Prof. Dr. André Schulz

Modul 31321
Grundlagen der Informatik

01657 Grundlagen der Theoretischen Informatik A
01658 Grundlagen der Theoretischen Informatik B

LESEPROBE

mathematik und informatik

FernUniversität in Hagen

Prof. Dr. André Schulz

Kurs 01657
Grundlagen der Theoretischen Informatik A

LESEPROBE

mathematik und informatik

FernUniversität in Hagen

Kurseinheit 2

Reguläre Sprachen I: Endliche Automaten und Reguläre Ausdrücke

In dieser Kurseinheit lernen wir mit den endlichen Automaten ein erstes Berechnungsmodell kennen. Dieses Modell orientiert sich am Rechnen mit begrenztem Speicher. Wir werden sehen, dass wir nur sehr einfache Probleme mit einem endlichen Automaten lösen können. Trotzdem ist der endliche Automat ein sehr wichtiges Modell, denn er bildet die Grundlage für weitere Modelle und findet als Modellierungswerkzug in der gesamten Informatik vielfältige Anwendungsmöglichkeiten.

2.1 Der Deterministische Endliche Automat

Wir beginnen mit einer informellen Beschreibung des Modells des endlichen Automaten. Genauer gesagt, handelt es sich hierbei um das Modell des *deterministischen* endlichen Automaten, den wir kurz DEA nennen (oder auch nur Automat, wenn keine Gefahr der Verwechslung zu anderen Modellen besteht). Mit einem DEA können wir das Wortproblem von bestimmten formalen Sprachen lösen (in der letzten Kurseinheit wurde besprochen, dass alle Entscheidungsprobleme als Wortprobleme formuliert werden können). In diesem Sinne verarbeitet der Automat ein Wort (die Eingabe) und gibt uns dann die Antwort, ob das Wort aus der zugehörigen Sprache ist, oder nicht. Wir sagen in diesem Zusammenhang auch, dass der DEA das Eingabewort **akzeptiert** (wenn er das Wort der Sprache zuordnet) oder **verwirft** (wenn er das Wort als nicht zur Sprache gehörig einsortiert). Um zu einer Antwort zu gelangen, *liest* der DEA das Anfragewort zeichenweise ein. Dabei kann er immer nur auf ein Zeichen der Eingabe zugreifen. Es ist ihm zudem nicht erlaubt, bereits gelesene Zeichen der Eingabe wieder anzufragen. Ein DEA hat nur beschränkten (konstanten) Speicher. Das heißt, während der Verarbeitung kann der DEA einen von endlich vielen *Zuständen* annehmen. Die eigentliche Berechnung wird dadurch festgelegt, wie man von einem Zustand in einen anderen Zustand gelangt. Dieser *Zustandsübergang* hängt vom aktuellen Zeichen der Eingabe ab. Am Ende, nachdem das letzte Zeichen der Eingabe gelesen wurde, können wir entscheiden, ob das Anfragewort aus der Sprache des DEAs ist. Diese Entscheidung wird vom Zustand abhängen, in dem der Automat sich am Ende

befindet.

Wir führen nun eine formale Definition des mathematischen Modells des deterministischen endlichen Automaten ein.

Definition 2.1 — Deterministischer Endlicher Automat.
Ein **deterministischer endlicher Automat (DEA)** M wird durch ein Tupel $(Q, \Sigma, \delta, q_0, F)$ dargestellt. Hierbei ist

- Q eine endliche nicht-leere Menge, genannt **Zustandsmenge**,
- Σ ein (endliches) Alphabet,
- δ eine Funktion $\delta\colon Q \times \Sigma \to Q$, genannt **Übergangsfunktion**,
- q_0 ein Element aus Q, genannt **Startzustand**,
- F eine Teilmenge von Q, genannt Menge der **akzeptierenden Zustände**.

Beispiel 2.1 Das folgende Quintupel $M_1 = (Q, \Sigma, \delta, q_0, F)$ gibt einen DEA an. Wir setzen hierbei $Q = \{q_0, q_1\}$, $\Sigma = \{\mathsf{a}, \mathsf{b}\}$ und $F = \{q_0\}$. Die Übergangsfunktion δ geben wir durch eine Tabelle an.

$q \in Q$	$x \in \Sigma$	$\delta(q, x)$
q_0	a	q_1
q_0	b	q_0
q_1	a	q_0
q_1	b	q_1

An dieser Stelle soll darauf hingewiesen werden, dass wir zwischen dem *Modell* des deterministischen endlichen Automaten und konkreten *Realisierungen* in diesem Modell, wie etwa in Beispiel 2.2 angegeben, unterscheiden. Es hat sich aber eingebürgert sowohl das Modell, als auch die Realisierungen, beides als deterministischen endlichen Automaten zu bezeichnen. Die Bedeutung ergibt sich aus dem Kontext. Trotzdem sollten Sie sich dieser Unterscheidung bewusst sein. Gleiches gilt auch für andere Modelle, die wir noch später im Kurs vorstellen werden (Kellerautomat, kontextfreie Grammatik, Turingmaschine).

Häufig werden wir eine graphische Notation namens **Zustandsdiagramm** benutzen, um einen DEA anzugeben. Aus dieser Darstellung lassen sich alle Bestandteile des Automaten leicht ablesen. Zustände werden wir als Kreise darstellen (in Ausnahmefällen als Rechtecke), die mit dem Zustand (in der Mitte) beschriftet sind. Akzeptierende Zustände heben wir zusätzlich hervor, indem wir deren Kreise mit einer doppelten Linie zeichnen. Falls $\delta(q, x) = p$, vermerken wir das, indem wir einen Pfeil einfügen, der den Zustand q mit dem Zustand p verbindet (Pfeil zeigt in Richtung p). Diesen Pfeil beschriften wir zusätzlich mit x. Es verbleibt, den Startzustand zu kennzeichnen. Dies realisieren wir, indem wir einen kleinen Pfeil an diesen Zustand anbringen. Der Pfeil zeigt auf den Startzustand, und sein Anfangspunkt ist mit keinem Zustand verbunden. Die Abbildung 2.1 zeigt noch einmal die Grundelemente der graphischen Darstellung. Das Zustandsdiagramm des DEAs aus dem Bespiel 2.1 ist in Abbildung 2.2 zu sehen.

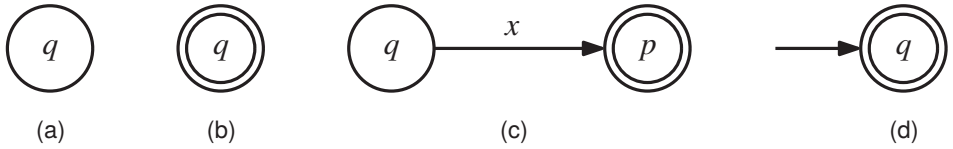

Abbildung 2.1: Bestandteile der graphischen Notation eines DEAs: (a) verwerfender Zustand, (b) akzeptierender Zustand, (c) Zustandsübergang, (d) Startzustand.

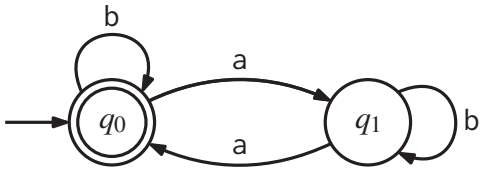

Abbildung 2.2: Zustandsdiagramm des DEAs M_1 aus Beispiel 2.1.

Die Übergangsfunktion ist das „Herzstück“ des DEAs. Es handelt sich hierbei um eine Funktion, deren Definitionsbereich Paare bestehend aus einem Zustand und einem Zeichen sind. Sie gibt also für einen Zustand und ein Zeichen einen neuen Zustand an. Diesen Zustand bezeichnen wir als **Folgezustand**. Wie bereits beschreiben, befindet sich der Automat während der Berechnung immer in einem Zustand. Zu Beginn der Berechnung ist dies der Startzustand. Während der Berechnung liest er die Eingabe Zeichen für Zeichen und gleicht seinen Zustand ab. Dazu nutzt er die Übergangsfunktion δ. Wenn q den aktuellen Zustand bezeichnet und x das nächste Zeichen der Eingabe ist, dann gibt $\delta(q, x)$ den Folgezustand an. Nachdem alle Zeichen der Eingabe gelesen wurden, befindet sich der DEA in einem Zustand. Ist dieser Zustand ein akzeptierender Zustand, wird das Eingabewort akzeptiert, ansonsten verworfen. Die Folge der Zustände, die der Automat während der Berechnung angenommen hat, bezeichnen wir als seinen **Lauf** für die gewählte Eingabe. Wir sprechen auch von einem w-Lauf, wenn der Lauf sich auf die Eingabe w bezieht. Ein Lauf ist eine **akzeptierender Lauf**, wenn er in einem akzeptierenden Zustand endet, ansonsten nennen wir ihn **verwerfenden Lauf**. Alle Zustände, die man im Zustandsdiagramm (als gerichteter Graph interpretiert) vom Startzustand erreichen kann, nennen wir **erreichbare Zustände**. Die *nicht-erreichbaren* Zustände spielen für die Akzeptanz eines Wortes keine Rolle und können immer entfernt werden.

Die Menge aller Wörter, die der Automat akzeptiert, nennen wir die **Sprache des Automaten** oder auch die vom Automaten akzeptierte Sprache. Die Sprachen eines Automaten M notieren wir mit $L(M)$.

In Abbildung 2.3 ist ein Berechnungsablauf des Automaten M_1 aus Beispiel 2.1 exemplarisch für das Wort aba dargestellt. Man kann sich für dieses Beispiel recht leicht überlegen, welche Wörter von diesem DEA akzeptiert werden. Wir erkennen, dass es zwei Zustände q_0 und q_1 gibt, von welchen nur q_0 akzeptierend ist. Wenn ein Zeichen b von der Eingabe gelesen wird, ist der Folgezustand gleich dem ursprünglichen Zustand. Deshalb hängt es nur von den Zeichen a ab, ob ein Wort akzeptiert wird. Genauer gesagt, hängt es von der Anzahl der Zeichen a ab, da a das einzig relevante Zeichen ist. Wenn ein Zeichen a gelesen wird, wird der Zustand gewechselt, und zwar von akzeptierend zu nicht-akzeptierend, oder umgekehrt. Das bedeutet, dass es von der Parität (gerade/ungerade) der Anzahl der Zeichen

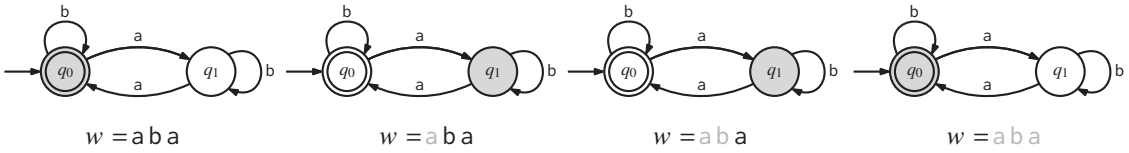

Abbildung 2.3: Ablauf der Berechnung von M_1 für die Eingabe $w = \mathsf{aba}$. Der aktuelle Zustand ist grau hinterlegt. Bereits gelesene Zeichen der Eingabe sind ebenfalls grau. Der Lauf des Automaten ist (q_0, q_1, q_1, q_0). Da der Lauf in einem akzeptierenden Zustand endet, wird die Eingabe aba akzeptiert.

a abhängt, ob die Eingabe akzeptiert wird. Wir sehen also, dass für dieses Beispiel

$$L(M_1) = \{w \in \{\mathsf{a}, \mathsf{b}\}^* \mid w \text{ enthält gerade Anzahl von as}\}$$

gilt.

Bevor wir den *Akzeptanzbegriff* des DEAs formal beschreiben, werden wir noch eine hilfreiche Notation einführen. Die Übergangsfunktion δ erlaubt uns den Folgezustand zu bestimmen, wenn wir ein Zeichen von der Eingabe gelesen haben. Oft ist es aber nützlich, den „Folgezustand“ zu beschreiben, wenn man statt eines Zeichens ein Wort liest. Dafür nutzen wir die **iterierte Übergangsfunktion** δ^*, welche direkt aus δ abgeleitet werden kann.

Definition 2.2 — Iterierte Übergangsfunktion eines DEAs.
Sei δ die Übergangsfunktion eines DEAs, dann definieren wir für alle $q \in Q$

$$\delta^0(q, \varepsilon) = q,$$

und für alle $i > 0$ und alle Wörter $w = ua \in \Sigma^i$ mit $u \in \Sigma^{i-1}$ und $a \in \Sigma$

$$\delta^i(q, w) = \delta(\delta^{i-1}(q, u), a).$$

Schließlich definieren wir die **iterierte Übergangsfunktion** $\delta^* \colon Q \times \Sigma^* \to Q$ als

$$\delta^*(q, w) := \delta^{|w|}(q, w).$$

Für den Automaten M_1 aus Beispiel 2.1 ergibt sich beispielsweise $\delta^*(q_0, \mathsf{aba}) = q_0$ und $\delta^*(q_1, \mathsf{aa}) = q_1$. Mit der iterierten Übergangsfunktion können wir nun kompakt die von einem DEA erkannte Sprache definieren.

Definition 2.3 — Sprache eines DEAs.
Sei $M = (Q, \Sigma, \delta, q_0, F)$ ein DEA, dann ist die von M akzeptierte Sprache definiert als

$$L(M) := \{w \in \Sigma^* \mid \delta^*(q_0, w) \in F\}.$$

Die Sprachen die von einem DEA erkannt werden, haben viele nützliche Eigenschaften und bilden eine interessante Struktur. Aus diesem Grunde geben wir dieser Sprachfamilie einen Namen.

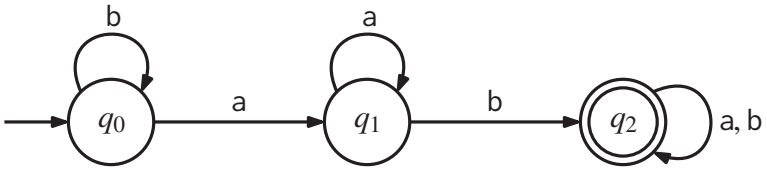

Abbildung 2.4: Der DEA M_2 zu Beispiel 2.2.

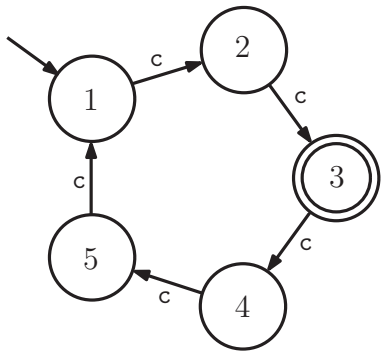

Abbildung 2.5: Der DEA M_3 zu Beispiel 2.3.

Definition 2.4 — Reguläre Sprache.
Wenn L eine Sprache ist, für die es einen DEA gibt, der L akzeptiert, nennen wir L eine **reguläre Sprache**. Wir nutzen die Bezeichnung

$$REG := \{L \mid L \text{ ist regulär}\}.$$

An dieser Stelle wollen wir noch zwei Beispiele besprechen.

Beispiel 2.2 Der DEA M_2 ist durch das Zustandsdiagramm in Abbildung 2.4 gegeben. Wir erkennen, dass es mit q_2 nur einen akzeptierenden Zustand gibt. Wenn q_2 während der Berechnung angenommen wird, verbleibt der DEA in diesem Zustand. Um nach q_2 zu gelangen, müssen wir vorher in q_1 sein, und das nächste zu lesende Zeichen muss ein b sein. Man befindet sich aber genau dann in q_1 (ohne vorher schon in q_2 zu sein), wenn als letztes Zeichen ein a gelesen wurde. Also akzeptiert M_2 alle Wörter, die als Teilwort ab erhalten. Das heißt

$$L(M_2) = \{w \in \{\mathsf{ab}\}^* \mid \mathsf{ab} \text{ ist Teilwort von } w\}.$$

Das Beispiel 2.2 gibt die erste praktische Anwendung für unser Berechnungsmodell. Die meisten Beispiele für reguläre Sprachen wirken sehr künstlich. Eigentlich gehen wir ja davon aus, dass es sich bei diesen Sprachen um Kodierungen der Ja-Instanzen von Entscheidungsproblemen handelt. Die Sprache $L(M_2)$ ist in dieser Beziehung interessant. Das zugrundeliegende Entscheidungsproblem fragt, ob ein Wort das Teilwort ab enthält. Es ist nicht schwer, den DEA umzuwandeln, sodass wir nach anderen Teilwörtern fragen können. Die Frage, ob ein Text ein Teilwort enthält, hat eine hohe praktische Relevanz (*pattern matching*). Viele Algorithmen zum Suchen von Wörtern in Texten benutzen endliche Automaten als Hilfsmittel. Zum Beispiel nutzt das Kommandozeilenprogramm *grep* einen solchen Ansatz.

Beispiel 2.3 Sei $M_3 = (\{1,2,3,4,5\}, \{\mathsf{c}\}, \delta, 1, \{3\})$ mit

$$\delta(x, \mathsf{c}) = \begin{cases} x+1 & \text{falls } x \neq 5 \\ 1 & \text{sonst.} \end{cases}$$

Das Zustandsdiagramm des Automaten ist in Abbildung 2.5 zu sehen. Wir erkennen, dass wir immer genau dann im Zustand 1 sind, wenn wir eine Anzahl von cs gelesen haben, die ein Vielfaches von 5 ist. Demnach akzeptiert M_3 genau die Wörter w mit $|w| \bmod 5 = 2$. Somit gilt

$$L(M_3) = \{\mathsf{c}^k \mid 5 \text{ teilt } k \text{ mit Rest } 2\}.$$

Test 2.1

Entwerfen Sie einen DEA, der die folgende Sprache akzeptiert:

$$L = \{w \in \{\mathtt{0},\mathtt{1}\}^+ \mid w \text{ hat unterschiedliches Anfangs- und Endzeichen}\}$$

Wir wollen nun einen ersten Satz zu den regulären Sprachen beweisen. Hierbei geht es um die Beziehung zu einer anderen Sprachklasse — den endlichen Sprachen. Wir nennen eine Sprache **endlich**, wenn sie nur endlich viele Wörter besitzt.

Satz 2.1
Jede endliche Sprache ist regulär.

Beweis. Sei $L \subseteq \Sigma^*$ eine endliche Sprache deren längstes Wort die Länge ℓ hat. Um zu zeigen, dass L regulär ist, müssen wir einen DEA M für L angeben. Wir nehmen vorerst an, dass $\Sigma = \{\mathsf{a}, \mathsf{b}\}$. Wir beschreiben M, indem wir sein Zustandsdiagramm angeben. In der Grundstruktur entspricht das Diagramm einem binären Baum der Tiefe ℓ. Von einem inneren Knoten gibt es zwei Kanten zu seinen Kindern. Eine dieser Kanten beschriften wir mit a und die andere mit b. Wir orientieren nun alle Kanten vom Vater zum Kind. Als Startzustand wählen wir die Wurzel des Baumes. Es gibt für jeden Knoten genau einen Pfad von der Wurzel. Wir benennen einen Zustand mit q_w, wenn w das Wort ist, was man lesen muss, um ihn zu erreichen. Nun machen wir genau die Zustände q_w zu akzeptierenden Zuständen, für die w ein Wort aus der Sprache L ist. Abschließend führen wir noch einen Müllzustand q_- ein (nicht-akzeptierend). Alle Übergänge die nun noch fehlen, gehen zum Müllzustand über. Abbildung 2.6 zeigt diese Konstruktion am Beispiel.

Es ist nun nicht schwer zu argumentieren, dass $M(L) = L$. Jedes Wort der Länge größer ℓ führt nach q_- und wird verworfen. Jedes andere Wort w führt zum Zustand q_w. Ist $w \in L$, dann ist $q_w \in F$ und wir akzeptieren w. Alle anderen Wörter werden verworfen.

Bei anderen Alphabeten erfolgt die Konstruktion analog. Statt eines binären Baumes nutzt man einen k-ären Baum, wobei $k = |\Sigma|$. ■

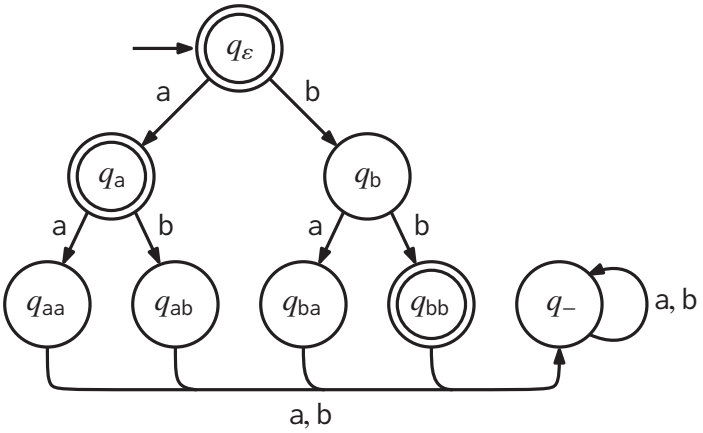

Abbildung 2.6: Konstruktion zum Beweis von Satz 2.1 für die Sprache $\{\varepsilon, \mathsf{a}, \mathsf{bb}\}$.

2.2 Nichtdeterministische Endliche Automaten

Als nächstes werden wir ein neues Berechnungsmodell einführen, welches sich an der Arbeitsweise von deterministischen endlichen Automaten anlehnt. Dies ist der sogenannte **nichtdeterministische endliche Automat (NEA)**. Wie auch der DEA arbeitet der NEA mit Zuständen, welche akzeptierend oder verwerfend sein können. Auch der NEA verarbeitet das Eingabewort zeichenweise und kann nicht auf bereits gelesene Zeichen direkt wieder zurückgreifen. Genau wie beim DEA gibt es auch eine Übergangsfunktion – diese weist jedoch jedem Zustand und Eingabezeichen nicht einen einzelnen Folgezustand zu, sondern eine Menge von Folgezuständen. In diesem Sinne gibt es nicht nur einen Lauf für jede Eingabe, sondern mitunter mehrere mögliche Läufe.

Es stellt sich natürlich die Frage, wie man damit umgeht, dass der mögliche Folgezustand nicht mehr eindeutig festgelegt ist. So könnte es durchaus sein, das bei ein und demselben Eingabewort ein Lauf in einem akzeptierenden Zustand endet, ein anderer Lauf aber in einem verwerfenden Zustand. Es ist also nicht offensichtlich, wie der Akzeptanzbegriff für NEAs gefasst ist. Unser Kriterium für die Akzeptanz eines Wortes wird anschaulich das folgende sein: **Existiert ein Lauf** vom Startzustand zu einem akzeptierenden Zustand, wird das Eingabewort akzeptiert. Dieser Akzeptanzbegriff scheint auf den ersten Blick künstlich, denn diese Art von Berechnung widerspricht unserem intuitiven Verständnis vom maschinellen Berechnen. Es wird sich jedoch zeigen, dass das Berechnungsmodell NEA seine Berechtigung hat. Durch die Nutzung des Nichtdeterminismus lassen sich viele Probleme leichter modellieren. Zusätzlich können wir durch die Verwendung von NEAs gegenüber von DEAs viele Beweise von Sätzen über reguläre Sprachen vereinfachen.

Es gibt noch einen weiteren Unterschied zwischen NEA und DEA. Bei einem DEA kann ein Zustandswechsel nur dann geschehen, wenn ein Zeichen von der Eingabe gelesen wurde. Wir erlauben beim NEA auch, den Zustand zu wechseln ohne dabei ein Zeichen zu lesen. Diese Übergänge sollen natürlich nicht beliebig stattfinden. Deshalb definieren wir sogenannte **ε-Übergänge** zwischen Zuständen. Ist ein ε-Übergang zwischen Zustand p und q vorhanden, kann man vom Zustand p in den Zustand q wechseln, ohne ein Zeichen der Eingabe zu lesen. Im Zustandsdiagramm werden solche Übergänge wie normale Übergänge eingezeichnet, statt eines Zeichen aus dem Alphabet werden sie jedoch mit ε beschriftet.

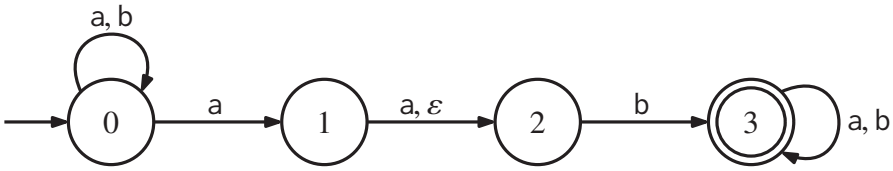

Abbildung 2.7: Zustandsdiagramm vom NEA N_1.

Bevor wir die NEAs formal definieren, erklären wir die prinzipielle Arbeitsweise eines NEAs am Beispiel. Sehen wir uns das Zustandsdiagramm von NEA N_1 in Abbildung 2.7 an. Wir erkennen an folgenden Merkmalen, dass es sich um das Diagramm eines NEAs handelt. Es gibt nicht immer genau einen möglichen Folgezustand. Zum Beispiel ist es möglich, vom Zustand 0 mit einem a sowohl zum Zustand 1 zu gelangen, als auch im Zustand 0 zu bleiben. Des Weiteren können wir beobachten, dass es vom Zustand 1 keinen möglichen Folgezustand gibt, den man mit einem b erreichen kann. Die Menge der Folgezustände kann also auch die leere Menge sein. Außerdem erkennen wir, dass der Automat einen ε-Übergang zwischen Zustand 1 und 2 aufweist. In diesem Automaten können wir vom Zustand 0 zum Zustand 3 gelangen, indem wir aab lesen. In diesem Sinne gibt es einen akzeptierenden Lauf für das Wort aab. Man kann aber auch erkennen, dass man mit demselben Wort auch einen Lauf realisieren kann, der die ganze Zeit im Zustand 0 verweilt. Ein anderes Wort mit einem akzeptierenden Lauf ist das Wort ab. Hier können wir von Zustand 0 zu Zustand 1 wechseln, indem wir ein a lesen, dann nutzen wir den ε-Übergang, um in den Zustand 3 zu gelangen, und anschließend können wir durch das Lesen des Zeichen b in den akzeptierenden Zustand 3 wechseln.

Wir werden nun das Modell NEA und den damit verbundenen Akzeptanzbegriff formal definieren. An dieser Stelle sei noch einmal daran erinnert, dass man mit der Potenzmenge $\mathcal{P}(X)$ die Menge aller Teilmengen von X bezeichnet, also $\mathcal{P}(X) := \{Y \subseteq X\}$.

Definition 2.5 — Nichtdeterministischer Endlicher Automat.
Ein **nichtdeterministischer endlicher Automat (NEA)** M wird durch ein Tupel $(Q, \Sigma, \delta, q_0, F)$ dargestellt. Hierbei ist

- Q eine endliche nicht-leere Menge, genannt **Zustandsmenge**,
- Σ ein (endliches) Alphabet,
- δ eine Funktion $\delta\colon Q \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q)$, genannt **Übergangsfunktion**,
- q_0 ein Element aus Q, genannt **Startzustand**,
- F eine Teilmenge von Q, genannt Menge der **akzeptierenden Zustände**.

Als nächstes werden wir die iterierte Übergangsfunktion eines NEAs aus seiner Übergangsfunktion ableiten. Die iterierte Übergangsfunktion soll uns angeben, in welchem Zustand man nach dem Lesen eines Wortes *sein könnte*. Für die Einbeziehung der ε-Übergänge benötigen wir noch eine Definition. Wir wollen ausdrücken können, welche Zustände wir von p aus erreichen können, ohne ein Zeichen zu lesen. Diese Zustandsmenge notieren wir

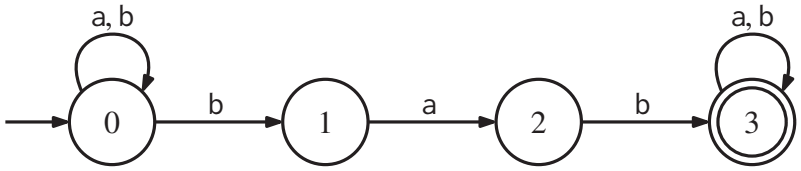

Abbildung 2.8: NEA N_2 für Beispiel 2.4.

mit $E(p)$. Es gilt also

$$E(p) := \{q \mid q \text{ ist von } p \text{ durch eine Sequenz von } \geq 0 \ \varepsilon\text{-Übergängen erreichbar}\}.$$

Für Mengen $P \subseteq Q$ definieren wir

$$E(P) := \bigcup_{p \in P} E(p).$$

Für den NEA N_1 aus Abbildung 2.7 gilt beispielsweise $E(\{0, 1\}) = \{0, 1, 2\}$.

Definition 2.6 — Iterierte Übergangsfunktion eines NEAs.
Sei δ die Übergangsfunktion eines NEAs, dann definieren wir für alle $P \subseteq Q$

$$\delta^0(P, \varepsilon) = E(P),$$

und für alle $i > 0$ und alle Wörter $w = ua \in \Sigma^i$ mit $u \in \Sigma^{i-1}$ und $a \in \Sigma$

$$\delta^i(P, w) = E(\textstyle\bigcup_{r \in \delta^{i-1}(P,u)} \delta(r, a)).$$

Schließlich definieren wir die **iterierte Übergangsfunktion** $\delta^* \colon Q \times \Sigma^* \to \mathcal{P}(Q)$ als

$$\delta^*(P, w) := \delta^{|w|}(P, w).$$

Analog zum DEA können wir die iterierte Übergangsfunktion benutzen, um die Akzeptanz eines Wortes und damit die Sprache eines NEAs zu definieren. Es sollen genau die Worte akzeptiert werden, für die es *möglich ist*, vom Startzustand durch Übergänge zu einem akzeptierenden Zustand zu gelangen.

Definition 2.7 — Sprache eines NEAs.
Sei $N = (Q, \Sigma, \delta, q_0, F)$ ein NEA, dann ist die von N akzeptierte Sprache definiert als

$$L(N) := \{w \in \Sigma^* \mid \delta^*(q_0, w) \cap F \neq \emptyset\}.$$

Beispiel 2.4 Als ein weiteres Beispiel sehen wir uns den in Abbildung 2.8 gezeigten NEA N_2 an. Um in den akzeptieren Zustand 3 zu kommen, muss man vorher das Teilwort bab gelesen haben. Das heißt, es können nur Wörter akzeptiert werden, die bab als Teilwort enthalten. Auf der anderen Seite gibt es für jedes Wort, welches bab als Teilwort enthält, einen akzeptierenden Lauf. Dieser verbleibt im Zustand 0 bis das Teilwort bab beginnt, dann liest er dieses Teilwort und geht dabei in den Zustand 3. Anschließend

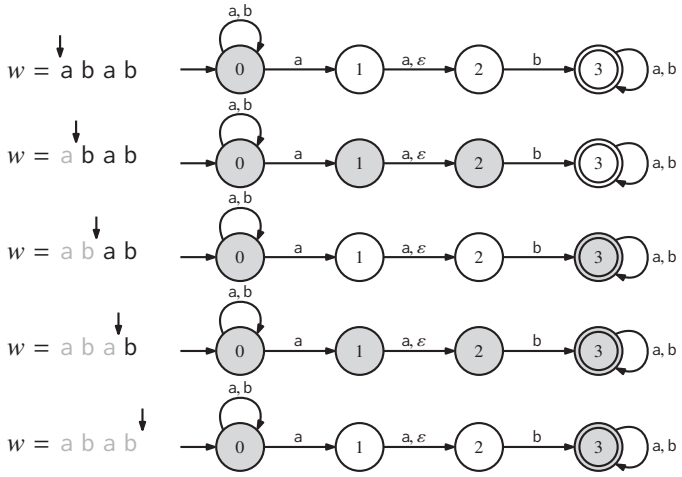

Abbildung 2.9: Mögliche Zustände (grau) des NEA N_1 aus Abbildung 2.7 bei der Verarbeitung des Eingabewortes $w = \mathsf{abab}$.

verbleibt er im Zustand 3. Wir erhalten also

$$L(N_2) := \{w \in \{\mathsf{a}, \mathsf{b}\}^* \mid w \text{ enthält } \mathsf{bab} \text{ als Teilwort}\}.$$

Wir werden uns nun ansehen, wie man (praktisch) überprüfen kann, ob ein NEA ein Wort akzeptiert oder nicht akzeptiert. Sei also ein NEA und ein Wort w gegeben, zum Beispiel der NEA N_1 aus Abbildung 2.7 und das Wort $w = \mathsf{abab}$. Wir wollen herausfinden, in welchen Zuständen man nach dem Lesen von w sein kann, wenn man vom Startzugang ausgeht, und ob einer dieser möglichen Zustände ein akzeptierender Zustand ist (wir wollen also $\delta^*(E(q_0), w) \cap F \neq \emptyset$ auswerten). Dazu werden wir das Wort w Zeichen für Zeichen verarbeiten und uns immer alle möglichen aktuellen Zustände merken. Am Anfang (ohne ein Zeichen zu lesen) können wir nur im Zustand 0 sein. Nach dem Lesen des ersten Zeichens von w (ein a) können wir sowohl im Zustand 0, 1, oder 2 sein, denn hier kommt der Nichtdeterminismus zum Tragen. Nach dem Lesen des nächsten Zeichens b können wir uns im Zustand 0 befinden (von Zustand 0 aus kommend) oder im Zustand 3 (von Zustand 2 aus kommend). Die möglichen Zustände sind also $\{0, 3\}$. Wenn wir das nächste Zeichen a lesen, kommen wir in die Zustände 0,1,2 (von Zustand 0 aus kommend) oder wir verbleiben im Zustand 3. Die Menge der möglichen Zustände ist somit $\{0, 1, 2, 3\}$. Das letzte zu lesende Zeichen ist ein b. Danach können wir uns im Zustand 0 (vom Zustand 0 aus kommend) oder im Zustand 3 (vom Zustand 2 oder 3 aus kommend) befinden. Also sind die möglichen Zustände nach dem Lesen von w gleich $\{0, 3\}$. Da in dieser Menge mit Zustand 3 ein akzeptierender Zustand enthalten ist, akzeptieren wir w. Die möglichen Zustände für dieses Beispiel sind in Abbildung 2.9 dargestellt.

Als nächstes wollen wir die Frage diskutieren, ob ein NEA mehr Sprachen erkennen kann als ein DEA. Es ist klar, dass es für jede reguläre Sprache einen NEA gibt, der diese akzeptiert, denn jeder DEA ist ein NEA, der den Nichtdeterminismus und die ε-Übergänge nicht verwendet. Wir werden aber auch zeigen, dass jede Sprache die ein NEA akzeptiert, auch von einem DEA akzeptiert wird. Dazu werden wir eine Konstruktion vorstellen, die aus einem NEA einen DEA konstruiert, der die gleiche Sprache akzeptiert. Diesen DEA nennen wir **Potenzautomat**.

Prof. Dr. André Schulz

Kurs 01658
Grundlagen der Theoretischen Informatik B

LESEPROBE

mathematik
und
informatik

FernUniversität in Hagen

Kurseinheit 1

Unentscheidbare Sprachen

Im Kursteil A haben wir verschiedene Rechenmodelle untersucht. Dies waren unter anderem der endliche Automat, der Kellerautomat und die Turingmaschine. Jedem dieser Modelle haben wir eine (oder auch mehrere) Sprachklassen zugeordnet: Dem endlichen Automaten die regulären Sprachen, dem Kellerautomaten die kontextfreien Sprachen und der Turingmaschine die entscheidbaren und erkennbaren/aufzählbaren[1] Sprachen. Bei der Untersuchung dieser Sprachklassen haben wir festgestellt, dass die regulären Sprachen eine echte Teilmenge der kontextfreien Sprachen sind, und diese wiederum eine echte Teilmenge der entscheidbaren Sprachen. Der Fokus im Teil A lag darauf, die „Berechenbarkeit“ von Sprachen (bzw. Entscheidungsproblemen) nachzuweisen. Am Anfang des Teils B werden wir uns im Gegensatz dazu auf den Nachweis der „Nichtberechenbarkeit“ konzentrieren. Wir versuchen zudem, die Grenze zwischen Nichtberechenbaren und Berechenbaren gut zu bestimmen.

Wir wissen bereits, dass es mit $\{\mathtt{a}^n\mathtt{b}^n \mid n \geq 0\}$ eine nichtreguläre und mit $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n \mid n \geq 0\}$ eine nichtkontextfreie Sprache gibt. Um dies zu zeigen, hatten wir als Werkzeuge das Pumpinglemma (kontextfrei/regulär) und den Satz von Myhill–Nerode zur Verfügung. Wir wissen noch nicht, ob es Sprachen gibt, welche nicht entscheidbar oder nicht aufzählbar sind. Aus der Definition der aufzählbaren und der entscheidbaren Sprachen folgt zwar direkt, dass $\mathbb{E} \subseteq \mathbb{A}$, doch bislang haben wir noch nicht untersucht, ob dies eine *echte* Teilmengenbeziehung ist.

1.1 Existenz von nichtaufzählbaren Sprachen

Wir beginnen unsere Untersuchung mit einer Existenzaussage. Wir wollen beweisen, dass es weniger Turingmaschinen gibt als Sprachen über dem Alphabet $\{\mathtt{0}, \mathtt{1}\}$. Da es ja maximal so viele aufzählbare (also erkennbare) Sprachen geben kann, wie es Turingmaschinen gibt, würde daraus folgen, dass es Sprachen gibt, die nicht aufzählbar sind. Nun ist es jedoch so, dass es ja unendlich viele Turingmaschinen gibt und auch unendlich viele Sprachen. Wir müssen also eine Methode erarbeiten, mit welcher wir die „Größen“ von unendlichen Mengen in sinnvoller Weise miteinander vergleichen können. Um solche

[1] Beachten Sie, dass wir die Begriffe *aufzählbar* und *erkennbar* synonym benutzen, siehe dazu auch Kursteil A.

Vergleiche vornehmen zu können, definieren wir den Begriff der **Gleichmächtigkeit**.

Definition 1.1
Zwei Mengen X und Y heißen *gleichmächtig*, falls eine Bijektion zwischen X und Y existiert.

Ein einfaches Beispiel zweier gleichmächtiger Mengen sind die Mengen $X_1 = \{\mathsf{a}, \mathsf{b}\}$ und $X_2 = \{\mathtt{0}, \mathtt{1}\}$, denn wir können eine Funktion $f_1 \colon X_1 \to X_2$ angeben, die eine Bijektion ist; zum Beispiel:

$$f_1(x) := \begin{cases} \mathtt{0} & \text{falls } x = \mathsf{a} \\ \mathtt{1} & \text{falls } x = \mathsf{b}. \end{cases}$$

Es liegt auf der Hand, dass zwei endliche Mengen genau dann gleichmächtig sind, wenn sie die gleiche Kardinalität haben. Aber auch unendliche Mengen können gleichmächtig sein. Ein einfaches Beispiel hierfür sind die Mengen $Y_1 = \{i \in \mathbb{Z} \mid i \geq 0\}$ und $Y_2 = \{i \in \mathbb{Z} \mid i \geq 1\}$. Zwischen diesen Mengen können wir die folgende Bijektion $f_2 \colon Y_1 \to Y_2$ angeben:

$$f_2(x) := x + 1$$

Wir sehen, dass f_2 keine zwei Zahlen auf eine gemeinsame Zahl abbildet, damit ist f_2 injektiv. Des Weiteren gibt es für jedes $y \in Y_2$ mit $y' = y - 1$ eine Zahl, sodass $f_2(y') = y$. Daraus folgt, dass f_2 surjektiv ist, und demnach eine Bijektion darstellt.

Test 1.1
Zeigen Sie, dass die Mengen $Z_1 = \{i \in \mathbb{N} \mid i \geq 1\}$ und $Z_2 = \{p \in \mathbb{N} \mid p \text{ ist Primzahl}\}$ gleichmächtig sind.

Von besonderem Interesse ist es, die Mengen zu bestimmen, die gleichmächtig zu den natürlichen Zahlen sind. Wir nehmen an, dass die 0 keine natürliche Zahl ist, alle Argumente des Kurses funktionieren aber genauso gut, wenn wir die 0 als natürliche Zahl zulassen würden.

Definition 1.2 — abzählbar.
Eine Menge X heißt *abzählbar*, falls (i) X endlich ist oder (ii) X und die Menge der natürlichen Zahlen $\mathbb{N}$ gleichmächtig sind.

Aus unseren bisherigen Beobachtungen folgt, dass zum Beispiel die Menge $\{p \in \mathbb{N} \mid p \text{ ist Primzahl}\}$ abzählbar ist. Wenn eine Menge X abzählbar ist, heißt dies also, dass wir alle ihre Elemente mit den natürlichen Zahlen durchnummerieren können. Eine Bijektion $f \colon \mathbb{N} \to X$ beschreibt die Vergabe der Nummern. Wir bezeichnen die dazugehörige Folge $f(1), f(2), f(3), \ldots$ auch als eine **Nummerierung** der Menge X. Beachten Sie aber, dass *abzählbar* und *aufzählbar* zwei verschiedene Eigenschaften sind.

Test 1.2
Zeigen Sie, dass wenn X abzählbar, dann ist auch jede Teilmenge $Y \subseteq X$ abzählbar.

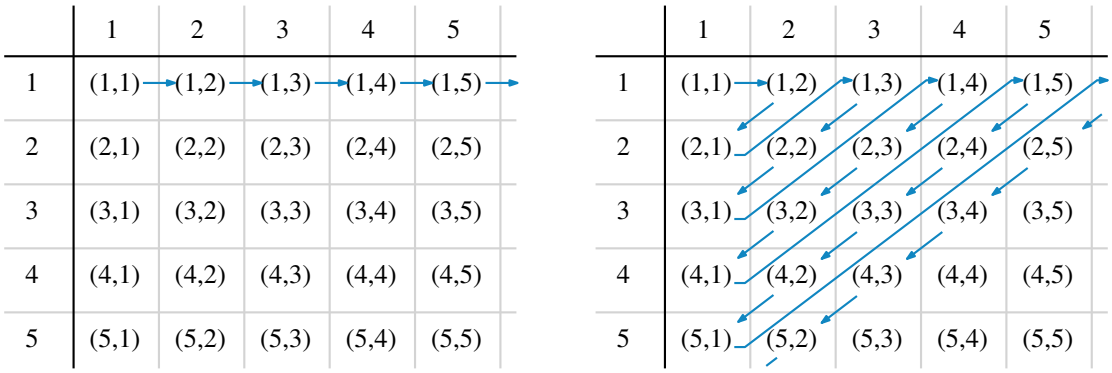

Abbildung 1.1: Strategie beim Durchnummerieren der Einträge einer Tabelle: Zeilenweises Abzählen (links), Cantornummerierung (rechts).

Lemma 1.1
Die Menge der ganzen Zahlen $\mathbb{Z}$ ist abzählbar.

Beweis. Wir geben eine Funktion $f : \mathbb{Z} \to \mathbb{N}$ an als

$$f(x) = \begin{cases} 2|x| + 1 & \text{falls } x \leq 0 \\ 2x & \text{falls } x > 0 \end{cases}$$

Wir erkennen, dass f die positiven ganzen Zahlen auf die geraden natürlichen Zahlen und die nichtpositiven ganzen Zahlen auf die ungeraden natürlichen Zahlen abbildet. Es ist leicht zu sehen, dass es keine ganzen Zahlen gibt, die auf die gleiche natürliche Zahl abgebildet werden. Des Weiteren gibt es für jede Zahl $i \in \mathbb{N}$ eine ganze Zahl x mit $f(x) = i$. Also ist f surjektiv und injektiv und damit eine Bijektion. ■

In den bisherigen Beispielen war es einfach, Gleichmächtigkeit (bzw. Abzählbarkeit) durch eine Bijektion anzugeben. Wir wollen uns nun ein weiteres, sehr interessantes Beispiel ansehen, bei welchem der Nachweis der Abzählbarkeit nicht so offensichtlich ist.

Lemma 1.2
Die Menge $\mathbb{N}^2 = \mathbb{N} \times \mathbb{N}$ ist abzählbar.

Beweis. Bevor wir die Abzählbarkeit von $\mathbb{N}^2$ beweisen, werden wir kurz darauf eingehen, welcher Ansatz nicht funktioniert. Die Menge $\mathbb{N} \times \mathbb{N}$ ist die Menge der geordneten Paare von natürlichen Zahlen. Diese Menge kann man sich gut in einer Tabelle vorstellen, bei welcher der Eintrag der i-ten Zeile und der j-ten Spalte das Tupel (i, j) enthält. Unser Ziel ist es, jedem Eintrag dieser Tabelle genau eine natürliche Zahl zuzuordnen. Am einfachsten wäre es natürlich, dass wir die Zahlen $1, 2, 3, \ldots$ Zeile für Zeile über die Tabelle verteilen. Dies funktioniert jedoch nicht. In der ersten Zeile gibt es ja unendlich viele Einträge, deshalb würden wir bereits für die erste Zeile alle natürlichen Zahlen „aufbrauchen“. Auch wenn wir spaltenweise vorgehen, haben wir das gleiche Problem. Es bedarf also einer anderen Strategie.

Anstatt die Nummern zeilen- oder spaltenweise zu vergeben, werden wir sie diagonal verteilen. Diese Idee (siehe Abbildung 1.1) ist so einfach wie genial. Wir bezeichnen mit der k-ten Gegendiagonalen der Tabelle alle Einträge (i, j) mit $i+j = k$. Die k-te Gegendiagonale

hat also $k-1$ Einträge, welche wir nach aufsteigender Zeilennummer sortieren. Zum Beispiel erhalten wir für die vierte Gegendiagonale die Folge $(1,3),(2,2),(3,1)$. Wir werden nun die Gegendiagonalen in aufsteigender Nummer abarbeiten. Dabei vergeben wir der Reihe nach die Nummern von $\mathbb{N}$ für die Einträge der Tabelle. Haben wir eine Gegendiagonale durchnummeriert, fahren wir mit der nächsten fort und beginnen das Nummerieren dort mit der nun folgenden Nummer. Auf diese Weise bekommen alle Einträge der Tabelle eine Nummer aus $\mathbb{N}$. Die so erhaltene Nummerierung beginnt demnach mit

$$(1,1),(1,2),(2,1),(1,3),(2,2),(3,1),(1,4),(2,3),(3,2),(4,1),(1,5),(2,4),\ldots.$$

Die Vergabe der Nummern impliziert eine Bijektion zwischen $\mathbb{N}$ und $\mathbb{N}^2$. Wir weisen hierbei der Zahl i das mit i beschriftete Paar der Tabelle zu. Offensichtlich ist diese Funktion injektiv und surjektiv. ■

Die Strategie, die Elemente von $\mathbb{N}\times\mathbb{N}$ in der im Beweis diskutierten Art zu nummerieren, ist als **Cantornummerierung** bekannt (nach ihrem Entdecker, dem deutschen Mathematiker Georg Cantor). Die daraus abgeleitete Funktion, die jedem Tabelleneintrag eine natürliche Zahl zuordnet, nennt man **cantorsche Paarungsfunktion**.

> **Test 1.3**
>
> Bestimmen Sie eine geschlossene Darstellung (Formel) für die cantorsche Paarungsfunktion.

Korollar 1.1
Die Menge der rationalen Zahlen $\mathbb{Q}$ ist abzählbar.

Beweis. Jede Zahl aus $\mathbb{Q}$ ist als Bruch p/q darstellbar. Diese Brüche tragen wir in eine Tabelle ein, sodass p/q in der Zelle (p,q) eingetragen wird. Nun nummerieren wir die Tabelleneinträge mit der Cantornummerierung. Daraus ergibt sich eine Folge F von Brüchen, die wie folgt beginnt:

$$1/1, 1/2, 2/1, 1/3, 2/2, 3/1, 1/4, 2/3, \ldots$$

Wir streichen in dieser Folge alle Brüche, die wir kürzen können, zum Beispiel 3/9. Die daraus entstandene Folge nennen wir F'. Die ersten Einträge von F' sind

$$1, 1/2, 2, 1/3, 3, 1/4, 2/3, 3/2, 4, 1/5, 5, \ldots.$$

Wir können nun die folgende Bijektion zum Nachweis der Gleichmächtigkeit zu $\mathbb{N}$ benutzen:

$$f(i) := i\text{-ter Eintrag in } F' \text{ als rationale Zahl.}$$

■

Man könnte den Eindruck bekommen, dass alle Mengen von Zahlen abzählbar sind. Das folgende Lemma zeigt aber, dass dem nicht so ist. Eine Menge, die nicht abzählbar ist, nennen wir **überabzählbar**.

$$
\begin{array}{lll}
f(1) & =?? . & \underline{9}\ 1\ 3\ 3\ 0\ 0\ 0\ 0\ 0\ 0\ 0 \\
f(2) & =?? . & 1\ \underline{5}\ 6\ 3\ 7\ 1\ 2\ 1\ 2\ 1\ 2 \\
f(3) & =?? . & 1\ 1\ \underline{1}\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1 \\
f(4) & =?? . & 4\ 5\ 1\ \underline{1}\ 0\ 1\ 9\ 3\ 3\ 1\ 2 \\
f(5) & =?? . & 2\ 2\ 2\ 2\ \underline{2}\ 0\ 0\ 0\ 0\ 0\ 0 \\
f(6) & =?? . & 2\ 2\ 2\ 2\ 2\ \underline{0}\ 0\ 0\ 0\ 0\ 0 \\
f(7) & =?? . & 1\ 4\ 1\ 5\ 9\ 2\ \underline{6}\ 5\ 3\ 5\ 8 \\
\\
r = 0 . & & 0\ 6\ 2\ 2\ 3\ 1\ 7\ \cdots
\end{array}
$$

Abbildung 1.2: Konstruktion einer reellen Zahl r, welche in der Nummerierung der reellen Zahlen durch f fehlt.

Lemma 1.3
Die Menge der reellen Zahlen $\mathbb{R}$ ist überabzählbar.

Beweis. Wir führen diesen Beweis als Widerspruchsbeweis. Dazu nehmen wir an, dass es eine Bijektion $f : \mathbb{N} \to \mathbb{R}$ gibt. Wir nutzen zum Beweis eine Tabelle T, in der wir die Nachkommastellen aller reellen Zahlen in Dezimaldarstellung „eintragen“. Die Tabelle hat unendlich viele Spalten und Zeilen. In der Zelle (i, j) tragen wir die j-te Nachkommastelle der reellen Zahl $f(i)$ ein. Wir schreiben für den Inhalt der Zelle (i, j) von T kurz $T[i, j]$.

Wir werden nun eine reelle Zahl definieren, welche nicht in der Tabelle als Zeile aufgelistet ist. Dazu nutzen wir die Funktion

$$
d(i, j) := \begin{cases} 0 & \text{falls } T[i, j] = 9 \\ T[i, j] + 1 & \text{sonst.} \end{cases}
$$

Diese Funktion garantiert, dass $T[i, j] \neq d(i, j)$ für alle $(i, j) \in \mathbb{N}^2$. Des Weiteren sind die Funktionswerte von d nicht großer als 9. Sei r nun folgende reelle Zahl in Dezimaldarstellung angegeben:

$$
r = 0.d(1, 1)d(2, 2)d(3, 3)d(4, 4)d(5, 5) \ldots
$$

Ein Beispiel für diese Konstruktion ist in Abbildung 1.2 zu sehen.

Wir nehmen nun an, dass es ein i gibt mit $f(i) = r$. Das heißt, dass die Nachkommastellen von r in der Tabelle T in Zeile i eingetragen wurden. Die i-te Nachkommastelle von r ist $d(i, i)$, der Eintrag in der Tabelle T an dieser Stelle ist aber $T[i, i] \neq d(i, i)$. Das ist ein Widerspruch, folglich ist die Zahl r nicht in T eingetragen worden, und damit fehlt r in der Nummerierung. Wir sehen also, dass $\mathbb{R}$ nicht abzählbar ist. ■

An dieser Stelle wollen wir uns die Zeit nehmen, über den Beweis von Lemma 1.3 noch etwas nachzudenken. Die Idee hinter dem Beweis ist sehr elegant, trotzdem kommt es hier oft zu Verwirrungen. Wir hatten gezeigt, dass in unserer Nummerierung der reellen Zahlen die Zahl r nicht vorkommen kann. Natürlich ist es möglich, eine andere Nummerierung zu finden, die r enthält. So könnte man zum Beispiel allen Zahlen eine um eins größere Nummer geben und dann r die Nummer 1 zuordnen. Bei dieser neuen Abbildung fehlt dann aber wieder eine (andere) Zahl.

Die Beweistechnik in Lemma 1.3 trägt den Namen **Diagonalisierung** und ist ein sehr mächtiges Werkzeug. Wir werden auch andere Sätze mit dieser Technik beweisen. Bei

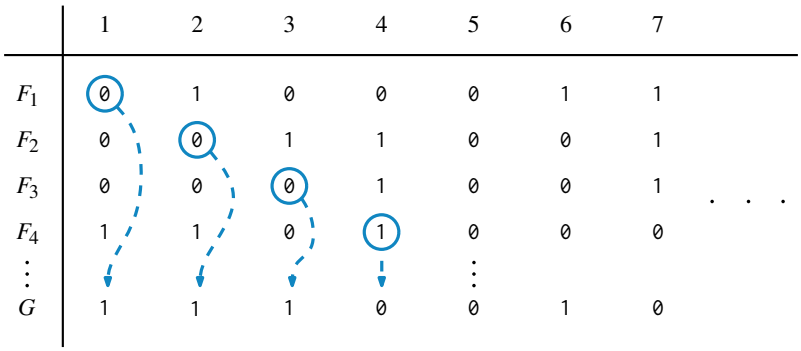

Abbildung 1.3: Allgemeines Prinzip der Diagonalisierung.

der Diagonalisierung geht es darum, nachzuweisen, dass eine Folge in einer Nummerierung von Folgen $(F_i)_{i\in\mathbb{N}}$ fehlt. Im vorigen Beweis ergab sich zum Beispiel F_i aus den Nachkommastellen der reellen Zahl $f(i)$. Man trägt die Folgen F_i als Zeilen in eine unbeschränkte Tabelle ein. Hierbei wird die Folge F_i in die i-te Zeile geschrieben. Nun nimmt man sich die Diagonale der Tabelle als neue Folge und verändert jeden Wert in dieser Folge. Dadurch erreicht man, dass die so konstruierte Folge G sich von allen Folgen der Tabelle unterscheidet. Konkret unterscheidet sich G von der Folge F_i an der i-ten Stelle (Tabelleneintrag (i,i)). Dieses Prinzip wird noch einmal in Abbildung 1.3 veranschaulicht.

Der Diagonalisierungsbeweis beruht auf dem Umstand, dass zwei Folgen schon dann unterschiedlich sind, wenn sie sich an einer Stelle unterscheiden. Wir konnten die erste Stelle benutzen, um die Ungleichheit zur ersten Folge zu erzwingen, die zweite Stelle für die Ungleichheit zur zweiten Folge, und so weiter.

Mit dem Lemma 1.3 haben wir gezeigt, dass es zwei unendliche Mengen gibt, die nicht gleichmächtig sind. In diesem Sinne konnten wir also nachweisen, dass es mehr reelle Zahlen als natürliche Zahlen gibt. Es gilt nun diese Ideen umzuformulieren, sodass man zeigen kann, dass es mehr Sprachen gibt als Turingmaschinen. Bislang haben wir nur Mengen von Zahlen verglichen. Wir wollen nun auch Sprachen und Mengen von Funktionen vergleichen.

Lemma 1.4
Die Menge Σ^* ist abzählbar.

Beweis. Wir haben bereits im Kursteil A die Standardnummerierung von Σ^* vorgestellt. Zur Erinnerung: Man sortiert alle Zeichen in Σ^* entsprechend ihrer Länge in aufsteigender Reihenfolge, wobei man Wörter mit gleicher Länge lexikographisch sortiert. Eine Bijektion f zwischen $\mathbb{N}$ und Σ^* ergibt sich direkt als

$$f(i) := \text{ das } i\text{-te Wort in der Standardaufzählung von } \Sigma^*.$$

■

Lemma 1.5
Die Menge $\{L \subseteq \Sigma^*\}$ aller Sprachen über Σ ist überabzählbar.

Beweis. Wir führen den Beweis als Diagonalisierungsbeweis. Wir nehmen also an, dass

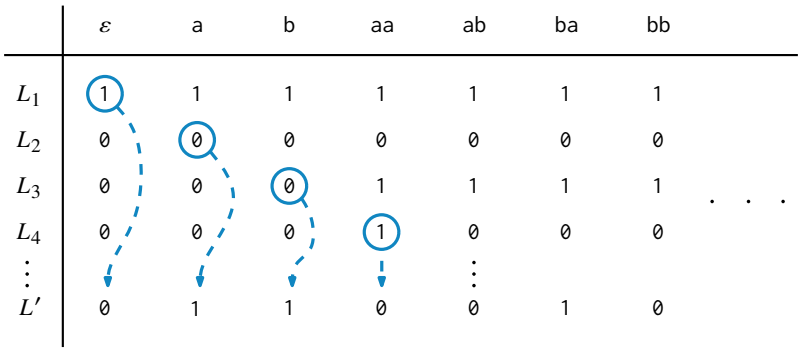

Abbildung 1.4: Ausführung des Beweises von Lemma 1.5 als Diagonalisierungsbeweis. Im Beispiel wurde $L_1 = \Sigma^*$, $L_2 = \emptyset$, $L_3 = \Sigma^2$, $L_4 = \{\mathtt{aa}\}$ gewählt.

eine Nummerierung aller Sprachen über Σ existiert. Sei dann L_i die i-te Sprache in dieser Nummerierung. Wir tragen alle Sprachen in eine Tabelle T ein. Hierbei werden wir L_i in die i-te Zeile eintragen. Die Spalten von T sind mit den Wörtern aus Σ^* beschriftet, sagen wir in der Reihenfolge der Standardnummerierung. Die Beschriftung der i-ten Spalte bezeichnen wir mit w_i. Wir markieren die Wörter, welche in L_i enthalten sind, indem wir in der korrespondierenden Zelle eine 1 eintragen. Wörter aus Σ^*, die nicht in L_i enthalten sind, markieren wir mit 0. Man könnte auch sagen, dass wir die charakteristische Funktion der Menge L_i in die i-te Zeile eintragen.

Wir konstruieren nun eine Sprache L', welche in der Nummerierung der L_i fehlt. Dazu „negieren wir die Diagonale der Tabelle“ und übernehmen die so erzeugte Sequenz als charakteristische Funktion von L' (siehe Abbildung 1.4). Das heißt konkret: Wenn $w_i \in L_i$, dann nehmen wir w_i nicht zu L' hinzu. Ist jedoch $w_i \notin L_i$, dann fügen wir w_i zu L' hinzu. Formal können wir definieren, dass

$$L' := \{w_i \mid w_i \notin L_i\}.$$

Die Sprache L' fehlt in der Nummerierung der L_i, da sie sich von jeder dieser Sprachen unterscheidet, da

$$w_i \in L_i \iff w_i \notin L'.$$

Somit haben wir durch den Diagonalisierungsbeweis gezeigt, dass $\{L \subseteq \Sigma^*\}$ überabzählbar ist.

■

Den letzten Beweis kann man natürlich auch kürzer führen und zum Beispiel auf die Verwendung einer expliziten Tabelle verzichten. Die Verwendung der Tabelle macht jedoch sehr deutlich, warum es sich bei diesem Beweis um einen Diagonalisierungsbeweis handelt.

Satz 1.1
Es gibt eine Sprache, die nicht aufzählbar ist.

Beweis. Für den Beweis fixieren wir ein Alphabet Σ. Es gibt höchstens so viele aufzählbare (erkennbare) Sprachen über Σ, wie es Turingmaschinen gibt. Jede Turingmaschine können wir als Wort über $\Sigma' = \{0, 1\}$ kodieren. Nach Lemma 1.4 ist $(\Sigma')^*$ abzählbar. Deshalb gibt es nur abzählbar viele Turingmaschinen und damit nur abzählbar viele erkennbare Sprachen.

Da es aber nach Lemma 1.4 überabzählbar viele Sprachen über Σ gibt, können nicht alle diese Sprachen erkennbare Sprachen sein. ■

Das Argument aus dem Beweis von Satz 1.1 ist eigentlich noch stärker. Man sieht sogar, dass es nicht nur eine nichtaufzählbare Sprache gibt sondern unendlich viele. Man könnte mit etwas mehr Aufwand sogar zeigen, dass fast alle Sprachen nicht aufzählbar sind.

1.2 Konstruktion von nichtaufzählbaren Sprachen

Wir wissen nun, dass es Sprachen gibt, die nicht aufzählbar sind. Unser nächstes Ziel ist es, eine solche Sprache auch konstruktiv zu bestimmen. Wir werden dies über einen kleinen Umweg machen, indem wir zuerst eine unentscheidbare Sprache konstruieren.

Satz 1.2
Die Sprache $A_{\text{TM}} := \{\langle M, w\rangle \mid M \text{ ist Turingmaschine die } w \text{ akzeptiert}\}$ ist nicht entscheidbar.

Beweis. Wir führen diesen Beweis als Widerspruchsbeweis. Dazu nehmen wir an, dass A_{TM} entscheidbar ist. Das heißt, es existiert eine Turingmaschine die A_{TM} erkennt und auf jeder Eingabe hält. Wir nennen diese Turingmaschine H. Es gilt also:

$$H(\langle M, w\rangle) = \begin{cases} \text{akzeptiert} & \text{M akzeptiert } w \\ \text{verwirft} & \text{M akzeptiert } w \text{ nicht} \end{cases}$$

Wir geben nun eine Turingmaschine $D(\langle M\rangle)$ an, welche wie folgt arbeitet.

1. D simuliert $H(\langle M, \langle M\rangle\rangle)$
2. D gibt das entgegengesetzte Ergebnis der Simulation zurück

Wir sehen, dass die Eingabe von D als Kodierung einer Turingmaschine aufgefasst wird. Im ersten Schritt wird die Maschine H als Unterprogramm mit der geforderten Eingabe aufgerufen. Offensichtlich existiert die Turingmaschine D, wenn die Turingmaschine H existiert. In diesem Fall ist D sogar ein Entscheider, hält also auf allen Eingaben.

Wir zeigen nun, dass D nicht existieren kann. Die Turingmaschine D liefert folgendes Ergebnis:

$$D(\langle M\rangle) = \begin{cases} \text{akzeptiert} & \text{wenn } M(\langle M\rangle) \text{ verwirft} \\ \text{verwirft} & \text{wenn } M(\langle M\rangle) \text{ akzeptiert} \end{cases}$$

Das heißt aber insbesondere, dass

$$D(\langle D\rangle) = \begin{cases} \text{akzeptiert} & \text{wenn } D(\langle D\rangle) \text{ verwirft} \\ \text{verwirft} & \text{wenn } D(\langle D\rangle) \text{ akzeptiert .} \end{cases}$$

Dies ist offensichtlich ein Widerspruch. Somit kann D nicht existieren und deshalb kann H nicht existieren. Unsere Annahme war daher falsch und demnach gibt es keinen Entscheider für A_{TM}. ■

Der letzte Beweis beruhte schon wieder auf einem Diagonalisierungsargument, auch wenn dieses etwas im Verborgenen bleibt. Die Turingmaschine H definiert uns eine Tabelle.

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$
M_1	akzeptiert	akzeptiert	akzeptiert	akzeptiert
M_2	verwirft	akzeptiert	verwirft	verwirft
M_3	verwirft	verwirft	verwirft	verwirft
M_4	verwirft	akzeptiert	akzeptiert	verwirft
$\vdots$			$\vdots$	
D	verwirft	verwirft	akzeptiert	akzeptiert

Abbildung 1.5: Das dem Beweis von Satz 1.2 zu Grunde liegende Diagonalisierungsargument.

Die Zeilen dieser Tabelle sind mit den Turingmaschinen M_i beschriftet, wobei es für jede mögliche Turingmaschine eine Zeile gibt. Die Spalten sind mit den Wörtern über Σ beschriftet, und zwar so, dass die i-te Spalte mit $\langle M_i \rangle$ beschriftet ist. Der Eintrag in Zelle $(M_i, \langle M_j \rangle)$ entspricht dem Ergebnis von $H(\langle M_i, \langle M_j \rangle \rangle)$. Siehe hierzu auch Abbildung 1.5. Die Maschine D übernimmt nun das Diagonalisieren. Dazu kehrt sie die Antworten auf der Diagonale um. Der Punkt ist, dass die zu D zugeordnete Sprache $L(D)$ entscheidbar ist, wenn H entscheidbar ist. Wir haben aber $L(D)$ von allen $L(M_i)$ unterschiedlich gemacht. Deshalb kann $L(D)$ nicht entscheidbar sein. Dadurch erreichen wir den Widerspruch zur Annahme, dass H existiert.

Der folgende Satz ist eine direkte Konsequenz aus Satz 1.2.

Satz 1.3
Das Komplement der Sprache A_{TM} ist nicht aufzählbar.

Beweis. Wir wissen nach Satz 1.2, dass $A_{\text{TM}} \notin \mathbb{E}$. Im Kursteil A haben wir zudem gesehen, dass A_{TM} von der universellen Turingmaschine erkannt wird, also ist $A_{\text{TM}} \in \mathbb{A}$. Ein weiterer Satz des A-Teils besagte, dass wenn $X \in \mathbb{A}$ und $X \in \text{co-}\mathbb{A}$, dann auch $X \in \mathbb{E}$. Angenommen $A_{\text{TM}} \in \text{co-}\mathbb{A}$, dann würde nach diesem Satz gelten, dass $A_{\text{TM}} \in \mathbb{E}$, was jedoch nicht gilt. Somit ist also $A_{\text{TM}} \notin \text{co-}\mathbb{A}$ oder anders gesagt, das Komplement von A_{TM} ist nicht aufzählbar. ■

An dieser Stelle wollen wir die Konsequenzen aus den Sätzen 1.2 und 1.3 diskutieren. Die Sprache A_{TM} ist nicht irgendeine künstlich konstruierte Sprache. Das dieser zur Sprache zu Grunde liegende Entscheidungsproblem hat eine hohe praktische Relevanz. Im Entscheidungsproblem fragen wir, ob eine Turingmaschine M eine Eingabe w akzeptiert. Etwas allgemeiner gefasst könnte man sagen, wir wollen wissen, ob ein gegebener Algorithmus A bei Eingabe x, die Ausgabe y produziert. Dieses Problem beschreibt die Verifikation eines Algorithmus/Programms. Die Aussage aus den beiden letzten Sätzen kann man so interpretieren, dass es keinen Algorithmus gibt, mit Hilfe dessen wir andere Algorithmen verifizieren können. Zwar können wir positive Ergebnisse verifizieren, da $A_{\text{TM}} \in \mathbb{A}$, wir werden aber nicht in allen Fällen die negativen Ergebnisse feststellen können, da $A_{\text{TM}} \notin \text{co-}\mathbb{A}$. Es ist natürlich möglich, für ausgewählte Algorithmen einen Nachweis zu erbringen, dass sie korrekt arbeiten. Was wir gezeigt haben ist, dass es kein allgemeines Verfahren für die Verifikation geben kann.

Ein zu A_{TM} eng verwandtes Problem ist das Problem

$$HALT := \{\langle M, w\rangle \mid \text{ die TM } M \text{ hält bei Eingabe } w\}.$$

Mit den Ideen aus Satz 1.2 kann man zeigen, dass auch $HALT \notin \mathbb{E}$.

Satz 1.4
Die Sprache *HALT* ist nicht entscheidbar und nicht co-aufzählbar.

Beweis. Der Beweis des Satzes ist völlig analog zum Beweis der Sätze 1.2 und 1.3. Wir nehmen an, dass *HALT* durch eine Maschine H entschieden wird und konstruieren daraufhin eine Turingmaschine D, welche bei Eingabe $\langle M\rangle$ zuerst $H(\langle M, \langle M\rangle\rangle)$ simuliert. Verwirft H die Eingabe, stoppt D akzeptierend. Akzeptiert H die Eingabe, gehen wir in eine Endlosschleife. Nun gilt

$$D(\langle D\rangle)\text{zykelt} \iff H(\langle D, \langle D\rangle\rangle)\text{akzeptiert} \iff D(\langle D\rangle) \text{ stoppt.}$$

Somit kann eine solche Turingmaschine H nicht existieren.

Als Nächstes zeigen wir, dass *HALT* erkennbar ist. Die Turingmaschine U' die *HALT* erkennt, arbeitet wie die universelle Turingmaschine, mit dem einzigen Unterschied, dass alle Übergänge in den verwerfenden Zustand nun in den akzeptierenden Zustand führen. Die Modifikation bewirkt, dass die Eingabe $\langle M, w\rangle$ von U' immer akzeptiert wird, wenn $M(w)$ stoppt (egal wie). Falls $M(w)$ nicht stoppt, wird $\langle M, w\rangle$ nicht akzeptiert, da U' auf dieser Eingabe zykeln wird. Da somit *HALT* erkennbar aber nicht entscheidbar ist, muss folgen, dass *HALT* nicht co-erkennbar (siehe Beweis zu Satz 1.3). ■

1.3 Entscheidbarkeit des universellen Wortproblems

Mit dem Wortproblem bezeichneten wir das Problem, zu entscheiden, ob ein Eingabewort w aus einer gegebenen Sprache ist, oder nicht. Alle Entscheidungsprobleme, werden bei uns als Wortprobleme modelliert. Für die (algorithmische) Lösung des Wortproblems ist es natürlich wichtig, in welcher Form die Sprache gegeben ist. Wir haben bislang im A-Teil verschiedene Möglichkeiten kennengelernt, wie man eine nichtendliche Sprache, durch ein endliches Wort beschreiben kann. Dazu wählt man sich ein Modell und gibt dann eine Kodierung einer konkreten Instanziierung (ein Programm, Beschreibung) an.

Es ist wünschenswert, alle Wortprobleme einer Sprachklasse *universell* lösen zu können. Damit ist gemeint, dass es einen Algorithmus gibt, der für jede Modellinstanz mit Eingabewort das Wortproblem löst. Ein solcher Algorithmus ist in diesem Sinne universell für das entsprechende Berechnungsmodell. Wir haben bereits gesehen, dass für das Berechnungsmodell Turingmaschine, kein solcher Algorithmus exisitieren kann, da $A_{\text{TM}} \notin \mathbb{E}$. Für die Modelle endlicher Automat und kontextfreie Grammatik existiert aber ein universeller Algorithmus für das Wortproblem. Dies wurde am Ende des A-Teils (Satz 7.5) gezeigt. Demnach gilt also, dass

$$A_{\text{CFG}} = \{\langle G, w\rangle \mid G \text{ ist CFG und } w \in L(G)\} \in \mathbb{E}, \text{ und}$$
$$A_{\text{DEA}} = \{\langle M, w\rangle \mid M \text{ ist DEA und } w \in L(M)\} \in \mathbb{E}.$$