Nachbau der Turing-Maschine

Dominikus Herzberg
1. Mai 2024, SoSe 2024

Inhaltsverzeichnis

1 Einleitung

Die Turing-Maschine (TM) stellt einen grundlegenden und sehr einfachen Rechenmechanismus dar; sie ist nach ihrem Erfinder, Alan Turing, benannt. Sie sollen mit dieser Aufgabe die Turing-Maschine als objektorientiertes Programm umsetzen. Dazu bekommen Sie einige Vorgaben gemacht, welche Klassen oder Datenstrukturen zur Modellierung zu verwenden sind. Die Klassen bilden logische Einheiten der Turing-Maschine ab, die sie zu einem funktionierenden Ganzen verknüpfen.

Die Turing-Maschine (Klasse `TM`) besteht aus einem Band (`Tape`), einem Schreib/Lese-Kopf (realisiert über Methoden zu `Tape`) und einer Tabelle (`Table`) aus Transitionen (realisiert durch eine Map). Zudem befindet sich die TM mit ihrem Start in einem gegebenen Zustand (*state*). Einige Aspekte, wie das Band der Turing-Maschine, können als Live-Views dargestellt werden.

Wenn Sie die Turing-Maschine nicht kennen oder ihre Funktionsweise aufarbeiten wollen, hier zwei Vorschläge, um sich in das Thema einzufinden:

- YouTube-Video: Wie funktioniert die Turingmaschine von Alan Turing?
- Podcast-Episode aus Herzbergs Hörsaal: GDI: Die Turing-Maschine, Vorlesung samt Folien

2 Tape: Das Band inkl. Schreib/Lese-Kopf

Das Band besteht aus einer Abfolge von Zellen und ist nach links und rechts (zumindest in der Vorstellung) unendlich lang. Jede Zelle fasst ein Zeichen. In der Grundeinstellung beinhaltet jede Zelle das sogenannte Vorbelegungszeichen, ein Symbol, das man mit der Initialisierung des Bands vorgibt. Das Band hat zudem einen Schreib/Lese-Kopf, kurz S/L-Kopf, der an der aktuellen Zelle positioniert ist. Der S/L-Kopf kann einen Zellenwert auslesen (Methode `read`), ihn beschreiben (Methode `write`) und zur linken bzw. rechten Zelle bewegt werden (Methode `move`).

Es bietet sich an, die Zellenfolge durch eine Liste aus `char`- bzw. `Character`-Werten zu realisieren. Die Anweisung für die `move`-Methode kommt in Form eines Arguments aus der Enumeration `Move` mit den Werten `LEFT` und `RIGHT`.

⚠ Auch wenn das Band von seiner Idee unendlich lang ist, realisieren Sie das Band auf eine Weise, dass es nur so lang wie irgend nötig ist und gegebenenfalls dynamisch verlängert bzw. verkürzt wird.

3 Table: Die Tabelle mit Transitionen

Die Tabelle besteht aus Transitionen, die dazu dienen, das Verhalten einer Turing-Maschine zu beschreiben: Wenn sich die Turing-Maschine in einem bestimmten Zustand befindet und ein gewisses Zeichen vom Band liest (Trigger), dann wird ein definiertes Zeichen auf das Band geschrieben, der S/L-Kopf bewegt und ein neuer Zustand von der Turing-Maschine eingenommen (Aktion). Diese Kombination aus Trigger (Ausgangszustand, gelesenes Zeichen) und Aktionen (zu schreibendes Zeichen, S/L-Bewegung, neuer Zustand) wird als Transition bezeichnet.

Die Transitionen lassen sich in einer Abbildung (*map*) erfassen als `Map<Trigger, Action>` mit einem Konstruktor für den Trigger als `Trigger(String fromState, char read)` und einem für die Aktion als `Action(char write, Move move, String toState)`. Wir gehen damit automatisch davon aus, dass es zu einem Trigger stets nur eine Aktion gibt; eine solche TM nennt man „deterministisch", d.h. eindeutig bestimmt in ihrem Verhalten.

Die Klasse `Table` beheimatet die Map mit den Transitionen aus ihrem jeweiligen Trigger- und Aktionsanteil, sie bietet geeignete Methoden zum Aufbau einer Transitionsmap an und zum Abruf einer Aktion anhand eines Triggers.

ⓘ Wenn Sie es sich einfach machen wollen, probieren Sie für `Trigger` und `Action` die Verwendung der Klassenart namens `record` aus.

4 TM: Die Klasse, die alles zusammenführt

Die Klasse `TM` bringt das Band mit dem S/L-Kopf und die Tabelle mit den Transitionen zusammen und koordiniert nach ihrer Initialisierung die schrittweise Ausführung der Turing-Maschine (Methode `step`). Dazu muss ein Startzustand gegeben sein. Die Maschine beendet ihre Arbeit, wenn der Haltezustand erreicht ist, der per Definition "`HALT`" heißt. Die Methode run dient dafür, die Maschine bis zum Haltezustand durchlaufen zu lassen.

5 Die Live-View zur Turing-Maschine

Die Tabelle mit den Transitionen kann als Markdown-Tabelle in einer *Live View* zur Ansicht gebracht werden. Mit jedem `step` wird eine Turtle-Grafik aktualisiert, die die aktuelle Zelle, auf die der S/L-Kopf zeigt, ausgibt; außerdem werden die fünf Nachbarzellen zur linken wie zur rechten Seite angezeigt. Zusätzlich wird das Band, wie in den Beispielen zu sehen, als Markdown-Text in der View ausgegeben, eine Durchnummerierung der Schritte ist jedoch nicht erforderlich.

6 Abgabe

Es sind zwei Dateien in Moodle einzureichen:

- Ihre Datei `TM.java` soll dieses Mal in der *Live View* nicht den Code dokumentieren, sondern die korrekte Abarbeitung der beiden mitgelieferten Beispiele demonstrieren. Zeigen Sie beim ersten Beispiel (Dekrementierung einer Binärzahl) auch die Tabelle als View und erläutern Sie mit den ersten drei Einzelschritten, dass die Turing-Maschine tut, was sie tun soll.
- Die erzeugte *View* ist, wie gehabt, als `TM.pdf` abzugeben.

⚠ Bitte achten Sie beim PDF darauf, das alles zu sehen ist. Gegebenenfalls geben Sie die HTML-Ansicht nicht hochkant sondern im Querformat aus und verkleinern zuvor die Darstellungsgröße im Browser.

Sie sollen mit der Aufgabe zeigen, dass Sie objekorientierten Programmcode schreiben können, der klar strukturiert und aufgebaut ist.

7 Umzusetzende Beispiele

7.1 Dekrementierung einer Binärzahl

Gegeben sei das nachstehende Programm für die Turing-Maschine.

fromState	read	write	move	toState
S	#	#	L	S
S	1	0	R	R
S	0	1	L	L
R	0	0	R	R
R	1	1	R	R
R	#	#	L	W
W	1	1	R	HALT
W	0	0	R	HALT
W	#	#	R	HALT
L	0	1	L	L
L	1	0	R	R
L	#	#	R	R

Das Band ist wie folgt initialisiert: `# 1 1 0 0 0{#}`

(Hinweis: Die Position des Schreib/Lese-Kopfes ist durch `{ }` markiert.)

Das Vorbelegungszeichen auf dem Band ist `#` und die Maschine befindet sich im Zustand `S`.

▼ Lösung

```
 0:  # 1 1 0 0 0{#} -- S
 1:  # 1 1 0 0{0}#  -- S
 2:  # 1 1 0{0}1 #  -- L
 3:  # 1 1{0}1 1 #  -- L
 4:  # 1{1}1 1 1 #  -- L
 5:  # 1 0{1}1 1 #  -- R
 6:  # 1 0 1{1}1 #  -- R
 7:  # 1 0 1 1{1}#  -- R
 8:  # 1 0 1 1 1{#} -- R
 9:  # 1 0 1 1{1}#  -- W
10:  # 1 0 1 1 1{#} -- HALT
```

7.2 Einsen nach rechts schieben

Gegeben sei das nachstehende Programm für die Turing-Maschine.

fromState	read	write	move	toState
S	1	1	L	S
S	S	S	R	HALT
S	0	0	L	0
0	0	0	L	0
0	1	0	R	1
0	S	S	R	HALT
1	0	0	R	1
1	1	1	L	D
1	S	S	L	D
D	0	1	L	S

Das Band ist wie folgt initialisiert: `S 0 1 0 1 0{1}S`

(Hinweis: Die Position des Schreib/Lese-Kopfes ist durch `{ }` markiert.)

Das Vorbelegungszeichen auf dem Band ist `0` und die Maschine befindet sich im Zustand `S`.

▼

```
 0:  S 0 1 0 1 0{1}S  -- S
 1:  S 0 1 0 1{0}1 S  -- S
 2:  S 0 1 0{1}0 1 S  -- 0
 3:  S 0 1 0 0{0}1 S  -- 1
 4:  S 0 1 0 0 0{1}S  -- 1
 5:  S 0 1 0 0{0}1 S  -- D
 6:  S 0 1 0{0}1 1 S  -- S
 7:  S 0 1{0}0 1 1 S  -- 0
 8:  S 0{1}0 0 1 1 S  -- 0
 9:  S 0 0{0}0 1 1 S  -- 1
10:  S 0 0 0{0}1 1 S  -- 1
11:  S 0 0 0 0{1}1 S  -- 1
12:  S 0 0 0{0}1 1 S  -- D
13:  S 0 0{0}1 1 1 S  -- S
14:  S 0{0}0 1 1 1 S  -- 0
15:  S{0}0 0 1 1 1 S  -- 0
16:  {S}0 0 0 1 1 1 S  -- 0
17:  S{0}0 0 1 1 1 S  -- HALT
```