Informatik III - Tutorium IX & X (SR -107)

Tut Nr. 7 – Üb6, Universelle Fkt., smn, Rekursionssatz

David Münch

Universität Karlsruhe (TH)
Institut für Informatik
IAKS Beth

10. Dezember 2008

Universität Karlsruhe (TH)
Forschungsuniversität • gegründet 1825

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Organisatorisches

Email: muenchdavid@gmail.com

https://www.stud.uni-karlsruhe.de/~uhbro/

Tutorium 09: Mittwochs 8:00 Uhr - Raum -107
Tutorium 10: Mittwochs 9:45 Uhr - Raum -107

Übungsblattabgabe Donnerstag.

Dank an Benjamin Niedermann und Max Kramer für die LaTeX-Quellen!

Aufgabe 1

(i) Beweisen Sie, dass es **while**-Programme mit 2 Variablen gibt, für die kein äquivalentes **while**-Programm existiert, das keine **while**-Schleife enthält.

Aufgabe 1

(i) Beweisen Sie, dass es **while**-Programme mit 2 Variablen gibt, für die kein äquivalentes **while**-Programm existiert, das keine **while**-Schleife enthält.

Lösungsidee:
while-Programme ohne **while**-Schleife terminieren immer, weil sie ja ohne **while** keine Endlosschleife enthalten können. Zum Beweis kann man also ein **while**-Programm mit höchstens 2 Variablen angeben, das nicht terminiert. Bei unserer Definition von **while**-Programmen gibt es sogar Programme mit nur einer Variablen, die man als Gegenbeispiel verwenden kann. Bsp.:

- $x_0 := 0$; $x_1 := 1$; **while** $x_0 \neq x_1$ **do end** oder
- $x_0 := 0$; **while** $x_0 = 0$ **do end**

Lösung:
Betrachte folgendes **while**-Programm:
$x_0 := 0$; $x_1 := 1$; **while** $x_0 \neq x_1$ **do end**
Dieses Programm terminiert nicht. Programme ohne **while**-Schleife bestehen jedoch nur aus einer endlichen Folge von Zuweisungen (und wenn man die Erweiterungen auch betrachtet: if-then-else-Anweisungen). Solche Programme terminieren immer. Deshalb lässt sich oben stehendes **while**-Programm, das nicht terminiert, nicht in ein äquivalentes **while**-Programm ohne **while**-Schleife umwandeln.

Aufgabe 1

(ii) Geben sei das folgende **while**-Programm:

(a,b):=squaresum(x):

$a := 0; b := 0;$
$y := 0; z := 1;$
while $y = 0$ **do**
  while $z \neq 0$ **do**
    $a := a + 1;$
    $z := x - (a^2 + b^2)$
  end
  if $a^2 + b^2 = x$ **then** $y := 1$ **else** $b := b + 1; a := 0; z := 1$ **end**
end

Welche Funktion berechnet das angegebene **while**-Programm? Geben Sie ein äquivalentes **while**-Programm an, daß nur eine einzige **while**-Schleife enthält.

Hinweis: Die bereits im Programm verwendeten Makros dürfen für das neue Programm ebenfalls verwendet werden und zählen nicht als **while**-Schleifen.

Lösung:
Das **while**-Programm (a,b):=squaresum(x) berechnet die Funktion

$$\text{squaresum}(x) = \begin{cases} \min_b \left\{(a,b) | a,b \in \mathbb{N}_0 : a^2 + b^2 = x, a \geq b\right\}, & \text{falls } x > 0 \wedge \exists a,b \in \mathbb{N}_0 : a^2 + b^2 = x \\ \bot, & \text{sonst} \end{cases},$$

wobei das Tupel (a, b) zurückgegeben wird, bei dem b minimal ist.

Umwandlung in ein **goto**-Programm:
(a,b):=squaresum(x):

1: $a := 0$
2: $b := 0$
3: $y := 0$
4: $z := 1$
5: **if** $y \neq 0$ **goto** 17
6: **if** $z = 0$ **goto** 10
7: $a := a + 1;$
8: $z := x - (a^2 + b^2)$
9: **goto** 6
10: **if** $a^2 + b^2 \neq x$ **goto** 13
11: $y := 1$
12: **goto** 16
13: $b := b + 1$
14: $a := 0$
15: $z := 1$
16: **goto** 5
17: **stop**

Umwandlung in ein **while**-Programm mit einer **while**-Schleife:

(a,b):=squaresum(x): $k := 1;$

while $k \neq 0$

if $k = 1$ **then** $a := 0; k := k + 1$ **end**;
if $k = 2$ **then** $b := 0; k := k + 1$ **end**;
if $k = 3$ **then** $y := 0; k := k + 1$ **end**;
if $k = 4$ **then** $z := 1; k := k + 1$ **end**;
if $k = 5$ **then if** $y \neq 0$ **then** $k := 17$ **end end**;
if $k = 6$ **then if** $z = 0$ **then** $k := 10$ **end end**;
if $k = 7$ **then** $a := a + 1; k := k + 1$ **end**;
if $k = 8$ **then** $z := x - (a^2 + b^2); k := k + 1$ **end**;
if $k = 9$ **then** $k := 6$ **end**;
if $k = 10$ **then if** $a^2 + b^2 \neq x$ **goto** 13 **end**;
if $k = 11$ **then** $y := 1; k := k + 1$ **end**;
if $k = 12$ **then** $k := 16$ **end**;
if $k = 13$ **then** $b := b + 1; k := k + 1$ **end**;
if $k = 14$ **then** $a := 0; k := k + 1$ **end**;
if $k = 15$ **then** $z := 1; k := k + 1$ **end**;
if $k = 16$ **then** $k := 5$ **end**;
if $k = 17$ **then** $k := 0$ **end**

end

Man kann das ursprüngliche **while**-Programm durch "scharfes Hinschauen" in ein äquivalentes **while**-Programm mit nur einer **while**-Schleife überführen:

(a,b):=squaresum(x):

$a := 1; b := 0;$

$y := 0; z := 1;$

while $y = 0$ **do**

$z := x - (a^2 + b^2)$

if $z \neq 0$ **then** $a := a + 1$ **else**

if $a^2 + b^2 = x$ **then** $y := 1$ **else** $b := b + 1; a := 1$ **end**

end

end

Wiederholung

- Jedes **while**-Programm kann in eine *Gödelnummer / Index* umgewandelt werden

Wiederholung

- Jedes **while**-Programm kann in eine *Gödelnummer / Index* umgewandelt werden
- Idee: Gödelnummer ist standardisierte Darstellung des Quellcodes

Wiederholung

- Jedes **while**-Programm kann in eine *Gödelnummer / Index* umgewandelt werden
- Idee: Gödelnummer ist standardisierte Darstellung des Quellcodes
- Schreibweisen:
  - Sei P **while**-Programm, dann ist "P" Index
  - Sei n Index, dann ist P_n **while**-Programm

Wiederholung

- Jedes **while**-Programm kann in eine *Gödelnummer / Index* umgewandelt werden
- Idee: Gödelnummer ist standardisierte Darstellung des Quellcodes
- Schreibweisen:
  - Sei P **while**-Programm, dann ist "P" Index
  - Sei n Index, dann ist P_n **while**-Programm
- die von dem **while**-Programm P_i berechnete Fkt. $\mathbb{N}_0^k \to \mathbb{N}_0^m$ wird bezeichnet als $\varphi_i^{k,m}$.

Wiederholung

- Jedes **while**-Programm kann in eine *Gödelnummer / Index* umgewandelt werden
- Idee: Gödelnummer ist standardisierte Darstellung des Quellcodes
- Schreibweisen:
  - Sei P **while**-Programm, dann ist "P" Index
  - Sei n Index, dann ist P_n **while**-Programm
- die von dem **while**-Programm P_i berechnete Fkt. $\mathbb{N}_0^k \to \mathbb{N}_0^m$ wird bezeichnet als $\varphi_i^{k,m}$.
- Schreibeweise: φ ist an Stelle x

Wiederholung

- Jedes **while**-Programm kann in eine *Gödelnummer / Index* umgewandelt werden
- Idee: Gödelnummer ist standardisierte Darstellung des Quellcodes
- Schreibweisen:
  - Sei P **while**-Programm, dann ist "P" Index
  - Sei n Index, dann ist P_n **while**-Programm
- die von dem **while**-Programm P_i berechnete Fkt. $\mathbb{N}_0^k \to \mathbb{N}_0^m$ wird bezeichnet als $\varphi_i^{k,m}$.
- Schreibeweise: φ ist an Stelle x
  - definiert: $\varphi(x)\downarrow$
  - nicht definiert: $\varphi(x) = \perp$ oder $\varphi(x)\uparrow$

Definition

Die Funktion

$$\Phi^{k,m}(n, x_1, \ldots, x_k) = \varphi_n^{k,m}(x_1, \ldots, x_k)$$

heißt universelle Funktion.

Definition

Die Funktion

$$\Phi^{k,m}(n, x_1, \ldots, x_k) = \varphi_n^{k,m}(x_1, \ldots, x_k)$$

heißt universelle Funktion.

Bem.: Das **while**-Programm $\mathcal{U}$, das die universelle Funktion Φ berechnet, heißt Interpreter.

Definition

Die Funktion

$$\Phi^{k,m}(n, x_1, \ldots, x_k) = \varphi_n^{k,m}(x_1, \ldots, x_k)$$

heißt universelle Funktion.

Bem.: Das **while**-Programm $\mathcal{U}$, das die universelle Funktion Φ berechnet, heißt Interpreter.

Idee:

- Man übergibt $\mathcal{U}$ die Gödelnummer eines Programmes P und dessen Parameter
- $\mathcal{U}$ beginnt nun das Programm P zu simmulieren und gibt am Ende die Ausgabe von P zurück

Beispiel: Sei $\varphi_n(x) = x^2$.
Dann gilt $\varphi_n(x) = \Phi(n, x) = x^2$

Programmumschreibefunktion

$f : \mathbb{N} \to \mathbb{N}$ heißt Programmumschreibefunktion oder Programmtransformation.

Programmumschreibefunktion

$f : \mathbb{N} \to \mathbb{N}$ heißt Programmumschreibefunktion oder Programmtransformation.

Idee:

- f erhält als Eingabe ein Programm P codiert als Gödelnummer
- f wandelt dieses Programm P in ein Programm P' ab
- f gibt das Programm P' zurück

Programmumschreibefunktion

$f : \mathbb{N} \to \mathbb{N}$ heißt Programmumschreibefunktion oder Programmtransformation.

Idee:

- f erhält als Eingabe ein Programm P codiert als Gödelnummer
- f wandelt dieses Programm P in ein Programm P' ab
- f gibt das Programm P' zurück

Beispiel: $f(n) = \text{“}x_1 = 1; P\text{”}$, wobei P das Programm zum Index n ist

Funktionstransformation

Sei φ_n eine Funktion bezüglich eines Programmes mit Index n und $g : \mathbb{N} \to \mathbb{N}$ eine Programmtransformation, dann ist $\varphi_{g(n)}$ eine Funktionstransformation.

Häufig verwendet man $T_s\varphi_i := \varphi_{s(i)}$

Funktionstransformation

Sei φ_n eine Funktion bezüglich eines Programmes mit Index n und $g : \mathbb{N} \to \mathbb{N}$ eine Programmtransformation, dann ist $\varphi_{g(n)}$ eine Funktionstransformation.

Häufig verwendet man $T_s\varphi_i := \varphi_{s(i)}$

Idee:

- Die Funktion eines Programmes P mit Index n, kann durch eine Programmtransformation verändert werden.
- Dies kann verwendet werden, um rekusive Programme aufzulösen.

Das smn-Theorem

Für alle $m, n \in \mathbb{N}_0$ gibt es eine berechenbare totale Funktion $s : \mathbb{N}_0^{m+1}$, so dass

$$\varphi_i^{m+n}(a_1, \ldots, a_m, x_1, \ldots, x_m) = \varphi_{s(i,a_1,\ldots,a_m)}^{n}(x_1, \ldots, x_m)$$

für alle $i, a_1, \ldots, a_m$ gilt.

Das smn-Theorem

Für alle $m, n \in \mathbb{N}_0$ gibt es eine berechenbare totale Funktion $s : \mathbb{N}_0^{m+1}$, so dass

$$\varphi_i^{m+n}(a_1, \ldots, a_m, x_1, \ldots, x_m) = \varphi_{s(i,a_1,\ldots,a_m)}^{n}(x_1, \ldots, x_m)$$

für alle $i, a_1, \ldots, a_m$ gilt.

Bem.: s bezeichnet eine Programmumschreibefunktion, in diesem Fall kodiert sie $a_1, \ldots, a_m$ in den Quellcode, so dass sie bei Berechnung von ϕ_i nicht mit übergeben werden müssen.

Das smn-Theorem

Für alle $m, n \in \mathbb{N}_0$ gibt es eine berechenbare totale Funktion $s : \mathbb{N}_0^{m+1}$, so dass

$$\varphi_i^{m+n}(a_1, \ldots, a_m, x_1, \ldots, x_m) = \varphi_{s(i,a_1,\ldots,a_m)}^{n}(x_1, \ldots, x_m)$$

für alle $i, a_1, \ldots, a_m$ gilt.

Bem.: s bezeichnet eine Programmumschreibefunktion, in diesem Fall kodiert sie $a_1, \ldots, a_m$ in den Quellcode, so dass sie bei Berechnung von ϕ_i nicht mit übergeben werden müssen.

Idee: Prinzipiell ist es egal, ob die Werte der Parameter direkt in das Programm kodiert oder übergeben werden.

Der Satz von Kleene(Rekursionsatz)

Für jede berechenbare totale Funktion f existiert ein $n \in \mathbb{N}$, so dass

$$\varphi_n = \varphi_{f(n)}$$

Der Satz von Kleene(Rekursionsatz)

Für jede berechenbare totale Funktion f existiert ein $n \in \mathbb{N}$, so dass

$$\varphi_n = \varphi_{f(n)}$$

Idee: f ist Programmtransformation, dann ist n semantischer Fixpunkt

oder: Für jedes Programm P, das ein anderes Programm abändert, gibt es ein Programm Q, so dass P angewendet auf Q keine Veränderung in Q bewirkt.

Übersicht für folgende Aufgaben

Das smn-Theorem

Für alle $m, n \in \mathbb{N}_0$ gibt es eine berechenbare totale Funktion $s : \mathbb{N}_0^{m+1}$, so dass

$$\varphi_i^{m+n}(a_1, \ldots, a_m, x_1, \ldots, x_m) = \varphi_{s(i,a_1,\ldots,a_m)}^{n}(x_1, \ldots, x_m)$$

für alle $i, a_1, \ldots, a_m$ gilt.

Der Satz von Kleene(Rekursionssatz)

Für jede berechenbare totale Funktion f existiert ein $n \in \mathbb{N}$, so dass

$$\varphi_n = \varphi_{f(n)}$$

Aufgabe

Sei $f : \mathbb{N}_0 \to \mathbb{N}_0$ eine totale Funktion, für die für alle $n \in \mathbb{N}_0$ gilt:

$$\varphi_{f(n)}(x) := \begin{cases} \varphi_n(0) + 1, & \text{falls} \varphi_n(0) \downarrow \\ 2, & \text{sonst} \end{cases}$$

Zeige, dass f nicht berechenbar ist.

Lösung:

Annahme: f ist berechenbar.

Da f auch total ist, gilt nach dem Rekursionssatz:$\exists n \in \mathbb{N}_0 : \varphi_n = \varphi_{f(n)}$

- 1.Fall: $\varphi_n(0) \downarrow$
  Daraus folgt $\varphi_n(0) = \varphi_{f(n)}(0) = \varphi_n(0) + 1$ im Widerspruch zu $\varphi_n(0) \downarrow$.
- 2.Fall: $\varphi_n(0) \uparrow$
  Daraus folgt $\varphi_n(0) = \varphi_{f(n)}(0) = 2$ im Widerspruch zu $\varphi_n(0) \uparrow$.

Aufgabe

Beweisen Sie, dass es einen Index $n \in \mathbb{N}_0$ gibt mit $\varphi_n(x) = (n + x)^2$.

Lösung:

Betrachte $\Theta := (i + x)^2$.

Da Θ berechenbar, gibt es einen Index e mit $\varphi_e = \Theta$.

Nach dem s-m-n-Theorem gibt es eine totale berechenbare Funktion s mit

$\varphi_e(i, x) = \varphi_{s(e,i)}(x)$.

Setze $g(i) := s(e, i)$.

Da g somit auch total und berechenbar ist, gibt es nach dem Rekursionssatz ein $n \in \mathbb{N}_0$ mit

$\varphi_n = \varphi_{g(n)}$.

Insgesamt ergibt sich also, dass ein Index $n \in \mathbb{N}_0$ existiert mit

$\varphi_n(x) = \varphi_{g(n)}(x) = \varphi_{s(e,n)}(x) = \varphi_e(n, x) = \Theta(n, x) = (n + x)^2$.

Fragestellung

Für rekursive Programme ist häufig eine iterative Variante, also z.B. ein **while**-Programmwünschenswert. Kann jedes rekursive Programm in ein Iteratives Programm umgewandelt werden?

Fragestellung

Für rekursive Programme ist häufig eine iterative Variante, also z.B. ein **while**-Programmwünschenswert. Kann jedes rekursive Programm in ein Iteratives Programm umgewandelt werden?

Lösungsansatz

- Sei P ein rekursives Programm
- Kopiere Code von P an die Stellen in P, wo es sich selbst aufruft
- wiederhole dies solange, bis gewünschte Rekursionstiefe erreicht ist

Formal

Sei s eine Programmumschreibefunktion, die sich wie in der Lösungsidee verhält und P ein rekursives Programm.

Die Funktion von P lässt sich beschreiben als

$$\varphi_P(x) = \begin{cases} T_s^{m+1}\varphi \perp & \text{,bei Eingabe x terminiert P nach m Schritten} \\ \perp & \text{,}sonst \end{cases}$$

Formal

Sei s eine Programmumschreibefunktion, die sich wie in der Lösungsidee verhält und P ein rekursives Programm.

Die Funktion von P lässt sich beschreiben als

$$\varphi_P(x) = \begin{cases} T_s^{m+1}\varphi \perp & \text{,bei Eingabe x terminiert P nach m Schritten} \\ \perp & \text{,}sonst \end{cases}$$

Fixpunkt

$\varphi_P(x) = sup_m T_s^{m+1}\varphi \perp$ ist kleinster Fixpunkt.

Aufgabe

Gegeben sei das folgende rekursive while-Programm:

procedure R_1
if $x_1 = 0$; **then** $x_1 := 1$;
else if $x_1 < 50$ **then** $x_1 := x_1 + 7$
else $x_1 := x_1 - 8; R_1; R_1; x_1 := x_1 - 1$;**end**
end

Berechne die zugehörige Funktionstransformationen und ihren kleinsten Fixpunkt. Welche Funktionen werden von den rekursiven Programmen berechnet?

Lösung:

Die Transformation T_1 ergibt sich zu

$$T_1(\varphi)(x) := \begin{cases} 1, & \text{falls } x = 0 \\ x + 7, & \text{falls } 0 < x < 50 \\ \varphi^2(x - 8) - 1, & \text{sonst} \end{cases}$$

und ihr kleinster Fixpunkt ist:

$$sup T_1^m(\bot)(x) := \begin{cases} 1, & \text{falls } x = 0 \\ x + 7, & \text{falls } 0 < x < 50 \\ 55, & \text{sonst} \end{cases}$$

Das Programm berechnet demnach die Funktion $\varphi = sup T_1^m(\bot)(x)$.

Aufgabe

Gegeben sei das folgende rekursive while-Programm:

procedure R_2
  if $x_1 = 0$; **then** $x_1 := 0$;
  else if $x_1 = 1$ **then** $x_1 := 1$
    else $x_1 := x_1 - 1; x_2 := x_1; R_2; x_3 := x_1;$
      $x_1 := x_2 - 1; R_2; x_1 := x_1 + x_3;$
    end
  end

Berechne die zugehörige Funktionstransformationen und ihren kleinsten Fixpunkt. Welche Funktionen werden von den rekursiven Programmen berechnet?

Lösung:
Die Transformation T_2 ergibt sich zu

$$T_2(\varphi)(x) := \begin{cases} 0, & \text{falls } x = 0 \\ 1, & \text{falls } x = 1 \\ \varphi(x-1) + \varphi(x-2), & \text{sonst} \end{cases}$$

und ihre kleinster Fixpunkt ist $supT_2^m(\perp)(x) = \frac{1}{\sqrt{5}} * (\gamma^n - \gamma'^n)$,
wobei $\gamma = \frac{1+\sqrt{5}}{2}$ der goldene Schnitt ist und $\gamma = \frac{1-\sqrt{5}}{2}$.
Das Programm berechnet demnach die Fibonacci-Zahlen.

Reflexion

Was haben wir heute gelernt?

Reflexion

Was haben wir heute gelernt?

- Universelle Funktion

Reflexion

Was haben wir heute gelernt?

- Universelle Funktion
- smn-Theorem

Reflexion

Was haben wir heute gelernt?

- Universelle Funktion
- smn-Theorem
- Rekursionssatz

Reflexion

Was haben wir heute gelernt?

- Universelle Funktion
- smn-Theorem
- Rekursionssatz
- Programmumschreibefunktionen

Noch Fragen?

Vorschau

Vorschau

Bis zum nächsten Mal