Definition: Sei $g : \mathbb{N}^{k+1} \to \mathbb{N}$ eine Funktion. Dann ist der **beschränkte μ-Operator** wie folgt definiert:

$$\mu_{i<m} i(g(\underline{n}, i) = 0) := \begin{cases} 0 & \text{falls } m = 0 \text{ oder} \\ & g(\underline{n}, j) \neq 0 \text{ für} \\ & \forall j \text{ mit } 0 \leq j < m; \\ i_0 & \text{falls } g(\underline{n}, i_0) = 0 \wedge \\ & \forall j < i_0 \; g(\underline{n}, j) \neq 0 \wedge \\ & 0 \leq i_0 < m. \end{cases}$$

Informelle Bedeutung:

- Der beschränkte μ-Operator findet das kleinste i mit $i < m$ s.d. $g(\underline{n}, i) = 0$.
- Falls es kein solches i gibt, liefert der Operator den Wert 0.
- Beachte: der Wert 0 ergibt sich wenn
  - $g(\underline{n}, 0) = 0$; oder
  - g hat für kein $i_0 < m$ den Wert 0.

Satz: REK_{pr} ist abgeschlossen gegen beschränkte μ-Operatoren, d.h. wenn $g : \mathbb{N}^{k+1} \to \mathbb{N}$ primitiv rekursiv ist, so auch $f : \mathbb{N}^{k+1} \to \mathbb{N}$ mit $f(\underline{n}, m) = \mu_{i<m} i(g(\underline{n}, i) = 0)$.

Beweis: f kann mittels primitiver Rekursion und Fallunterscheidung dargestellt werden:

$$f(\underline{n}, 0) = 0$$

$$f(\underline{n}, m+1) = \begin{cases} 0 & \text{falls } \alpha_1; \\ m & \text{falls } \alpha_2; \\ f(\underline{n}, m) & \text{falls } \alpha_3, \end{cases}$$

wobei die Bedingungen α_i wie folgt sind:

α_1: $m = 0 \wedge g(\underline{n}, 0) = 0$;

α_2: $g(\underline{n}, m) = 0 \wedge f(\underline{n}, m) = 0 \wedge g(\underline{n}, 0) \neq 0 \wedge m > 0$;

α_3: $\neg\alpha_1 \wedge \neg\alpha_2$.

Damit kann $f(\underline{n}, m+1)$ mittels $h(\underline{n}, m, f(\underline{n}, m))$ geschrieben werden, wo $h(\underline{n}, m, o)$ die Bedingungen $\alpha_1, \alpha_2, \alpha_3$ kodiert.

Und zwar:

$$h(\underline{n}, m, o) = 0 \cdot h_1(\underline{n}, m, o) + m \cdot h_2(\underline{n}, m, o) + o \cdot h_3(\underline{n}, m, o)$$

mit

- $h_i(\underline{n}, m, f(\underline{n}, m)) \neq 0$ gdw α_i ist erfüllt.

Diese Funktionen sind wie folgt gegeben:

$$h_1(\underline{n}, m, o) = 1 \dot{-} (\pi^{k+2}_{k+1}(\underline{n}, m, o) + g(\underline{n}, 0));$$
$$h_2(\underline{n}, m, o) = 1 \dot{-} \big(g(\underline{n}, m) + o + (1 \dot{-} g(\underline{n}, 0)) + (1 \dot{-} m)\big);$$
$$h_3(\underline{n}, m, o) = 1 \dot{-} \big(h_1(\underline{n}, m, o) + h_2(\underline{n}, m, o)\big).$$

■

Bemerkung:

- Der beschränkte μ-Operator bleibt primitiv rekursiv, wenn statt $i < m$ die Bedingung $i \leq m$ verwendet wird.

Satz: Die folgenden Funktionen sind primitiv rekursiv:

$$t(n, m) := \begin{cases} 1 & \text{falls } m \; mod \; n = 0; \\ 0 & \text{sonst;} \end{cases}$$

$$pr(n) := \begin{cases} 1 & \text{falls } n \text{ prim;} \\ 0 & \text{sonst;} \end{cases}$$

$p(n) := m$, wo m ist n-te Primzahl;

$D(n, i) := max(\{j \mid n \; mod \; p(i)^j = 0\}$.

Beweis: Diese Funktionen können wie folgt dargestellt werden:

$t(n, m)$:

n teilt $m \iff \exists z \leq m$ s.d. $z \cdot n = m$
$\iff \prod_{z \leq m} |(z \cdot n, m)| = 0$

$\implies t(n, m) = 1 \dot{-} \prod_{z \leq m} |(z \cdot n, m)|$.

$pr(n)$:

n ist prim $\iff n \geq 2 \wedge \forall y < n \ \big((y = 0) \vee (y = 1) \vee (y$ teilt n nicht$)\big)$.

$\implies pr(n) = 1 \dot{-} \big((2 \dot{-} n) + \sum_{y<n}(y \cdot |(y,1)| \cdot t(y,n)\big)$.

$p(n)$: Es gilt:

- $p(0) = 0$ und $p(1) = 2$, d.h. die erste Primzahl ist 2.
- Die $(m+1)$-te Primzahl kann rekursiv bestimmt werden als die kleinste Zahl i s.d.
  - $i > p(m)$ und
  - i ist eine Primzahl.

  Weiters gilt nach Euklid: $i \leq p(m)! + 1$.

$\implies p(n) = \mu_{i \leq p(m)!+1}\Big(\big((p(m)+1) \dot{-} i\big) + (1 \dot{-} pr(i)) = 0\Big)$.

$D(n,i)$: Für $n = 0$ ist $D(n,i) = 0$, d.h. die Null hat keine Primteiler.

Für $n > 0$ kann $D(n,i)$ mittels beschränktem μ-Operator beschrieben werden:

$D(n,i) = \mu_{z \leq n}(t(p(i)^{z+1}, n) = 0)$. ■

Relationen zu LOOP-Programmen

- Im folgenden zeigen wir:

$$REK_{pr} = LOOP,$$

  - d.h. die Klasse der primitiv rekursiven Funktionen ist identisch mit der Klasse der LOOP-berechenbaren Funktionen.

- Für die Relation $LOOP \subseteq REK_{pr}$ verwenden wir **strukturelle Induktion** über den Aufbau von LOOP-Programmen.
  - Jedem LOOP-Konstrukt wird eine REK_{pr}-Funktion f zugeordnet s.d.:
    - Argument von f repräsentiert Registerbelegungen *vor* Abarbeitung des LOOP-Konstrukts;
    - Funktionswert von f repräsentiert die neuen Registerinhalte.
  - Dazu verwenden wir eine geeignete Gödelisierung die den Inhalt sämtlicher Register in *einer einzigen* Zahl speichert.

- Gödelisierungs-Methode: **Primzahlkodierung**
- Grundidee:
  - Betrachte LOOP-Programm mit Registern $x_1, \ldots, x_\ell$.
  - Aktuelle Inhalte n_i aller Register x_i wird in Zahl n kodiert s.d.:
    - n soll n_i-mal durch die i-te Primzahl teilbar sein, für $1 \leq i \leq \ell$;
    - n soll nicht durch die $(\ell+1)$-te Primzahl teilbar sein.
- Beispiel:
  - Betrachte ein LOOP-Programm mit Registern x_1 bis x_4.
  - Die aktuelle Belegung sei:

$$x_1 = 1 \qquad x_2 = 3$$
$$x_3 = 0 \qquad x_4 = 5$$

  $\implies$ Der Inhalt dieser Register kann mittels Gödelisierung als folgende Zahl kodiert werden:

$$p(1)^1 \cdot p(2)^3 \cdot p(3)^0 \cdot p(4)^5 = 2^1 \cdot 3^3 \cdot 5^0 \cdot 7^5 = 907\,578$$

Definition: Die **Kodierungsfunktionen**

$$K^k : \mathbb{N}^k \to \mathbb{N}$$

und die **Dekodierungsfunktionen**

$$D_i : \mathbb{N} \to \mathbb{N}$$

sind gegeben als:

$$K^k(n_1, \ldots, n_k) := \prod_{i \leq k} p(i)^{n_i};$$
$$D_i(n) := D(n,i).$$

Satz:

1. Die Funktionen K^k und D_i sind primitiv rekursiv.
2. $D_i(K^k(n_1, \ldots, n_k)) = n_i$.
3. $K^k(n_1, \ldots, n_k) = K^{k+1}(n_1, \ldots, n_k, 0)$.

Beweis:

1. Gilt, da Funktionen $p(\cdot)$, $D(\cdot,\cdot)$ und beschränkte Multiplikation primitiv rekursiv sind.
2. Gilt, da Primfaktorzerlegung eindeutig.
3. Folgt wegen $n^0 = 1$, für alle $n \in \mathbb{N}$. ■

Notation:

- Wir schreiben

$$\begin{aligned}\langle n_1, \ldots, n_k \rangle &:= K^k(n_1, \ldots, n_k);\\ (n)_i &:= D_i(n).\end{aligned}$$

- Für $k = 0$ setzen wir $\langle\rangle = 1$ und $(\langle\rangle)_i = 0$, für alle $i \in \mathbb{N}$.

Satz: REK_{pr} ist abgeschlossen gegenüber simultane primitive Rekursion, d.h., wenn $g_1, \ldots, g_r$ und $h_1, \ldots, h_r$ primitiv rekursiv sind, so auch $f_1, \ldots, f_r$ mit

$$\begin{aligned}f_1(\underline{n}, 0) &= g_1(\underline{n})\\ &\vdots\\ f_r(\underline{n}, 0) &= g_r(\underline{n})\\ f_1(\underline{n}, m+1) &= h_1(\underline{n}, m, f_1(\underline{n}, m), \ldots, f_r(\underline{n}, m))\\ &\vdots\\ f_r(\underline{n}, m+1) &= h_r(\underline{n}, m, f_1(\underline{n}, m), \ldots, f_r(\underline{n}, m))\end{aligned}$$

Beweis: Sei $f(\underline{n}, m) := \langle f_1(\underline{n}, m), \ldots, f_r(\underline{n}, m)\rangle$.

Dann ist f primitiv rekursiv, da f sich wie folgt berechnen läßt:

$$f(\underline{n}, 0) = \langle g_1(\underline{n}), \ldots, g_r(\underline{n})\rangle;$$

$$\begin{aligned}f(\underline{n}, m+1) = \\ \langle h_1(\underline{n}, m, (f(\underline{n}, m))_1, \ldots, (f(\underline{n}, m))_r), \ldots,\\ h_r(\underline{n}, m, (f(\underline{n}, m))_1, \ldots, (f(\underline{n}, m))_r)\rangle.\end{aligned}$$

Damit lassen sich die Funktionen $f_i(\underline{n}, m)$, für alle $1 \leq i \leq r$, primitiv rekursiv ausdrücken mittels

$$f_i(\underline{n}, m) = (f(\underline{n}, m))_i.$$

■

Lemma 1 Es gilt $REK_{pr} \subseteq LOOP$, d.h., jede primitiv rekursive Funktion ist LOOP-berechenbar.

Beweis: Wir zeigen folgende Eigenschaften:

1. Die Basisfunktionen
   - 0 (nullstellige Nullfunktion),
   - N (Nachfolgerfunktion) und
   - π_i^k (Projektionsfunktionen)

   sind LOOP-berechenbar;
2. LOOP ist abgeschlossen gegenüber simultanem Einsetzen und primitiver Rekursion, d.h.,
   - wenn die Teilfunktionen g, $h_1, \ldots, h_r$ und h LOOP-berechenbar sind, so auch $sub_r(g, h_1, \ldots, h_r)$ und $pr(g, h)$.

ad 1) Simulation der Basisfunktionen.

$0 : \mathbb{N} \to \mathbb{N}$: Diese Funktion wird vom LOOP-Programm $P_0 :\equiv$ NOP berechnet.
- P_0 bekommt keine Eingabewerte und hat einen Ausgabewert in Register 1, nämlich 0.

$N : \mathbb{N} \to \mathbb{N}$: Simulation mittels folgendem LOOP-Programm P_N:

LOOP x_1 DO $x_2 := x_2 + 1$ END;

$x_2 := x_2 + 1$

$\pi_i^k : \mathbb{N}^k \to \mathbb{N}$: Simulation mittels LOOP-Programm $P_{\pi_i^k}$ mit $(k+1)$ Register:

LOOP x_i DO $x_{k+1} := x_{k+1} + 1$ END

ad 2) Simulation des simulatanen Einsetzens und der primitiven Rekursion mittels LOOP.

(a) Wir zeigen zunächst die Abgeschlossenheit bezüglich simultanem Einsetzen.

Seien $g : \mathbb{N}^r \to \mathbb{N}$ und $h_1, \ldots, h_r : \mathbb{N}^k \to \mathbb{N}$ LOOP-berechenbar.

Wir zeigen: die Funktion $f : \mathbb{N}^k \to \mathbb{N}$ mit

$$f(\underline{n}) = g(h_1(\underline{n}), \ldots, h_r(\underline{n}))$$

ist LOOP-berechenbar (wo $\underline{n} = (n_1, \ldots, n_k)$).

Sei P_α ein LOOP-Programm, daß die Funktion α berechnet, für $\alpha \in \{g, h_1, \ldots, h_r\}$.

Wir konstruieren ein LOOP-Programm P_f, daß f berechnet.

Nach Definition einer Registermaschine ist die Registerbelegung für die Programme P_α, $\alpha \in \{g, h_1, \ldots, h_r\}$, wie folgt ($i_\alpha = r$ für $\alpha = g$ und $i_\alpha = k$ für $\alpha = h_i$):

- $x_1, \ldots, x_{i_\alpha}$: Eingaberegister
- $x_{i_\alpha+1}$: Ausgaberegister
- $x_{i_\alpha+j}$, $j > 1$: restliche Register

Um Belegungskonflikte bei Verwendung der Programme P_α für das zu konstruierende Programm P_f auszuschließen, definieren wir:

- P'_{h_i} entstehe aus P_{h_i}, indem die Register x_{k+j} in x_{k+j+r} unbenannt werden, für $j \geq 1$.
  $\Rightarrow$ Register $x_{k+1}, \ldots, x_{k+r}$ werden zum Speichern der Ergebnisse der Programme P_{h_i} freigehalten.
- P'_g entstehe aus P_g durch Umbenennen aller Register x_j in x_{j+k}.
  - P'_g hat Eingaberegister $x_{k+1}, \ldots, x_{k+r}$ und schreibt Ausgabe in x_{k+r+1}.
    $\Rightarrow$ Register $x_1, \ldots, x_k$ bleiben für die Eingabe von f reserviert.

Übersicht über die neue Registerverteilung:

- $x_1, \ldots, x_k$:
  - Eingabe für P_f und P'_{h_i}.
- $x_{k+1}, \ldots, x_{k+r}$:
  - Eingabe für P'_g.
  - x_{k+i} wird als Zwischenregister für die Ausgabe von P'_{h_i} verwendet ($1 \leq i \leq r$).
  - x_{k+1} ist Ausgabe von P_f.
- x_{k+r+1} :
  - Ausgabe von P'_g und aller P'_{h_i}.
- x_{k+r+j}, $j > 1$:
  - sonstige Register von P'_g und aller P'_{h_i}.

$\Longrightarrow$ Programm P_f sieht wie folgt aus:

P'_{h_1}; $x_{k+1} := x_{k+r+1}$; $x_{k+r+1} := 0$;

$\vdots$

P'_{h_i}; $x_{k+i} := x_{k+r+1}$; $x_{k+r+1} := 0$;

$\vdots$

P'_{h_r}; $x_{k+r} := x_{k+r+1}$; $x_{k+r+1} := 0$;

P'_g; $x_{k+1} := x_{k+r+1}$; $x_{k+2} := 0; \ldots x_{k+r+1} := 0$

(b) Wir zeigen als nächstes die Abgeschlossenheit bezüglich primitiver Rekursion.

Seien $g : \mathbb{N}^k \to \mathbb{N}$ und $h : \mathbb{N}^{k+2} \to \mathbb{N}$ LOOP-berechenbar.

Wir zeigen: die Funktion $f : \mathbb{N}^{k+1} \to \mathbb{N}$ mit

$$\begin{aligned} f(\underline{n}, 0) &= g(\underline{n}) \\ f(\underline{n}, m+1) &= h(\underline{n}, m, f(\underline{n}, m)) \end{aligned}$$

ist LOOP-berechenbar (für $\underline{n} = (n_1, \ldots, n_k)$).

Beachte:

- Zur Berechnung von $f(\underline{n}, m+1)$ ist die Berechnung der Werte $f(\underline{n}, 0), f(\underline{n}, 1), \ldots, f(\underline{n}, m)$ notwendig.
- Wir werden dies mit einer LOOP-Schleife simulieren!

Sei P_g ein LOOP-Programm, daß die Funktion g berechnet, und sei P_h ein LOOP-Programm, daß h berechnet.

Wir konstruieren ein LOOP-Programm P_f, daß f berechnet.

Wir modifizieren zunächst P_g, um Belegungskonflikte zu vermeiden:

- P'_g entstehe aus P_g indem
  - x_{k+1} nicht verwendet wird und
  - wo stattdessen die Ausgabe in x_{k+2} gespeichert wird.

Weiters sei x_{store} ein neues Register, das weder von P'_g noch von P_h verwendet wird.

- Dient zur Speicherung der maximalen Anzahl von Schleifen für Berechnung von $f(\underline{n}, m+1)$.

$\Longrightarrow$ Das Programm P_f wird wie folgt konstruiert:

$x_{store} := x_{k+1}$; % x_{k+1} *enthält den*
% *Eingabewert* m
% (*Anzahl der Schleifen*)

$x_{k+1} := 0$; % *aktueller Schleifenwert;*
% *anfangs* 0

P'_g; % *berechnet* $f(\underline{n}, 0)$*;*
% *Ergebnis in Register* x_{k+2}

LOOP x_{store} DO

P_h; % *berechnet* $f(\underline{n}, x_{k+1}+1) =$
% $h(\underline{n}, x_{k+1}, f(\underline{n}, x_{k+1}))$

$x_{k+2} := x_{k+2+1}$; % $x_{k+2} = f(\underline{n}, x_{k+1}+1)$

$x_{k+2+1} := 0$;

$x_{k+1} := x_{k+1} + 1$; % $m := m+1$

END

$x_{store} := 0$

Damit haben wir gezeigt, daß jede primitiv rekursive Funktion LOOP-berechenbar ist. ■

Lemma 2 Es gilt $LOOP \subseteq \mathcal{REK}_{pr}$, d.h., jede LOOP-berechenbare Funktion ist primitiv rekursiv.

Beweis: Sei $g : \mathbb{N}^k \to \mathbb{N}$ eine LOOP-berechenbare Funktion welche durch ein LOOP-Programm P bestimmt ist.

Wir konstruieren eine primitiv rekursive Funktion $f_P : \mathbb{N} \to \mathbb{N}$ s.d.

$$\begin{aligned} g(n_1, \ldots, n_k) &= (f_P(\langle n_1, \ldots, n_k \rangle))_{k+1} \\ &= D_{k+1}(f_P(K^k(n_1, \ldots, n_k))) \\ &= D(f_P(K^k(n_1, \ldots, n_k)), k+1) \end{aligned}$$

f_P muß geeignet konstruiert sein, s.d. LOOP-Befehle der Form

LOOP x_i DO P' END

korrekt ausgeführt werden, d.h.

- falls $x_i = n$ zu Beginn der ersten Ausführung des Schleifenrumpfes ist, so soll P' n-mal ausgeführt werden.

Dies wird durch geeignete "Hilfsvariablen" realisiert.

Genauer:

- Angenommen P verwendet ausschließlich Register aus der Menge $\{x_1, \ldots, x_\ell\}$ und enthält m LOOP-Anweisungen.
  $\Longrightarrow$ f_P operiert mit
  - "Variablen" $n_1, \ldots, n_\ell$ für die Register $x_1, \ldots, x_\ell$;
  - "Hilfsvariablen" $n_{\ell+j}$, $1 \leq j \leq m$, zur Speicherung wie oft die j-te LOOP-Anweisung ausgeführt werden soll.
- f_P verwaltet Werte $(n_1, \ldots, n_\ell, n_{\ell+1}, \ldots, n_{\ell+m})$ mittels Kodierung $\langle n_1, \ldots, n_\ell, n_{\ell+1}, \ldots, n_{\ell+m} \rangle$.
- Zu Anfang und Ende der Simulation von P ist $n_{\ell+j} = 0$ für $1 \leq j \leq m$.
  $\Longrightarrow$ Also gilt für Anfang und Ende der Berechnung von f_P:
  $\langle n_1, \ldots, n_\ell, n_{\ell+1}, \ldots, n_{\ell+m} \rangle = \langle n_1, \ldots, n_\ell \rangle$
- Insgesamt muß somit gelten:
  $f_P(\langle n_1, \ldots, n_\ell \rangle) = \langle n'_1, \ldots, n'_\ell \rangle \Longleftrightarrow$
  P gestartet mit Werten n_i in Register x_i hält mit Registerinhalten n'_i in x_i, für $1 \leq i \leq \ell$.

Wir definieren f_P mittels Induktion über den Aufbau von LOOP-Programmen.

- D.h., für jeden Punkt der rekursiven Definition eines LOOP-Programmes geben wir eine primitiv rekursive Funktion an, die dasselbe berechnet.

$P \equiv x_i := x_i \pm 1$:

Es muss hier gelten:

$$f_P(n) = \langle (n)_1, \ldots, (n)_{i-1}, (n)_i \pm 1, \ldots, (n)_{\ell+m} \rangle$$

- Für $P \equiv x_i := x_i + 1$ setzen wir:
  $$f_P(n) = n \cdot p(i).$$
- Für $P \equiv x_i := x_i - 1$ konstruieren wir f_P wie folgt.
  - Sei $DIV(n, m) = \mu_{i \leq n} i(|n, m \cdot i| = 0)$, d.h., man sucht das kleinste i mit $m \cdot i = n$.
  - Dann setzen wir:
  $$f_P(n) = \begin{cases} n & \text{falls } D(n, i) = 0 \\ DIV(n, p(i)) & \text{sonst} \end{cases}$$

$P =$ LOOP x_i DO P_1 END :

Der betrachtete LOOP-Befehl sei der j-te, und für dessen Schleifenzähler die Hilfsvariable $n_{\ell+j}$ benutzt wird.

Wir simulieren diesen Befehl mittels zwei Funktionen: $f_P(n) = f_2(f_1(n))$.

- Funktion f_1 initialisiert $(n)_{\ell+j}$ mit dem Wert, den x_i bei Betreten der Schleife hat:

$$f_1(n) = \langle (n)_1, \ldots, (n)_{\ell+j-1}, (n)_i, (n)_{\ell+j+1}, \ldots, (n)_{\ell+m} \rangle$$

$$\Longrightarrow \quad f_1(n) = n \cdot p(\ell+j)^{(n)_i}$$

- f_2 führt die Simulation des LOOP-Befehles aus:
  - Sei f_{P_1} die Funktion die P_1 berechnet und
  - $n_{\ell+j}$ der Schleifenzähler, der im Laufe der Rekursion heruntergezählt wird.
  - In jedem Rekursionsschritt wird auf das Ergebnis des rekursiven Aufrufs die Funktion f_{P_1} angewendet.

Es muß gelten:

$$f_2(n) = \begin{cases} n & \text{falls } n_{\ell+j} = 0 \\ f_{P_1}\Big(f_2\big(\langle (n)_1, \ldots, (n)_{\ell+j-1}, \\ \qquad (n)_{\ell+j} - 1, (n)_{\ell+j+1}, \\ \qquad \ldots, (n)_{\ell+m} \rangle\big)\Big) & \text{sonst} \end{cases}$$

Anders ausgedrückt:

$$f_2(n) = \begin{cases} n & \text{falls } n_{\ell+j} = 0 \\ f_{P_1}\Big(f_2\big(DIV(n, p(\ell+j))\big)\Big) & \\ \qquad \text{sonst} & \end{cases}$$

f_2 kann mittels Hilfsfunktion F auf das Standardschema der primitiven Rekursion zurückgeführt werden:

$$F(n,0) = n$$
$$F(n,m+1) = f_{P_1}\big(DIV(F(n,m), p(\ell+j))\big)$$
$$\Longrightarrow \quad f_2(n) = F(n, D(n, \ell+j)).$$

$P \equiv P_1; P_2$: Dann sei $f_P(n) = f_{P_2}(f_{P_1}(n))$. ■

Lemma 1 und 2 impliziert:

Satz: $LOOP = REK_{pr}$.