Theorie der Informatik

Was bisher geschah... (Teil III)

Malte Helmert Christian Tschudin

Universität Basel

15. April 2013

1 Wegweiser

Überblick: Vorlesung

Vorlesungsteile

I. ~~Logik~~
II. ~~Automatentheorie und formale Sprachen~~
III. Berechenbarkeitstheorie ⟵ Sie sind hier!
IV. Komplexitätstheorie

Überblick: Berechenbarkeitstheorie

III. Berechenbarkeitstheorie
III.1. Turing-Berechenbarkeit $\rightsquigarrow$ Wiederholung
III.2. LOOP-, WHILE-, GOTO-Berechenbarkeit $\rightsquigarrow$ Wiederholung
III.3. primitive Rekursion und μ-Rekursion $\rightsquigarrow$ Wiederholung
III.4. Entscheidbarkeit, Reduktionen, Halteproblem $\longleftarrow$ hier!
III.5. Postsches Korrespondenzproblem
~~III.6. Unentscheidbare Grammatik-Probleme~~ $\rightsquigarrow$ Übungen
~~III.7. Gödelscher Satz und diophantische Gleichungen~~

Berechnung

Was ist eine Berechnung?

- intuitives Berechenbarkeitsmodell (Papier und Bleistift)
- vs. Berechnung auf physikalischen Computern
- vs. formale mathematische Modelle

Formale Berechnungsmodelle

- Turingmaschinen
- LOOP-, WHILE-, GOTO-Programme
- primitiv rekursive Funktionen, μ-rekursive Funktionen

2 Turingmaschinen

Formale Berechnungsmodelle

Formale Berechnungsmodelle: Turingmaschinen

- Turingmaschinen
- LOOP-, WHILE-, GOTO-Programme
- primitiv rekursive Funktionen, μ-rekursive Funktionen

Turingmaschinen: Definition

Definition (Turingmaschine)

Turingmaschine ist 7-Tupel $M = \langle Z, \Sigma, \Gamma, \delta, z_0, \square, E \rangle$:

- Z endliche, nicht-leere Menge von Zuständen
- $\Sigma \neq \emptyset$ endliches Eingabealphabet
- $\Gamma \supset \Sigma$ endliches Bandalphabet
- $\delta : (Z \setminus E) \times \Gamma \to Z \times \Gamma \times \{L, R, N\}$ Übergangsfunktion
- $z_0 \in Z$ Startzustand
- $\square \in \Gamma \setminus \Sigma$ Blank-Zeichen
- $E \subseteq Z$ Endzustände

$\rightsquigarrow$ auch Variante mit mehreren Bändern angesprochen

Turingmaschinen: Konfigurationen und Berechnungen

Wie arbeiten Turingmaschinen?

- Konfiguration: $\alpha z \beta$ mit $\alpha \in \Gamma^*$, $z \in Z$, $\beta \in \Gamma^+$
- Berechnungsschritt: $c \vdash c'$, wenn aus Konfiguration c in einem Berechnungsschritt Konfiguration c' entsteht
- Berechnung: $c \vdash^* c'$, wenn aus Konfiguration c in 0 oder mehr Schritten Konfiguration c' entsteht ($c = c_0 \vdash c_1 \vdash c_2 \vdash \cdots \vdash c_{n-1} \vdash c_n = c'$, $n \geq 0$)

(Definition von $\vdash$, also wie ein Berechnungsschritt die aktuelle Konfiguration ändert, wird hier nicht wiederholt.)

Turingmaschinen: berechnete Funktion

Definition (von einer Turingmaschine berechnete Funktion)

Eine Turingmaschine mit Eingabealphabet Σ berechnet die (partielle) Funktion $f : \Sigma^* \to \Sigma^*$, für die gilt:

für alle $x, y \in \Sigma^*$: $f(x) = y$ gdw. $z_0 x \vdash^* \square \ldots \square z_e y \square \ldots \square$

mit $z_e \in E$. (Spezialfall: $z_0\square$ statt $z_0 x$ wenn $x = \epsilon$)

- Was passiert, wenn Zeichen aus $\Gamma \setminus \Sigma$ (z. B. $\square$) in y stehen?
- Was passiert, wenn der Lesekopf am Ende nicht auf dem ersten Zeichen von y steht?
- Ist f durch die Definition eindeutig definiert? Warum?

Turingmaschinen: Turing-berechenbar

Definition (Turing-berechenbar)

Eine (partielle) Funktion $f : \Sigma^* \to \Sigma^*$ heisst Turing-berechenbar, wenn eine Turingmaschine existiert, die sie berechnet.

Analoge Definition für (partielle) Funktionen $f : \mathbb{N}_0^k \to \mathbb{N}_0$ über natürlichen Zahlen. In diesem Fall werden Eingabe und Ausgabe über $\Sigma = \{0, 1, \#\}$ binärkodiert.

Beispiel Binärkodierung: $f(101\#10\#11) = 100$ (als Wörter) entspricht $f(5, 2, 3) = 4$ (als Zahlen)

3 LOOP-, WHILE-, GOTO-Programme

Formale Berechnungsmodelle: LOOP/WHILE/GOTO

Formale Berechnungsmodelle

- Turingmaschinen
- LOOP-, WHILE-, GOTO-Programme
- primitiv rekursive Funktionen, μ-rekursive Funktionen

LOOP-, WHILE- und GOTO-Programme: Grundkonzepte

- LOOP-, WHILE- und GOTO-Programme entsprechen strukturell (einfachen) „traditionellen“ Programmiersprachen
- verwenden endlich viele Variablen $x_0, x_1, x_2, \ldots$, die Werte in $\mathbb{N}_0$ annehmen
- unterscheiden sich in Art der erlaubten „Anweisungen“

LOOP-Programme: Definition

Definition (LOOP-Programm)

LOOP-Programme sind definiert durch endliche Anwendung folgender Regeln:

- $x_i := x_i + c$ ist ein LOOP-Programm für jedes $i, c \in \mathbb{N}_0$ (Addition)
- $x_i := x_i - c$ ist ein LOOP-Programm für jedes $i, c \in \mathbb{N}_0$ (modifizierte Subtraktion)
- Wenn P_1 und P_2 LOOP-Programme sind, dann auch $P_1;P_2$ (Komposition)
- Wenn P ein LOOP-Programm ist, dann auch LOOP x_i DO P END für jedes $i \in \mathbb{N}_0$ (Schleife)

(Definition, wie diese Programme die Variableninhalte beeinflussen, wird hier nicht wiederholt.)

LOOP-Programme: berechnete Funktion

Definition (von einem LOOP-Programm berechnete Funktion)

Ein LOOP-Programm berechnet folgende Funktion $f : \mathbb{N}_0^k \to \mathbb{N}_0$ über natürlichen Zahlen:

$$f(a_1, \ldots, a_k) = a_0,$$

wenn bei Ausführung des Programms mit Anfangswerten $x_1 = a_1, \ldots, x_k = a_k$ (und $x_i = 0$ für alle anderen Variablen) am Ende x_0 den Wert a_0 enthält.

- f ist immer eine totale Funktion. Warum?

LOOP-Programme: LOOP-berechenbar

Definition (LOOP-berechenbar)

Eine Funktion $f : \mathbb{N}_0^k \to \mathbb{N}_0$ heisst LOOP-berechenbar, wenn ein LOOP-Programm existiert, das sie berechnet.

Anmerkung: Nicht-totale Funktionen sind nie LOOP-berechenbar.

WHILE-Programme: Definition

Definition (WHILE-Programm)

WHILE-Programme sind definiert durch endliche Anwendung folgender Regeln:

- $x_i := x_i + c$ ist ein WHILE-Programm für jedes $i, c \in \mathbb{N}_0$ (Addition)
- $x_i := x_i - c$ ist ein WHILE-Programm für jedes $i, c \in \mathbb{N}_0$ (modifizierte Subtraktion)
- Wenn P_1 und P_2 WHILE-Programme sind, dann auch $P_1;P_2$ (Komposition)
- Wenn P ein WHILE-Programm ist, dann auch WHILE $x_i \neq 0$ DO P END für jedes $i \in \mathbb{N}_0$ (Schleife)

(Definition, wie diese Programme die Variableninhalte beeinflussen, wird hier nicht wiederholt.)

WHILE-Programme: berechnete Funktion

Definition (von einem WHILE-Programm berechnete Funktion)

Ein WHILE-Programm berechnet folgende (partielle) Funktion $f : \mathbb{N}_0^k \to \mathbb{N}_0$ über natürlichen Zahlen:

$$f(a_1, \ldots, a_k) = a_0,$$

wenn bei Ausführung des Programms mit Anfangswerten $x_1 = a_1, \ldots, x_k = a_k$ (und $x_i = 0$ für alle anderen Variablen) am Ende x_0 den Wert a_0 enthält.

WHILE-Programme: WHILE-berechenbar

Definition (WHILE-berechenbar)

Eine (partielle) Funktion $f : \mathbb{N}_0^k \to \mathbb{N}_0$ heisst WHILE-berechenbar, wenn ein WHILE-Programm existiert, das sie berechnet.

Anmerkung: Der Definitionsbereich von f muss genau den Eingaben entsprechen, auf denen das WHILE-Programm anhält (= nicht in eine Endlosschleife gerät).

GOTO-Programme: Definition

Definition (GOTO-Programm)

Ein GOTO-Programm ist gegeben durch eine endliche Folge $M_1 : A_1, M_2 : A_2, \ldots, M_n : A_n$ von Marken und Anweisungen.

Anweisungen sind von folgender Form:

- $x_i := x_i + c$ für jedes $i, c \in \mathbb{N}_0$ (wie gehabt)
- $x_i := x_i - c$ für jedes $i, c \in \mathbb{N}_0$ (wie gehabt)
- HALT (Programmende)
- GOTO M_j für $1 \leq j \leq n$ (Sprung)
- IF $x_i = c$ THEN GOTO M_j für $i, c \in \mathbb{N}_0$, $1 \leq j \leq n$ (bedingter Sprung)

(Definition, wie diese Programme die Variableninhalte beeinflussen, wird hier nicht wiederholt.)

GOTO-Programme: berechnete Funktion

Definition (von einem GOTO-Programm berechnete Funktion)

Ein GOTO-Programm berechnet folgende (partielle) Funktion $f : \mathbb{N}_0^k \to \mathbb{N}_0$ über natürlichen Zahlen:

$$f(a_1, \ldots, a_k) = a_0,$$

wenn bei Ausführung des Programms mit Anfangswerten $x_1 = a_1, \ldots, x_k = a_k$ (und $x_i = 0$ für alle anderen Variablen) am Ende x_0 den Wert a_0 enthält.

GOTO-Programme: GOTO-berechenbar

Definition (GOTO-berechenbar)

Eine (partielle) Funktion $f : \mathbb{N}_0^k \to \mathbb{N}_0$ heisst GOTO-berechenbar, wenn ein GOTO-Programm existiert, das sie berechnet.

Anmerkung: Der Definitionsbereich von f muss genau den Eingaben entsprechen, auf denen das GOTO-Programm anhält (= nicht in eine Endlosschleife gerät).

4 primitiv und μ-rekursive Funktionen

Formale Berechnungsmodelle: primitive und μ-Rekursion

Formale Berechnungsmodelle

- Turingmaschinen
- LOOP-, WHILE-, GOTO-Programme
- primitiv rekursive Funktionen, μ-rekursive Funktionen

Primitiv rekursive Funktionen

Definition (primitiv rekursiv)

Die Menge der primitiv rekursiven Funktionen (PRF) besteht aus allen Funktionen $f : \mathbb{N}_0^k \to \mathbb{N}_0$, die durch endliche Anwendung folgender Regeln gebildet werden können.

Die folgenden Basisfunktionen sind PRF:

- konstante Funktionen: $f(x_1, \ldots, x_k) = c$ mit $c \in \mathbb{N}_0$
- Projektionen: $f(x_1, \ldots, x_k) = x_i$ mit $1 \leq i \leq k$
- Nachfolgerfunktion: $f(x) = x + 1$

Komposition (Einsetzung von PRFs in PRFs) ergibt PRFs.

...

Primitiv rekursive Funktionen

Definition (primitiv rekursiv)

Primitive Rekursion: wenn g und h PRFs sind, dann ist die durch folgendes Gleichungssystem gegebene Funktion f PRF:

$$f(0, x_2, \ldots, x_k) = g(x_2, \ldots, x_k)$$
$$f(n+1, x_2, \ldots, x_k) = h(n, f(n, x_2, \ldots, x_k), x_2, \ldots, x_k)$$

Anmerkung: primitiv rekursive Funktionen sind immer total

μ-Operator

Definition (μ-Operator)

Der μ-Operator definiert zu einer $k+1$-stelligen Funktion f eine k-stellige Funktion μf wie folgt:

$$(\mu f)(x_1, \ldots, x_k) = \min\{n \in \mathbb{N}_0 \mid f(n, x_1, \ldots, x_k) = 0 \text{ und für alle } m < n \text{ ist } f(m, x_1, \ldots, x_k) \text{ definiert}\}$$

Falls die zu minimierende Menge leer ist, ist $(\mu f)(x_1, \ldots, x_k)$ undefiniert.

μ-rekursive Funktionen

Definition (μ-rekursiv)

Die Menge der μ-rekursiven Funktionen (μRF) besteht aus allen (partiellen) Funktionen $f : \mathbb{N}_0^k \to \mathbb{N}_0$, die durch endliche Anwendung der folgenden Regeln gebildet werden können:

- Basis-Funktionen (wie bei PRF)
- Einsetzung
- primitive Rekursion
- μ-Operator (wenn f μRF, dann ist auch μf μRF)

5 Zusammenhänge

Äquivalente Berechnungsmodelle (1)

Satz (Äquivalenz von Berechnungsmodellen)

Sei $f : \mathbb{N}_0^k \to \mathbb{N}_0$ eine partielle Funktion über den natürlichen Zahlen. Dann gilt:

- *f ist Turing-berechenbar*
- *genau dann, wenn f ist WHILE-berechenbar*
- *genau dann, wenn f ist GOTO-berechenbar*
- *genau dann, wenn f ist μ-rekursiv*

Anmerkung: Ferner ist „Turing-berechenbar durch normale 1-Band-Maschine“ äquivalent zu „Turing-berechenbar durch Mehr-Band-Maschine“

Äquivalente Berechnungsmodelle (2)

Satz (Äquivalenz von Berechnungsmodellen)

Sei $f : \mathbb{N}_0^k \to \mathbb{N}_0$ eine totale Funktion über den natürlichen Zahlen. Dann gilt:

- *f ist LOOP-berechenbar*
- *genau dann, wenn f ist primitiv rekursiv*

Unterschiede in Berechnungsmodellen

Satz (Unterschiede in Berechnungsmodellen)

1. *Jede LOOP-berechenbare Funktion ist WHILE-berechenbar.*
2. *Es gibt WHILE-berechenbare Funktionen, die nicht LOOP-berechenbar sind.*
3. *Dies gilt auch, wenn wir uns auf totale Funktionen beschränken.*

Frage: Wie kann man Teil 2+3 beweisen?

6 Church-Turing-These

Church-Turing-These

Church-Turing-These

Die Klasse der Turing-berechenbaren Funktionen ist genau die Klasse der intuitiv berechenbaren Funktionen.

- kann man nicht beweisen (Warum?)
- Wieso sind die Ergebnisse des vorigen Abschnitts hierfür relevant?