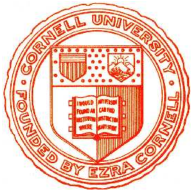
Theoretische Informatik



Einheit 2

Berechenbarkeitsmodelle

1. Turingmaschinen
2. Registermaschinen
3. μ-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)
- **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)
- **μ-rekursive Funktionen** (Mathematisches Rechnen)
- **λ-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

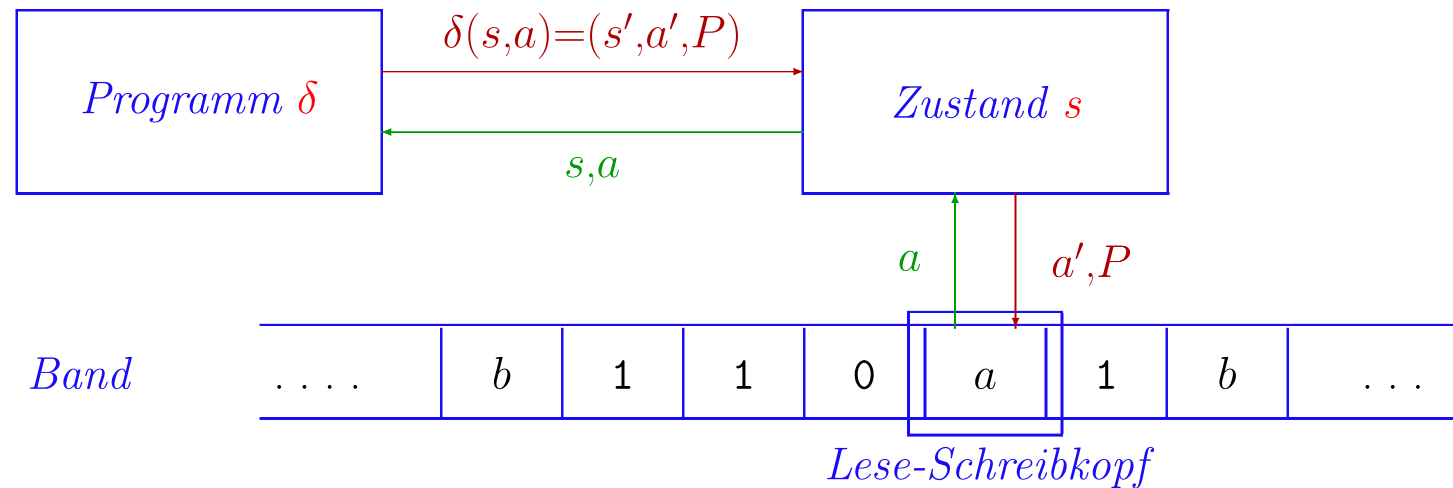
Theoretische Informatik

Einheit 2.1

Turingmaschinen

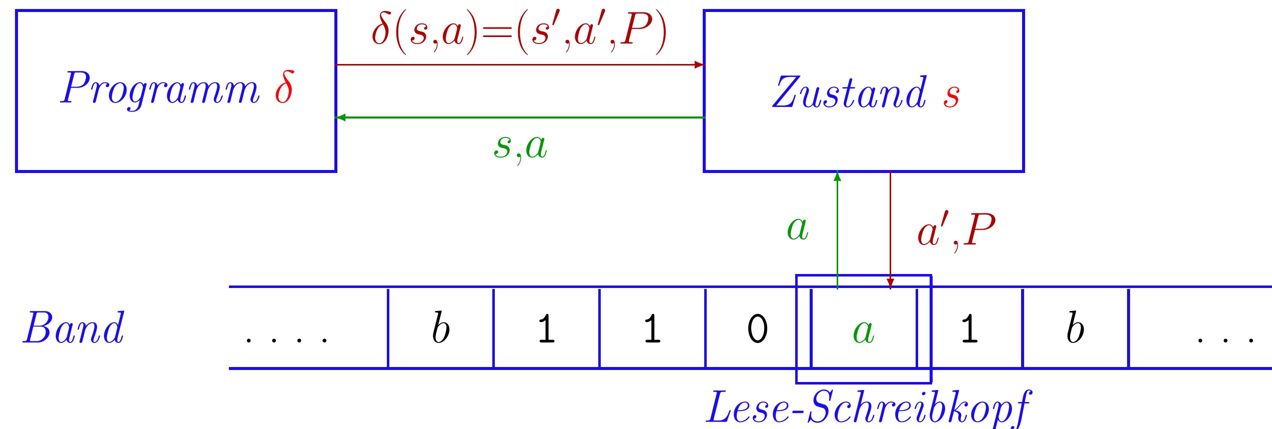
1. Arbeitsweise
2. Formale Semantik
3. Turing-Berechenbarkeit
4. Varianten von Turingmaschinen

Turingmaschinen



- **Automaten mit potentiell unendlichem Band**
  - Band fast überall unbeschrieben (Leersymbol b ≡ "Blank")
  - Lese-Schreibkopf kann Symbole lesen, schreiben und bewegt werden

Turingmaschinen – mathematisch



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

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

Beschreibung von Turingmaschinen

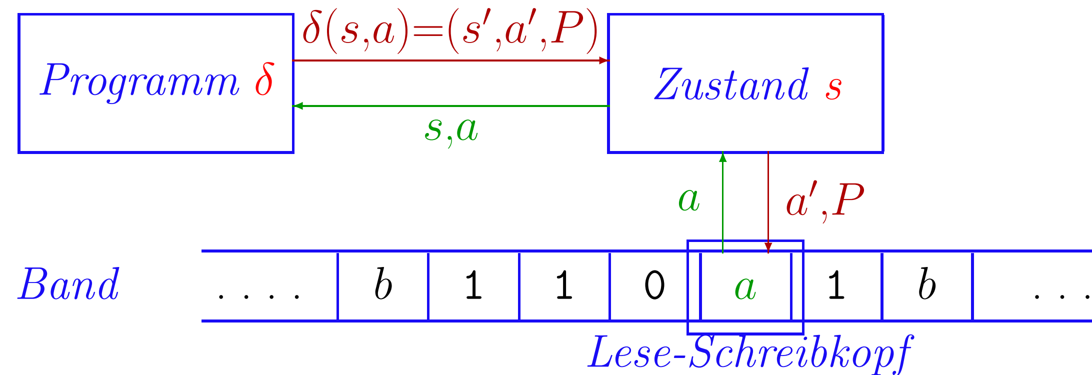
Übergangstabelle für δ

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	Γ	$= \{0, 1, b\}$
Eingabealphabet	X	$= \{0, 1\}$
Blanksymbol	b	$=$ b

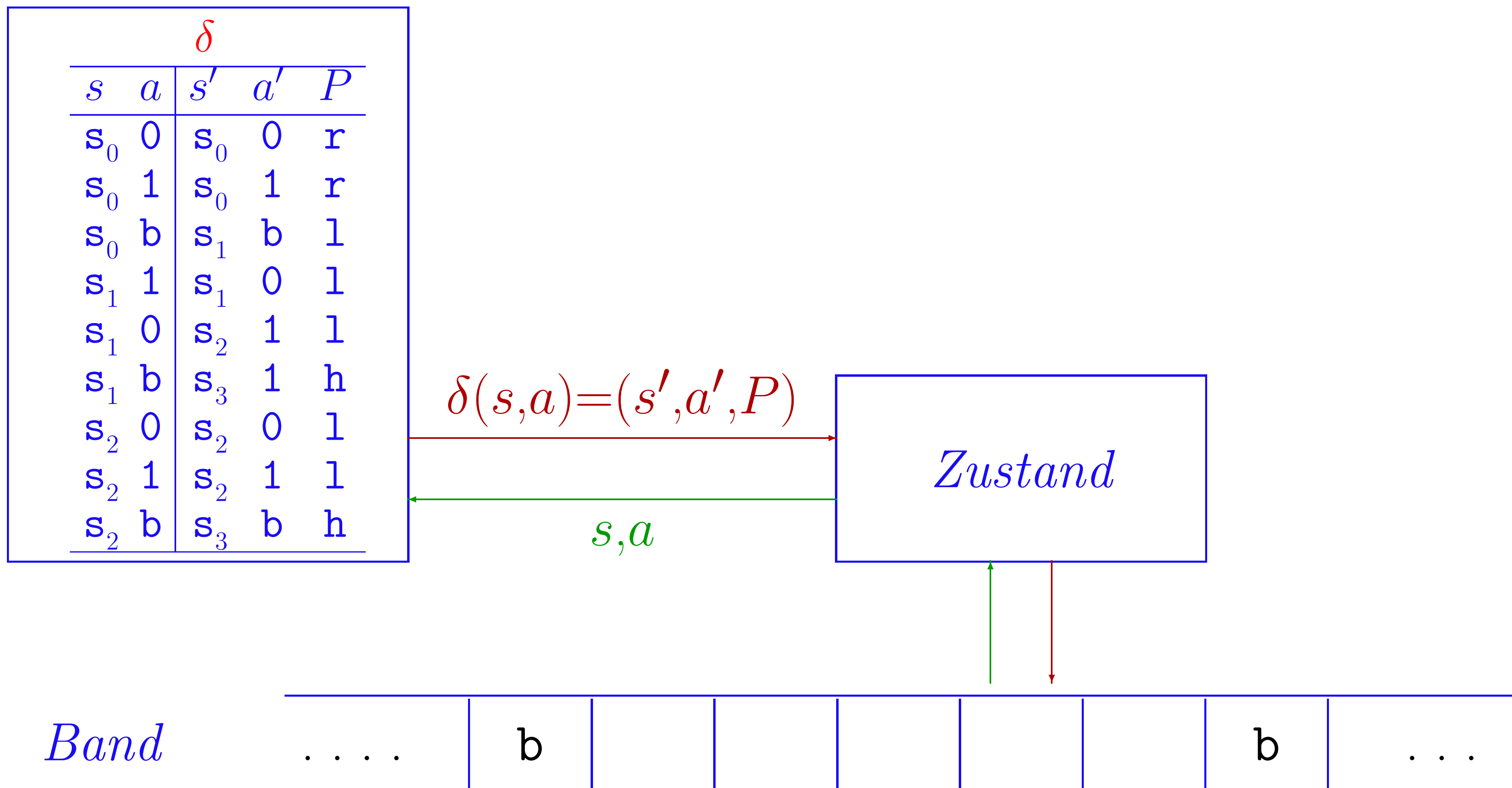
Arbeitsweise von Turingmaschinen



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



Verarbeitung von Turing-Programmen – präzisiert

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

Konfiguration von Turing-Programmen

- **Eine Konfiguration ist ein Tripel $\kappa = (s,f,i)$ mit**
  - $s \in S$ aktueller Zustand
  - $f{:}\mathbb{Z}{\to}\Gamma$ Bandinhaltsfunktion

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

    ($f(j) = b$ 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_τ: Menge aller Konfigurationen von τ**

Arbeitsweise von Turingmaschinen

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

- **Nachfolgekonfiguration** $\hat{\boldsymbol{\delta}}: K_\tau \rightarrow K_\tau$
  - 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** $\boldsymbol{\omega}: K_\tau \rightarrow \Gamma^*$
  - Für eine Konfiguration $\kappa = (s,f,i)$ ist
    $\omega(\kappa) = \begin{cases} \epsilon & \text{falls } f(j) = b \text{ für alle } j, \\ f(k)f(k+1)...f(k+n) & \text{sonst} \end{cases}$
    wobei $k = \max\{i \mid \forall j < i \;\; f(j) = b\}$ und $n = \min\{i \mid \forall j > k+i \;\; f(j) = b\}$

Semantik von Turing-Programmen

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

- **Mathematische Semantik von $\tau = (S, X, \Gamma, \delta, s_0, b)$**
  - Die von τ = **berechnete Funktion h_τ**$:X^* \to \Gamma^*$ ist definiert durch

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

  - **Definitionsbereich** von τ: $\{w \in X^* \mid h_\tau(w) \neq \bot\}$ (Haltebereich, domain)
  - **Wertebereich** von τ: $\{v \in \Gamma^* \mid \exists w \in X^* \;\; h_\tau(w) = v\}$ (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

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

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

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

Beispiele für Turing-Maschinen II

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

s	a	s'	a'	P
$\mathtt{s_0}$	1	$\mathtt{s_0}$	b	r
$\mathtt{s_0}$	b	$\mathtt{s_0}$	b	h

  Löscht ein Wort $w \in \mathtt{1}^*$:

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

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

s	a	s'	a'	P
$\mathtt{s_0}$	1	$\mathtt{s_1}$	1	r
$\mathtt{s_0}$	b	$\mathtt{s_1}$	1	h
$\mathtt{s_1}$	1	$\mathtt{s_0}$	1	r
$\mathtt{s_1}$	b	$\mathtt{s_1}$	b	r

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

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

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

Beispiele für Turing-Maschinen III

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

s	a	s'	a'	P
$\mathtt{s_0}$	1	$\mathtt{s_1}$	b	r
$\mathtt{s_0}$	c	$\mathtt{s_0}$	c	h
$\mathtt{s_0}$	b	$\mathtt{s_0}$	b	h
$\mathtt{s_1}$	1	$\mathtt{s_1}$	1	r
$\mathtt{s_1}$	c	$\mathtt{s_1}$	c	r
$\mathtt{s_1}$	b	$\mathtt{s_2}$	c	r
$\mathtt{s_2}$	1	$\mathtt{s_2}$	1	h
$\mathtt{s_2}$	c	$\mathtt{s_2}$	c	h
$\mathtt{s_2}$	b	$\mathtt{s_3}$	c	l
$\mathtt{s_3}$	1	$\mathtt{s_3}$	1	l
$\mathtt{s_3}$	c	$\mathtt{s_3}$	c	l
$\mathtt{s_3}$	b	$\mathtt{s_0}$	b	r

Verdoppelt Anzahl der Einsen

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

Kombinierbar mit isomorpher Variante von τ_3: $h_{\tau_3'} \circ h_{\tau_4}(\mathtt{1}^n) = \mathtt{c}^{2n+1}$

Turing-Berechenbarkeit

- f**:** $X^* \to Y^*$ **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**
  - $\mathcal{T}_{X,Y} = \{f{:}X^* \to Y^* \mid f \text{ ist Turing-berechenbar}\}$
  - $\mathcal{T} \quad = \bigcup\{\mathcal{T}_{X,Y} \mid X, Y \text{ endliches Alphabet}\}$

Übertragung des Berechenbarkeitbegriffs

- **Berechenbarkeit von Mengen** $M \subseteq X^*$
  - Semi-Entscheidbarkeit: Berechenbarkeit von $\psi_M{:}X^* \to \{0,1\}^*$,
  - Entscheidbarkeit: Berechenbarkeit von $\chi_M{:}X^* \to \{0,1\}^*$,

  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** $f{:}\mathbb{N} \to \mathbb{N}$

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

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

- **Berechenbarkeit auf Tupeln** $f{:}X^* \times X^* \to Y^*$

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

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?
  - τ_s muß Ziffern von rechts beginnend umwandeln, ggf. mit Übertrag

s	a	s'	a'	P	
s_0	0	s_0	0	r	*rechtes Ende suchen*
s_0	1	s_0	1	r	*rechtes Ende suchen*
s_0	b	s_1	b	l	*rechtes Ende gefunden*
s_1	1	s_1	0	l	*Addieren mit Übertrag*
s_1	0	s_2	1	h	*Addieren ohne Übertrag*
s_1	b	s_2	1	h	*Übertrag am linken Ende*

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 τ 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	*Erste* 1
s_0	b	s_6	b	h	*keine erste* 1
s_1	1	s_2	b	r	*Zweite* 1
s_1	b	s_6	b	h	*keine zweite* 1
s_2	1	s_2	1	r	*Nach rechts zum Eingabeende*
s_2	b	s_3	b	r	*Ende der Eingabe*
s_3	1	s_3	1	r	*Nach rechts zum Ausgabeende*
s_3	b	s_4	1	l	*Ende der Ausgabe, 1 schreiben*
s_4	1	s_4	1	l	*Nach links über Ausgabe*
s_4	b	s_5	b	l	
s_5	1	s_5	1	l	*Nach links über Eingabe*
s_5	b	s_0	b	r	

- Bei binärer Codierung muß τ die letzte Ziffer löschen

Varianten von Turingmaschinen

- **Vereinfachung für theoretische Analysen**
  - Binäres Bandalphabet $\Gamma = \{\mathtt{1}, \mathtt{b}\}$
  - Halbseitig unendliches Band
  - Restriktivere Ausgabekonvention
  - Endzustand statt Halteinstruktion
- **Erweiterung des Modells für Programmierzwecke**
  - Unvollständige Tabellen für δ
  - Mehrspurmaschinen
  - Mehrkopfmaschinen
  - Mehrbandmaschinen
  - Mehrdimensionale Maschinen
  - Unterprogramme

Alle Varianten führen zum gleichen Berechenbarkeitsbegriff

Kein Verlust der Ausdruckskraft

Simulation normaler Turingmaschinen möglich

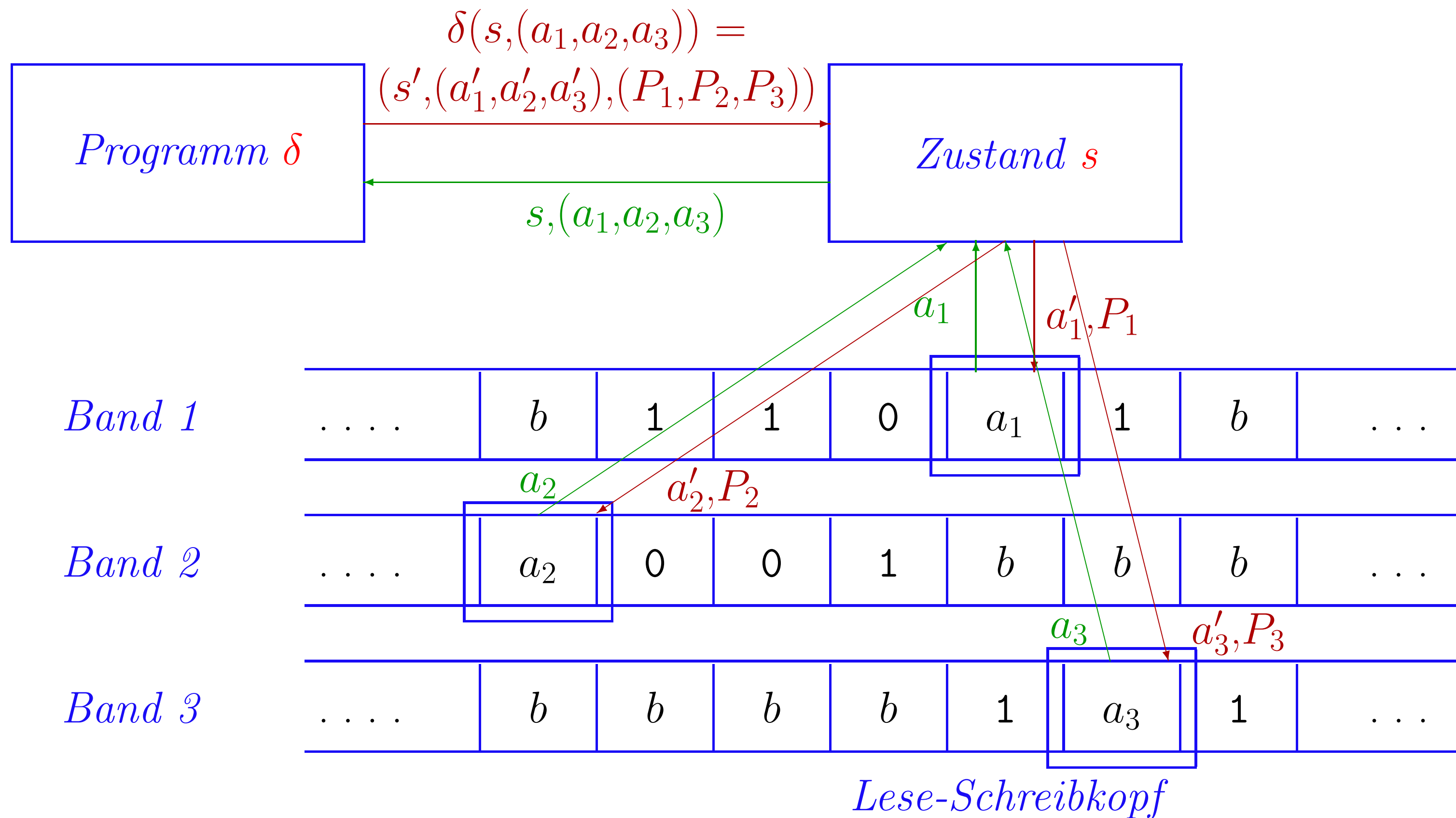
- **Halbseitig unendliches Band**
  - Simulation eines beidseitig unendlichen Bands durch Tupelalphabet (a_l, a_r)
    a_l repräsentiert die linke, a_r die rechte Bandhälfte
- **Binäres Bandalphabet** $\Gamma = \{\texttt{1}, \texttt{b}\}$
  - Binärcodierung beliebiger Alphabete als Strings über $\{\texttt{1b},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

Programmiertechniken für Turingmaschinen

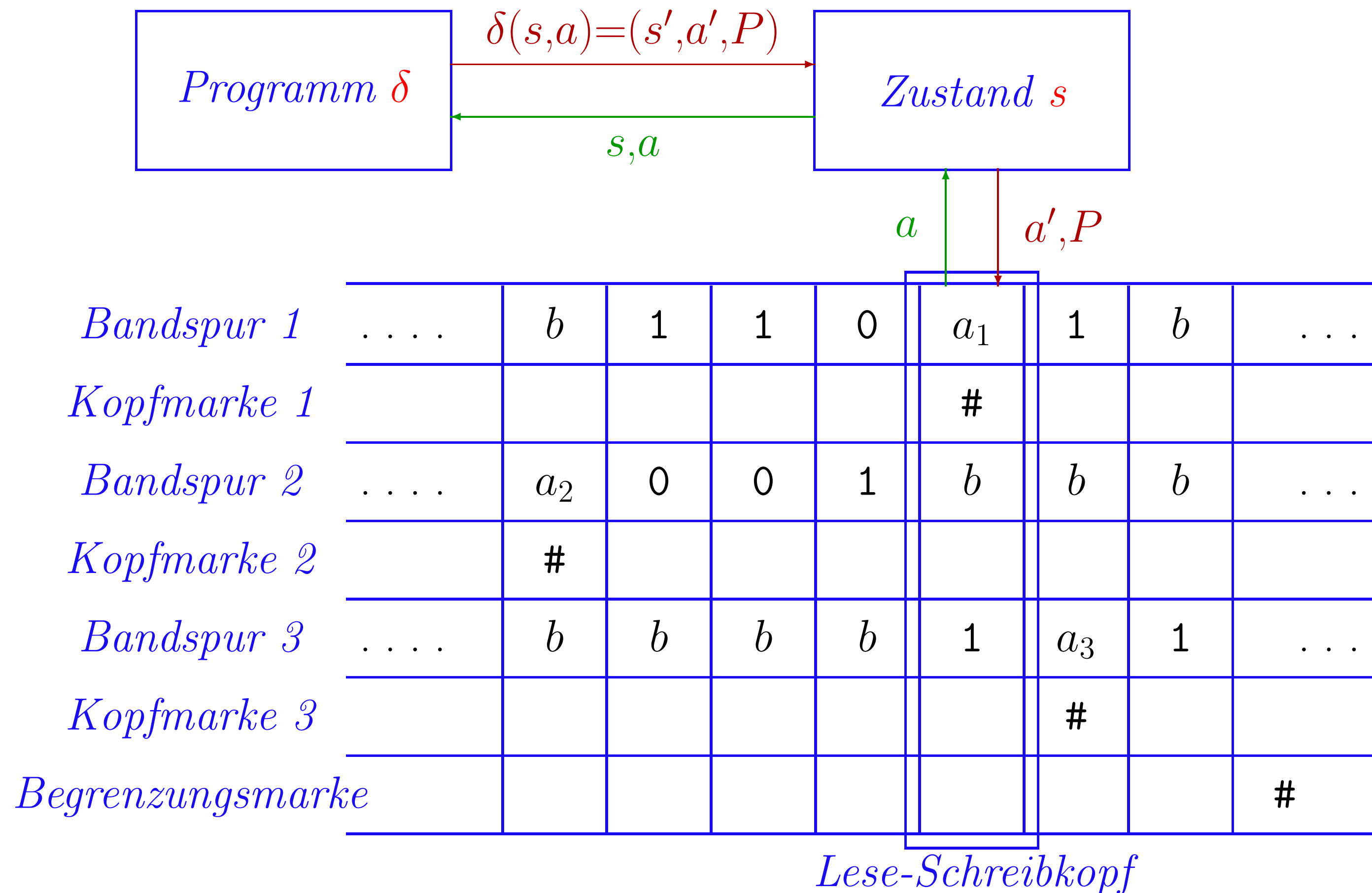
Keine Erweiterung der Ausdruckskraft
Simulation durch normale Turingmaschinen möglich

- **Unvollständige Tabellen für δ**
  - Ergänze nichtgenannte Einträge als $\delta(s,a) = (s,a,h)$
- **Mehrspurmaschinen**
  - Simulation von k Spuren durch Tupelalphabet $(a_1, ..., a_k)$
    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 Begrenzungsmarke, laufe zurück bis zu Kopfmarken
  - Schreiben und Kopfbewegung analog
  - Codiere Symbole und Kopfinstruktionen im Zustand
  - Simulation benötigt quadratische Zeit

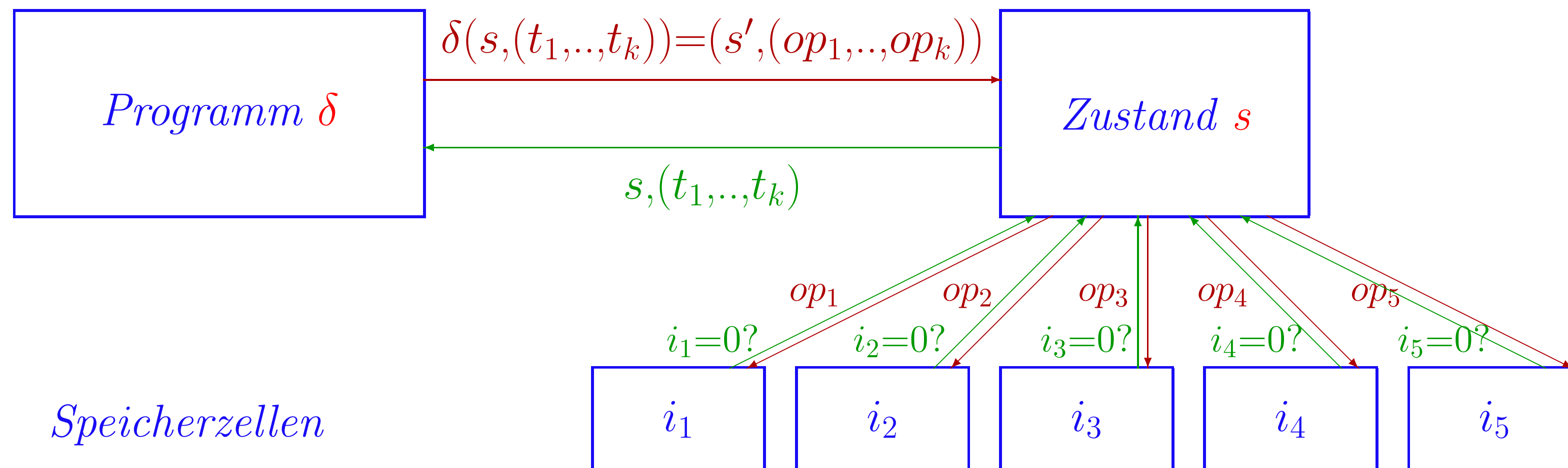
Theoretische Informatik

Einheit 2.2

Registermaschinen

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

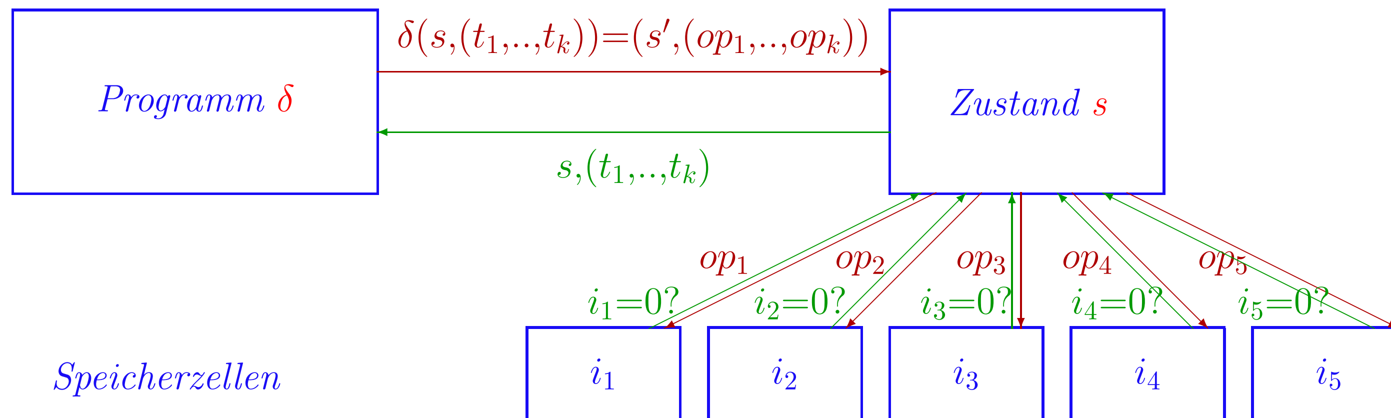
Registermaschinen



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

Achtung! Modelle in Literatur oft flexibler

Registermaschinen – mathematisch



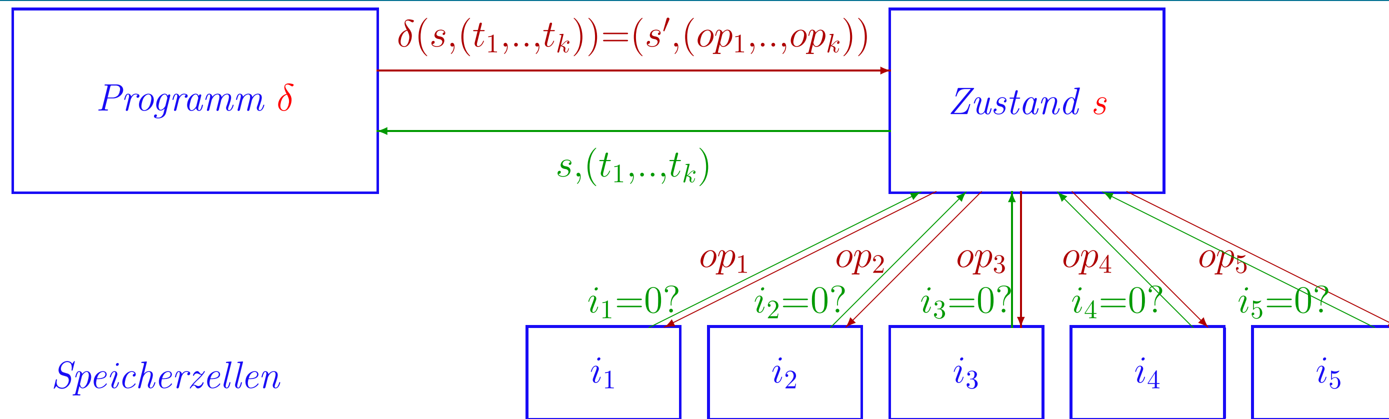
Eine **Registermaschine** ist ein 5-Tupel $\boldsymbol{\rho} = (S,\, k,\, \delta,\, s_0,\, F)$

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

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



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

Semantik von Registermaschinen

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

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

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

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

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

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

Beispiele für Registermaschinen

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

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

  Zähle Eingabewert n auf Null herunter und addiere Eins

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

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

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

- **Analyse**

$$n \xrightarrow{\alpha} (s_0, n, 0)$$
$$\xrightarrow{\delta} (s_1, n-1, 1) \xrightarrow{\delta} (s_0, n-1, 2) \xrightarrow{\delta} \ldots \xrightarrow{\delta} (s_2, 0, 2n)$$
$$\xrightarrow{\delta} (s_3, 1, 2n-1) \xrightarrow{\delta} (s_2, 2, 2n-1) \xrightarrow{\delta} \ldots \xrightarrow{\delta} (s_4, 4n, 0)$$
$$\xrightarrow{\omega} 4n$$

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

Register-Berechenbarkeit

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

Beispiele Register-Berechenbarer Funktionen

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

$\rho_3 = (\{s_0,..,s_c\}, 1, \delta_3, s_0, \{s_c\})$ mit $\delta_3 =$

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

- **Addition** $f_4(n,m) = n+m$
  - ρ_4 muß Register r_1 auf Null herunterzählen und dabei r_2 hochzählen

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

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** $f_5(n,m) = n*m$
  - ρ_5 muß r_1 auf Null herunterzählen und dabei jeweils r_2+n berechnen
  - n muß zuvor in Hilfsregister kopiert werden

Erweiterung des Modells für Programmierzwecke

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

- **RM-Programmiersprache**

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

  - Jeder Befehl kann durch RM-Unterprogramme simuliert werden

- **Befehlsmacros**
  - Abkürzungen für Programmfragmente in RM-Programmiersprache

Befehlsmacros (Auswahl)

- $\texttt{r}_j$ `:=` $\texttt{r}_i$
  - Verschiebe r_i in Hilfsregister r' und kopiere r' simultan nach r_j und r_i

    $\texttt{r}_j\texttt{:=0; r':=0;}$ $\texttt{while r}_i\texttt{>0 do r}_i\texttt{:=r}_i\dot{-}\texttt{1; r':=r'+1 od;}$
    $\texttt{while r'>0 do r':=r'}\dot{-}\texttt{1; r}_j\texttt{:=r}_j\texttt{+1; r}_i\texttt{:=r}_i\texttt{+1 od}$

- $\texttt{r}_j$ `:=` $\texttt{r}_j\texttt{+r}_i$
  - Kopiere r_i in nach r' und zähle simultan r' herunter und r_j hoch

    $\texttt{r' := r}_i\texttt{;}$ $\texttt{while r'>0 do r':=r'}\dot{-}\texttt{1; r}_j\texttt{:=r}_j\texttt{+1 od}$

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

  $\texttt{r}_j \texttt{ := r}_j\texttt{*r}_i$ $\qquad$ $\texttt{r}_j \texttt{ := r}_j^{\texttt{r}_i}$

  $\texttt{r}_j \texttt{ := r}_j\div\texttt{r}_i$ $\qquad$ $\texttt{r}_j \texttt{ := r}_j \texttt{ mod r}_i$

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

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

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

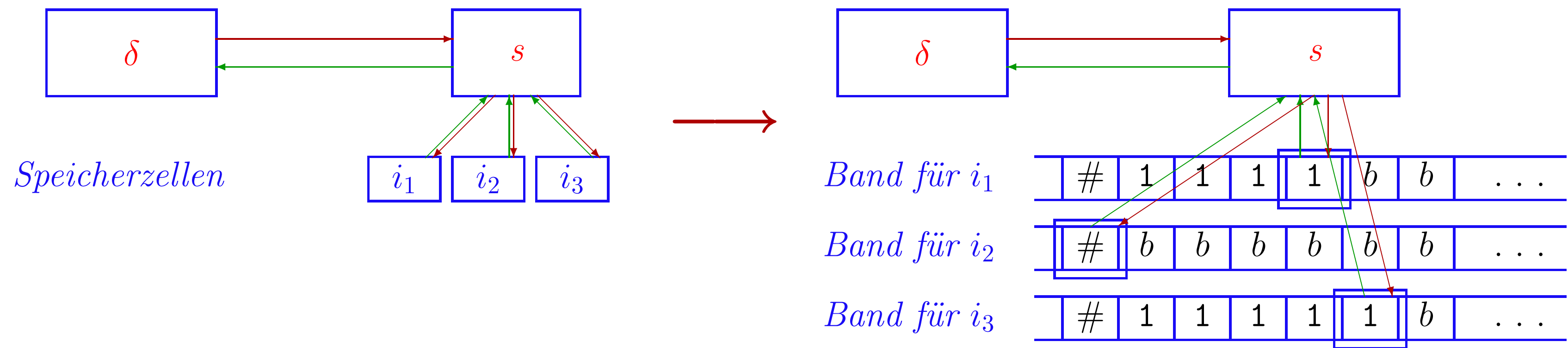
  $\texttt{if r}_j\leq\texttt{r}_i\texttt{ then}$ op `fi`

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

Beweis durch gegenseitige Simulation

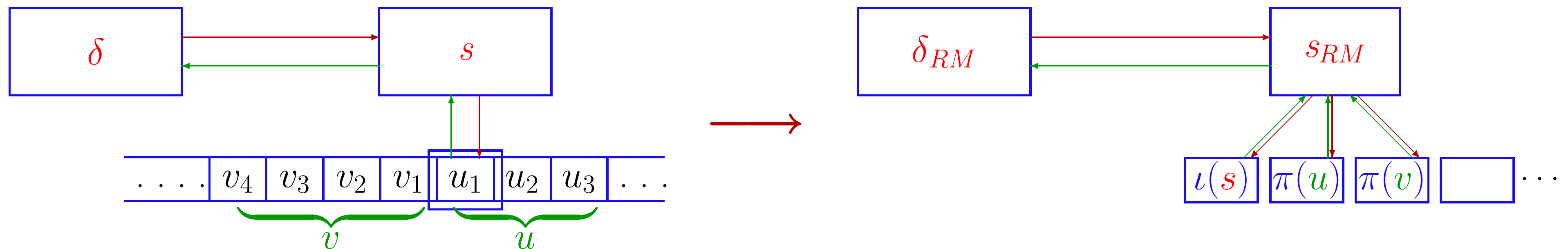
- $\mathcal{RM} \subseteq \mathcal{T}_{\{1\},\{1\}}$
  - Simuliere jedes Register durch separates (einseitiges) Turingband
  - Simuliere Registeroperationen durch Hinzufügen bzw. Löschen von Einsen
  - k-Band-Maschine durch Einbandmaschine simulierbar
- $\mathcal{RM} \supseteq \mathcal{T}_{\{1\},\{1\}}$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



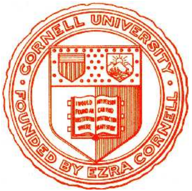
- **Band für Register r wird kellerartig verarbeitet**
  - Kopf am rechten Ende der unären Codierung des Registerinhalts
- **Überführungsfunktion direkt simulierbar**
  - Registerinhaltstest 0: Lesen des Bandanfangsmarkers #
  - Registerinhaltstest 1: Lesen einer 1
  - Registerinhalt vergrößern: nach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

Simulation einer TM durch eine RM



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

Theoretische Informatik



Einheit 2.3

Andere Berechenbarkeitsmodelle

1. μ-rekursive Funktionen
2. λ-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
- **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 μ-rekursiver Funktionen

1. **Nachfolgerfunktion** $\boldsymbol{s}:\mathbb{N}\rightarrow\mathbb{N}$ mit $s(x) = x+1$ für alle $x\in\mathbb{N}$
2. **Projektionsfunktionen** $\boldsymbol{pr_k^n}:\mathbb{N}^n\rightarrow\mathbb{N}$ mit $pr_k^n(x_1,...,x_n) = x_k$ $(1{\leq}k{\leq}n)$
3. **Konstantenfunktion** $\boldsymbol{c_k^n}:\mathbb{N}^n\rightarrow\mathbb{N}$ mit $c_k^n(x_1,...,x_n) = k$ $(0{\leq}n)$
4. **Komposition** $\boldsymbol{f\circ(g_1...g_n)}:\mathbb{N}^k\rightarrow\mathbb{N}$

   $h = f\circ(g_1...g_n)$, wenn $h(\vec{x}) = f(\,g_1(\vec{x}),...,g_n(\vec{x})\,)$

   h entsteht aus $f:\mathbb{N}^n\rightarrow\mathbb{N}$ und $g_1...g_n:\mathbb{N}^k\rightarrow\mathbb{N}$ durch simultane Einsetzung
5. **Primitive Rekursion** $\boldsymbol{Pr[f,g]}:\mathbb{N}^k\rightarrow\mathbb{N}$

   $h = Pr[f,g]$, 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. **μ-Operator** $\boldsymbol{\mu f}:\mathbb{N}^k\rightarrow\mathbb{N}$

   $$h = \mu f,\ \text{wenn}\ h(\vec{x}) = \begin{cases} min\{y \mid f(\vec{x},y)=0\} & \text{falls dies existiert und} \\ & \text{alle } f(\vec{x},i), i{<}y \text{ definiert} \\ \bot & \text{sonst} \end{cases}$$

   h entsteht aus $f:\mathbb{N}^{k+1}\rightarrow\mathbb{N}$ durch Minimierung

Operationen entsprechen Programmstrukturen

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

```
y_1 := g_1(x_1,..,x_m);
  ⋮
y_n := g_n(x_1,..,x_m);
h  := f(y_1,..,y_n)
```
$(\mathtt{h} = h(x_1, .., x_m))$

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

```
h := f(x_1,..,x_n);
for i:=1 to y do h := g(x_1,..,x_n,i-1,h) od
```
$(\mathtt{h} = h(x_1, .., x_n, y))$

  – Primitive Rekursion arbeitet in umgekehrter Reihenfolge

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

```
y := 0;
while f(x_1..x_n,y)≠0 do y:=y+1 od;
h := y
```
$(\mathtt{h} = h(x_1, .., x_n))$

  – Ergebnis ist Anzahl der Schleifendurchläufe bis zum Erfolg

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

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

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

Beispiel einer primitiv-rekursiven Funktion

- $\boldsymbol{f_1 = Pr[pr_1^1, s \circ pr_3^3]}$ **Was macht f_1?**

- **Stelligeitsanalyse:**

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

- **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

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

Analyse μ-rekursiver Funktionen

- $\boldsymbol{f_2 = \mu c_1^2}$

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

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

$$= \perp$$

- $\boldsymbol{f_3 = \mu f_1}$

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

- $\boldsymbol{f_4 = \mu h}$ 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" μ-rekursiver Funktionen

- **Vorgängerfunktion** $\boldsymbol{p}$ **:** $\mathbb{N}\to\mathbb{N}$ $\quad p(n) = n\dot{-}1$

- **Rekursives Verhalten:**
  - $p(0) = 0\dot{-}1 = 0$
  - $p(y+1) = (y+1)\dot{-}1 = y$

- **Beschreibung durch Primitive Rekursion:**
  - Benötigt: $f{:}\mathbb{N}^0\to\mathbb{N}$ mit $f() = 0$ $\quad\mapsto f = c_0^0$

    und $g{:}\mathbb{N}^2\to\mathbb{N}$ mit $g(y, p(y)) = p(y+1) = y$ $\quad\mapsto g = pr_1^2$

$$\mapsto \quad \boldsymbol{p = Pr[c_0^0, pr_1^2]}$$

Beispiele primitiv-rekursiver Funktionen

- **Subtraktion** $\boldsymbol{sub}: \mathbb{N}^2 \to \mathbb{N}$ $\qquad sub(n,m) = n \dot{-} m$
  - $sub(x,0) = x = pr_1^1(x)$
  - $sub(x,y+1) = x \dot{-} (y+1) = (x \dot{-} y) \dot{-} 1 = p(x \dot{-} y) = (p \circ pr_3^3)(x,y,sub(x,y))$

  $\mapsto \; sub = Pr[pr_1^1, p \circ pr_3^3]$

- **Multiplikation** $\boldsymbol{mul}: \mathbb{N}^2 \to \mathbb{N}$ $\qquad mul(n,m) = n * m$
  - $mul(x,0) = 0 = c_0^1(x)$
  - $mul(x,y+1) = mul(x,y) + x = (add \circ (pr_1^3, pr_3^3))(x,y,mul(x,y))$

  $\mapsto \; mul = Pr[c_0^1, (add \circ (pr_1^3, pr_3^3))]$

- **Exponentiierung** $\boldsymbol{exp}: \mathbb{N}^2 \to \mathbb{N}$ $\qquad exp(n,m) = n^m$

  $exp = Pr[c_1^1, (mul \circ (pr_1^3, pr_3^3))]$

- **Fakultät** $\boldsymbol{fak}: \mathbb{N} \to \mathbb{N}$ $\qquad fak(n) = n! = 1*2*...*n$

  $fak = Pr[c_1^0, (mul \circ (s \circ pr_1^2, pr_2^2))]$

- **Signum-Funktion** $\boldsymbol{sign}: \mathbb{N} \to \mathbb{N}$ $\qquad sign(n) = \begin{cases} 0 & \text{falls } n = 0 \\ 1 & \text{sonst} \end{cases}$

  $sign = Pr[c_0^0, c_1^2]$

Programmiertechniken für rekursive Funktionen

- **Definition durch Fallunterscheidung**

$$h(\vec{x}) = \begin{cases} f(\vec{x}) & \text{falls } test(\vec{x}) = 0 \\ g(\vec{x}) & \text{sonst} \end{cases}$$

(f, g und $test$:$\mathbb{N}^k \to \mathbb{N}$ primitiv-rekursiv)

Wende Signum-Funktion auf Testergebnis an und multipliziere auf

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

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

- **Generelle Summe** $\boldsymbol{\Sigma_{i=0}^{r} f(\vec{x}, i)}$

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

- **Beschränkte Minimierung**

$$h(\vec{x}, t) = \begin{cases} min\{y \leq t \mid f(\vec{x}, y) = 0\} & \text{falls dies existiert} \\ t+1 & \text{sonst} \end{cases}$$

(f:$\mathbb{N}^{k+1} \to \mathbb{N} \in \mathcal{T}_{prim}$)

Programmierbar mit Fallunterscheidung & primitiver Rekursion (aufwendig)

Weitere primitiv-rekursive Funktionen

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

- $\langle\rangle{:}\mathbb{N}^2{\to}\mathbb{N}$ **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** $\mathbb{N}^k{\to}\mathbb{N}$ **und auf** $\mathbb{N}^*{\to}\mathbb{N}$ **fortgesetzt werden**
  - $\langle x,y,z\rangle^3 = \langle x,\langle y,z\rangle\rangle, \ldots, \quad \langle x_1...x_k\rangle^* = \langle k,\langle x_1,..,x_k\rangle^k\rangle$
  - Alle Funktionen sind bijektiv und primitiv-rekursiv
  - Alle Umkehrfunktionen π_i^k und π_i^* sind primitiv-rekursiv
- **Jede rekursive Funktion kann einstellig simuliert werden**
  - Für $f{:}\mathbb{N}^2{\to}\mathbb{N}$ und $g := f\circ(\pi_1^2,\pi_2^2)$ gilt $g{:}\mathbb{N}{\to}\mathbb{N}$ und $f(x,y) = g(\langle x,y\rangle)$

Ackermann-Funktionen (1928)

- **Definiere Funktionen A_n iterativ:**

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

- **Jede der Funktionen A_n ist primitiv-rekursiv**

- **Wachstumsverhalten**

$$A_1(x) = 2x \ (x \geq 1) \qquad A_4(0) = 1 \qquad A_4(3) = 2^{2^{2^2}} = 65536$$

$$A_2(x) = 2^x \qquad A_4(1) = 2 \qquad A_4(4) = \underbrace{2^{(2^{(2^{\cdots^2})})}}_{65536-mal}$$

$$A_3(x) = \underbrace{2^{(2^{(2^{\cdots^2})})}}_{x-mal} \qquad A_4(2) = 2^2 = 4 \qquad A_4(5) = \underbrace{2^{(2^{(2^{\cdots^2})})}}_{A_4(4)-mal}$$

- **Definiere $A(x) := A_x(x)$** (**Große Ackermann-Funktion**)
  - $A \notin \mathcal{T}_{prim}$: A wächst schneller als jede primitiv-rekursive Funktion
  - $A \in \mathcal{R}_\mu$: programmiere Abarbeitung eines Berechnungsstacks für A

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 μ-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 μ-rekursiv

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` $\hat{=}$ `λx. 2*x+3` — λ-Notation
  - Auswertung der Funktion: `(λx. 2*x+3)(4)` $\xrightarrow{\beta}$ 11 — Applikation + Reduktion

Name der Funktion ist irrelevant

λ-Kalkül – Syntax und Semantik

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

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

- **Auswertung von λ-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

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

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

Darstellung Boolescher Operatoren im λ-Kalkül

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

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

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

Konditional ist invers zu T und F

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

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

Bildung und Analyse von Paaren

$$\langle s , t \rangle \equiv \lambda \mathtt{p}.\ \mathtt{p}\, s\, t$$

$$pair.\mathbf{1} \equiv pair\ (\lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{x})$$

$$pair.\mathbf{2} \equiv pair\ (\lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{y})$$

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

Analyseoperator ist invers zur Paarbildung

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

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

$$\equiv (\lambda \mathtt{p}.\ \mathtt{p}\, u\, v)(\lambda x.\lambda y.t)$$

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

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

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

λ-berechenbare Funktionen auf ℕ

- **Darstellung von Zahlen durch iterierte Terme**
  - Semantisch: wiederholte Anwendung von Funktionen
  - Repräsentiere die Zahl n durch den Term $\underbrace{\lambda\texttt{f.}\lambda\texttt{x.f (f..(f x)..)}}_{n\text{-}mal}$
  - Notation: $\overline{\boldsymbol{n}} \equiv \lambda\texttt{f.}\lambda\texttt{x.f}^{n}\,\texttt{x}$
  - Bezeichnung: **Church Numerals**
- $\boldsymbol{f}$**:**$\mathbb{N}^{\boldsymbol{n}}\longrightarrow\mathbb{N}$ **λ-berechenbar:**
  - Es gibt einen λ-Term t mit $f(x_1, ..., x_n) = m \;\Leftrightarrow\; t\ \overline{x_1}\ \ldots \overline{x_n} = \overline{m}$
- **Operationen müssen Termvielfachheit verändern**
  - z.B. **add** $\overline{\boldsymbol{m}}$ $\overline{\boldsymbol{n}}$ muß als Wert immer den Term $\overline{\boldsymbol{m+n}}$ ergeben

Programmierung im λ-Kalkül

- **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 $\overline{n+1}$

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

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

- **Vorgängerfunktion:**

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

Auswertung der Additionsfunktion

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

$$
\begin{array}{lcl}
\textbf{add}\ \overline{m}\ \overline{n} & \equiv & (\lambda\texttt{m}.\lambda\texttt{n}.\lambda\texttt{f}.\lambda\texttt{x}.\ \texttt{m f (n f x)})\ \overline{m}\ \overline{n} \\
 & \longrightarrow & (\lambda\texttt{n}.\lambda\texttt{f}.\lambda\texttt{x}.\ \overline{m}\ \texttt{f (n f x)})\ \overline{n} \\
 & \longrightarrow & \lambda\texttt{f}.\lambda\texttt{x}.\ \overline{m}\ \texttt{f}\ (\overline{n}\ \texttt{f x}) \\
 & \equiv & \lambda\texttt{f}.\lambda\texttt{x}.\ (\lambda\texttt{f}.\lambda\texttt{x}.\ \texttt{f}^{m}\,\texttt{x})\ \texttt{f}\ (\overline{n}\ \texttt{f x}) \\
 & \longrightarrow & \lambda\texttt{f}.\lambda\texttt{x}.\ (\lambda\texttt{x}.\ \texttt{f}^{m}\,\texttt{x})\ (\overline{n}\ \texttt{f x}) \\
 & \longrightarrow & \lambda\texttt{f}.\lambda\texttt{x}.\ \texttt{f}^{m}\,(\overline{n}\ \texttt{f x}) \\
 & \equiv & \lambda\texttt{f}.\lambda\texttt{x}.\ \texttt{f}^{m}\,((\lambda\texttt{f}.\lambda\texttt{x}.\ \texttt{f}^{n}\,\texttt{x})\ \texttt{f x}) \\
 & \longrightarrow & \lambda\texttt{f}.\lambda\texttt{x}.\ \texttt{f}^{m}\,((\lambda\texttt{x}.\ \texttt{f}^{n}\,\texttt{x})\ \texttt{x}) \\
 & \longrightarrow & \lambda\texttt{f}.\lambda\texttt{x}.\ \texttt{f}^{m}\,(\texttt{f}^{n}\,\texttt{x}) \\
 & \longrightarrow & \lambda\texttt{f}.\lambda\texttt{x}.\ \texttt{f}^{m+n}\,\texttt{x} \qquad\qquad \equiv \quad \overline{m+n}
\end{array}
$$

Rekursion im λ-Kalkül

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

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

$$
\begin{array}{lcl}
\mathbf{Y}\ t & \equiv & \lambda \mathtt{f}.\, (\lambda \mathtt{x}.\, \mathtt{f}\ (\mathtt{x}\ \mathtt{x}))\, (\lambda \mathtt{x}.\, \mathtt{f}\ (\mathtt{x}\ \mathtt{x}))\ t \\
 & \longrightarrow & (\lambda \mathtt{x}.\, t\ (\mathtt{x}\ \mathtt{x}))\ (\lambda \mathtt{x}.\, t\ (\mathtt{x}\ \mathtt{x})) \\
 & \longrightarrow & t\ (\ (\lambda \mathtt{x}.\, t\ (\mathtt{x}\ \mathtt{x}))\ (\lambda \mathtt{x}.\, t\ (\mathtt{x}\ \mathtt{x}))\ ) \\
t\ (\mathbf{Y}\ t) & \equiv & t\ (\lambda \mathtt{f}.\, (\lambda \mathtt{x}.\, \mathtt{f}\ (\mathtt{x}\ \mathtt{x}))\, (\lambda \mathtt{x}.\, \mathtt{f}\ (\mathtt{x}\ \mathtt{x}))\ t) \\
 & \longrightarrow & t\ (\ (\lambda \mathtt{x}.\, t\ (\mathtt{x}\ \mathtt{x}))\ (\lambda \mathtt{x}.\, t\ (\mathtt{x}\ \mathtt{x}))\ )
\end{array}
$$

- **Rekursion darstellbar als**

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

Ausdruckskraft des λ-Kalküls

Alle μ-rekursiven Funktione sind λ-berechenbar

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

Weitere Modelle

- **Nichtdeterministische Turingmaschine**
  - Arbeitsweise wie gewöhnliche Turingmaschine
  - Zustandsüberführungsfunktion 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
- **Church'sche These:**
  Die Klasse der Turing-berechenbaren Funktionen stimmt mit der Klasse der intuitiv berechenbaren Funktionen überein
  - Unbeweisbare, aber wahrscheinlich richtige Behauptung
  - Arbeitshypothese für theoretische Argumente
    - man darf in Beweisen "intuitive" Programme angeben