4.3 Entscheidbarkeit und Aufzählbarkeit

Nachdem wir nun die Turingmaschine und verschiedene Varianten kennengelernt haben, wollen wir nun, so wie wir es auch bei den regulären und den kontextfreien Sprachen gemacht haben, eine zugehörige Sprachfamilie definieren. Dies machen wir zunächst ganz genauso wie bei endlichen Automaten und betrachten die Familie all jener Sprachen, die von Turingmaschinen akzeptiert werden. Dann werden wir aber sogar noch eine zweite Sprachfamilie einführen, bei der es um Sprachen geht, die nicht nur von Turingmaschinen akzeptiert werden, sondern für die sogar Turingmaschinen existieren, die stets anhalten. Diese Sprachfamilie ist für die Informatik von besonderer Bedeutung, denn da wir Turingmaschinen als abstraktes Modell für das Berechenbare bzw. für einen Algorithmus kennengelernt haben und da wir gesehen haben, dass das Akzeptieren von Sprachen und das Berechnen von Funktionen im Grunde gleich sind, erfasst diese Sprachfamilie gerade jene Probleme, die von Algorithmen, die stets terminieren, lösbar sind und dies sind gerade jene Algorithmen, die man typischerweise versucht zu entwerfen.

Im Anschluss werden wir dann ganz so wie schon bei REG und CF sehen, dass es Probleme gibt, die nicht mehr in diesen neuen Sprachfamilien liegen. Dies ist aber deshalb von deutlich schwerwiegenderer Bedeutung, da dies dann bedeutet, dass es keine Turingmaschine für ein Problem gibt, was bedeutet, dass es auch keinen Algorithmus ausgeführt auf irgendeinem Computer gibt, der das Problem löst. Da wir sehen werden, dass dies auch Probleme betrifft, die von großer praktischer Bedeutung sind, lernen wir damit eine wichtige Grenze unserer Möglichkeiten mit dem Computer kennen.

4.3.1 Die Sprachfamilien RE und REC

Als erstes können wir so wie wir es auch bei z.B. endlichen Automaten gemacht haben, einfach alle Sprachen, die von Turingmaschinen akzeptiert werden, in einer Sprachfamilie sammeln. Dies ergibt die Sprachfamilie der *aufzählbaren Sprachen*. Sie wird mit RE abgekürzt.

Definition 4.3.1 (Die aufzählbaren Sprachen)**.** *1. Eine Menge $M \subseteq \Sigma^*$ ist **(rekursiv) aufzählbar** genau dann, wenn eine DTM A existiert mit $L(A) = M$.*

2. Die Familie aller aufzählbaren Mengen bzw. Sprachen wird mit RE (recursively enumerable) bezeichnet.

Da die Turingmaschine anders als die bisherigen Automatenmodelle einen Unterschied macht zwischen Akzeptanz und dem tatsächlichen Halten (ein Wort muss ja nicht bis zum Ende gelesen werden, um es zu akzeptieren) und da das Halten aus algorithmischer Sicht von großer Bedeutung ist, da dies gleichbedeutend damit ist, dass ein Algorithmus bei jeder Eingabe terminiert, lohnt

es sich, die Sprachen, die von Turingmaschinen akzeptiert werden können, die bei jeder Eingabe halten, in einer gesonderten Familie zu sammeln.

Definition 4.3.2 (Die entscheidbaren Sprachen)**.** *1. Eine Menge $M \subseteq \Sigma^*$ ist **entscheidbar** genau dann, wenn eine DTM A existiert mit $L(A) = M$ und derart, dass A auf jeder Eingabe anhält.*

2. Die Familie aller entscheidbaren Mengen wird mit REC (recursive sets) bezeichnet.

Die von Turingmaschinen akzeptierten Sprachen bilden also die Sprachfamilie RE der aufzählbaren Sprachen. Die von Turingmaschinen, die stets anhalten, akzeptierten Sprachen bilden die Sprachfamilie REC der entscheidbaren Sprachen.

Ist eine Sprache M also entscheidbar, dann gibt es eine Turingmaschine A mit $L(A) = M$ *und* ferner hält A auf jeder Eingabe (in einem Endzustand, wenn das vorgelegte Wort in M ist, sonst in einem Nicht-Endzustand).

Ist eine Sprache M aufzählbar, dann gibt es (nur) eine Turingmaschine A mit $L(A) = M$.

Bei einer aufzählbaren Menge muss die Turingmaschine also insb. nicht bei jeder Eingabe anhalten. Bei Worten die in M sind, tut sie es (zumindest ist es leicht möglich A so zu konstruieren, indem man alle Kanten, die aus Endzuständen rausführen, entfernt). Bei Worten, die nicht in M sind, tut sie es aber nicht zwingend. Rechnet A dann lange, dann kann dies also daran liegen, dass das Wort zwar in M ist, A aber noch einige Zeit braucht, um dies zu berechnen, oder daran, dass das Wort nicht in M ist. Wir werden später noch sehen, dass es aufzählbare Sprachen gibt, die nicht entscheidbar sind. Für diese Sprachen ist dann genau das Problem, dass man den oben beschriebenen Fall nicht auflösen kann, d.h. man weiß dann nicht (und hat auch keine Möglichkeit dies herauszufinden), ob A nur noch länger rechnen muss, um zu einem Ergebnis zu kommen, oder ob A nie zu einem Ergebnis kommt.

Bisher haben wir noch kein Problem, das in RE ist, nicht aber in REC. Aus der Definition folgt aber zumindest sofort REC $\subseteq$ RE, denn eine Turingmaschine, die ein Problem entscheidet, die zählt dieses auch auf (dass sie zusätzlich sogar immer hält, wird sozusagen gar nicht benötigt).

Es gibt einige äquivalente Definitionen der Sprachfamilien REC und RE. Einige sind in den folgende beiden Sätze zusammengefasst. Die charakteristische Funktion χ_M einer Menge M ist dabei definiert durch $\chi_M(x) = 1$ gdw. $x \in M$.

Satz 4.3.3. *Eine Menge $M \subseteq \Sigma^*$ heißt **entscheidbar** genau dann, wenn*

1. die charakteristische Funktion $\chi_M : \Sigma^* \to \{0, 1\}$ berechenbar ist,

2. eine DTM A mit $L(A) = M$ existiert, die auf jeder Eingabe anhält.

Satz 4.3.4. *Eine Menge $M \subseteq \Sigma^*$ heißt* ***(rekursiv) aufzählbar*** *genau dann, wenn*

1. *$M = \emptyset$ ist oder eine totale, berechenbare Funktion $g : \mathbb{N} \to \Sigma^*$ existiert mit $g(\mathbb{N}) = M$,*
2. *eine k-Band off-line TM existiert, die jedes Wort der Menge M genau einmal auf ihr Ausgabeband schreibt,*
3. *$M = L(A)$ für eine DTM A gilt.*

Es ist eine gute Übung zu zeigen, dass die einzelnen Möglichkeiten tatsächlich äquivalent sind. Im ersten Fall zeigt man dafür, dass 1. und 2. sich gegenseitig implizieren und im zweiten Fall zeigt man, dass 2. aus 1. folgt, 3. aus 2. und zuletzt 1. aus 3. Dann hat man einen Ringschluss und ist fertig.

4.3.2 Beispiele entscheidbarer Sprachen

Wir wollen einige Beispiele für entscheidbare Sprache betrachten. Daraus wird dann auch als Zusammenhang zwischen allen bisher betrachteten Sprachfamilien

$$\text{REG} \subsetneq \text{CF} \subsetneq \text{REC} \subseteq \text{RE}$$

folgen.

Als Notation werden nachfolgend in den Sprachbeschreibungen eckige Klammern benutzt, um eine Kodierung zu notieren. Wir benutzen also z.B. $\langle A, w\rangle$, um eine Kodierung von A und w zu bezeichnen (z.B. ist dann A ein Automat und w ein Wort). Wir notieren allgemein $\langle M\rangle$ für ein mathematisches Objekt M (DFA, PDA, TM, Wort, ...). Wichtig ist, dass man M endlich beschreiben kann. Hat man z.B. einen DFA A, so könnte $\langle A\rangle$ einfach die Zeichenkette sein, die man erhält, wenn man alle Elemente von A (Zustandsmenge, Eingabealphabet, Übergangsfunktion usw.) hintereinander schreibt. Wir wollen nicht genau auf die Kodierungen eingehen, aber es sollte klar sein, dass es eine gibt. Auch wenn wir als Menschen den DFA aufschreiben, benutzen wir ja eine Kodierung. Notieren wir diese als Zeichenkette im Computer, ergibt sich wieder eine Kodierung. Wichtig für uns ist, dass solche Kodierungen existieren und dass die Turingmaschine mit diesen arbeiten kann. So kann sie z.B. aus der Kodierung eines DFAs die Überführungsfunktion auslesen und damit arbeiten.

Satz 4.3.5. *Die Sprache*

$$DFA_{acc} := \{\langle A, w\rangle \mid A \text{ ist ein DFA und akzeptiert } w\}$$

ist entscheidbar.

Beweis. Eine TM M, die DFA_{acc} entscheidet, arbeitet wie folgt:

1. Simuliere A mit Eingabe w
2. Endet die Simulation in einem Endzustand von A, akzeptiere, sonst lehne ab.

Für diese TM muss nun noch argumentiert werden, dass sie stets hält und dass sie stets die korrekte Antwort liefert. Wir wollen vorher aber noch einige Details liefern, die wir später dann meist nicht so ausführlich wiedergeben werden.

Zunächst soll die Eingabe $\langle A, w\rangle$ als "sinnvolle Repräsentation" gegeben sein, z.B. als Liste der Komponenten von A und diese Komponenten wieder als Listen der Elemente (alle Zustände hintereinander, dann die Symbole, dann die Übergangsfunktion als Liste von Tupeln usw.). Die DTM überprüft dann zu Anfang, ob wirklich ein DFA und ein Eingabewort vorliegt. Diese syntaktische Überprüfung geht stets problemlos, auch wenn die tatsächliche Implementierung in einer TM recht umständlich wird.

Bei der Simulation von A merkt M sich dann die aktuelle Position in w und den aktuellen Zustand von A, also im Grunde die Konfiguration von A.

Dann wird immer für jedes Tupel aus Zustand z (in dem A sich gerade befindet) und Symbol a (das A gerade lesen soll) der Zustand entsprechend $\delta(z, a)$ gewechselt und die Position im Wort um eins erhöht (d.h. die Nachfolgekonfiguration wird bestimmt).

Liest A das Wort zu Ende und ist dann in einem Endzustand, so akzeptiert M, sonst (A blockiert oder ist nicht in einem Endzustand) lehnt M ab.

So konstruiert, hält die TM M auf jeder Eingabe, denn wenn die Eingabe syntaktischer Unsinn ist, merkt sie dies sofort, ansonsten wird A auf w simuliert, was wegen der Eigenschaften des DFAs auch stets nach endlichen vielen Schritten zu einem Ergebnis führt (entweder hat A das Wort w zu Ende gelesen und ist in einem Endzustand oder A hat das Wort w zu Ende gelesen und ist nicht in einem Endzustand oder A hat das Wort nicht zu Ende gelesen, hat aber für das gerade zu lesende Symbol in dem aktuellen Zustand keinen Übergang). Ferner gibt M auch die korrekte Antwort aus, denn es ist leicht zu überprüfen, ob am Ende der Simulation A in einem Endzustand ist oder nicht und ob das Wort zu Ende gelesen wurde oder nicht. Damit hält M immer und gibt immer die korrekte Antwort aus, also entscheidet M die Sprache DFA_{acc}. □

In der Literatur werden einige Schritte oft übersprungen. Z.B. wird auf die Syntaxprüfung meist nicht explizit eingegangen. Die Eingabe wird als syntaktisch korrekt vorausgesetzt. Dies liegt daran, dass stets gute Kodierungen möglich sind und daran, dass die Turingmaschine dann ggf. die Eingabe schnell auf syntaktische Korrektheit testen kann. Wir werden hierauf in der Zukunft auch meist verzichten.

Ferner wird die Arbeitsweise einer Turingmaschine stets auf einem gewissen Abstraktionslevel beschrieben. Welcher geeignet ist, hängt wieder von den Kenntnissen der Zielgruppe ab. So kann z.B. der Schritt "die Turingmaschine

bestimmt den Nachfolgezustand des DFA" völlig in Ordnung sein, wenn jeder Leser gewisse Kenntnisse über DFAs hat und sich ggf. schnell selbst überlegen kann, dass dies klappt. Ein Schritt wie "die Turingmaschine bestimmt alle Zustände des DFAs, die durch ein Wort aus a^* erreichbar sind" würde aber vermutlich eine ausführlichere Begründung, wie sie dies tut, erfordern.

Zur Übung ist es wieder zu Anfang gut, sich sehr kleinschrittig zu überlegen, was die Turingmaschine machen muss und dies auch aufzuschreiben. Beim Lesen anderer Beweise ist es zur Übung gut, sich zu überlegen, wie dargestellte abstraktere Schritte durch kleinere Schritte tatsächlich ausgeführt werden können.

Wir betrachten als weiteres Beispiel die ganz ähnliche Frage der Akzeptanz eines Wortes durch einen NFA.

Satz 4.3.6. *Die Sprache*

$$NFA_{acc} := \{\langle A, w\rangle \mid A \text{ ist ein NFA und akzeptiert } w\}$$

ist entscheidbar

Beweis. Man kann hier so ähnlich vorgehen wie oben, muss sich dann aber nach jedem gelesenen Buchstaben die Menge aller Zustände, in denen der NFA nichtdeterministisch ist, merken und diese Menge manipulieren. Dies gelingt dann auch, da man wieder zeigen kann, dass die Turingmaschine stets anhalten muss und auch die richtigen Antworten liefert. Alternativ kann man auf von uns bereits gezeigte Konstruktionen zurückgreifen. Eine Turingmaschine, die obiges Problem löst, kann bei Vorlage von A und w wie folgt arbeiten:

1. Konstruiere zu A den Potenzautomaten B.
2. B ist ein DFA. Benutze die im vorherigen Satz konstruierte TM M, um $\langle B, w\rangle \in DFA_{acc}$ zu entscheiden.
3. Akzeptiere, falls M akzeptiert, sonst lehne ab.

Dieses Vorgehen klappt insb. deswegen, da wir von der Potenzautomatenkonstruktion wissen, dass sie korrekt ist und dass sie in einen stets terminierenden Algorithmus umgewandelt werden kann, der auch auf einer TM implementiert werden kann. □

Die nächsten zwei Probleme hängen abermals mit endlichen Automaten zusammen.

Satz 4.3.7. *Die Sprache*

$$DFA_{\emptyset} := \{\langle A\rangle \mid A \text{ ist ein DFA und } L(A) = \emptyset\}$$

ist entscheidbar.

Beweis. Die Entscheidbarkeit dieser Sprache behandeln wir in den Präsenzaufgaben. □

Satz 4.3.8. *Die Sprache*

$$DFA_{=} := \{\langle A, B\rangle \mid A \text{ und } B \text{ sind DFAs mit } L(A) = L(B)\}$$

ist entscheidbar.

Beweis. $L(A) = L(B)$ gilt genau dann, wenn jedes Wort, das in A ist, auch in B ist und wenn jedes Wort, das in B ist, auch in A ist. Dies kann man umformulieren. Bspw. darf es kein Wort geben, das in A ist und nicht in B. Mengentheoretisch ausgedrückt bedeutet dies, dass $L(A) \cap \overline{L(B)}$ leer sein muss. Ebenso muss auch $L(B) \cap \overline{L(A)}$ leer sein, woraus insgesamt

$$(L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B)) = \emptyset \text{ gdw. } L(A) = L(B)$$

folgt.

Will man sich von der Richtigkeit der obigen Aussage noch einmal genau überzeugen, gelingt der Beweis recht schnell. Sei $L(A) = L(B)$. Dann kann man auf der linken Seite überall $L(A)$ durch $L(B)$ ersetzen und erhält dann $L(B) \cap \overline{L(B)}$ und $(\overline{L(B)} \cap L(B)$, was beides offensichtlich $\emptyset$ ist. Daher gilt die linke Seite. Gilt andererseits die linke Seite, so wollen wir $L(A) = L(B)$ zeigen. Sei dazu $w \in L(A)$, dann kann nicht $w \notin L(B)$ sein, da dann $w \in \overline{L(B)}$ gelten würde und dann $L(A) \cap \overline{L(B)}$ nicht leer wäre. Daher muss $w \in L(B)$ gelten. Entsprechend folgt für $w \in L(B)$ durch eine analoge Argumentation mit der Menge $\overline{L(A)} \cap L(B)$, dass $w \in L(A)$ gelten muss, womit insgesamt $L(A) = L(B)$ folgt.

Es ist nun mit unsere bisherigen Techniken möglich aus den DFAs A und B einen DFA C mit

$$L(C) = (L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$$

zu konstruieren. Dazu baut man zu A und B jeweils einmal einen Komplementautomaten, dann mit diesen und den ursprünglichen Automaten zwei Produktautomaten und zuletzt einen Automaten, der gerade die Vereinigung akzeptiert. Es ist dann mit dem eben gezeigten

$$L(C) = \emptyset \text{ gdw. } L(A) = L(B)$$

und wir können den Leerheitstest des vorherigen Satzes nutzen, um $L(A) = L(B)$ zu entscheiden.

Im einzelnen arbeitet eine Turingmaschine, die $DFA_{=}$ bei Eingabe zweier DFAs A und B entscheidet, wie folgt:

1. Konstruiere C wie oben beschrieben.

2. Benutze eine TM M, die $DFA_\emptyset$ entscheidet, auf C.

3. Falls M akzeptiert, akzeptiere, sonst nicht.

Wichtig ist hier wieder, dass die einzelnen Schritte in endlicher Zeit möglich sind, also terminieren. Dies ist der Fall, da die einzelnen Konstruktionen, die beim ersten Schritt nötig sind, alle terminieren und da im zweiten Schritt M eine Turingmaschine ist, die bei jeder Eingabe anhält. Mit dem oben ausgeführten ist dann auch klar, dass genau dann akzeptiert wird, wenn $L(A) = L(B)$ gilt. Damit entscheidet die Turingmaschine das Problem $DFA_=$. □

Wir wollen nun noch ein weiteres Problem betrachten, das uns hilft, unsere bisherigen vier Sprachfamilien in Beziehung zu setzen.

Satz 4.3.9. *Die Sprache*

$$CFG_{acc} := \{\langle G, w\rangle \mid G \text{ ist eine CFG und generiert } w\}$$

ist entscheidbar.

Beweis. In den Übungsaufgaben wird gezeigt, dass zu jeder Grammatik G die Sprache $L(G)$ entscheidbar ist. Damit ist dieses Problem dann leicht zu entscheiden. Sei M_G die Turingmaschine, die $L(G)$ entscheidet. Bei Eingabe von G und w wird M_G konstruiert (es ist wichtig, dass dies möglich ist, was aber der Fall ist) und dann wird w einfach M_G vorgelegt. Akzeptiert M_G so wird akzeptiert, sonst abgelehnt.

Da es stets möglich ist M_G zu konstruieren und diese dann entscheidet, ob $w \in L(G)$ gilt oder nicht, terminiert das Verfahren immer und da M_G korrekt arbeitet, akzeptiert auch das hier vorgestellte Verfahren genau dann, wenn tatsächlich w von G generiert wird. □

Mit DFA_{acc} und CFG_{acc} bzw. den sehr ähnlichen Fragestellungen, dass $L(A)$ und $L(G)$ für jeden DFA A und jede CFG G entscheidbar sind und da wir eine Turingmaschine für z.B. $\{a^n b^n c^n \mid n \in \mathbb{N}\}$ angeben können, eine Sprache, die nicht kontextfrei ist, folgt

$$\text{REG} \subsetneq \text{CF} \subsetneq \text{REC} \subseteq \text{RE}$$

Neben den bisher gesehenen Sprachen gibt es noch etliche weitere entscheidbare Sprachen. Diese Sprachfamilie spielt in der Informatik eine besondere Rolle, da dies die Probleme sind, die wir von einem Computer lösen lassen können. Ein Algorithmus kann bei Vorlage einer Eingabe stets entscheiden, ob diese zu der Sprache gehört oder nicht, was aufgrund der Analogie zwischen Sprachen entscheiden und Funktionen berechnen damit gleichzusetzen ist, Lösungen zu berechnen. Zwei Probleme können nun auftreten. Eine Sprache könnte aus dieser Sprachfamilie herausfallen. Dies ist deswegen problematisch, da eine

Sprache, die bspw. nur aufzählbar, nicht aber entscheidbar ist, unangenehme Eigenschaften hat. Bei Vorlage eines Wortes w würde eine Turingmaschine zwar anhalten und akzeptieren, wenn w in der Sprache ist. Wenn w aber nicht in der Sprache ist, dann arbeitet die Turingmaschine unter Umständen unendlich lange und für einen Benutzer ist das Problem, dass er nie weiß, ob vielleicht noch berechnet wird, dass w in der Sprache ist oder ob die Turingmaschine nur immer weiter rechnet. Tatsächlich gibt es solche unangenehmen Probleme und tatsächlich sind sie weder selten noch unwichtig – im Gegenteil! Wir werden uns mit diesen Problemen im Abschnitt über Unentscheidbarkeit noch genauer befassen.

Ein zweites Problem betrifft die Zeit, die benötigt wird, um ein Problem zu lösen. Nur weil ein Problem entscheidbar ist, bedeutete dies nämlich nicht, dass die Zeit, die benötigt wird, um es tatsächlich zu lösen, akzeptabel ist. Brauchen wir länger zur Berechnung einer Lösung als das Universum existieren wird, so ist dies ganz bestimmt nicht akzeptabel, aber auch schon wenn eine Berechnung länger als ein paar Sekunden dauert, kann dies, je nach Anwendungsfall, nicht akzeptabel sein. Wir kommen hierauf im Kapitel zur Komplexitätstheorie zu sprechen.

4.3.3 Beispiele aufzählbarer Sprachen

Alle im vorherigen Abschnitt vorgestellten entscheidbaren Sprachen sind wegen REC $\subseteq$ RE auch aufzählbar. Eine weitere Sprache, die aufzählbar ist, aber für die einem auf den ersten Blick keine Möglichkeit einfällt, sie zu entscheiden, ist die folgende Sprache TM_{acc}.

Satz 4.3.10. *Die Sprache*

$$TM_{acc} := \{\langle M, w\rangle \mid M \text{ ist eine TM und akzeptiert } w\}$$

ist aufzählbar.

Beweis. Sei $\langle M, w\rangle$ die Eingabe, wobei M eine TM und w ein Wort ist.

1. Simuliere M auf w.
2. Wenn M jemals einen Endzustand erreicht, akzeptiere. Lehnt M ab, so lehne ab.

□

Man müsste sich wieder im einzelnen überlegen, wie M simuliert wird, dies ist aber ähnlich wie beim DFA recht einfach. Die Rechnung einer Turingmaschine ist zwar komplizierter, aber aus algorithmischer Sicht sind lediglich mehr Fälle zu beachten. Ein wirkliches Problem stellt dies nicht dar.

Es gibt also eine Turingmaschine M' mit $L(M') = TM_{acc}$. Sollte bei der Simulation von M auf w allerdings M in eine Endlosschleife geraten, so simuliert M' immer weiter. Dies ist in Ordnung, denn da M nicht akzeptiert, soll M' dies auch nicht. Die Sprache wird so aber nicht entschieden, sondern nur aufgezählt. Wir werden später zeigen, dass es tatsächlich nicht möglich ist, TM_{acc} zu entscheiden. Eine Aussage, die man gar nicht genug betonen kann, denn sie zeigt uns unsere ganz grundsätzlichen Grenzen bei der Arbeit mit dem Computer auf. Ist TM_{acc} nicht entscheidbar, dann ist es nicht möglich einen Algorithmus zu schreiben, der uns für andere Algorithmen berechnet, ob diese bei einer bestimmten Eingabe ein bestimmtes Ergebnis liefern oder nicht. Die Rolle des Algorithmus übernimmt hier die Turingmaschine M, die wir als Eingabe erhalten. Die Rolle der Eingabe des Algorithmus übernimmt das Eingabewort w der Turingmaschine M und die Rolle des Ergebnisses spielt die Frage, ob w von M akzeptiert wird oder nicht. Damit sind dann ganz grundsätzliche Probleme der Verifikation von Programmen außerhalb unserer Reichweite.

Weitere Probleme, die aufzählbar sind, sind im folgenden Satz aufgelistet. Bei allen wird zum Nachweis einfach die Turingmaschine aus der Eingabe simuliert und gewartet, ob das gewünschte Ereignis wie das Anhalten der Turingmaschine oder die Ausgabe einer 0 passiert.

Satz 4.3.11. *Die Sprachen*

$$
\begin{aligned}
TM_{halt} &:= \{\langle M, w\rangle \mid M \textit{ ist eine TM und hält auf } w\} \\
TM_{print0} &:= \{\langle M\rangle \mid \textit{gestartet auf } \lambda \textit{ gibt } M \textit{ irgendwann } 0 \textit{ aus}\} \\
TM_{reach} &:= \{\langle M, z\rangle \mid \textit{es gibt eine Eingabe, bei der } M \textit{ den Zustand } z \textit{ besucht}\}
\end{aligned}
$$

sind aufzählbar.

Bei der letzten Sprache genügt es nicht M nur auf einer Eingabe zu simulieren, stattdessen ist es nötig, die Turingmaschine nach und nach auf allen Worten starten zu lassen. Dazu lässt man sie erst einen Schritt auf jedem Wort der Länge 1 simulieren, dann einen weiteren Schritt auf jedem Wort der Länge 1 und einen Schritt auf jedem Wort der Länge 2, dann einen weiteren Schritt auf jedem Wort der Länge 1 und 2 und einen Schritt auf jedem Wort der Länge 3 und immer so weiter. Gibt es ein Eingabewort w der Länge n, bei dem z nach m Schritten besucht wird, so werden auf diese Art und Weise irgendwann auch der $m$te Schritt bei Worten der Länge n simuliert und dann wird dies erkannt. Diese Technik werden wir auch bei Abschlusseigenschaften nutzen, insb. wenn wir zeigen wollen, dass die aufzählbaren Sprachen gegenüber Vereinigung abgeschlossen sind.

4.3.4 Abschlusseigenschaften

Bei den regulären Sprachen haben wir gesehen, dass diese gegenüber allen wichtigen Operationen abgeschlossen sind. So sind bspw. $L_1 \cup L_2$, $L_1 \cap L_2$ und $\overline{L_1}$ wieder regulär, wenn L_1 und L_2 dies sind. Bei den kontextfreien Sprachen haben wir gesehen, dass diese zwar gegenüber Vereinigung, Konkatenation und Sternbildung abgeschlossen sind, nicht aber gegenüber Durchschnitt und Komplementbildung. Man kann nun wieder fragen, wie sich dies bei den aufzählbaren und entscheidbaren Sprachen verhält? Wir betrachten hier nur die drei wichtigsten Operationen.

Satz 4.3.12. *1. Die entscheidbaren Sprachen sind gegenüber $\cap$, $\cup$ und Komplementbildung abgeschlossen.*

2. Die aufzählbaren Sprachen sind gegenüber $\cap$ und $\cup$ abgeschlossen. Sie sind nicht gegenüber Komplementbildung abgeschlossen.

Beweis. Der Fall der entscheidbaren Sprachen ist recht einfach. Seien L_1 und L_2 entscheidbare Sprachen und seien M_1 und M_2 Turingmaschinen, die L_1 bzw. L_2 entscheiden. Um $L_1 \cap L_2$ zu entscheiden, führt man auf einer Eingabe w erst M_1, dann M_2 aus. Da beide stets halten, geht dies problemlos. Akzeptieren beide, wird akzeptiert, sonst abgelehnt.

Für $L_1 \cup L_2$ verfährt man genauso, akzeptiert aber, wenn bereits eine der beiden Turingmaschinen M_1 oder M_2 akzeptiert. Für die Komplementbildung dreht man einfach das Ergebnis um und ist dann sofort fertig.

Der Fall der aufzählbaren Sprachen ist spannender. Für $L_1 \cap L_2$ kann man wieder genauso verfahren. Problematisch ist ja ohnehin nur der Fall, dass M_1 oder M_2 unendlich lange laufen. Die ist hier aber noch nicht schlimm, denn wenn z.B. M_1 unendlich lange läuft und daher M_2 nie gestartet wird, so ist dies nicht tragisch, denn wenn M_1 ein Wort nicht akzeptiert, so ist dieses Wort ganz bestimmt nicht in $L_1 \cap L_2$ und es ist also egal, dass M_1 immer weiter arbeitet, ohne dass M_2 jemals gestartet wird.

Anders bei $L_1 \cup L_2$ hier funktioniert diese Methode nicht, denn sollte M_2 ein Wort w akzeptieren, so ist das Wort in $L_1 \cup L_2$. Dies würde aber nie bemerkt werden, wenn M_1 das Wort w nicht akzeptiert, unendlich lange arbeitet und so verhindert, dass M_2 je gestartet wird. Wir verfahren hier daher anders. Statt erst M_1 und dann M_2 zu starten, simulieren wir von beiden jeweils nur einen Schritt, d.h. wir simulieren zunächst einen Schritt von M_1 und dann einen Schritt von M_2. Im Anschluss simulieren wir wieder von beiden einen Schritt und so weiter. Ist ein Eingabewort w in L_1 oder L_2, so wird eine der beiden Maschinen letztendlich w akzeptieren. Sind beide in einer Endlosschleife, so ist auch die Maschine, die beide abwechselnd simuliert in einer Endlosschleife und hält nie, das ist dann aber nicht weiter schlimm, da wir $L_1 \cup L_2$ ja nur aufzählen wollen.

Betrachten wir zuletzt die Komplementbildung. Auch hier können wir nicht einfach wie bei den entscheidbaren Sprachen das Ergebnis umdrehen, denn eine Turingmaschine für L_1 könnte in Endlosschleifen geraten und dies liefert kein Ergebnis, dass eine Turingmaschine für $\overline{L_1}$ umdrehen könnte. Tatsächlich sind die aufzählbaren Sprachen auch gar nicht gegenüber Komplement abgeschlossen, so dass dieser Versuch scheitern muss. Dies folgt aus dem gleich folgenden Satz und der später noch zu zeigenden Tatsache, dass es aufzählbare Sprachen gibt, die nicht entscheidbar sind. □

Satz 4.3.13. *Eine Sprache L ist entscheidbar genau dann, wenn L und $\overline{L}$ (das Komplement von L) aufzählbar sind.*

Beweis. Zunächst die Richtung von links nach rechts. Ist L entscheidbar, dann ist L auch aufzählbar. Dreht man zudem die Ergebnisse einer Turingmaschine, die L entscheidet, um, so hat man eine Turingmaschine, die $\overline{L}$ entscheidet. Damit ist $\overline{L}$ dann auch aufzählbar.

Nun die spannendere Richtung von rechts nach links. Seien A und $\overline{A}$ Turingmaschinen, die L bzw. $\overline{L}$ akzeptieren. Eine Turingmaschine B, die L entscheidet, kann nach der gleichen Idee wie beim obigen Beweis, dass RE gegenüber Vereinigung abgeschlossen ist, konstruiert werden. B arbeitet wie folgt: Bei Eingabe von w wird erst ein Schritt von A simuliert, dann ein Schritt von $\overline{A}$. Im Anschluss wird wieder ein Schritt von A simuliert, dann wieder einer von $\overline{A}$ und so weiter. Da entweder $w \in L$ oder $w \in \overline{L}$ gelten *muss*, muss A oder $\overline{A}$ akzeptieren. Akzeptiert A, so wird akzeptiere. Akzeptiert $\overline{A}$, so wird abgelehnt. Damit haben wir eine Maschine konstruiert, die stets anhält und stets das korrekte Ergebnis liefert, die also L entscheidet. □

Aus dem eben gezeigten Satz folgt, dass RE = REC gilt, wenn die aufzählbaren Sprachen gegenüber Komplement abgeschlossen wären. Warum? Wären die aufzählbaren Sprachen gegenüber Komplement abgeschlossen, so würde für jedes $L \in$ RE auch $\overline{L} \in$ RE gelten. Der eben gezeigte Satz besagt dann aber, dass $L \in$ REC gilt. Damit hätten wir gezeigt, dass jedes $L \in$ RE auch in REC ist, dass also RE $\subseteq$ REC gilt, woraus RE = REC folgt, da REC $\subseteq$ RE ja ohnehin gilt.

Da wir später noch sehen werden, dass tatsächlich REC $\subsetneq$ RE gilt, folgt, dass die aufzählbaren Sprachen nicht gegenüber Komplement abgeschlossen sein können.

Wir fassen die vergangenen Abschnitte zusammen. Wir haben die Sprachfamilie REC der entscheidbaren und die Sprachfamilie RE der aufzählbaren Sprachen eingeführt. Eine Sprache L ist aufzählbar, wenn es eine Turingmaschine M gibt, die L akzeptiert, also eine mit $L(M) = L$. Eine Sprache ist entscheidbar, wenn es eine Turingmaschine M gibt, die L akzeptiert und M zudem bei jeder Eingabe anhält.

Wir haben dann verschiedene entscheidbare und aufzählbare Sprachen gesehen. Die entscheidbaren Sprachen sind für die Informatik von ganz besonderer Bedeutung, da diese Sprachen Probleme sind, die durch einen Algorithmus gelöst werden können. Wenn eine Sprache entscheidbar ist, ist das also erstmal ein gutes Zeichen.

Im letzten Abschnitt haben wir uns dann noch mit Abschlusseigenschaften von REC und RE beschäftigt. Wir haben gesehen, dass REC gegenüber Vereinigung, Durchschnitt und Komplement abgeschlossen ist, RE auch gegenüber Vereinigung und Durchschnitt, nicht aber gegenüber Komplement, wobei wir für die letzte Aussage noch benötigen, dass es tatsächlich Sprachen gibt, die aufzählbar, aber nicht entscheidbar sind. Wir werden dies im nächsten Abschnitt sehen. Bisher stellt sich unser Wissen über die vier Sprachfamilien REG, CF, REC und RE so dar

$$\text{REG} \subsetneq \text{CF} \subsetneq \text{REC} \subseteq \text{RE}$$

Eine typische reguläre Sprache ist z.B. durch den regulären Ausdruck a^* gegeben. Eine typische kontextfreie Sprache, die nicht mehr regulär ist, ist z.B. die Sprache $\{a^n b^n \mid n \in \mathbb{N}\}$. Eine typische entscheidbare Sprache, die nicht mehr kontextfrei ist, ist $\{a^n b^n c^n \mid n \in \mathbb{N}\}$. Eine typische aufzählbare Sprache, ist TM_{acc} und wir werden noch sehen, dass diese nicht mehr entscheidbar ist.

Fragen

1. Sind alle regulären Sprachen entscheidbar?
   a) Ja!
   b) Nein!
   c) Keine Teilmengenbeziehung!
2. Sind alle kontextfreien Sprachen entscheidbar?
   a) Ja!
   b) Nein!
   c) Keine Teilmengenbeziehung!
3. Wenn eine Sprache entscheidbar ist, ist dann auch das Komplement entscheidbar?
   a) Ja!
   b) Nein!
4. Wenn zwei Sprachen L_1, L_2 aufzählbar sind, ist dann auch $L_1 \cup L_2$ aufzählbar?
   a) Ja!
   b) Nein!