Theoretische Informatik II
Ausarbeitung der Vorlesung vom 15.12.2000 bzw. 22.12.2000

Lutz Dittrich[1] Sven Friedrich[2]

23. Januar 2001

[1]lditt@rz.uni-potsdam.de
[2]sfried@rz.uni-potsdam.de

6 Aufzählbare Sprachen Ch-0

6.1 Turing-Maschinen

6.2 Registermaschinen

6.3 μ-rekursive Funktionen

Folgend soll ein Modell vorgestellt werden, das sich der Frage der Berechenbarkeit von der Algorithmenseite her nähert. Die rekursiven Funktionen beschreiben Lösungen von Problemen und nicht die einzelnen Schritte des Rechenweges wie bei Turing-Maschinen. Daher hat das Modell der rekursiven Funktionen Ähnlichkeit mit funktionalen Programmiersprachen.

Die einfachste Klasse von rekursiven Funktionen sind die primitiv rekursiven Funktionen. Sie umfassen drei Arten von Funktionen und zwei Methoden, sie zu komplexeren Funktionen zusammenzusetzen. Die drei Funktionstypen sind die Nachfolgerfunktion[1], die Projektion[2], die das i-te von n Argumenten auswählt und eine Konstante.

- Nachfolgerfunktion $N : \mathbb{N}_0 \to \mathbb{N}_0$ mit $N(x) = x + 1$
- Projektion $\Pi_i^n : \mathbb{N}_0^n \to \mathbb{N}_0$ mit $\Pi_i^n(x_1, \ldots, x_n) = x_i$
- Konstante $C_i^n : \mathbb{N}_0^n \to \mathbb{N}_0$ mit $C_i^n(x_1, \ldots, x_x) = i$

Für die Bildung der konstanten Funktion ist es ausreichend, die Konstante C_0^n zu definieren, da alle weiteren Konstanten C_i^n aus C_0^n und der Nachfolgerfunktion gebildet werden können:

$$\begin{aligned} C_1^n(x_1, \ldots, x_n) &= N(C_0^n(x_1, \ldots, x_n)) \\ C_2^n(x_1, \ldots, x_n) &= N(N(C_0^n(x_1, \ldots, x_n))) \end{aligned}$$

Um diese Funktionstypen kombinieren zu können, steht zum einem das "simultane Einsetzen"[3] und zum anderen die primitive Rekursion zur Verfügung.

Definition K:
Seien $g : \mathbb{N}_0^m \to \mathbb{N}_0$, $h_i : \mathbb{N}_0^k \to \mathbb{N}_0$, $(i = 1, \ldots, m)$ *Funktionen.*
Die Funktionen $f := g(h_1, \ldots, h_m)$ *mit* $f : \mathbb{N}_0^k \to \mathbb{N}_0$ *und*
$f(x_1, \ldots, x_k) = g(h_1(x_1, \ldots, x_k), \ldots, h_m(x_1, \ldots, x_k))$ *geht aus*
$g, h_1, \ldots, h_m$ *durch simultane Einsetzung hervor.*

[1] auch Successor-Funktion
[2] auch Auswahl
[3] auch Komposition

Definition L:

1. $f : \mathbb{N}_0^k \to \mathbb{N}_0$ *geht aus* $g : \mathbb{N}_0^{k-1} \to \mathbb{N}_0$ *und* $h : \mathbb{N}_0^{k+1} \to \mathbb{N}_0$ *durch primitive Rekursion hervor, wenn* $f(x_1, \ldots, x_{k-1}, 0) = g(x_1, \ldots, x_{k-1}), \forall x_1, \ldots, x_{k-1} \in \mathbb{N}_0$ *und* $f(x_1, \ldots, x_{k-1}, y+1) = h(x_1, \ldots, x_{k-1}, y, f(x_1, \ldots, x_{k-1}, y)), \forall x_1, \ldots, x_{k-1}, y \in \mathbb{N}_0$

2. f *entsteht aus* $g : \mathbb{N}_0^{k+1} \to \mathbb{N}_0$ *durch Anwendung des* μ*-Operators, wenn* $f(x_1, \ldots, x_k) = min\{y \mid g(x_1, \ldots, x_k, y) = 0$ *und* $g(x_1, \ldots, x_k, j)$ *ist def. für alle* $j < y\}$ *falls das Minimum definiert ist, ansonsten ist* f *undefiniert. Schreibweise:* $f = \mu g$

Der in der Definition L angesprochene μ-Operator ist das "funktionale Äquivalent" zu einer while-Schleife. Er ermittelt den kleinsten Wert, für den eine bestimmte Bedingung erfüllt ist. D.h., analog zu einer while-Schleife, die solange durchlaufen wird, bis ein bestimmtes Register den Wert 0 hat.

Definition M:

1. $f : \mathbb{N}_0^k \to \mathbb{N}_0$ *heißt primitiv-rekursiv, wenn* f *Nachfolgerfunktion, Projektion, Konstante ist, oder wenn* f *durch simultane Einsetzung oder durch primitive Rekursion aus primitiv-rekursiven Funktionen hervorgeht. Die Klasse der primitv-rekursiven Funktionen bezeichnen wir mit* $\mathcal{F}_{prim}$.

2. f *heißt* μ*-rekursiv, wenn* f *primitiv-rekursiv ist oder durch Anwenden des* μ*-Operators aus einer* μ*-rekursiven Funktion entsteht. Die Klasse der* μ*-rekursiven Funktionen bezeichenen wir mit* $\mathcal{F}_\mu$*, die Teilklasse der total* μ*-rekursiven Funktionen heißt* $\mathcal{R}_\mu$.

Bemerkung:
Hier offenbart sich eine fundamentale Idee der Informatik – das Baukastensystem. Ein Baukasten besteht bekanntlich aus Bausteinen, in vorliegendem Fall die Nachfolgerfunktion, die Projektion und die Konstante. Mit Hilfe dieser Bausteine lassen sich komplexere Funktion bilden, was unten dargestellt wird.

$B =$ (Elementares, Operationen)

Folgend sollen nun aus den obigen elementaren Bausteinen komplexere Funktionen konstruiert werden. Diese neu geschaffenen Funktionen werden sofort für die Bildung weiterer Funktionen genutzt. Alle nun folgenden Funktionen lassen sich wie folgt aus den Bausteinen und Operationen darstellen.

Grundlegende arithmetische Funktionen:

1. **Addition:** $x + y = (x + y - 1) + 1$ sei primitiv-rekursiv.

   - Die Projektion $g := \Pi_1^1, h := N \cdot \Pi_3^3$, also $f(x, y) = x + y$ wird definiert durch $f(x, 0) = \Pi_1^1(x)$. D.h., der Rekursionsanfang sei $x + 0 = x$.
   - $f(x, y + 1) = N(\Pi_3^3(x, y, f(x, y)))$ Die Funktion $f(x, y + 1)$ wird aus dem Nachfolger des Funktionswertes $f(x, y)$ gebildet.

2. **modifizierte Subtraktion** $\dot{-}1$ ist definiert durch:

$$(\dot{-}1)(n) := \begin{cases} 0, & \text{falls } n = 0 \\ n - 1, & \text{falls } n > 0. \end{cases}$$

$$\begin{aligned} (\dot{-}1)(0) &= g() \quad \text{mit } g() = 0 \\ (\dot{-}1)(m + 1) &= h(m, (\dot{-}1)(m)) \quad \text{mit } h(n_1, n_2) = \Pi_1^2(n_1, n_2) \end{aligned}$$

   Das heißt, $(\dot{-}1)(m + 1) = \Pi_1^2(m, (\dot{-}1)(m)) = m$.

3. **allgemeine modifizierte Subtraktion** $x \dot{-} y$ ist definiert durch:

$$n \dot{-} m := \begin{cases} 0, & \text{falls } m \geq n \\ n - m, & \text{sonst.} \end{cases}$$

$$\begin{aligned} x \dot{-} 0 &= n \\ x \dot{-} (y + 1) &= (x \dot{-} y) \dot{-} 1 \forall n_1, n_2 \in \mathbb{N}_0 \end{aligned}$$

   Nun wurde zweimal im Detail gezeigt, wie man die Definition der primitiven Rekursion anwendet. Im weiteren soll auf eine ebenso ausführliche Schreibweise zugunsten einer besser lesbaren Variante verzichtet werden.

4. **Multiplikation** $n * m$:
   Die Multiplikation kann mit man mit Hilfe der Addition rekursiv beschreiben:

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

5. **Potenz** n^m:
   Aufbauend auf der Multiplikation ergibt sich die m-te Potenz wie folgt:

$$\begin{aligned} (n^0) &= 1 \\ (n^{(m+1)}) &= (n^m) * n \end{aligned}$$

6. **Fakultät** $n!$
   Aufbauend auf der Multiplikation und der Nachfolgerfunktion ergibt sich die Fakultätsfunktion.

$$\begin{aligned}(0!) &= 1\\ (m+1)! &= (m!) * (m+1)\end{aligned}$$

7. **Fallunterscheidung**

$$f(x,y) = \begin{cases} f_1(x), & \text{falls } y = 0\\ f_2(x), & \text{falls } y \neq 0\end{cases}$$

   Die Fallunterscheidung läßt sich auch durch folgende programmiersprachenähnliche Notation verdeutlichen: $f(x,y) =$ **if** $(y=0)$ **then** $f_1(x)$ **else** $f_2(x)$

$$\begin{aligned}f(x,0) &= f_1(x)\\ f(x,y+1) &= f_2\left(\Pi_1^3\left(x,y,f(x,y)\right)\right)\end{aligned}$$

8. **Signum-Funktion:**

$$\text{sign}(x) = \begin{cases} 1, & \text{falls } \ x > 0\\ 0, & \text{falls } \ x = 0\end{cases}$$

$$\begin{aligned}f(x,0) &= C_0^1(x)\\ f(x,y+1) &= \Pi_3^3\left(x,y,N\left(C_0^2(x,y)\right)\right)\end{aligned}$$

9. **Beispiel für μ-Operator:**
   Sei $f(x,y) = c \in \mathbb{N}_0$ $(c > 0)$, dann ist $\mu f(x) = \min\{y | f(x,y) = 0 \wedge$
   $\forall j < y : f(x,j)$ ist definiert$\} = \perp$ da $c > 0$
   Da $f(x,y)$ eine konstante Funktion ist die nie den Funktionswert 0 annimmt, ist $\mu f(x,y)$ nicht definiert, da das Minimum nicht existiert.

   Sei $f'(x,y) = 0 \Rightarrow \mu f'(x) = \min\{y \mid f(x,y) = 0 \wedge \forall j < y : f(x,j)$ ist def.$\} = 0$
   Da $f'(x,y)$ die konstante Funktion 0 ist, ist das Minimum stets 0 und somit $\mu f'(x) = 0$.

10. **Beispiel für μ-Operator:**
    Sei $f(x,y) = x + y \quad x, y \in \mathbb{N}_0$.
    Dann ist $\mu f(x) = \min\{y | f(x,y) = x + y = 0 \wedge \forall j < y : f(x,j)$ ist definiert$\}$.

$$= \begin{cases} 0, & \text{falls } \ x = 0\\ \perp, & \text{falls } \ x \neq 0\end{cases}$$

11. **Beispiel für μ-Operator:**
$f(x,y) = \begin{cases} 0, & \text{falls } x = y \\ \perp, & \text{sonst} \end{cases}$
Dann ist $\mu f(x) = \begin{cases} 0, & \text{falls } x = 0 \\ \perp, & \text{sonst} \end{cases}$
Beachte: Für $x > 0$ sind die $f(x,j)$ mit $j < y$ nicht definiert.

Satz M: $\mathcal{R}_m = \mathcal{F}_\mu$
Die Klasse der durch Registermaschinen berechenbaren Funktionen ist gleich der Klasse der μ-rekursiven Funktionen.

Beweis: Kerngedanke: Simulation des einen Modells durch das Andere.
"$\mathcal{F}_\mu \subseteq \mathcal{R}_m$":
Zu zeigen, daß $\mathcal{R}_m$ die Grundfunktionen enthält, und gegen simultanes Einsetzen, primitive Rekursion und μ-Rekursion abgeschlossen ist.

1. Offenbar $N, \Pi_k^n, C_k^n \in \mathcal{R}_m$:
   - N (Nachfolgeroperation) – Registeroperation +1
   - Π_k^n (Projektion) – Es ist möglich, mit einzelnen Registern zu arbeiten, indem man für alle nicht betroffenen Register die Registeroperation 0 ausführt.[4]
   - C_k^n (Konstante) – Die Konstante 0 wird rzeugt, indem man die Registeroperation -1 solange ausführt, bis der Nulltest[5] 0 ergibt. Die Konstante k läßt sich dann durch k-maliges Ausführen der Registeroperation $+1$ erzeugen.

2. Simultanes Einsetzen:
   Es seinen $g, h_1, \ldots, h_m \in \mathcal{R}_m$, $g : \mathbb{N}_0^m \to \mathbb{N}_0$, $h_i : \mathbb{N}_0^k \to \mathbb{N}_0 (i = 1, \ldots, m)$. Dann kann man $f : \mathbb{N}_0^k \to \mathbb{N}_0$ mit $f = g(h_1, \ldots, h_m)$ folgendermaßen berechnen:
   - h_i möge j_i Register benutzen.
   - g benötige j Register, in den Registern $1, \ldots, k$ stehen die Eingaben $(x_1, \ldots, x_k) \in \mathbb{N}_0^k$.
   - Sei $l = j_1 + \cdots + j_m + j$ die Anzahl der insgesamt benötigten Register.

   Eine RM ϱ mit l Registern arbeitet wie folgt:
   - Kopiere $x_1, \ldots, x_k$ in Register 1 bis k, sowie in $j_1 + 1, \ldots, j_1 + k$ bzw. in $j_1+j_2+1, \ldots, j_1+j_2+k$ bzw. ... bzw. $j_1+\ldots+j_{m-1}+1, \ldots, j_1+\ldots+j_{m-1}+k$
   - Berechne nacheinander in den Registerblöcken j_k die Funktionen h_k, $(k = 1, \ldots, m)$

[4] siehe Definition F, Punkt (v) vom 01.12.2000
[5] siehe Definition F, Punkt (v) vom 01.12.2000

- Kopiere Register $j_1 + 1$ (Funktionswert von h_2) nach Reg[6] 2
  Kopiere Register $j_1 + j_2 + 1$ (Funktionswert von h_3) nach Reg 3
  $\vdots$
  Kopiere Register $j_1 + j_2 + \cdots + j_{m-1} + 1$ (Fkt.wert von h_k) nach Reg m
- Berechne g für das die Eingabe in den ersten m Registern steht. Die RM ϱ berechnet somit f. Es ist also $f \in \mathcal{R}_m$.

3. Primitive Rekursion:
   Seien $g, h \in \mathcal{R}_m$; $\quad g : \mathbb{N}_0^{k-1} \to \mathbb{N}_0$; $\quad h : \mathbb{N}_0^{k+1} \to \mathbb{N}_0$ zu $f(x_1, \ldots, x_{k-1}, y)$ mit:

$$\begin{aligned} f(x_1, \ldots, x_{k-1}, 0) &= g(x_1, \ldots, x_{k-1}) \\ f(x_1, \ldots, x_{k-1}, y+1) &= h(x_1, \ldots, x_{k-1}, y, f(x_1, \ldots, x_{k-1}, y)) \end{aligned}$$

   Berechne nacheinander (Simulation der Rekursion durch Iteration):

$$f(x_1, \ldots, x_{k-1}, 0), f(x_1, \ldots, x_{k-1}, 1), \ldots, f(x_1, \ldots, x_{k-1}, y)$$

   Die Funktion g möge j_1 Register und h möge j_2 Register benötigen.
   Sei $j = \max(j_1, j_2) + k + 2$.

   Folgende Registermaschine mit j Registern berechnet dann f_i:

   - Kopiere $x_1, \ldots, x_{k-1}, y$ nach Reg $j-k, \ldots, j-1$
     $\{x_1, \ldots, x_{k-1}, y$ werden gerettet, weil sie bei jeder Iteration gebraucht werden.$\}$
   - Setze Reg j auf Anzahl der bisherigen Iterationen (anfangs 0)
   - Berechne Funktion g mit den ersten j_1 Registern.
   - Lösche Reg k (setzte Wert auf 0 wo bisher y stand)
   - Schaffe Ergebnis (aus Reg 1) nach Reg $k+1$
   - **while** Reg $j <$ Reg $(j-1)$ **do** [d.h. #Operation ist noch kleiner y]

     Kopiere $x_1, \ldots, x_{k+1}$[7] nach Reg 1 bis Reg $k-1$
     Berechne h
     Erhöhe Reg j um 1
     Kopiere Reg j nach Reg k
     Schaffe Ergebnis nach Reg $k+1$

     od
   - Schaffe Reg $k+1$ nach Reg 1.

   Nach Beendigung des Algorithmus steht im Register 1 der Funktionswert von $f(x_1, \ldots, x_{k-1}, y)$.

[6] Bezeichnung der Register der RM
[7] aus Reg $j-k, \cdots$, Reg $j-1$

4. μ-Operator:
   Simulation erfolgt wie bei primitiver Rekursion nur daß es eine Funktion g gibt und daß als Schleifenbedingung "Reg $1 \neq 0$" zu verwenden ist.

Ergebnis: $\mathcal{R}_m$ ist abgeschlossen gegen alle Operationen $\Rightarrow \mathcal{F}_\mu \subseteq \mathcal{R}_m$

"$\mathcal{R}_m \subseteq \mathcal{F}_\mu$":
Idee: Zeige $\hat{\delta}$ (Überführungsfunktion) ist μ-rekursiv.
Sei $\varrho = (S, k, \delta, s_0, F)$ eine RM. Sei o.B.d.A. $S = \{0, 1, \ldots, r\}$ und $F = \{0\}$ und $s_0 = 1$. Dann ist $\hat{\delta} : \mathbb{N}_0^{k+1} \to \mathbb{N}_0^{k+1}$ gegeben durch

$$\begin{aligned} \hat{\delta}(s, x_1, \ldots, x_n) &= (s', y_1 \ldots, y_k) \text{ sei} \\ \delta(s, \text{sign}(x_1), \ldots, \text{sign}(x_k)) &= (s', j_1, \ldots, j_k) \text{ dann ist} \\ (y_1, \ldots, y_k) &= (x_1, \ldots, x_k) \oplus (j_1, \ldots, j_k) \end{aligned}$$

$\hat{\delta}$ ist durch eine endliche Fallunterscheidung beschrieben.
z.B. für $k = 2$
$\hat{\delta}(s, x_1, x_2)$
if s=1 **then**
if $\text{sign}(x_1) = 1$ **then**
if $\text{sign}(x_2) = 1$ **then** $(s', x_1 \oplus j_1, x_2 \oplus j_2)$

Alle Funktionen auf der rechten Seite sind primitiv-rekursiv.

Bilde nun die iterierte Anwendung von $\hat{\delta}$ auf sich selbst.

$$\begin{aligned} \phi(s, x_1, \ldots, x_k, 0) &= (s, x_1, \ldots, x_k) \\ \phi(s, x_1, \ldots, x_k, y+1) &= \hat{\delta}\left(\phi(s, x_1, \ldots, x_k, y)\right) \\ \phi(s, x_1, \ldots, x_k, y) &= \hat{\delta}^y(s, x_1, \ldots, x_k) \end{aligned}$$

ϕ ist primitv-rekursiv: $\phi : \mathbb{N}_0^{k+2} \to \mathbb{N}_0^{k+1}$

Simulation der wiederholten Anwendung von $\hat{\delta}$ bis zum Endzustand

$$a(s, x_1 \ldots, x_k) = \mu \Pi_1^{k+1} \circ \phi =$$

$$\min\left\{m \left(\Pi_1^{k+1}\left(\phi(s, x_1, \ldots, x_k, m)\right) = 0 \wedge \Pi_1^{k+1}\left(\phi(s, x_1, \ldots, x_k, j)\right) \text{ definiert } \forall j < m\right)\right\}$$

a ist μ-rekursiv
Sei $w : \mathbb{N}_0^{k+1} \to \mathbb{N}_0^k$ gegeben durch $w(s, x_1, \ldots, x_k) = (x_1, \ldots, x_k)$
Dann ist die von ϱ berechnete Funktion:
$h_\varrho(x_1, \ldots, x_k) = w\left(\phi\left(1, x_1, \ldots, x_k, a(1, x_1, \ldots, x_k)\right)\right)$

$\Rightarrow h_\varrho \in \mathcal{F}_\mu$

□