Grundzüge der Theoretischen Informatik

19. November 2021

Markus Bläser

Universität des Saarlandes

Kapitel 9: Gödelisierungen

Abzählbare Menge

inj: $f(x)=f(y) \Rightarrow x=y$

surj: $\forall y \in \mathbb{N} : \exists x \in S : f(x)=y$

Definition

Eine Menge S ist *abzählbar*, falls eine injektive Funktion von $S \xrightarrow{f} \mathbb{N}$ gibt. Falls es eine bijektive Funktion $S \to \mathbb{N}$ gibt, dann ist S *abzählbar unendlich*.

Bemerkung:

$\{y \mid \exists x \in S\ f(x)=y\}$

- Wenn es eine injektive Funktion $f : S \to \mathbb{N}$ gibt, so dass $\mathrm{im}\, f \overset{!!}{=} \{f(x) \mid x \in S\}$ unendlich ist, dann gibt es eine Bijektion von $S \to \mathbb{N}$.
- Wenn S abzählbar ist, dann ist S endlich oder abzählbar unendlich.

Gödelisierung

Menge aller WHILE-Prog

$\mathcal{W} = \bigcup_{i \in \mathbb{N}} \mathcal{W}_i$

~~inj~~ + surj

- Bijektive Funktion $\text{göd} : \mathcal{W} \to \mathbb{N}$
- WHILE-Programm U, das auf Eingabe $\underline{i}, x \in \mathbb{N}$ $\varphi_P(x)$ ausgibt, wobei $P = \text{göd}^{-1}(i)$.

Eingabe für "i"

Gödelnummer

Gödelisierung

Fakt

$(r, m) \mapsto \langle r, m\rangle_5 := 5m + r$ *ist eine Bijektion* $\{0, 1, 2, 3, 4\} \times \mathbb{N} \to \mathbb{N}$.

$\langle \cdot, \cdot \rangle : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$

$\langle \cdot, \cdot \rangle_5 : \{0,1,2,3,4\} \times \mathbb{N} \to \mathbb{N}$

Art der Anweisung "Parameter"

Einfache Anweisungen:

Add.

1. $x_i := x_j + x_k$ wird kodiert durch $\langle 0, \langle i, \langle j, k\rangle\rangle\rangle_5$.
2. $x_i := x_j - x_k$ wird kodiert durch $\langle 1, \langle i, \langle j, k\rangle\rangle\rangle_5$.
3. $x_i := c$ wird kodiert durch $\langle 2, \langle i, c\rangle\rangle_5$.

While-Schleife und Konkatenation:

1. Falls $P =$ **while** $x_i \neq 0$ **do** P_1 **od**, dann ist $\text{göd}(P) = \langle 3, \langle i, \text{göd}(P_1)\rangle\rangle_5$.
2. Falls $P = [P_1; P_2]$, dann ist $\text{göd}(P) = \langle 4, \langle \text{göd}(P_1), \text{göd}(P_2)\rangle\rangle_5$.

Gödelisierung (2)

Lemma (9.6)

göd *ist wohl-definiert.*

Lemma (9.7)

göd *ist injektiv.*

Folgerung (9.8) $\quad göd: W \to \mathbb{N}$

Die Menge der WHILE-Programme is abzählbar.

Lemma (9.9)

göd *ist surjektiv.*

Lemma 9.6

IA: $P \in W_0$ göd(P) ist wohl-def.

IS. $P \in W_n \setminus W_{n-1}$ für $n > 0$

$P =$ while $x_i \neq 0$ do P_1 od

$P_1 \in W_{n-1}$

$\Rightarrow$ göd(P_1) ist wohldef.

$\Rightarrow$ göd(P) ist wohldef

$P = [P_1; P_2]$

$P_1, P_2 \in W_{n-1}$

$\Rightarrow$ göd(P_1), göd(P_2) wohldef.

$\Rightarrow$ göd(P) ist wohldef.

Lemma 9.7

Wir zeigen für alle $n \in \mathbb{N}$ $göd(P) = göd(Q) \Rightarrow P = Q$ für $P, Q \in W_n$

IA: $P, Q \in W_0$ mit $göd(P) = göd(Q)$

$\exists s_1, t_1, s_2, t_2 \in \mathbb{N}$: $\langle s_1, t_1 \rangle_5 = göd(P)$ und $\langle s_2, t_2 \rangle_5 = göd(Q)$

$\Rightarrow s_1 = s_2$ und $t_1 = t_2$ ($\langle .,. \rangle_5$ bij)

gleiche Anweisung gleiche Parameter ($\langle ., \langle .,. \rangle \rangle$ ist bij $\mathbb{N}^3 \to \mathbb{N}$)

$\Rightarrow P = Q$

IS: $P \in W_n \setminus W_{n-1}$, $Q \in W_n$ mit $göd(P) = göd(Q)$

$s_1 = s_2 = 3$ P = while ...

$t_1 = \langle i, göd(P_1) \rangle$ $t_2 = \langle j, göd(Q_1) \rangle$

$\Rightarrow i = j$ und $göd(P_1) = göd(Q_1) \overset{IV}{\Rightarrow} P_1 = Q_1$

$\Rightarrow P = Q$

$s_1 = s_2 = 4$ $Q = [Q_1; Q_2]$ $P = [P_1; P_2]$

$t_1 = \langle göd(P_1), göd(P_2) \rangle$

$t_2 = \langle göd(Q_1), göd(Q_2) \rangle$

$\overset{IV}{\Rightarrow} P_1 = Q_1$ und $P_2 = Q_2 \Rightarrow P = Q$

Lemma 9.9

Ind. über $n \in \mathbb{N}$: Es gibt ein WHILE-Prog P mit $göd(P) = n$

IA: $n = 0$ $\quad 0 = \langle 0, \langle 0, \langle 0, 0 \rangle\rangle\rangle = göd(x_0 := x_0 + x_0)$

IS: Sei $n \in \mathbb{N}$ $\quad n = \langle r, m \rangle_5$ mit $0 \leq r \leq 4$

$r = 0$

$\exists i, j, k \in \mathbb{N} : \langle i, \langle j, k \rangle\rangle = m$

$n = \langle 0, m \rangle_5 = \langle 0, \langle i, \langle j, k \rangle\rangle\rangle_5 = göd(x_i := x_j + x_k)$

$r = 1, 2$ sehr ähnlich $\qquad \langle x, y \rangle \geq x, y$

$r = 3$ $\quad \exists i, s \in \mathbb{N}$ $\quad \langle i, s \rangle = m$

$n > m \geq s \quad \Rightarrow s < n \quad \overset{IV}{\Rightarrow} \exists P_1 \in W : göd(P_1) = s$

$n = \langle 3, \langle i, göd(P_1) \rangle\rangle_5 = göd(\text{while } x_i \neq 0 \text{ do } P_1 \text{ od})$

$r = 4$ ähnlich zu $r = 3$

Programmiersysteme

Alles, was wir in diesem Teil der Vorlesung beweisen, gilt allgemein:

Definition

1. Eine Folge $(\psi_i)_{i \in \mathbb{N}}$ heißt *Programmiersystem*, falls die Menge aller ψ_i gleich der Menge aller WHILE-berechenbaren Funktionen R ist.
2. Es ist *universell*, falls es ein universelles Programm gibt, d.h. es gibt ein u, so dass $\psi_u(\langle j, x\rangle) = \psi_j(x)$ für alle $j, x \in \mathbb{N}$.
3. Ein universelles Programmiersystem heißt *zulässig* oder *akzeptabel*, falls es ein c gibt, so dass $\psi_{\psi_c(\langle j,k\rangle)} = \psi_j \circ \psi_k$.

Programmiersysteme

Alles, was wir in diesem Teil der Vorlesung beweisen, gilt allgemein:

Definition

1. Eine Folge $(\psi_i)_{i \in \mathbb{N}}$ heißt *Programmiersystem*, falls die Menge aller ψ_i gleich der Menge aller WHILE-berechenbaren Funktionen R ist.
   alle Java-Programme (Σ^* statt $\mathbb{N}$)
2. Es ist *universell*, falls es ein universelles Programm gibt, d.h. es gibt ein u, so dass $\psi_u(\langle j, x\rangle) = \psi_j(x)$ für alle $j, x \in \mathbb{N}$.
   Java-Interpreter in Java geschrieben
3. Ein universelles Programmiersystem heißt *zulässig* oder *akzeptabel*, falls es ein c gibt, so dass $\psi_{\psi_c(\langle j,k\rangle)} = \psi_j \circ \psi_k$.
   Java-Programm, das Java-Programme “konkateniert”.

Kapitel 10: Diagonalisierung


Es gibt Funktionen, die nicht
While-berechenbar sind.
Aber ich sag nicht welche.
Georg Cantor, 1845–1918

Beweis durch "Abzählen"

Satz (10.1) $\mathcal{F}$

Die Menge aller totalen Funktionen $\mathbb{N} \to \{0, 1\}$ *ist nicht abzählbar.*

- Zweites Cantorsches Diagonalargument
- Annahme: F ist abzählbar
- Sei $n : F \to \mathbb{N}$ Bijektion, $f_i := n^{-1}(i)$.

Beweis durch "Abzählen"

Satz (10.1)

Die Menge aller totalen Funktionen $\mathbb{N} \to \{0,1\}$ ist nicht abzählbar.

- Zweites Cantorsches Diagonalargument
- Annahme: F ist abzählbar
- Sei $n : F \to \mathbb{N}$ Bijektion, $f_i := n^{-1}(i)$.

	0	1	2	3	...
0	$\underline{f_0(0)}$	$f_0(1)$	$f_0(2)$	$f_0(3)$	...
1	$f_1(0)$	$\underline{f_1(1)}$	$f_1(2)$	$f_1(3)$	...
2	$f_2(0)$	$f_2(1)$	$\underline{f_2(2)}$	$f_2(3)$	...
3	$f_3(0)$	$f_3(1)$	$f_3(2)$	$\underline{f_3(3)}$	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

$g \in F$

$i = n(F)$

$g(i) = f_i(i) \neq 1 - f_i(i) = g(i)$

$\lightning \quad g \notin F$

$g(i) = 1 - f_i(i)$

$\Rightarrow n$ kann nicht ex.

Beweis durch "Abzählen" (2)

$$f_L(x) = \begin{cases} 1 & \text{falls } x \in L \\ 0 & \text{sonst} \end{cases}$$

Folgerung (10.2)

überabzählbar viele

W ist abzählbar

jedes Prog berechnet

Es gibt eine totale Funktion $\mathbb{N} \to \{0,1\}$, die nicht WHILE-berechenbar ist.

1 Fkt

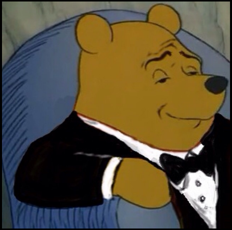
$\Rightarrow$ R ist abzählbar

Folgerung (10.3)

Es gibt eine Teilmenge von $\mathbb{N}$, die nicht rekursiv ist.

Alternativer Beweis . . .

Proof by Counting



Proof by Diagonalization

Alternativer Beweis durch Diagonalisierung

Folgerung (10.2)

Es gibt eine totale Funktion $\mathbb{N} \to \{0, 1\}$, die nicht WHILE-berechenbar ist.

- Definiere $f_0, f_1, f_2, \ldots$ durch

partiell

$$f_i(j) = \varphi_{\text{göd}^{-1}(i)}(j).$$

- Definiere c durch

$$c(n) = \begin{cases} 1 & \underbrace{\text{falls } f_n(n) = 0 \text{ oder undefiniert}}_{\text{„Diagonalsprache"}} \text{ ist} \\ 0 & \text{sonst} \end{cases}$$

$c \notin R$ Ang. $c \in R$

$\Rightarrow \exists P \in W \quad \varphi_P = c \qquad i = \text{göd}(P)$

$c(i) = \varphi_{\text{göd}^{-1}(i)}(i)$

$c(i) \to 0 \ \lightning$ 1

$c(i) \to 1 \ \lightning$ undef. oder 0

Kapitel 11: Ein universelles WHILE-Programm

Ein universelles WHILE-Programm

Eingabe: Gödelnummer g, Zahl m
Ausgabe: $\varphi_{\text{göd}^{-1}(g)}(m)$

Variablen in U:

- X: Array, das die Inhalte der Variablen von $P := \text{göd}^{-1}(g)$ speichert.
- S: Stack, der Teile von P speichert und den Programmfluss steuert.
- cur: Teil von P, der momentan simuliert werden soll.
- $term$: 0, falls die Simulation beendet ist, 1 sonst.
- $type$: speichert den Typ der aktuellen Anweiung (0 bis 4)

Einfache Anweisungen

Inhalt von cur:

- $\langle 0, \langle i, \langle j, k\rangle\rangle\rangle_5$ (Addition)
- $\langle 1, \langle i, \langle j, k\rangle\rangle\rangle_5$ (Subtraktion)
- $\langle 2, \langle i, c\rangle\rangle_5$ (Zuweisung)
- $\langle 3, \langle i, \text{göd}(P_1)\rangle\rangle_5$ (Whileschleife)
- $\langle 4, \langle \text{göd}(P_1), \text{göd}(P_2)\rangle\rangle_5$ (Konkatenation)

Routine für Addition

Input: $\langle i, \langle j, k\rangle\rangle_5$ gespeichert in x_2

i $x_3 := \pi_1^{(5)}(x_2)$;
j $x_4 := \pi_1(\pi_2^{(5)}(x_2))$;
k $x_5 := \pi_2(\pi_2^{(5)}(x_2))$;
$X[x_3] := X[x_4] + X[x_5]$

Universelles Programm $\mathfrak{U}$

while $x_j \neq 0$ do P_1 od

Input: g, m
Output: $\varphi_P(m)$

$X := 0$; {Clear entries}
$X[0] := m$; {Prepare input}
$S := \langle 0, 0\rangle$; {Empty stack}
$term := 1$;
$cur := g$;
while $term \neq 0$ **do**
  $type := \pi_1^{(5)}(cur)$;
  if $type = 0$ **then**
    simulate addition.
  fi
  if $type = 1$ **then**
    simulate subtraction.
  fi
  if $type = 2$ **then**
    simulate initialization.
  fi
  if $type = 3$ **then**
    $i := \pi_1(\pi_2^{(5)}(cur))$;
    if $X[i] \neq 0$ **then**
      $\mathrm{push}(S, cur)$;
      $\mathrm{push}(S, \pi_2(\pi_2^{(5)}(cur)))$
    fi
  fi
  if $type = 4$ **then**
    $\mathrm{push}(S, \pi_2(\pi_2^{(5)}(cur)))$;
    $\mathrm{push}(S, \pi_1(\pi_2^{(5)}(cur)))$
  fi
  if $\mathrm{isempty}(S) = 0$ **then**
    $cur := \mathrm{top}(S)$; $\mathrm{pop}(S)$;
  else
    $term := 0$
  fi
od;
$x_0 := X[0]$;

P_1
while ...
S

P_1
P_2
S

Wie beweist man die Korrektheit von so etwas?

Lemma (11.1)

Sei

- T *der Zustand, der dem Inhalt von* X *entspricht,*
- σ *der Inhalt des Stacks* S *und*
- $\mathrm{P} = \text{göd}^{-1}(\text{cur})$

in Zeile 6 (Beginn der While-Schleife). Sei

- T' *der Zustand, der dem Inhalt von* X *entspricht,*
- *wenn der Inhalt von* S *zum ersten Mal wieder* σ *ist*

in Zeile ~~2~~6 (Ende der While-Schleife).
Dann gilt

- $\mathrm{T}' = \Phi_{\mathrm{P}}(\mathrm{T})$, *sofern der Inhalt von* S *irgendwann wieder* σ *ist.*
- *Sonst ist* $\Phi_{\mathrm{P}}(\mathrm{T})$ *undefiniert.*

→ strukturelle Induktion

Kleenesche Normalform

Folgerung (11.3, Kleenesche Normalform)

Sei f *eine WHILE-berechenbare Funktion. Dann gibt es FOR-Programme* P_1, P_2 *und* P_3, *so dass das Programm*

$$P_1;\ \textbf{\textit{while}}\ x_1 \neq 0\ \textbf{\textit{do}}\ P_2\ \textbf{\textit{od}};\ P_3$$

f *berechnet.*[1]

$\exists P \in W; \varphi_P = f$

Hardcode $g = \text{göd}(P)$ in U

[1] Formal haben wir nie die Semantik von gemischten WHILE- und FOR-Programmen definiert. Aber das sollten Sie inzwischen können.