4.4 Unentscheidbarkeit (und darüberhinaus)

Wir haben nun etliche entscheidbare Sprachen gesehen und mit TM_{acc} eine Sprache gesehen, die wir zwar aufzählen können, für die es uns aber schwer fällt, eine Turingmaschine anzugeben, die sie entscheidet. Tatsächlich ist dies auch nicht möglich und dies wollen wir in diesem Abschnitt zeigen.

Darauf aufbauend wollen wir dann noch weitere Probleme als unentscheidbar nachweisen und auf diese Weise eine Technik kennenlernen, mit der wir Probleme als unentscheidbar nachweisen können.

Zum Abschluss wollen wir dann noch eine Sprache sehen, die nicht einmal mehr aufzählbar ist.

4.4.1 Ein unentscheidbares Problem

Wir wollen in diesem Abschnitt zeigen, dass die Sprache

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

nicht entscheidbar ist. Wir wissen bereits, dass TM_{acc} aufzählbar ist (man starte einfach M auf w und akzeptiere, wenn M dies tut) und Versuche TM_{acc} zu entscheiden schlagen fehl, aber ein Beweis, dass dies tatsächlich nicht möglich ist, steht noch aus. Dies werden wir gleich nachholen. Vorher wollen wir noch einmal betonen, wie bedeutend diese und verwandte Fragestellungen sind. Die Frage, ob eine Turingmaschine M auf einem Eingabewort w anhält, ist im Grunde die selbe wie die Frage, ob ein Programm in einer Programmiersprache wie Java oder C bei einer bestimmten Eingabe anhält. Diese Frage nicht algorithmisch lösen zu können ist ein ernstes Problem in der Verifikation von Software und Hardware und schränkt uns bedeutend ein. Zudem macht uns die Erkenntnis, dass es ein nicht entscheidbares Problem gibt, ganz deutlich, dass ein Computer nicht alles kann. Es gibt Probleme, die sind einer Lösung durch ihn nicht zugänglich.

Satz 4.4.1. *Die folgende Sprache ist* nicht entscheibar

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

Beweis. Angenommen TM_{acc} wäre entscheidbar. Sei H eine Turingmaschine, die TM_{acc} entscheidet. Wir wollen nun einen Widerspruch erzeugen. Dies gelingt uns, indem wir mit erlaubten Konstruktionsschritten eine Turingmaschine bauen, die H als Unterroutine benutzt und letztendlich dann aber paradoxe Dinge tut, also Dinge, die nicht sein können. Wir haben dann einen Widerspruch und da alle Zwischen- und Konstruktionsschritte aber möglich sind, muss die ursprüngliche Annahme falsch sein und H kann also nicht existieren.

Sei also H eine Turingmaschine, die TM_{acc} entscheidet. H tut folgendes:

- H akzeptiert $\langle M, w\rangle$, wenn M das Wort w akzeptiert.
- H lehnt $\langle M, w\rangle$ ab, wenn M das Wort w nicht akzeptiert.

Kürzer notieren wir dies hier als

- $H(\langle M, w\rangle) = 1$, wenn $M(w) = 1$
- $H(\langle M, w\rangle) = 0$, wenn $M(w) \neq 1$

Wir konstruieren nun eine TM D, die H als Subroutine benutzt. D nimmt als Eingabe eine TM M (die Eingabe ist also $\langle M\rangle$) und arbeitet wie folgt:

1. Starte H mit Eingabe $\langle M, \langle M\rangle\rangle$.
2. Drehe das Ergebnis von H um und gebe es aus.

D tut also nichts anderes als H mit einer speziellen Eingabe zu starten. Damit ist nun

- $D(\langle M\rangle) = 1$, wenn M die Eingabe $\langle M\rangle$ nicht akzeptiert
- $D(\langle M\rangle) = 0$, wenn M die Eingabe $\langle M\rangle$ akzeptiert

Insgesamt haben wir damit jetzt

$$\begin{aligned} H(\langle M, w\rangle) &= \begin{cases} 1, & \text{wenn } M(w) = 1 \\ 0, & \text{wenn } M(w) \neq 1 \end{cases} \\ D(\langle M\rangle) &= \begin{cases} 1, & \text{wenn } M(\langle M\rangle) \neq 1 \\ 0, & \text{wenn } M(\langle M\rangle) = 1 \end{cases} \end{aligned}$$

Was passiert nun, wenn man D mit $\langle D\rangle$ ausführt? Wenn man D also sich selbst als Eingabe vorlegt? D ruft dann ja H auf, um zu ermitteln ob D auf D anhält – dreht dann aber noch das Ergebnis um! Setzt man oben bei $D(\langle M\rangle) = \ldots$ überall wo M steht ein D ein, so erhält man nun

$$D(\langle D\rangle) = \begin{cases} 1, & \text{wenn } D(\langle D\rangle) \neq 1 \\ 0, & \text{wenn } D(\langle D\rangle) = 1 \end{cases}$$

Dies ist ein Widerspruch! $D(\langle D\rangle)$ kann ja unmöglich 1 sein, wenn es nicht 1 ist oder 0 wenn es 1 ist. Dies macht keinen Sinn und wir haben einen Widerspruch. Damit kann die Annahme nicht stimmen und H kann nicht existieren. TM_{acc} ist also unentscheidbar! □

Mit diesem Ergebnis haben wir eine erste nicht entscheidbare Sprache gefunden. Da wir wissen, dass TM_{acc} aufzählbar ist, haben wir damit nun REC $\subsetneq$ RE und damit insgesamt

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

Man mache sich einmal klar, wie obiger Beweis funktionieren würde, wenn man statt mit einer Turingmaschine H mit einer Java-Routine argumentieren würde. Nehmen wir also an, es gäbe eine Java-Routine f, die eine andere Java-Routine g und deren Argumente als Eingabe nimmt und *true* zurückliefert, wenn g *true* liefert und *false* sonst. Die Argumentation liefe dann ganz genau so! Das obige Problem ist nicht eines, das auf Turingmaschinen beschränkt ist. Es gilt ganz genauso für gängige Programmiersprachen und zeigt uns damit sehr drastisch unsere Grenzen auf.

4.4.2 Weitere unentscheidbare Probleme

Wir wollen nun eine Technik vorstellen, mit der man Probleme als unentscheidbar nachweisen kann. Diese Technik ist im Kern ein Widerspruchsbeweis. Wir wollen die Technik dann im Anschluss an einem einfachen und einem komplizierteren Problem einüben.

Will man ein neues Problem, ausgedrückt durch eine Sprache L, als unentscheidbar nachweisen, so ist das übliche Vorgehen das folgende:

1. Wir nehmen an, L wäre entscheidbar. Sei M eine Turingmaschine, die L entscheidet.

2. Unter Nutzung von M (M wird also quasi als Unterroutine benutzt) wird nun eine Turingmaschine konstruieren, die eine schon als unentscheidbar nachgewiesene Sprache entscheidet.

3. Das ist dann ein Widerspruch, also kann M nicht existieren.

Man nennt dies eine *Reduktion*, und sagt, dass das unentscheidbare Problem auf das neue Problem reduziert wurde (bzw. die Sprachen aufeinander reduziert wurden).

Um die Technik zu illustrieren, zeigen wir zunächst die Unentscheidbarkeit einer Sprache, die der Sprache TM_{acc} sehr ähnelt. Bei TM_{acc} ging es um die Akzeptanz eines Wortes. Nun geht es nur noch um das Halten der Turingmaschine.

Satz 4.4.2. *Die Sprache*

$$TM_{halt} := \{\langle M, w\rangle \mid M \textit{ ist eine TM und hält auf } w\}$$

ist nicht entscheidbar.

Beweis. Wie in dem Vorgehen oben beschrieben, nehmen wir zunächst an TM_{halt} wäre entscheidbar. Sei H eine Turingmaschine, die TM_{halt} entscheidet. Unser Ziel ist es nun, eine neue Turingmaschine zu konstruieren, die H als Unterroutine benutzt, und die eine Sprache entscheidet, von der wir schon wissen,

dass sie unentscheidbar ist. Da wir bisher nur eine unentscheidbare Sprache kennen, ist das Ziel hier einfach zu formulieren: Wir wollen eine Turingmaschine konstruieren, die TM_{acc} entscheidet.

Wir geben nun die Arbeitsweise einer neuen Turingmaschine S an, die TM_{acc} entscheiden soll. S benutzt die Turingmaschine H, die nach Annahme TM_{halt} entscheidet. Bei Eingabe $\langle M, w\rangle$ arbeitet S wie folgt:

1. Starte H mit Eingabe $\langle M, w\rangle$.
2. Lehnt H ab, lehne ab.
3. Akzeptiert H, simuliere M auf w.
4. Wenn M akzeptiert, akzeptiere, sonst lehne ab.

Bevor wir gleich zeigen, dass S tatsächlich TM_{acc} entscheidet, wollen wir einmal darauf eingehen, wie man auf die Arbeitsweise von S kommt. Unser Ziel ist es ja, TM_{acc} zu entscheiden. Unsere Eingabe ist also die Kodierung einer Turingmaschine M und eines Wortes w und wir wollen entscheiden, ob M das Wort w akzeptiert. Würden wir nun einfach M auf w starten bzw. simulieren, so hätten wir Probleme, wenn diese Simulation nicht abbricht. Dies ist tatsächlich das einzige Problem, denn es gibt ja nur drei Fälle: Entweder erreicht M auf w einen Endzustand. Dann kann die Simulation aufhören, da wir nun wissen, dass M das Wort w akzeptiert. Dann kann es sein, dass M anhält und im Laufe der Rechnung nie einen Endzustand besucht hat. Auch dann können wir mit Sicherheit die Eingabe ablehnen, da M also w nicht akzeptiert. Und zuletzt gibt es noch den Fall, dass M immer weiter läuft und zwar nie einen Endzustand besucht aber auch nie aufhört zu rechnen, die Simulation stoppt also nie. Diesen dritten, störenden Fall, können wir nun aber gerade durch eine Turingmaschine, die H entscheidet, abfangen. Wir fragen H, ob M auf w anhält. Ist dies nicht der Fall, können wir (ohne M zu simulieren!) sofort ablehnen, da wir dann wissen, dass M immer weiter rechnet und nicht akzeptieren kann. – Moment! An dieser Stelle tritt nun noch ein störender Fall auf. Nach unserer Definition kann eine Turingmaschine immer weiter rechnen und trotzdem (vorher mal) einen Endzustand besucht haben, also das Wort akzeptieren. Es gibt also auch unendlich lange Erfolgsrechnungen! Um mit diesem Problem fertig zu werden, geht man entweder davon aus, dass die Turingmaschine sofort als Eingabe so gegeben ist, dass aus einem Endzustand keine Kanten mehr herausführen (eine Rechnung in einem Endzustand also stets endet) oder aber man (und dies rechtfertigt auch, dass man das zuvor gesagte überhaupt annehmen darf) lässt obige Turingmaschine S als erste noch den Schritt machen, dass sie M gerade so anpasst, das sie alle Kanten, die bei M aus Endzuständen herausführen, entfernt.

Nun klappt die Überlegung von eben. Wir fragen also H, ob M auf w anhält. Tut M dies nicht, dann wissen wir, dass M nicht akzeptiert. Sagt uns H aber,

dass M auf w anhält, dann können wir nun sorglos, M auf w simulieren, denn die Simulation muss nach endlich vielen Schritten anhalten (gerade dies hat uns H ja eben gesagt). Die Simulation endet dann entweder in einem Endzustand oder nicht und wir wissen dann, ob M das Wort w akzeptiert oder nicht und können entsprechend antworten.

So kann man auf die Idee kommen, wie S zu konstruieren ist. Die Fragen, die man sich bei der Konstruktion stellen muss sind, welche Probleme treten auf, wenn ich die Sprache TM_{acc} entscheiden will und wie kann mir eine Turingmaschine, die TM_{halt} entscheidet, dabei helfen?

Wir zeigen nun noch, dass S tatsächlich die Sprache TM_{acc} entscheidet. Zunächst merken wir noch an, dass S als erstes, was wir oben nicht extra erwähnt haben, die Eingabe auf syntaktische Korrektheit prüft und dass S alle Kanten, die bei M aus Endzuständen herausführen, entfernt. Eine Erfolgsrechnung von M hält damit in einem Endzustand, sobald ein Endzustand das erste Mal besucht wird. Beides wird oft nicht mehr explizit erwähnt.

Wir wollen nun zeigen, dass S die Sprache TM_{acc} entscheidet. Dazu müssen wir zeigen, dass S stets anhält und dass tatsächlich $L(S) = TM_{acc}$ gilt. Zunächst terminieren die ersten beiden Schritte nach endlich viel Zeit, da H die Sprache TM_{halt} entscheidet und daher auch stets terminiert. Nun starten wir im dritten Schritt M nur dann auf w, wenn vorher H akzeptiert hat. Dies bedeutet aber ja gerade, dass M auf w anhält. Daher endet die Simulation im dritten Schritt auch nach endlich vielen Schritten. Der vierte Schritt terminiert auch und damit terminiert S stets und hält an.

Nun zu der Korrektheit der Antwort. Diese ist schnell zu sehen. Wenn S bereits im zweiten Schritt ablehnt, dann tut sie dies, wenn H ablehnt, was bedeutet, dass M auf w nicht anhält. Mit der Manipulation von eben kann also kein Endzustand besucht werden (da M sonst ja anhalten würde) und also akzeptiert M ganz bestimmt nicht w. Daher ist diese Antwort richtig. Sonst wird im dritten Schritt M auf w simuliert und ja gerade akzeptiert, wenn M akzeptiert und abgelehnt, wenn M dies tut. Damit akzeptiert S genau dann, wenn M das Wort w akzeptiert und wir sind fertig.

Da wird nun eine Turingmaschine haben, die eine unentscheidbare Sprache entscheidet, ist dies ein Widerspruch und die ursprüngliche Annahme muss falsch sein. TM_{halt} ist also unentscheidbar. □

TM_{halt} wird als *Halteproblem* bezeichnet und tritt oft in der Literatur auf. Die Sprachen TM_{halt} und TM_{acc} sind allerdings so ähnlich, dass einige Autoren auch TM_{acc} als Halteproblem bezeichnen.

Der Beweis, dass TM_{halt} unentscheidbar ist, verlief noch recht übersichtlich. Als zweites Beispiel für einen Unentscheidbarkeitsbeweis wollen wir ein komplizierteres Problem betrachten. Wir nennen dafür einen Zustand einer Turingmaschine einen *nutzlosen Zustand*, wenn er bei keinem Eingabewort jemals besucht wird.

Satz 4.4.3. *Die Sprache*

$$\texttt{UselessState} = \{\langle A, q\rangle \mid A \text{ ist eine TM und } q \text{ ein nutzloser Zustand von } A\}$$

ist unentscheidbar.

Beweis. Wir nehmen zunächst wieder an, `UselessState` wäre entscheidbar. Sei A_{US} eine Turingmaschine, die das Problem entscheidet. Wir wollen damit nun das Halteproblem TM_{halt} entscheiden.

Wir beachten zunächst, dass die folgende Konstruktion möglich ist: Zu einer gegebenen Turingmaschine M mit Zustandsmenge Z und Bandalphabet Γ kann eine Turingmaschine M' konstruiert werden mit

- einem neuen Zustand z_{neu} und
- für jeden Zustand $z \in Z$ von M und jedes Symbol $x \in \Gamma$, für das es keinen Übergang aus z in M gab (d.h. es gab keine Kante (z, x, X, Y, z') in M mit X, Y, z' beliebig) wird eine neue Kante (z, x, x, R, z_{neu}) hinzugefügt

Mit dieser Konstruktion erreicht man folgendes: hält M an, so kann M' noch einen Übergang nach z_{neu} machen. Dort hält dann M'. M hält also genau dann in irgendeinem Zustand, wenn M' in z_{neu} hält. Man beachte, wie hier ein Zusammenhang zwischen dem Halten von M und dem Benutzen des Zustandes von M' hergestellt wird. Ein Problem ist nun noch, dass z_{neu} nur dann nutzlos wäre, wenn M auf keinem Eingabewort anhält. Für das Halteproblem ist aber nur gefragt, ob M auf w anhält. Nun ist es aber zu M und w möglich eine Turingmaschine M'' zu konstruieren, die bei jeder Eingabe das Band löscht und dann w auf M' startet, wobei M' wie eben beschreiben aus M hervorgeht. Damit tut M'' dann bei jeder Eingabe das gleiche (nämlich im Grunde M auf w zu starten). Wenn nun aber M auf w hält, dann hält M'' in z_{neu} (und besucht daher insbesondere z_{neu}). Wenn M auf w nicht hält, dann besucht M'' den Zustand z_{neu} nicht und tut dies bei keinem Eingabewort (da ja bei jedem M auf w gestartet wird), d.h. z_{neu} wäre dann nutzlos.

Mit diesen Überlegungen können wir nun die Arbeitsweise einer Turingmaschine S angeben, die `UselessState` entscheidet. Bei Eingabe $\langle M, w\rangle$ arbeitet S wie folgt:

1. Konstruiere aus $\langle M, w\rangle$ die oben beschriebene Turingmaschine M''.
2. Starte A_{US} auf $\langle M'', z_{neu}\rangle$.
3. Akzeptiert A_{US}, dann ist z_{neu} ein nutzloser Zustand und nach obigem hält dann M nicht auf w und wir lehnen ab.
4. Lehnt A_{US} ab, so ist z_{neu} nicht nutzlos, wird also besucht. Dann aber hält M auf w an und wir akzeptieren.

Man kann wieder schnell argumentieren, dass S stets terminiert. Insbesondere terminieren die ersten beiden Schritte und daher auch S. Mit obigen Erklärungen ist auch klar, dass S gerade in den richtigen Fällen die Eingabe akzeptiert. Damit entscheidet S das Halteproblem, was aber nicht sein kann. A_{US} existiert also nicht und `UselessState` ist folglich unentscheidbar. □

Will man nicht ganz so ausführlich vorgehen und wie oben neue Kanten explizit beschreiben, so genügt es auch wie folgt ähnlich aber mit einer kleinen Variation vorzugehen und insb. einen etwas höheren Abstraktionsgrad beim Beweis zu wählen.

Wir nehmen wieder an das Problem sei entscheidbar und A eine Turingmaschine, die es entscheidet. Man kann nun bei Vorlage von M und w (für das Halteproblem) eine Turingmaschine M'' konstruieren, die ihre Eingabe ignoriert und die stets M auf w simuliert. M'' wechselt, sollte M anhalten, dann noch in einen neuen Zustand z_{neu}, der sonst nicht besucht wird. (Wichtig hierfür ist sich klar zu machen, dass M'' tatsächlich M simulieren kann und prüfen kann, ob M anhält.) Eine Turingmaschine, die das Halteproblem entscheidet, arbeitet nun wie folgt: Bei Vorlage von M und w, wird M'' konstruiert und A wird M'' und z_{neu} vorgelegt. Akzeptiert A, so ist z_{neu} nutzlos, was bedeutet, dass M auf w nicht anhält und wir lehnen ab. Lehnt A ab, so ist z_{neu} nicht nutzlos, also hält M auf w und wir akzeptieren. Die Turingmaschine entscheidet damit das Halteproblem, was nicht sein kann.

Hier haben wir also durch die auf einem höheren Abstraktionsgrad beschriebene Konstruktion erheblich den Beweis abgekürzt. Dafür wird aber mehr vom Leser verlangt.

Damit haben wir von zwei weiteren Problemen gezeigt, dass sie unentscheidbar sind. Will man von einem neuen Problem zeigen, dass es unentscheidbar ist, so wendet man die eingangs vorgestellte Technik der Reduktion an. Es ist dabei aber ein kreativer Vorgang, die Turingmaschine zu konstruieren, die das Problem entscheidet, von dem man schon weiß, dass es unentscheidbar ist. Das zweite Beispiel illustriert recht gut, dass dies recht schnell zu einer Herausforderung werden kann. Mit etwas Übung kann man aber lernen, unentscheidbare Probleme in vielen Fällen recht gut erkennen zu können.

4.4.3 Nicht mal mehr aufzählbar

Nachdem wir nun mehrere unentscheidbare Probleme kennengelernt haben und in der Kette der Sprachfamilien

$$\mathrm{REG} \subsetneq \mathrm{CF} \subsetneq \mathrm{REC} \subsetneq \mathrm{RE}.$$

für jede Sprachfamilie Sprachen kennen, die in dieser Klasse liegen, aber nicht mehr in den darunter liegenden Klassen, kann man sich nun noch die Frage stellen, ob es nicht sogar Sprachen gibt, die nicht einmal mehr in RE sind?

Solche Sprachen gibt es tatsächlich. Für eine solche Sprache gibt es also nicht einmal mehr eine Turingmaschine, die sie akzeptiert.

Um diese Sprache ausdrücken zu können, überlegen wir uns zunächst folgendes. Wir halten ein Alphabet wie z.B. $\{0,1\}$ fest. Die Worte, die mit diesem Alphabet gebildet werden können, können wir erst der Länge nach sortieren. Worte gleicher Länge sortieren wir dann noch lexikalisch, wobei 0 in der Ordnung vor 1 kommt. Die Worte lassen sich dann auflisten

$$0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, 0000, \ldots$$

So kann man dann von einem Wort $w_1, w_2, w_3, \ldots$ sprechen.

Ganz genauso können wir eine Turingmaschine M über dem Alphabet $\{0,1\}$ kodieren, z.B. indem wir zunächst jedes Element des Tupels $M = (Z, \Sigma, \Gamma, \delta, z_0, Z_{end})$ mit unserem normalen Alphabet aufschreiben und die so entstandene Zeichenkette dann binär kodieren (ein Computer tut ja auch nichts anderes, wenn er eine Zeichenkette speichert). Wir können dann die Zeichenketten, die tatsächlich eine Turingmaschine kodieren so wie eben zunächst nach Länge und dann lexikalisch sortieren. Dies ergibt dann wieder eine Auflistung $M_1, M_2, M_3, \ldots$ aller Turingmaschinen.

Insgesamt haben wir jetzt also über dem Alphabet $\{0,1\}$ eine Auflistung aller Worte $w_1, w_2, w_3, \ldots$ und eine Auflistung aller Turingmaschinen $M_1, M_2, M_3, \ldots$. Man beachte, dass dies alle Turingmaschinen sind, nicht bloss z.B. alle Turingmaschinen auf dem Eingabealphabet $\{0,1\}$. Die Turingmaschinen werden lediglich mit dem Alphabet $\{0,1\}$ *kodiert*, es werden aber alle möglichen Turingmaschinen aufgelistet.

Wir können nun die Sprache L_d definieren und zeigen, dass diese nicht aufzählbar ist. Der Index d steht dabei für diagonal. Warum wird im Beweis deutlich.

Satz 4.4.4. *Die Sprache*

$$L_d := \{w_i \mid w_i \notin L(M_i)\}$$

ist nicht aufzählbar.

Beweis. Man kann die Wörter und die Turingmaschinen in einer Matrix anordnen:

	M_1	M_2	M_3	M_4	$\ldots$	L_d
w_1	1	1	0	0		$Nein(w_1 \notin L_d)$
w_2	1	0	0	0	$\ldots$	$Ja(w_2 \in L_d)$
w_3	0	1	1	1	$\ldots$	$Nein(w_3 \notin L_d)$
w_4	0	0	1	0		$Ja(w_4 \in L_d)$
.	.	.	.	.	.	.

Eine 1 bei z.B. w_2 und M_1 bedeutet, dass die Turingmaschine M_1 das Wort w_2 akzeptiert, während eine 0 bei z.B. w_2 und M_2 bedeutet, dass M_2 das Wort

w_2 nicht akzeptiert. Es ist nicht wichtig, ob die Turingmaschinen M_1 und M_2 anhalten oder ähnliches. Wichtig ist nur, dass M_1 und M_2 Turingmaschinen sind und als solche eine Sprache akzeptiert. Ein Wort ist also entweder in dieser Sprache enthalten oder nicht. Dies wird durch die Matrix ausgedrückt. Es wird nicht behauptet, dass dies gerade berechnet wird oder ähnliches.

Die Sprache L_d wird nun gerade aus der Diagonalen in obiger Matrix gewonnen, wobei nur die Einträge mit 0 in L_d aufgenommen werden, d.h. ein Wort w_i ist genau dann in L_d enthalten, wenn M_i das Wort w_i *nicht* akzeptiert.

Die Kernidee des Beweises ist nun, dass eine Turingmaschine, die L_d akzeptiert, ein M_j in der Matrix sein müsste. Dies wird dann aber zu einem Widerspruch mit dem Wort w_j führen, dass gleichzeitig enthalten und nicht enthalten sein müsste.

Wir führen dies genauer aus. Angenommen L_d wäre aufzählbar, dann gibt es eine Turingmaschine M_j aus der Auflistung mit $L(M_j) = L_d$, denn in der Auflistung sind ja alle Turingmaschinen enthalten. Betrachten wir nun das Wort w_j. Es muss entweder $w_j \in L_d$ oder $w_j \notin L_d$ sein. Beide Fälle führen aber zu einem Widerspruch:

1. Ist $w_j \in L_d$, dann ist nach der Definition von L_d also $w_j \notin L(M_j)$. Da aber $L(M_j) = L_d$ heißt dies, dass $w_j \notin L_d$ sein müsste. Ein Widerspruch zu $w_j \in L_d$, wovon wir ausgegangen waren.

2. Ist $w_j \notin L_d$, dann ist wegen $L(M_j) = L_d$ also $w_j \notin L(M_j)$. Nach der Definition von L_d ist dann aber $w_j \in L_d$ und wir haben wieder einen Widerspruch, denn wir waren ja von $w_j \notin L_d$ ausgegangen.

Daher kann es keine Turingmaschine geben mit $L(M_j) = L_d$ und damit ist L_d nicht aufzählbar. □

Damit haben wir eine Sprache kennengelernt, die nicht einmal mehr aufzählbar ist. Zumindest ist L_d aber abzählbar. Nicht einmal mehr abzählbar sind dann z.B. die reellen Zahlen.

4.4.4 Zusammenfassung und Ausblick

Wir haben in diesem Abschnitt gesehen, dass es unentscheidbare Probleme gibt und mehrere kennengelernt. Ferner haben wir noch eine Sprache gesehen, die nicht einmal mehr aufzählbar ist. Unser Wissen bezüglich unserer vier bisher eingeführten Sprachfamilien REG, CF, REC und RE stellt sich nun so dar

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

Eine typische Sprache in REG ist z.B. die durch den regulären Ausdruck a^* beschriebene. Eine typische Sprache in CF, die nicht mehr in REG ist, ist z.B. $\{a^n b^n \mid n \in \mathbb{N}\}$. Eine entscheidbare Sprache, die nicht mehr kontextfrei ist,

ist z.B. $\{a^n b^n c^n \mid n \in \mathbb{N}\}$ und eine typische unentscheidbare, aber aufzählbare Sprache ist durch das Halteproblem gegeben. Nicht einmal mehr aufzählbar ist dann die Sprache L_d.

Die entscheidbaren Sprachen sind aus der Sicht der Informatik besonders interessant, da sie einer Lösung durch einen Algorithmus zugänglich sind. Bei den entscheidbaren Sprachen geht es allerdings zunächst nur darum, dass sie überhaupt lösbar sind. Für die Praxis genügt dies oft nicht. Hier sind noch die benötigten Ressourcen zur Lösung des Problems von Bedeutung. Insbesondere fragt man, wie viel Zeit und wie viel Speicher zur Lösung eines Problemes benötigt wird. Es kann passieren, dass ein Problem zwar lösbar ist, dass aber zuviel Zeit benötigt wird und dies daher aus praktischer Sicht nicht umsetzbar ist. Diese Fragen führen in den Bereich der Komplexitätstheorie, die versucht Aussagen über den benötigten Zeit- und Platzbedarf bei der Lösung eines Problems zu machen und einen Begriff von "praktisch nicht lösbar" zu etablieren. Wir kommen darauf im späteren Verlauf noch zu sprechen.

Fragen

1. Wenn zwei Sprachen L_1, L_2 entscheidbar sind, ist dann auch $L_1 \cap L_2$ entscheidbar?
   a) Ja!
   b) Nein!
2. Wenn eine Sprache L unentscheidbar ist, gibt es dann eine Turingmaschine M mit $L(M) = L$?
   a) Ja!
   b) Nein!
   c) Manchmal!
3. Welche der folgenden Sprachen sind unentscheidbar?
   a) TM_{acc}
   b) TM_{halt}
   c) L_d
4. Welche der folgenden Sprachen sind aufzählbar?
   a) TM_{acc}
   b) TM_{halt}
   c) L_d