Technische Universität München
Prof. J. Esparza / Dr. M. Luttenberger, S. Sickert

Sommer 2016
21. Juni 2016

Einführung in die theoretische Informatik – Aufgabenblatt 10

Beachten Sie: Soweit nicht explizit angegeben, sind Ergebnisse stets zu begründen!

Hausaufgaben: Abgabe bis zum 29.06.2016 (Mittwoch) um 12:00

Hinweis: Bei allen Aufgaben zu Turingmaschinen wird erwartet, dass zunächst umgangssprachlich das Verhalten der TM beschrieben wird. Desweiteren müssen die Transitionen einer TM stets graphisch entsprechend den Vorlesungsfolien dargestellt werden.

Aufgabe 10.1 — 5P

Geben Sie eine nichtdeterministische 1-Band-TM mit $\Sigma = \{a, b\}$ an, die nichtdeterministisch die Eingabe permutiert, genauer: Ist w die Eingabe, dann terminiert die TM auf jedem Rechenpfad, und zu jeder Permutation w' von w gibt es mindestens eine Rechnung der TM, an deren Ende der Bandinhalt gerade w' ist.

Beispiel: Auf Eingabe $w = aabb$ soll die TM für jede der möglichen Permutation $w' \in \{abab, abba, baba, bbaa, baab, aabb\}$ jeweils mindestens eine Berechnung besitzen, an deren Ende genau w' auf dem Band steht. Beachten Sie, dass die TM stets terminieren muss.

Aufgabe 10.2 — 5P

Sei $\mathsf{A} = \{a, b\}$, $\overline{\mathsf{A}} = \{\overline{a}, \overline{b}\}$ und $\Sigma = \mathsf{A} \cup \overline{\mathsf{A}}$. Ein Wort $ux\overline{x}v$ mit $u, v \in \Sigma^*$ und $x \in \mathsf{A}$ kann zu uv reduziert werden, z.B. kann $ab a\overline{a}\overline{b}a$ zu $ab\overline{b}a$ und weiter zu aa reduziert werden. $\overline{a}a$ kann jedoch nicht zu ε reduziert werden.

Geben Sie eine 1-Band-TM an, die eine Eingabe w maximal reduziert, d.h. nach Terminierung soll auf dem Band ein Wort w' stehen, so dass w' aus w durch eine endliche Anzahl von Reduktionen hervorgeht, w' selbst jedoch nicht weiter reduzierbar ist.

Hinweis: Gehen Sie bei der Aufgabe modular vor. Geben Sie verschiedene TMs für Teilfunktionen an und wie diese kombiniert werden.

Aufgabe 10.3 — 5P

Sei $f\colon \{0,1\}^* \to \{0,1\}^*$ eine Funktion. Der Graph von f ist dann $G_f = \{x\#f(x)\colon x \in \{0,1\}^*\}$.

Zeigen Sie: f ist Turing-berechenbar genau dann, wenn G_f von einer Turing-Maschine akzeptiert wird.

Hinweis: Es reicht jeweils unter Verwendung der Resultate aus der Vorlesung zu beschreiben, wie man unter Verwendung einer Turing-Maschine für das eine Problem eine Turing-Maschine für das andere Problem konstruieren kann.

Aufgabe 10.4 — 5P

Eine 1-Band-k-Kopf-TM unterscheidet sich von einer gewöhnlichen 1-Band-TM darin, dass es $k > 0$ Köpfe gibt, die unabhängig von einander bewegt werden können. Die Köpfe sind dabei von 1 bis k durchnummeriert. In jedem Schritt lesen alle Köpfe den Inhalt der jeweiligen Zelle, auf der sie gerade stehen. Das gelesene k-Tupel zusammen mit dem aktuellen Zustand der TM bestimmt dann die Transition wie gewöhnlich. Hierbei kann jeder Kopf den Inhalt der Zelle, die er gelesen hat, überschreiben. Die Köpfe schreiben dabei in der Reihenfolge, in der sie durchnummeriert sind, d.h. zuerst schreibt Kopf 1, dann Kopf 2, usw. ganz zu Schluss schreibt Kopf k. Effektiv bedeutet dies, schreiben mehrere Köpfe in dieselbe Bandzelle, so bestimmt der Kopf mit der höchsten Nummer, welches Zeichen tatsächlich geschrieben wird.

Zeigen Sie: 1-Band-k-Kopf-TM sind äquivalent zu 1-Band-TM. Skizzieren Sie hierfür – u.U. in mehreren Schritten und unter Verwendung der Resultate aus der Vorlesung – wie sich eine gegebene 1-Band-k-Kopf-TM in eine äquivalente 1-Band-TM übersetzen lässt. Äquivalent bedeutet hier, dass beide TM dieselbe Sprache akzeptieren sollen.

Tutoraufgaben: Besprechung in KW26

Aufgabe 10.1

Geben Sie für jede der folgenden Funktionen sowohl ein WHILE-Programm als auch ein GOTO-Programm an, das die jeweilige Funktion implementiert:

(a) $f(x_1, x_2) = x_1 \bmod x_2$

(b) $g(x_1) = \max\{k \in \mathbb{N} \colon \frac{x_1}{2^k} \in \mathbb{N}\}$

(c) $h(x_1, x_2) = (2x_1 + 1) \cdot 2^{x_2}$

Halten Sie sich an die Konventionen aus der Vorlesung: Soll ein WHILE-Programm eine Funktion $F \colon \mathbb{N}^k \to \mathbb{N}$ berechnen, so werden die Variablen $x_1, \ldots, x_k$ entsprechend mit den Eingabewerten $n_1, \ldots, n_k \in \mathbb{N}$ initialisiert, während alle anderen Variablen zu Beginn auf 0 initialisiert werden. Nach Terminierung speichert die Variable x_0 den Funktionswert $F(n_1, \ldots, n_k)$.

Erinnerung: $0 \in \mathbb{N}$

Aufgabe 10.2

Wir erweitern GOTO-Programme um die Möglichkeit, Variablenwerte auf einem globalen Stack zwischenzuspeichern. Mittels der Anweisung **PUSH** x_i wird der aktuelle Wert der Variable x_i auf den Stack gelegt, mittels $x_i :=$ **POP** wird der aktuell oberste Wert vom Stack genommen und in der Variable x_i gespeichert – sollte der Stack aktuell keine Werte enthalten, wird x_i auf 0 gesetzt. Ob der Stack aktuell einen Wert enthält, kann mittels $x_i :=$ **EMPTY** erfragt werden, wobei x_i auf 1 gesetzt wird, falls der Stack leer ist, ansonsten wird x_i auf 0 gesetzt.

Beispiel:

```
M1: IF x1 = 0 GOTO M2;
    IF x1 = 1 GOTO M3;
    x1 := x1 − 1;
    PUSH x1;
    x1 := x1 − 1;
    GOTO M1;
M2: x0 := x0 + 0;
    GOTO M4;
M3: x0 := x0 + 1;
    GOTO M4;
M4: x1 := EMPTY;
    IF x1 = 1 GOTO M5;
    x1 := POP;
    GOTO M1;
M5: HALT
```

(a) Bestimmen Sie die Funktion $\mathbb{N} \to \mathbb{N}$, die von Programm aus dem Beispiel berechnet wird.

(b) Skizzieren Sie, wie sich jedes GOTO-Programm mit Stack in ein GOTO-Programm ohne Stack übersetzen lässt.

Aufgabe 10.3

Wir betrachten vereinfachte JAVA/C/C++-Programme:

Der einzige Datentyp ist $\mathbb{N}$ ("BigInteger"), der allerdings per *call-by-value* analog zu **int** bei Funktionsaufrufen übergeben wird. Innerhalb von Funktionen sind nur die üblichen arithmetischen Operationen x+=1;x+=y;x=y+z;x=2*z;..., if-Anweisungen der Form **if**(x == n){ ... } **else** { ... } (nur Test auf eine Konstante $n \in \mathbb{N}$) und return-Anweisungen der Form **return** x; (nur eine Variable, keine Terme) erlaubt, sonst sind keine weiteren Anweisungen erlaubt, insbesondere kein new, **for**(;;) {}, **while**() {} etc.

Mit *main* sei die Hauptfunktion bezeichnet, welche die von dem Programm berechnete Funktion bestimmt.

Beispiel:

```
ℕ main(ℕ x) {
  ℕ y;
  y = f(x,0);
  return y;
}

ℕ f(ℕ x, ℕ y) {
  if( x == 0 )
    return y;
  if( x == 1 ) {
    y += 1;
    return y;
  }
```

```
  x -= 1;
  y = f(x,y);
  x -= 1;
  y = f(x,y);
  return y;
}
```

Skizzieren Sie, wie sich jedes solche Programm in ein GOTO-Programm mit Stack (siehe TA10.2) übersetzen lässt.