rightarrow \mathbb{N}$   $sub(n, m) = n \dot{-} m$ 
  - $sub(x, 0) = x = pr_1^1(x)$
  - $sub(x, y+1) = x \dot{-} (y+1) = (x \dot{-} y) \dot{-} 1 = p(x \dot{-} y) = (p \circ pr_3^3)(x, y, sub(x, y))$
  - $\mapsto sub = Pr[pr_1^1, p \circ pr_3^3]$
- **Multiplikation  $mul$** :  $\mathbb{N}^2 \rightarrow \mathbb{N}$   $mul(n, m) = n * m$

# BEISPIELE PRIMITIV-REKURSIVER FUNKTIONEN

- **Subtraktion  $sub$** :  $\mathbb{N}^2 \rightarrow \mathbb{N}$   $sub(n, m) = n \dot{-} m$ 
  - $sub(x, 0) = x = pr_1^1(x)$
  - $sub(x, y+1) = x \dot{-} (y+1) = (x \dot{-} y) \dot{-} 1 = p(x \dot{-} y) = (p \circ pr_3^3)(x, y, sub(x, y))$
  - $\mapsto sub = Pr[pr_1^1, p \circ pr_3^3]$
- **Multiplikation  $mul$** :  $\mathbb{N}^2 \rightarrow \mathbb{N}$   $mul(n, m) = n * m$ 
  - $mul(x, 0) = 0 = c_0^1(x)$
  - $mul(x, y+1) = mul(x, y) + x = (add \circ (pr_1^3, pr_3^3))(x, y, mul(x, y))$
  - $\mapsto mul = Pr[c_0^1, (add \circ (pr_1^3, pr_3^3))]$

# BEISPIELE PRIMITIV-REKURSIVER FUNKTIONEN

- **Subtraktion  $sub$** :  $\mathbb{N}^2 \rightarrow \mathbb{N}$   $sub(n, m) = n \dot{-} m$ 
  - $sub(x, 0) = x = pr_1^1(x)$
  - $sub(x, y+1) = x \dot{-} (y+1) = (x \dot{-} y) \dot{-} 1 = p(x \dot{-} y) = (p \circ pr_3^3)(x, y, sub(x, y))$
  - $\mapsto sub = Pr[pr_1^1, p \circ pr_3^3]$
- **Multiplikation  $mul$** :  $\mathbb{N}^2 \rightarrow \mathbb{N}$   $mul(n, m) = n * m$ 
  - $mul(x, 0) = 0 = c_0^1(x)$
  - $mul(x, y+1) = mul(x, y) + x = (add \circ (pr_1^3, pr_3^3))(x, y, mul(x, y))$
  - $\mapsto mul = Pr[c_0^1, (add \circ (pr_1^3, pr_3^3))]$
- **Exponentiierung  $exp$** :  $\mathbb{N}^2 \rightarrow \mathbb{N}$   $exp(n, m) = n^m$

# BEISPIELE PRIMITIV-REKURSIVER FUNKTIONEN

- **Subtraktion  $sub$** :  $\mathbb{N}^2 \rightarrow \mathbb{N}$   $sub(n, m) = n \dot{-} m$ 
  - $sub(x, 0) = x = pr_1^1(x)$
  - $sub(x, y+1) = x \dot{-} (y+1) = (x \dot{-} y) \dot{-} 1 = p(x \dot{-} y) = (p \circ pr_3^3)(x, y, sub(x, y))$
  - $\mapsto sub = Pr[pr_1^1, p \circ pr_3^3]$
- **Multiplikation  $mul$** :  $\mathbb{N}^2 \rightarrow \mathbb{N}$   $mul(n, m) = n * m$ 
  - $mul(x, 0) = 0 = c_0^1(x)$
  - $mul(x, y+1) = mul(x, y) + x = (add \circ (pr_1^3, pr_3^3))(x, y, mul(x, y))$
  - $\mapsto mul = Pr[c_0^1, (add \circ (pr_1^3, pr_3^3))]$
- **Exponentiierung  $exp$** :  $\mathbb{N}^2 \rightarrow \mathbb{N}$   $exp(n, m) = n^m$ 
  - $exp = Pr[c_1^1, (mul \circ (pr_1^3, pr_3^3))]$

# BEISPIELE PRIMITIV-REKURSIVER FUNKTIONEN

- **Subtraktion  $sub$** :  $\mathbb{N}^2 \rightarrow \mathbb{N}$   $sub(n, m) = n \dot{-} m$ 
  - $sub(x, 0) = x = pr_1^1(x)$
  - $sub(x, y+1) = x \dot{-} (y+1) = (x \dot{-} y) \dot{-} 1 = p(x \dot{-} y) = (p \circ pr_3^3)(x, y, sub(x, y))$
  - $\mapsto sub = Pr[pr_1^1, p \circ pr_3^3]$
- **Multiplikation  $mul$** :  $\mathbb{N}^2 \rightarrow \mathbb{N}$   $mul(n, m) = n * m$ 
  - $mul(x, 0) = 0 = c_0^1(x)$
  - $mul(x, y+1) = mul(x, y) + x = (add \circ (pr_1^3, pr_3^3))(x, y, mul(x, y))$
  - $\mapsto mul = Pr[c_0^1, (add \circ (pr_1^3, pr_3^3))]$
- **Exponentiierung  $exp$** :  $\mathbb{N}^2 \rightarrow \mathbb{N}$   $exp(n, m) = n^m$ 
  - $exp = Pr[c_1^1, (mul \circ (pr_1^3, pr_3^3))]$
- **Fakultät  $fak$** :  $\mathbb{N} \rightarrow \mathbb{N}$   $fak(n) = n! = 1 * 2 * \dots * n$

# BEISPIELE PRIMITIV-REKURSIVER FUNKTIONEN

- Subtraktion  $sub$** :  $\mathbb{N}^2 \rightarrow \mathbb{N}$   $sub(n, m) = n \dot{-} m$ 
  - $sub(x, 0) = x = pr_1^1(x)$
  - $sub(x, y+1) = x \dot{-} (y+1) = (x \dot{-} y) \dot{-} 1 = p(x \dot{-} y) = (p \circ pr_3^3)(x, y, sub(x, y))$
  - $\mapsto sub = Pr[pr_1^1, p \circ pr_3^3]$
- Multiplikation  $mul$** :  $\mathbb{N}^2 \rightarrow \mathbb{N}$   $mul(n, m) = n * m$ 
  - $mul(x, 0) = 0 = c_0^1(x)$
  - $mul(x, y+1) = mul(x, y) + x = (add \circ (pr_1^3, pr_3^3))(x, y, mul(x, y))$
  - $\mapsto mul = Pr[c_0^1, (add \circ (pr_1^3, pr_3^3))]$
- Exponentiierung  $exp$** :  $\mathbb{N}^2 \rightarrow \mathbb{N}$   $exp(n, m) = n^m$ 
  - $exp = Pr[c_1^1, (mul \circ (pr_1^3, pr_3^3))]$
- Fakultät  $fak$** :  $\mathbb{N} \rightarrow \mathbb{N}$   $fak(n) = n! = 1 * 2 * \dots * n$ 
  - $fak = Pr[c_1^0, (mul \circ (s \circ pr_1^2, pr_2^2))]$



# BEISPIELE PRIMITIV-REKURSIVER FUNKTIONEN

- Subtraktion  $sub$** :  $\mathbb{N}^2 \rightarrow \mathbb{N}$   $sub(n, m) = n \dot{-} m$ 
  - $sub(x, 0) = x = pr_1^1(x)$
  - $sub(x, y+1) = x \dot{-} (y+1) = (x \dot{-} y) \dot{-} 1 = p(x \dot{-} y) = (p \circ pr_3^3)(x, y, sub(x, y))$
  - $\mapsto sub = Pr[pr_1^1, p \circ pr_3^3]$
- Multiplikation  $mul$** :  $\mathbb{N}^2 \rightarrow \mathbb{N}$   $mul(n, m) = n * m$ 
  - $mul(x, 0) = 0 = c_0^1(x)$
  - $mul(x, y+1) = mul(x, y) + x = (add \circ (pr_1^3, pr_3^3))(x, y, mul(x, y))$
  - $\mapsto mul = Pr[c_0^1, (add \circ (pr_1^3, pr_3^3))]$
- Exponentiierung  $exp$** :  $\mathbb{N}^2 \rightarrow \mathbb{N}$   $exp(n, m) = n^m$ 
  - $exp = Pr[c_1^1, (mul \circ (pr_1^3, pr_3^3))]$
- Fakultät  $fak$** :  $\mathbb{N} \rightarrow \mathbb{N}$   $fak(n) = n! = 1 * 2 * \dots * n$ 
  - $fak = Pr[c_1^0, (mul \circ (s \circ pr_1^2, pr_2^2))]$
- Signum-Funktion  $sign$** :  $\mathbb{N} \rightarrow \mathbb{N}$   $sign(n) = \begin{cases} 0 & \text{falls } n = 0 \\ 1 & \text{sonst} \end{cases}$

# BEISPIELE PRIMITIV-REKURSIVER FUNKTIONEN

- Subtraktion  $sub : \mathbb{N}^2 \rightarrow \mathbb{N}$**   $sub(n, m) = n \dot{-} m$ 
  - $sub(x, 0) = x = pr_1^1(x)$
  - $sub(x, y+1) = x \dot{-} (y+1) = (x \dot{-} y) \dot{-} 1 = p(x \dot{-} y) = (p \circ pr_3^3)(x, y, sub(x, y))$
  - $\mapsto sub = Pr[pr_1^1, p \circ pr_3^3]$
- Multiplikation  $mul : \mathbb{N}^2 \rightarrow \mathbb{N}$**   $mul(n, m) = n * m$ 
  - $mul(x, 0) = 0 = c_0^1(x)$
  - $mul(x, y+1) = mul(x, y) + x = (add \circ (pr_1^3, pr_3^3))(x, y, mul(x, y))$
  - $\mapsto mul = Pr[c_0^1, (add \circ (pr_1^3, pr_3^3))]$
- Exponentiierung  $exp : \mathbb{N}^2 \rightarrow \mathbb{N}$**   $exp(n, m) = n^m$ 
  - $exp = Pr[c_1^1, (mul \circ (pr_1^3, pr_3^3))]$
- Fakultät  $fak : \mathbb{N} \rightarrow \mathbb{N}$**   $fak(n) = n! = 1 * 2 * \dots * n$ 
  - $fak = Pr[c_1^0, (mul \circ (s \circ pr_1^2, pr_2^2))]$
- Signum-Funktion  $sign : \mathbb{N} \rightarrow \mathbb{N}$**   $sign(n) = \begin{cases} 0 & \text{falls } n = 0 \\ 1 & \text{sonst} \end{cases}$ 
  - $sign = Pr[c_0^0, c_1^2]$

- **Definition durch Fallunterscheidung**

$$h(\vec{x}) = \begin{cases} f(\vec{x}) & \text{falls } test(\vec{x}) = 0 \\ g(\vec{x}) & \text{sonst} \end{cases} \quad (f, g \text{ und } test: \mathbb{N}^k \rightarrow \mathbb{N} \text{ primitiv-rekursiv})$$

## ● Definition durch Fallunterscheidung

$$h(\vec{x}) = \begin{cases} f(\vec{x}) & \text{falls } test(\vec{x}) = 0 \\ g(\vec{x}) & \text{sonst} \end{cases} \quad (f, g \text{ und } test: \mathbb{N}^k \rightarrow \mathbb{N} \text{ primitiv-rekursiv})$$

Wende Signum-Funktion auf Testergebnis an und multipliziere auf

$$- h(\vec{x}) = (1 - sign(test(\vec{x}))) * f(\vec{x}) + sign(test(\vec{x})) * g(\vec{x})$$

$$\mapsto h = add \circ (mul \circ (sub \circ (c_1^1, sign \circ test), f), mul \circ (sign \circ test, g))$$

## ● Definition durch Fallunterscheidung

$$h(\vec{x}) = \begin{cases} f(\vec{x}) & \text{falls } test(\vec{x}) = 0 \\ g(\vec{x}) & \text{sonst} \end{cases} \quad (f, g \text{ und } test: \mathbb{N}^k \rightarrow \mathbb{N} \text{ primitiv-rekursiv})$$

Wende Signum-Funktion auf Testergebnis an und multipliziere auf

$$- h(\vec{x}) = (1 - sign(test(\vec{x}))) * f(\vec{x}) + sign(test(\vec{x})) * g(\vec{x})$$

$$\mapsto h = add \circ (mul \circ (sub \circ (c_1^1, sign \circ test), f), mul \circ (sign \circ test, g))$$

## ● Generelle Summe $\Sigma_{i=0}^r f(\vec{x}, i)$

$$\text{Generelles Produkt } \Pi_{i=0}^r f(\vec{x}, i) \quad (f: \mathbb{N}^{k+1} \rightarrow \mathbb{N} \text{ primitiv-rekursiv})$$

## ● Definition durch Fallunterscheidung

$$h(\vec{x}) = \begin{cases} f(\vec{x}) & \text{falls } test(\vec{x}) = 0 \\ g(\vec{x}) & \text{sonst} \end{cases} \quad (f, g \text{ und } test: \mathbb{N}^k \rightarrow \mathbb{N} \text{ primitiv-rekursiv})$$

Wende Signum-Funktion auf Testergebnis an und multipliziere auf

$$- h(\vec{x}) = (1 - sign(test(\vec{x}))) * f(\vec{x}) + sign(test(\vec{x})) * g(\vec{x})$$

$$\mapsto h = add \circ (mul \circ (sub \circ (c_1^1, sign \circ test), f), mul \circ (sign \circ test, g))$$

## ● Generelle Summe $\Sigma_{i=0}^r f(\vec{x}, i)$

$$\text{Generelles Produkt } \Pi_{i=0}^r f(\vec{x}, i) \quad (f: \mathbb{N}^{k+1} \rightarrow \mathbb{N} \text{ primitiv-rekursiv})$$

## ● Beschränkte Minimierung

$$h(\vec{x}, t) = \begin{cases} \min\{y \leq t \mid f(\vec{x}, y) = 0\} & \text{falls dies existiert} \\ t+1 & \text{sonst} \end{cases} \quad (f: \mathbb{N}^{k+1} \rightarrow \mathbb{N} \in \mathcal{T}_{prim})$$

## ● Definition durch Fallunterscheidung

$$h(\vec{x}) = \begin{cases} f(\vec{x}) & \text{falls } test(\vec{x}) = 0 \\ g(\vec{x}) & \text{sonst} \end{cases} \quad (f, g \text{ und } test: \mathbb{N}^k \rightarrow \mathbb{N} \text{ primitiv-rekursiv})$$

Wende Signum-Funktion auf Testergebnis an und multipliziere auf

$$- h(\vec{x}) = (1 - sign(test(\vec{x}))) * f(\vec{x}) + sign(test(\vec{x})) * g(\vec{x})$$

$$\mapsto h = add \circ (mul \circ (sub \circ (c_1^1, sign \circ test), f), mul \circ (sign \circ test, g))$$

## ● Generelle Summe $\Sigma_{i=0}^r f(\vec{x}, i)$

$$\text{Generelles Produkt } \Pi_{i=0}^r f(\vec{x}, i) \quad (f: \mathbb{N}^{k+1} \rightarrow \mathbb{N} \text{ primitiv-rekursiv})$$

## ● Beschränkte Minimierung

$$h(\vec{x}, t) = \begin{cases} \min\{y \leq t \mid f(\vec{x}, y) = 0\} & \text{falls dies existiert} \\ t+1 & \text{sonst} \end{cases} \quad (f: \mathbb{N}^{k+1} \rightarrow \mathbb{N} \in \mathcal{T}_{prim})$$

Programmierbar mit Fallunterscheidung & primitiver Rekursion

(aufwendig)

## ● Definition durch Fallunterscheidung

$$h(\vec{x}) = \begin{cases} f(\vec{x}) & \text{falls } test(\vec{x}) = 0 \\ g(\vec{x}) & \text{sonst} \end{cases} \quad (f, g \text{ und } test: \mathbb{N}^k \rightarrow \mathbb{N} \text{ primitiv-rekursiv})$$

Wende Signum-Funktion auf Testergebnis an und multipliziere auf

$$- h(\vec{x}) = (1 - sign(test(\vec{x}))) * f(\vec{x}) + sign(test(\vec{x})) * g(\vec{x})$$

$$\mapsto h = add \circ (mul \circ (sub \circ (c_1^1, sign \circ test), f), mul \circ (sign \circ test, g))$$

## ● Generelle Summe $\Sigma_{i=0}^r f(\vec{x}, i)$

$$\text{Generelles Produkt } \Pi_{i=0}^r f(\vec{x}, i) \quad (f: \mathbb{N}^{k+1} \rightarrow \mathbb{N} \text{ primitiv-rekursiv})$$

## ● Beschränkte Minimierung

$$h(\vec{x}, t) = \begin{cases} \min\{y \leq t \mid f(\vec{x}, y) = 0\} & \text{falls dies existiert} \\ t+1 & \text{sonst} \end{cases} \quad (f: \mathbb{N}^{k+1} \rightarrow \mathbb{N} \in \mathcal{T}_{prim})$$

Programmierbar mit Fallunterscheidung & primitiver Rekursion

(aufwendig)



# WEITERE PRIMITIV-REKURSIVE FUNKTIONEN

- **Absolute Differenz**  $absdiff: \mathbb{N}^2 \rightarrow \mathbb{N}$   $absdiff(n, m) = |n - m|$

# WEITERE PRIMITIV-REKURSIVE FUNKTIONEN

- **Absolute Differenz**  $absdiff: \mathbb{N}^2 \rightarrow \mathbb{N}$   $absdiff(n, m) = |n - m|$
- **Maximum**  $max: \mathbb{N}^2 \rightarrow \mathbb{N}$   $max(n, m) = \begin{cases} n & \text{falls } n \geq m \\ m & \text{sonst} \end{cases}$

## WEITERE PRIMITIV-REKURSIVE FUNKTIONEN

- **Absolute Differenz** *absdiff*:  $\mathbb{N}^2 \rightarrow \mathbb{N}$   $absdiff(n, m) = |n - m|$
- **Maximum** *max*:  $\mathbb{N}^2 \rightarrow \mathbb{N}$   $max(n, m) = \begin{cases} n & \text{falls } n \geq m \\ m & \text{sonst} \end{cases}$
- **Minimum** *min*:  $\mathbb{N}^2 \rightarrow \mathbb{N}$   $min(n, m) = \begin{cases} m & \text{falls } n \geq m \\ n & \text{sonst} \end{cases}$

## WEITERE PRIMITIV-REKURSIVE FUNKTIONEN

- **Absolute Differenz**  $absdiff: \mathbb{N}^2 \rightarrow \mathbb{N}$   $absdiff(n, m) = |n - m|$
- **Maximum**  $max: \mathbb{N}^2 \rightarrow \mathbb{N}$   $max(n, m) = \begin{cases} n & \text{falls } n \geq m \\ m & \text{sonst} \end{cases}$
- **Minimum**  $min: \mathbb{N}^2 \rightarrow \mathbb{N}$   $min(n, m) = \begin{cases} m & \text{falls } n \geq m \\ n & \text{sonst} \end{cases}$
- **Division**  $div: \mathbb{N}^2 \rightarrow \mathbb{N}$   $div(n, m) = n \div m$

## WEITERE PRIMITIV-REKURSIVE FUNKTIONEN

- **Absolute Differenz** *absdiff*:  $\mathbb{N}^2 \rightarrow \mathbb{N}$   $absdiff(n, m) = |n - m|$
- **Maximum** *max*:  $\mathbb{N}^2 \rightarrow \mathbb{N}$   $max(n, m) = \begin{cases} n & \text{falls } n \geq m \\ m & \text{sonst} \end{cases}$
- **Minimum** *min*:  $\mathbb{N}^2 \rightarrow \mathbb{N}$   $min(n, m) = \begin{cases} m & \text{falls } n \geq m \\ n & \text{sonst} \end{cases}$
- **Division** *div*:  $\mathbb{N}^2 \rightarrow \mathbb{N}$   $div(n, m) = n \div m$
- **Divisionsrest** *mod*:  $\mathbb{N}^2 \rightarrow \mathbb{N}$   $mod(n, m) = n \bmod m$

## WEITERE PRIMITIV-REKURSIVE FUNKTIONEN

- **Absolute Differenz** *absdiff*:  $\mathbb{N}^2 \rightarrow \mathbb{N}$   $absdiff(n, m) = |n - m|$
- **Maximum** *max*:  $\mathbb{N}^2 \rightarrow \mathbb{N}$   $max(n, m) = \begin{cases} n & \text{falls } n \geq m \\ m & \text{sonst} \end{cases}$
- **Minimum** *min*:  $\mathbb{N}^2 \rightarrow \mathbb{N}$   $min(n, m) = \begin{cases} m & \text{falls } n \geq m \\ n & \text{sonst} \end{cases}$
- **Division** *div*:  $\mathbb{N}^2 \rightarrow \mathbb{N}$   $div(n, m) = n \div m$
- **Divisionsrest** *mod*:  $\mathbb{N}^2 \rightarrow \mathbb{N}$   $mod(n, m) = n \bmod m$
- **Quadratwurzel** *sqrt*:  $\mathbb{N} \rightarrow \mathbb{N}$   $sqrt(n) = \lfloor \sqrt{n} \rfloor$

## WEITERE PRIMITIV-REKURSIVE FUNKTIONEN

- **Absolute Differenz** *absdiff*:  $\mathbb{N}^2 \rightarrow \mathbb{N}$   $absdiff(n, m) = |n - m|$
- **Maximum** *max*:  $\mathbb{N}^2 \rightarrow \mathbb{N}$   $max(n, m) = \begin{cases} n & \text{falls } n \geq m \\ m & \text{sonst} \end{cases}$
- **Minimum** *min*:  $\mathbb{N}^2 \rightarrow \mathbb{N}$   $min(n, m) = \begin{cases} m & \text{falls } n \geq m \\ n & \text{sonst} \end{cases}$
- **Division** *div*:  $\mathbb{N}^2 \rightarrow \mathbb{N}$   $div(n, m) = n \div m$
- **Divisionsrest** *mod*:  $\mathbb{N}^2 \rightarrow \mathbb{N}$   $mod(n, m) = n \bmod m$
- **Quadratwurzel** *sqrt*:  $\mathbb{N} \rightarrow \mathbb{N}$   $sqrt(n) = \lfloor \sqrt{n} \rfloor$
- **Logarithmus** *ld*:  $\mathbb{N} \rightarrow \mathbb{N}$   $ld(n) = \lfloor \log_2 n \rfloor$

## WEITERE PRIMITIV-REKURSIVE FUNKTIONEN

- **Absolute Differenz**  $absdiff: \mathbb{N}^2 \rightarrow \mathbb{N}$   $absdiff(n, m) = |n - m|$
- **Maximum**  $max: \mathbb{N}^2 \rightarrow \mathbb{N}$   $max(n, m) = \begin{cases} n & \text{falls } n \geq m \\ m & \text{sonst} \end{cases}$
- **Minimum**  $min: \mathbb{N}^2 \rightarrow \mathbb{N}$   $min(n, m) = \begin{cases} m & \text{falls } n \geq m \\ n & \text{sonst} \end{cases}$
- **Division**  $div: \mathbb{N}^2 \rightarrow \mathbb{N}$   $div(n, m) = n \div m$
- **Divisionsrest**  $mod: \mathbb{N}^2 \rightarrow \mathbb{N}$   $mod(n, m) = n \bmod m$
- **Quadratwurzel**  $sqrt: \mathbb{N} \rightarrow \mathbb{N}$   $sqrt(n) = \lfloor \sqrt{n} \rfloor$
- **Logarithmus**  $ld: \mathbb{N} \rightarrow \mathbb{N}$   $ld(n) = \lfloor \log_2 n \rfloor$
- **Größter gemeinsamer Teiler**  $ggT: \mathbb{N}^2 \rightarrow \mathbb{N}$



## WEITERE PRIMITIV-REKURSIVE FUNKTIONEN

- **Absolute Differenz**  $absdiff: \mathbb{N}^2 \rightarrow \mathbb{N}$   $absdiff(n, m) = |n - m|$
- **Maximum**  $max: \mathbb{N}^2 \rightarrow \mathbb{N}$   $max(n, m) = \begin{cases} n & \text{falls } n \geq m \\ m & \text{sonst} \end{cases}$
- **Minimum**  $min: \mathbb{N}^2 \rightarrow \mathbb{N}$   $min(n, m) = \begin{cases} m & \text{falls } n \geq m \\ n & \text{sonst} \end{cases}$
- **Division**  $div: \mathbb{N}^2 \rightarrow \mathbb{N}$   $div(n, m) = n \div m$
- **Divisionsrest**  $mod: \mathbb{N}^2 \rightarrow \mathbb{N}$   $mod(n, m) = n \bmod m$
- **Quadratwurzel**  $sqrt: \mathbb{N} \rightarrow \mathbb{N}$   $sqrt(n) = \lfloor \sqrt{n} \rfloor$
- **Logarithmus**  $ld: \mathbb{N} \rightarrow \mathbb{N}$   $ld(n) = \lfloor \log_2 n \rfloor$
- **Größter gemeinsamer Teiler**  $ggT: \mathbb{N}^2 \rightarrow \mathbb{N}$
- **Kleinstes gemeinsames Vielfaches**  $kgV: \mathbb{N}^2 \rightarrow \mathbb{N}$

# BERECHENBARE NUMERIERUNG VON ZAHLENPAAREN

	0	1	2	3	4	5	...
0							
1							
2							
3							
4							
5							
⋮							

# BERECHENBARE NUMERIERUNG VON ZAHLENPAAREN

	0	1	2	3	4	5	...
0	0						
1							
2							
3							
4							
5							
⋮							

# BERECHENBARE NUMERIERUNG VON ZAHLENPAAREN

	0	1	2	3	4	5	...
0	0						
1	1						
2							
3							
4							
5							
⋮							

# BERECHENBARE NUMERIERUNG VON ZAHLENPAAREN

	0	1	2	3	4	5	...
0	0	2					
1	1						
2							
3							
4							
5							
⋮							

# BERECHENBARE NUMERIERUNG VON ZAHLENPAAREN

	0	1	2	3	4	5	...
0	0	2					
1	1						
2	3						
3							
4							
5							
⋮							

# BERECHENBARE NUMERIERUNG VON ZAHLENPAAREN

	0	1	2	3	4	5	...
0	0	2					
1	1	4					
2	3						
3							
4							
5							
⋮							

# BERECHENBARE NUMERIERUNG VON ZAHLENPAAREN

	0	1	2	3	4	5	...
0	0	2	5				
1	1	4					
2	3						
3							
4							
5							
⋮							



# BERECHENBARE NUMERIERUNG VON ZAHLENPAAREN

	0	1	2	3	4	5	...
0	0	2	5	9			
1	1	4	8				
2	3	7					
3	6						
4							
5							
⋮							

# BERECHENBARE NUMERIERUNG VON ZAHLENPAAREN

	0	1	2	3	4	5	...
0	0	2	5	9	14		
1	1	4	8	13			
2	3	7	12				
3	6	11					
4	10						
5							
⋮							

# BERECHENBARE NUMERIERUNG VON ZAHLENPAAREN

	0	1	2	3	4	5	...
0	0	2	5	9	14	20	
1	1	4	8	13	19		
2	3	7	12	18			
3	6	11	17				
4	10	16					
5	15						
⋮							

# BERECHENBARE NUMERIERUNG VON ZAHLENPAAREN

	0	1	2	3	4	5	...
0	0	2	5	9	14	20	...
1	1	4	8	13	19		...
2	3	7	12	18			...
3	6	11	17				...
4	10	16					...
5	15						...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	...

# BERECHENBARE NUMERIERUNG VON ZAHLENPAAREN

	0	1	2	3	4	5	...
0	0	2	5	9	14	20	...
1	1	4	8	13	19		...
2	3	7	12	18			...
3	6	11	17				...
4	10	16					...
5	15						...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	...

$\langle x, y \rangle$

$:=$

$$(x+y)(x+y+1) \div 2 + y$$

“Standard-Tupelfunktion”

# BERECHENBARE NUMERIERUNG VON ZAHLENPAAREN

	0	1	2	3	4	5	...
0	0	2	5	9	14	20	...
1	1	4	8	13	19		...
2	3	7	12	18			...
3	6	11	17				...
4	10	16					...
5	15						...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	...

$\langle x, y \rangle$

$:=$

$$(x+y)(x+y+1) \div 2 + y$$

“Standard-Tupelfunktion”

- $\langle \rangle: \mathbb{N}^2 \rightarrow \mathbb{N}$  ist **primitiv-rekursiv** und **bijektiv**

# BERECHENBARE NUMERIERUNG VON ZAHLENPAAREN

	0	1	2	3	4	5	...
0	0	2	5	9	14	20	...
1	1	4	8	13	19		...
2	3	7	12	18			...
3	6	11	17				...
4	10	16					...
5	15						...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	...

$\langle x, y \rangle$

$:=$

$$(x+y)(x+y+1) \div 2 + y$$

“Standard-Tupelfunktion”

- $\langle \rangle : \mathbb{N}^2 \rightarrow \mathbb{N}$  ist **primitiv-rekursiv** und **bijektiv**
- Die **Umkehrfunktionen**  $\pi_i^2 := pr_i^2 \circ \langle \rangle^{-1}$  sind **primitiv-rekursiv**

# BERECHENBARE NUMERIERUNG VON ZAHLENPAAREN

	0	1	2	3	4	5	...
0	0	2	5	9	14	20	...
1	1	4	8	13	19		...
2	3	7	12	18			...
3	6	11	17				...
4	10	16					...
5	15						...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	...

$\langle x, y \rangle$

$:=$

$$(x+y)(x+y+1) \div 2 + y$$

“Standard-Tupelfunktion”

- $\langle \rangle : \mathbb{N}^2 \rightarrow \mathbb{N}$  ist **primitiv-rekursiv** und **bijektiv**
- Die **Umkehrfunktionen**  $\pi_i^2 := pr_i^2 \circ \langle \rangle^{-1}$  sind **primitiv-rekursiv**
- $\langle \rangle$  kann **iterativ** auf  $\mathbb{N}^k \rightarrow \mathbb{N}$  und auf  $\mathbb{N}^* \rightarrow \mathbb{N}$  fortgesetzt werden
  - $\langle x, y, z \rangle^3 = \langle x, \langle y, z \rangle \rangle, \dots, \langle x_1 \dots x_k \rangle^* = \langle k, \langle x_1, \dots, x_k \rangle^k \rangle$
  - Alle Funktionen sind **bijektiv** und **primitiv-rekursiv**
  - Alle **Umkehrfunktionen**  $\pi_i^k$  und  $\pi_i^*$  sind **primitiv-rekursiv**



# BERECHENBARE NUMERIERUNG VON ZAHLENPAAREN

	0	1	2	3	4	5	...
0	0	2	5	9	14	20	...
1	1	4	8	13	19		...
2	3	7	12	18			...
3	6	11	17				...
4	10	16					...
5	15						...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	...

$\langle x, y \rangle$

$:=$

$$(x+y)(x+y+1) \div 2 + y$$

“**Standard-Tupelfunktion**”

- $\langle \rangle : \mathbb{N}^2 \rightarrow \mathbb{N}$  ist **primitiv-rekursiv** und **bijektiv**
- Die **Umkehrfunktionen**  $\pi_i^2 := pr_i^2 \circ \langle \rangle^{-1}$  sind **primitiv-rekursiv**
- $\langle \rangle$  kann **iterativ** auf  $\mathbb{N}^k \rightarrow \mathbb{N}$  und auf  $\mathbb{N}^* \rightarrow \mathbb{N}$  fortgesetzt werden
  - $\langle x, y, z \rangle^3 = \langle x, \langle y, z \rangle \rangle, \dots, \langle x_1 \dots x_k \rangle^* = \langle k, \langle x_1, \dots, x_k \rangle^k \rangle$
  - Alle Funktionen sind **bijektiv** und **primitiv-rekursiv**
  - Alle **Umkehrfunktionen**  $\pi_i^k$  und  $\pi_i^*$  sind **primitiv-rekursiv**
- **Jede rekursive Funktion kann einstellig simuliert werden**
  - Für  $f: \mathbb{N}^2 \rightarrow \mathbb{N}$  und  $g := f \circ (\pi_1^2, \pi_2^2)$  gilt  $g: \mathbb{N} \rightarrow \mathbb{N}$  und  $f(x, y) = g(\langle x, y \rangle)$

# ACKERMANN-FUNKTIONEN (1928)

# ACKERMANN-FUNKTIONEN (1928)

- Definiere Funktionen  $A_n$  iterativ:

$$A_0(x) := \begin{cases} 1 & \text{falls } x = 0 \\ 2 & \text{falls } x = 1 \\ x+2 & \text{sonst} \end{cases} \quad \begin{aligned} A_{n+1}(0) &:= 1 \\ A_{n+1}(x+1) &:= A_n(A_{n+1}(x)) \end{aligned}$$

# ACKERMANN-FUNKTIONEN (1928)

- Definiere Funktionen  $A_n$  iterativ:

$$A_0(x) := \begin{cases} 1 & \text{falls } x = 0 \\ 2 & \text{falls } x = 1 \\ x+2 & \text{sonst} \end{cases} \quad \begin{aligned} A_{n+1}(0) &:= 1 \\ A_{n+1}(x+1) &:= A_n(A_{n+1}(x)) \end{aligned}$$

- Jede der Funktionen  $A_n$  ist primitiv-rekursiv

# ACKERMANN-FUNKTIONEN (1928)

- Definiere Funktionen  $A_n$  iterativ:

$$A_0(x) := \begin{cases} 1 & \text{falls } x = 0 \\ 2 & \text{falls } x = 1 \\ x+2 & \text{sonst} \end{cases} \quad \begin{aligned} A_{n+1}(0) &:= 1 \\ A_{n+1}(x+1) &:= A_n(A_{n+1}(x)) \end{aligned}$$

- Jede der Funktionen  $A_n$  ist primitiv-rekursiv

- Wachstumsverhalten

$$A_1(x) = 2x \quad (x \geq 1)$$

$$A_4(0) = 1$$

$$A_4(3) = 2^{2^{2^2}} = 65536$$

$$A_2(x) = 2^x$$

$$A_4(1) = 2$$

$$A_4(4) = \underbrace{2^{(2^{(2^{\dots^2})})}}_{65536 - \text{mal}}$$

$$A_3(x) = \underbrace{2^{(2^{(2^{\dots^2})})}}_{x - \text{mal}}$$

$$A_4(2) = 2^2 = 4$$

$$A_4(5) = \underbrace{2^{(2^{(2^{\dots^2})})}}_{A_4(4) - \text{mal}}$$

# AUSDRUCKSKRAFT REKURSIVER FUNKTIONEN

$$\mathcal{T}_{prim} \subset \mathcal{R}_\mu \subset \mathcal{T}_\mu = \mathcal{RM} = \mathcal{T}$$

# AUSDRUCKSKRAFT REKURSIVER FUNKTIONEN

$$\mathcal{T}_{prim} \subset \mathcal{R}_\mu \subset \mathcal{T}_\mu = \mathcal{RM} = \mathcal{T}$$

- $\mathcal{T}_{prim} \subseteq \mathcal{R}_\mu \subseteq \mathcal{T}_\mu$  gilt offensichtlich
  - Grundfunktionen und Anwendungen von p.r. Operationen sind total

# AUSDRUCKSKRAFT REKURSIVER FUNKTIONEN

$$\mathcal{T}_{prim} \subset \mathcal{R}_\mu \subset \mathcal{T}_\mu = \mathcal{RM} = \mathcal{T}$$

- $\mathcal{T}_{prim} \subseteq \mathcal{R}_\mu \subseteq \mathcal{T}_\mu$  gilt offensichtlich
  - Grundfunktionen und Anwendungen von p.r. Operationen sind total
- $\mathcal{R}_\mu \neq \mathcal{T}_\mu$ 
  - Nicht alle  $\mu$ -rekursiven Funktionen sind total (Beispiel:  $f_3 = \mu add$ )



# AUSDRUCKSKRAFT REKURSIVER FUNKTIONEN

$$\mathcal{T}_{prim} \subset \mathcal{R}_\mu \subset \mathcal{T}_\mu = \mathcal{RM} = \mathcal{T}$$

- $\mathcal{T}_{prim} \subseteq \mathcal{R}_\mu \subseteq \mathcal{T}_\mu$  gilt offensichtlich
  - Grundfunktionen und Anwendungen von p.r. Operationen sind total
- $\mathcal{R}_\mu \neq \mathcal{T}_\mu$ 
  - Nicht alle  $\mu$ -rekursiven Funktionen sind total (Beispiel:  $f_3 = \mu add$ )
- $\mathcal{T}_{prim} \neq \mathcal{R}_\mu$ 
  - Primitiv-rekursive Funktionen haben **endliche Schachtelungstiefe**
  - **Unbegrenzte Iteration** über Schachtelungstiefe ist intuitiv **berechenbar**
  - Konkretes Beispiel: **Ackermann-Funktion**

# AUSDRUCKSKRAFT REKURSIVER FUNKTIONEN

$$\mathcal{T}_{prim} \subset \mathcal{R}_\mu \subset \mathcal{T}_\mu = \mathcal{RM} = \mathcal{T}$$

- $\mathcal{T}_{prim} \subseteq \mathcal{R}_\mu \subseteq \mathcal{T}_\mu$  gilt offensichtlich
  - Grundfunktionen und Anwendungen von p.r. Operationen sind total
- $\mathcal{R}_\mu \neq \mathcal{T}_\mu$ 
  - Nicht alle  $\mu$ -rekursiven Funktionen sind total (Beispiel:  $f_3 = \mu add$ )
- $\mathcal{T}_{prim} \neq \mathcal{R}_\mu$ 
  - Primitiv-rekursive Funktionen haben **endliche Schachtelungstiefe**
  - **Unbegrenzte Iteration** über Schachtelungstiefe ist intuitiv **berechenbar**
  - Konkretes Beispiel: **Ackermann-Funktion**
- $\mathcal{T}_\mu = \mathcal{RM} = \mathcal{T}$ 
  - $\subseteq$ : Gebe RM-Unterprogramme für Grundfunktionen und Operationen
  - $\supseteq$ : Beschreibe RM-Konfigurationsübergänge und Terminierung  $\mu$ -rekursiv

## Grundlage funktionaler Programmiersprachen

## Grundlage funktionaler Programmiersprachen

### ● Einfacher mathematischer Mechanismus

- Funktionen werden **definiert** und **angewandt**
- Die Beschreibung des Funktionsverhaltens ist der Name der Funktion
- Funktionswerte werden ausgerechnet durch **Einsetzen** von Werten

## Grundlage funktionaler Programmiersprachen

- **Einfacher mathematischer Mechanismus**

- Funktionen werden **definiert** und **angewandt**
- Die Beschreibung des Funktionsverhaltens ist der Name der Funktion
- Funktionswerte werden ausgerechnet durch **Einsetzen** von Werten

- **Leicht zu verstehen**

- **Definition** einer Funktion:  $f(x) = 2*x+3$

## Grundlage funktionaler Programmiersprachen

### ● Einfacher mathematischer Mechanismus

- Funktionen werden **definiert** und **angewandt**
- Die Beschreibung des Funktionsverhaltens ist der Name der Funktion
- Funktionswerte werden ausgerechnet durch **Einsetzen** von Werten

### ● Leicht zu verstehen

- **Definition** einer Funktion:  $f(x) = 2*x+3$
- **Auswertung** der Funktion:  $f(4) = 2*4+3 = 11$

## Grundlage funktionaler Programmiersprachen

### ● Einfacher mathematischer Mechanismus

- Funktionen werden **definiert** und **angewandt**
- Die Beschreibung des Funktionsverhaltens ist der Name der Funktion
- Funktionswerte werden ausgerechnet durch **Einsetzen** von Werten

### ● Leicht zu verstehen

- **Definition** einer Funktion:  $f \hat{=} x \mapsto 2*x+3$
- **Auswertung** der Funktion:  $f(4) = 2*4+3 = 11$

**Name der Funktion ist irrelevant**

## Grundlage funktionaler Programmiersprachen

### ● Einfacher mathematischer Mechanismus

- Funktionen werden **definiert** und **angewandt**
- Die Beschreibung des Funktionsverhaltens ist der Name der Funktion
- Funktionswerte werden ausgerechnet durch **Einsetzen** von Werten

### ● Leicht zu verstehen

- **Definition** einer Funktion:  $f \hat{=} \lambda x. 2*x+3$       **Abstraktion** von  $x$
- **Auswertung** der Funktion:  $f(4) = 2*4+3 = 11$

**Name der Funktion ist irrelevant**



## Grundlage funktionaler Programmiersprachen

- **Einfacher mathematischer Mechanismus**

- Funktionen werden **definiert** und **angewandt**
- Die Beschreibung des Funktionsverhaltens ist der Name der Funktion
- Funktionswerte werden ausgerechnet durch **Einsetzen** von Werten

- **Leicht zu verstehen**

- **Definition** einer Funktion:  $f \hat{=} \lambda x. 2*x+3$   $\lambda$ -Notation
- **Auswertung** der Funktion:

**Name der Funktion ist irrelevant**

## Grundlage funktionaler Programmiersprachen

### ● Einfacher mathematischer Mechanismus

- Funktionen werden **definiert** und **angewandt**
- Die Beschreibung des Funktionsverhaltens ist der Name der Funktion
- Funktionswerte werden ausgerechnet durch **Einsetzen** von Werten

### ● Leicht zu verstehen

- **Definition** einer Funktion:  $f \hat{=} \lambda x. 2*x+3$   **$\lambda$ -Notation**
- **Auswertung** der Funktion:  $(\lambda x. 2*x+3)(4)$  **Applikation**

**Name der Funktion ist irrelevant**

## Grundlage funktionaler Programmiersprachen

### ● Einfacher mathematischer Mechanismus

- Funktionen werden **definiert** und **angewandt**
- Die Beschreibung des Funktionsverhaltens ist der Name der Funktion
- Funktionswerte werden ausgerechnet durch **Einsetzen** von Werten

### ● Leicht zu verstehen

- **Definition** einer Funktion:  $f \hat{=} \lambda x. 2*x+3$  λ-Notation
  - **Auswertung** der Funktion:  $(\lambda x. 2*x+3) (4) \xrightarrow{\beta} 11$  Applikation  
+ Reduktion
- Name der Funktion ist irrelevant**

## ● $\lambda$ -Terme

- Variablen  $x$
- $\lambda x.t$ , wobei  $x$  Variable und  $t$   $\lambda$ -Term  
Vorkommen von  $x$  in  $t$  werden **gebunden**
- $f t$ , wobei  $t$  und  $f$   $\lambda$ -Terme
- $(t)$ , wobei  $t$   $\lambda$ -Term

$\lambda$ -Abstraktion

Applikation

## ● $\lambda$ -Terme

- Variablen  $x$
- $\lambda x.t$ , wobei  $x$  Variable und  $t$   $\lambda$ -Term  
Vorkommen von  $x$  in  $t$  werden **gebunden**  $\lambda$ -Abstraktion
- $f t$ , wobei  $t$  und  $f$   $\lambda$ -Terme Applikation
- $(t)$ , wobei  $t$   $\lambda$ -Term

## ● Konventionen

- Applikation bindet stärker als  $\lambda$ -Abstraktion
- Applikation ist **links**-assoziativ:  $f t_1 t_2 \hat{=} (f t_1) t_2$
- Notation  $f(t_1, \dots, t_n)$  entspricht iterierter Applikation  $f t_1 \dots t_n$

## ● $\lambda$ -Terme

- Variablen  $x$
- $\lambda x.t$ , wobei  $x$  Variable und  $t$   $\lambda$ -Term  
Vorkommen von  $x$  in  $t$  werden **gebunden**  $\lambda$ -Abstraktion
- $f t$ , wobei  $t$  und  $f$   $\lambda$ -Terme Applikation
- $(t)$ , wobei  $t$   $\lambda$ -Term

## ● Konventionen

- Applikation bindet stärker als  $\lambda$ -Abstraktion
- Applikation ist **links**-assoziativ:  $f t_1 t_2 \hat{=} (f t_1) t_2$
- Notation  $f(t_1, \dots, t_n)$  entspricht iterierter Applikation  $f t_1 \dots t_n$

## ● **Auswertung** von $\lambda$ -Termen

- Ersetze Funktionsparameter durch Funktionsargumente
- **Reduktion**  $(\lambda x.t)(b) \xrightarrow{\beta} t[b/x]$
- **Substitution**  $t[b/x]$ : ersetze **freie** Vorkommen von  $x$  in  $t$  durch  $b$

# SUBSTITUTION UND REDUKTION AM BEISPIEL

$$(\lambda n. \lambda f. \lambda x. \ n \ f \ (f \ x)) \ (\lambda f. \lambda x. x)$$

# SUBSTITUTION UND REDUKTION AM BEISPIEL

$$\begin{aligned} & (\lambda n. \lambda f. \lambda x. n \ f \ (f \ x)) \ (\lambda f. \lambda x. x) \\ \longrightarrow & \llbracket \lambda f. \lambda x. n \ f \ (f \ x) \rrbracket [\lambda f. \lambda x. x / n] \end{aligned}$$



# SUBSTITUTION UND REDUKTION AM BEISPIEL

$$\begin{aligned} & (\lambda n. \lambda f. \lambda x. n \ f \ (f \ x)) \ (\lambda f. \lambda x. x) \\ \longrightarrow & \lambda f. \llbracket \lambda x. n \ f \ (f \ x) \rrbracket [\lambda f. \lambda x. x / n] \end{aligned}$$

# SUBSTITUTION UND REDUKTION AM BEISPIEL

$$\begin{aligned} & (\lambda n. \lambda f. \lambda x. n \ f \ (f \ x)) \ (\lambda f. \lambda x. x) \\ \longrightarrow & \lambda f. \lambda x. \llbracket n \ f \ (f \ x) \rrbracket [\lambda f. \lambda x. x / n] \end{aligned}$$

# SUBSTITUTION UND REDUKTION AM BEISPIEL

$$\begin{aligned} & (\lambda n. \lambda f. \lambda x. n \ f \ (f \ x)) \ (\lambda f. \lambda x. x) \\ \longrightarrow & \lambda f. \lambda x. (\lambda f. \lambda x. x) \ f \ (f \ x) \end{aligned}$$

# SUBSTITUTION UND REDUKTION AM BEISPIEL

$$\begin{aligned} & (\lambda n. \lambda f. \lambda x. n \ f \ (f \ x)) \ (\lambda f. \lambda x. x) \\ \longrightarrow & \lambda f. \lambda x. (\lambda f. \lambda x. x) \ f \ (f \ x) \end{aligned}$$

# SUBSTITUTION UND REDUKTION AM BEISPIEL

$$\begin{aligned} & (\lambda n. \lambda f. \lambda x. \ n \ f \ (f \ x)) \ (\lambda f. \lambda x. x) \\ \longrightarrow & \lambda f. \lambda x. \ (\lambda f. \lambda x. x) \ f \ (f \ x) \end{aligned}$$

# SUBSTITUTION UND REDUKTION AM BEISPIEL

$$\begin{aligned} & (\lambda n. \lambda f. \lambda x. \ n \ f \ (f \ x)) \ (\lambda f. \lambda x. x) \\ \longrightarrow & \lambda f. \lambda x. \ (\lambda f. \lambda x. x) \ f \ (f \ x) \end{aligned}$$

# SUBSTITUTION UND REDUKTION AM BEISPIEL

$$\begin{aligned} & (\lambda n. \lambda f. \lambda x. \ n \ f \ (f \ x)) \ (\lambda f. \lambda x. x) \\ \longrightarrow & \ \lambda f. \lambda x. \ (\lambda f. \lambda x. x) \ f \ (f \ x) \\ \longrightarrow & \ [\lambda x. x] [f / f] \end{aligned}$$

# SUBSTITUTION UND REDUKTION AM BEISPIEL

$$\begin{aligned} & (\lambda n. \lambda f. \lambda x. n \ f \ (f \ x)) \ (\lambda f. \lambda x. x) \\ \longrightarrow & \lambda f. \lambda x. \ (\lambda f. \lambda x. x) \ f \ (f \ x) \\ \longrightarrow & \lambda x. x \end{aligned}$$



# SUBSTITUTION UND REDUKTION AM BEISPIEL

$$\begin{aligned} & (\lambda n. \lambda f. \lambda x. n \ f \ (f \ x)) \ (\lambda f. \lambda x. x) \\ \longrightarrow & \lambda f. \lambda x. \ (\lambda f. \lambda x. x) \ f \ (f \ x) \\ \longrightarrow & (\lambda x. x) \ (f \ x) \end{aligned}$$

# SUBSTITUTION UND REDUKTION AM BEISPIEL

$$\begin{aligned} & (\lambda n. \lambda f. \lambda x. n \ f \ (f \ x)) \ (\lambda f. \lambda x. x) \\ \longrightarrow & \lambda f. \lambda x. (\lambda f. \lambda x. x) \ f \ (f \ x) \\ \longrightarrow & \lambda x. (\lambda x. x) \ (f \ x) \end{aligned}$$

# SUBSTITUTION UND REDUKTION AM BEISPIEL

$$\begin{aligned} & (\lambda n. \lambda f. \lambda x. n \ f \ (f \ x)) \ (\lambda f. \lambda x. x) \\ \longrightarrow & \lambda f. \lambda x. (\lambda f. \lambda x. x) \ f \ (f \ x) \\ \longrightarrow & \lambda f. \lambda x. (\lambda x. x) \ (f \ x) \end{aligned}$$

# SUBSTITUTION UND REDUKTION AM BEISPIEL

$$\begin{aligned} & (\lambda n. \lambda f. \lambda x. n \ f \ (f \ x)) \ (\lambda f. \lambda x. x) \\ \longrightarrow & \lambda f. \lambda x. (\lambda f. \lambda x. x) \ f \ (f \ x) \\ \longrightarrow & \lambda f. \lambda x. (\lambda x. x) \ (f \ x) \end{aligned}$$

# SUBSTITUTION UND REDUKTION AM BEISPIEL

$$\begin{aligned} & (\lambda n. \lambda f. \lambda x. n \ f \ (f \ x)) \ (\lambda f. \lambda x. x) \\ \longrightarrow & \lambda f. \lambda x. (\lambda f. \lambda x. x) \ f \ (f \ x) \\ \longrightarrow & \lambda f. \lambda x. (\lambda x. x) \ (f \ x) \end{aligned}$$

# SUBSTITUTION UND REDUKTION AM BEISPIEL

$$\begin{aligned} & (\lambda n. \lambda f. \lambda x. n \ f \ (f \ x)) \ (\lambda f. \lambda x. x) \\ \longrightarrow & \lambda f. \lambda x. (\lambda f. \lambda x. x) \ f \ (f \ x) \\ \longrightarrow & \lambda f. \lambda x. (\lambda x. x) \ (f \ x) \\ \longrightarrow & [\mathbf{x}] [\mathbf{f} \ x / x] \end{aligned}$$

# SUBSTITUTION UND REDUKTION AM BEISPIEL

$$\begin{aligned} & (\lambda n. \lambda f. \lambda x. n \ f \ (f \ x)) \ (\lambda f. \lambda x. x) \\ \longrightarrow & \lambda f. \lambda x. (\lambda f. \lambda x. x) \ f \ (f \ x) \\ \longrightarrow & \lambda f. \lambda x. (\lambda x. x) \ (f \ x) \\ \longrightarrow & f \ x \end{aligned}$$

# SUBSTITUTION UND REDUKTION AM BEISPIEL

$$\begin{aligned} & (\lambda n. \lambda f. \lambda x. n \ f \ (f \ x)) \ (\lambda f. \lambda x. x) \\ \longrightarrow & \lambda f. \lambda x. (\lambda f. \lambda x. x) \ f \ (f \ x) \\ \longrightarrow & \lambda f. \lambda x. (\lambda x. x) \ (f \ x) \\ \longrightarrow & \lambda x. f \ x \end{aligned}$$



# SUBSTITUTION UND REDUKTION AM BEISPIEL

$$\begin{aligned} & (\lambda n. \lambda f. \lambda x. n \ f \ (f \ x)) \ (\lambda f. \lambda x. x) \\ \longrightarrow & \lambda f. \lambda x. (\lambda f. \lambda x. x) \ f \ (f \ x) \\ \longrightarrow & \lambda f. \lambda x. (\lambda x. x) \ (f \ x) \\ \longrightarrow & \lambda f. \lambda x. f \ x \end{aligned}$$

# SUBSTITUTION UND REDUKTION AM BEISPIEL

$$\begin{aligned} & (\lambda n. \lambda f. \lambda x. n \ f \ (f \ x)) \ (\lambda f. \lambda x. x) \\ \longrightarrow & \lambda f. \lambda x. (\lambda f. \lambda x. x) \ f \ (f \ x) \\ \longrightarrow & \lambda f. \lambda x. (\lambda x. x) \ (f \ x) \\ \longrightarrow & \lambda f. \lambda x. f \ x \end{aligned}$$



$$(\lambda n. \lambda f. \lambda x. n \ f \ (f \ x)) \ (\lambda f. \lambda x. x) \xrightarrow{3} \lambda f. \lambda x. f \ x$$

# DARSTELLUNG BOOLESCHER OPERATOREN IM $\lambda$ -KALKÜL

**T**  $\equiv \lambda x. \lambda y. x$

**F**  $\equiv \lambda x. \lambda y. y$

**if**  $b$  **then**  $s$  **else**  $t$   $\equiv b\ s\ t$

# DARSTELLUNG BOOLESCHER OPERATOREN IM $\lambda$ -KALKÜL

$$\mathbf{T} \equiv \lambda x. \lambda y. x$$

$$\mathbf{F} \equiv \lambda x. \lambda y. y$$

$$\text{if } b \text{ then } s \text{ else } t \equiv b \, s \, t$$

Konditional ist invers zu **T** und **F**

$$\begin{aligned} & \text{if } \mathbf{T} \text{ then } s \text{ else } t \\ \equiv & \mathbf{T} \, s \, t \\ \equiv & (\lambda x. \lambda y. x) \, s \, t \\ \longrightarrow & (\lambda y. s) \, t \\ \longrightarrow & s \end{aligned}$$

# DARSTELLUNG BOOLESCHER OPERATOREN IM $\lambda$ -KALKÜL

$$\mathbf{T} \equiv \lambda x. \lambda y. x$$

$$\mathbf{F} \equiv \lambda x. \lambda y. y$$

$$\text{if } b \text{ then } s \text{ else } t \equiv b s t$$

Konditional ist invers zu **T** und **F**

$$\begin{aligned} & \text{if } \mathbf{T} \text{ then } s \text{ else } t \\ \equiv & \mathbf{T} s t \\ \equiv & (\lambda x. \lambda y. x) s t \\ \longrightarrow & (\lambda y. s) t \\ \longrightarrow & s \end{aligned}$$

$$\begin{aligned} & \text{if } \mathbf{F} \text{ then } s \text{ else } t \\ \equiv & \mathbf{F} s t \\ \equiv & (\lambda x. \lambda y. y) s t \\ \longrightarrow & (\lambda y. y) t \\ \longrightarrow & t \end{aligned}$$

## BILDUNG UND ANALYSE VON PAAREN

$$\begin{aligned}\langle s, t \rangle &\equiv \lambda p. p \ s \ t \\ pair.1 &\equiv pair \ (\lambda x. \lambda y. x) \\ pair.2 &\equiv pair \ (\lambda x. \lambda y. y) \\ \text{let } \langle x, y \rangle = pair \text{ in } t &\equiv pair \ (\lambda x. \lambda y. t)\end{aligned}$$

## BILDUNG UND ANALYSE VON PAAREN

$$\langle s, t \rangle \equiv \lambda p. p s t$$

$$pair.1 \equiv pair (\lambda x. \lambda y. x)$$

$$pair.2 \equiv pair (\lambda x. \lambda y. y)$$

$$\mathbf{let} \langle x, y \rangle = pair \mathbf{in} t \equiv pair (\lambda x. \lambda y. t)$$

Analyseoperator ist invers zur Paarbildung

$$\mathbf{let} \langle x, y \rangle = \langle u, v \rangle \mathbf{in} t$$

$$\equiv \langle u, v \rangle (\lambda x. \lambda y. t)$$

$$\equiv (\lambda p. p u v) (\lambda x. \lambda y. t)$$

$$\longrightarrow (\lambda x. \lambda y. t) u v$$

$$\longrightarrow (\lambda y. t[u/x]) v$$

$$\longrightarrow t[u, v/x, y]$$

- **Darstellung von Zahlen durch iterierte Terme**
  - Semantisch: wiederholte Anwendung von Funktionen



- **Darstellung von Zahlen durch iterierte Terme**

- Semantisch: wiederholte Anwendung von Funktionen
- Repräsentiere die **Zahl**  $n$  durch den Term  $\lambda f . \lambda x . \underbrace{f (f \dots (f x) \dots)}_{n\text{-mal}}$

## ● Darstellung von Zahlen durch iterierte Terme

- Semantisch: wiederholte Anwendung von Funktionen
- Repräsentiere die Zahl  $n$  durch den Term  $\lambda f . \lambda x . \underbrace{f (f \dots (f x) \dots)}_{n\text{-mal}}$
- Notation:  $\overline{n} \equiv \lambda f . \lambda x . f^n x$

## ● Darstellung von Zahlen durch iterierte Terme

- Semantisch: wiederholte Anwendung von Funktionen
- Repräsentiere die Zahl  $n$  durch den Term  $\lambda f . \lambda x . \underbrace{f (f \dots (f x) \dots)}_{n\text{-mal}}$
- Notation:  $\overline{n} \equiv \lambda f . \lambda x . f^n x$
- Bezeichnung: **Church Numerals**

- **Darstellung von Zahlen durch iterierte Terme**

- Semantisch: wiederholte Anwendung von Funktionen
- Repräsentiere die Zahl  $n$  durch den Term  $\lambda f . \lambda x . \underbrace{f (f \dots (f x) \dots)}_{n\text{-mal}}$
- Notation:  $\overline{n} \equiv \lambda f . \lambda x . f^n x$
- Bezeichnung: **Church Numerals**

- $f: \mathbb{N}^n \rightarrow \mathbb{N}$   **$\lambda$ -berechenbar:**

- Es gibt einen  $\lambda$ -Term  $t$  mit  $f(x_1, \dots, x_n) = m \Leftrightarrow t \overline{x_1} \dots \overline{x_n} = \overline{m}$

- **Darstellung von Zahlen durch iterierte Terme**

- Semantisch: wiederholte Anwendung von Funktionen
- Repräsentiere die Zahl  $n$  durch den Term  $\underbrace{\lambda f . \lambda x . f (f \dots (f x) \dots)}_{n\text{-mal}}$
- Notation:  $\overline{n} \equiv \lambda f . \lambda x . f^n x$
- Bezeichnung: **Church Numerals**

- $f: \mathbb{N}^n \rightarrow \mathbb{N}$   **$\lambda$ -berechenbar:**

- Es gibt einen  $\lambda$ -Term  $t$  mit  $f(x_1, \dots, x_n) = m \Leftrightarrow t \overline{x_1} \dots \overline{x_n} = \overline{m}$

- **Operationen müssen Termvielfachheit verändern**

- z.B. **add**  $\overline{m} \overline{n}$  muß als Wert immer den Term  $\overline{m+n}$  ergeben

- Nachfolgerfunktion:  $s \equiv \lambda n. \lambda f. \lambda x. n f (f x)$

- **Nachfolgerfunktion:**  $\mathbf{s} \equiv \lambda n. \lambda f. \lambda x. n \ f \ (f \ x)$ 
  - Zeige: Der Wert von  $\mathbf{s} \ \overline{n}$  ist der Term  $\overline{n+1}$

● **Nachfolgerfunktion:**  $\mathbf{s} \equiv \lambda n. \lambda f. \lambda x. n \ f \ (f \ x)$

– Zeige: Der Wert von  $\mathbf{s} \ \overline{n}$  ist der Term  $\overline{n+1}$

$$\mathbf{s} \ \overline{n} \equiv (\lambda n. \lambda f. \lambda x. n \ f \ (f \ x)) (\lambda f. \lambda x. f^n x)$$



● **Nachfolgerfunktion:**  $\mathbf{s} \equiv \lambda n. \lambda f. \lambda x. n \ f \ (f \ x)$

– Zeige: Der Wert von  $\mathbf{s} \ \overline{n}$  ist der Term  $\overline{n+1}$

$$\begin{aligned} \mathbf{s} \ \overline{n} &\equiv (\lambda n. \lambda f. \lambda x. n \ f \ (f \ x)) (\lambda f. \lambda x. f^n x) \\ &\longrightarrow \lambda f. \lambda x. (\lambda f. \lambda x. f^n x) f \ (f \ x) \end{aligned}$$

● **Nachfolgerfunktion:**  $\mathbf{s} \equiv \lambda n. \lambda f. \lambda x. n \ f \ (f \ x)$

– Zeige: Der Wert von  $\mathbf{s} \ \overline{n}$  ist der Term  $\overline{n+1}$

$$\begin{aligned} \mathbf{s} \ \overline{n} &\equiv (\lambda n. \lambda f. \lambda x. n \ f \ (f \ x)) (\lambda f. \lambda x. f^n x) \\ &\longrightarrow \lambda f. \lambda x. (\lambda f. \lambda x. f^n x) f \ (f \ x) \\ &\longrightarrow \lambda f. \lambda x. (\lambda x. f^n x) (f \ x) \end{aligned}$$

● **Nachfolgerfunktion:**  $\mathbf{s} \equiv \lambda n. \lambda f. \lambda x. n \ f \ (f \ x)$

– Zeige: Der Wert von  $\mathbf{s} \ \overline{n}$  ist der Term  $\overline{n+1}$

$$\begin{aligned} \mathbf{s} \ \overline{n} &\equiv (\lambda n. \lambda f. \lambda x. n \ f \ (f \ x)) (\lambda f. \lambda x. f^n x) \\ &\longrightarrow \lambda f. \lambda x. (\lambda f. \lambda x. f^n x) f \ (f \ x) \\ &\longrightarrow \lambda f. \lambda x. (\lambda x. f^n x) (f \ x) \\ &\longrightarrow \lambda f. \lambda x. f^n (f \ x) \end{aligned}$$

● **Nachfolgerfunktion:**  $\mathbf{s} \equiv \lambda n. \lambda f. \lambda x. n \ f \ (f \ x)$

– Zeige: Der Wert von  $\mathbf{s} \ \overline{n}$  ist der Term  $\overline{n+1}$

$$\begin{aligned} \mathbf{s} \ \overline{n} &\equiv (\lambda n. \lambda f. \lambda x. n \ f \ (f \ x)) (\lambda f. \lambda x. f^n x) \\ &\longrightarrow \lambda f. \lambda x. (\lambda f. \lambda x. f^n x) f \ (f \ x) \\ &\longrightarrow \lambda f. \lambda x. (\lambda x. f^n x) (f \ x) \\ &\longrightarrow \lambda f. \lambda x. f^n (f \ x) \\ &\longrightarrow \lambda f. \lambda x. f^{n+1} x \end{aligned}$$

● **Nachfolgerfunktion:**  $\mathbf{s} \equiv \lambda n. \lambda f. \lambda x. n \ f \ (f \ x)$

– Zeige: Der Wert von  $\mathbf{s} \ \overline{n}$  ist der Term  $\overline{n+1}$

$$\begin{aligned} \mathbf{s} \ \overline{n} &\equiv (\lambda n. \lambda f. \lambda x. n \ f \ (f \ x)) (\lambda f. \lambda x. f^n x) \\ &\longrightarrow \lambda f. \lambda x. (\lambda f. \lambda x. f^n x) f \ (f \ x) \\ &\longrightarrow \lambda f. \lambda x. (\lambda x. f^n x) (f \ x) \\ &\longrightarrow \lambda f. \lambda x. f^n (f \ x) \\ &\longrightarrow \lambda f. \lambda x. f^{n+1} x \qquad \qquad \qquad \equiv \overline{n+1} \end{aligned}$$

● **Nachfolgerfunktion:**  $\mathbf{s} \equiv \lambda n. \lambda f. \lambda x. n \ f \ (f \ x)$

– Zeige: Der Wert von  $\mathbf{s} \ \overline{n}$  ist der Term  $\overline{n+1}$

$$\begin{aligned} \mathbf{s} \ \overline{n} &\equiv (\lambda n. \lambda f. \lambda x. n \ f \ (f \ x)) (\lambda f. \lambda x. f^n x) \\ &\longrightarrow \lambda f. \lambda x. (\lambda f. \lambda x. f^n x) f \ (f \ x) \\ &\longrightarrow \lambda f. \lambda x. (\lambda x. f^n x) (f \ x) \\ &\longrightarrow \lambda f. \lambda x. f^n (f \ x) \\ &\longrightarrow \lambda f. \lambda x. f^{n+1} x \equiv \overline{n+1} \end{aligned}$$

● **Addition:**  $\mathbf{add} \equiv \lambda m. \lambda n. \lambda f. \lambda x. m \ f \ (n \ f \ x)$

● **Nachfolgerfunktion:**  $\mathbf{s} \equiv \lambda n. \lambda f. \lambda x. n \ f \ (f \ x)$

– Zeige: Der Wert von  $\mathbf{s} \ \overline{n}$  ist der Term  $\overline{n+1}$

$$\begin{aligned} \mathbf{s} \ \overline{n} &\equiv (\lambda n. \lambda f. \lambda x. n \ f \ (f \ x)) (\lambda f. \lambda x. f^n x) \\ &\longrightarrow \lambda f. \lambda x. (\lambda f. \lambda x. f^n x) f \ (f \ x) \\ &\longrightarrow \lambda f. \lambda x. (\lambda x. f^n x) (f \ x) \\ &\longrightarrow \lambda f. \lambda x. f^n (f \ x) \\ &\longrightarrow \lambda f. \lambda x. f^{n+1} x \qquad \qquad \qquad \equiv \overline{n+1} \end{aligned}$$

● **Addition:**  $\mathbf{add} \equiv \lambda m. \lambda n. \lambda f. \lambda x. m \ f \ (n \ f \ x)$

● **Multiplikation:**  $\mathbf{mul} \equiv \lambda m. \lambda n. \lambda f. \lambda x. m \ (n \ f) \ x$

● **Nachfolgerfunktion:**  $\mathbf{s} \equiv \lambda n. \lambda f. \lambda x. n \ f \ (f \ x)$

– Zeige: Der Wert von  $\mathbf{s} \ \overline{n}$  ist der Term  $\overline{n+1}$

$$\begin{aligned} \mathbf{s} \ \overline{n} &\equiv (\lambda n. \lambda f. \lambda x. n \ f \ (f \ x)) (\lambda f. \lambda x. f^n x) \\ &\longrightarrow \lambda f. \lambda x. (\lambda f. \lambda x. f^n x) f \ (f \ x) \\ &\longrightarrow \lambda f. \lambda x. (\lambda x. f^n x) (f \ x) \\ &\longrightarrow \lambda f. \lambda x. f^n (f \ x) \\ &\longrightarrow \lambda f. \lambda x. f^{n+1} x \equiv \overline{n+1} \end{aligned}$$

● **Addition:**  $\mathbf{add} \equiv \lambda m. \lambda n. \lambda f. \lambda x. m \ f \ (n \ f \ x)$

● **Multiplikation:**  $\mathbf{mul} \equiv \lambda m. \lambda n. \lambda f. \lambda x. m \ (n \ f) \ x$

● **Test auf Null:**  $\mathbf{zero} \equiv \lambda n. n \ (\lambda n. \mathbf{F}) \ \mathbf{T}$



- **Nachfolgerfunktion:**  $\mathbf{s} \equiv \lambda n. \lambda f. \lambda x. n \ f \ (f \ x)$

– Zeige: Der Wert von  $\mathbf{s} \ \overline{n}$  ist der Term  $\overline{n+1}$

$$\begin{aligned}
 \mathbf{s} \ \overline{n} &\equiv (\lambda n. \lambda f. \lambda x. n \ f \ (f \ x)) (\lambda f. \lambda x. f^n x) \\
 &\longrightarrow \lambda f. \lambda x. (\lambda f. \lambda x. f^n x) f \ (f \ x) \\
 &\longrightarrow \lambda f. \lambda x. (\lambda x. f^n x) (f \ x) \\
 &\longrightarrow \lambda f. \lambda x. f^n (f \ x) \\
 &\longrightarrow \lambda f. \lambda x. f^{n+1} x \equiv \overline{n+1}
 \end{aligned}$$

- **Addition:**  $\mathbf{add} \equiv \lambda m. \lambda n. \lambda f. \lambda x. m \ f \ (n \ f \ x)$

- **Multiplikation:**  $\mathbf{mul} \equiv \lambda m. \lambda n. \lambda f. \lambda x. m \ (n \ f) \ x$

- **Test auf Null:**  $\mathbf{zero} \equiv \lambda n. n \ (\lambda n. \mathbf{F}) \ \mathbf{T}$

- **Vorgängerkfunktion:**

$$\mathbf{p} \equiv \lambda n. (n \ (\lambda f x. \langle \mathbf{s}, \text{let } \langle f, x \rangle = f x \text{ in } f \ x \rangle) \ (\lambda z. \overline{0}, \overline{0})).2$$

## AUSWERTUNG DER ADDITIONSFUNKTION

- Zeige: **add**  $\overline{m}$   $\overline{n}$  reduziert zu  $\overline{m+n}$

## AUSWERTUNG DER ADDITIONSFUNKTION

- Zeige: **add**  $\overline{m}$   $\overline{n}$  reduziert zu  $\overline{m+n}$

$$\mathbf{add} \ \overline{m} \ \overline{n} \quad \equiv \quad (\lambda m. \lambda n. \lambda f. \lambda x. \ m \ f \ (n \ f \ x)) \ \overline{m} \ \overline{n}$$

# AUSWERTUNG DER ADDITIONSFUNKTION

- Zeige: **add**  $\overline{m}$   $\overline{n}$  reduziert zu  $\overline{m+n}$

$$\begin{aligned}\mathbf{add} \ \overline{m} \ \overline{n} &\equiv (\lambda m. \lambda n. \lambda f. \lambda x. m \ f \ (n \ f \ x)) \ \overline{m} \ \overline{n} \\ &\longrightarrow (\lambda n. \lambda f. \lambda x. \overline{m} \ f \ (n \ f \ x)) \ \overline{n}\end{aligned}$$

# AUSWERTUNG DER ADDITIONSFUNKTION

- Zeige: **add**  $\overline{m}$   $\overline{n}$  reduziert zu  $\overline{m+n}$

$$\begin{aligned}\mathbf{add} \ \overline{m} \ \overline{n} &\equiv (\lambda m. \lambda n. \lambda f. \lambda x. m \ f \ (n \ f \ x)) \ \overline{m} \ \overline{n} \\ &\longrightarrow (\lambda n. \lambda f. \lambda x. \overline{m} \ f \ (n \ f \ x)) \ \overline{n} \\ &\longrightarrow \lambda f. \lambda x. \overline{m} \ f \ (\overline{n} \ f \ x)\end{aligned}$$

## AUSWERTUNG DER ADDITIONSFUNKTION

- Zeige: **add**  $\overline{m}$   $\overline{n}$  reduziert zu  $\overline{m+n}$

$$\begin{aligned}\mathbf{add} \ \overline{m} \ \overline{n} &\equiv (\lambda m. \lambda n. \lambda f. \lambda x. m \ f \ (n \ f \ x)) \ \overline{m} \ \overline{n} \\ &\longrightarrow (\lambda n. \lambda f. \lambda x. \overline{m} \ f \ (n \ f \ x)) \ \overline{n} \\ &\longrightarrow \lambda f. \lambda x. \overline{m} \ f \ (\overline{n} \ f \ x) \\ &\equiv \lambda f. \lambda x. (\lambda f. \lambda x. f^m x) \ f \ (\overline{n} \ f \ x)\end{aligned}$$

# AUSWERTUNG DER ADDITIONSFUNKTION

- Zeige: **add**  $\overline{m}$   $\overline{n}$  reduziert zu  $\overline{m+n}$

$$\begin{aligned}\mathbf{add} \ \overline{m} \ \overline{n} &\equiv (\lambda m. \lambda n. \lambda f. \lambda x. m \ f \ (n \ f \ x)) \ \overline{m} \ \overline{n} \\ &\longrightarrow (\lambda n. \lambda f. \lambda x. \overline{m} \ f \ (n \ f \ x)) \ \overline{n} \\ &\longrightarrow \lambda f. \lambda x. \overline{m} \ f \ (\overline{n} \ f \ x) \\ &\equiv \lambda f. \lambda x. (\lambda f. \lambda x. f^m x) \ f \ (\overline{n} \ f \ x) \\ &\longrightarrow \lambda f. \lambda x. (\lambda x. f^m x) \ (\overline{n} \ f \ x)\end{aligned}$$

## AUSWERTUNG DER ADDITIONSFUNKTION

- Zeige: **add**  $\overline{m}$   $\overline{n}$  reduziert zu  $\overline{m+n}$

$$\begin{aligned}\mathbf{add} \ \overline{m} \ \overline{n} &\equiv (\lambda m. \lambda n. \lambda f. \lambda x. m \ f \ (n \ f \ x)) \ \overline{m} \ \overline{n} \\ &\longrightarrow (\lambda n. \lambda f. \lambda x. \overline{m} \ f \ (n \ f \ x)) \ \overline{n} \\ &\longrightarrow \lambda f. \lambda x. \overline{m} \ f \ (\overline{n} \ f \ x) \\ &\equiv \lambda f. \lambda x. (\lambda f. \lambda x. f^m x) \ f \ (\overline{n} \ f \ x) \\ &\longrightarrow \lambda f. \lambda x. (\lambda x. f^m x) \ (\overline{n} \ f \ x) \\ &\longrightarrow \lambda f. \lambda x. f^m (\overline{n} \ f \ x)\end{aligned}$$



# AUSWERTUNG DER ADDITIONSFUNKTION

- Zeige: **add**  $\overline{m}$   $\overline{n}$  reduziert zu  $\overline{m+n}$

$$\begin{aligned}\mathbf{add} \ \overline{m} \ \overline{n} &\equiv (\lambda m. \lambda n. \lambda f. \lambda x. m \ f \ (n \ f \ x)) \ \overline{m} \ \overline{n} \\ &\longrightarrow (\lambda n. \lambda f. \lambda x. \overline{m} \ f \ (n \ f \ x)) \ \overline{n} \\ &\longrightarrow \lambda f. \lambda x. \overline{m} \ f \ (\overline{n} \ f \ x) \\ &\equiv \lambda f. \lambda x. (\lambda f. \lambda x. f^m x) \ f \ (\overline{n} \ f \ x) \\ &\longrightarrow \lambda f. \lambda x. (\lambda x. f^m x) \ (\overline{n} \ f \ x) \\ &\longrightarrow \lambda f. \lambda x. f^m (\overline{n} \ f \ x) \\ &\equiv \lambda f. \lambda x. f^m ((\lambda f. \lambda x. f^n x) \ f \ x)\end{aligned}$$

# AUSWERTUNG DER ADDITIONSFUNKTION

- Zeige: **add**  $\overline{m}$   $\overline{n}$  reduziert zu  $\overline{m+n}$

$$\begin{aligned}\mathbf{add} \ \overline{m} \ \overline{n} &\equiv (\lambda m. \lambda n. \lambda f. \lambda x. m \ f \ (n \ f \ x)) \ \overline{m} \ \overline{n} \\ &\longrightarrow (\lambda n. \lambda f. \lambda x. \overline{m} \ f \ (n \ f \ x)) \ \overline{n} \\ &\longrightarrow \lambda f. \lambda x. \overline{m} \ f \ (\overline{n} \ f \ x) \\ &\equiv \lambda f. \lambda x. (\lambda f. \lambda x. f^m x) \ f \ (\overline{n} \ f \ x) \\ &\longrightarrow \lambda f. \lambda x. (\lambda x. f^m x) \ (\overline{n} \ f \ x) \\ &\longrightarrow \lambda f. \lambda x. f^m (\overline{n} \ f \ x) \\ &\equiv \lambda f. \lambda x. f^m ((\lambda f. \lambda x. f^n x) \ f \ x) \\ &\longrightarrow \lambda f. \lambda x. f^m ((\lambda x. f^n x) \ x)\end{aligned}$$

# AUSWERTUNG DER ADDITIONSFUNKTION

- Zeige: **add**  $\overline{m}$   $\overline{n}$  reduziert zu  $\overline{m+n}$

$$\begin{aligned}\mathbf{add} \ \overline{m} \ \overline{n} &\equiv (\lambda m. \lambda n. \lambda f. \lambda x. m \ f \ (n \ f \ x)) \ \overline{m} \ \overline{n} \\&\longrightarrow (\lambda n. \lambda f. \lambda x. \overline{m} \ f \ (n \ f \ x)) \ \overline{n} \\&\longrightarrow \lambda f. \lambda x. \overline{m} \ f \ (\overline{n} \ f \ x) \\&\equiv \lambda f. \lambda x. (\lambda f. \lambda x. f^m x) \ f \ (\overline{n} \ f \ x) \\&\longrightarrow \lambda f. \lambda x. (\lambda x. f^m x) \ (\overline{n} \ f \ x) \\&\longrightarrow \lambda f. \lambda x. f^m (\overline{n} \ f \ x) \\&\equiv \lambda f. \lambda x. f^m ((\lambda f. \lambda x. f^n x) \ f \ x) \\&\longrightarrow \lambda f. \lambda x. f^m ((\lambda x. f^n x) \ x) \\&\longrightarrow \lambda f. \lambda x. f^m (f^n x)\end{aligned}$$

# AUSWERTUNG DER ADDITIONSFUNKTION

- Zeige: **add**  $\overline{m}$   $\overline{n}$  reduziert zu  $\overline{m+n}$

$$\begin{aligned}\mathbf{add} \ \overline{m} \ \overline{n} &\equiv (\lambda m. \lambda n. \lambda f. \lambda x. m \ f \ (n \ f \ x)) \ \overline{m} \ \overline{n} \\&\longrightarrow (\lambda n. \lambda f. \lambda x. \overline{m} \ f \ (n \ f \ x)) \ \overline{n} \\&\longrightarrow \lambda f. \lambda x. \overline{m} \ f \ (\overline{n} \ f \ x) \\&\equiv \lambda f. \lambda x. (\lambda f. \lambda x. f^m x) \ f \ (\overline{n} \ f \ x) \\&\longrightarrow \lambda f. \lambda x. (\lambda x. f^m x) \ (\overline{n} \ f \ x) \\&\longrightarrow \lambda f. \lambda x. f^m (\overline{n} \ f \ x) \\&\equiv \lambda f. \lambda x. f^m ((\lambda f. \lambda x. f^n x) \ f \ x) \\&\longrightarrow \lambda f. \lambda x. f^m ((\lambda x. f^n x) \ x) \\&\longrightarrow \lambda f. \lambda x. f^m (f^n x) \\&\longrightarrow \lambda f. \lambda x. f^{m+n} x\end{aligned}$$

# AUSWERTUNG DER ADDITIONSFUNKTION

- Zeige: **add**  $\overline{m}$   $\overline{n}$  reduziert zu  $\overline{m+n}$

$$\begin{aligned} \mathbf{add} \ \overline{m} \ \overline{n} &\equiv (\lambda m. \lambda n. \lambda f. \lambda x. m \ f \ (n \ f \ x)) \ \overline{m} \ \overline{n} \\ &\longrightarrow (\lambda n. \lambda f. \lambda x. \overline{m} \ f \ (n \ f \ x)) \ \overline{n} \\ &\longrightarrow \lambda f. \lambda x. \overline{m} \ f \ (\overline{n} \ f \ x) \\ &\equiv \lambda f. \lambda x. (\lambda f. \lambda x. f^m x) \ f \ (\overline{n} \ f \ x) \\ &\longrightarrow \lambda f. \lambda x. (\lambda x. f^m x) \ (\overline{n} \ f \ x) \\ &\longrightarrow \lambda f. \lambda x. f^m (\overline{n} \ f \ x) \\ &\equiv \lambda f. \lambda x. f^m ((\lambda f. \lambda x. f^n x) \ f \ x) \\ &\longrightarrow \lambda f. \lambda x. f^m ((\lambda x. f^n x) \ x) \\ &\longrightarrow \lambda f. \lambda x. f^m (f^n x) \\ &\longrightarrow \lambda f. \lambda x. f^{m+n} x \qquad \qquad \qquad \equiv \overline{m+n} \end{aligned}$$

**Y-Kombinator:**  $\mathbf{Y} \equiv \lambda f. (\lambda x. f (x x)) (\lambda x. f (x x))$

- $\mathbf{Y}$  ist **Fixpunktkombinator**
  - $\mathbf{Y} t = t (\mathbf{Y} t)$  für beliebige Terme  $t$

**Y-Kombinator:**  $\mathbf{Y} \equiv \lambda f. (\lambda x. f (x x)) (\lambda x. f (x x))$

- $\mathbf{Y}$  ist **Fixpunktkombinator**

- $\mathbf{Y} t = t (\mathbf{Y} t)$  für beliebige Terme  $t$

$$\begin{aligned} \mathbf{Y} t &\equiv \lambda f. (\lambda x. f (x x)) (\lambda x. f (x x)) t \\ &\longrightarrow (\lambda x. t (x x)) (\lambda x. t (x x)) \\ &\longrightarrow t ( (\lambda x. t (x x)) (\lambda x. t (x x)) ) \end{aligned}$$

$$\begin{aligned} t (\mathbf{Y} t) &\equiv t (\lambda f. (\lambda x. f (x x)) (\lambda x. f (x x)) t) \\ &\longrightarrow t ( (\lambda x. t (x x)) (\lambda x. t (x x)) ) \end{aligned}$$

**Y-Kombinator:**  $\mathbf{Y} \equiv \lambda f. (\lambda x. f (x x)) (\lambda x. f (x x))$

- $\mathbf{Y}$  ist **Fixpunktkombinator**

- $\mathbf{Y} t = t (\mathbf{Y} t)$  für beliebige Terme  $t$

$$\begin{aligned}\mathbf{Y} t &\equiv \lambda f. (\lambda x. f (x x)) (\lambda x. f (x x)) t \\ &\longrightarrow (\lambda x. t (x x)) (\lambda x. t (x x)) \\ &\longrightarrow t ( (\lambda x. t (x x)) (\lambda x. t (x x)) )\end{aligned}$$

$$\begin{aligned}t (\mathbf{Y} t) &\equiv t (\lambda f. (\lambda x. f (x x)) (\lambda x. f (x x)) t) \\ &\longrightarrow t ( (\lambda x. t (x x)) (\lambda x. t (x x)) )\end{aligned}$$

- Rekursion darstellbar als

$$\mathbf{letrec} \ f(x) = t \equiv \mathbf{Y}(\lambda f. \lambda x. t)$$



Alle  $\mu$ -rekursiven Funktionen sind  $\lambda$ -berechenbar

- Nachfolgerfunktion  $s$ :  $s \equiv \lambda n. \lambda f. \lambda x. n \ f \ (f \ x)$

Alle  $\mu$ -rekursiven Funktionen sind  $\lambda$ -berechenbar

- Nachfolgerfunktion  $s$ :  $s \equiv \lambda n. \lambda f. \lambda x. n \ f \ (f \ x)$
- Projektionsfunktionen  $pr_m^n$   $pr_m^n \equiv \lambda x_1. \dots \lambda x_n. x_m$

## Alle $\mu$ -rekursiven Funktionen sind $\lambda$ -berechenbar

- Nachfolgerfunktion  $s$ :  $s \equiv \lambda n. \lambda f. \lambda x. n \ f \ (f \ x)$
- Projektionsfunktionen  $pr_m^n$ :  $pr_m^n \equiv \lambda x_1. \dots \lambda x_n. x_m$
- Konstantenfunktion  $c_m^n$ :  $c_m^n \equiv \lambda x_1. \dots \lambda x_n. \bar{m}$

## Alle $\mu$ -rekursiven Funktionen sind $\lambda$ -berechenbar

- Nachfolgerfunktion  $s$ :  $s \equiv \lambda n. \lambda f. \lambda x. n \ f \ (f \ x)$
- Projektionsfunktionen  $pr_m^n$ :  $pr_m^n \equiv \lambda x_1. \dots \lambda x_n. x_m$
- Konstantenfunktion  $c_m^n$ :  $c_m^n \equiv \lambda x_1. \dots \lambda x_n. \bar{m}$
- Komposition  $f \circ (g_1 \dots g_n)$ :
  - $\circ \equiv \lambda f. \lambda g_1. \dots \lambda g_n. \lambda x. f \ (g_1 \ x) \dots (g_n \ x)$

## Alle $\mu$ -rekursiven Funktionen sind $\lambda$ -berechenbar

- Nachfolgerfunktion  $s$ :  $s \equiv \lambda n. \lambda f. \lambda x. n \ f \ (f \ x)$
- Projektionsfunktionen  $pr_m^n$ :  $pr_m^n \equiv \lambda x_1. \dots \lambda x_n. x_m$
- Konstantenfunktion  $c_m^n$ :  $c_m^n \equiv \lambda x_1. \dots \lambda x_n. \bar{m}$
- Komposition  $f \circ (g_1 \dots g_n)$ :
  - $\circ \equiv \lambda f. \lambda g_1. \dots \lambda g_n. \lambda x. f \ (g_1 \ x) \dots (g_n \ x)$
- Primitive Rekursion  $Pr[f, g]$ :
  - $PR \equiv \lambda f. \lambda g.$   
 $\text{letrec } h(x) = \lambda y. \text{if zero } y \text{ then } f \ x \text{ else } g \ x \ (p \ y) \ (h \ x \ (p \ y))$

## Alle $\mu$ -rekursiven Funktionen sind $\lambda$ -berechenbar

- Nachfolgerfunktion  $s$ :  $s \equiv \lambda n. \lambda f. \lambda x. n \ f \ (f \ x)$
- Projektionsfunktionen  $pr_m^n$   $pr_m^n \equiv \lambda x_1. \dots \lambda x_n. x_m$
- Konstantenfunktion  $c_m^n$ :  $c_m^n \equiv \lambda x_1. \dots \lambda x_n. \bar{m}$
- Komposition  $f \circ (g_1 \dots g_n)$ :
  - $\circ \equiv \lambda f. \lambda g_1. \dots \lambda g_n. \lambda x. f \ (g_1 \ x) \dots (g_n \ x)$
- Primitive Rekursion  $Pr[f, g]$ :
  - $PR \equiv \lambda f. \lambda g.$   
 $\text{letrec } h(x) = \lambda y. \text{if zero } y \text{ then } f \ x \text{ else } g \ x \ (p \ y) \ (h \ x \ (p \ y))$
- Minimierung  $\mu[f]$ :
  - $Mu \equiv \lambda f. \lambda x.$   
 $(\text{letrec } \min(y) = \text{if zero}(f \ x \ y) \text{ then } y \text{ else } \min(s \ y)) \ \bar{0}$

- **Nichtdeterministische Turingmaschine**
  - Arbeitsweise wie gewöhnliche Turingmaschine
  - Zustandsüberföhrungsfunktion erlaubt alternative Resultate

- **Nichtdeterministische Turingmaschine**

- Arbeitsweise wie gewöhnliche Turingmaschine
- Zustandsüberföhrungsfunktion erlaubt alternative Resultate

- **Abakus**

- Erweiterung des mechanischen Abakus: beliebig viele Stangen und Kugeln
- Zwei Operationen: Kugel hinzunehmen / Kugel wegnehmen



- **Nichtdeterministische Turingmaschine**

- Arbeitsweise wie gewöhnliche Turingmaschine
- Zustandsüberföhrungsfunktion erlaubt alternative Resultate

- **Abakus**

- Erweiterung des mechanischen Abakus: beliebig viele Stangen und Kugeln
- Zwei Operationen: Kugel hinzunehmen / Kugel wegnehmen

- **Markov-Algorithmen**

- Wie Typ-0 Grammatiken, aber mit fester Strategie für Regelanwendung
- Verarbeitet Eingabeworte, statt mit einem Startsymbol zu beginnen

- **Nichtdeterministische Turingmaschine**

- Arbeitsweise wie gewöhnliche Turingmaschine
- Zustandsüberföhrungsfunktion erlaubt alternative Resultate

- **Abakus**

- Erweiterung des mechanischen Abakus: beliebig viele Stangen und Kugeln
- Zwei Operationen: Kugel hinzunehmen / Kugel wegnehmen

- **Markov-Algorithmen**

- Wie Typ-0 Grammatiken, aber mit fester Strategie für Regelanwendung
- Verarbeitet Eingabeworte, statt mit einem Startsymbol zu beginnen

- **Arithmetische Repräsentierbarkeit**

- Spezifikation von Funktionen in arithmetisch-logischem Kalkül
- $f$  ist repräsentierbar, wenn das Ein-/Ausgabeverhalten von  $f$  eindeutig durch eine Formel spezifiziert werden kann
- Eindeutigkeit muß ausschließlich aus logischen Axiomen beweisbar sein

# DIE CHURCH'SCHE THESE

- **Alle Berechenbarkeitsmodelle sind äquivalent**
  - Keines kann mehr berechnen als Turingmaschinen
  - Es ist keine intuitiv berechenbare Funktion bekannt, die nicht von Turingmaschinen berechnet werden kann

# DIE CHURCH'SCHE THESE

- **Alle Berechenbarkeitsmodelle sind äquivalent**
  - Keines kann mehr berechnen als Turingmaschinen
  - Es ist keine intuitiv berechenbare Funktion bekannt, die nicht von Turingmaschinen berechnet werden kann
- **Church'sche These:**

Die Klasse der Turing-berechenbaren Funktionen stimmt mit der Klasse der intuitiv berechenbaren Funktionen überein

# DIE CHURCH'SCHE THESE

- **Alle Berechenbarkeitsmodelle sind äquivalent**

- Keines kann mehr berechnen als Turingmaschinen
- Es ist keine intuitiv berechenbare Funktion bekannt, die nicht von Turingmaschinen berechnet werden kann

- **Church'sche These:**

**Die Klasse der Turing-berechenbaren Funktionen stimmt mit der Klasse der intuitiv berechenbaren Funktionen überein**

- **Unbeweisbar**, aber wahrscheinlich richtige Behauptung
- **Arbeitshypothese** für theoretische Argumente
  - man darf in Beweisen “intuitive” Programme angeben