

# Berechenbarkeit und Komplexitätstheorie

Wintersemester 2021/2022

Aufgabenblatt 2

Abgabe: 19. November 2021 um 12 Uhr

## Aufgabe 2.1 (2 + 3 Punkte)

- a) Zeigen Sie, dass man zu jeder (Einband-)Turingmaschine eine neue Turingmaschine konstruieren kann, die zwei Bänder und nur die beiden Zustände  $z_0$  (Startzustand) und  $z_e$  (Endzustand) besitzt.
- b) Zeigen Sie, dass man eine beliebige Mehrband-Turingmaschine mit  $k$  Bändern durch eine Einband-Turingmaschine simulieren kann.

## Lösung

- a) Die Idee bei dieser Aufgabe ist, dass man aus einer TM eine neue konstruiert, die die Zustände der alten TM explizit auf dem zweiten Band speichert. Für jeden Berechnungsschritt der ursprünglichen TM muss die neue TM erst das zweite Band lesen und dann anhand dessen Inhalts auf dem ersten Band agieren. Danach muss sie den neuen Zustand auf das zweite Band schreiben. Etwas formaler ersetzt man jeden Übergang  $\delta(z_i, w) = (z_j, v, B)$  (mit  $B \in \{L, N, R\}$ ) durch den Übergang  $\delta(z_0, w, z_i) = (z_0, v, z_j, B, N)$  für alle  $z_j \notin F$ . Falls  $z_i$  der Startzustand ist, dann muss man noch den Übergang  $\delta(z_0, w, \square) = (z_0, v, z_j, B, N)$  hinzufügen, da das zweite Band zu Beginn leer ist. Falls  $z_j \in F$ , so muss man den Übergang durch  $\delta(z_0, w, z_i) = (z_e, v, \square, B, N)$  ersetzen.
- b) Die grobe Idee ist es, das einzelne Band der Einband-TM (ETM) in mehrere "Spuren" aufzuteilen – wenn bei der Mehrband-TM (MTM)  $a_1, \dots, a_k$  untereinander auf den verschiedenen Bändern stehen, schreibt man bei der ETM  $(a_1, \dots, a_k)$  als einzelnes Symbol auf das Band. Bei dieser einfachen Überlegung ergibt sich allerdings die Problematik, dass die Positionen der einzelnen Köpfe der MTM nicht gespeichert werden. Um dieses Problem zu umgehen kann man  $k$  weitere Spuren einfügen, die sich die Positionen der Köpfe merken. Am einfachsten, in dem die jeweilige Spur bis auf ein einzelnes Symbol  $*$  sonst überall leer ist. Dieses Symbol markiert die Position des jeweiligen Kopfes. Sei also eine MTM mit  $k$  Bändern und einem Arbeitsalphabet  $\Sigma$  gegeben, dann konstruieren wir eine ETM wie folgt:
  - Unterteile das Arbeitsband der ETM in  $2 \cdot k$  Spuren – jedes Symbol auf dem Arbeitsband ist also ein  $2k$ -Tupel.
  - Spur  $2 \cdot i - 1$  enthält den Inhalt von Band  $i$  der MTM.
  - Spur  $2 \cdot i$  speichert die Kopfposition von Kopf  $i$  – die Spur enthält ein einzelnes Symbol  $*$ , welches die Position angibt. Sonst enthält die Spur nur Blanksymbole.
  - Das neue Arbeitsalphabet ist damit  $\Gamma' = (\Gamma \cup \{*\})^{2k} \cup \Gamma$ . Das äußere  $\Gamma$  dient dazu, dass die Ein- und Ausgabe eins-zu-eins übernommen werden kann. (Potentiell ist  $\Gamma'$  eine Obermenge der gültigen Symbole, da auf den ungerade Spuren kein  $*$  stehen kann. Genauer wäre das Alphabet also  $\Gamma' = (\Gamma \times \{*, \square\})^k \cup \Gamma$ )

Die Arbeitsweise der ETM bei Eingabe  $a_1 \cdots a_n$  ist beschreibt sich wie folgt: Ersetze  $a_1 \cdots a_n \vdash^* (a_1, *, \square, *, \dots \square, *) \cdots (a_n, *, \square, *, \dots \square, *)$ . Die Eingabe steht jetzt also auf der ersten Spur und alle anderen (ungeraden) Spuren sind leer. Weiter sind alle Kopfpositionen auf der “ersten” Zelle. Mit dieser Ersetzung stehen sowohl links als auch rechts vom in Spuren unterteilten Abschnitt einzelne Blanksymbole. Die TM kann also den Anfang und das Ende des Bandes erkennen. Laufe (mehrmals) über das Band und behandle jede Spur (also jedes Band der MTM) einzeln, nacheinander. Hierbei bietet es sich an, in den Zuständen zu speichern, welches Band (bzw. welche Spur) gerade betrachtet wird (intuitiv kann man zu jedem Zustand der MTM  $k$  Zustände in der ETM definieren). Sei bspw. der Übergang der MTM  $\delta(z, a_1, \dots, a_k) = (z', b_1, \dots, b_k, L, \dots, L)$ . Die ETM simuliert diesen Übergang wie folgt:

(Beginne in Zustand  $z_{Spur1}$ )

- Suche auf Spur 2 das Kopfsymbol  $*$ .
- Ersetze Symbol  $a_1$  auf Spur 1 zu  $b_1$ .
- Verschiebe das Kopfsymbol  $*$  auf Spur 2 nach links.

(Wechsle in Zustand  $z_{Spur2}$ )

- Wiederhole die drei Schritte für alle anderen Spur-Paare  $(3, 4), \dots, (2k-1, 2k)$ .

(Wechsle in Zustand  $z'_{Spur1}$ )

## Aufgabe 2.2 (6 Punkte)

Die Firma GLOBEX verkauft die Programmiersprache “LOOP++”. Zusätzlich zu den Anweisungen üblicher (also aus der Vorlesung bekannter) LOOP-Programme besitzt LOOP++ noch die folgenden Makros (mit der üblichen Semantik aus bekannten Programmiersprachen):

- **if**  $x_k = 0$  **then**  $A$  **else**  $B$  **end** ( $A$  und  $B$  sind beliebige LOOP++-Programme.)
- $x_i := x_j + x_k$
- $x_i := x_j \cdot x_k$

LOOP++ kostet wesentlich mehr als LOOP. Ist dieser höhere Preis gerechtfertigt, oder kann LOOP++ nicht mehr berechnen als LOOP?

## Lösung

- “if-then-else” lässt sich mit den üblichen LOOP-Befehlen simulieren. Eine Idee ist es, wenn  $x_k \neq 0$  eine Variable, die das Ausführen von  $A$  steuert auf 0 zu setzen und eine Variable, die das Ausführen von  $B$  steuert auf 1 zu setzen.

|                                |                                            |
|--------------------------------|--------------------------------------------|
| 1: $x_a := 1$                  | ▷ Steuert, ob $A$ ausgeführt wird.         |
| 2: $x_b := 0$                  | ▷ Steuert, ob $B$ ausgeführt wird.         |
| 3: <b>loop</b> $x_k$ <b>do</b> | ▷ Wenn $x_k \neq 0$ , ...                  |
| 4: $x_a := 0$                  | ▷ ..., setze $x_a$ auf 0 ...               |
| 5: $x_b := 1$                  | ▷ ...und setze $x_b$ auf 1.                |
| 6: <b>end</b>                  |                                            |
| 7: <b>loop</b> $x_a$ <b>do</b> | ▷ Wenn $x_a \neq 0$ , also $x_k = 0$ , ... |
| 8: $A$                         | ▷ (...führe $A$ aus.                       |
| 9: <b>end</b>                  |                                            |

```

10: loop  $x_b$  do
11:    $B$ 
12: end

```

▷ Wenn  $x_b \neq 0$ , also  $x_k \neq 0$ , ...  
 ▷ ...führe  $B$  aus.

- “ $x_j + x_k$ ” lässt sich auch mit den üblichen LOOP-Befehlen simulieren. Man muss im Prinzip nur  $x_k$ -mal 1 auf  $x_j$  addieren. Formal sind in der VL bei der Addition “ $x_i := x_j + 1$ ”  $i$  und  $j$  verschieden, darum das etwas “umständliche” Programm mit der Extrazuweisung.

```

1:  $x_i := x_j$ 
2: loop  $x_k$  do
3:    $x_t := x_i$ 
4:    $x_i := x_t + 1$ 
5: end

```

▷  $x_k$ -mal addieren

- “ $x_j \cdot x_k$ ” lässt sich auch dann mithilfe der Addition implementieren, indem man  $x_k$ -mal  $x_j$  aufaddiert.

```

1:  $x_i := 0$ 
2: loop  $x_k$  do
3:    $x_i := x_i + x_j$ 
4: end

```

### Aufgabe 2.3 (4 Punkte)

Die *Lucas*-Zahlen sind wie folgt definiert:

$$\begin{aligned}
 L_1 &= 2 \\
 L_2 &= 1 \\
 L_{n+1} &= L_n + L_{n-1} \quad (\text{Für } n \geq 2)
 \end{aligned}$$

Schreiben Sie ein LOOP-Programm, das die  $n$ -te Lucas-Zahl  $L_n$  berechnet. Wie üblich ist der Wert  $n$  zu Beginn des Programms in Variable  $x_1$  gespeichert. (Sie dürfen hierzu das Additions-Makro aus der vorigen Aufgabe nutzen.)

### Lösung

```

1:  $x_0 := 2$ 
2:  $x_2 := 1$ 
3:  $x_3 := 0$ 
4:  $x_4 := x_1 - 1$ 
5: loop  $x_4$  do
6:    $x_3 := x_0 + x_2$ 
7:    $x_0 := x_2$ 
8:    $x_2 := x_3$ 
9: end

```

▷ Entspricht  $L_n$   
 ▷ Entspricht  $L_{n+1}$   
 ▷ Temporäre Variable zum Tauschen  
 ▷ Zählvariable

Statt mit einer Hilfsvariable zu arbeiten kann man auch die drei Zeilen in der LOOP-Schleife ersetzen durch:  $x_2 := x_0 + x_2$  und  $x_0 := x_2 - x_0$ .