LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Formale Sprachen und Komplexität
Theoretische Informatik für Medieninformatiker
Sommersemester 2023

Primitiv und μ-rekursive Funktionen

Prof. Dr. Jasmin Blanchette

Lehr- und Forschungseinheit für
Theoretische Informatik

Stand: 27. Juni 2023

Folien ursprünglich von PD Dr. David Sabel

Rekursive Funktionen

Weitere Formalismen zur Definition der Berechenbarkeit

- Primitiv rekursive Funktionen und
- μ-rekursive Funktionen

Wir werden schließlich sehen:

- Primitiv rekursive Funktionen entsprechen genau den LOOP-berechenbaren Funktionen
- μ-rekursive Funktionen entsprechen genau den Turingberechenbaren (WHILE-, GOTO-berechenbaren) Funktionen

Primitiv rekursive Funktionen

Definition (Primitiv rekursive Funktionen)

Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ ist **primitiv rekursiv**, wenn sie der folgenden induktiven Definition genügt:

- Jede konstante Funktion $f(x_1, \ldots, x_k) = c \in \mathbb{N}$ ist primitiv rekursiv.
- Die Projektionsfunktionen $\pi_i^k(x_1, \ldots, x_k) = x_i$ sind primitiv rekursiv.
- Die Nachfolgerfunktion $succ(x) = x + 1$ ist primitiv rekursiv.
- Komposition/Einsetzung: Wenn $g : \mathbb{N}^m \to \mathbb{N}$ und für $i = 1, \ldots, m$: $h_i : \mathbb{N}^k \to \mathbb{N}$ primitiv rekursiv sind, dann ist auch f mit $f(x_1, \ldots, x_k) = g(h_1(x_1, \ldots, x_k), \ldots, h_m(x_1, \ldots, x_k))$ primitiv rekursiv.
- Rekursion: Wenn $g : \mathbb{N}^{k-1} \to \mathbb{N}$ und $h : \mathbb{N}^{k+1} \to \mathbb{N}$ primitiv rekursiv sind, dann ist auch f mit

  $$f(x_1, \ldots, x_k) = \begin{cases} g(x_2, \ldots, x_k), & \text{wenn } x_1 = 0 \\ h(f(x_1 - 1, x_2, \ldots, x_k), x_1 - 1, x_2, \ldots, x_k), & \text{sonst} \end{cases}$$

  primitiv rekursiv.

Konstruktionen (1)

Komponenten eines Tupels entfernen/vertauschen/vervielfachen

Wenn $g : \mathbb{N}^4 \to \mathbb{N}$ primitiv rekursiv ist, dann ist auch $f : \mathbb{N}^3 \to \mathbb{N}$ mit

$$f(n_1, n_2, n_3) = g(n_2, n_3, n_3, n_2)$$

denn

$$f(n_1, n_2, n_3) = g(\pi_2^3(n_1, n_2, n_3), \pi_3^3(n_1, n_2, n_3), \pi_3^3(n_1, n_2, n_3), \pi_2^3(n_1, n_2, n_3))$$

Konstruktionen (2)

Rekursion durch das i-te Argument

Für $1 \leq i \leq k$ kann man

$$f(x_1,\ldots,x_k) = \begin{cases} g(x_1,\ldots,\boldsymbol{x_{i-1}},\boldsymbol{x_{i+1}},x_k), & \text{falls } x_i = 0 \\ h(f(x_1,\ldots,\boldsymbol{x_{i-1}},\boldsymbol{x_i}-1,\boldsymbol{x_{i+1}},\ldots,x_k),x_1,\ldots,\boldsymbol{x_{i-1}},\boldsymbol{x_i}-1,\boldsymbol{x_{i+1}},\ldots,x_k), & \text{sonst} \end{cases}$$

durch $f(x_1,\ldots,x_k) = f'(x_i,x_1\ldots,x_{i-1},x_{i+1},\ldots,x_k)$ darstellen, wobei

$$\begin{aligned} f'(y_1,\ldots,y_k) &= \begin{cases} g(y_2,\ldots,y_k), & \text{falls } y_1 = 0 \\ h'(f'(y_1-1,y_2\ldots,y_k),y_1-1,y_2,\ldots,y_k), & \text{sonst} \end{cases} \\ h'(y_0,\ldots,y_k) &= h(y_0,y_2,y_3,\ldots,y_i,y_1,y_{i+1},y_{i+2},\ldots,y_k) \end{aligned}$$

Beispiele (1)

Additionsfunktion

$add(x_1, x_2) = x_1 + x_2$ ist primitiv rekursiv:

$$add(x_1, x_2) = \begin{cases} x_2, & \text{falls } x_1 = 0 \\ succ(add(x_1 - 1, x_2)), & \text{sonst} \end{cases}$$

Idee: x_1-mal 1 zu x_2 addieren.

Die verwendeten Funktionen g und h aus der Definition der primitiv rekursiven Funktionen sind hier

- $g = \pi_1^1$
- $h(x_1, x_2, x_3) = succ(\pi_1^3(x_1, x_2, x_3))$

Beispiele (2)

Multiplikationsfunktion

$$mult(x_1, x_2) = \begin{cases} 0, & \text{falls } x_1 = 0 \\ add(mult(x_1 - 1, x_2), x_2), & \text{sonst} \end{cases}$$

Idee: x_1-mal x_2 zu 0 addieren.

Beispiele (3)

Differenz

Allgemein $x_1 - x_2$ nicht primitiv rekursiv, undefinierter Fall $x_1 < x_2$ nicht darstellbar.

Angepasste Differenz

Liefert 0 falls $x_1 < x_2$, ist primitiv rekursiv:

$$sub(x_1, x_2) = \begin{cases} x_1, & \text{falls } x_2 = 0 \\ pred(sub(x_1, x_2 - 1)), & \text{sonst} \end{cases}$$

wobei

$$pred(x_1) = \begin{cases} 0, & \text{falls } x_1 = 0 \\ x_1 - 1, & \text{sonst} \end{cases}$$

Nächstes Ziel

Wir wollen zeigen:

Primitiv rekursive Funktionen sind genau die LOOP-berechenbaren Funktionen.

Benötigt:

- Darstellung einer Variablenbelegung ρ als **eine einzige Zahl**, um sie der primitiv rekursiven Funktion als Argument zu übergeben.
- D.h. eindeutige Darstellung eines Tupels natürlicher Zahlen als eine einzige Zahl.
- Operationen zum Konvertieren in beide Richtungen.

Eine solches Verfahren nennt man auch „Gödelisierung“ (nach Kurt Gödel).

Gödelisierung (1)

- Tupel von natürlichen Zahlen $(x_0, \ldots, x_k)$ bijektiv in die natürlichen Zahlen abbilden
- mit primitiv rekursiven Funktionen

$$c(x, y) = \binom{x + y + 1}{2} + x$$

die mithilfe des Binomialkoeffizient definiert ist.

Werte von $c(x, y)$ für $x, y \in \{0, \ldots, 5\}$:

$x \backslash y$	0	1	2	3	4	5
0	0	1	3	6	10	15
1	2	4	7	11	16	22
2	5	8	12	17	23	30
3	9	13	18	24	31	39
4	14	19	25	32	40	49
5	20	26	33	41	50	60

Gödelisierung (2)

Funktion c ist primitiv rekursiv, da

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

Für $(k+1)$-Tupel definieren wir

$$\langle x_0, \ldots, x_k \rangle = c(x_0, c(x_1, \ldots, c(x_k, 0) \ldots))$$

Beachte: $\langle \cdot \rangle$ ist primitiv rekursiv
(da c primitiv rekursiv und Komposition primitiv rekursiv ist).

Gödelisierung (3)

Rückgewinnung der Komponenten:

Seien *left* und *right* Funktionen mit

- $\mathit{left}(c(x, y)) = x$ und
- $\mathit{right}(c(x, y)) = y$.

Im Skript wird gezeigt:

- *left* und *right* existieren.
- *left* und *right* sind primitiv rekursiv.

Gödelisierung (4)

Zugriff auf beliebige Komponenten:

Programmiere $d_i(\langle x_0, \ldots, x_k \rangle) = x_i$ durch

$$\begin{array}{rcl} d_0(x) & = & \mathit{left}(x) \\ d_1(x) & = & \mathit{left}(\mathit{right}(x)) \\ d_i(x) & = & \mathit{left}(\underbrace{\mathit{right}(\mathit{right} \ldots \mathit{right}}_{i\text{-mal}}(x) \ldots)) \end{array}$$

Damit sind auch die d_i-Funktionen primitiv rekursiv.

Von LOOP-Programm berechnete Funktion

- Sei P ein LOOP-Programm.
- Seien $x_0, x_1, \dots, x_n$ alle vom Programm P verwendeten Variablen.
- Die von P berechnete Funktion:

$$g_P(\langle x_0, \dots, x_n \rangle) = \langle x_0', \dots, x_n' \rangle$$

LOOP-Programme berechnen primitiv rekursive Funktionen

Lemma

Für jedes LOOP-Programm P ist die zugehörige Funktion g_P primitiv rekursiv.

LOOP-Programme berechnen primitiv rekursive Funktionen

Lemma

Für jedes LOOP-Programm P ist die zugehörige Funktion g_P primitiv rekursiv.

Beweis: Strukturelle Induktion über jedes Teilprogramm Q und die zugehörige Funktion g_Q.

- Basis: Q ist Zuweisung $x_i := x_j \pm c$.
  Für g_Q muss gelten:

$$g_Q(\langle m_0, \ldots, m_n \rangle) = \langle m_0, \ldots, m_{i-1}, m_j \pm c, m_i, \ldots, m_n \rangle$$

  Primitiv rekursive Implementierung:

$$g_Q(x) = \langle d_0(x), \ldots, d_{i-1}(x), d_j(x) + c, d_{i+1}(x), \ldots d_n(x) \rangle$$

LOOP-Programme berechnen primitiv rekursive Funktionen (2)

Induktionsschritt:

- Q ist eine Sequenz $Q_1; Q_2$.

  Induktionshypothese: primitiv rekursive Funktionen g_{Q_1}, g_{Q_2}.

  Funktion $g_Q(x) = g_{Q_2}(g_{Q_1}(x))$ ist primitiv rekursiv.

- Q ist **LOOP** x_i **DO** P **END**.

  Induktionshypothese liefert primitiv rekursive Funktion g_P.

  Konstruktion von g_Q: x_i-mal wird g_P angewendet.

$$
\begin{aligned}
g_Q(x) &= run(d_i(x), x) \\
run(n, x) &= \begin{cases} x, & \text{falls } n = 0 \\ g_P(run(n-1, x)), & \text{sonst} \end{cases}
\end{aligned}
$$

run ist primitiv rekursiv. Damit ist auch g_Q primitiv rekursiv. □

LOOP-berechenbare Funktionen sind primitiv rekursiv

Satz 11.1.6

Jede LOOP-berechenbare Funktion ist primitiv rekursiv.

LOOP-berechenbare Funktionen sind primitiv rekursiv

Satz 11.1.6

Jede LOOP-berechenbare Funktion ist primitiv rekursiv.

Beweis:

- Sei $f : \mathbb{N}^k \to \mathbb{N}$ LOOP-berechenbar.
- Sei P LOOP-Programm mit $\rho = \{x_1 \mapsto n_1, \ldots, x_k \mapsto n_k\}$, $(\rho, P) \xrightarrow[\text{LOOP}]{}^{*} (\rho', \varepsilon)$ und $\rho'(x_0) = f(n_1, \ldots, n_k)$.
- Es gilt $f(n_1, \ldots, n_k) = d_0(g_P(\langle 0, n_1, \ldots, n_k, 0, \ldots, 0\rangle))$.
- Da $\langle\cdot\rangle$, d_0 und g_P primitiv rekursiv sind, ist auch f primitiv rekursiv. $\square$

Primitiv rekursive Funktionen sind LOOP-berechenbar

Satz 11.1.7

Jede primitiv rekursive Funktion ist LOOP-berechenbar.

Primitiv rekursive Funktionen sind LOOP-berechenbar

Satz 11.1.7

Jede primitiv rekursive Funktion ist LOOP-berechenbar.

Beweis: Induktion über Struktur der primitiv rekursiven Funktion:

- Wenn $f(x) = c$, $f = succ$, oder $f = \pi_n^k$, dann gibt es auch LOOP-Programm dazu.
- Wenn $f(x_1, \ldots, x_k) = h(g_1(x_1, \ldots, x_k), \ldots, g_n(x_1, \ldots, x_k))$: Induktionshypothese liefert LOOP-Programme P_h und $P_{g_1}, \ldots, P_{g_n}$, die h, $g_1, \ldots, g_n$ berechnen. Konstruiere Programm für f nach dem Schema

$$
\begin{aligned}
y_1 &:= g_1(x_1, \ldots, x_k); \\
&\vdots \\
y_n &:= g_n(x_1, \ldots, x_k); \\
x_0 &:= h(y_1, \ldots, y_n)
\end{aligned}
$$

- ...

Primitiv rekursive Funktionen sind LOOP-berechenbar (2)

- ...
- Wenn $f(x_1, \dots, x_k) = \begin{cases} g(x_2, \dots, x_k), & \text{wenn } x_1 = 0 \\ h(f(x_1 - 1, x_2, \dots, x_k), x_1 - 1, x_2, \dots, x_k), & \text{sonst} \end{cases}$

  wobei $g : \mathbb{N}^{k-1} \to \mathbb{N}$ und $h : \mathbb{N}^{k+1} \to \mathbb{N}$ primitiv rekursiv sind. Induktionshypothese liefert LOOP-Programme, die g, h berechnen. Konstruiere LOOP-Programm für f nach dem Schema

$$
\begin{array}{l}
y := 0; \\
x_0 := g(x_2, \dots, x_k); \\
\textbf{LOOP } x_1 \textbf{ DO} \\
\quad x_0 := h(x_0, y, x_2, \dots, x_k); \\
\quad y := y + 1 \\
\textbf{END}
\end{array}
$$

□

LOOP-berechenbar = primitiv rekursiv

Theorem 11.1.8

Die primitiv rekursiven Funktionen sind genau die LOOP-berechenbaren Funktionen.

μ-Operator

Definition μ-Operator

Sei $h : \mathbb{N}^{k+1} \to \mathbb{N}$ eine (partielle oder totale) Funktion.
Dann ist $(\mu h) : \mathbb{N}^k \to \mathbb{N}$ definiert als

$$(\mu h)(x_1, \ldots, x_k) = \begin{cases} n, & \text{falls } h(n, x_1, \ldots, x_k) = 0 \text{ und für} \\ & \text{alle } m < n\text{: } h(m, x_1, \ldots, x_k) \text{ ist} \\ & \text{definiert und } h(m, x_1, \ldots, x_k) > 0 \\ \text{undefiniert}, & \text{sonst} \end{cases}$$

- μ-Operator „sucht“ nach einer kleinsten Nullstelle von h.
- Wenn diese nicht existiert (entweder da h keine Nullstelle hat, oder da h undefiniert ist für Werte, die kleiner als die Nullstelle sind), dann ist auch der μ-Operator angewendet auf h undefiniert.

μ-rekursive Funktionen

Definition (μ-rekursiven Funktionen)

Die Menge aller μ-rekursiven Funktionen sei die kleinste Menge, sodass gilt:

- Jede konstante Funktion $f(x_1, \dots, x_k) = c \in \mathbb{N}$ ist μ-rekursiv.
- Die Projektionsfunktionen $\pi_i^k(x_1, \dots, x_k) = x_i$ sind μ-rekursiv.
- Die Nachfolgerfunktion $succ(x) = x + 1$ ist μ-rekursiv.
- Komposition/Einsetzung: Wenn $g : \mathbb{N}^m \to \mathbb{N}$ und für $i = 1, \dots, m$: $h_i : \mathbb{N}^k \to \mathbb{N}$ μ-rekursiv sind, dann ist auch f mit $f(x_1, \dots, x_k) = g(h_1(x_1, \dots, x_k), \dots, h_m(x_1, \dots, x_k))$ μ-rekursiv.
- Rekursion: Wenn $g : \mathbb{N}^{k-1} \to \mathbb{N}$ und $h : \mathbb{N}^{k+1} \to \mathbb{N}$ μ-rekursiv sind, dann ist

  $$f(x_1, \dots, x_k) = \begin{cases} g(x_2, \dots, x_k), & \text{wenn } x_1 = 0 \\ h(f(x_1 - 1, x_2, \dots, x_k), x_1 - 1, x_2, \dots, x_k), & \text{sonst} \end{cases}$$

  auch μ-rekursiv.
- μ-Operator: Wenn $h : \mathbb{N}^{k+1} \to \mathbb{N}$ μ-rekursiv ist, dann ist auch $f = \mu h$ μ-rekursiv.

WHILE-berechenbare Funktionen sind μ-rekursiv

Satz 11.2.3

Jede WHILE-berechenbare Funktion ist μ-rekursiv.

WHILE-berechenbare Funktionen sind μ-rekursiv

Satz 11.2.3

Jede WHILE-berechenbare Funktion ist μ-rekursiv.

Beweis: Analog zu Satz 11.1.6 über Struktur des WHILE-Programms P
Neuer Fall: P ist **WHILE** $x_i \neq 0$ **DO** Q **END**
Induktionshypothese liefert μ-rekursive Funktion g_Q für Q. Konstruiere:

$$\begin{aligned}
g_P(x) &= run(\mu(run_i)(x), x) \\
run_i(n, x) &= d_i(run(n, x)) \\
run(n, x) &= \begin{cases} x, & \text{falls } n = 0 \\ g_Q(run(n-1, x)), & \text{sonst} \end{cases}
\end{aligned}$$

- $run(n, x)$ führt n-mal g_Q aus.
- $\mu(run_i)(x)$ berechnet, wie oft die Schleife minimal durchlaufen werden muss, bis x_i den Wert 0 hat.
- Divergiert die Schleife, so ist $\mu(run_i)$ undefiniert und g_P undefiniert. $\square$

μ-rekursive Funktionen sind WHILE-berechenbar

Satz 11.2.4

Jede μ-rekursive Funktion ist WHILE-berechenbar.

μ-rekursive Funktionen sind WHILE-berechenbar

Satz 11.2.4

Jede μ-rekursive Funktion ist WHILE-berechenbar.

Beweis: Analog zu Satz 11.1.7 über die Struktur der Funktion.
Neuer Fall: $f = \mu h$ für eine μ-rekursive Funktion h: Die Induktionshypothese liefert WHILE-Programm P_h, das h berechnet. Konstruiere P_f nach dem Schema

$x_0 := 0;$
$y := h(0, x_1, \ldots, x_n);$
WHILE $y \neq 0$ **DO**
  $x_0 := x_0 + 1; y := h(x_0, \ldots, x_n)$
END

- **WHILE**-Schleife berechnet minimalen Wert für $h(x_0, \ldots, x_n) = 0$
- Wenn dieser nicht existiert, terminiert die Schleife nicht.
- Entspricht der Berechnung von μh. □

WHILE-berechenbar = μ-rekursiv

Theorem 11.2.5

Die μ-rekursiven Funktionen entsprechen genau den WHILE-berechenbaren (und damit auch den GOTO- und Turingberechenbaren) Funktionen.

Überblick: Berechenbarkeitsformalismen