Primitiv rekursive und μ-rekursive Funktionen

LOOP-, WHILE- und GOTO-Programme sind vereinfachte imperative Programme und stehen für imperative Programmiersprachen, bei denen Programme als Folgen von Befehlen aufgefaßt werden.

Parallel dazu gibt es jedoch auch funktionale Programme, deren Hauptbestandteil die Definition rekursiver Funktionen ist. Es gibt auch Berechnungsbegriffe, die sich eher an funktionalen Programmen orientieren.

Zum Beispiel:

- λ-Kalkül (Alonzo Church, 1932)
- μ-rekursive und primitiv rekursive Funktionen (werden hier in der Vorlesung betrachtet)

Primitiv rekursive Funktionen

Wir definieren nun Klassen von (totalen) Funktionen der Form $f : \mathbb{N}^k \to \mathbb{N}$.

Definition primitiv rekursive Funktionen

Die Klasse der primitiv rekursive Funktionen ist induktiv wie folgt definiert:

- Alle konstanten Funktionen der Form $k_m : \mathbb{N} \to \mathbb{N} : n \mapsto m$ (für ein festes $m \in \mathbb{N}$) sind primitiv rekursiv.
- Alle Projektionen der Form $\pi_i^k : \mathbb{N}^k \to \mathbb{N} : (n_1, \ldots, n_k) \mapsto n_i$ sind primitiv rekursiv.
- Die Nachfolgerfunktion $s : \mathbb{N} \to \mathbb{N} : n \mapsto n + 1$ ist primitiv rekursiv.
- Wenn $g : \mathbb{N}^k \to \mathbb{N}$ und $f_1, \ldots, f_k : \mathbb{N}^r \to \mathbb{N}$ primitiv rekursiv sind, dann ist auch die Abbildung $f : \mathbb{N}^r \to \mathbb{N}$ mit

$$f(n_1, \ldots, n_r) = g(f_1(n_1, \ldots, n_r), \ldots, f_k(n_1, \ldots, n_r))$$

primitiv rekursiv (Einsetzung/Komposition),

Primitiv rekursive Funktionen

Definition primitiv rekursive Funktionen (Fortsetzung)

- Jede Funktion f, die durch primitive Rekursion aus primitiv rekursiven Funktionen entsteht, ist primitiv rekursiv.

  Das heißt $f\colon \mathbb{N}^{k+1} \to \mathbb{N}$ muss folgende Gleichungen erfüllen (für primitiv rekursive Funktionen $g\colon \mathbb{N}^k \to \mathbb{N}$, $h\colon \mathbb{N}^{k+2} \to \mathbb{N}$):

$$\begin{aligned} f(0, n_1, \ldots, n_k) &= g(n_1, \ldots, n_k) \\ f(n+1, n_1, \ldots, n_k) &= h(f(n, n_1, \ldots, n_k), n, n_1, \ldots, n_k) \end{aligned}$$

Anschaulich: bei primitiver Rekursion wird die Definition von $f(n+1, \ldots)$ zurückgeführt auf $f(n, \ldots)$. Das bedeutet, dass primitive Rekursion immer terminiert.

$\rightsquigarrow$ Berechnungsmodell analog zu LOOP-Programmen.

Primitiv rekursive Funktionen

Beispiele für primitiv rekursive Funktionen:

Additionsfunktion

Die Funktion $add : \mathbb{N}^2 \to \mathbb{N} : (m, n) \mapsto m + n$ ist primitiv rekursiv.

$$\begin{aligned} add(0, n) &= n \\ add(m + 1, n) &= s(add(m, n)) \end{aligned}$$

Multiplikationsfunktion

Die Funktion $mult : \mathbb{N}^2 \to \mathbb{N} : (m, n) \mapsto m \cdot n$ ist primitiv rekursiv.

$$\begin{aligned} mult(0, n) &= 0 \\ mult(m + 1, n) &= add(mult(m, n), n) \end{aligned}$$

Primitiv rekursive Funktionen

Dekrementierung

Die Funktion $dec : \mathbb{N} \to \mathbb{N} : n \mapsto \max(0, n-1)$ ist primitiv rekursiv.

$$\begin{aligned} dec(0) &= 0 \\ dec(n+1) &= n \end{aligned}$$

Subtraktion

Die Funktion $sub : \mathbb{N}^2 \to \mathbb{N} : (m, n) \mapsto \max\{0, m-n\}$ ist primitiv rekursiv.

$$\begin{aligned} sub(m, 0) &= m \\ sub(m, n+1) &= dec(sub(m, n)) \end{aligned}$$

Primitiv rekursive Funktionen

Binomialkoeffizient

Die Funktion $\mathbb{N} \to \mathbb{N} : n \mapsto \binom{n}{2}$ ist primitiv rekursiv.

$$\binom{0}{2} = 0$$
$$\binom{n+1}{2} = \binom{n}{2} + n$$

Durch Komposition folgt, dass auch die Abbildung

$$c : \mathbb{N}^2 \to \mathbb{N} : (m, n) \mapsto m + \binom{m+n+1}{2}$$

primitiv rekursiv ist.

Primitiv rekursive Funktionen

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

	$m = 0$	1	2	3	4
$n = 0$	0	2	5	9	14
1	1	4	8	13	19
2	3	7	12	18	25
3	6	11	17	24	32
4	10	16	23	31	40

Beachte: $c(m, n) = m + \sum_{i=1}^{m+n} i$.
Die Abbildung c ist eine Bijektion von $\mathbb{N}^2$ nach $\mathbb{N}$ (Paarungsfunktion).

Die Funktion c kann verwendet werden, um beliebige $(k+1)$-Tupel von natürlichen Zahlen durch eine Zahl zu kodieren:

$$\langle n_0 \rangle = c(n_0, 0) \qquad \langle n_0, n_1, \ldots, n_k \rangle = c(n_0, \langle n_1, n_2, \ldots, n_k \rangle)$$

Die Abbildung $(n_0, n_1, \ldots, n_k) \mapsto \langle n_0, n_1, \ldots, n_k \rangle$ ist dann auch primitiv rekursiv (für jedes feste k).

Primitiv rekursive Funktionen

Es seien $\ell : \mathbb{N} \to \mathbb{N}$ und $r : \mathbb{N} \to \mathbb{N}$ die eindeutigen Funktionen mit:

$$\forall m, n \in \mathbb{N} : \ell(c(m, n)) = m \text{ und } r(c(m, n)) = n$$

Wir werden nun zeigen, dass die Funktionen ℓ und r ebenfalls primitiv rekursiv sind. Mit diesen lassen sich dann auch primitiv rekursive Dekodierfunktionen für kodierte $(k+1)$-Tupel definieren, die $d_i(\langle n_0, n_1, \ldots, n_k \rangle) = n_i$ erfüllen:

$$\begin{aligned} d_0(n) &= \ell(n) \\ d_1(n) &= \ell(r(n)) \\ &\vdots \\ d_k(n) &= \ell(r^k(n)) \end{aligned}$$

Primitiv rekursive Funktionen

Definition beschränkter min-Operator

Sei $P : \mathbb{N}^{k+1} \to \{0, 1\}$ ein Prädikat und $q : \mathbb{N}^{k+1} \to \mathbb{N}$ Funktion mit

$$q(n, \bar{n}) = \begin{cases} 0 & \text{falls } P(i, \bar{n}) = 0 \text{ für alle } 0 \leq i \leq n \\ \min\{i \mid P(i, \bar{n}) = 1\} & \text{sonst.} \end{cases}$$

Dann sagen wir, daß q durch den beschränkten min-Operator aus P hervorgeht.

Lemma

Wenn P primitiv rekursiv ist, dann ist auch q primitiv rekursiv.

Primitiv rekursive Funktionen

Lemma

Wenn P primitiv rekursiv ist, dann ist auch q primitiv rekursiv.

Wir zeigen dies zunächst für die Funktion $q' : \mathbb{N}^{k+1} \to \mathbb{N}$ mit

$$q'(n, \bar{n}) = \begin{cases} 0 & \text{falls } P(i, \bar{n}) = 0 \text{ für alle } 0 \leq i \leq n \\ \min\{i \mid P(i, \bar{n}) = 1\} + 1 & \text{sonst} \end{cases}$$

$$\begin{aligned} q'(0, \bar{n}) &= P(0, \bar{n}) \\ q'(n+1, \bar{n}) &= q'(n, \bar{n}) + (1 - q'(n, \bar{n})) \cdot P(n+1, \bar{n}) \cdot (n+1) \end{aligned}$$

Es gilt $q(n, \bar{n}) = q'(n, \bar{n}) - 1$ □

Primitiv rekursive Funktionen

Definition beschränkter Existenzquantor

Seien $P, Q : \mathbb{N}^{k+1} \to \{0,1\}$ Prädikate mit

$$Q(n, \bar{n}) = \begin{cases} 1 & \text{falls } \exists x \leq n : P(x, \bar{n}) = 1 \\ 0 & \text{sonst.} \end{cases}$$

Dann sagen wir, daß Q durch den beschränkten Existenzquantor aus P hervorgeht.

Lemma

Wenn P primitiv rekursiv ist, dann ist auch Q primitiv rekursiv.

Die Funktion $q' : \mathbb{N}^{k+1} \to \mathbb{N}$ mit

$$q'(n, \bar{n}) = \begin{cases} 0 \text{ falls } P(i, \bar{n}) = 0 \text{ für alle } 0 \leq i \leq n \\ \min\{i \mid P(i, \bar{n}) = 1\} + 1 \text{ sonst} \end{cases}$$

ist primitiv rekursiv und es gilt $Q(n, \bar{n}) = 1 - (1 - q'(n, \bar{n}))$

Primitiv rekursive Funktionen

Nun können wir zeigen, dass die Umkehrfunktionen ℓ und r von $c : \mathbb{N}^2 \to \mathbb{N}$ primitiv rekursiv sind.

Das Prädikat $C : \mathbb{N}^3 \to \{0, 1\}$ mit „$C(x, y, z) = 1$ gdw. $c(x, y) = z$“ ist primitiv rekursiv:

$$C(x, y, z) = \Big(1 - (c(x, y) - z)\Big) \cdot \Big(1 - (z - c(x, y))\Big).$$

Deshalb sind auch die Funktionen ℓ' und r' mit

$$\begin{aligned} \ell'(k, m, n) &= \min\{x \leq k \mid \exists y \leq m : C(x, y, n) = 1\} \\ r'(k, m, n) &= \min\{y \leq k \mid \exists x \leq m : C(x, y, n) = 1\} \end{aligned}$$

primitiv rekursiv.

Schließlich gilt:

$$\ell(n) = \ell'(n, n, n) \text{ und } r(n) = r'(n, n, n).$$

Primitiv rekursive Funktionen

Lemma 20 (LOOP-Programm $\rightarrow$ primitiv rekursiv)

Jede LOOP-berechenbare Funktion $f : \mathbb{N}^r \rightarrow \mathbb{N}$ ist primitiv rekursiv.

Beweis:
Es gibt $k \geq r$ und ein LOOP-Programm P, in dem keine Variable x_i mit $i \geq k$ vorkommt, mit

$$\forall n_0, \ldots, n_{r-1} \in \mathbb{N} : f(n_0, \ldots, n_{r-1}) = \pi_0([P]_k(n_0, \ldots, n_{r-1}, 0, \ldots, 0)).$$

Definiere die Funktion $g_P : \mathbb{N} \rightarrow \mathbb{N}$ mit

$$\forall n_0, \ldots, n_{k-1} \in \mathbb{N} : g_P(\langle n_0, \ldots, n_{k-1} \rangle) = \langle [P]_k(n_0, \ldots, n_{k-1}) \rangle.$$

Wir werden zeigen, daß g_P primitiv rekursiv ist.
Wegen $f(n_0, \ldots, n_{r-1}) = d_0(g_P(\langle n_0, \ldots, n_{r-1}, 0, \ldots, 0 \rangle))$ ist dann auch f primitiv rekursiv.

Primitiv rekursive Funktionen

Fall 1. $P = (x_i := x_j \theta d)$:

$$g_P(n) = \langle d_0(n), \ldots, d_{i-1}(n), d_j(n) \theta d', d_{i+1}(n), \ldots, d_{k-1}(n) \rangle$$

mit $d' = d = k_d(n)$ falls $d \in \mathbb{N}$ und $d' = d_\ell(n)$ falls $d = x_\ell$.

Fall 2. $P = (Q; R)$:

$$g_P(n) = g_R(g_Q(n)).$$

Fall 3. $P = (\textsc{Loop}\ x_i\ \textsc{Do}\ Q\ \textsc{End})$:

Definiere zunächst die primitiv rekursive Funktion h durch

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

Dann gilt also $h(n, m) = g_Q^n(m)$.

Es gilt somit

$$g_P(x) = h(d_i(x), x).$$

Primitiv rekursive Funktionen

Satz 21 (LOOP-Programm $\leftrightarrow$ primitiv rekursiv)

Eine Funktion $f : \mathbb{N}^r \to \mathbb{N}$ ist genau dann LOOP-berechenbar, wenn sie primitiv rekursiv ist.

Es reicht die Implikation „$\Leftarrow$“ zu beweisen.Durch Induktion über den Aufbau von f zeigen wir, dass f LOOP-berechenbar ist.

Fall 1: f ist eine der Basisfunktionen (konstante Funktionen, Projektionen, Nachfolgerfunktion).

Dann ist f offensichtlich LOOP-berechenbar.

Fall 2: Es gibt primitiv rekursive Funktionen $g, f_1, \ldots, f_k$ mit

$$f(n_1, \ldots, n_r) = g(f_1(n_1, \ldots, n_r), \ldots, f_k(n_1, \ldots, n_r)).$$

Nach Induktionsvoraussetzung existieren LOOP-Programme $G, F_1, \ldots, F_k$, die $g, f_1, \ldots, f_k$ berechen.

Dann berechnet das folgende LOOP-Programm die Funktion f:

Primitiv rekursive Funktionen

$y_0 := x_0; \ldots; y_{r-1} := x_{r-1}$; (sichern der Argumente)
F_1;
$z_0 := x_0$; (jetzt gilt $z_0 = f_1(x_0, \ldots, x_{r-1})$)
$x_0 := y_0; \ldots; x_{r-1} := y_{r-1}$; (rekonstruieren der Argumente)
F_2;
$z_1 := x_0$; (jetzt gilt $z_1 = f_2(x_0, \ldots, x_{r-1})$)
$\vdots$
$x_0 := y_0; \ldots; x_{r-1} := y_{r-1}$; (rekonstruieren der Argumente)
F_k;
$z_{k-1} := x_0$; (jetzt gilt $z_{k-1} = f_k(x_0, \ldots, x_{r-1})$)
$x_0 := z_0; \ldots; x_{k-1} := z_{k-1}$; (jetzt gilt $x_i = f_{i+1}(x_0, \ldots, x_{r-1})$)
G

Primitiv rekursive Funktionen

Fall 3: f entsteht aus g und h durch primitive Rekursion.

Es gibt also primitiv rekursive Funktionen $g\colon \mathbb{N}^{r-1} \to \mathbb{N}$ und $h\colon \mathbb{N}^{r+1} \to \mathbb{N}$ mit

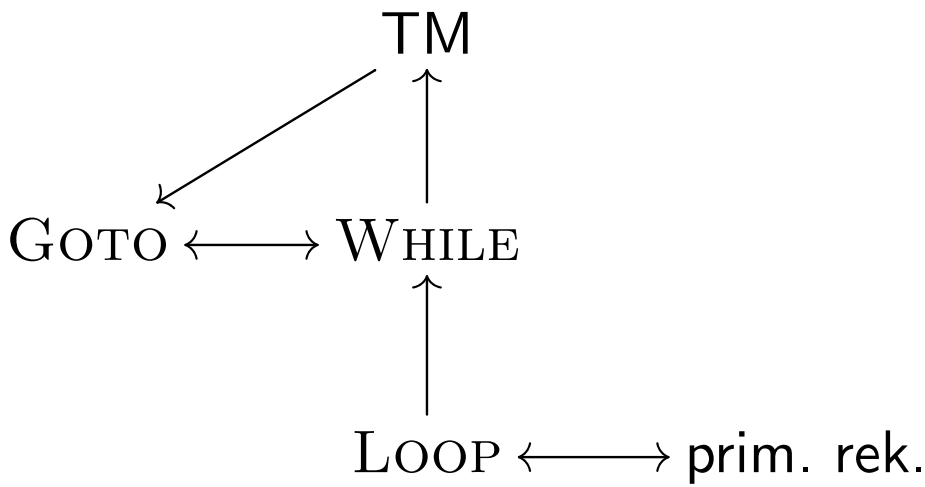
$$\begin{aligned} f(0, n_1, \ldots, n_{r-1}) &= g(n_1, \ldots, n_{r-1}) \\ f(n+1, n_1, \ldots, n_{r-1}) &= h(f(n, n_1, \ldots, n_{r-1}), n, n_1, \ldots, n_{r-1}) \end{aligned}$$

Die Funktion f lässt sich dann durch das folgende (Pseudocode-) LOOP-Programm berechnen:

$y := g(x_1, \ldots, x_{r-1})$; $k := 0$;
LOOP x_0 DO
    $y := h(y, k, x_1, \ldots, x_{r-1})$; $k := k + 1$
END

Unter Verwendung von LOOP-Programmen für g und h sowie Zwischenspeicherung ähnlich zu Fall 2 kann dieser Pseudocode in ein LOOP-Programm für f umgesetzt werden. □

Zwischenbilanz



μ-rekursive Funktionen

Wir werden eine weitere Klasse von Funktionen definieren, die gleichmächtig zu WHILE-Programmen, GOTO-Programmen und Turingmaschinen ist.

Definition μ-Operator

Sei $f : \mathbb{N}^{k+1} \dashrightarrow \mathbb{N}$ eine partielle Funktion. Dann ist $\mu f : \mathbb{N}^k \dashrightarrow \mathbb{N}$ definiert durch

$$\begin{aligned} \mu f(x_1, \ldots, x_k) \quad = \quad & \min\{n \mid f(n, x_1, \ldots, x_k) = 0 \text{ und} \\ & \forall m < n : f(m, x_1, \ldots, x_k) \text{ definiert}\} \end{aligned}$$

Dabei gilt $\min \emptyset =$ *undefiniert*.

μ-rekursive Funktionen

Intuitive Berechnung von $\mu f(\bar{n})$ (falls f berechenbar ist):
Berechne $f(0, \bar{n})$, $f(1, \bar{n})$, ... bis ein n gefunden wird mit $f(n, \bar{n}) = 0$. In diesem Augenblick gib n als Funktionswert zurück.
Dieser Algorithmus terminiert nicht, falls

- $f(m, \bar{n})$ undefiniert ist (ohne, dass vorher einmal der Funktionswert gleich 0 war) oder
- der Funktionswert 0 nie erreicht wird.

In diesem Fall ist $\mu f(\bar{n})$ undefiniert.

Analogie zu WHILE-Programmen: es ist nicht klar, ob die Abbruchbedingung jemals erfüllt wird.

μ-rekursive Funktionen

Definition μ-rekursive Funktionen

- Alle konstanten Funktionen $k_m : \mathbb{N} \to \mathbb{N} : n \mapsto m$, alle Projektionen $\pi_i^k : \mathbb{N}^k \to \mathbb{N} : (n_1, \ldots, n_k) \mapsto n_i$ und die Nachfolgerfunktion $s : \mathbb{N} \to \mathbb{N} : n \mapsto n + 1$ sind μ-rekursiv.
- Sind $g : \mathbb{N}^k \dashrightarrow \mathbb{N}$ und $f_1, \ldots, f_k : \mathbb{N}^r \dashrightarrow \mathbb{N}$ μ-rekursiv, so auch $f : \mathbb{N}^r \dashrightarrow \mathbb{N}$ mit $f(\bar{n}) = g(f_1(\bar{n}), \ldots, f_k(\bar{n}))$
  (wobei $f(\bar{n})$ genau dann definiert ist, wenn $f_i(\bar{n})$ für alle i definiert ist und wenn g auf diesen Werten definiert ist).
- Jede partielle Funktion f, die durch primitive Rekursion aus μ-rekursiven Funktionen entsteht, ist μ-rekursiv.
- Ist f μ-rekursiv, so auch μf.

μ-rekursive Funktionen

Durch den μ-Operator können nun auch echt partielle Funktionen erzeugt werden.

Überall undefinierte Funktion

Die partielle Funktion $\Omega\colon \mathbb{N} \dashrightarrow \mathbb{N}$ mit $\Omega(n) = \textit{undefiniert}$ für alle $n \in \mathbb{N}$ ist μ-rekursiv.

Es gilt $f(m,n) := k_1(\pi_1^2(m,n)) = 1$ für alle $m, n \in \mathbb{N}$ und f ist μ-rekursiv.

Dann gilt $\Omega = \mu f$ und Ω ist μ-rekursiv. □

μ-rekursive Funktionen

Weiteres Beispiel:

Wurzelfunktion

Die Funktion $sqrt : \mathbb{N} \to \mathbb{N}$ mit $sqrt(n) = \lceil \sqrt{n} \rceil$ ist μ-rekursiv.

(Dabei rundet $\lceil \ldots \rceil$ eine reelle Zahl auf die nächstgrößere (oder gleiche) ganze Zahl auf.)

Sei $f(m, n) = n - m \cdot m$. (beachte: die Multiplikationsfunktion und Subtraktionsfunktion sind primitiv rekursiv und damit μ-rekursiv).

Dann gilt $sqrt = \mu f$.

Diese Funktion ist jedoch auch primitiv rekursiv. Intuition: bei Berechnung von $sqrt(n)$ sind immer höchstens n Iterationen notwendig.)

μ-rekursive Funktionen

Lemma 22 (WHILE-Programm $\rightarrow$ μ-rekursiv)

Jede WHILE-berechenbare partielle Funktion ist μ-rekursiv.

Beweis:

Es genügt, den Beweis von Lemma 20 um die WHILE-Schleife zu erweitern.

Sei also $P = (\text{WHILE } x_i \neq 0 \text{ DO } Q \text{ End})$ ein WHILE-Programm, in dem die Variablen x_j für $j \geq k$ nicht vorkommen.

Wir müssen zeigen, dass
$g_P : \mathbb{N} \dashrightarrow \mathbb{N} : \langle n_0, n_1, \ldots, n_{k-1} \rangle \mapsto \langle [P]_k(n_0, \ldots, n_{k-1}) \rangle$ μ-rekursiv ist (vgl. Folie 116).

Nach Induktion ist dies für g_Q bereits der Fall.

μ-rekursive Funktionen

Sei also $P = (\textsc{While}\ x_i \neq 0\ \textsc{Do}\ Q\ \textsc{End})$ ein $\textsc{While}$-Programm, in dem die Variablen x_j für $j \geq k$ nicht vorkommen.

Wir müssen zeigen, dass
$g_P : \mathbb{N} \dashrightarrow \mathbb{N} : \langle n_0, n_1, \ldots, n_{k-1} \rangle \mapsto \langle [P]_k(n_0, \ldots, n_{k-1}) \rangle$ μ-rekursiv ist (vgl. Folie 116).

Nach Induktion ist dies für g_Q bereits der Fall.

Die Funktion $h : \mathbb{N}^2 \dashrightarrow \mathbb{N} : (n, m) \mapsto g_Q^n(m)$ ist μ-rekursiv (vgl. Beweis von Lemma 20 (Fall 3))

Dann ist auch $k : \mathbb{N}^2 \dashrightarrow \mathbb{N} : (n, m) \mapsto d_i(h(n, m))$ μ-rekursiv

und es gilt $g_P(x) = h((\mu k)(x), x)$,d.h. auch g_P ist μ-rekursiv. □

μ-rekursive Funktionen

Satz 23 (WHILE-Programm $\leftrightarrow$ μ-rekursiv)

Eine partielle Funktion $f : \mathbb{N}^r \dashrightarrow \mathbb{N}$ ist genau dann WHILE-berechenbar, wenn sie μ-rekursiv ist.

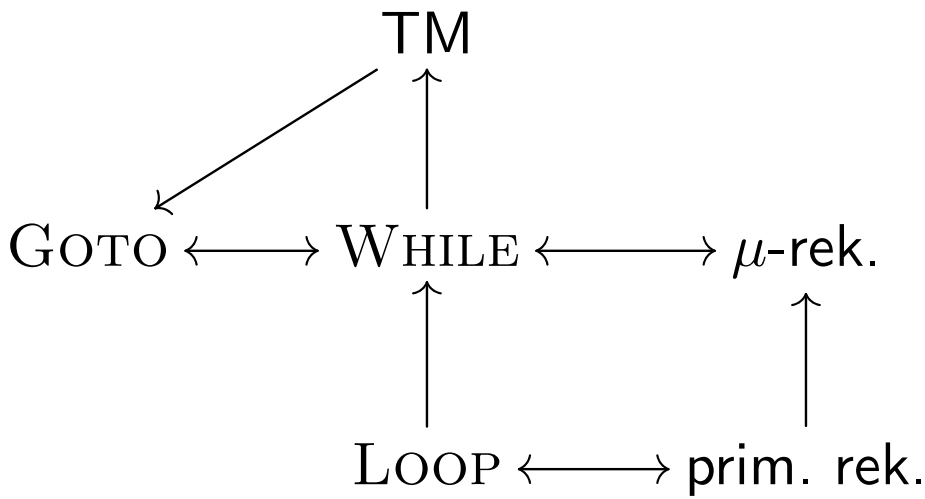
Wegen Lemma 22 reicht es, die Implikation „$\Leftarrow$“ zu beweisen. Hierzu müssen wir den Beweis von Satz 21 um den μ-Operator erweitern.
Sei also $f = \mu g : \mathbb{N}^r \to \mathbb{N}$ für eine μ-rekursive Funktion $g : \mathbb{N}^{r+1} \to \mathbb{N}$.

Dann kann f durch das folgende (Pseudocode-) WHILE-Programm berechnet werden:

$x_0 := 0;\ y := g(0, x_1, \ldots, x_r);$
WHILE $y \neq 0$ DO
    $x_0 := x_0 + 1;\ y := g(x_0, x_1, \ldots, x_r);$
END

Unter Verwendung eines WHILE-Programms für g kann dieser Pseudocode in ein WHILE-Programm für f umgesetzt werden.

Bilanz



Insbesondere sind also 4 Berechnungsbegriffe äquivalent, was die Churchsche These stützt:

Churchsche These

Die im intuitiven Sinne berechenbaren Funktionen sind genau die durch die formale Definition der Turingmaschinen berechenbaren.

Wir wissen auch, daß es WHILE-berechenbare Funktionen gibt, die nicht LOOP-berechenbar sind (denn diese sind total).

Totale WHILE-berechenbare Funktionen

Frage: Ist jede totale WHILE-berechenbare Funktion auch LOOP-berechenbar?

Totale, nicht LOOP-berechenbare Funktionen

Es gibt totale (Turing-)berechenbare Funktionen, die nicht LOOP-berechenbar sind.

Wir zeigen die Existenz solcher Funktionen durch einen Diagonalisierungsbeweis.

Totale WHILE-berechenbare Funktionen

Diagonalisierungsbeweis:

1. Schritt: Wir sehen LOOP-Programme als Wörter über einem endlichen Alphabet Σ (bestehend aus den Zeichen $:=$, $+$, $-$, LOOP, x_0, x_1, ...) an. Damit kann man sie längen-lexikographisch anordnen und erhält eine Aufzählung aller LOOP-Programme.

2. Schritt: Die Funktion p, die einer natürlichen Zahl n das n-te LOOP-Programm in dieser Aufzählung zuordnet, ist berechenbar. Beispielsweise kann man nacheinander alle Wörter aus Σ^* längen-lexikographisch geordnet aufzählen, überprüfen, ob sie gültige LOOP-Programme sind und das n-te gültige Programm ausgeben.

Totale WHILE-berechenbare Funktionen

3. Schritt: Wir definieren eine Funktion $g : \mathbb{N} \to \mathbb{N}$, die bei Eingabe von n folgende Berechnungsschritte durchführt

- Das LOOP-Programm $P = p(n)$ wird mit n als Eingabe, d.h. n in der Variablen x_0, alle anderen Variablen mit 0 belegt, gestartet.
- Der bei Terminierung in der Variablen x_0 befindliche Wert wird um eins inkrementiert und ist dann der Funktionswert $g(n)$.Das heißt $g(n) = \pi_0([P]_k(n, 0, \ldots, 0)) + 1$ (wobei die Variablen x_i für $i \geq k$ in P nicht vorkommen).

Diese Funktion ist total, da jedes LOOP-Programm terminiert.
Sie ist auch Turing-berechenbar(zumindest nach der Churchschen These, man kann es aber auch formal beweisen) und daher WHILE-berechenbar.

Totale WHILE-berechenbare Funktionen

4. Schritt: Wir zeigen nun, dass g nicht LOOP-berechenbar sein kann. Angenommen, es gibt ein LOOP-Programm P, das g berechnet. Sei i der Index von P in der Aufzählung, d.h. $P = p(i)$. Es gilt aufgrund der Wahl von P und der Definition von g:

$$\pi_0([P]_k(i, 0, \ldots, 0)) = g(i) = \pi_0([P]_k(i, 0, \ldots, 0)) + 1$$

Das ist jedoch ein Widerspruch! □

Totale WHILE-berechenbare Funktionen

Die Ackermann-Funktion ist das klassische Beispiel einer totalen WHILE-, aber nicht LOOP-berechenbaren Funktion

Ackermannfunktion $a\colon \mathbb{N}^2 \to \mathbb{N}$ (Ackermann 1928)

$$
\begin{aligned}
a(0,y) &= y+1 \\
a(x,0) &= a(x-1,1), \text{ falls } x > 0 \\
a(x,y) &= a(x-1,a(x,y-1)), \text{ falls } x,y > 0
\end{aligned}
$$

Totale WHILE-berechenbare Funktionen

Wertetabelle der Ackermannfunktion für kleine Werte:

$y =$	0	1	2	3	4	...	$a(x, y)$
$x = 0$	1	2	3	4	5	...	$y + 1$
$x = 1$	2	3	4	5	6	...	$y + 2$
$x = 2$	3	5	7	9	11	...	$2y + 3$
$x = 3$	5	13	29	61	125	...	$2^{y+3} - 3$
$x = 4$	13	65533	$> 10^{19727}$				$\underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{y+3 \text{ Zweier}} - 3$
...							

Lemma 24

Die Ackermannfunktion ist total und WHILE-berechenbar.

Beweis siehe z.B. [Schöning].

Totale WHILE-berechenbare Funktionen

Sei P ein LOOP Programm, in dem keine Variable x_i mit $i \geq k$ vorkommt.

Für ein Tupel $(n_1, \ldots, n_k) \in \mathbb{N}^k$ schreiben wir im folgenden

$$\sum(n_1, \ldots, n_k) = n_1 + \cdots + n_k.$$

Dann definieren wir

$$f_P(n) = \max\{\sum[P]_k(n_1, \ldots, n_k) \mid n_1, \ldots, n_k \in \mathbb{N}, \sum(n_1, \ldots, n_k) \leq n\}.$$

Lemma 25

Für jedes LOOP Programm P gibt es eine Zahl ℓ, so dass für alle $n \in \mathbb{N}$ gilt: $f_P(n) < a(\ell, n)$.

Beweis siehe z.B. [Schöning].

Totale WHILE-berechenbare Funktionen

Satz 26 (Ackermann-Funktion)

Die Ackermann-Funktion ist total, WHILE*-berechenbar, aber nicht* LOOP*-berechenbar.*

Angenommen die Ackermannfunktion a wäre LOOP-berechenbar.
Dann ist auch die Funktion $g : \mathbb{N} \to \mathbb{N}$ mit $g(n) = a(n, n)$ LOOP-berechenbar.
Sei P ein LOOP-Programm mit

$$\forall n \in \mathbb{N} : g(n) = \pi_0([P]_k(n, 0, \ldots, 0)).$$

Nach Lemma 25 existiert eine Konstante ℓ mit

$$\forall n \in \mathbb{N} : f_P(n) < a(\ell, n).$$

Für $n = \ell$ folgt dann

$$g(\ell) = \pi_0([P]_k(\ell, 0, \ldots, 0)) \leq f_P(\ell) < a(\ell, \ell) = g(\ell).$$

Dies ist ein Widerspruch. □