Theoretische Informatik Sommer 2011
Formalisierungen des Berechenbarkeitsbegriffs

Mischa Dieterle
Fachbereich Mathematik und Informatik
Philipps Universität Marburg

Inhalt

- *LOOP-Berechenbarkeit und primitiv rekursive Funktionen*
  - *While - Berechenbarkeit und μ-rekursive Funktionen*

Inhalt

Die Sprache LOOP

- **LOOP**-berechenbare Funktionen sind eine echte **Teilmenge** der **Turing**-berechenbaren Funktionen.
- Wir definieren zunächst die **Syntax** und **Semantik** von LOOP-Programmen.
- Die Syntax wird in vier **syntaktischen Grundbereichen** definiert:

Die Sprache LOOP

- **LOOP**-berechenbare Funktionen sind eine echte **Teilmenge** der **Turing**-berechenbaren Funktionen.
- Wir definieren zunächst die **Syntax** und **Semantik** von LOOP-Programmen.
- Die Syntax wird in vier **syntaktischen Grundbereichen** definiert:

Die Sprache LOOP

- **LOOP**-berechenbare Funktionen sind eine echte **Teilmenge** der **Turing**-berechenbaren Funktionen.
- Wir definieren zunächst die **Syntax** und **Semantik** von LOOP-Programmen.
- Die Syntax wird in vier **syntaktischen Grundbereichen** definiert:

Syntax von LOOP

Definition:

- *Variablen*: $\mathcal{V} := \{X_n \mid n \geq 1\}$
- *Wertzuweisungen*: $\mathcal{W} := \{ \quad X_i := X_j + 1, \quad X_i := 0 \quad \mid i, j \geq 1\}$
- *Anweisungen*: kleinste Menge $\mathcal{A}$ mit den Eigenschaften
  1. $\mathcal{W} \subseteq \mathcal{A}$
  2. $\alpha, \beta \in \mathcal{A}$ impliziert $\alpha; \beta \in \mathcal{A}$
  3. $\alpha \in \mathcal{A}, V \in \mathcal{V}$ impliziert $\underline{\texttt{loop}}\ V\ (\alpha) \in \mathcal{A}$
- *LOOP-Programme*: Die Menge $\mathcal{P}_{\mathrm{LOOP}}$ ist wie folgt definiert:

$$\mathcal{P}_{\mathrm{LOOP}} = \{\ \underline{\texttt{in}}(X_1, \ldots, X_n); \underline{\texttt{var}}(X_{n+1}, \ldots, X_m); \alpha, \underline{\texttt{out}} X_1 \mid m \geq 1, \alpha \in \mathcal{A}, X_i \text{ Variable in } \alpha \curvearrowright 1 \leq i \leq m\}$$

Syntax von LOOP

Definition:

- *Variablen*: $\mathcal{V} := \{X_n \mid n \geq 1\}$
- *Wertzuweisungen*: $\mathcal{W} := \{ \quad X_i := X_j + 1, \quad X_i := 0 \quad \mid i, j \geq 1\}$
- *Anweisungen*: kleinste Menge $\mathcal{A}$ mit den Eigenschaften
  1. $\mathcal{W} \subseteq \mathcal{A}$
  2. $\alpha, \beta \in \mathcal{A}$ impliziert $\alpha; \beta \in \mathcal{A}$
  3. $\alpha \in \mathcal{A}, V \in \mathcal{V}$ impliziert $\underline{\texttt{loop}}\ V\ (\alpha)\ \in \mathcal{A}$
- *LOOP-Programme*: Die Menge $\mathcal{P}_{\mathrm{LOOP}}$ ist wie folgt definiert:

$$\mathcal{P}_{\mathrm{LOOP}} = \{\ \underline{\texttt{in}}(X_1, \ldots, X_n); \underline{\texttt{var}}(X_{n+1}, \ldots, X_m); \alpha, \underline{\texttt{out}} X_1 \mid m \geq 1, \alpha \in \mathcal{A}, X_i \text{ Variable in } \alpha \curvearrowright 1 \leq i \leq m\}$$

Syntax von LOOP

Definition:

- *Variablen*: $\mathcal{V} := \{X_n \mid n \geq 1\}$
- *Wertzuweisungen*: $\mathcal{W} := \{ \quad X_i := X_j + 1, \quad X_i := 0 \quad \mid i, j \geq 1\}$
- *Anweisungen*: kleinste Menge $\mathcal{A}$ mit den Eigenschaften
  1. $\mathcal{W} \subseteq \mathcal{A}$
  2. $\alpha, \beta \in \mathcal{A}$ impliziert $\alpha; \beta \in \mathcal{A}$
  3. $\alpha \in \mathcal{A}, V \in \mathcal{V}$ impliziert $\underline{\texttt{loop}}\ V\ (\alpha)\ \in \mathcal{A}$
- *LOOP-Programme*: Die Menge $\mathcal{P}_{\mathrm{LOOP}}$ ist wie folgt definiert:

$$\mathcal{P}_{\mathrm{LOOP}} = \{\ \underline{\texttt{in}}(X_1, \ldots, X_n); \underline{\texttt{var}}(X_{n+1}, \ldots, X_m); \alpha, \underline{\texttt{out}} X_1 \mid m \geq 1, \alpha \in \mathcal{A}, X_i \text{ Variable in } \alpha \curvearrowright 1 \leq i \leq m\}$$

Syntax von LOOP

Definition:

- *Variablen*: $\mathcal{V} := \{X_n \mid n \geq 1\}$
- *Wertzuweisungen*: $\mathcal{W} := \{ \quad X_i := X_j + 1, \quad X_i := 0 \quad \mid i, j \geq 1\}$
- *Anweisungen*: kleinste Menge $\mathcal{A}$ mit den Eigenschaften
  1. $\mathcal{W} \subseteq \mathcal{A}$
  2. $\alpha, \beta \in \mathcal{A}$ impliziert $\alpha; \beta \in \mathcal{A}$
  3. $\alpha \in \mathcal{A}, V \in \mathcal{V}$ impliziert $\underline{\texttt{loop}}\ V\ (\alpha)\ \in \mathcal{A}$
- *LOOP-Programme*: Die Menge $\mathcal{P}_{\mathrm{LOOP}}$ ist wie folgt definiert:

$$\mathcal{P}_{\mathrm{LOOP}} = \{\ \underline{\texttt{in}}(X_1, \ldots, X_n); \underline{\texttt{var}}(X_{n+1}, \ldots, X_m); \alpha, \underline{\texttt{out}} X_1 \mid m \geq 1, \alpha \in \mathcal{A}, X_i \text{ Variable in } \alpha \curvearrowright 1 \leq i \leq m\}$$

Syntax von LOOP

LOOP-Programme:

$$\mathcal{P}_{\text{LOOP}} = \{\ \underline{\texttt{in}}(X_1, \ldots, X_n); \underline{\texttt{var}}(X_{n+1}, \ldots, X_m); \alpha, \underline{\texttt{out}} X_1 \mid m \geq 1, \alpha \in \mathcal{A}, X_i \text{ Variable in } \alpha \curvearrowright 1 \leq i \leq m\}$$

- LOOP-Programme haben normierte **Ein-/Ausgabe- und Programmvariablen**.
- Dabei ist die **leere Eingabe** ($n = 0$) erlaubt.
- Jede im Anweisungsblock verwendete Variable muss **deklariert** sein.

Syntax von LOOP

LOOP-Programme:

$$\mathcal{P}_{\text{LOOP}} = \{\ \underline{\texttt{in}}(X_1, \ldots, X_n); \underline{\texttt{var}}(X_{n+1}, \ldots, X_m); \alpha, \underline{\texttt{out}} X_1 \mid m \geq 1, \alpha \in \mathcal{A}, X_i \text{ Variable in } \alpha \curvearrowright 1 \leq i \leq m\}$$

- LOOP-Programme haben normierte **Ein-/Ausgabe- und Programmvariablen**.
- Dabei ist die **leere Eingabe** ($n = 0$) erlaubt.
- Jede im Anweisungsblock verwendete Variable muss **deklariert** sein.

Syntax von LOOP

LOOP-Programme:

$$\mathcal{P}_{\text{LOOP}} = \{\ \underline{\texttt{in}}(X_1, \ldots, X_n); \underline{\texttt{var}}(X_{n+1}, \ldots, X_m); \alpha, \underline{\texttt{out}} X_1 \mid m \geq 1, \alpha \in \mathcal{A}, X_i \text{ Variable in } \alpha \curvearrowright 1 \leq i \leq m\}$$

- LOOP-Programme haben normierte **Ein-/Ausgabe- und Programmvariablen**.
- Dabei ist die **leere Eingabe** ($n = 0$) erlaubt.
- Jede im Anweisungsblock verwendete Variable muss **deklariert** sein.

Syntax von LOOP

Beispiel:

$$
\begin{aligned}
P_0 = \quad & \underline{\texttt{in}}(X_1, X_2);\ \underline{\texttt{var}}(X_3); \\
& \underline{\texttt{loop}}\ X_1\ (\underline{\texttt{loop}}\ X_2\ (X_3 := X_3 + 1)); \\
& X_1 := 0; \\
& \underline{\texttt{loop}}\ X_3\ (X_1 := X_1 + 1); \\
& \underline{\texttt{out}}\ X_1
\end{aligned}
$$

Syntax von LOOP

Beispiel:

$$
\begin{aligned}
P_0 = \quad & \underline{\texttt{in}}(X_1, X_2);\ \underline{\texttt{var}}(X_3); \\
& \underline{\texttt{loop}}\ X_1\ (\underline{\texttt{loop}}\ X_2\ (X_3 := X_3 + 1)); \\
& X_1 := 0; \\
& \underline{\texttt{loop}}\ X_3\ (X_1 := X_1 + 1); \\
& \underline{\texttt{out}}\ X_1
\end{aligned}
$$

Das LOOP-Beispielprogramm berechnet die Funktion:

$$[\![P_0]\!] : \mathbb{N}^2 \to \mathbb{N}, \qquad (a_1, a_2) \mapsto a_1 * a_2.$$

Diese Funktion ist die **Semantik** des LOOP-Programms.

Semantik von LOOP

Im Gegensatz zu einer Turingmaschine berechnet ein **LOOP-Programm** nur **totale Funktionen**.

Definition (1):

Der Zielbereich der Semantik ist die Menge $Ops(\mathbb{N})$ der arithmetischen Operationen:

$$Ops(\mathbb{N}) := \{f : \mathbb{N}^k \to \mathbb{N} \mid k \geq 0\}$$

Als Sonderfall ist $k = 0$ zugelassen: $f : \begin{cases} \mathbb{N}^0 & \to & \mathbb{N} \\ () & \mapsto & a \end{cases}$

Solche nullstelligen Funktionen können mit **Konstanten** identifiziert werden.

Semantik von LOOP

Definition (2):

Die *Semantikfunktion* $[\![.]\!] : \mathcal{P}_{\mathrm{LOOP}} \to \mathit{Ops}(\mathbb{N})$ wird für

$$P = \underline{\texttt{in}}(X_1, \ldots, X_n); \underline{\texttt{var}}(X_{n+1}, \ldots, X_m); \alpha, \underline{\texttt{out}}\, X_1$$

definiert durch

$$[\![P]\!] := \underline{\texttt{out}}_1^{(m)} \circ [\![\alpha]\!]^{(m)} \circ \underline{\texttt{in}}_m^{(n)}$$

mit $[\![.]\!]^{(m)} : \mathcal{A} \to \{f \mid f : \mathbb{N}^m \to \mathbb{N}^m\}$. $\mathbb{N}^m$ ist der Zustandsraum, in dem Berechnungen ausgeführt werden.

Semantik von LOOP

Definition (3):

- $[\![.]\!] : \mathcal{P}_{\mathrm{LOOP}} \to Ops(\mathbb{N})$
- $[\![P]\!] := \underline{\mathtt{out}}_1^{(m)} \circ [\![\alpha]\!]^{(m)} \circ \underline{\mathtt{in}}_m^{(n)}$

Die Komponentenfunktionen der Semantik werden wie folgt definiert:

- *Eingabeabbildung*:
$$\underline{\mathtt{in}}_m^{(n)} : \begin{cases} \mathbb{N}^n & \to & \mathbb{N}^m \\ (a_1, \ldots, a_n) & \mapsto & (a_1, \ldots, a_n, 0, \ldots, 0) \end{cases}$$
- *Ausgabeabbildung*: $\underline{\mathtt{out}}_1^{(m)} : \begin{cases} \mathbb{N}^m & \to & \mathbb{N} \\ (a_1, \ldots, a_m) & \mapsto & a_1 \end{cases}$

Semantik von LOOP

Definition (3):

- $[\![.]\!] : \mathcal{P}_{\mathrm{LOOP}} \to Ops(\mathbb{N})$
- $[\![P]\!] := \underline{\mathtt{out}}_1^{(m)} \circ [\![\alpha]\!]^{(m)} \circ \underline{\mathtt{in}}_m^{(n)}$

Die Komponentenfunktionen der Semantik werden wie folgt definiert:

- *Eingabeabbildung*:
$$\underline{\mathtt{in}}_m^{(n)} : \begin{cases} \mathbb{N}^n & \to & \mathbb{N}^m \\ (a_1, \ldots, a_n) & \mapsto & (a_1, \ldots, a_n, 0, \ldots, 0) \end{cases}$$
- *Ausgabeabbildung*: $\underline{\mathtt{out}}_1^{(m)} : \begin{cases} \mathbb{N}^m & \to & \mathbb{N} \\ (a_1, \ldots, a_m) & \mapsto & a_1 \end{cases}$

Semantik von LOOP

Definition (4):

- $[\![.]\!] : \mathcal{P}_{\mathrm{LOOP}} \to Ops(\mathbb{N})$
- $[\![P]\!] := \underline{\mathtt{out}}_1^{(m)} \circ [\![\alpha]\!]^{(m)} \circ \underline{\mathtt{in}}_m^{(n)}$

Die Semantik der Anweisungen ist je eine *Zustandstransformation*

$$[\![\alpha]\!]^{(m)} : \mathbb{N}^m \to \mathbb{N}^m.$$

Diese wird **induktiv** über den Aufbau von $\mathcal{A}$ definiert. $\mathbb{N}^m$ heißt Zustandsraum.

1. $[\![X_i := X_j + 1]\!]^{(m)}(a_1, \ldots, a_m) := (a_1, \ldots, a_{i-1}, a_j + 1, a_{i+1}, \ldots, a_m)$
   $[\![X_i := 0]\!]^{(m)}(a_1, \ldots, a_m) := (a_1, \ldots, a_{i-1}, 0, a_{i+1}, \ldots, a_m)$
2. $[\![\alpha; \beta]\!]^{(m)} := [\![\beta]\!]^{(m)} \circ [\![\alpha]\!]^{(m)}$
3. $[\![\underline{\mathtt{loop}}\ X_i\ (\alpha)]\!]^{(m)}(a_1, \ldots, a_m) := ([\![\alpha]\!]^{(m)})^{a_i}(a_1, \ldots, a_m)$

Semantik von LOOP

Definition (4):

- $[\![.]\!] : \mathcal{P}_{\mathrm{LOOP}} \to Ops(\mathbb{N})$
- $[\![P]\!] := \underline{\mathtt{out}}_1^{(m)} \circ [\![\alpha]\!]^{(m)} \circ \underline{\mathtt{in}}_m^{(n)}$

Die Semantik der Anweisungen ist je eine *Zustandstransformation*

$$[\![\alpha]\!]^{(m)} : \mathbb{N}^m \to \mathbb{N}^m.$$

Diese wird **induktiv** über den Aufbau von $\mathcal{A}$ definiert. $\mathbb{N}^m$ heißt Zustandsraum.

1. $[\![X_i := X_j + 1]\!]^{(m)}(a_1, \ldots, a_m) := (a_1, \ldots, a_{i-1}, a_j + 1, a_{i+1}, \ldots, a_m)$
   $[\![X_i := 0]\!]^{(m)}(a_1, \ldots, a_m) := (a_1, \ldots, a_{i-1}, 0, a_{i+1}, \ldots, a_m)$
2. $[\![\alpha; \beta]\!]^{(m)} := [\![\beta]\!]^{(m)} \circ [\![\alpha]\!]^{(m)}$
3. $[\![\underline{\mathtt{loop}}\ X_i\ (\alpha)]\!]^{(m)}(a_1, \ldots, a_m) := ([\![\alpha]\!]^{(m)})^{a_i}(a_1, \ldots, a_m)$

Semantik von LOOP

Definition (4):

- $[\![.]\!] : \mathcal{P}_{\mathrm{LOOP}} \to Ops(\mathbb{N})$
- $[\![P]\!] := \underline{\mathtt{out}}_1^{(m)} \circ [\![\alpha]\!]^{(m)} \circ \underline{\mathtt{in}}_m^{(n)}$

Die Semantik der Anweisungen ist je eine *Zustandstransformation*

$$[\![\alpha]\!]^{(m)} : \mathbb{N}^m \to \mathbb{N}^m.$$

Diese wird **induktiv** über den Aufbau von $\mathcal{A}$ definiert. $\mathbb{N}^m$ heißt Zustandsraum.

1. $[\![X_i := X_j + 1]\!]^{(m)}(a_1, \ldots, a_m) := (a_1, \ldots, a_{i-1}, a_j + 1, a_{i+1}, \ldots, a_m)$
   $[\![X_i := 0]\!]^{(m)}(a_1, \ldots, a_m) := (a_1, \ldots, a_{i-1}, 0, a_{i+1}, \ldots, a_m)$
2. $[\![\alpha; \beta]\!]^{(m)} := [\![\beta]\!]^{(m)} \circ [\![\alpha]\!]^{(m)}$
3. $[\![\underline{\mathtt{loop}}\ X_i\ (\alpha)]\!]^{(m)}(a_1, \ldots, a_m) := ([\![\alpha]\!]^{(m)})^{a_i}(a_1, \ldots, a_m)$

Semantik von LOOP

Definition (kurz):

- $[\![.]\!] : \mathcal{P}_{\mathrm{LOOP}} \to Ops(\mathbb{N})$
- $[\![P]\!] := \underline{\mathtt{out}}_1^{(m)} \circ [\![\alpha]\!]^{(m)} \circ \underline{\mathtt{in}}_m^{(n)}$
- $\underline{\mathtt{in}}_m^{(n)} : \left\{ \begin{array}{rcl} \mathbb{N}^n & \to & \mathbb{N}^m \\ (a_1, \ldots, a_n) & \mapsto & (a_1, \ldots, a_n, 0, \ldots, 0) \end{array} \right.$
- $\underline{\mathtt{out}}_1^{(m)} : \left\{ \begin{array}{rcl} \mathbb{N}^m & \to & \mathbb{N} \\ (a_1, \ldots, a_m) & \mapsto & a_1 \end{array} \right.$
- $[\![\alpha]\!]^{(m)} : \mathbb{N}^m \to \mathbb{N}^m$.
  - $[\![X_i := X_j + 1]\!]^{(m)}(a_1, \ldots, a_m) := (a_1, \ldots, a_{i-1}, a_j + 1, a_{i+1}, \ldots, a_m)$
    $[\![X_i := 0]\!]^{(m)}(a_1, \ldots, a_m) := (a_1, \ldots, a_{i-1}, 0, a_{i+1}, \ldots, a_m)$
  - $[\![\alpha; \beta]\!]^{(m)} := [\![\beta]\!]^{(m)} \circ [\![\alpha]\!]^{(m)}$
  - $[\![\underline{\mathtt{loop}}\ X_i\ (\alpha)]\!]^{(m)}(a_1, \ldots, a_m) := ([\![\alpha]\!]^{(m)})^{a_i}(a_1, \ldots, a_m)$

Semantik von LOOP

Beispiel: Semantik des Beispielprogramms P_0 (1)

Wir setzen

$$\begin{aligned} \alpha_1 &:= \underline{\texttt{loop}}\ X_1(\underline{\texttt{loop}}\ X_2(X_3 := X_3 + 1)) \\ \alpha_2 &:= X_1 := 0 \\ \alpha_3 &:= \underline{\texttt{loop}}\ X_3(X_1 := X_1 + 1) \end{aligned}$$

und berechnen

$$\begin{aligned} [\![\alpha_1]\!]^{(3)}(a_1, a_2, a_3) &= [\![\underline{\texttt{loop}}\ X_1(\underline{\texttt{loop}}\ X_2(X_3 := X_3 + 1))]\!]^{(3)}(a_1, a_2, a_3) \\ &= ([\![\underline{\texttt{loop}}\ X_2(X_3 := X_3 + 1)]\!]^{(3)})^{a_1}(a_1, a_2, a_3) \\ &= ([\![X_3 := X_3 + 1]\!]^{(3)})^{a_1 \cdot a_2}(a_1, a_2, a_3) \\ &= (a_1, a_2, a_3 + a_1 \cdot a_2) \end{aligned}$$

Semantik von LOOP

Beispiel: Semantik des Beispielprogramms P_0 (1)

Wir setzen

$$\begin{aligned} \alpha_1 &:= \underline{\texttt{loop}}\ X_1(\underline{\texttt{loop}}\ X_2(X_3 := X_3 + 1)) \\ \alpha_2 &:= X_1 := 0 \\ \alpha_3 &:= \underline{\texttt{loop}}\ X_3(X_1 := X_1 + 1) \end{aligned}$$

und berechnen

$$\begin{aligned} [\![\alpha_1]\!]^{(3)}(a_1, a_2, a_3) &= [\![\underline{\texttt{loop}}\ X_1(\underline{\texttt{loop}}\ X_2(X_3 := X_3 + 1))]\!]^{(3)}(a_1, a_2, a_3) \\ &= ([\![\underline{\texttt{loop}}\ X_2(X_3 := X_3 + 1)]\!]^{(3)})^{a_1}(a_1, a_2, a_3) \\ &= ([\![X_3 := X_3 + 1]\!]^{(3)})^{a_1 \cdot a_2}(a_1, a_2, a_3) \\ &= (a_1, a_2, a_3 + a_1 \cdot a_2) \end{aligned}$$

Semantik von LOOP

Beispiel: Semantik des Beispielprogramms P_0 (1)

Wir setzen

$$\begin{aligned} \alpha_1 &:= \underline{\texttt{loop}}\ X_1(\underline{\texttt{loop}}\ X_2(X_3 := X_3 + 1)) \\ \alpha_2 &:= X_1 := 0 \\ \alpha_3 &:= \underline{\texttt{loop}}\ X_3(X_1 := X_1 + 1) \end{aligned}$$

und berechnen

$$\begin{aligned} [\![\alpha_1]\!]^{(3)}(a_1, a_2, a_3) &= [\![\underline{\texttt{loop}}\ X_1(\underline{\texttt{loop}}\ X_2(X_3 := X_3 + 1))]\!]^{(3)}(a_1, a_2, a_3) \\ &= ([\![\underline{\texttt{loop}}\ X_2(X_3 := X_3 + 1)]\!]^{(3)})^{a_1}(a_1, a_2, a_3) \\ &= ([\![X_3 := X_3 + 1]\!]^{(3)})^{a_1 \cdot a_2}(a_1, a_2, a_3) \\ &= (a_1, a_2, a_3 + a_1 \cdot a_2) \end{aligned}$$

Semantik von LOOP

Beispiel: Semantik des Beispielprogramms P_0 (1)

Wir setzen

$$\begin{aligned} \alpha_1 &:= \underline{\texttt{loop}}\ X_1(\underline{\texttt{loop}}\ X_2(X_3 := X_3 + 1)) \\ \alpha_2 &:= X_1 := 0 \\ \alpha_3 &:= \underline{\texttt{loop}}\ X_3(X_1 := X_1 + 1) \end{aligned}$$

und berechnen

$$\begin{aligned} [\![\alpha_1]\!]^{(3)}(a_1,a_2,a_3) &= [\![\underline{\texttt{loop}}\ X_1(\underline{\texttt{loop}}\ X_2(X_3 := X_3 + 1))]\!]^{(3)}(a_1,a_2,a_3) \\ &= ([\![\underline{\texttt{loop}}\ X_2(X_3 := X_3 + 1)]\!]^{(3)})^{a_1}(a_1,a_2,a_3) \\ &= ([\![X_3 := X_3 + 1]\!]^{(3)})^{a_1 \cdot a_2}(a_1,a_2,a_3) \\ &= (a_1,a_2,a_3 + a_1 \cdot a_2) \\ [\![\alpha_2]\!]^{(3)}(a_1,a_2,a_3) &= [\![X_1 := 0]\!]^{(3)}(a_1,a_2,a_3) \\ &= (0,a_2,a_3) \end{aligned}$$

Semantik von LOOP

Beispiel: Semantik des Beispielprogramms P_0 (1)

Wir setzen

$$\begin{aligned}\alpha_1 &:= \underline{\texttt{loop}}\ X_1(\underline{\texttt{loop}}\ X_2(X_3 := X_3 + 1))\\ \alpha_2 &:= X_1 := 0\\ \alpha_3 &:= \underline{\texttt{loop}}\ X_3(X_1 := X_1 + 1)\end{aligned}$$

und berechnen

$$\begin{aligned}[\![\alpha_1]\!]^{(3)}(a_1,a_2,a_3) &= [\![\underline{\texttt{loop}}\ X_1(\underline{\texttt{loop}}\ X_2(X_3 := X_3 + 1))]\!]^{(3)}(a_1,a_2,a_3)\\ &= ([\![\underline{\texttt{loop}}\ X_2(X_3 := X_3 + 1)]\!]^{(3)})^{a_1}(a_1,a_2,a_3)\\ &= ([\![X_3 := X_3 + 1]\!]^{(3)})^{a_1\cdot a_2}(a_1,a_2,a_3)\\ &= (a_1,a_2,a_3 + a_1\cdot a_2)\\ [\![\alpha_2]\!]^{(3)}(a_1,a_2,a_3) &= [\![X_1 := 0]\!]^{(3)}(a_1,a_2,a_3)\\ &= (0,a_2,a_3)\\ [\![\alpha_3]\!]^{(3)}(a_1,a_2,a_3) &= [\![\underline{\texttt{loop}}\ X_3(X_1 := X_1 + 1)]\!]^{(3)}(a_1,a_2,a_3)\\ &= ([\![X_1 := X_1 + 1]\!]^{(3)})^{a_3}(a_1,a_2,a_3)\\ &= (a_1 + a_3,a_2,a_3)\end{aligned}$$

Semantik von LOOP

Beispiel: Semantik des Beispielprogramms P_0 (1)

Wir setzen

$$\begin{aligned}\alpha_1 &:= \underline{\texttt{loop}}\ X_1(\underline{\texttt{loop}}\ X_2(X_3 := X_3 + 1))\\ \alpha_2 &:= X_1 := 0\\ \alpha_3 &:= \underline{\texttt{loop}}\ X_3(X_1 := X_1 + 1)\end{aligned}$$

und berechnen

$$\begin{aligned}[\![\alpha_1]\!]^{(3)}(a_1,a_2,a_3) &= [\![\underline{\texttt{loop}}\ X_1(\underline{\texttt{loop}}\ X_2(X_3 := X_3 + 1))]\!]^{(3)}(a_1,a_2,a_3)\\ &= ([\![\underline{\texttt{loop}}\ X_2(X_3 := X_3 + 1)]\!]^{(3)})^{a_1}(a_1,a_2,a_3)\\ &= ([\![X_3 := X_3 + 1]\!]^{(3)})^{a_1\cdot a_2}(a_1,a_2,a_3)\\ &= (a_1,a_2,a_3 + a_1\cdot a_2)\\ [\![\alpha_2]\!]^{(3)}(a_1,a_2,a_3) &= [\![X_1 := 0]\!]^{(3)}(a_1,a_2,a_3)\\ &= (0,a_2,a_3)\\ [\![\alpha_3]\!]^{(3)}(a_1,a_2,a_3) &= [\![\underline{\texttt{loop}}\ X_3(X_1 := X_1 + 1)]\!]^{(3)}(a_1,a_2,a_3)\\ &= ([\![X_1 := X_1 + 1]\!]^{(3)})^{a_3}(a_1,a_2,a_3)\\ &= (a_1 + a_3,a_2,a_3)\end{aligned}$$

Semantik von LOOP

Beispiel: Semantik des Beispielprogramms P_0 (2)

$$\begin{aligned}
[\![\alpha_1]\!]^{(3)}(a_1, a_2, a_3) &= (a_1, a_2, a_3 + a_1 \cdot a_2) \\
[\![\alpha_2]\!]^{(3)}(a_1, a_2, a_3) &= (0, a_2, a_3) \\
[\![\alpha_3]\!]^{(3)}(a_1, a_2, a_3) &= (a_1 + a_3, a_2, a_3)
\end{aligned}$$

Hiermit ergibt sich nun

$$\begin{aligned}
[\![P_0]\!](a_1, a_2) &= (\underline{\mathtt{out}}_1^{(3)} \circ [\![\alpha_3]\!]^{(3)} \circ [\![\alpha_2]\!]^{(3)} \circ [\![\alpha_1]\!]^{(3)} \circ \underline{\mathtt{in}}_3^{(2)})(a_1, a_2) \\
&= (\underline{\mathtt{out}}_1^{(3)} \circ [\![\alpha_3]\!]^{(3)} \circ [\![\alpha_2]\!]^{(3)} \circ [\![\alpha_1]\!]^{(3)})(a_1, a_2, 0) \\
&= (\underline{\mathtt{out}}_1^{(3)} \circ [\![\alpha_3]\!]^{(3)} \circ [\![\alpha_2]\!]^{(3)})(a_1, a_2, a_1 \cdot a_2) \\
&= (\underline{\mathtt{out}}_1^{(3)} \circ [\![\alpha_3]\!]^{(3)})(0, a_2, a_1 \cdot a_2) \\
&= \underline{\mathtt{out}}_1^{(3)}(a_1 \cdot a_2, a_2, a_1, \cdot a_2) = a_1 \cdot a_2.
\end{aligned}$$

Semantik von LOOP

Beispiel: Semantik des Beispielprogramms P_0 (2)

$$\begin{aligned}
[\![\alpha_1]\!]^{(3)}(a_1, a_2, a_3) &= (a_1, a_2, a_3 + a_1 \cdot a_2) \\
[\![\alpha_2]\!]^{(3)}(a_1, a_2, a_3) &= (0, a_2, a_3) \\
[\![\alpha_3]\!]^{(3)}(a_1, a_2, a_3) &= (a_1 + a_3, a_2, a_3)
\end{aligned}$$

Hiermit ergibt sich nun

$$\begin{aligned}
[\![P_0]\!](a_1, a_2) &= (\underline{\mathtt{out}}_1^{(3)} \circ [\![\alpha_3]\!]^{(3)} \circ [\![\alpha_2]\!]^{(3)} \circ [\![\alpha_1]\!]^{(3)} \circ \underline{\mathtt{in}}_3^{(2)})(a_1, a_2) \\
&= (\underline{\mathtt{out}}_1^{(3)} \circ [\![\alpha_3]\!]^{(3)} \circ [\![\alpha_2]\!]^{(3)} \circ [\![\alpha_1]\!]^{(3)})(a_1, a_2, 0) \\
&= (\underline{\mathtt{out}}_1^{(3)} \circ [\![\alpha_3]\!]^{(3)} \circ [\![\alpha_2]\!]^{(3)})(a_1, a_2, a_1 \cdot a_2) \\
&= (\underline{\mathtt{out}}_1^{(3)} \circ [\![\alpha_3]\!]^{(3)})(0, a_2, a_1 \cdot a_2) \\
&= \underline{\mathtt{out}}_1^{(3)}(a_1 \cdot a_2, a_2, a_1, \cdot a_2) = a_1 \cdot a_2.
\end{aligned}$$

Semantik von LOOP

Beispiel: Semantik des Beispielprogramms P_0 (2)

$$\begin{aligned}
[\![\alpha_1]\!]^{(3)}(a_1, a_2, a_3) &= (a_1, a_2, a_3 + a_1 \cdot a_2) \\
[\![\alpha_2]\!]^{(3)}(a_1, a_2, a_3) &= (0, a_2, a_3) \\
[\![\alpha_3]\!]^{(3)}(a_1, a_2, a_3) &= (a_1 + a_3, a_2, a_3)
\end{aligned}$$

Hiermit ergibt sich nun

$$\begin{aligned}
[\![P_0]\!](a_1, a_2) &= (\underline{\mathtt{out}}_1^{(3)} \circ [\![\alpha_3]\!]^{(3)} \circ [\![\alpha_2]\!]^{(3)} \circ [\![\alpha_1]\!]^{(3)} \circ \underline{\mathtt{in}}_3^{(2)})(a_1, a_2) \\
&= (\underline{\mathtt{out}}_1^{(3)} \circ [\![\alpha_3]\!]^{(3)} \circ [\![\alpha_2]\!]^{(3)} \circ [\![\alpha_1]\!]^{(3)})(a_1, a_2, 0) \\
&= (\underline{\mathtt{out}}_1^{(3)} \circ [\![\alpha_3]\!]^{(3)} \circ [\![\alpha_2]\!]^{(3)})(a_1, a_2, a_1 \cdot a_2) \\
&= (\underline{\mathtt{out}}_1^{(3)} \circ [\![\alpha_3]\!]^{(3)})(0, a_2, a_1 \cdot a_2) \\
&= \underline{\mathtt{out}}_1^{(3)}(a_1 \cdot a_2, a_2, a_1, \cdot a_2) = a_1 \cdot a_2.
\end{aligned}$$

Semantik von LOOP

Beispiel: Semantik des Beispielprogramms P_0 (2)

$$\begin{aligned}
[\![\alpha_1]\!]^{(3)}(a_1, a_2, a_3) &= (a_1, a_2, a_3 + a_1 \cdot a_2) \\
[\![\alpha_2]\!]^{(3)}(a_1, a_2, a_3) &= (0, a_2, a_3) \\
[\![\alpha_3]\!]^{(3)}(a_1, a_2, a_3) &= (a_1 + a_3, a_2, a_3)
\end{aligned}$$

Hiermit ergibt sich nun

$$\begin{aligned}
[\![P_0]\!](a_1, a_2) &= (\underline{\mathtt{out}}_1^{(3)} \circ [\![\alpha_3]\!]^{(3)} \circ [\![\alpha_2]\!]^{(3)} \circ [\![\alpha_1]\!]^{(3)} \circ \underline{\mathtt{in}}_3^{(2)})(a_1, a_2) \\
&= (\underline{\mathtt{out}}_1^{(3)} \circ [\![\alpha_3]\!]^{(3)} \circ [\![\alpha_2]\!]^{(3)} \circ [\![\alpha_1]\!]^{(3)})(a_1, a_2, 0) \\
&= (\underline{\mathtt{out}}_1^{(3)} \circ [\![\alpha_3]\!]^{(3)} \circ [\![\alpha_2]\!]^{(3)})(a_1, a_2, a_1 \cdot a_2) \\
&= (\underline{\mathtt{out}}_1^{(3)} \circ [\![\alpha_3]\!]^{(3)})(0, a_2, a_1 \cdot a_2) \\
&= \underline{\mathtt{out}}_1^{(3)}(a_1 \cdot a_2, a_2, a_1, \cdot a_2) = a_1 \cdot a_2.
\end{aligned}$$

Semantik von LOOP

Beispiel: Semantik des Beispielprogramms P_0 (2)

$$\begin{aligned}
[\![\alpha_1]\!]^{(3)}(a_1, a_2, a_3) &= (a_1, a_2, a_3 + a_1 \cdot a_2) \\
[\![\alpha_2]\!]^{(3)}(a_1, a_2, a_3) &= (0, a_2, a_3) \\
[\![\alpha_3]\!]^{(3)}(a_1, a_2, a_3) &= (a_1 + a_3, a_2, a_3)
\end{aligned}$$

Hiermit ergibt sich nun

$$\begin{aligned}
[\![P_0]\!](a_1, a_2) &= (\underline{\mathtt{out}}_1^{(3)} \circ [\![\alpha_3]\!]^{(3)} \circ [\![\alpha_2]\!]^{(3)} \circ [\![\alpha_1]\!]^{(3)} \circ \underline{\mathtt{in}}_3^{(2)})(a_1, a_2) \\
&= (\underline{\mathtt{out}}_1^{(3)} \circ [\![\alpha_3]\!]^{(3)} \circ [\![\alpha_2]\!]^{(3)} \circ [\![\alpha_1]\!]^{(3)})(a_1, a_2, 0) \\
&= (\underline{\mathtt{out}}_1^{(3)} \circ [\![\alpha_3]\!]^{(3)} \circ [\![\alpha_2]\!]^{(3)})(a_1, a_2, a_1 \cdot a_2) \\
&= (\underline{\mathtt{out}}_1^{(3)} \circ [\![\alpha_3]\!]^{(3)})(0, a_2, a_1 \cdot a_2) \\
&= \underline{\mathtt{out}}_1^{(3)}(a_1 \cdot a_2, a_2, a_1, \cdot a_2) = a_1 \cdot a_2.
\end{aligned}$$

Semantik von LOOP

Beispiel: Semantik des Beispielprogramms P_0 (2)

$$\begin{aligned}
[\![\alpha_1]\!]^{(3)}(a_1, a_2, a_3) &= (a_1, a_2, a_3 + a_1 \cdot a_2) \\
[\![\alpha_2]\!]^{(3)}(a_1, a_2, a_3) &= (0, a_2, a_3) \\
[\![\alpha_3]\!]^{(3)}(a_1, a_2, a_3) &= (a_1 + a_3, a_2, a_3)
\end{aligned}$$

Hiermit ergibt sich nun

$$\begin{aligned}
[\![P_0]\!](a_1, a_2) &= (\underline{\mathtt{out}}_1^{(3)} \circ [\![\alpha_3]\!]^{(3)} \circ [\![\alpha_2]\!]^{(3)} \circ [\![\alpha_1]\!]^{(3)} \circ \underline{\mathtt{in}}_3^{(2)})(a_1, a_2) \\
&= (\underline{\mathtt{out}}_1^{(3)} \circ [\![\alpha_3]\!]^{(3)} \circ [\![\alpha_2]\!]^{(3)} \circ [\![\alpha_1]\!]^{(3)})(a_1, a_2, 0) \\
&= (\underline{\mathtt{out}}_1^{(3)} \circ [\![\alpha_3]\!]^{(3)} \circ [\![\alpha_2]\!]^{(3)})(a_1, a_2, a_1 \cdot a_2) \\
&= (\underline{\mathtt{out}}_1^{(3)} \circ [\![\alpha_3]\!]^{(3)})(0, a_2, a_1 \cdot a_2) \\
&= \underline{\mathtt{out}}_1^{(3)}(a_1 \cdot a_2, a_2, a_1, \cdot a_2) = a_1 \cdot a_2.
\end{aligned}$$

Semantik von LOOP

Beispiel: Semantik des Beispielprogramms P_0 (2)

$$\begin{aligned}
[\![\alpha_1]\!]^{(3)}(a_1, a_2, a_3) &= (a_1, a_2, a_3 + a_1 \cdot a_2) \\
[\![\alpha_2]\!]^{(3)}(a_1, a_2, a_3) &= (0, a_2, a_3) \\
[\![\alpha_3]\!]^{(3)}(a_1, a_2, a_3) &= (a_1 + a_3, a_2, a_3)
\end{aligned}$$

Hiermit ergibt sich nun

$$\begin{aligned}
[\![P_0]\!](a_1, a_2) &= (\underline{\mathtt{out}}_1^{(3)} \circ [\![\alpha_3]\!]^{(3)} \circ [\![\alpha_2]\!]^{(3)} \circ [\![\alpha_1]\!]^{(3)} \circ \underline{\mathtt{in}}_3^{(2)})(a_1, a_2) \\
&= (\underline{\mathtt{out}}_1^{(3)} \circ [\![\alpha_3]\!]^{(3)} \circ [\![\alpha_2]\!]^{(3)} \circ [\![\alpha_1]\!]^{(3)})(a_1, a_2, 0) \\
&= (\underline{\mathtt{out}}_1^{(3)} \circ [\![\alpha_3]\!]^{(3)} \circ [\![\alpha_2]\!]^{(3)})(a_1, a_2, a_1 \cdot a_2) \\
&= (\underline{\mathtt{out}}_1^{(3)} \circ [\![\alpha_3]\!]^{(3)})(0, a_2, a_1 \cdot a_2) \\
&= \underline{\mathtt{out}}_1^{(3)}(a_1 \cdot a_2, a_2, a_1, \cdot a_2) = a_1 \cdot a_2.
\end{aligned}$$

Das LOOP-Programm P_0 berechnet demnach die Funktion

$$[\![P_0]\!] : \mathbb{N}^2 \to \mathbb{N},\ (a_1, a_2) \mapsto a_1 \cdot a_2.$$

LOOP-Berechenbarkeit

Definition:

Eine Funktion $f : \mathbb{N}^n \to \mathbb{N}$ heißt **LOOP-berechenbar**, falls ein $P \in \mathcal{P}_{LOOP}$ existiert mit $[\![P]\!] = f$.

LOOP-Berechenbarkeit

Definition:

Eine Funktion $f : \mathbb{N}^n \to \mathbb{N}$ heißt **LOOP-berechenbar**, falls ein $P \in \mathcal{P}_{LOOP}$ existiert mit $[\![P]\!] = f$.

Die **primitiv rekursiven** Funktionen sind eine induktive Charakterisierung der LOOP-berechenbaren Funktionen.

Primitiv rekursive Funktionen

Definition:

Eine Funktion $f : \mathbb{N}^n \to \mathbb{N}$ heißt **primitiv rekursiv**, falls f entweder eine **primitiv rekursive Grundfunktion** ist oder sich aus diesen in endlich vielen Schritten durch **Komposition** und/oder **primitive Rekursion** erzeugen lässt.

Primitiv rekursive Funktionen

Definition:

Eine Funktion $f : \mathbb{N}^n \to \mathbb{N}$ heißt **primitiv rekursiv**, falls f entweder eine **primitiv rekursive Grundfunktion** ist oder sich aus diesen in endlich vielen Schritten durch **Komposition** und/oder **primitive Rekursion** erzeugen lässt.

Primitiv rekursiven Grundfunktionen:

- *Nachfolgerfunktion:* $S : \begin{cases} \mathbb{N} & \to & \mathbb{N} \\ a & \mapsto & a+1 \end{cases}$
- *Nullkonstante:* $0 : \begin{cases} \mathbb{N}^0 & \to & \mathbb{N} \\ () & \mapsto & 0 \end{cases}$
- *Projektionen:* $pr_i^{(n)} : \begin{cases} \mathbb{N}^n & \to & \mathbb{N} \\ (a_1, \ldots, a_n) & \mapsto & a_i \end{cases}$

Primitiv rekursive Funktionen

Definition:

Eine Funktion $f : \mathbb{N}^n \to \mathbb{N}$ heißt **primitiv rekursiv**, falls f entweder eine **primitiv rekursive Grundfunktion** ist oder sich aus diesen in endlich vielen Schritten durch **Komposition** und/oder **primitive Rekursion** erzeugen lässt.

Primitiv rekursiven Grundfunktionen:

- *Nachfolgerfunktion:* $S : \begin{cases} \mathbb{N} & \to & \mathbb{N} \\ a & \mapsto & a+1 \end{cases}$
- *Nullkonstante:* $0 : \begin{cases} \mathbb{N}^0 & \to & \mathbb{N} \\ () & \mapsto & 0 \end{cases}$
- *Projektionen:* $pr_i^{(n)} : \begin{cases} \mathbb{N}^n & \to & \mathbb{N} \\ (a_1, \ldots, a_n) & \mapsto & a_i \end{cases}$

Primitiv rekursive Funktionen

Definition:

Eine Funktion $f : \mathbb{N}^n \to \mathbb{N}$ heißt **primitiv rekursiv**, falls f entweder eine **primitiv rekursive Grundfunktion** ist oder sich aus diesen in endlich vielen Schritten durch **Komposition** und/oder **primitive Rekursion** erzeugen lässt.

Primitiv rekursiven Grundfunktionen:

- *Nachfolgerfunktion:* $S : \begin{cases} \mathbb{N} & \to & \mathbb{N} \\ a & \mapsto & a+1 \end{cases}$
- *Nullkonstante:* $0 : \begin{cases} \mathbb{N}^0 & \to & \mathbb{N} \\ () & \mapsto & 0 \end{cases}$
- *Projektionen:* $pr_i^{(n)} : \begin{cases} \mathbb{N}^n & \to & \mathbb{N} \\ (a_1, \ldots, a_n) & \mapsto & a_i \end{cases}$

Primitiv rekursive Funktionen

Komposition:

Für $f : \mathbb{N}^n \to \mathbb{N}$ und $f_1, \ldots, f_n : \mathbb{N}^k \to \mathbb{N}$ definiert man die *Komposition* durch

$$f \circ [f_1, \ldots, f_n] : \left\{ \begin{array}{ccc} \mathbb{N}^k & \to & \mathbb{N} \\ (a_1, \ldots, a_k) & \mapsto & f(f_1(a_1, \ldots, a_k), \ldots, f_n(a_1, \ldots, a_k)) \end{array} \right.$$

Primitiv rekursive Funktionen

Komposition:

Für $f : \mathbb{N}^n \to \mathbb{N}$ und $f_1, \ldots, f_n : \mathbb{N}^k \to \mathbb{N}$ definiert man die *Komposition* durch

$$f \circ [f_1, \ldots, f_n] : \left\{ \begin{array}{rcl} \mathbb{N}^k & \to & \mathbb{N} \\ (a_1, \ldots, a_k) & \mapsto & f(f_1(a_1, \ldots, a_k), \ldots, f_n(a_1, \ldots, a_k)) \end{array} \right.$$

Schema der primitiven Rekursion:

Für $f : \mathbb{N}^n \to \mathbb{N}$ und $g : \mathbb{N}^{n+2} \to \mathbb{N}$ heißt $h : \mathbb{N}^{n+1} \to \mathbb{N}$ mit

$$\begin{array}{rcl} h(a_1, \ldots, a_n, 0) & := & f(a_1, \ldots, a_n) \\ h(a_1, \ldots, a_n, a+1) & := & g(a_1, \ldots, a_n, a, h(a_1, \ldots, a_n, a)) \end{array}$$

durch primitive Rekursion aus f und g erzeugt.

Primitiv rekursive Funktionen

- *Nachfolgerfunktion:* $S : \left\{ \begin{array}{lcl} \mathbb{N} & \to & \mathbb{N} \\ a & \mapsto & a+1 \end{array} \right.$
- *Projektionen:* $pr_i^{(n)} : \left\{ \begin{array}{lcl} \mathbb{N}^n & \to & \mathbb{N} \\ (a_1, \ldots, a_n) & \mapsto & a_i \end{array} \right.$
- *Komposition:* $f \circ [f_1, \ldots, f_n] : \left\{ \begin{array}{lcl} \mathbb{N}^k & \to & \mathbb{N} \\ (a_1, \ldots, a_k) & \mapsto & f(f_1(a_1, \ldots, a_k), \ldots, f_n(a_1, \ldots, a_k)) \end{array} \right.$
- *Schema:* $\begin{array}{lcl} h(a_1, \ldots, a_n, 0) & := & f(a_1, \ldots, a_n) \\ h(a_1, \ldots, a_n, a+1) & := & g(a_1, \ldots, a_n, a, h(a_1, \ldots, a_n, a)) \end{array}$

Beispiel: $add : \mathbb{N}^2 \to \mathbb{N},\ (a,b) \mapsto a+b$ ist primitiv rekursiv

Es gilt:

$$\begin{array}{lcl} add(a,0) & = & a \\ add(a,b+1) & = & (a+b)+1 = S \circ pr_3^{(3)}(a,b,add(a,b)) \end{array}$$

Entspricht dem Schema mit $f = pr_1^{(1)}$ und $g := S \circ pr_3^{(3)}$.
Damit ist *add* primitiv rekursiv.

Primitiv rekursive Funktionen

- *Nullkonstante:* $0 : \left\{ \begin{array}{lcl} \mathbb{N}^0 & \to & \mathbb{N} \\ () & \mapsto & 0 \end{array} \right.$
- *Projektionen:* $pr_i^{(n)} : \left\{ \begin{array}{lcl} \mathbb{N}^n & \to & \mathbb{N} \\ (a_1, \ldots, a_n) & \mapsto & a_i \end{array} \right.$
- *Schema:* $\begin{array}{lcl} h(a_1, \ldots, a_n, 0) & := & f(a_1, \ldots, a_n) \\ h(a_1, \ldots, a_n, a+1) & := & g(a_1, \ldots, a_n, a, h(a_1, \ldots, a_n, a)) \end{array}$

Beispiel: $C_0^{(1)} : \mathbb{N} \to \mathbb{N},\ (a) \mapsto 0$ ist primitiv rekursiv

Es gilt

$$\begin{array}{lcl} C_0^{(1)}(0) & = & 0() = 0 \\ C_0^{(1)}(x+1) & = & C_0^{(1)}(x) = pr_2^{(2)}(x, C_0^{(1)}(x)) \end{array}$$

Entspricht dem Schema mit $f = 0$ und $g := pr_2^{(2)}$.

Damit ist $C_0^{(1)}$ primitiv rekursiv.

Primitiv rekursive Funktionen

- *Projektionen:* $pr_i^{(n)} : \left\{ \begin{array}{lcl} \mathbb{N}^n & \to & \mathbb{N} \\ (a_1, \ldots, a_n) & \mapsto & a_i \end{array} \right.$
- *Komposition:* $f \circ [f_1, \ldots, f_n] : \left\{ \begin{array}{lcl} \mathbb{N}^k & \to & \mathbb{N} \\ (a_1, \ldots, a_k) & \mapsto & f(f_1(a_1, \ldots, a_k), \ldots, f_n(a_1, \ldots, a_k)) \end{array} \right.$
- *Schema:* $\begin{array}{lcl} h(a_1, \ldots, a_n, 0) & := & f(a_1, \ldots, a_n) \\ h(a_1, \ldots, a_n, a+1) & := & g(a_1, \ldots, a_n, a, h(a_1, \ldots, a_n, a)) \end{array}$

Beispiel: $mult : \mathbb{N}^2 \to \mathbb{N},\ (a, b) \mapsto ab$ ist primitiv rekursiv

Für die Funktion *mult* gilt:

$$\begin{array}{lcl} mult(a, 0) & = & 0 = C_0^{(1)}(a) \\ mult(a, b+1) & = & mult(a, b) + a \\ & = & add \circ [pr_3^{(3)}, pr_1^{(3)}](a, b, mult(a, b)) \end{array}$$

Entspricht dem Schema mit $f = C_0^{(1)}$ und $g = add \circ [pr_3^{(3)}, pr_1^{(3)}]$.
Somit ist auch *mult* primitiv rekursiv.

Primitiv rekursiv und LOOP berechenbar

Satz:

Eine Funktion f ist **genau dann** primitiv rekursiv, wenn sie LOOP-berechenbar ist.

Primitiv rekursiv und LOOP berechenbar

Satz:

Eine Funktion f ist **genau dann** primitiv rekursiv, wenn sie LOOP-berechenbar ist.

Beweisidee: Im Wesentlichen wird gezeigt, dass die folgenden Komponenten korrespondieren:

LOOP	prim.rekursiv
Wertzuweisung	prim. Grundfunktionen
$X_i := 0$	Nullkonstante
$X_i := X_j + 1$	Nachfolgerfunktion & Projektion
Sequenz von Anw.	Komposition
Loop-Schleife	prim. Rekursion

f primitiv rekursiv $\curvearrowright$ f LOOP berechenbar

Beweisskizze $\curvearrowright$ (1):

Induktiv über die Definition der primitiv rekursiven Funktionen.

- Der Nachweis ist Trivial für die **Grundfunktionen**.
- **Komposition**: Zeige: $f, f_1, \ldots, f_n$ LOOP berechenbar $\curvearrowright$ $f \circ [f_1, \ldots, f_n]$ LOOP berechenbar.

f primitiv rekursiv $\curvearrowright$ f LOOP berechenbar

Beweisskizze $\curvearrowright$ (1):

Induktiv über die Definition der primitiv rekursiven Funktionen.

- Der Nachweis ist Trivial für die **Grundfunktionen**.
- **Komposition**: Zeige: f, $f_1, \ldots, f_n$ LOOP berechenbar $\curvearrowright$ $f \circ [f_1, \ldots, f_n]$ LOOP berechenbar.

f primitiv rekursiv $\curvearrowright$ f LOOP berechenbar

Beweisskizze $\curvearrowright$ (1):

Induktiv über die Definition der primitiv rekursiven Funktionen.

- Der Nachweis ist Trivial für die **Grundfunktionen**.
- **Komposition**: Zeige: f, $f_1, \ldots, f_n$ LOOP berechenbar $\curvearrowright$ $f \circ [f_1, \ldots, f_n]$ LOOP berechenbar.
  Hinweis: **Kopieranweisungen** der Gestalt $X_i := X_j$ sind in LOOP simulierbar.

  $\rightarrow$ Sichern der Eingabe/Zwischenresultate.

f primitiv rekursiv $\curvearrowright$ f LOOP berechenbar

Beweisskizze $\curvearrowright$ (1):

Induktiv über die Definition der primitiv rekursiven Funktionen.

- Der Nachweis ist Trivial für die **Grundfunktionen**.
- **Komposition**: Zeige: f, $f_1, \ldots, f_n$ LOOP berechenbar $\curvearrowright$ $f \circ [f_1, \ldots, f_n]$ LOOP berechenbar.
  Hinweis: **Kopieranweisungen** der Gestalt $X_i := X_j$ sind in LOOP simulierbar.

  $\rightarrow$ Sichern der Eingabe/Zwischenresultate.

  Korrektheitsnachweis mit denotationeller Semantik.

f primitiv rekursiv $\curvearrowright$ f LOOP berechenbar

Beweisskizze $\curvearrowright$ (2):

- **primitive Rekursion:** Zeige:

$f : \mathbb{N}^n \to \mathbb{N}, g : \mathbb{N}^{n+2} \to \mathbb{N}$ *LOOP-berechenbar*
$\curvearrowright h : \mathbb{N}^{n+1} \to \mathbb{N}$ *mit*

$$\begin{aligned} h(\overline{a}, 0) &= f(\overline{a}) \\ h(\overline{a}, b+1) &= g(\overline{a}, b, h(\overline{a}, b)) \end{aligned}$$

ist LOOP-berechenbar. Dabei stehe $\overline{a}$ *für* $a_1, \ldots, a_n$.

f primitiv rekursiv $\curvearrowright$ f LOOP berechenbar

Beweisskizze $\curvearrowright$ (2):

- **primitive Rekursion:** Zeige:

$f : \mathbb{N}^n \to \mathbb{N}, g : \mathbb{N}^{n+2} \to \mathbb{N}$ *LOOP-berechenbar*
$\curvearrowright h : \mathbb{N}^{n+1} \to \mathbb{N}$ *mit*

$$\begin{aligned} h(\overline{a}, 0) &= f(\overline{a}) \\ h(\overline{a}, b+1) &= g(\overline{a}, b, h(\overline{a}, b)) \end{aligned}$$

ist LOOP-berechenbar. Dabei stehe $\overline{a}$ *für* $a_1, \ldots, a_n$.

Berechne $h(\overline{a}, b)$ iterativ durch Berechnung von

$h(\overline{a}, 0)$	$h(\overline{a}, 1)$	$h(\overline{a}, 2)$	$\cdots$	$h(\overline{a}, b)$
$\shortparallel$	$\shortparallel$	$\shortparallel$		
$f(\overline{a})$	$g(\overline{a}, 0, f(\overline{a}))$	$g(\overline{a}, 1, g(\overline{a}, 0, f(\overline{a})))$	$\cdots$	

f primitiv rekursiv $\curvearrowright$ f LOOP berechenbar

Iterationsschema:

$$\begin{array}{cccc} h(\overline{a},0) & h(\overline{a},1) & h(\overline{a},2) & \cdots & h(\overline{a},b) \\ \| & \| & \| & & \\ f(\overline{a}) & g(\overline{a},0,f(\overline{a})) & g(\overline{a},1,g(\overline{a},0,f(\overline{a}))) & \cdots & \end{array}$$

Beweisskizze $\curvearrowright$ (3):

Formal: Für $\varphi : \begin{cases} \mathbb{N}^{n+2} & \to & \mathbb{N}^{n+2} \\ (\overline{a},b,c) & \mapsto & (\overline{a},b+1,g(\overline{a},b,c)) \end{cases}$
folgt leicht durch Induktion über b, dass

$$\circledast \qquad (\overline{a},b,h(\overline{a},b)) = \varphi^b(\overline{a},0,f(\overline{a})).$$

$\rightsquigarrow$ iterative Darstellung von h:

$$\boxed{h(\overline{a},b)) = pr_{n+2}^{(n+2)}(\varphi^b(\overline{a},0,f(\overline{a})))}$$

LOOP-Programm für h berechnet in LOOP-Schleife Iteration von φ.

f primitiv rekursiv $\curvearrowleft$ f LOOP berechenbar

Beweisskizze $\curvearrowleft$ (1):

Zeige: $[\![P]\!]$ ist für LOOP-Programm P primitiv rekursiv.

- Es gilt: $[\![P]\!] = \underline{\mathtt{out}}_1^{(m)} \circ [\![\alpha]\!]^{(m)} \circ \underline{\mathtt{in}}_m^{(n)}$, wobei
- $\underline{\mathtt{out}}_1^{(m)} = pr_1^{(m)}$ primitiv rekursiv ist.
- Zeige noch: Komponentenfunktionen von $\underline{\mathtt{in}}_m^{(n)}$ und $[\![\alpha]\!]^{(m)}$ sind primitiv rekursiv.
  - **Eingabefkt.:** Trivial (Projektionen, Nullkonstanten)
  - **Anweisungssemantik:** induktiv über die Definition der Anweisungsmenge.

f primitiv rekursiv $\curvearrowleft$ f LOOP berechenbar

Beweisskizze $\curvearrowleft$ (2):

Anweisungssem: induktiv über Definition der Anweisungsmenge

- **Wertzuweisungen, Sequenz**: klar
- **Laufanweisung**: Sei $\alpha = \underline{\texttt{loop}}\ X_j\ (\tilde{\alpha})$

  IV: $f_i := pr_i^{(m)} \circ [\![\tilde{\alpha}]\!]^{(m)}$ sind für $1 \leq i \leq m$ primitiv rekursiv.
  Wir definieren $g_i : \mathbb{N}^{m+1} \to \mathbb{N}$ $(1 \leq i \leq m)$ durch **simultane Rekursion**

$$\begin{aligned} g_i(\overline{a}, 0) &= a_i \\ g_i(\overline{a}, b+1) &= (f_i \circ [g_1, \ldots, g_m])(\overline{a}, b) \end{aligned}$$

f primitiv rekursiv $\curvearrowright$ f LOOP berechenbar

Wir definieren $g_i : \mathbb{N}^{m+1} \to \mathbb{N}$ $(1 \leq i \leq m)$ durch **simultane Rekursion**

$$\begin{aligned} g_i(\overline{a}, 0) &= a_i \\ g_i(\overline{a}, b+1) &= (f_i \circ [g_1, \ldots, g_m])(\overline{a}, b) \end{aligned}$$

Beweisskizze $\curvearrowright$ (3):

Zeige: alle g_i für $1 \leq i \leq m$ sind primitiv rekursiv.

$\rightsquigarrow$

$\square$

f primitiv rekursiv $\curvearrowright$ f LOOP berechenbar

Wir definieren $g_i : \mathbb{N}^{m+1} \to \mathbb{N}$ $(1 \leq i \leq m)$ durch **simultane Rekursion**

$$\begin{aligned} g_i(\overline{a}, 0) &= a_i \\ g_i(\overline{a}, b+1) &= (f_i \circ [g_1, \ldots, g_m])(\overline{a}, b) \end{aligned}$$

Beweisskizze $\curvearrowright$ (3):

Zeige: alle g_i für $1 \leq i \leq m$ sind primitiv rekursiv.

$\rightsquigarrow$

$$\begin{aligned} & pr_i^{(m)}([\![\underline{\texttt{loop}}\ X_j\ (\tilde{\alpha})]\!]^{(m)}(\overline{a}))) \\ &= pr_i^{(m)}(([\![\tilde{\alpha}]\!]^{(m)})^{a_j}(\overline{a})) \end{aligned}$$

□

f primitiv rekursiv $\curvearrowright$ f LOOP berechenbar

Wir definieren $g_i : \mathbb{N}^{m+1} \to \mathbb{N}$ $(1 \leq i \leq m)$ durch **simultane Rekursion**

$$\begin{aligned} g_i(\overline{a}, 0) &= a_i \\ g_i(\overline{a}, b+1) &= (f_i \circ [g_1, \ldots, g_m])(\overline{a}, b) \end{aligned}$$

Beweisskizze $\curvearrowright$ (3):

Zeige: alle g_i für $1 \leq i \leq m$ sind primitiv rekursiv.

$\rightsquigarrow$

$$\begin{aligned} & pr_i^{(m)}([\![\underline{\texttt{loop}}\ X_j\ (\tilde{\alpha})]\!]^{(m)}(\overline{a}))) \\ &= pr_i^{(m)}(([\![\tilde{\alpha}]\!]^{(m)})^{a_j}(\overline{a})) \\ &= g_i(\overline{a}, a_j) \end{aligned}$$

□

f primitiv rekursiv $\curvearrowright$ f LOOP berechenbar

Wir definieren $g_i : \mathbb{N}^{m+1} \to \mathbb{N}$ $(1 \leq i \leq m)$ durch **simultane Rekursion**

$$\begin{aligned} g_i(\overline{a}, 0) &= a_i \\ g_i(\overline{a}, b+1) &= (f_i \circ [g_1, \ldots, g_m])(\overline{a}, b) \end{aligned}$$

Beweisskizze $\curvearrowright$ (3):

Zeige: alle g_i für $1 \leq i \leq m$ sind primitiv rekursiv.

$\rightsquigarrow$

$$\begin{aligned} & pr_i^{(m)}([\![\underline{\texttt{loop}}\ X_j\ (\tilde{\alpha})]\!]^{(m)}(\overline{a}))) \\ &= pr_i^{(m)}(([\![\tilde{\alpha}]\!]^{(m)})^{a_j}(\overline{a})) \\ &= g_i(\overline{a}, a_j) \end{aligned}$$

□

f primitiv rekursiv $\curvearrowright$ f LOOP berechenbar

Wir definieren $g_i : \mathbb{N}^{m+1} \to \mathbb{N}$ $(1 \leq i \leq m)$ durch **simultane Rekursion**

$$\begin{aligned} g_i(\overline{a}, 0) &= a_i \\ g_i(\overline{a}, b+1) &= (f_i \circ [g_1, \ldots, g_m])(\overline{a}, b) \end{aligned}$$

Beweisskizze $\curvearrowright$ (3):

Zeige: alle g_i für $1 \leq i \leq m$ sind primitiv rekursiv.

$\rightsquigarrow$

$$\begin{aligned} & pr_i^{(m)}([\![\underline{\texttt{loop}}\ X_j\ (\tilde{\alpha})]\!]^{(m)}(\overline{a}))) \\ &= pr_i^{(m)}(([\![\tilde{\alpha}]\!]^{(m)})^{a_j}(\overline{a})) \\ &= g_i(\overline{a}, a_j) \\ &= g_i \circ [pr_1^{(m)}, \ldots, pr_m^{(m)}, pr_j^{(m)}](\overline{a}). \end{aligned}$$

$\square$

Ackermann-Funktion

David Hilbert 1926

Frage:

pimititiv rekursive Funktionen $=$ total und berechenbar?

Ackermann-Funktion

David Hilbert 1926

Frage:

$$\text{pimititiv rekursive Funktionen} \quad \begin{matrix} = \\ \subseteq \end{matrix} \quad \text{total und berechenbar?}$$

Ackermann-Funktion

David Hilbert 1926

Frage:

$$\begin{array}{lll} \text{pimititiv rekursive Funktionen} & = & \text{total und berechenbar?} \\ & \subseteq & \checkmark \end{array}$$

Ackermann-Funktion

David Hilbert 1926

Frage:

$$\begin{array}{lcl} \text{pimititiv rekursive Funktionen} & = & \text{total und berechenbar?} \\ & \subseteq & \checkmark \\ & \supseteq & \end{array}$$

Ackermann-Funktion

David Hilbert 1926

Frage:

$$\begin{array}{lll} \text{pimititiv rekursive Funktionen} & = & \text{total und berechenbar?} \\ & \subseteq & \checkmark \\ & \supseteq & ? \end{array}$$

Ackermann-Funktion

David Hilbert 1926

Frage:

pimititiv rekursive Funktionen	$=$	total und berechenbar?
	$\subseteq$	✓
	$\supseteq$	$\times$

Antwort: nein

Ackermann-Funktion

David Hilbert 1926

Frage:

pimititiv rekursive Funktionen	$=$	total und berechenbar?
	$\subseteq$	✓
	$\supseteq$	

Antwort: nein

Satz:

Die **Ackermann-Funktion** ist total und berechenbar, aber nicht primitiv rekursiv und damit auch nicht LOOP-berechenbar.

Ackermann-Funktion

Definition

Die Funktion $ack : \mathbb{N}^2 \to \mathbb{N}$ mit

$$\begin{array}{lcll} ack(0,0) & := & 1 & \\ ack(0,1) & := & 2 & \\ ack(0,y) & := & y+2 & (y \geq 2) \\ ack(x+1,0) & := & 1 & (x \geq 0) \\ ack(x+1,y+1) & := & ack(x, ack(x+1,y)) & \end{array}$$

ist eine *Ackermann-Funktion*.

Ackermann-Funktion

Beweisidee:

Die Ackermann-Funktion wächst stärker als jede primitiv rekursive Funktion.

Berechnungsstärke von LOOP-Programmen wächst mit Schachtelungstiefe.
$\rightarrow$ Hierarchie LOOP-berechenbarer Funktionsklassen, z.B:

$$
\begin{array}{lcll}
ack(0,y) & = & y+2 & (y \geq 2) \quad \text{Summe} \\
ack(1,y) & = & 2 * y & (y \geq 1) \quad \text{Produkt} \\
ack(2,y) & = & 2^y & (y \geq 0) \quad \text{Potenz} \\
ack(3,y) & = & \left. 2^{2^{\cdots^2}} \right\} y\text{-mal} & (y \geq 1) \quad \text{Hyperpotenz}
\end{array}
$$

Ackermann-Funktion

Wachstumslemma (ohne Beweis)

Zu jeder primitiv rekursiven Funktion $f : \mathbb{N}^r \to \mathbb{N}$ existiert ein $c \in \mathbb{N}$, so dass

$$f(n_1, \ldots, n_r) < ack(c, \sum_{j=1}^{r} n_j)$$

für alle $n_1, \ldots, n_r \in \mathbb{N}$.

Beweis: (Satz von Ackermann)

Annahme, *ack* sei primitiv rekursiv.
Dann auch $g(x) = ack(x, x)$, d.h. es existiert ein $c \in \mathbb{N}$ mit

$$g(x) < ack(c, x) \text{ für alle } x$$

$$\lightning \text{ für } x = c$$

Inhalt

Die Sprache WHILE

LOOP $\rightarrow$ Zählschleife, d.h. # Durchläufe steht vor Ausführung fest

$\rightarrow$ totale Funktionen

WHILE $\rightarrow$ Bedingte Schleifen, d.h. # Durchläufe hängt von Tests auf Zustandsraum ab.

$\rightarrow$ eventuell unendliche Berechnungen

$\rightarrow$ partielle Funktionen

Die Sprache WHILE

LOOP $\rightarrow$ Zählschleife, d.h. # Durchläufe steht vor Ausführung fest

$\rightarrow$ totale Funktionen

WHILE $\rightarrow$ Bedingte Schleifen, d.h. # Durchläufe hängt von Tests auf Zustandsraum ab.

$\rightarrow$ eventuell unendliche Berechnungen

$\rightarrow$ partielle Funktionen

Die Sprache WHILE

LOOP $\rightarrow$ Zählschleife, d.h. # Durchläufe steht vor Ausführung fest

$\rightarrow$ totale Funktionen

WHILE $\rightarrow$ Bedingte Schleifen, d.h. # Durchläufe hängt von Tests auf Zustandsraum ab.

$\rightarrow$ eventuell unendliche Berechnungen

$\rightarrow$ partielle Funktionen

Die Sprache WHILE

LOOP $\rightarrow$ Zählschleife, d.h. # Durchläufe steht vor Ausführung fest

$\rightarrow$ totale Funktionen

WHILE $\rightarrow$ Bedingte Schleifen, d.h. # Durchläufe hängt von Tests auf Zustandsraum ab.

$\rightarrow$ eventuell unendliche Berechnungen

$\rightarrow$ partielle Funktionen

Die Sprache WHILE

LOOP $\rightarrow$ Zählschleife, d.h. # Durchläufe steht vor Ausführung fest

$\rightarrow$ totale Funktionen

WHILE $\rightarrow$ Bedingte Schleifen, d.h. # Durchläufe hängt von Tests auf Zustandsraum ab.

$\rightarrow$ eventuell unendliche Berechnungen

$\rightarrow$ partielle Funktionen

Syntax von WHILE

Definition:

- *Variablen*: $\mathcal{V} := \{X_i \mid i \geq 1\}$
- *Wertzuweisungen*: $\mathcal{W} := \{ \quad X_i := X_j + 1, \quad X_i := X_j - 1, \quad X_i := X_j, \quad X_i := 0 \quad \mid i, j \geq 1\}$
- *Anweisungen*: kleinste Menge $\mathcal{A}$ mit den Eigenschaften
  1. $\mathcal{W} \subseteq \mathcal{A}$
  2. $\alpha, \beta \in \mathcal{A}$ impliziert $\alpha; \beta \in \mathcal{A}$
  3. $\alpha \in \mathcal{A}$, $X_i \in \mathcal{V}$ impliziert $\underline{\texttt{while}}\ X_i \neq 0\ \underline{\texttt{do}}\ \alpha\ \underline{\texttt{od}} \in \mathcal{A}$

Syntax von WHILE

Definition:

- *Variablen*: $\mathcal{V} := \{X_i \mid i \geq 1\}$
- *Wertzuweisungen*: $\mathcal{W} := \{ \quad X_i := X_j + 1, \quad X_i := X_j - 1, \quad X_i := X_j, \quad X_i := 0 \quad \mid i, j \geq 1\}$
- *Anweisungen*: kleinste Menge $\mathcal{A}$ mit den Eigenschaften
  1. $\mathcal{W} \subseteq \mathcal{A}$
  2. $\alpha, \beta \in \mathcal{A}$ impliziert $\alpha; \beta \in \mathcal{A}$
  3. $\alpha \in \mathcal{A}$, $X_i \in \mathcal{V}$ impliziert $\underline{\texttt{while}}\ X_i \neq 0\ \underline{\texttt{do}}\ \alpha\ \underline{\texttt{od}} \in \mathcal{A}$

Syntax von WHILE

Definition:

- *Variablen*: $\mathcal{V} := \{X_i \mid i \geq 1\}$
- *Wertzuweisungen*: $\mathcal{W} := \{ \quad X_i := X_j + 1, \quad X_i := X_j - 1, \quad X_i := X_j, \quad X_i := 0 \quad \mid i, j \geq 1\}$
- *Anweisungen*: kleinste Menge $\mathcal{A}$ mit den Eigenschaften
  1. $\mathcal{W} \subseteq \mathcal{A}$
  2. $\alpha, \beta \in \mathcal{A}$ impliziert $\alpha; \beta \in \mathcal{A}$
  3. $\alpha \in \mathcal{A},\ X_i \in \mathcal{V}$ impliziert $\underline{\texttt{while}}\ X_i \neq 0\ \underline{\texttt{do}}\ \alpha\ \underline{\texttt{od}}\ \in \mathcal{A}$

Syntax von WHILE

Definition:

- *Variablen*: $\mathcal{V} := \{X_i \mid i \geq 1\}$
- *Wertzuweisungen*: $\mathcal{W} := \{ \quad X_i := X_j + 1, \quad X_i := X_j - 1, \quad X_i := X_j, \quad X_i := 0 \quad \mid i, j \geq 1\}$
- *Anweisungen*: kleinste Menge $\mathcal{A}$ mit den Eigenschaften
  1. $\mathcal{W} \subseteq \mathcal{A}$
  2. $\alpha, \beta \in \mathcal{A}$ impliziert $\alpha; \beta \in \mathcal{A}$
  3. $\alpha \in \mathcal{A},\ X_i \in \mathcal{V}$ impliziert $\texttt{while}\ X_i \neq 0\ \texttt{do}\ \alpha\ \texttt{od} \in \mathcal{A}$

Syntax von WHILE

Definition:

- *Variablen*: $\mathcal{V} := \{X_i \mid i \geq 1\}$
- *Wertzuweisungen*: $\mathcal{W} := \{ \quad X_i := X_j + 1, \quad X_i := X_j - 1, \quad X_i := X_j, \quad X_i := 0 \quad \mid i, j \geq 1\}$
- *Anweisungen*: kleinste Menge $\mathcal{A}$ mit den Eigenschaften
  1. $\mathcal{W} \subseteq \mathcal{A}$
  2. $\alpha, \beta \in \mathcal{A}$ impliziert $\alpha; \beta \in \mathcal{A}$
  3. $\alpha \in \mathcal{A},\ X_i \in \mathcal{V}$ impliziert $\texttt{while}\ X_i \neq 0\ \texttt{do}\ \alpha\ \texttt{od} \in \mathcal{A}$

Syntax von WHILE

Definition:

- *Variablen*: $\mathcal{V} := \{X_i \mid i \geq 1\}$
- *Wertzuweisungen*: $\mathcal{W} := \{ \quad X_i := X_j + 1, \quad X_i := X_j - 1, \quad X_i := X_j, \quad X_i := 0 \quad \mid i, j \geq 1\}$
- *Anweisungen*: kleinste Menge $\mathcal{A}$ mit den Eigenschaften
  1. $\mathcal{W} \subseteq \mathcal{A}$
  2. $\alpha, \beta \in \mathcal{A}$ impliziert $\alpha; \beta \in \mathcal{A}$
  3. $\alpha \in \mathcal{A},\ X_i \in \mathcal{V}$ impliziert $\underline{\texttt{while}}\ X_i \neq 0\ \underline{\texttt{do}}\ \alpha\ \underline{\texttt{od}}\ \in \mathcal{A}$

Syntax von WHILE

Definition:

- *Variablen*: $\mathcal{V} := \{X_i \mid i \geq 1\}$
- *Wertzuweisungen*: $\mathcal{W} := \{ \quad X_i := X_j + 1, \quad X_i := X_j - 1, \quad X_i := X_j, \quad X_i := 0 \quad \mid i, j \geq 1\}$
- *Anweisungen*: kleinste Menge $\mathcal{A}$ mit den Eigenschaften
  1. $\mathcal{W} \subseteq \mathcal{A}$
  2. $\alpha, \beta \in \mathcal{A}$ impliziert $\alpha; \beta \in \mathcal{A}$
  3. $\alpha \in \mathcal{A},\ X_i \in \mathcal{V}$ impliziert $\underline{\texttt{while}}\ X_i \neq 0\ \underline{\texttt{do}}\ \alpha\ \underline{\texttt{od}}\ \in \mathcal{A}$

Die Menge aller While-Programme wird mit $\mathcal{P}_{\text{WHILE}}$bezeichnet.

Syntax von WHILE

WHILE-Beispielprogramm:

$$
\begin{aligned}
P_1 = \quad & \underline{\texttt{in}}(X_1, X_2);\ \underline{\texttt{var}}(X_3, X_4); \\
& \underline{\texttt{while}}\ X_1 \neq 0\ \underline{\texttt{do}} \\
& \qquad X_4 := X_2; \\
& \qquad \underline{\texttt{while}}\ X_4 \neq 0\ \underline{\texttt{do}} \\
& \qquad\qquad X_3 := X_3 + 1; \\
& \qquad\qquad X_4 := X_4 - 1 \\
& \qquad \underline{\texttt{od}}; \\
& \qquad X_1 := X_1 - 1 \\
& \underline{\texttt{od}}; \\
& X_1 := X_3; \\
& \underline{\texttt{out}}\ X_1
\end{aligned}
$$

Syntax von WHILE

WHILE-Beispielprogramm:

$$
\begin{aligned}
P_1 = \quad & \underline{\texttt{in}}(X_1, X_2);\ \underline{\texttt{var}}(X_3, X_4); \\
& \underline{\texttt{while}}\ X_1 \neq 0\ \underline{\texttt{do}} \\
& \qquad X_4 := X_2; \\
& \qquad \underline{\texttt{while}}\ X_4 \neq 0\ \underline{\texttt{do}} \\
& \qquad\qquad X_3 := X_3 + 1; \\
& \qquad\qquad X_4 := X_4 - 1 \\
& \qquad \underline{\texttt{od}}; \\
& \qquad X_1 := X_1 - 1 \\
& \underline{\texttt{od}}; \\
& X_1 := X_3; \\
& \underline{\texttt{out}}\ X_1
\end{aligned}
$$

$[\![P_1]\!] : \mathbb{N}^2 \to \mathbb{N}, \qquad (a_1, a_2) \mapsto a_1 * a_2$

Semantik von WHILE

Definition (1):

Im wesentlichen wie bei LOOP, mit wichtigen Unterschieden:

$$[\![.]\!] : \mathcal{P}_{\text{WHILE}} \to \{f : \mathbb{N}^r \dashrightarrow \mathbb{N} \mid r \geq 0\}$$

Die Bildmenge der Semantik ist die Menge der **partiellen** arithmetischen Operationen.

Semantik von WHILE

Definition (1):

Im wesentlichen wie bei LOOP, mit wichtigen Unterschieden:

$$[\![.]\!] : \mathcal{P}_{\mathrm{WHILE}} \to \{f : \mathbb{N}^r \dashrightarrow \mathbb{N} \mid r \geq 0\}$$

Die Bildmenge der Semantik ist die Menge der **partiellen** arithmetischen Operationen.

$P \in \mathcal{P}_{\mathrm{WHILE}}$ wie bei LOOP: $[\![P]\!] = \underline{\mathtt{out}}_1^{(m)} \circ [\![\alpha]\!]^{(m)} \circ \underline{\mathtt{in}}_m^{(n)}$, **aber**

1. $[\![\alpha]\!]^{(m)} : \mathbb{N}^m \dashrightarrow \mathbb{N}^m$ ist **partielle** Zustandstransformation.

Semantik von WHILE

Definition (1):

Im wesentlichen wie bei LOOP, mit wichtigen Unterschieden:

$$[\![.]\!] : \mathcal{P}_{\mathrm{WHILE}} \to \{f : \mathbb{N}^r \dashrightarrow \mathbb{N} \mid r \geq 0\}$$

Die Bildmenge der Semantik ist die Menge der **partiellen** arithmetischen Operationen.

$P \in \mathcal{P}_{\mathrm{WHILE}}$ wie bei LOOP: $[\![P]\!] = \underline{\mathtt{out}}_1^{(m)} \circ [\![\alpha]\!]^{(m)} \circ \underline{\mathtt{in}}_m^{(n)}$, **aber**

1. $[\![\alpha]\!]^{(m)} : \mathbb{N}^m \dashrightarrow \mathbb{N}^m$ ist **partielle** Zustandstransformation.
2. Die Komposition $\circ$ ist **strikt**, also

$$(g \circ f)(a) = \begin{cases} g(f(a)) & \text{falls } f(a) \text{ def.} \\ \text{nicht definiert} & \text{sonst.} \end{cases}$$

Semantik von WHILE

Definition (2):

Definiere die Zustandstransformation der While-Anweisung:

$$[\![\underline{\texttt{while}}\ X_i \neq 0\ \underline{\texttt{do}}\ \alpha\ \underline{\texttt{od}}]\!]^{(m)}(\underbrace{a_1, \ldots, a_m}_{=:\bar{a}})$$

$$:= \begin{cases} ([\![\alpha]\!]^{(m)})^k(\bar{a}) & \text{falls ein } k \in \mathbb{N} \text{ existiert, so dass} \\ & \forall\ 1 \leq \nu < k : ([\![\alpha]\!]^{(m)})^\nu(\bar{a}) \text{ def.} \wedge \\ & pr_i^{(m)}(([\![\alpha]\!]^{(m)})^\nu(\bar{a})) > 0 \\ & \text{und außerdem gilt} \\ & ([\![\alpha]\!]^{(m)})^k(\bar{a}) \text{ def.} \wedge \\ & pr_i^{(m)}(([\![\alpha]\!]^{(m)})^k(\bar{a})) = 0 \\ \text{nicht definiert} & \text{sonst.} \end{cases}$$

Semantik von WHILE

Definition (2):

Definiere die Zustandstransformation der While-Anweisung:

$$[\![\underline{\texttt{while}}\ X_i \neq 0\ \underline{\texttt{do}}\ \alpha\ \underline{\texttt{od}}]\!]^{(m)}(\underbrace{a_1, \ldots, a_m}_{=:\bar{a}})$$

$$:= \begin{cases} ([\![\alpha]\!]^{(m)})^k(\bar{a}) & \text{falls ein } k \in \mathbb{N} \text{ existiert, so dass} \\ & \forall\ 1 \leq \nu < k : ([\![\alpha]\!]^{(m)})^\nu(\bar{a}) \text{ def.} \wedge \\ & pr_i^{(m)}(([\![\alpha]\!]^{(m)})^\nu(\bar{a})) > 0 \\ & \text{und außerdem gilt} \\ & ([\![\alpha]\!]^{(m)})^k(\bar{a}) \text{ def.} \wedge \\ & pr_i^{(m)}(([\![\alpha]\!]^{(m)})^k(\bar{a})) = 0 \\ \text{nicht definiert} & \text{sonst.} \end{cases}$$

$[\![\underline{\texttt{while}}\ X_i \neq 0\ \underline{\texttt{do}}\ \alpha\ \underline{\texttt{od}}]\!]^{(m)}$ ist für $\bar{a}$ nicht definiert, falls kein solches k existiert.

WHILE vs. LOOP

- Jede LOOP-Schleife kann mit einer WHILE-Schleife simuliert werden. Daher folgt:

Satz

Jede LOOP-berechenbare Funktion ist auch WHILE-berechenbar.

- Die Umkehrung gilt i.A. nicht.
- Charakterisiere WHILE-berechenbare Funktionen durch μ-**rekursive** Funktionen.

WHILE vs. LOOP

- Jede LOOP-Schleife kann mit einer WHILE-Schleife simuliert werden. Daher folgt:

Satz

Jede LOOP-berechenbare Funktion ist auch WHILE-berechenbar.

- Die Umkehrung gilt i.A. nicht.
- Charakterisiere WHILE-berechenbare Funktionen durch μ-**rekursive** Funktionen.

WHILE vs. LOOP

- Jede LOOP-Schleife kann mit einer WHILE-Schleife simuliert werden. Daher folgt:

Satz

Jede LOOP-berechenbare Funktion ist auch WHILE-berechenbar.

- Die Umkehrung gilt i.A. nicht.
- Charakterisiere WHILE-berechenbare Funktionen durch μ-**rekursive** Funktionen.

Minimalisierung

Definition:

Für eine Funktion $f : \mathbb{N}^{n+1} \dashrightarrow \mathbb{N}$ heißt
$\mu(f) : \mathbb{N}^n \dashrightarrow \mathbb{N}$ mit

$$\mu(f)(\bar{a}) := \begin{cases} b & \text{falls} \quad f(\bar{a}, b) = 0 \wedge \\ & \qquad\quad f(\bar{a}, c) > 0 \qquad \forall\ c < b \\ n.d. & \textit{sonst} \end{cases}$$

die *Minimalisierung*.

Minimalisierung

Definition:

Für eine Funktion $f : \mathbb{N}^{n+1} \dashrightarrow \mathbb{N}$ heißt
$\mu(f) : \mathbb{N}^n \dashrightarrow \mathbb{N}$ mit

$$\mu(f)(\bar{a}) := \begin{cases} b & \text{falls} \quad f(\bar{a}, b) = 0 \wedge \\ & \qquad\quad f(\bar{a}, c) > 0 \qquad \forall\ c < b \\ n.d. & \textit{sonst} \end{cases}$$

die *Minimalisierung*.

Beispiel:

- Für $f(x, y) = x \cdot y$ ist $\mu(f)(x) \equiv 0$,
- Für $g(x, y) = x + y$ ist $\mu(g)(x) = \begin{cases} 0 & , \text{falls } x = 0 \\ \text{n.d.} & , \text{sonst .} \end{cases}$

Minimalisierung

Definition:

Für eine Funktion $f : \mathbb{N}^{n+1} \dashrightarrow \mathbb{N}$ heißt
$\mu(f) : \mathbb{N}^n \dashrightarrow \mathbb{N}$ mit

$$\mu(f)(\bar{a}) := \begin{cases} b & \text{falls} \quad f(\bar{a}, b) = 0 \wedge \\ & \qquad\quad f(\bar{a}, c) > 0 \qquad \forall\ c < b \\ n.d. & \textit{sonst} \end{cases}$$

die *Minimalisierung*.

Beispiel:

- Für $f(x, y) = x \cdot y$ ist $\mu(f)(x) \equiv 0$,
- Für $g(x, y) = x + y$ ist $\mu(g)(x) = \begin{cases} 0 & \text{, falls } x = 0 \\ \text{n.d.} & \text{, sonst .} \end{cases}$

μ-rekursive Funktionen

Definition:

Eine Funktion $f : \mathbb{N}^n \dashrightarrow \mathbb{N}$ heißt μ-**rekursiv**, falls sie eine **primitive Grundfunktion** ist oder sich aus diesen in endlich vielen Schritten durch

- **Komposition**,
- **primitive Rekursion** oder
- **Minimalisierung**

erzeugen lässt.

μ-rekursive Funktionen

Beispiel: μ-Rekursivität der partiellen Subtraktion

$f : \mathbb{N}^2 \dashrightarrow \mathbb{N}$ mit

$$f(a,b) = \begin{cases} a - b & \text{falls } a \geq b \\ \text{nicht def.} & \text{sonst.} \end{cases}$$

- Gesucht: $h : \mathbb{N}^3 \dashrightarrow \mathbb{N}$, deren Minimalisierung die partielle Subtraktion ergibt, d.h. $f = \mu(h)$.
- h muss folgende Eigenschaften besitzen:
  1. Falls $a < b$ soll gelten: $h(a,b,y) > 0$.
  2. Falls $a \geq b$ soll gelten: $h(a,b,f(a,b)) = 0$ und $h(a,b,c) > 0$ für alle $c < f(a,b)$.
- $h(a,b,y) = \mid a \dot{-} (b+y) \mid$ erfüllt diese Anforderungen.

Minimalisierung

Beispiel: μ-Rekursivität der partiellen Subtraktion (2)

- $h(a, b, y) = \mid a \dot{-} (b + y) \mid$ als Komposition primitiv rekursiver Funktionen:

$$h(a, b, y) = add(\quad sub(add(b, y), a), \\ sub(a, add(b, y))).$$

... dabei sei $sub : \mathbb{N}^2 \to \mathbb{N}$ die totale Subtraktion mit

$$sub(a, b) = \begin{cases} a - b & \text{falls } a \geq b \\ 0 & \text{sonst.} \end{cases}$$

- Bleibt die primitive Rekursivität von *sub* und der für *h* geforderten Eigenschaften zu zeigen.

Minimalisierung

Beispiel: μ-Rekursivität der partiellen Subtraktion (2)

- $h(a, b, y) = \mid a \dot{-} (b + y) \mid$ als Komposition primitiv rekursiver Funktionen:

$$h(a, b, y) = add(\quad sub(add(b, y), a), \\ sub(a, add(b, y))).$$

. . . dabei sei $sub : \mathbb{N}^2 \to \mathbb{N}$ die totale Subtraktion mit

$$sub(a, b) = \begin{cases} a - b & \text{falls } a \geq b \\ 0 & \text{sonst.} \end{cases}$$

- Bleibt die primitive Rekursivität von sub und der für h geforderten Eigenschaften zu zeigen.

μ-rekursiv und WHILE berechenbar

Satz:

Sei $f : \mathbb{N}^k \dashrightarrow \mathbb{N}$ eine partielle Funktion.
Ist f μ-rekursiv, so ist f auch WHILE-berechenbar.

Beweis (1):

f primitiv-rekursive $\curvearrowright$
f LOOP-berechenbar $\curvearrowright$
f WHILE-berechenbar.

Zeige noch: Die Minimalisierung einer While-berechenbaren Funktion ist wieder While-berechenbar

μ-rekursiv und WHILE berechenbar

Beweis (f μ-rekursiv $\curvearrowright$ f WHILE-berechenbar) (2):

Zeige noch: Die Minimalisierung einer While-berechenbaren Funktion ist wieder While-berechenbar.

- Sei

$$P = \underline{\texttt{in}}(X_1, \ldots, X_{n+1});\ \underline{\texttt{var}}(X_{n+2}, \ldots, X_m);\ \alpha;\ \underline{\texttt{out}}(X_1).$$

  ein WHILE-Programm zur Berechnung einer Funktion $f = [\![P]\!] : \mathbb{N}^{n+1} \dashrightarrow \mathbb{N}$.

- Konstruiere ein WHILE-Programm

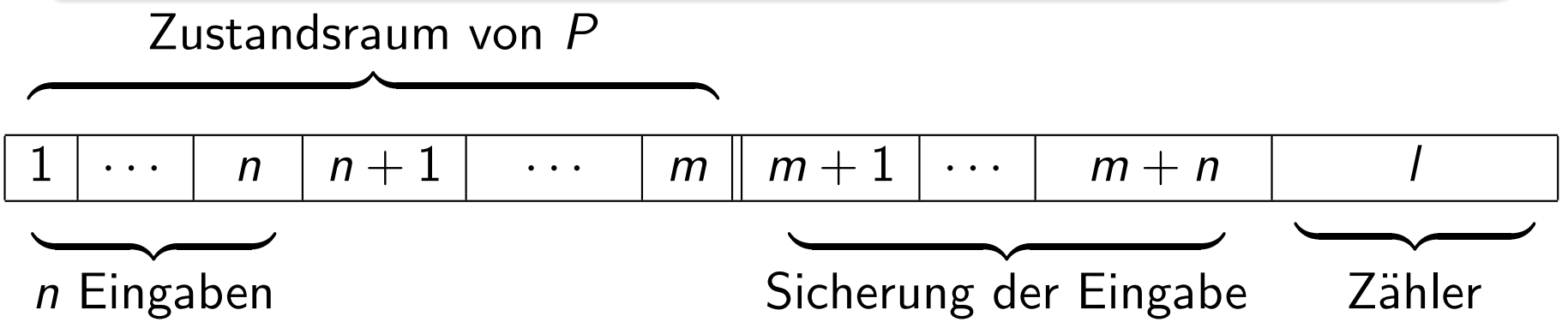
$$Q := \underline{\texttt{in}}(X_1, \ldots, X_n);\ \underline{\texttt{var}}(X_{n+1}, \ldots, X_l);\ \beta;\ \underline{\texttt{out}}(X_1).$$

  mit $l = m + n + 1$, welches $\mu(f)$ berechnet.

μ-rekursiv und WHILE berechenbar

Beweis (f μ-rekursiv $\curvearrowright$ f WHILE-berechenbar) (3):

Der Zustandsraum von Q wird wie folgt organisiert:



Sei: β = $X_{m+1} := X_1; \ldots; X_{m+n} := X_n;\ \alpha$

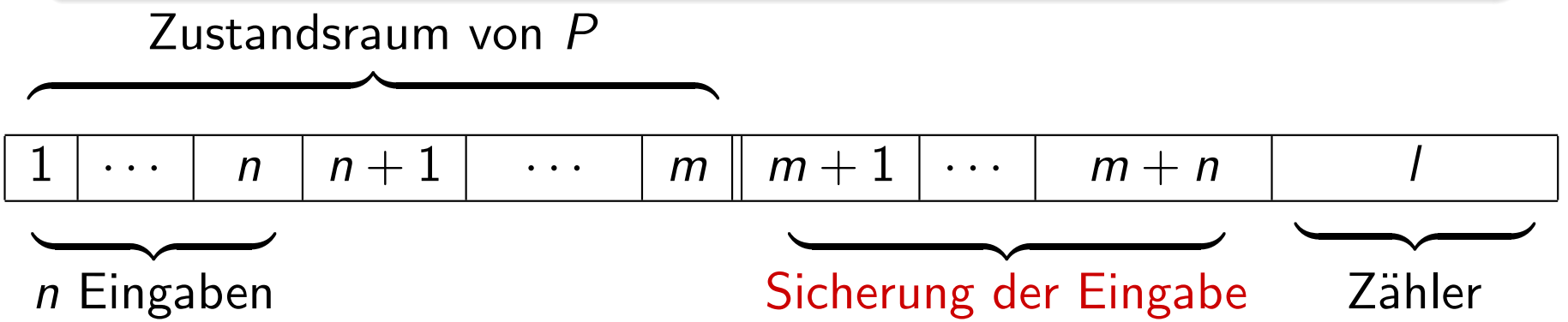
```
while X_1 ≠ 0 do
    X_l := X_l + 1;  X_1 := X_{m+1}; ...; X_n := X_{m+n}
    X_{n+1} := X_l;  X_{n+2} := 0; ...; X_m := 0;  α
od ;
X_1 := X_l
```

$\curvearrowright$ Q berechnet genau $\mu(f)$. ■

μ-rekursiv und WHILE berechenbar

Beweis (f μ-rekursiv $\curvearrowright$ f WHILE-berechenbar) (3):

Der Zustandsraum von Q wird wie folgt organisiert:



Sei: β = $X_{m+1} := X_1; \ldots; X_{m+n} := X_n;\ \alpha$

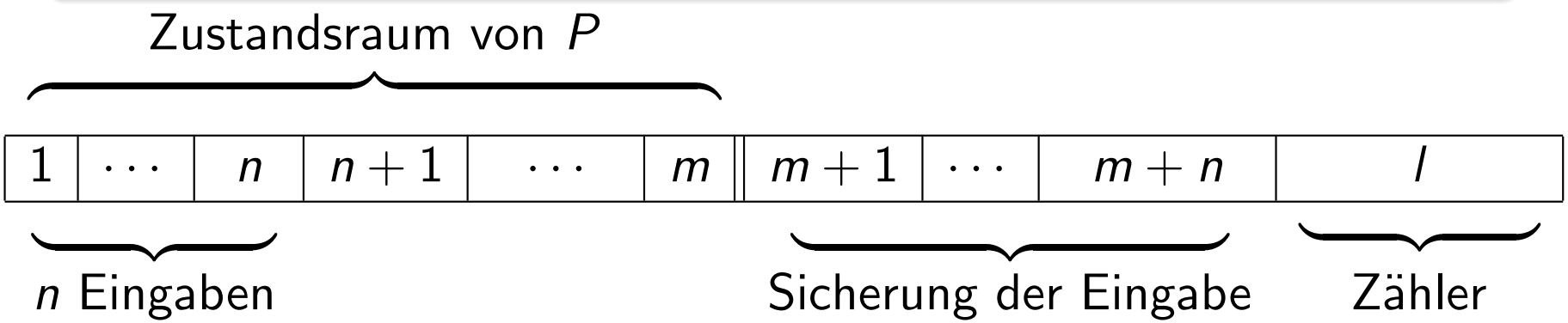
```
while X_1 ≠ 0 do
    X_l := X_l + 1;  X_1 := X_{m+1}; ...; X_n := X_{m+n}
    X_{n+1} := X_l;  X_{n+2} := 0; ...; X_m := 0;  α
od ;
X_1 := X_l
```

$\curvearrowright$ Q berechnet genau $\mu(f)$. ■

μ-rekursiv und WHILE berechenbar

Beweis (f μ-rekursiv $\curvearrowright$ f WHILE-berechenbar) (3):

Der Zustandsraum von Q wird wie folgt organisiert:



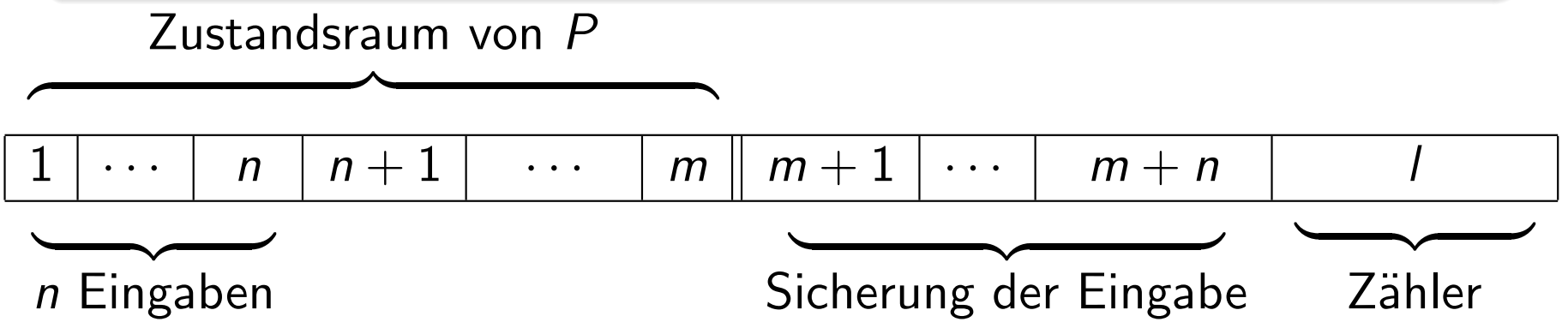
Sei: β = $X_{m+1} := X_1; \ldots; X_{m+n} := X_n;$ α
$\underline{\texttt{while}}\ X_1 \neq 0\ \underline{\texttt{do}}$
$X_l := X_l + 1;\ X_1 := X_{m+1}; \ldots; X_n := X_{m+n}$
$X_{n+1} := X_l;\ X_{n+2} := 0; \ldots; X_m := 0;$ α
$\underline{\texttt{od}}$;
$X_1 := X_l$

$\curvearrowright$ Q berechnet genau $\mu(f)$. ■

μ-rekursiv und WHILE berechenbar

Beweis (f μ-rekursiv $\curvearrowright$ f WHILE-berechenbar) (3):

Der Zustandsraum von Q wird wie folgt organisiert:



Sei: β = $X_{m+1} := X_1; \ldots; X_{m+n} := X_n;\ \alpha$

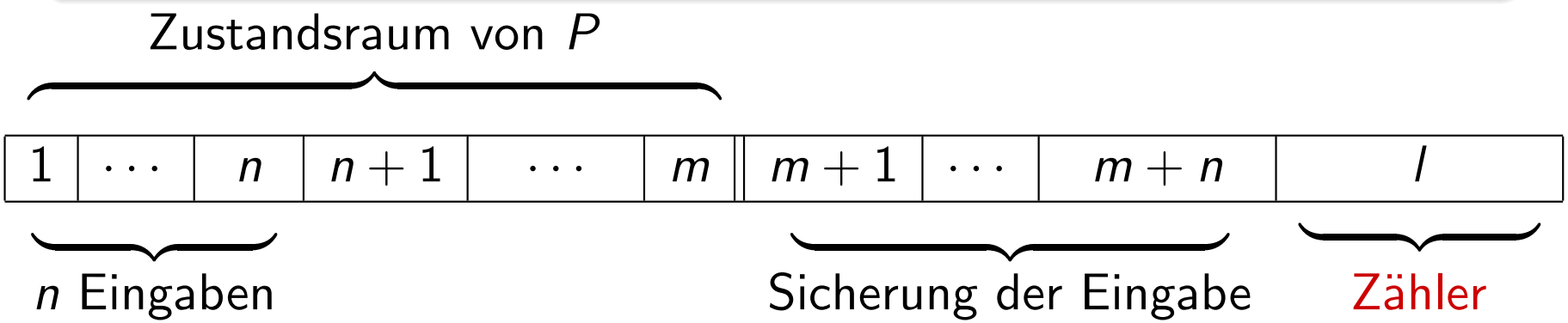
```
while X_1 ≠ 0 do
    X_l := X_l + 1;  X_1 := X_{m+1}; ...; X_n := X_{m+n}
    X_{n+1} := X_l;  X_{n+2} := 0; ...; X_m := 0;  α
od ;
X_1 := X_l
```

$\curvearrowright$ Q berechnet genau $\mu(f)$. ■

μ-rekursiv und WHILE berechenbar

Beweis (f μ-rekursiv $\curvearrowright$ f WHILE-berechenbar) (3):

Der Zustandsraum von Q wird wie folgt organisiert:



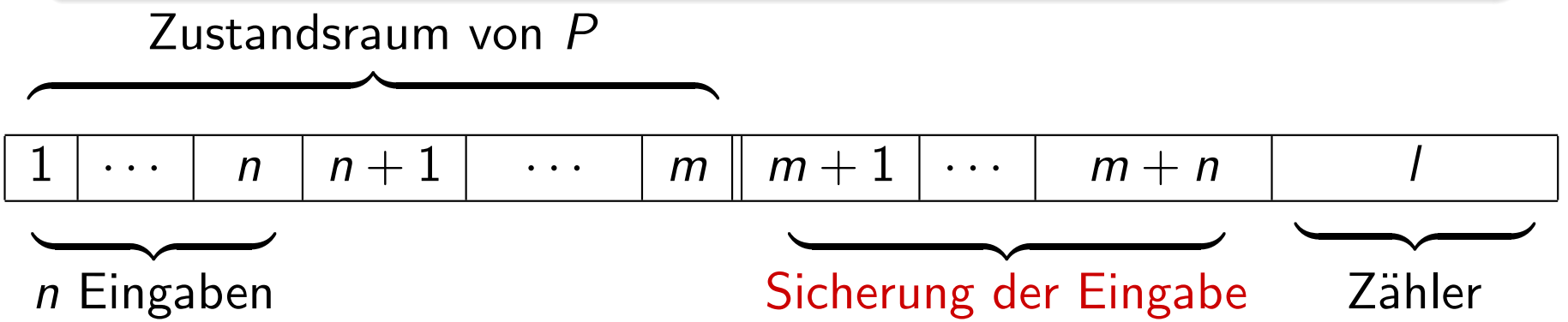
Sei: β = $X_{m+1} := X_1; \ldots; X_{m+n} := X_n;\ \alpha$
$\underline{\texttt{while}}\ X_1 \neq 0\ \underline{\texttt{do}}$
$X_l := X_l + 1;\ X_1 := X_{m+1}; \ldots; X_n := X_{m+n}$
$X_{n+1} := X_l;\ X_{n+2} := 0; \ldots; X_m := 0;\ \alpha$
$\underline{\texttt{od}}$;
$X_1 := X_l$

$\curvearrowright$ Q berechnet genau $\mu(f)$. ■

μ-rekursiv und WHILE berechenbar

Beweis (f μ-rekursiv $\curvearrowright$ f WHILE-berechenbar) (3):

Der Zustandsraum von Q wird wie folgt organisiert:



Sei: β = $X_{m+1} := X_1; \ldots; X_{m+n} := X_n;\ \alpha$

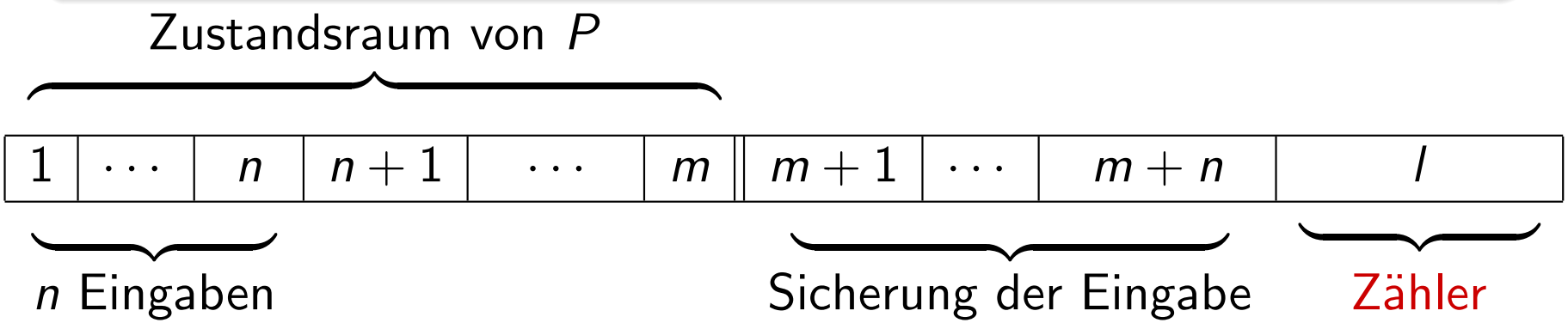
```
while X_1 ≠ 0 do
    X_l := X_l + 1;  X_1 := X_{m+1}; ...; X_n := X_{m+n}
    X_{n+1} := X_l;  X_{n+2} := 0; ...; X_m := 0;  α
od ;
X_1 := X_l
```

$\curvearrowright$ Q berechnet genau $\mu(f)$. ■

μ-rekursiv und WHILE berechenbar

Beweis (f μ-rekursiv $\curvearrowright$ f WHILE-berechenbar) (3):

Der Zustandsraum von Q wird wie folgt organisiert:



Sei: β = $X_{m+1} := X_1; \ldots; X_{m+n} := X_n;\ \alpha$

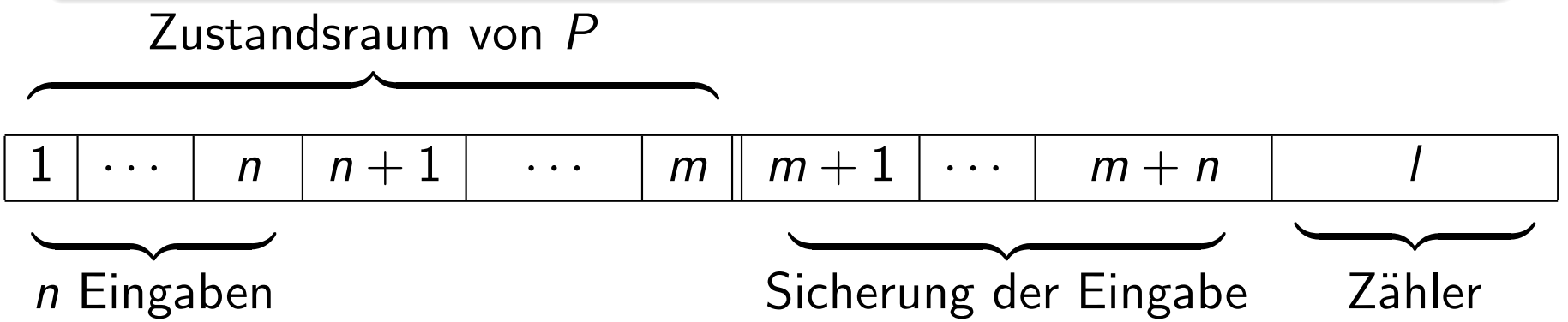
```
while X_1 ≠ 0 do
    X_l := X_l + 1;  X_1 := X_{m+1}; ...; X_n := X_{m+n}
    X_{n+1} := X_l;  X_{n+2} := 0; ...; X_m := 0;  α
od ;
X_1 := X_l
```

$\curvearrowright$ Q berechnet genau $\mu(f)$. ■

μ-rekursiv und WHILE berechenbar

Beweis (f μ-rekursiv $\curvearrowright$ f WHILE-berechenbar) (3):

Der Zustandsraum von Q wird wie folgt organisiert:



Sei: β = $X_{m+1} := X_1; \ldots; X_{m+n} := X_n;\ \alpha$

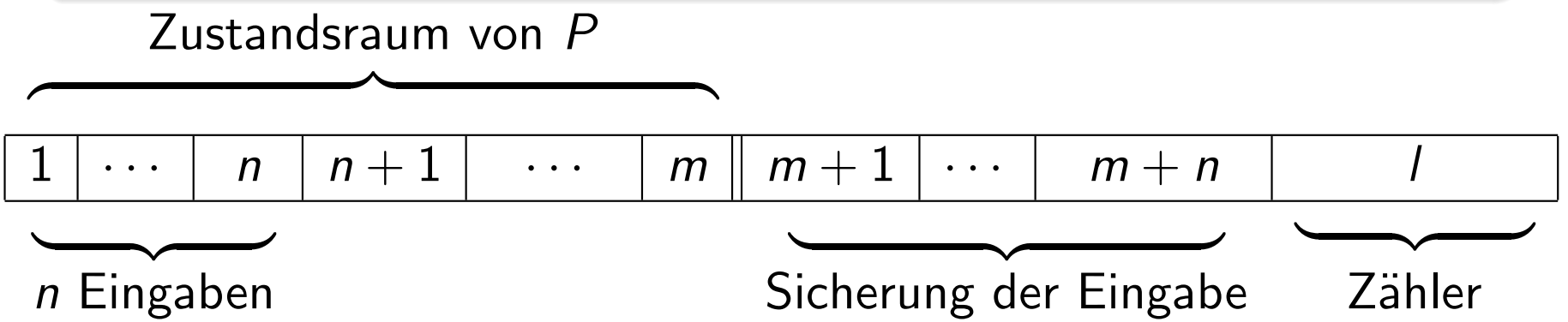
```
while X_1 ≠ 0 do
    X_l := X_l + 1;  X_1 := X_{m+1}; ...; X_n := X_{m+n}
    X_{n+1} := X_l;  X_{n+2} := 0; ...; X_m := 0;  α
od ;
X_1 := X_l
```

$\curvearrowright$ Q berechnet genau $\mu(f)$. ■

μ-rekursiv und WHILE berechenbar

Beweis (f μ-rekursiv $\curvearrowright$ f WHILE-berechenbar) (3):

Der Zustandsraum von Q wird wie folgt organisiert:



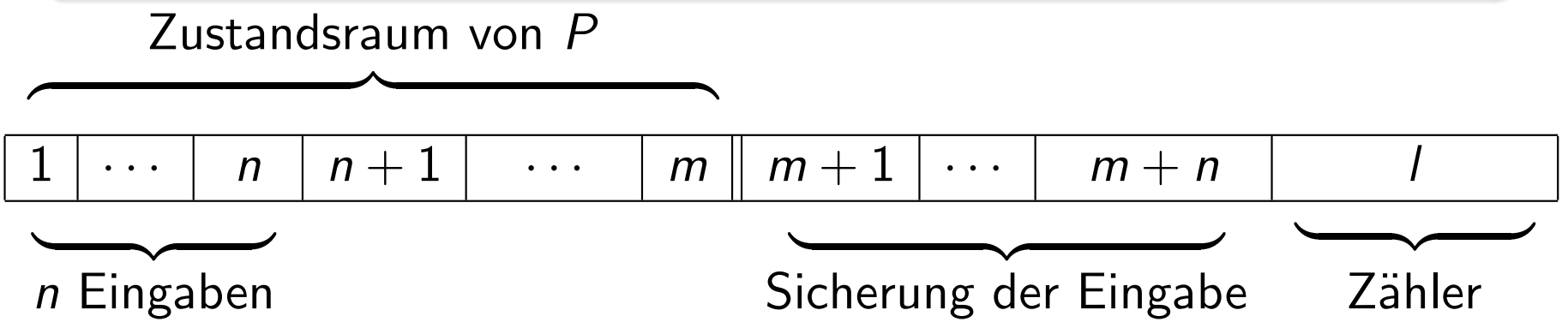
Sei: β = $X_{m+1} := X_1; \ldots; X_{m+n} := X_n;\ \alpha$
$\underline{\texttt{while}}\ X_1 \neq 0\ \underline{\texttt{do}}$
$X_l := X_l + 1;\ X_1 := X_{m+1}; \ldots; X_n := X_{m+n}$
$X_{n+1} := X_l;\ X_{n+2} := 0; \ldots; X_m := 0;\ \alpha$
$\underline{\texttt{od}}$;
$X_1 := X_l$

$\curvearrowright$ Q berechnet genau $\mu(f)$. ■

μ-rekursiv und WHILE berechenbar

Beweis (f μ-rekursiv $\curvearrowright$ f WHILE-berechenbar) (3):

Der Zustandsraum von Q wird wie folgt organisiert:



Sei: β = $X_{m+1} := X_1; \ldots; X_{m+n} := X_n;\ \alpha$

```
while X_1 ≠ 0 do
    X_l := X_l + 1;  X_1 := X_{m+1}; ...; X_n := X_{m+n}
    X_{n+1} := X_l;  X_{n+2} := 0; ...; X_m := 0;  α
od ;
X_1 := X_l
```

$\curvearrowright$ Q berechnet genau $\mu(f)$. ■

Sätze ohne Beweis

Normalformsatz von Kleene:

Jede μ-rekursive Funktion kann mit einem WHILE-Programm mit **höchstens einer** While-Schleife berechnet werden.

Entsprechend gilt, dass jede μ-rekursive Funktion mit genau **einer Minimalisierung** ausgedrückt werden kann.

Sätze ohne Beweis

Normalformsatz von Kleene:

Jede μ-rekursive Funktion kann mit einem WHILE-Programm mit **höchstens einer** While-Schleife berechnet werden.

Entsprechend gilt, dass jede μ-rekursive Funktion mit genau **einer Minimalisierung** ausgedrückt werden kann.

Satz:

f WHILE-berechenbar $\curvearrowright$ f Turing-berechenbar.

Beweisskizze:

- Simuliere WHILE-Programm mit Mehrband-TM
- i-tes Band $\hat{=}$ Variable x_i des WHILE-Programms
- Induktion über WHILE-Anweisungen

Die Sprache GOTO

Sprunganweisungen anstelle von Schleifen

- Charakteristisch für Maschinensprachen
- in Hochsprachen unerwünscht, da uneingeschränkter Gebrauch modularer Programmstrukturen verletzt
- Darstellung über Flussdiagramme

Syntax von GOTO

Definition:

- **Variablen**: $\mathcal{V} := \{X_i \mid i \geq 1\}$
- **Wertzuweisungen**: $\mathcal{W} := \{ \quad X_i := X_j + 1, \quad X_i := X_j - 1, \quad X_i := X_j, \quad X_i := 0 \quad \mid i, j \geq 1\}$
- **Befehle**: $\mathcal{B} := \{ \quad p : \alpha \ \underline{\texttt{goto}} \ q;$

$$p : \underline{\texttt{if}} \ X_i = 0 \ \underline{\texttt{then}} \ \underline{\texttt{goto}} \ q \ \underline{\texttt{else}} \ \underline{\texttt{goto}} \ r \mid$$

$$p, q, r \in \mathbb{N}, \alpha \in \mathcal{W}, i \in \mathbb{N} \}$$

Syntax von GOTO

Definition:

- **Variablen**: $\mathcal{V} := \{X_i \mid i \geq 1\}$
- **Wertzuweisungen**: $\mathcal{W} := \{ \quad X_i := X_j + 1, \quad X_i := X_j - 1, \quad X_i := X_j, \quad X_i := 0 \quad \mid i, j \geq 1\}$
- **Befehle**: $\mathcal{B} := \{ \quad p : \alpha \ \underline{\texttt{goto}} \ q;$

$p : \underline{\texttt{if}} \ X_i = 0 \ \underline{\texttt{then}} \ \underline{\texttt{goto}} \ q \ \underline{\texttt{else}} \ \underline{\texttt{goto}} \ r \mid$

$p, q, r \in \mathbb{N}, \alpha \in \mathcal{W}, i \in \mathbb{N} \}$

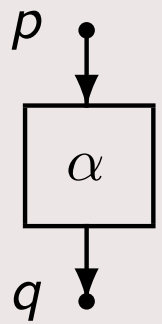
Syntax von GOTO

Definition:

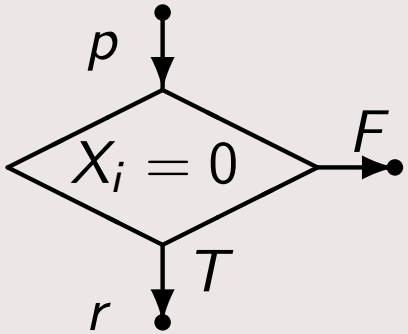
- **Variablen**: $\mathcal{V} := \{X_i \mid i \geq 1\}$
- **Wertzuweisungen**: $\mathcal{W} := \{ \quad X_i := X_j + 1,$
  $X_i := X_j - 1,$
  $X_i := X_j,$
  $X_i := 0 \qquad \mid i, j \geq 1\}$
- **Befehle**: $\mathcal{B} := \{ \quad p : \alpha \ \underline{\texttt{goto}} \ q;$
  $p : \underline{\texttt{if}} \ X_i = 0 \ \underline{\texttt{then}} \ \underline{\texttt{goto}} \ q$
  $\underline{\texttt{else}} \ \underline{\texttt{goto}} \ r \mid$
  $p, q, r \in \mathbb{N}, \alpha \in \mathcal{W}, i \in \mathbb{N} \}$

Flußdiagramme zu **GOTO**-Programmen (1)

Definition:



repräsentiert $p : \alpha$ `goto` q

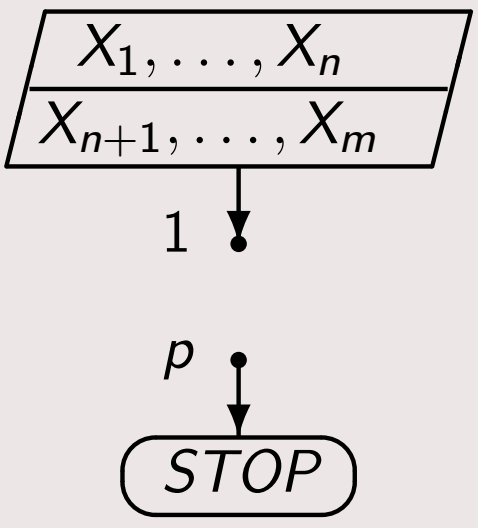


repräsentiert

p : `if` $X_i = 0$ `then goto` r `else goto` s

Flußdiagramme zu **GOTO**-Progrbammen (2)

Definition:



repräsentiert $\underline{\texttt{in}}(X_1, \ldots, X_n);$
$\underline{\texttt{var}}(X_{n+1}, \ldots, X_m)$

falls p keine Linksmarke im Programm

Beispiel:

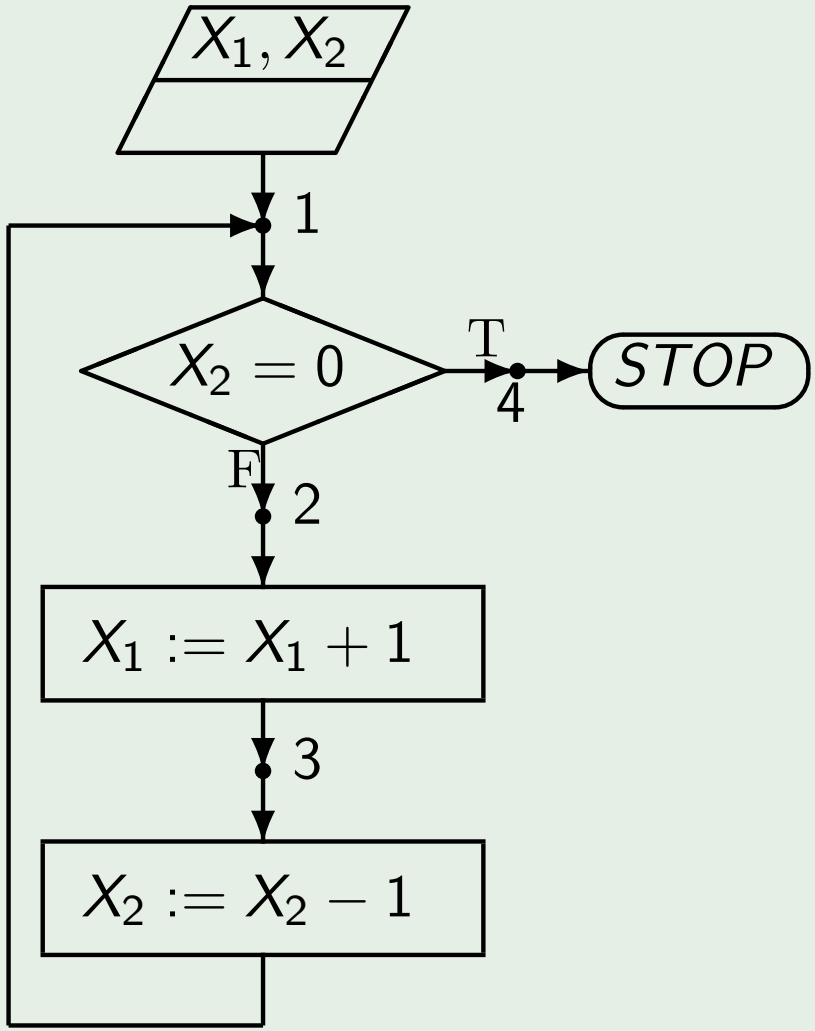
$\underline{\texttt{in}}(X_1, X_2)$; $\underline{\texttt{var}}()$;

$1:$ $\underline{\texttt{if}}$ $X_2 = 0$
$\underline{\texttt{then}}$ $\underline{\texttt{goto}}$ 4
$\underline{\texttt{else}}$ $\underline{\texttt{goto}}$ 2;

$2:$ $X_1 := X_1 + 1$
$\underline{\texttt{goto}}$ 3;

$3:$ $X_2 := X_2 - 1$
$\underline{\texttt{goto}}$ 1;

$\underline{\texttt{out}}$ X_1



Sätze ohne Beweis

Satz:

Jede GOTO-berechenbare Funktion ist μ-rekursiv.

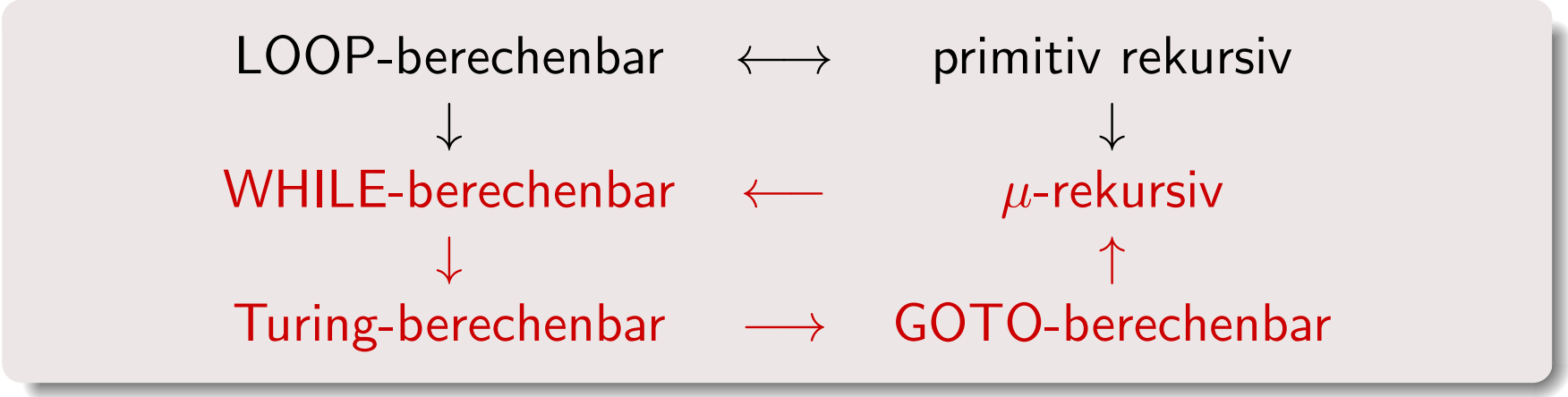
Satz

Jede Turing-berechenbare Funktion ist GOTO-berechenbar.

Berechenbarkeitszusammenhänge:

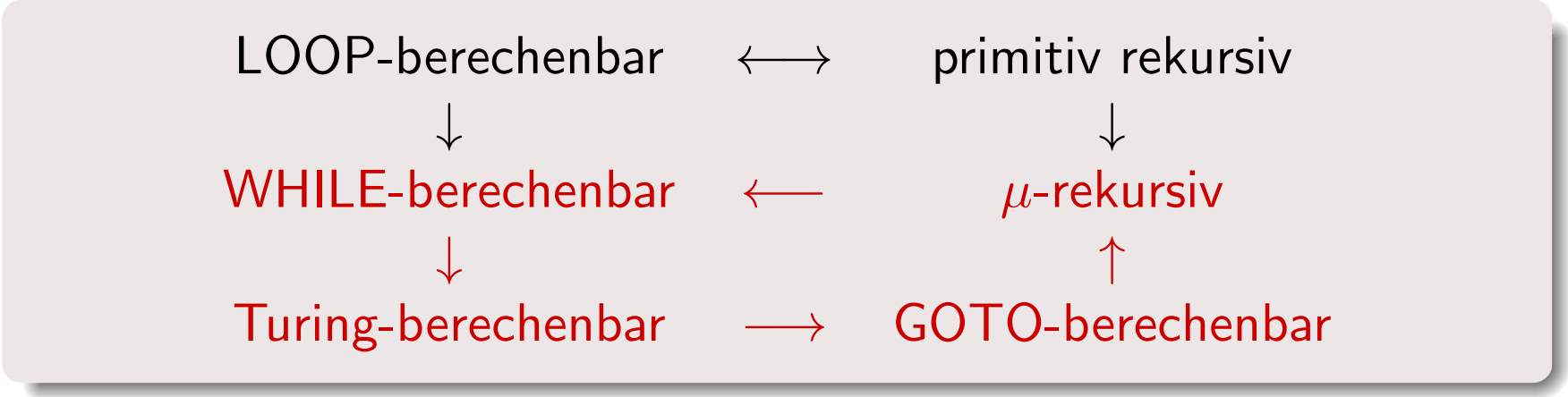
- Per Ringschluss folgt, dass Turing-, WHILE-, GOTO-Berechenbarkeit und μ-Rekursivität **äquivalente** Formalisierungen des Berechenbarkeitsbegriffs sind.
- Dies untermauert die Church'sche These.

Berechenbarkeitszusammenhänge:



- Per Ringschluss folgt, dass Turing-, WHILE-, GOTO-Berechenbarkeit und μ-Rekursivität **äquivalente** Formalisierungen des Berechenbarkeitsbegriffs sind.
- Dies untermauert die Church'sche These.

Berechenbarkeitszusammenhänge:



- Per Ringschluss folgt, dass Turing-, WHILE-, GOTO-Berechenbarkeit und μ-Rekursivität **äquivalente** Formalisierungen des Berechenbarkeitsbegriffs sind.
- Dies untermauert die Church'sche These.