Turing-Maschinen

Nachdem wir endliche Automaten und (die mächtigeren) Kellerautomaten kennengelernt haben, werden wir nun ein letztes, noch mächtigeres Automatenmodell kennenlernen: Die *Turing-Maschine* (TM). Während die bisherigen Automaten Sprachen akzeptiert haben, kann die Turing-Maschine neben dieser Akzeptanz von Sprachen auch *Funktionen berechnen*. Darauf kommen wir gleich zu sprechen. Zunächst wollen wir informal beschreiben, wie eine Turing-Maschine aufgebaut ist und wie sie arbeitet.

Eine Turing-Maschine ist erstmal ähnlich aufgebaut wie ein endlicher Automat. Sie hat eine endliche Zustandsmenge, ein Eingabealphabet und ein Eingabeband, auf dem zu Beginn die Eingabe steht. Anders als der endliche Automat kann sie aber auf dem Eingabeband nicht nur lesen, sondern auch schreiben. Sie hat dazu einen Lese-/Schreibkopf (LSK), der zu Beginn auf dem ersten Eingabesymbol steht. Abhängig von dem Zustand, in dem die TM ist, und dem Symbol, dass sie gerade liest, kann sie nun den Zustand wechseln, das gerade gelesene Symbol durch ein anderes ersetzen und dann den LSK nach rechts oder links bewegen oder an der Position verharren.

Genauer gibt es neben dem Eingabealphabet Σ noch das Bandalphabet $\Gamma \supset \Sigma$. Dieses enthält mindestens noch das Symbol # zusätzlich, das zur Kennzeichnung des leeren Feldes benutzt wird. Eine Konfiguration der TM ist dann ein Element aus $\Gamma^* \cdot Z \cdot \Gamma^*$, wobei die Position des Zustandes die Position des LSK angibt. Ist man z.B. in der Konfiguration $abbz_1ca$, so ist der LSK gerade über dem Feld auf dem Eingabeband, in dem c notiert ist. Links vom LSK sind noch die Buchstaben abb und rechts vom LSK noch der Buchstabe a auf dem Band. Ferner sind links und rechts davon noch unendlich viele # bei einem beidseitig unendlichen Eingabeband, bzw. rechts noch unendlich viele # bei einem einseitig unendlichem Eingabeband. Links sind dann im letzten Fall noch gerade so viele # wie das Band es zulässt.

Die vollständige Definition einer Turing-Maschine sieht wie folgt aus:

Definition 1. *Eine* deterministische Turing-Maschine *(kurz DTM) ist ein* 6-*Tupel* $A = (Z, \Sigma, \Gamma, \delta, z_0, Z_{end})$ *mit*

- *Der endlichen Menge von* Zuständen Z.
- *Dem endlichen Alphabet* Σ *von* Eingabesymbolen.
- *Dem endlichen Alphabet* Γ *von* Bandsymbolen*, wobei* $\Gamma \supsetneq \Sigma$ *und* $\Gamma \cap Z = \emptyset$ *gilt.*

- *Der (partiellen)* Überführungsfunktion $\delta : (Z \times \Gamma) \to (\Gamma \times \{L, R, H\} \times Z)$.
- *Dem* Startzustand $z_0 \in Z$.
- *Der Menge der* Endzustände $Z_{end} \subseteq Z$.
- *Dem* Symbol für das leere Feld $\# \in \Gamma \setminus \Sigma$.

Statt der Überführungsfunktion δ kann auch mit einer Relation $K \subseteq Z \times \Gamma \times \Gamma \times \{L, R, H\} \times Z$ gearbeitet werden. Dabei gibt es dann im deterministischem Fall aber zu jedem Pärchen $(z, x) \in Z \times \Gamma$ nur maximal ein Tripel (x', B, z') derart, dass $(z, x, x', B, z') \in K$ gilt.

Mit $\delta(z, x) = (x', B, z')$ ist gemeint, dass, wenn die TM im Zustand z ist und der LSK gerade über einem Feld mit dem Symbol x ist, nun das x durch x' überschrieben wird, dann der LSK nach $B \in \{L, R, H\}$ bewegt wird (L heißt nach links bewegen, R nach rechts und H den LSK an der Stelle zu halten) und in den Zustand z' gewechselt wird.

Man kann eine TM wieder mit einem Zustandsdiagramm angeben. Abbildung 1 zeigt eine TM M, die die Sprache $L = \{w \in \{a, b\}^* \mid w \text{ endet auf } b\}$ akzeptiert. Wir wollen die Arbeit von M etwas genauer beschreiben: Sei $w = abbaab$ das Eingabewort. Die TM startet dann in der Konfiguration $z_0abbaab$. Sie kann nun mittels der Kante (z_0, a, x, R, z_0) das a lesen, durch X ersetzen, im Zustand z_0 bleiben und den LSK nach rechts bewegen. Sie ist dann in der Konfiguration Xz_0bbaab. Man beachte, wie durch die Position des Zustandes in der Konfiguration angezeigt wird, wo der LSK sich befindet (nämlich jetzt über dem b). Nun kann zweimal die Kante (z_0, b, b, R, z_0) benutzt werden, was die TM in die Konfiguration $Xbbz_0aab$ bringt. Arbeitet man weiter mit den Kanten an dem Zustand z_0, so gelangt man in die Konfiguration $XbbXXbz_0\#$. (Das $\#$ wird nun explizit erwähnt, damit klar ist, welches Symbol der LSK gerade liest.)

Nun ist mit der Kante von z_0 nach z_1 ein Zustandswechsel möglich. Hierbei wird der LSK wieder ein Feld nach links bewegt und wir gelangen in die Konfiguration $XbbXXz_1b$. Zuletzt wird nun mit der Kante $z_1, b, b, R, z_2)$ geprüft, ob das letzte Symbol (das Symbol über dem der LSK gerade steht) ein b ist und falls ja, wird in z_2 gewechselt, wo das Eingabewort akzeptiert wird. Die letzte Konfiguration ist dann $XbbXXbz_2\#$.

Zusammengefasst arbeitet die TM also so: Bei Eingabe eines Wortes $w \in \{a, b\}^*$ wird in z_0 über das ganze Wort herübergelesen, wobei jedes a zu einem X gemacht wird und die b unangetastet bleiben. Ist das Ende des Wortes erreicht

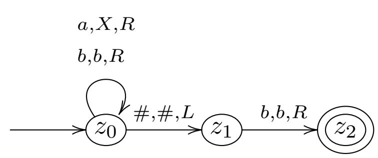

Abbildung 1: Eine TM M für die Sprache $L = \{w \in \{a, b\}^* \mid w \text{ endet auf } b\}$.

(was man dadurch merkt, dass man erstmals zu einem # kommt), so wird der LSK nach links bewegt und befindet sich nun über dem letzten Symbol des Eingabewortes. Ist dies ein b, was wie oben schon beschreiben durch die Kante von z_1 nach z_2 getestet wird, so akzeptiert die TM.

Dass die as in X umgewandelt werden, ist eigentlich unnötig und soll hier nur illustrieren, dass die TM die Symbole auf dem Eingabeband manipulieren kann. Das X ist insb. kein Symbol des Eingabealphabets, sondern nur des Bandalphabets, was man zu Anfang genauer definieren sollte. (Hier kann man implizit erahnen, dass jedes Eingabewort nur aus as und bs bestehen soll, da die Menge L eine Teilmenge von $\{a,b\}^*$ ist.)

Zusammengefasst:

- Die TM hat endlich viele Zustände.
- Sie hat ein Eingabealphabet, aus dessen Buchstaben die Eingabewörter aufgebaut sind.
- Sie hat ein (größeres) Bandalphabet, mit weiteren Symbolen, die benutzt werden können. Dieses enthält mindestens noch das spezielle Symbol #.
- Das Eingabeband ist nach rechts hin unendlich. Zu Anfang steht das Eingabewort ganz links auf diesem Band. Danach folgen unendlich viele #.
- Die TM liest nicht nur das Eingabewort, sondern kann auch auf dem Eingabeband mit dem LSK nach links und rechts wandern und die Symbole manipulieren. Anders als der PDA kommt sie dabei an jedes Symbol heran und nicht nur an ein spezielles (das oberste des Kellers beim PDA)!

Zudem ist noch wichtig, was bisher nicht erwähnt wurde, dass die TM ein Eingabewort bereits dann akzeptiert, wenn sie in einen Endzustand gelangt. Anders als alle bisher betrachteten Automatenmodell muss sie sich das Eingabewort nicht bis zum Ende ansehen! Die TM in Abbildung 2 akzeptiert daher alle Wörter, die mit b beginnen, da sie bei einem solchen Wort von der Konfiguration z_0bv in die Konfiguration bz_1v wechselt und dann in einem Endzustand ist, also $w = bv$ akzeptiert - unabhängig vom Rest des Wortes v.

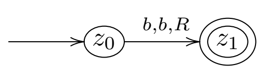

Abbildung 2: Eine TM M für die Sprache $L = \{w \in \{a,b\}^* \mid w \text{ beginnt mit } b\}$.

Aufgrund ihrer Fähigkeit, Symbole auf dem Band zu manipulieren, kann die Turing-Maschine genutzt werden, um Funktionen zu berechnen. Bspw. kann man sich eine Funktion vorstellen, die ein Wort nimmt und an dieses ein b heranhängt. Formal: $f : \{a,b\}^* \to \{a,b\}^*$ mit $f(w) = wb$. Eine TM für diese Funktion würde ähnlich wie die TM in Abbildung 1 über das Eingabewort herüberlesen (allerdings dabei nicht a in X umwandeln, sondern die as wie auch die bs unberührt lassen) und das erste Auftreten eines # durch b ersetzen. Dann steht wb auf dem Band. Formal wird dann noch verlangt, dass der LSK wieder an den Anfang des Wortes bewegt wird. Man sagt dann, dass eine TM eine Funktion f berechnet, wenn sie gestartet in der Konfiguration z_0w in der Konfiguration $z_ef(w)$ endet, wobei z_e ein Endzustand ist.

Neben solchen sogenannten Wortfunktionen, kann die TM auch benutzt werden, um andere Funktionen zu berechnen. Z.B. kann man, um die Funktion $f(x, y) = x + y$ zu berechnen (wobei x und y natürliche Zahlen seien), x und y binär durch 0en und 1en kodieren und mit diesen arbeiten. Am Ende steht dann die Binärkodierung von $x + y$ auf dem Band.

Man vermutet, dass die Turing-Maschine so mächtig ist, dass sie alles berechnen kann, was man intuitiv berechnen kann. Diese Aussage ist nicht zu beweisen, da der Begriff des "intuitiv Berechenbaren" nicht definierbar ist. Gemeint ist, dass jede Funktion, die von einem Menschen berechnet werden kann, auch von ihr berechnet werden kann. Bisher wurde hierzu kein Gegenbeispiel gefunden und man nimmt an, dass es auch keines gibt. Die TM eignet sich daher um ganz grundsätzlich über algorithmische Fragestellung wie 'Was kann berechnet werden?' oder 'Wie teuer ist die Berechnung?' nachzudenken.

Nichtdeterministische Turing-Maschinen

So wie für die anderen Automatenmodelle auch, kann auch für die Turing-Maschine ein nichtdeterministisches Modell definiert werden. Wie bei den DFAs und NFAs kann gezeigt werden, dass dieses Modell äquivalent zu dem deterministischem ist. Allerdings sind nichtdeterministische Modelle vermutlich erheblich effizienter, was uns in den letzten Wochen der Vorlesung im Rahmen der Komplexitätstheorie noch beschäftigen wird und letztendlich zur $P \stackrel{?}{=} NP$ Frage führen wird.

Aufzählbarkeit und Entscheidbarkeit

Man kann sich leicht ein Beispiel überlegen, bei dem die TM in eine Endlosschleife gerät. Zum Beispiel könnte sie bei Lesen eines as den LSK nach rechts bewegt und bei Lesen eines bs den LSK nach links. Beim Eingabewort ab würde sie dann den LSK immer hin und her bewegen. Da dies auch in anderen Kontexten relevant ist, wurden zwei Sprachfamilien für TMs definiert. Die *aufzählbaren* und die *entscheidbaren* Sprachen. Bei den entscheidbaren Sprachen wird verlangt, dass die TM bei jeder Eingabe letztendlich anhält und mit 'Ja' oder 'Nein' antwortet (was gleichbedeutend damit ist, in einem Endzustand zu landen oder nicht). Bei den aufzählbaren Sprachen wird nur verlangt, dass sie bei Worten der Sprache anhält (und akzeptiert). Für die anderen Worte wird nichts verlangt. Arbeitet die TM also länger, weiß man dann nicht, ob sie das Wort noch akzeptieren wird oder ob das nie geschehen wird.

Algorithmisch wichtig sind natürlich die entscheidbaren Sprachen. Umso erstaunlicher ist es, dass es ganz grundlegende Probleme gibt, die nicht entschieden werden können. Da die TM als Modell eines jeden Computers oder einer jeden

Programmiersprache dienen kann, bedeutet dies auch, dass es ganz grundlegende Probleme gibt, die nicht von uns gelöst werden können. Ein Problem dieser Art ist das *Halteproblem*. Bei diesem Problem erhält man eine TM M und ein Wort w und soll entscheiden, ob M auf w anhält. Eine äquivalente Frage ist, ob ein Computerprogramm terminiert oder nicht. Eine wichtiges Ergebnis ist nun, dass dies nicht berechnet werden kann. Während es also einen Compiler gibt, der uns auf Syntaxfehler aufmerksam machen kann, gibt es keine Möglichkeit ein Computerprogramm zu schreiben, dass uns auf diese ganz grundlegende semantische Frage eine Antwort gibt.

In der Vorlesung werden wir die deterministische und die nichtdeterministische Turing-Maschine einführen und zeigen, dass diese äquivalent sind. Wir werden uns ansehen, wie eine TM Sprachen akzeptieren und wie sie Funktionen berechnen kann. Wir werden die Begriffe aufzählbar und entscheidbar definieren und mit dem Halteproblem eine wichtige Fragestellung betrachten, von der wir dann aber einsehen müssen, dass wir sie leider gar nicht algorithmisch lösen können.

Diesmal entfällt die mathematische Seite. Wir kommen gleich zum Selbsttest und auf der nächsten Seite dann zu dessen Lösung.

Selbsttest (Lösung auf Seite 6)

1. Wann akzeptiert eine Turing-Maschine M ein Eingabewort w?
2. Was passiert, wenn eine Turing-Maschine eine Funktion berechnet?
3. Können Band- und Eingabealphabet gleich sein?
4. Wie verhalten sich die deterministischen und nichtdeterministischen Varianten endlicher Automaten, Kellerautomaten und Turing-Maschinen zueinander? Welche sind mächtiger?
5. Was ist der Unterschied zwischen entscheidbar und aufzählbar?
6. Haben Sie gerade Spaß?

Lösungen zum Selbsttest auf Seite 5

1. **F:** Wann akzeptiert eine Turing-Maschine M ein Eingabewort w?

   A: Wenn sie vom Startzustand aus und mit w auf dem Eingabeband so rechnen kann, dass sie in einen Endzustand gelangt. Ob sie dabei w ganz angesehen hat, ist, anders als bei DFAs und PDAs, egal. Ebenso was sie während der Rechnung auf das Band geschrieben hat. Wichtig ist also nur, *dass* die TM in einen Endzustand gelangt.

2. **F:** Was passiert, wenn eine Turing-Maschine eine Funktion berechnet?

   A: Ist f eine Funktion, die von einer Turing-Maschine M berechnet wird, so startet M in der Konfiguration z_0w (wobei z_0 der Startzustand ist) und endet in $z_ef(w)$, wobei z_e ein Endzustand ist. Man beachte, dass der Lese-/Schreibkopf wieder ganz links von dem Wort $f(w)$ ist. Zudem sind wir hier von einer Funktion f mit einem Argument ausgegangen.

3. **F:** Können Band- und Eingabealphabet gleich sein?

   A: Nein! Das Bandalphabet enthält immer mindestens ein Symbol, das nicht im Eingabealphabet ist, nämlich das Symbol # zur Darstellung des leeren Feldes.

4. **F:** Wie verhalten sich die deterministischen und nichtdeterministischen Varianten endlicher Automaten, Kellerautomaten und Turing-Maschinen zueinander? Welche sind mächtiger?

   A: Bei endlichen Automaten sind deterministische (DFA) und nichtdeterministische (NFA) Modelle äquivalent (Potenzautomatenkonstruktion). Bei Kellerautomaten ist die nichtdeterministische Variante echt mächtiger als die deterministische. Und bei Turing-Maschinen sind sie wieder äquivalent. Zudem sind endliche Automaten echt schwächer als Kellerautomaten, die wiederum echt schwächer als Turing-Maschinen sind. Komplexitätstheoretisch sind die nichtdeterministischen Modelle aber oft besser als die deterministischen (bzgl. der Laufzeit bei der Akzeptierung eines Wortes). Hierzu kommen wir später noch.

5. **F:** Was ist der Unterschied zwischen entscheidbar und aufzählbar?

   A: Eine Menge $L \subseteq \Sigma^*$ ist entscheidbar genau dann, wenn es eine Turing-Maschine gibt, die bei jeder Eingabe eines Wortes $w \in \Sigma^*$ nach endlich vielen Schritten anhält und korrekt Ja oder Nein ausgibt (Ja für $w \in L$, Nein im anderen Fall). Aufzählbar ist eine Menge, wenn die Turing-Maschine nur bei Ja-Instanzen (bei Worten, die in L sind) anhalten muss. Bei Nein-Instanzen ist dies nicht gefordert. (Man weiß dann aber bei einer laufenden TM nicht, ob das Eingabewort in L ist (und nur noch nicht akzeptiert wurde) oder nicht.)

6. **F:** Haben Sie gerade Spaß?

   A: Der Schreiber hat gerade Spaß! Und wollte umbedingt wieder auf sechs Fragen kommen... ;-)