

# Theoretische Informatik II

## Einheit 4.3

### Rekursive Funktionen



1. Primitiv- und  $\mu$ -rekursive Funktionen
2. Analyse und Programmierung
3. Äquivalenz zu Turingmaschinen

# BERECHENBARKEIT OHNE MASCHINENMODELL

Welche Arten von Funktionen sind berechenbar?

# BERECHENBARKEIT OHNE MASCHINENMODELL

## Welche Arten von Funktionen sind berechenbar?

- **Einfache Funktionen müssen berechenbar sein**
  - **Nachfolger**funktion: von einer Zahl zur nächsten weiterzählen (s)

## Welche Arten von Funktionen sind berechenbar?

- **Einfache Funktionen müssen berechenbar sein**

- **Nachfolger**funktion: von einer Zahl zur nächsten weiterzählen  $(s)$
- **Projektion**: aus einer Gruppe von Werten einen herauswählen  $(pr_k^n)$

## Welche Arten von Funktionen sind berechenbar?

- **Einfache Funktionen müssen berechenbar sein**

- **Nachfolger**funktion: von einer Zahl zur nächsten weiterzählen  $(s)$
- **Projektion**: aus einer Gruppe von Werten einen herauswählen  $(pr_k^n)$
- **Konstante**: unabhängig von der Eingabe eine feste Zahl ausgeben  $(c_k^n)$

## Welche Arten von Funktionen sind berechenbar?

- **Einfache Funktionen müssen berechenbar sein**
  - **Nachfolger**funktion: von einer Zahl zur nächsten weiterzählen  $(s)$
  - **Projektion**: aus einer Gruppe von Werten einen herauswählen  $(pr_k^n)$
  - **Konstante**: unabhängig von der Eingabe eine feste Zahl ausgeben  $(c_k^n)$
- **Berechenbare Funktionen sind zusammensetzbar**
  - Einfache **Operationen** erzeugen neue berechenbare Funktionen

## Welche Arten von Funktionen sind berechenbar?

- **Einfache Funktionen müssen berechenbar sein**
  - **Nachfolger**funktion: von einer Zahl zur nächsten weiterzählen  $(s)$
  - **Projektion**: aus einer Gruppe von Werten einen auswählen  $(pr_k^n)$
  - **Konstante**: unabhängig von der Eingabe eine feste Zahl ausgeben  $(c_k^n)$
- **Berechenbare Funktionen sind zusammensetzbar**
  - Einfache **Operationen** erzeugen neue berechenbare Funktionen
  - **Komposition**: Hintereinanderausführen mehrerer Berechnungen

## Welche Arten von Funktionen sind berechenbar?

- **Einfache Funktionen müssen berechenbar sein**
  - **Nachfolger**funktion: von einer Zahl zur nächsten weiterzählen  $(s)$
  - **Projektion**: aus einer Gruppe von Werten einen herauswählen  $(pr_k^n)$
  - **Konstante**: unabhängig von der Eingabe eine feste Zahl ausgeben  $(c_k^n)$
- **Berechenbare Funktionen sind zusammensetzbar**
  - Einfache **Operationen** erzeugen neue berechenbare Funktionen
  - **Komposition**: Hintereinanderausführen mehrerer Berechnungen
  - **Rekursion**: Bei der Ausführung ruft eine Funktion sich selbst wieder auf

## Welche Arten von Funktionen sind berechenbar?

- **Einfache Funktionen müssen berechenbar sein**
  - **Nachfolger**funktion: von einer Zahl zur nächsten weiterzählen  $(s)$
  - **Projektion**: aus einer Gruppe von Werten einen herauswählen  $(pr_k^n)$
  - **Konstante**: unabhängig von der Eingabe eine feste Zahl ausgeben  $(c_k^n)$
- **Berechenbare Funktionen sind zusammensetzbar**
  - Einfache **Operationen** erzeugen neue berechenbare Funktionen
  - **Komposition**: Hintereinanderausführen mehrerer Berechnungen
  - **Rekursion**: Bei der Ausführung ruft eine Funktion sich selbst wieder auf
  - **Suche**: Bestimmung der ersten Nullstelle einer Funktion

## Welche Arten von Funktionen sind berechenbar?

- **Einfache Funktionen müssen berechenbar sein**
  - **Nachfolger**funktion: von einer Zahl zur nächsten weiterzählen  $(s)$
  - **Projektion**: aus einer Gruppe von Werten einen herauswählen  $(pr_k^n)$
  - **Konstante**: unabhängig von der Eingabe eine feste Zahl ausgeben  $(c_k^n)$
- **Berechenbare Funktionen sind zusammensetzbar**
  - Einfache **Operationen** erzeugen neue berechenbare Funktionen
  - **Komposition**: Hintereinanderausführen mehrerer Berechnungen
  - **Rekursion**: Bei der Ausführung ruft eine Funktion sich selbst wieder auf
  - **Suche**: Bestimmung der ersten Nullstelle einer Funktion
- **Beschreibung benötigt keine Funktionsargumente**
  - Es reicht, die **Grundfunktionen** und **Operationen** zu benennen
  - Programmiersprache ist abstrakt: wende Operationen auf Funktionen an

## Welche Arten von Funktionen sind berechenbar?

- **Einfache Funktionen müssen berechenbar sein**
  - **Nachfolger**funktion: von einer Zahl zur nächsten weiterzählen  $(s)$
  - **Projektion**: aus einer Gruppe von Werten einen herauswählen  $(pr_k^n)$
  - **Konstante**: unabhängig von der Eingabe eine feste Zahl ausgeben  $(c_k^n)$
- **Berechenbare Funktionen sind zusammensetzbar**
  - Einfache **Operationen** erzeugen neue berechenbare Funktionen
  - **Komposition**: Hintereinanderausführen mehrerer Berechnungen
  - **Rekursion**: Bei der Ausführung ruft eine Funktion sich selbst wieder auf
  - **Suche**: Bestimmung der ersten Nullstelle einer Funktion
- **Beschreibung benötigt keine Funktionsargumente**
  - Es reicht, die **Grundfunktionen** und **Operationen** zu benennen
  - Programmiersprache ist abstrakt: wende Operationen auf Funktionen an

Mathematischer Kalkül für informatiktypisches 'Baukastensystem'

# BEISPIELE FÜR BAUKASTENPROGRAMMIERUNG

- Addition von 2

# BEISPIELE FÜR BAUKASTENPROGRAMMIERUNG

- **Addition von 2**

- Ergebnis entsteht durch zweifache Addition von 1

$$x+2 = x+1+1$$

# BEISPIELE FÜR BAUKASTENPROGRAMMIERUNG

- **Addition von 2**

- Ergebnis entsteht durch zweifache Addition von 1  $x+2 = x+1+1$
- Abstrahiert: doppelte Anwendung der Nachfolgerfunktion  $x+2 = s(s(x))$

# BEISPIELE FÜR BAUKASTENPROGRAMMIERUNG

## ● Addition von 2

- Ergebnis entsteht durch zweifache Addition von 1  $x+2 = x+1+1$
- Abstrahiert: doppelte Anwendung der Nachfolgerfunktion  $x+2 = s(s(x))$
- Die Funktion “Addition von 2” ist also die **Komposition** von  $s$  und  $s$

# BEISPIELE FÜR BAUKASTENPROGRAMMIERUNG

- **Addition von 2**

- Ergebnis entsteht durch zweifache Addition von 1  $x+2 = x+1+1$
- Abstrahiert: doppelte Anwendung der Nachfolgerfunktion  $x+2 = s(s(x))$
- Die Funktion “Addition von 2” ist also die Komposition von  $s$  und  $s$

- **Addition zweier Zahlen  $x$  und  $y$**

# BEISPIELE FÜR BAUKASTENPROGRAMMIERUNG

## ● Addition von 2

- Ergebnis entsteht durch zweifache Addition von 1  $x+2 = x+1+1$
- Abstrahiert: doppelte Anwendung der Nachfolgerfunktion  $x+2 = s(s(x))$
- Die Funktion “Addition von 2” ist also die Komposition von  $s$  und  $s$

## ● Addition zweier Zahlen $x$ und $y$

- Ergebnis entsteht durch  $y$ -fache Addition von 1  $x+y = x+\underbrace{1 + \dots + 1}_{y\text{-mal}}$

# BEISPIELE FÜR BAUKASTENPROGRAMMIERUNG

## ● Addition von 2

- Ergebnis entsteht durch zweifache Addition von 1  $x+2 = x+1+1$
- Abstrahiert: doppelte Anwendung der Nachfolgerfunktion  $x+2 = s(s(x))$
- Die Funktion “Addition von 2” ist also die Komposition von  $s$  und  $s$

## ● Addition zweier Zahlen $x$ und $y$

- Ergebnis entsteht durch  $y$ -fache Addition von 1  $x+y = x+\underbrace{1 + \dots + 1}_{y\text{-mal}}$
- Iteration muß rekursiv beschrieben werden
  - Im Basisfall ( $y=0$ ) ist das Ergebnis  $x$   $x+0 = x$

# BEISPIELE FÜR BAUKASTENPROGRAMMIERUNG

## ● Addition von 2

- Ergebnis entsteht durch zweifache Addition von 1  $x+2 = x+1+1$
- Abstrahiert: doppelte Anwendung der Nachfolgerfunktion  $x+2 = s(s(x))$
- Die Funktion “Addition von 2” ist also die Komposition von  $s$  und  $s$

## ● Addition zweier Zahlen $x$ und $y$

- Ergebnis entsteht durch  $y$ -fache Addition von 1  $x+y = x+\underbrace{1 + \dots + 1}_{y\text{-mal}}$
- Iteration muß rekursiv beschrieben werden
  - Im Basisfall ( $y=0$ ) ist das Ergebnis  $x$   $x+0 = x$
  - Im Schrittfall verwenden wir die Addition von  $x$  und  $y$ , um die Addition von  $x$  und  $y+1$  zu bestimmen  $x+(y+1) = (x+y)+1$

# BEISPIELE FÜR BAUKASTENPROGRAMMIERUNG

## ● Addition von 2

- Ergebnis entsteht durch zweifache Addition von 1  $x+2 = x+1+1$
- Abstrahiert: doppelte Anwendung der Nachfolgerfunktion  $x+2 = s(s(x))$
- Die Funktion “Addition von 2” ist also die Komposition von  $s$  und  $s$

## ● Addition zweier Zahlen $x$ und $y$

- Ergebnis entsteht durch  $y$ -fache Addition von 1  $x+y = x+\underbrace{1 + \dots + 1}_{y\text{-mal}}$
- Iteration muß rekursiv beschrieben werden
  - Im Basisfall ( $y=0$ ) ist das Ergebnis  $x$   $x+0 = x$
  - Im Schrittfall verwenden wir die Addition von  $x$  und  $y$ ,  
um die Addition von  $x$  und  $y+1$  zu bestimmen  $x+(y+1) = (x+y)+1$
- Abstrahiert:
  - Basisfall ergibt sich durch Anwendung der Projektion  $pr_1^1$  auf  $x$
  - Schrittfall ergibt sich durch Anwendung von  $s$  auf das “alte” Ergebnis

# BEISPIELE FÜR BAUKASTENPROGRAMMIERUNG

## ● Addition von 2

- Ergebnis entsteht durch zweifache Addition von 1  $x+2 = x+1+1$
- Abstrahiert: doppelte Anwendung der Nachfolgerfunktion  $x+2 = s(s(x))$
- Die Funktion “Addition von 2” ist also die Komposition von  $s$  und  $s$

## ● Addition zweier Zahlen $x$ und $y$

- Ergebnis entsteht durch  $y$ -fache Addition von 1  $x+y = x+\underbrace{1 + \dots + 1}_{y\text{-mal}}$
- Iteration muß rekursiv beschrieben werden
  - Im Basisfall ( $y=0$ ) ist das Ergebnis  $x$   $x+0 = x$
  - Im Schrittfall verwenden wir die Addition von  $x$  und  $y$ , um die Addition von  $x$  und  $y+1$  zu bestimmen  $x+(y+1) = (x+y)+1$
- Abstrahiert:
  - Basisfall ergibt sich durch Anwendung der Projektion  $pr_1^1$  auf  $x$
  - Schrittfall ergibt sich durch Anwendung von  $s$  auf das “alte” Ergebnis

## ● Subtraktion zweier Zahlen $x$ und $y$

# BEISPIELE FÜR BAUKASTENPROGRAMMIERUNG

## ● Addition von 2

- Ergebnis entsteht durch zweifache Addition von 1  $x+2 = x+1+1$
- Abstrahiert: doppelte Anwendung der Nachfolgerfunktion  $x+2 = s(s(x))$
- Die Funktion “Addition von 2” ist also die Komposition von  $s$  und  $s$

## ● Addition zweier Zahlen $x$ und $y$

- Ergebnis entsteht durch  $y$ -fache Addition von 1  $x+y = x+\underbrace{1 + \dots + 1}_{y\text{-mal}}$
- Iteration muß rekursiv beschrieben werden
  - Im Basisfall ( $y=0$ ) ist das Ergebnis  $x$   $x+0 = x$
  - Im Schrittfall verwenden wir die Addition von  $x$  und  $y$ , um die Addition von  $x$  und  $y+1$  zu bestimmen  $x+(y+1) = (x+y)+1$
- Abstrahiert:
  - Basisfall ergibt sich durch Anwendung der Projektion  $pr_1^1$  auf  $x$
  - Schrittfall ergibt sich durch Anwendung von  $s$  auf das “alte” Ergebnis

## ● Subtraktion zweier Zahlen $x$ und $y$

- Suche, von Null beginnend, die kleinste Zahl  $z$  mit  $z + y = x$

# BEISPIELE FÜR BAUKASTENPROGRAMMIERUNG

## ● Addition von 2

- Ergebnis entsteht durch zweifache Addition von 1  $x+2 = x+1+1$
- Abstrahiert: doppelte Anwendung der Nachfolgerfunktion  $x+2 = s(s(x))$
- Die Funktion “Addition von 2” ist also die Komposition von  $s$  und  $s$

## ● Addition zweier Zahlen $x$ und $y$

- Ergebnis entsteht durch  $y$ -fache Addition von 1  $x+y = x+\underbrace{1 + \dots + 1}_{y\text{-mal}}$
- Iteration muß rekursiv beschrieben werden
  - Im Basisfall ( $y=0$ ) ist das Ergebnis  $x$   $x+0 = x$
  - Im Schrittfall verwenden wir die Addition von  $x$  und  $y$ , um die Addition von  $x$  und  $y+1$  zu bestimmen  $x+(y+1) = (x+y)+1$
- Abstrahiert:
  - Basisfall ergibt sich durch Anwendung der Projektion  $pr_1^1$  auf  $x$
  - Schrittfall ergibt sich durch Anwendung von  $s$  auf das “alte” Ergebnis

## ● Subtraktion zweier Zahlen $x$ und $y$

- Suche, von Null beginnend, die kleinste Zahl  $z$  mit  $z + y = x$

Suche ist nicht wirklich erforderlich – iterierte Anwendung der Vorgängerfunktion reicht

## ● Grundfunktionen

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

# BAUSTEINE REKURSIVER FUNKTIONEN – PRÄZISIERT

## ● Grundfunktionen

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

## ● Operationen auf Funktionen

- **Komposition**  $f \circ (g_1, \dots, g_n) : \mathbb{N}^k \rightarrow \mathbb{N}$  ( $g_1, \dots, g_n : \mathbb{N}^k \rightarrow \mathbb{N}$ ,  $f : \mathbb{N}^n \rightarrow \mathbb{N}$ )

Für  $h = f \circ (g_1, \dots, g_n)$  gilt  $h(\vec{x}) = f(g_1(\vec{x}), \dots, g_n(\vec{x}))$

# BAUSTEINE REKURSIVER FUNKTIONEN – PRÄZISIERT

## ● Grundfunktionen

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

## ● Operationen auf Funktionen

- **Komposition**  $f \circ (g_1, \dots, g_n) : \mathbb{N}^k \rightarrow \mathbb{N}$  ( $g_1, \dots, g_n : \mathbb{N}^k \rightarrow \mathbb{N}, f : \mathbb{N}^n \rightarrow \mathbb{N}$ )  
Für  $h = f \circ (g_1, \dots, g_n)$  gilt  $h(\vec{x}) = f(g_1(\vec{x}), \dots, g_n(\vec{x}))$
- **Primitive Rekursion**  $Pr[f, g] : \mathbb{N}^k \rightarrow \mathbb{N}$  ( $f : \mathbb{N}^{k-1} \rightarrow \mathbb{N}, g : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$ )  
Für  $h = Pr[f, g]$  gilt  $h(\vec{x}, 0) = f(\vec{x})$ ,  
und  $h(\vec{x}, y+1) = g(\vec{x}, y, h(\vec{x}, y))$

# BAUSTEINE REKURSIVER FUNKTIONEN – PRÄZISIERT

## ● Grundfunktionen

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

## ● Operationen auf Funktionen

- **Komposition**  $f \circ (g_1, \dots, g_n) : \mathbb{N}^k \rightarrow \mathbb{N}$  ( $g_1, \dots, g_n : \mathbb{N}^k \rightarrow \mathbb{N}$ ,  $f : \mathbb{N}^n \rightarrow \mathbb{N}$ )

Für  $h = f \circ (g_1, \dots, g_n)$  gilt  $h(\vec{x}) = f(g_1(\vec{x}), \dots, g_n(\vec{x}))$

- **Primitive Rekursion**  $Pr[f, g] : \mathbb{N}^k \rightarrow \mathbb{N}$  ( $f : \mathbb{N}^{k-1} \rightarrow \mathbb{N}$ ,  $g : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$ )

Für  $h = Pr[f, g]$  gilt  $h(\vec{x}, 0) = f(\vec{x})$ ,  
und  $h(\vec{x}, y+1) = g(\vec{x}, y, h(\vec{x}, y))$

- **$\mu$ -Operator (Minimierung)**  $\mu f : \mathbb{N}^k \rightarrow \mathbb{N}$  ( $f : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$ )

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

Notation auch:  $h(\mathbf{x}) = \mu_z [f(\mathbf{x}, z) = 0]$

# OPERATIONEN ENTSPRECHEN PROGRAMMSTRUKTUREN

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

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

$$\vdots$$

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

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

$$(h \hat{=} h(x_1, \dots, x_k) \text{ für } h = f \circ (g_1, \dots, g_n))$$

# OPERATIONEN ENTSPRECHEN PROGRAMMSTRUKTUREN

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

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

$\vdots$

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

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

( $\mathbf{h} \hat{=} h(x_1, \dots, x_k)$  für  $h = f \circ (g_1, \dots, g_n)$ )

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

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

for  $i:=1$  to  $y$  do

$h := g(x_1, \dots, x_k, i-1, h)$

od

( $\mathbf{h} \hat{=} h(x_1, \dots, x_k, y)$  für  $h = Pr[f, g]$ )

# OPERATIONEN ENTSPRECHEN PROGRAMMSTRUKTUREN

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

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

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

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

( $\mathbf{h} \hat{=} h(x_1, \dots, x_k)$  für  $h = f \circ (g_1, \dots, g_n)$ )

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

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

for  $i:=1$  to  $y$  do

$h := g(x_1, \dots, x_k, i-1, h)$

od

( $\mathbf{h} \hat{=} h(x_1, \dots, x_k, y)$  für  $h = Pr[f, g]$ )

- **Minimierung  $\hat{=}$  Suche mit While-schleife**

$y := 0;$

while  $f(x_1, \dots, x_k, y) \neq 0$  do

$y := y + 1$

od;

$h := y$

(unbegrenzte Suche)

( $\mathbf{h} \hat{=} h(x_1, \dots, x_k)$  für  $h = \mu f$ )

# PRIMITIV- UND $\mu$ -REKURSIVE FUNKTIONEN

- **$\mathcal{PR}$ : Menge der primitiv-rekursiven Funktionen**
  - Nachfolger-, Projektions- oder Konstantenfunktion sowie
  - Alle Funktionen, die aus primitiv-rekursiven Funktionen durch Komposition oder primitive Rekursion entstehen

# PRIMITIV- UND $\mu$ -REKURSIVE FUNKTIONEN

- **$\mathcal{PR}$ : Menge der primitiv-rekursiven Funktionen**

- Nachfolger-, Projektions- oder Konstantenfunktion sowie
- Alle Funktionen, die aus primitiv-rekursiven Funktionen durch Komposition oder primitive Rekursion entstehen

**Primitiv-rekursive Funktionen sind total** (terminieren immer)

# PRIMITIV- UND $\mu$ -REKURSIVE FUNKTIONEN

- **$\mathcal{PR}$ : Menge der primitiv-rekursiven Funktionen**

- Nachfolger-, Projektions- oder Konstantenfunktion sowie
- Alle Funktionen, die aus primitiv-rekursiven Funktionen durch Komposition oder primitive Rekursion entstehen

**Primitiv-rekursive Funktionen sind total** (terminieren immer)

- **$\mathcal{R}$ : Menge der ( $\mu$ -)rekursiven Funktionen**

- Nachfolger-, Projektions- oder Konstantenfunktion sowie
- Alle Funktionen, die aus  $\mu$ -rekursiven Funktionen durch Komposition, primitive Rekursion oder Minimierung entstehen

# PRIMITIV- UND $\mu$ -REKURSIVE FUNKTIONEN

- **$\mathcal{PR}$ : Menge der primitiv-rekursiven Funktionen**

- Nachfolger-, Projektions- oder Konstantenfunktion sowie
- Alle Funktionen, die aus primitiv-rekursiven Funktionen durch Komposition oder primitive Rekursion entstehen

**Primitiv-rekursive Funktionen sind total** (terminieren immer)

- **$\mathcal{R}$ : Menge der ( $\mu$ -)rekursiven Funktionen**

- Nachfolger-, Projektions- oder Konstantenfunktion sowie
- Alle Funktionen, die aus  $\mu$ -rekursiven Funktionen durch Komposition, primitive Rekursion oder Minimierung entstehen

**$\mu$ -rekursive Funktionen können partiell sein**

# PRIMITIV- UND $\mu$ -REKURSIVE FUNKTIONEN

- **$\mathcal{PR}$** : Menge der **primitiv-rekursiven Funktionen**

- Nachfolger-, Projektions- oder Konstantenfunktion sowie
- Alle Funktionen, die aus primitiv-rekursiven Funktionen durch Komposition oder primitive Rekursion entstehen

**Primitiv-rekursive Funktionen sind total** (terminieren immer)

- **$\mathcal{R}$** : Menge der **( $\mu$ -)rekursiven Funktionen**

- Nachfolger-, Projektions- oder Konstantenfunktion sowie
- Alle Funktionen, die aus  $\mu$ -rekursiven Funktionen durch Komposition, primitive Rekursion oder Minimierung entstehen

**$\mu$ -rekursive Funktionen können partiell sein**

- **$\mathcal{TR}$** : Menge der **totalen rekursiven Funktionen**

# PRIMITIV- UND $\mu$ -REKURSIVE FUNKTIONEN

- **$\mathcal{PR}$** : Menge der **primitiv-rekursiven Funktionen**

- Nachfolger-, Projektions- oder Konstantenfunktion sowie
- Alle Funktionen, die aus primitiv-rekursiven Funktionen durch Komposition oder primitive Rekursion entstehen

**Primitiv-rekursive Funktionen sind total** (terminieren immer)

- **$\mathcal{R}$** : Menge der ( **$\mu$** -)rekursiven Funktionen

- Nachfolger-, Projektions- oder Konstantenfunktion sowie
- Alle Funktionen, die aus  $\mu$ -rekursiven Funktionen durch Komposition, primitive Rekursion oder Minimierung entstehen

**$\mu$ -rekursive Funktionen können partiell sein**

- **$\mathcal{TR}$** : Menge der **totalen** rekursiven Funktionen

**Es gilt  $\mathcal{PR} \subset \mathcal{TR} \subset \mathcal{R}$**

- Teilmengenbeziehung gilt offensichtlich (Beispiele für Unterschiede später)

## 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  $f_1$ ?

- Stelligkeitsanalyse:

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

## BEISPIEL EINER PRIMITIV-REKURSIVEN FUNKTION

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

Was macht  $f_1$ ?

- **Stelligkeitsanalyse:**

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

- **Auswertung durch schrittweises Einsetzen**

Beispiel:  $f_1(2, 2)$

## BEISPIEL EINER PRIMITIV-REKURSIVEN FUNKTION

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

Was macht  $f_1$ ?

- **Stelligkeitsanalyse:**

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

- **Auswertung durch schrittweises Einsetzen**

Beispiel:  $f_1(2, 2) = Pr[pr_1^1, s \circ pr_3^3](2, 2)$

## BEISPIEL EINER PRIMITIV-REKURSIVEN FUNKTION

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

Was macht  $f_1$ ?

- **Stelligkeitsanalyse:**

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

- **Auswertung durch schrittweises Einsetzen**

Beispiel:  $f_1(2, 2) = Pr[pr_1^1, s \circ pr_3^3](2, 2)$

– Einsetzen des Definitionsschemas bei Operationen

## BEISPIEL EINER PRIMITIV-REKURSIVEN FUNKTION

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

Was macht  $f_1$ ?

- **Stelligkeitsanalyse:**

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

- **Auswertung durch schrittweises Einsetzen**

Beispiel:  $f_1(2, 2) = (s \circ pr_3^3)(2, 1, f_1(2, 1))$

– Einsetzen des Definitionsschemas bei Operationen

## BEISPIEL EINER PRIMITIV-REKURSIVEN FUNKTION

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

Was macht  $f_1$ ?

- **Stelligkeitsanalyse:**

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

- **Auswertung durch schrittweises Einsetzen**

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

– Einsetzen des Definitionsschemas bei Operationen

## BEISPIEL EINER PRIMITIV-REKURSIVEN FUNKTION

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

Was macht  $f_1$ ?

- **Stelligkeitsanalyse:**

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

- **Auswertung durch schrittweises Einsetzen**

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

– Einsetzen des Definitionsschemas bei Operationen

## BEISPIEL EINER PRIMITIV-REKURSIVEN FUNKTION

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

Was macht  $f_1$ ?

- **Stelligkeitsanalyse:**

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

- **Auswertung durch schrittweises Einsetzen**

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

– Einsetzen des Definitionsschemas bei Operationen

– Direkte Auswertung von Argumenten bei Grundfunktionen

## BEISPIEL EINER PRIMITIV-REKURSIVEN FUNKTION

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

Was macht  $f_1$ ?

- **Stelligkeitsanalyse:**

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

- **Auswertung durch schrittweises Einsetzen**

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

- Einsetzen des Definitionsschemas bei Operationen
- Direkte Auswertung von Argumenten bei Grundfunktionen

## BEISPIEL EINER PRIMITIV-REKURSIVEN FUNKTION

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

- **Stelligkeitsanalyse:**

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

- **Auswertung durch schrittweises Einsetzen**

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

- Einsetzen des Definitionsschemas bei Operationen
- Direkte Auswertung von Argumenten bei Grundfunktionen

## BEISPIEL EINER PRIMITIV-REKURSIVEN FUNKTION

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

Was macht  $f_1$ ?

- **Stelligkeitsanalyse:**

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

- **Auswertung durch schrittweises Einsetzen**

Beispiel:  $f_1(2, 2) = (s \circ pr_3^3)(2, 1, s(2))$

- Einsetzen des Definitionsschemas bei Operationen
- Direkte Auswertung von Argumenten bei Grundfunktionen

## BEISPIEL EINER PRIMITIV-REKURSIVEN FUNKTION

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

Was macht  $f_1$ ?

- **Stelligkeitsanalyse:**

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

- **Auswertung durch schrittweises Einsetzen**

Beispiel:  $f_1(2, 2) = (s \circ pr_3^3)(2, 1, 3)$

- Einsetzen des Definitionsschemas bei Operationen
- Direkte Auswertung von Argumenten bei Grundfunktionen

## BEISPIEL EINER PRIMITIV-REKURSIVEN FUNKTION

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

Was macht  $f_1$ ?

- **Stelligkeitsanalyse:**

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

- **Auswertung durch schrittweises Einsetzen**

Beispiel:  $f_1(2, 2) = s(pr_3^3(2, 1, 3))$

- Einsetzen des Definitionsschemas bei Operationen
- Direkte Auswertung von Argumenten bei Grundfunktionen

## BEISPIEL EINER PRIMITIV-REKURSIVEN FUNKTION

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

Was macht  $f_1$ ?

- **Stelligkeitsanalyse:**

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

- **Auswertung durch schrittweises Einsetzen**

Beispiel:  $f_1(2, 2) = s(3)$

- Einsetzen des Definitionsschemas bei Operationen
- Direkte Auswertung von Argumenten bei Grundfunktionen

## BEISPIEL EINER PRIMITIV-REKURSIVEN FUNKTION

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

Was macht  $f_1$ ?

- **Stelligkeitsanalyse:**

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

- **Auswertung durch schrittweises Einsetzen**

Beispiel:  $f_1(2, 2) = 4$

- Einsetzen des Definitionsschemas bei Operationen
- Direkte Auswertung von Argumenten bei Grundfunktionen

## BEISPIEL EINER PRIMITIV-REKURSIVEN FUNKTION

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

Was macht  $f_1$ ?

- **Stelligkeitsanalyse:**

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

- **Auswertung durch schrittweises Einsetzen**

Beispiel:  $f_1(2, 2) = 4$

- Einsetzen des Definitionsschemas bei Operationen
- Direkte Auswertung von Argumenten bei Grundfunktionen

- **Analyse des rekursiven Verhaltens:**

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

## BEISPIEL EINER PRIMITIV-REKURSIVEN FUNKTION

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

Was macht  $f_1$ ?

- **Stelligkeitsanalyse:**

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

- **Auswertung durch schrittweises Einsetzen**

Beispiel:  $f_1(2, 2) = 4$

- Einsetzen des Definitionsschemas bei Operationen
- Direkte Auswertung von Argumenten bei Grundfunktionen

- **Analyse des rekursiven Verhaltens:**

- $f_1(x, 0) = pr_1^1(x) = x$
- $f_1(x, y+1) = (s \circ pr_3^3)(x, y, f_1(x, y)) = s(f_1(x, y)) = f_1(x, y) + 1$

## BEISPIEL EINER PRIMITIV-REKURSIVEN FUNKTION

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

- **Stelligkeitsanalyse:**

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

- **Auswertung durch schrittweises Einsetzen**

Beispiel:  $f_1(2, 2) = 4$

- Einsetzen des Definitionsschemas bei Operationen
- Direkte Auswertung von Argumenten bei Grundfunktionen

- **Analyse des rekursiven Verhaltens:**

- $f_1(x, 0) = pr_1^1(x) = x$
- $f_1(x, y+1) = (s \circ pr_3^3)(x, y, f_1(x, y)) = s(f_1(x, y)) = f_1(x, y) + 1$

**Das ist die Rekursionsgleichung der Addition**

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

# 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

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

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

$= \perp$
- $f_3 = \mu f_1$       $f_3(x) = \begin{cases} 0 & \text{falls } x = 0 \\ \perp & \text{sonst} \end{cases}$
- $f_4 = \mu h$  mit  $h(x, y) = \begin{cases} 0 & \text{falls } x = y \\ \perp & \text{sonst} \end{cases}$

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

# 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  $h(x, y) = \begin{cases} 0 & \text{falls } x = y \\ \perp & \text{sonst} \end{cases}$ 

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

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

## ‘PROGRAMMIERUNG’ $\mu$ -REKURSIVER FUNKTIONEN

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

Finde ‘Programm’ im Kalkül der  $\mu$ -rekursiven Funktionen

## ‘PROGRAMMIERUNG’ $\mu$ -REKURSIVER FUNKTIONEN

- Vorgängerfunktion  $p: \mathbb{N} \rightarrow \mathbb{N}$   $p(n) = n - 1$   
Finde ‘Programm’ im Kalkül der  $\mu$ -rekursiven Funktionen
- Analysiere rekursives Verhalten:

# ‘PROGRAMMIERUNG’ $\mu$ -REKURSIVER FUNKTIONEN

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

Finde ‘Programm’ im Kalkül der  $\mu$ -rekursiven Funktionen

- Analysiere rekursives Verhalten:

–  $p(0) = 0 - 1 = 0$  (Subtraktion liefert niemals weniger als 0!!)

# ‘PROGRAMMIERUNG’ $\mu$ -REKURSIVER FUNKTIONEN

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

Finde ‘Programm’ im Kalkül der  $\mu$ -rekursiven Funktionen

- **Analysiere rekursives Verhalten:**

–  $p(0) = 0 \dot{-} 1 = 0$  (Subtraktion liefert niemals weniger als 0!!)

–  $p(y+1) = (y+1) \dot{-} 1 = y$

# ‘PROGRAMMIERUNG’ $\mu$ -REKURSIVER FUNKTIONEN

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

Finde ‘Programm’ im Kalkül der  $\mu$ -rekursiven Funktionen

- **Analysiere rekursives Verhalten:**

–  $p(0) = 0 \dot{-} 1 = 0$  (Subtraktion liefert niemals weniger als 0!!)

–  $p(y+1) = (y+1) \dot{-} 1 = y$

- **Beschreibe Verhalten als Primitive Rekursion:**

– Benötigt:  $f: \mathbb{N}^0 \rightarrow \mathbb{N}$  mit  $p(0) = f() = 0 \quad \mapsto f = c_0^0$

# ‘PROGRAMMIERUNG’ $\mu$ -REKURSIVER FUNKTIONEN

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

Finde ‘Programm’ im Kalkül der  $\mu$ -rekursiven Funktionen

- **Analysiere rekursives Verhalten:**

–  $p(0) = 0 - 1 = 0$  (Subtraktion liefert niemals weniger als 0!!)

–  $p(y+1) = (y+1) - 1 = y$

- **Beschreibe Verhalten als Primitive Rekursion:**

– Benötigt:  $f: \mathbb{N}^0 \rightarrow \mathbb{N}$  mit  $p(0) = f() = 0$   $\mapsto f = c_0^0$

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

# ‘PROGRAMMIERUNG’ $\mu$ -REKURSIVER FUNKTIONEN

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

Finde ‘Programm’ im Kalkül der  $\mu$ -rekursiven Funktionen

- **Analysiere rekursives Verhalten:**

–  $p(0) = 0 \dot{-} 1 = 0$  (Subtraktion liefert niemals weniger als 0!!)

–  $p(y+1) = (y+1) \dot{-} 1 = y$

- **Beschreibe Verhalten als Primitive Rekursion:**

– Benötigt:  $f: \mathbb{N}^0 \rightarrow \mathbb{N}$  mit  $p(0) = f() = 0$   $\mapsto f = c_0^0$

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

Abstraktes “Programm”:  $p = Pr[c_0^0, pr_1^2]$

# BEISPIELE PRIMITIV-REKURSIVER FUNKTIONEN

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

$$sub(n, m) = n - m$$

# BEISPIELE PRIMITIV-REKURSIVER FUNKTIONEN

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

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

- $sub(x, 0) = x = pr_1^1(x)$

- $sub(x, y+1) = x \dot{-} (y+1) = (x \dot{-} y) \dot{-} 1 = p(x \dot{-} y) = (p \circ pr_3^3)(x, y, sub(x, y))$

## BEISPIELE PRIMITIV-REKURSIVER FUNKTIONEN

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

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

- $sub(x, 0) = x = pr_1^1(x)$

- $sub(x, y+1) = x \dot{-} (y+1) = (x \dot{-} y) \dot{-} 1 = p(x \dot{-} y) = (p \circ pr_3^3)(x, y, sub(x, y))$

Abstraktes Programm:  $sub = Pr[pr_1^1, p \circ pr_3^3]$

## BEISPIELE PRIMITIV-REKURSIVER FUNKTIONEN

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

# BEISPIELE PRIMITIV-REKURSIVER FUNKTIONEN

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

# BEISPIELE PRIMITIV-REKURSIVER FUNKTIONEN

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

# BEISPIELE PRIMITIV-REKURSIVER FUNKTIONEN

- **Subtraktion**  $sub : \mathbb{N}^2 \rightarrow \mathbb{N}$   $sub(n, m) = n \dot{-} m$

  - $sub(x, 0) = x = pr_1^1(x)$
  - $sub(x, y+1) = x \dot{-} (y+1) = (x \dot{-} y) \dot{-} 1 = p(x \dot{-} y) = (p \circ pr_3^3)(x, y, sub(x, y))$

Abstraktes Programm:  $sub = Pr[pr_1^1, p \circ pr_3^3]$
- **Multiplikation**  $mul : \mathbb{N}^2 \rightarrow \mathbb{N}$   $mul(n, m) = n * m$

  - $mul(x, 0) = 0 = c_0^1(x)$
  - $mul(x, y+1) = mul(x, y) + x = (add \circ (pr_1^3, pr_3^3))(x, y, mul(x, y))$

$mul = Pr[c_0^1, (add \circ (pr_1^3, pr_3^3))]$
- **Exponentiierung**  $exp : \mathbb{N}^2 \rightarrow \mathbb{N}$   $exp(n, m) = n^m$

# BEISPIELE PRIMITIV-REKURSIVER FUNKTIONEN

- **Subtraktion**  $sub : \mathbb{N}^2 \rightarrow \mathbb{N}$   $sub(n, m) = n - m$

  - $sub(x, 0) = x = pr_1^1(x)$
  - $sub(x, y+1) = x - (y+1) = (x - y) - 1 = p(x - y) = (p \circ pr_3^3)(x, y, sub(x, y))$

Abstraktes Programm:  $sub = Pr[pr_1^1, p \circ pr_3^3]$
- **Multiplikation**  $mul : \mathbb{N}^2 \rightarrow \mathbb{N}$   $mul(n, m) = n * m$

  - $mul(x, 0) = 0 = c_0^1(x)$
  - $mul(x, y+1) = mul(x, y) + x = (add \circ (pr_1^3, pr_3^3))(x, y, mul(x, y))$

$mul = Pr[c_0^1, (add \circ (pr_1^3, pr_3^3))]$
- **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**  $sub : \mathbb{N}^2 \rightarrow \mathbb{N}$   $sub(n, m) = n - m$

  - $sub(x, 0) = x = pr_1^1(x)$
  - $sub(x, y+1) = x - (y+1) = (x - y) - 1 = p(x - y) = (p \circ pr_3^3)(x, y, sub(x, y))$

Abstraktes Programm:  $sub = Pr[pr_1^1, p \circ pr_3^3]$
- **Multiplikation**  $mul : \mathbb{N}^2 \rightarrow \mathbb{N}$   $mul(n, m) = n * m$

  - $mul(x, 0) = 0 = c_0^1(x)$
  - $mul(x, y+1) = mul(x, y) + x = (add \circ (pr_1^3, pr_3^3))(x, y, mul(x, y))$

$mul = Pr[c_0^1, (add \circ (pr_1^3, pr_3^3))]$
- **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**  $sub : \mathbb{N}^2 \rightarrow \mathbb{N}$   $sub(n, m) = n - m$

  - $sub(x, 0) = x = pr_1^1(x)$
  - $sub(x, y+1) = x - (y+1) = (x - y) - 1 = p(x - y) = (p \circ pr_3^3)(x, y, sub(x, y))$

Abstraktes Programm:  $sub = Pr[pr_1^1, p \circ pr_3^3]$
- **Multiplikation**  $mul : \mathbb{N}^2 \rightarrow \mathbb{N}$   $mul(n, m) = n * m$

  - $mul(x, 0) = 0 = c_0^1(x)$
  - $mul(x, y+1) = mul(x, y) + x = (add \circ (pr_1^3, pr_3^3))(x, y, mul(x, y))$

$mul = Pr[c_0^1, (add \circ (pr_1^3, pr_3^3))]$
- **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**  $sub : \mathbb{N}^2 \rightarrow \mathbb{N}$   $sub(n, m) = n - m$

  - $sub(x, 0) = x = pr_1^1(x)$
  - $sub(x, y+1) = x - (y+1) = (x - y) - 1 = p(x - y) = (p \circ pr_3^3)(x, y, sub(x, y))$

Abstraktes Programm:  $sub = Pr[pr_1^1, p \circ pr_3^3]$
- **Multiplikation**  $mul : \mathbb{N}^2 \rightarrow \mathbb{N}$   $mul(n, m) = n * m$

  - $mul(x, 0) = 0 = c_0^1(x)$
  - $mul(x, y+1) = mul(x, y) + x = (add \circ (pr_1^3, pr_3^3))(x, y, mul(x, y))$

$mul = Pr[c_0^1, (add \circ (pr_1^3, pr_3^3))]$
- **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**  $sub : \mathbb{N}^2 \rightarrow \mathbb{N}$   $sub(n, m) = n - m$

  - $sub(x, 0) = x = pr_1^1(x)$
  - $sub(x, y+1) = x - (y+1) = (x - y) - 1 = p(x - y) = (p \circ pr_3^3)(x, y, sub(x, y))$

Abstraktes Programm:  $sub = Pr[pr_1^1, p \circ pr_3^3]$
- **Multiplikation**  $mul : \mathbb{N}^2 \rightarrow \mathbb{N}$   $mul(n, m) = n * m$

  - $mul(x, 0) = 0 = c_0^1(x)$
  - $mul(x, y+1) = mul(x, y) + x = (add \circ (pr_1^3, pr_3^3))(x, y, mul(x, y))$

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

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

- **Definition durch Fallunterscheidung**

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

Wende Signum-Funktion auf Testergebnis an und multipliziere auf

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

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

# PRIMITIV-REKURSIVE PROGRAMMIERTECHNIKEN

- **Definition durch Fallunterscheidung**

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

Wende Signum-Funktion auf Testergebnis an und multipliziere auf

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

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

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

- Generelles Produkt**  $\Pi_{i=0}^r f(\vec{x}, i)$  ( $f: \mathbb{N}^{k+1} \rightarrow \mathbb{N}$  p.r.)

# PRIMITIV-REKURSIVE PROGRAMMIERTECHNIKEN

## ● Definition durch Fallunterscheidung

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

Wende Signum-Funktion auf Testergebnis an und multipliziere auf

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

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

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

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

$$- \sum_{i=0}^0 f(\vec{x}, i) = f(\vec{x}, 0)$$

$$- \sum_{i=0}^{y+1} f(\vec{x}, i) = (\sum_{i=0}^y f(\vec{x}, i)) + f(\vec{x}, y + 1)$$

$$\Sigma f = Pr[f \circ (pr_1^1, c_0^1), add \circ (pr_3^3, f \circ (pr_1^3, s \circ pr_2^3))] \quad (\text{für } k=1)$$

# PRIMITIV-REKURSIVE PROGRAMMIERTECHNIKEN

## ● Definition durch Fallunterscheidung

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

Wende Signum-Funktion auf Testergebnis an und multipliziere auf

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

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

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

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

$$- \Sigma_{i=0}^0 f(\vec{x}, i) = f(\vec{x}, 0)$$

$$- \Sigma_{i=0}^{y+1} f(\vec{x}, i) = (\Sigma_{i=0}^y f(\vec{x}, i)) + f(\vec{x}, y + 1)$$

$$\Sigma f = Pr[f \circ (pr_1^1, c_0^1), add \circ (pr_3^3, f \circ (pr_1^3, s \circ pr_2^3))] \quad (\text{für } k=1)$$

$$\Pi f = Pr[f \circ (pr_1^1, c_0^1), mul \circ (pr_3^3, f \circ (pr_1^3, s \circ pr_2^3))] \quad (\text{Lösungen für } k > 1 \text{ analog})$$

- **Beschränkte Minimierung**

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

Schreibweise:  $h(\vec{x}, t) = Mn_t[f](\vec{x})$

- **Beschränkte Minimierung**

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

Schreibweise:  $h(\vec{x}, t) = Mn_t[f](\vec{x})$

**Rekursives Verhalten:**

$$- h(\vec{x}, 0) = \begin{cases} 0 & \text{falls } f(\vec{x}, 0) = 0 \\ 1 & \text{sonst} \end{cases} = \text{sign}(f(\vec{x}, 0))$$

## ● Beschränkte Minimierung

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

Schreibweise:  $h(\vec{x}, t) = \mathbf{Mn}_t[f](\vec{x})$

**Rekursives Verhalten:**

$$- h(\vec{x}, 0) = \begin{cases} 0 & \text{falls } f(\vec{x}, 0) = 0 \\ 1 & \text{sonst} \end{cases} = \text{sign}(f(\vec{x}, 0))$$

$$- h(\vec{x}, t+1) = \begin{cases} h(\vec{x}, t) & \text{falls } h(\vec{x}, t) \leq t \\ t+1 & \text{falls } h(\vec{x}, t) = t+1 \text{ und } f(\vec{x}, t+1) = 0 \\ t+2 & \text{sonst} \end{cases}$$

## ● Beschränkte Minimierung

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

Schreibweise:  $h(\vec{x}, t) = \mathbf{Mn}_t[f](\vec{x})$

**Rekursives Verhalten:**

$$\begin{aligned} - h(\vec{x}, 0) &= \begin{cases} 0 & \text{falls } f(\vec{x}, 0) = 0 \\ 1 & \text{sonst} \end{cases} = \text{sign}(f(\vec{x}, 0)) \\ - h(\vec{x}, t+1) &= \begin{cases} h(\vec{x}, t) & \text{falls } h(\vec{x}, t) \leq t \\ t+1 & \text{falls } h(\vec{x}, t) = t+1 \text{ und } f(\vec{x}, t+1) = 0 \\ t+2 & \text{sonst} \end{cases} \end{aligned}$$

**Programmierbar mit Fallunterscheidung und primitiver Rekursion (aufwendiger Ausdruck)**

**$h$  ist primitiv rekursiv, wenn  $f$  primitiv rekursiv ist**

## 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*** :  $\mathbb{N}^2 \rightarrow \mathbb{N}$

Beweise in Übungen selbst durchführen !

# BERECHENBARE NUMERIERUNG VON ZAHLENPAAREN

5							
4							
3							
2							
1							
0							
$y/x$	0	1	2	3	4	5	...

# BERECHENBARE NUMERIERUNG VON ZAHLENPAAREN

5							
4							
3							
2							
1							
0	0						
$y/x$	0	1	2	3	4	5	...

# BERECHENBARE NUMERIERUNG VON ZAHLENPAAREN

5							
4							
3							
2							
1							
0	0	1					
$y/x$	0	1	2	3	4	5	...

# BERECHENBARE NUMERIERUNG VON ZAHLENPAAREN

5							
4							
3							
2							
1	2						
0	0	1					
$y/x$	0	1	2	3	4	5	...

# BERECHENBARE NUMERIERUNG VON ZAHLENPAAREN

5							
4							
3							
2							
1	<b>2</b>						
0	<b>0</b>	<b>1</b>	<b>3</b>				
$y/x$	0	1	2	3	4	5	...

# BERECHENBARE NUMERIERUNG VON ZAHLENPAAREN

5							
4							
3							
2							
1	<b>2</b>	<b>4</b>					
0	<b>0</b>	<b>1</b>	<b>3</b>				
$y/x$	<b>0</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	...

# BERECHENBARE NUMERIERUNG VON ZAHLENPAAREN

5							
4							
3							
2	<b>5</b>						
1	<b>2</b>	<b>4</b>					
0	<b>0</b>	<b>1</b>	<b>3</b>				
$y/x$	<b>0</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>...</b>

# BERECHENBARE NUMERIERUNG VON ZAHLENPAAREN

5							
4							
3	9						
2	5	8					
1	2	4	7				
0	0	1	3	6			
$y/x$	0	1	2	3	4	5	...

# BERECHENBARE NUMERIERUNG VON ZAHLENPAAREN

5							
4	14						
3	9	13					
2	5	8	12				
1	2	4	7	11			
0	0	1	3	6	10		
$y/x$	0	1	2	3	4	5	...

# BERECHENBARE NUMERIERUNG VON ZAHLENPAAREN

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

# BERECHENBARE NUMERIERUNG VON ZAHLENPAAREN

⋮	⋮	⋮	⋮	⋮	⋮	⋮	...
<b>5</b>	<b>20</b>						...
<b>4</b>	<b>14</b>	<b>19</b>					...
<b>3</b>	<b>9</b>	<b>13</b>	<b>18</b>				...
<b>2</b>	<b>5</b>	<b>8</b>	<b>12</b>	<b>17</b>			...
<b>1</b>	<b>2</b>	<b>4</b>	<b>7</b>	<b>11</b>	<b>16</b>		...
<b>0</b>	<b>0</b>	<b>1</b>	<b>3</b>	<b>6</b>	<b>10</b>	<b>15</b>	...
<i>y/x</i>	<b>0</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	...

# BERECHENBARE NUMERIERUNG VON ZAHLENPAAREN

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

$$\langle x, y \rangle := (\sum_{i=1}^{x+y} i) + y$$

=

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

“Standard-Tupelfunktion”

# BERECHENBARE NUMERIERUNG VON ZAHLENPAAREN

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

$$\langle x, y \rangle := (\sum_{i=1}^{x+y} i) + y$$

=

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

“Standard-Tupelfunktion”

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

# BERECHENBARE NUMERIERUNG VON ZAHLENPAAREN

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

$$\begin{aligned} \langle x, y \rangle &:= (\sum_{i=1}^{x+y} i) + y \\ &= \\ &(x+y)(x+y+1) \div 2 + y \end{aligned}$$

“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

⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	...
<b>5</b>	<b>20</b>							...
<b>4</b>	<b>14</b>	<b>19</b>						...
<b>3</b>	<b>9</b>	<b>13</b>	<b>18</b>					...
<b>2</b>	<b>5</b>	<b>8</b>	<b>12</b>	<b>17</b>				...
<b>1</b>	<b>2</b>	<b>4</b>	<b>7</b>	<b>11</b>	<b>16</b>			...
<b>0</b>	<b>0</b>	<b>1</b>	<b>3</b>	<b>6</b>	<b>10</b>	<b>15</b>		...
<b>y/x</b>	<b>0</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>		...

$$\begin{aligned}
 \langle x, y \rangle &:= (\sum_{i=1}^{x+y} i) + y \\
 &= \\
 &(x+y)(x+y+1) \div 2 + y
 \end{aligned}$$

**“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

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

$$\begin{aligned}
 \langle x, y \rangle &:= (\sum_{i=1}^{x+y} i) + y \\
 &= \\
 &(x+y)(x+y+1) \div 2 + y
 \end{aligned}$$

**“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  $g \langle x, y \rangle = f(x, y)$

# DIE ACKERMANN-FUNKTION (1928)

- Definiere Funktionen  $A_n$  iterativ:

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

# DIE ACKERMANN-FUNKTION (1928)

- Definiere Funktionen  $A_n$  iterativ:

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

Jede der Funktionen  $A_n$  ist primitiv-rekursiv

# DIE ACKERMANN-FUNKTION (1928)

- Definiere Funktionen  $A_n$  iterativ:

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

Jede der Funktionen  $A_n$  ist primitiv-rekursiv

- Wachstumsverhalten

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

# DIE ACKERMANN-FUNKTION (1928)

- Definiere Funktionen  $A_n$  iterativ:

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

Jede der Funktionen  $A_n$  ist primitiv-rekursiv

- Wachstumsverhalten

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

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

# DIE ACKERMANN-FUNKTION (1928)

- Definiere Funktionen  $A_n$  iterativ:

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

Jede der Funktionen  $A_n$  ist primitiv-rekursiv

- Wachstumsverhalten

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

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

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

# DIE ACKERMANN-FUNKTION (1928)

- Definiere Funktionen  $A_n$  iterativ:

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

Jede der Funktionen  $A_n$  ist primitiv-rekursiv

- Wachstumsverhalten

$$A_1(x) = 2x \quad (x \geq 1) \quad A_4(0) = 1$$

$$A_2(x) = 2^x \quad A_4(1) = 2$$

$$A_3(x) = \underbrace{2^{(2^{(2^{\dots^2})})}}_{x\text{-mal}} \quad A_4(2) = 2^2 = 4$$

# DIE ACKERMANN-FUNKTION (1928)

- Definiere Funktionen  $A_n$  iterativ:

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

Jede der Funktionen  $A_n$  ist primitiv-rekursiv

- Wachstumsverhalten

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

$$A_2(x) = 2^x \quad A_4(1) = 2$$

$$A_3(x) = \underbrace{2^{(2^{(2^{\dots^2})})}}_{x\text{-mal}} \quad A_4(2) = 2^2 = 4$$

# DIE ACKERMANN-FUNKTION (1928)

- Definiere Funktionen  $A_n$  iterativ:

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

Jede der Funktionen  $A_n$  ist primitiv-rekursiv

- Wachstumsverhalten

$$\begin{array}{lll} A_1(x) = 2x \quad (x \geq 1) & A_4(0) = 1 & A_4(3) = 2^{2^{2^2}} = 65536 \\ A_2(x) = 2^x & A_4(1) = 2 & A_4(4) = \underbrace{2^{(2^{(2^{\dots^2})})}}_{65536\text{-mal}} \\ A_3(x) = \underbrace{2^{(2^{(2^{\dots^2})})}}_{x\text{-mal}} & A_4(2) = 2^2 = 4 & \end{array}$$

# DIE ACKERMANN-FUNKTION (1928)

- Definiere Funktionen  $A_n$  iterativ:

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

Jede der Funktionen  $A_n$  ist primitiv-rekursiv

- Wachstumsverhalten

$$\begin{array}{lll} A_1(x) = 2x \quad (x \geq 1) & A_4(0) = 1 & A_4(3) = 2^{2^{2^2}} = 65536 \\ A_2(x) = 2^x & A_4(1) = 2 & A_4(4) = \underbrace{2^{(2^{(2^{\dots^2})})}}_{65536\text{-mal}} \\ A_3(x) = \underbrace{2^{(2^{(2^{\dots^2})})}}_{x\text{-mal}} & A_4(2) = 2^2 = 4 & A_4(5) = \underbrace{2^{(2^{(2^{\dots^2})})}}_{A_4(4)\text{-mal}} \end{array}$$

# DIE ACKERMANN-FUNKTION (1928)

- Definiere Funktionen  $A_n$  iterativ:

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

Jede der Funktionen  $A_n$  ist primitiv-rekursiv

- Wachstumsverhalten

$$\begin{array}{lll} A_1(x) = 2x \quad (x \geq 1) & A_4(0) = 1 & A_4(3) = 2^{2^{2^2}} = 65536 \\ A_2(x) = 2^x & A_4(1) = 2 & A_4(4) = \underbrace{2^{(2^{(2^{\dots^2})})}}_{65536\text{-mal}} \\ A_3(x) = \underbrace{2^{(2^{(2^{\dots^2})})}}_{x\text{-mal}} & A_4(2) = 2^2 = 4 & A_4(5) = \underbrace{2^{(2^{(2^{\dots^2})})}}_{A_4(4)\text{-mal}} \end{array}$$

- Definiere  $A(x) := A_x(x)$  (**Große Ackermann-Funktion**)

# DIE ACKERMANN-FUNKTION (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

$$\begin{array}{lll} A_1(x) = 2x \quad (x \geq 1) & A_4(0) = 1 & A_4(3) = 2^{2^{2^2}} = 65536 \\ A_2(x) = 2^x & A_4(1) = 2 & A_4(4) = \underbrace{2^{(2^{(2^{\dots^2})})}}_{65536\text{-mal}} \\ A_3(x) = \underbrace{2^{(2^{(2^{\dots^2})})}}_{x\text{-mal}} & A_4(2) = 2^2 = 4 & A_4(5) = \underbrace{2^{(2^{(2^{\dots^2})})}}_{A_4(4)\text{-mal}} \end{array}$$

- Definiere  $A(x) := A_x(x)$  (**Große Ackermann-Funktion**)  
 $A$  ist nicht primitiv-rekursiv aber  $\mu$ -rekursiv und total

## WARUM KANN $A$ NICHT PRIMITIV-REKURSIV SEIN ?

- **Betrachte Schachtelungstiefe einer Funktion**

Anzahl der ineinander verschachtelten For-Schleifen

- Funktionen ohne Minimierung und primitive Rekursion  $\mapsto$  Tiefe 0
- Komposition mit Funktionen der Tiefe  $n$   $\mapsto$  Tiefe  $n$
- Primitive Rekursion mit Funktionen der Tiefe  $n$   $\mapsto$  Tiefe  $n+1$

## WARUM KANN $A$ NICHT PRIMITIV-REKURSIV SEIN ?

- **Betrachte Schachtelungstiefe einer Funktion**

Anzahl der ineinander verschachtelten For-Schleifen

- Funktionen ohne Minimierung und primitive Rekursion  $\mapsto$  Tiefe 0
- Komposition mit Funktionen der Tiefe  $n$   $\mapsto$  Tiefe  $n$
- Primitive Rekursion mit Funktionen der Tiefe  $n$   $\mapsto$  Tiefe  $n+1$

Primitiv-rekursive Funktionen haben eine begrenzte Schachtelungstiefe

# WARUM KANN $A$ NICHT PRIMITIV-REKURSIV SEIN ?

## ● Betrachte **Schachtelungstiefe** einer Funktion

Anzahl der ineinander verschachtelten For-Schleifen

- Funktionen ohne Minimierung und primitive Rekursion  $\mapsto$  **Tiefe 0**
- Komposition mit Funktionen der Tiefe  $n$   $\mapsto$  **Tiefe  $n$**
- Primitive Rekursion mit Funktionen der Tiefe  $n$   $\mapsto$  **Tiefe  $n+1$**

Primitiv-rekursive Funktionen haben eine begrenzte Schachtelungstiefe

## ● Beispiele

- Tiefe 1: Addition *add*, Vorgänger *p*, Signum *sign*
- Tiefe 2: Multiplikation *mul*, Subtraktion *sub*
- Tiefe 3: Exponentiation *exp*, Fakultät *fak*

# WARUM KANN $A$ NICHT PRIMITIV-REKURSIV SEIN ?

- **Betrachte Schachtelungstiefe einer Funktion**

Anzahl der ineinander verschachtelten For-Schleifen

- Funktionen ohne Minimierung und primitive Rekursion  $\mapsto$  **Tiefe 0**
- Komposition mit Funktionen der Tiefe  $n$   $\mapsto$  **Tiefe  $n$**
- Primitive Rekursion mit Funktionen der Tiefe  $n$   $\mapsto$  **Tiefe  $n+1$**

Primitiv-rekursive Funktionen haben eine begrenzte Schachtelungstiefe

- **Beispiele**

- Tiefe 1: Addition *add*, Vorgänger *p*, Signum *sign*
- Tiefe 2: Multiplikation *mul*, Subtraktion *sub*
- Tiefe 3: Exponentiation *exp*, Fakultät *fak*

- **Die Schachtelungstiefe von  $A$  ist unbegrenzbar**

- Die Berechnung von  $A(x)$  benötigt Schachtelungstiefe  $x$
- **$A$  kann nicht primitiv-rekursiv sein**

Präzises formales Argument ist sehr aufwendig

# WARUM IST DIE ACKERMANNFUNKTION BERECHENBAR?

- **Intuitiver Berechnungsmechanismus einfach**
  - Schrittweise Abarbeitung der Rekursion

# WARUM IST DIE ACKERMANNFUNKTION BERECHENBAR?

- **Intuitiver Berechnungsmechanismus einfach**
  - Schrittweise Abarbeitung der Rekursion
  - Abarbeitung des Rekursionsstacks ist ‘programmierbar’

# WARUM IST DIE ACKERMANNFUNKTION BERECHENBAR?

- **Intuitiver Berechnungsmechanismus einfach**

- Schrittweise Abarbeitung der Rekursion
- Abarbeitung des Rekursionsstacks ist ‘programmierbar’

- **A ist total und  $\mu$ -rekursiv**

- Beschreibe **Abarbeitungsfunktion**  $\delta$  eines Berechnungsstacks für  $A$  unter Verwendung der Standardtupelfunktion auf Worten über  $\mathbb{N}$

$$\delta\langle wx0 \rangle^* = w1$$

$$\delta\langle w01 \rangle^* = w2$$

$$\delta\langle w0(y+2) \rangle^* = w(y+4)$$

$$\delta\langle w(x+1)(y+1) \rangle^* = \langle wx(x+1)y \rangle^*$$

# WARUM IST DIE ACKERMANNFUNKTION BERECHENBAR?

- **Intuitiver Berechnungsmechanismus einfach**

- Schrittweise Abarbeitung der Rekursion
- Abarbeitung des Rekursionsstacks ist ‘programmierbar’

- **A ist total und  $\mu$ -rekursiv**

- Beschreibe **Abarbeitungsfunktion**  $\delta$  eines Berechnungsstacks für  $A$  unter Verwendung der Standardtupelfunktion auf Worten über  $\mathbb{N}$

$$\delta\langle wx0 \rangle^* = w1$$

$$\delta\langle w01 \rangle^* = w2$$

$$\delta\langle w0(y+2) \rangle^* = w(y+4)$$

$$\delta\langle w(x+1)(y+1) \rangle^* = \langle wx(x+1)y \rangle^*$$

- $\delta$  ist primitiv-rekursiv (Beschreibung des Programms aufwendig)

# WARUM IST DIE ACKERMANNFUNKTION BERECHENBAR?

- **Intuitiver Berechnungsmechanismus einfach**

- Schrittweise Abarbeitung der Rekursion
- Abarbeitung des Rekursionsstacks ist ‘programmierbar’

- **A ist total und  $\mu$ -rekursiv**

- Beschreibe **Abarbeitungsfunktion**  $\delta$  eines Berechnungsstacks für  $A$  unter Verwendung der Standardtupelfunktion auf Worten über  $\mathbb{N}$

$$\delta\langle wx0 \rangle^* = w1$$

$$\delta\langle w01 \rangle^* = w2$$

$$\delta\langle w0(y+2) \rangle^* = w(y+4)$$

$$\delta\langle w(x+1)(y+1) \rangle^* = \langle wx(x+1)y \rangle^*$$

- $\delta$  ist primitiv-rekursiv (Beschreibung des Programms aufwendig)
- Berechne  $A(x) = \delta^k \langle xx \rangle^*$ , wobei  $k = \mu_j [\pi_1^2(\delta^j \langle xx \rangle^*) = 1]$

# WARUM IST DIE ACKERMANNFUNKTION BERECHENBAR?

- **Intuitiver Berechnungsmechanismus einfach**

- Schrittweise Abarbeitung der Rekursion
- Abarbeitung des Rekursionsstacks ist ‘programmierbar’

- **A ist total und  $\mu$ -rekursiv**

- Beschreibe **Abarbeitungsfunktion**  $\delta$  eines Berechnungsstacks für  $A$  unter Verwendung der Standardtupelfunktion auf Worten über  $\mathbb{N}$

$$\delta\langle wx0 \rangle^* = w1$$

$$\delta\langle w01 \rangle^* = w2$$

$$\delta\langle w0(y+2) \rangle^* = w(y+4)$$

$$\delta\langle w(x+1)(y+1) \rangle^* = \langle wx(x+1)y \rangle^*$$

- $\delta$  ist primitiv-rekursiv (Beschreibung des Programms aufwendig)
- Berechne  $A(x) = \delta^k \langle xx \rangle^*$ , wobei  $k = \mu_j [\pi_1^2(\delta^j \langle xx \rangle^*) = 1]$
- **Berechnung terminiert**, da in jedem  $\delta$ -Schritt entweder die erste bearbeitete Zahl im Stack oder die Anzahl der Zahlen kleiner wird

# MIN-REKURSIVE FUNKTIONEN

## Funktionsdefinition ohne primitive Rekursion

## Funktionsdefinition ohne primitive Rekursion

- $\mathcal{R}_{min}$ : Menge der **min-rekursiven Funktionen**
  - Addition, Nachfolger, Projektions- oder Konstantenfunktion sowie
  - Alle Funktionen, die aus min-rekursiven Funktionen durch Komposition oder Minimierung entstehen

## Funktionsdefinition ohne primitive Rekursion

- $\mathcal{R}_{min}$ : Menge der **min-rekursiven Funktionen**
  - Addition, Nachfolger, Projektions- oder Konstantenfunktion sowie
  - Alle Funktionen, die aus min-rekursiven Funktionen durch Komposition oder Minimierung entstehen

Wichtiger Sonderfall für Vergleiche mit anderen Modellen

## Funktionsdefinition ohne primitive Rekursion

- $\mathcal{R}_{min}$ : Menge der **min-rekursiven Funktionen**
  - Addition, Nachfolger, Projektions- oder Konstantenfunktion sowie
  - Alle Funktionen, die aus min-rekursiven Funktionen durch Komposition oder Minimierung entstehen

Wichtiger Sonderfall für Vergleiche mit anderen Modellen

- $\mathcal{R}_{min} \subseteq \mathcal{R}$ : min-rekursive Funktionen sind  $\mu$ -rekursiv
  - Offensichtlich, da Addition  $\mu$ -rekursiv ist

## Funktionsdefinition ohne primitive Rekursion

- $\mathcal{R}_{min}$ : Menge der **min-rekursiven Funktionen**
  - Addition, Nachfolger, Projektions- oder Konstantenfunktion sowie
  - Alle Funktionen, die aus min-rekursiven Funktionen durch Komposition oder Minimierung entstehen

Wichtiger Sonderfall für Vergleiche mit anderen Modellen

- $\mathcal{R}_{min} \subseteq \mathcal{R}$ : min-rekursive Funktionen sind  $\mu$ -rekursiv
  - Offensichtlich, da Addition  $\mu$ -rekursiv ist
- $\mathcal{R} \subseteq \mathcal{R}_{min}$ :  $\mu$ -rekursive Funktionen sind min-rekursiv
  - Beschreibe Abarbeitung des Stacks einer primitiven Rekursion
  - Suche nach erstem erzeugten Stack der Länge 1 (Details aufwendig)

# AUSDRUCKSKRAFT REKURSIVER FUNKTIONEN

- $\mathcal{R} \subseteq \mathcal{T}$ : rekursive Funktionen sind Turing-berechenbar

# AUSDRUCKSKRAFT REKURSIVER FUNKTIONEN

- $\mathcal{R} \subseteq \mathcal{T}$ : rekursive Funktionen sind Turing-berechenbar
  - Alle Grundfunktionen sind konventionell berechenbar
  - Komposition, Primitive Rekursion und  $\mu$ -Operator sind berechenbar
  - Konventionell berechenbare Funktionen sind Turing-berechenbar

# AUSDRUCKSKRAFT REKURSIVER FUNKTIONEN

- $\mathcal{R} \subseteq \mathcal{T}$ : rekursive Funktionen sind Turing-berechenbar
  - Alle Grundfunktionen sind konventionell berechenbar
  - Komposition, Primitive Rekursion und  $\mu$ -Operator sind berechenbar
  - Konventionell berechenbare Funktionen sind Turing-berechenbar
- $\mathcal{T} \subseteq \mathcal{R}$ : Turing-berechenbare Funktionen sind rekursiv

# AUSDRUCKSKRAFT REKURSIVER FUNKTIONEN

- $\mathcal{R} \subseteq \mathcal{T}$ : rekursive Funktionen sind Turing-berechenbar
  - Alle Grundfunktionen sind konventionell berechenbar
  - Komposition, Primitive Rekursion und  $\mu$ -Operator sind berechenbar
  - Konventionell berechenbare Funktionen sind Turing-berechenbar
- $\mathcal{T} \subseteq \mathcal{R}$ : Turing-berechenbare Funktionen sind rekursiv
  - Codiere Konfigurationen (Worttupel) als Zahlentupel
  - Simuliere Konfigurationsübergänge  $\vdash$  als primitiv-rekursive Funktionen
  - Beschreibe Terminierung von  $\vdash^*$  als Suche nach Endkonfiguration
  - Semantik der Turingmaschine ist Iteration von  $\vdash$  bis Terminierung

# AUSDRUCKSKRAFT REKURSIVER FUNKTIONEN

- $\mathcal{R} \subseteq \mathcal{T}$ : rekursive Funktionen sind Turing-berechenbar
  - Alle Grundfunktionen sind konventionell berechenbar
  - Komposition, Primitive Rekursion und  $\mu$ -Operator sind berechenbar
  - Konventionell berechenbare Funktionen sind Turing-berechenbar
- $\mathcal{T} \subseteq \mathcal{R}$ : Turing-berechenbare Funktionen sind rekursiv
  - Codiere Konfigurationen (Worttupel) als Zahlentupel
  - Simuliere Konfigurationsübergänge  $\vdash$  als primitiv-rekursive Funktionen
  - Beschreibe Terminierung von  $\vdash^*$  als Suche nach Endkonfiguration
  - Semantik der Turingmaschine ist Iteration von  $\vdash$  bis Terminierung
- $\mathcal{PR}$  und  $\mathcal{TR}$  sind echte Unterklassen von  $\mathcal{R}$ 
  - $\mathcal{PR} \subseteq \mathcal{TR} \subseteq \mathcal{R}$  gilt offensichtlich
  - $\mathcal{TR} \neq \mathcal{R}$ : nicht alle rekursiven Funktionen sind total (z.B.  $f_3 = \mu \text{ add}$ )
  - $\mathcal{PR} \neq \mathcal{TR}$ : die Ackermannfunktion ist total, aber nicht primitiv rekursiv

## KONSEQUENZEN DER ÄQUIVALENZBEWEISE

- Für formale Argumente zur Berechenbarkeit können wahlweise Turingmaschinen oder  $\mu$ -rekursive Funktionen eingesetzt werden

## KONSEQUENZEN DER ÄQUIVALENZBEWEISE

- Für formale Argumente zur Berechenbarkeit können wahlweise Turingmaschinen oder  $\mu$ -rekursive Funktionen eingesetzt werden
- **Kleene Normalform Theorem:**  
Für jede berechenbare Funktion  $h$  kann man primitiv-rekursive Funktionen  $f$  und  $g$  konstruieren, so daß

$$h(x) = g(x, \mu f(x))$$

## KONSEQUENZEN DER ÄQUIVALENZBEWEISE

- Für formale Argumente zur Berechenbarkeit können wahlweise Turingmaschinen oder  $\mu$ -rekursive Funktionen eingesetzt werden
- **Kleene Normalform Theorem:**  
Für jede berechenbare Funktion  $h$  kann man primitiv-rekursive Funktionen  $f$  und  $g$  konstruieren, so daß

$$h(x) = g(x, \mu f(x))$$

- Simuliere Turingmaschine für  $h$  als  $\mu$ -rekursive Funktion
- $f$  ist die Funktion, die Terminierung charakterisiert
- $\mu f$  berechnet die Anzahl der Schritte bis zur Terminierung
- $g$  berechnet die Iteration der Konfigurationsübergänge

## KONSEQUENZEN DER ÄQUIVALENZBEWEISE

- Für formale Argumente zur Berechenbarkeit können wahlweise Turingmaschinen oder  $\mu$ -rekursive Funktionen eingesetzt werden
- **Kleene Normalform Theorem:**
  - Für jede berechenbare Funktion  $h$  kann man primitiv-rekursive Funktionen  $f$  und  $g$  konstruieren, so daß
$$h(x) = g(x, \mu f(x))$$
  - Simuliere Turingmaschine für  $h$  als  $\mu$ -rekursive Funktion
  - $f$  ist die Funktion, die Terminierung charakterisiert
  - $\mu f$  berechnet die Anzahl der Schritte bis zur Terminierung
  - $g$  berechnet die Iteration der Konfigurationsübergänge
- Berechenbare Funktionen kommen mit einer einzigen Minimierung (While-Schleife) aus

# REKURSIVE FUNKTIONEN IM RÜCKBLICK

- **Axiomatische Definition von Berechenbarkeit**
  - Abstrakter Funktionenkalkül ohne direkte Benennung der Argumente
  - Bausteine: Grundfunktionen  $s$ ,  $pr_k^n$ ,  $c_k^n$  und Operatoren  $\circ$ ,  $Pr$ ,  $\mu$

# REKURSIVE FUNKTIONEN IM RÜCKBLICK

- **Axiomatische Definition von Berechenbarkeit**
  - Abstrakter Funktionenkalkül ohne direkte Benennung der Argumente
  - Bausteine: Grundfunktionen  $s$ ,  $pr_k^n$ ,  $c_k^n$  und Operatoren  $\circ$ ,  $Pr$ ,  $\mu$
  - Rekursive Funktionen sind Turing-mächtig

# REKURSIVE FUNKTIONEN IM RÜCKBLICK

- **Axiomatische Definition von Berechenbarkeit**
  - Abstrakter Funktionenkalkül ohne direkte Benennung der Argumente
  - Bausteine: Grundfunktionen  $s$ ,  $pr_k^n$ ,  $c_k^n$  und Operatoren  $\circ$ ,  $Pr$ ,  $\mu$
  - Rekursive Funktionen sind Turing-mächtig
- **Minimierung ist der mächtigste Mechanismus**
  - Oft einfacher und eleganter als Anwendung der primitiven Rekursion
  - Terminierung im Allgemeinfall jedoch nicht gesichert

# REKURSIVE FUNKTIONEN IM RÜCKBLICK

- **Axiomatische Definition von Berechenbarkeit**
  - Abstrakter Funktionenkalkül ohne direkte Benennung der Argumente
  - Bausteine: Grundfunktionen  $s$ ,  $pr_k^n$ ,  $c_k^n$  und Operatoren  $\circ$ ,  $Pr$ ,  $\mu$
  - Rekursive Funktionen sind Turing-mächtig
- **Minimierung ist der mächtigste Mechanismus**
  - Oft einfacher und eleganter als Anwendung der primitiven Rekursion
  - Terminierung im Allgemeinfall jedoch nicht gesichert
  - Theoretisch braucht jede rekursive Funktion maximal eine Minimierung

# REKURSIVE FUNKTIONEN IM RÜCKBLICK

- **Axiomatische Definition von Berechenbarkeit**
  - Abstrakter Funktionenkalkül ohne direkte Benennung der Argumente
  - Bausteine: Grundfunktionen  $s$ ,  $pr_k^n$ ,  $c_k^n$  und Operatoren  $\circ$ ,  $Pr$ ,  $\mu$
  - Rekursive Funktionen sind Turing-mächtig
- **Minimierung ist der mächtigste Mechanismus**
  - Oft einfacher und eleganter als Anwendung der primitiven Rekursion
  - Terminierung im Allgemeinfall jedoch nicht gesichert
  - Theoretisch braucht jede rekursive Funktion maximal eine Minimierung
- **Primitiv-rekursive Funktionen als Unterklasse**
  - Abstraktes Programm darf keine Minimierung enthalten
  - Fast alle wichtigen Funktionen sind primitiv-rekursiv
  - Fast alle Programmier Techniken erhalten primitive Rekursivität
  - Es gibt totale berechenbare Funktionen, die nicht primitiv-rekursiv sind

# REKURSIVE FUNKTIONEN IM RÜCKBLICK

- **Axiomatische Definition von Berechenbarkeit**
  - Abstrakter Funktionenkalkül ohne direkte Benennung der Argumente
  - Bausteine: Grundfunktionen  $s$ ,  $pr_k^n$ ,  $c_k^n$  und Operatoren  $\circ$ ,  $Pr$ ,  $\mu$
  - Rekursive Funktionen sind Turing-mächtig
- **Minimierung ist der mächtigste Mechanismus**
  - Oft einfacher und eleganter als Anwendung der primitiven Rekursion
  - Terminierung im Allgemeinfall jedoch nicht gesichert
  - Theoretisch braucht jede rekursive Funktion maximal eine Minimierung
- **Primitiv-rekursive Funktionen als Unterklasse**
  - Abstraktes Programm darf keine Minimierung enthalten
  - Fast alle wichtigen Funktionen sind primitiv-rekursiv
  - Fast alle Programmier Techniken erhalten primitive Rekursivität
  - Es gibt totale berechenbare Funktionen, die nicht primitiv-rekursiv sind

Mehr Details in Lewis & Papadimitriou §4.7 und den auf der Webseite genannten Skripten