Berechenbarkeitstheorie

Ralf Schindler

geTEXt von Martina Pfeifer

11. Oktober 2009

Dieses Skript basiert auf Vorlesungen, die ich im WiSe 99-00 (Einf. i. d. theoretische Informatik), WiSe 01-02 (Logik 1), SoSe 02 (Grundbegriffe der Logik), SoSe 02 (Logik 2), WiSe 02-03 (Einf. i. d. theoretische Informatik), SoSe 03 (Grundbegriffe der Logik) an der Universität Wien und im WiSe 05-06 (Einf. i. d. math. Logik und theoretische Informatik) und WiSe 08-09 (Berechenbarkeitstheorie) an der WWU Münster gehalten habe. Dabei habe ich vor allem die folgenden sehr schönen Textbücher verarbeitet:

N.J. Cutland, *Computability*, Cambridge, New York, Melbourne, 1980.

Herbert B. Enderton, *A mathematical introduction to logic*, San Diego, 1972 + 2001.

Michael Sipser, *Introduction to the theory of computation*, Boston, 1997.

Ralf Schindler

Inhaltsverzeichnis

Kapitel 1

Automaten und reguläre Sprachen

Die Berechenbarkeitstheorie (früher auch Rekursionstheorie genannt) ist die Theorie von Berechenbarkeit und Entscheidung. Die Theoretische Informatik ist nah an der Berechenbarkeitstheorie angesiedelt. Dieses Skript bietet eine Einführung in die Berechenbarkeitstheorie und in die Theoretische Informatik, die sowohl für angehende Informatiker als auch Logiker interessant sein sollte.

Wir wollen uns zunächst Entscheidungsverfahren an einem einfachen Beispiel ansehen.

Wenn Sie mit Münzen an einem Automaten eine Fahrkarte zu 1,50 € kaufen wollen, dann weiß eben dieser Automat, wann er von Ihnen ausreichend Geld erhalten hat. Nehmen wir der Einfachheit halber an, Sie können an diesem Automaten nur mit 50 Cent– und 1 € –Münzen bezahlen. Dieser Automat könnte dann vom Prinzip her wie folgt funktionieren.

Der Automat kennt 4 *Zustände*, einen 0 € –Zustand (den *Anfangszustand*), sowie die 50 Cent– und 1,50 € –Zustände. Zu Beginn ist der Automat im 0 € –Zustand. Wenn der Automat im 0 € –, 50 Cent– oder 1 € –Zustand ist und es werden 50 Cent eingeworfen, dann geht er in den 50 Cent–, 1 € – bzw. 1,50 € –Zustand über, und wenn 1 € eingeworfen werden, dann geht er in den 1 € –, 1,50 € – bzw. ebenfalls 1,50 € –Zustand über. Sobald der Automat im 1,50 € –Zustand ist, dann haben Sie mindestens (!) 1,50 € eingeworfen und der Automat wird Ihre Eingabe "akzeptieren" und Ihre Fahrkarte ausgeben. Der 1,50 € –Zustand ist daher ein *positiver Endzustand* (oder: *Akzeptanzzustand*).

Etwas formaler können wir die Arbeitsweise unseres Automaten mit Hilfe des folgenden Begriffes wiedergeben.

Ein (*endlicher*) *Automat* ist ein 5–Tupel $(Q, \Sigma, \delta, q_0, F)$, wobei Folgendes gilt.

(1) Q ist eine (nichtleere) endliche Menge (die Menge der *Zustände*).

(2) Σ ist eine (nichtleere) endliche Menge (das *Alphabet*; die Elemente von Σ heißen *Symbole*).

(3) $\delta : Q \times \Sigma \to Q$ ist eine Funktion (die *Übergangsfunktion*).

(4) $q_0 \in Q$ ist der *Startzustand*.

(5) $F \subset Q$ ist die Menge der Akzeptanzzustände.

Sei A ein endlicher Automat, $A = (Q, \Sigma, \delta, q_0, F)$. Dann kann δ auch durch eine *Tafel* (eine Matrix) der Größe $n \cdot m$ wiedergegeben werden, wobei n die Zahl der Element von Q und m die Zahl der Elemente von Σ ist.

Sei etwa $Q = \{q_0, q_1, q_2, q_3\}, \Sigma = \{0, 1\}$ und $F = \{q_3\}$. Dann könnte eine Übergangsfunktion $\delta : Q \times \Sigma \to Q$ durch folgende Tafel gegeben sein.

	0	1
q_0	q_1	q_2
q_1	q_2	q_3
q_2	q_3	q_3
q_3	q_3	q_3

Die Arbeitsweise eines solchen Automaten ist dann wie folgt zu verstehen.

Der Automat erhält eine endliche *Eingabe*, nämlich ein *Wort* über dem Alphabet Σ, d.h. eine endliche Folge von Symbolen aus Σ. Der Automat liest sodann die folgende Eingabe von links nach rechts. Bei jedem gelesenen Symbol geht er in Abhängigkeit vom Zustand, in dem er gerade ist, und vom vorgefundenen Symbol in einen neuen Zustand über. Beim Start ist er dabei im Zustand q_0.

Das eingegebene Wort wird dann *akzeptiert*, falls der Automat nach dem vollständigen Einlesen des Wortes in einem Zustand aus F ist.

Werde etwa das Wort 00 eingegeben. Die Zustände, die der Automat beim Einlesen dieses Wortes dann durchläuft, sind: q_0, q_1, q_2. Damit wird dieses Wort nicht akzeptiert. Bei der Eingabe von 10 etwa durchläuft der

Automat die Zustände q_0, q_2, q_3, und damit wird dieses Wort akzeptiert. Es ist unschwer erkennbar, dass das Wort

$$k_0 \dots k_i$$

genau dann akzeptiert wird, wenn

$$\sum_{j=0}^{i} (k_j + 1) \cdot 0,50 \text{ €} \geq 1,50 \text{ €}$$

ist, so dass dieser Automat genau derjenige ist, den wir eingangs informell beschrieben haben, wenn wir 0 als 50 Cent, 1 als 1 € , und q_0, q_1, q_2 und q_3 jeweils als 0– €–, 50 Cent–, 1 €und 1,50 €–Zustand interpretieren.

Sei Σ ein Alphabet, d.h. eine (nichtleere) endliche Menge von Symbolen. Wir schreiben dann Σ^* für die Menge aller Worte aus Σ, d.h. aller endlichen Symbolfolgen

$$x_0 x_1 \dots x_{k-1}$$

mit $x_0, x_1, \dots, x_{k-1} \in \Sigma$. Jede Teilmenge von Σ^* heißt eine *Sprache (über* Σ*)*.

Der Begriff "A akzeptiert w" wird formal wie folgt definiert. Sei $A = (Q, \Sigma, \delta, q_0, F)$ ein Automat, und sei

$$w = x_0 x_1 \dots x_{l-1} \in \Sigma^*$$

ein Wort über Σ, d.h. $x_i \in \Sigma$ für alle $i < l$. Dann *akzeptiert* A das Wort w gdw. eine Folge

$$q^0, q^1, \dots, q^l$$

von Zuständen existiert, so dass $q^0 = q_0$, $q^{i+1} = \delta((q^i, x_i))$ für alle $i < l$ und $q^l \in F$.

Definition 1.1 *Sei Σ ein Alphabet. Dann heißt eine Sprache $B \subset \Sigma^*$* regulär *gdw. es einen endlichen Automaten $A = (Q, \Sigma, \delta, q_0, F)$ gibt, so dass für alle $w \in \Sigma^*$ gilt:*

$$w \in B \Leftrightarrow A \text{ akzeptiert } w.$$

Um ein weiteres Beispiel kennen zu lernen wollen wir uns überlegen, dass die Menge aller endlichen 0–1–Folgen, die mit ein und demselben Symbol beginnen und enden, d.h. die Sprache

$$B = \{x_0 \dots x_{k-1} : k \in \mathbb{N}, x_0, \dots, x_{k-1} \in \{0, 1\}, x_0 = x_{k-1}\},$$

regulär ist.

Hierzu sei $Q = \{q_0, q_1, q_2, q_3, q_4\}, \Sigma = \{0, 1\}, F = \{q_1, q_2\}$ und die Übergangsfunktion $\delta : Q \times \Sigma \to Q$ durch die folgende Tafel gegeben.

	0	1
q_0	q_1	q_2
q_1	q_1	q_3
q_2	q_4	q_2
q_3	q_1	q_3
q_4	q_4	q_2

Wir wollen uns nun etwas systematischer überlegen, welche Sprachen regulär sind. Zunächst zeigen wir, dass die Klasse der regulären Sprachen durch gewisse Abschlusseigenschaften charakterisiert ist.

Lemma 1.2 *Sei Σ ein Alphabet, und seien $B, D \subset \Sigma^*$ regulär. Dann ist auch $B \cup D$ regulär.*

Beweis: Sei $A = (Q, \Sigma, \delta, q_0, F)$ ein Automat, der bezeugt, dass B regulär ist, und sei $A' = (Q', \Sigma', \delta', q'_0, F')$ ein Automat, der bezeugt, dass D regulär ist. Wir bauen nun einen Automaten $A^* = (Q^*, \Sigma^*, \delta^*, q^*_0, F^*)$, der bezeugt, dass $B \cup D$ regulär ist. Hierzu setzen wir

$$\begin{aligned}
&Q^* = Q \times Q' \\
&q^* = (q_0, q'_0), \\
&F^* = \{(q, q') \in Q^* : q \in F \text{ oder } q' \in F'\}, \\
&\text{und} \\
&\delta'(((q, q'), x)) = (\delta(q, x), \delta'(q', x))
\end{aligned}$$

für $q \in Q, q' \in Q'$ und $x \in \Sigma$.

Der Automat A^* simuliert gleichzeitig das Vorgehen von A als auch von A'. Man erkennt unschwer, dass A^* ein Wort $w \in \Sigma^*$ akzeptiert gdw. A oder A' das Wort w akzeptiert. $\square$

Wenn Σ ein Alphabet ist und wenn $w, w' \in \Sigma^*$, dann schreiben wir ww' für die Hintereinanderkettung von w und w', d.h.

$$ww' = x_0 \ldots x_{k-1} y_0 \ldots y_{l-1},$$

falls $w = x_0 \ldots x_{k-1}$ und $w' = y_0 \ldots y_{l-1}$.

Lemma 1.3 *Sei Σ ein Alphabet, und seien $B, D \subset \Sigma^*$ regulär. Dann ist auch $BD = \{ww' : w \in B$ und $w' \in D\}$ regulär.*

Beweis: Sei wieder $A = (Q, \Sigma, \delta, q_0, F)$ ein Automat, der bezeugt, dass B regulär ist, und sei $A' = (Q', \Sigma, \delta', q_0', F')$ ein Automat, der bezeugt, dass D regulär ist. Wir wollen einen Automaten $A^* = (Q^*, \Sigma, \delta^*, q_0^*, F^*)$ bauen, der bezeugt, dass BD regulär ist. Die Idee ist dabei die Folgende. Sei $v \in \Sigma^*$ gegeben. Für jede "Zerlegung" $v = ww'$ von v in zwei Folgen w und w' können wir zunächst mit Hilfe von A verifizieren, ob $w \in B$, und sodann mit Hilfe von A' berechnen, ob $w' \in D$. Da wir aber im Vorhinein nicht wissen, wie das Wort v gegebenenfalls erfolgreich zu zerlegen ist, müssen wir *alle* möglichen Zerlegungen $v = ww'$ von V gleichzeitig untersuchen lassen. Technisch realisieren wir dies dadurch, dass wir jedes Mal, wenn A ein Anfangsstück von v als zu B gehörig verifiziert hat, damit beginnen, A' nachrechnen zu lassen, ob der jeweilige Rest von v zu D gehört.

Zur notationellen Vereinfachung wollen wir annehmen, dass Q und Q' disjunkt sind. Wir definieren dann $A^* = (Q^*, \Sigma, \delta^*, q_0^*, F^*)$ wie folgt. Wir setzen $Q^* = \mathcal{P}(Q \cup Q')$, die Menge aller Teilmengen von $Q \cup Q'$. Weiterhin sei $q_0' = \{q_0, \}$ und

$$F^* = \{R \subset Q \cup Q' : R \cap F' \neq \emptyset\},$$

d.h. die Menge aller Teilmengen von $Q \cup Q'$, in denen ein Element von $F \cup F'$ liegt.

Schließlich werde δ^* wie folgt definiert. Sei $R \subset Q \cup Q'$, und sei $x \in \Sigma$. Dann sei $q \in \delta^*(R, x)$ gdw.:

(a) es gibt ein $\overline{q} \in R \cap Q$, so dass $q = \delta(\overline{q}, x)$ (und damit $q \in Q$), oder

(b) es gibt ein $\overline{q} \in R \cap Q$, so dass $\delta(\overline{q}, x) \in F$ und $q = q_0'$ (und damit $q \in Q'$), oder

(c) es gibt ein $\overline{q} \in R \cap Q'$, so dass $\delta'(\overline{q}, x) = q$ (und damit $q \in Q'$).

Genau im Falle (b) tritt der Automat A^* vom Modus des Rechnens gemäß A in den Modus des Rechnens gemäß A' über.

Es ist nun leicht, sich davon zu überzeugen, dass A^* das Gewünschte leistet. $\square$

Lemma 1.4 *Sei Σ ein Alphabet, und sei $B \subset \Sigma^*$ regulär. Dann ist auch*

$$B^* = \{w_0 w_1 \dots w_{l-1} : l \geq 1 \text{ und } w_i \in B \text{ für jedes } i < l\}$$

regulär.

Beweis: Die Beweisidee ist dieselbe wie für den vorigen Beweis. Sei $A = (Q, \Sigma, \delta, q_0, F)$ ein Automat, der bezeugt, dass B regulär ist. Wir bauen dann einen Automaten $A^* = (Q^*, \Sigma, \delta^*, q_0^*, F^*)$, der bezeugt, dass B^* regulär ist, wie folgt.

Sei $Q^* = \mathcal{P}(Q)$, die Menge aller Teilmengen von Q. Sei $q_0^* = \{q_0\}$, und sei

$$F^* = \{R \subset Q : R \cap F \neq \emptyset\}.$$

Schließlich sei δ^* wie folgt definiert. Sei $R \subset Q$, und sei $x \in \Sigma$. Dann sei $q \in \delta^*(R, x)$ gdw.:

(a) es gibt ein $\overline{q} \in R$, so dass $q = \delta(\overline{q}, x)$, oder

(b) es gibt ein $\overline{q} \in R$, so dass $\delta(\overline{q}, x) \in F$ und $q = q_0$.

Dann akzeptiert A^* das Wort w gdw. $w \in B^*$. $\square$

Wir wollen nun beweisen, dass die Abschlusseigenschaften, die durch die letzten drei Lemmata wiedergegeben werden, die Menge der regulären Sprachen charakterisiert.

Zu diesem Zwecke sollten wir zunächst eine Möglichkeit besitzen, Sprachen, die durch diese Abschlusseigenschaften gegeben werden, zu beschreiben. Dies geschieht mit Hilfe des Begriffs "regulärer Ausdruck".

Sei Σ ein Alphabet. Wir nehmen dann o.B.d.A. an, dass die Symbole $(,), \cup, \circ, ^*$ nicht in Σ enthalten sind. Ein *regulärer Ausdruck über* Σ ist dann eine (endliche) Folge w von Symbolen aus $\Sigma \cup \{(,), \cup, \circ, ^*\}$, so dass w in jeder Menge M mit folgenden Eigenschaften liegt.

(a) $x \in M$ für jedes $x \in \Sigma \cup \{\epsilon\}$,

(b) wenn $w_1, w_2 \in M$, dann sind auch die Folgen $(w_1 \cup w_2), (w_1 \circ w_2)$ und w_1^* in M.

Sei etwa $\Sigma = \{0, 1\}$. Reguläre Ausdrücke über Σ sind dann z.B.

$$1,\ (0 \cup 1)^* \text{ und } (0 \circ 1) \circ 0^*.$$

Jeder reguläre Ausdruck w über Σ beschreibt eine Sprache $A \subset \Sigma^*$. Für $x \in \Sigma \cup \{\epsilon\}$ beschreibt x die Sprache $\{x\}$. Wenn w_1 die Sprache A_1 und w_2 die Sprache A_2 beschreibt, dann beschreibt:

(i) $(w_1 \cup w_2)$ die Sprache $A_1 \cup A_2$,

(ii) $(w_1 \circ w_2)$ die Sprache $A_1 A_2 = \{ww' : w \in A_1 \text{ und } w \in A_2\}$, und

(iii) w_1^* die Sprache $A_1^* = \{v_0 v_1 \dots v_{l-1} : l \geq 1 \text{ und } v_i \in A_1 \text{ für jedes } i < l\}$.

Die letzten drei Lemmata zeigen damit den folgenden

Satz 1.5 *Jeder reguläre Ausdruck beschreibt eine reguläre Sprache.*

Wir wollen nun die Umkehrung hierzu beweisen.

Satz 1.6 *Jede reguläre (nichtleere) Sprache wird durch einen regulären Ausdruck beschrieben.*

Beweis: Sei $\emptyset \neq B \subset \Sigma^*$ eine reguläre Sprache, wobei Σ ein Alphabet ist, welches o.B.d.A. die Symbole $(,), \cup, \circ, ^*$ nicht enthält. Sei $A = (Q, \Sigma, \delta, q_0, F)$ ein Automat, der genau die Worte aus B akzeptiert. Wir wollen uns zunächst die Wirkungsweise von A etwas anders vorstellen als bisher und führen dazu ad hoc den Begriff des "variierten Automaten" ein.

Ein *variierter Automat* A' ist ein 5–Tupel $(Q', \Sigma, \delta', q^B, q^E)$, wobei Folgendes gilt.

(a) Q' ist eine endliche Menge mit $q^B, q^E \in Q$, die Menge der *Zustände*, wobei q^B den *Beginn*– und q^E der *Endzustand* ist.

(b) δ' ist eine Funktion mit Definitionsbereich $(\mathbb{Q}' \setminus \{q^E\}) \times (Q' \setminus \{q^B\})$, so dass für alle $p \in Q' \setminus \{q^E\}$ und $q \in Q' \setminus \{q^B\}$ gilt: $\delta'((p, q))$ ist entweder ein regulärer Ausdruck über Σ, oder das leere Symbol ε, oder die leere Menge $\emptyset$.

Wir bauen zuerst den Automaten A in einen variierten Automaten A' um. Hierzu nehmen wir o.B.d.A. an, dass $q^B \notin Q$ und $q^E \notin Q$, und wir setzen $Q' = Q \cup \{q^B, q^E\}$. q^B ist ein neuer Beginnzustand, und q^E ist ein neuer (eindeutiger) Endzustand. Die Funktion δ' beschreibt, auf welche Weise A aus dem Zustand $p \in Q'$ in den Zustand $q \in Q'$ geraten kann.

Wir setzen:

(i) $\delta((q^B, q_0)) = \varepsilon$,

(ii) $\delta((q, q^E)) = \varepsilon$ für jedes $q \in F$.

A' gelangt also trivial aus dem Beginnzustand q^B in den Anfangszustand q_0 von A, und A' gelangt trivial von jedem Akzeptanzzustand $q \in F$ von A in den Endzustand q^E von A'.

Weiterhin setzen wir:

(iii) für $p, q \in Q$ sei

$$\delta((p, q)) = (\ldots(x_0 \cup x_1) \cup \ldots \cup x_{l-1}),$$

wobei

$$\{x_0, x_1, \ldots, x_{l-1}\} = \{x \in \Sigma : \delta((p, x)) = q\}.$$

(Falls letztere Menge leer ist, dann setzen wir $\delta((p, q)) = \emptyset$.)

A' weist also dem Paar (p, q) einen regulären Ausdruck zu, so dass die zugehörige reguläre Sprache die Menge aller Symbole aus Σ ist, die es A erlaubt, durch das Lesen *dieses einen* Symbols aus dem Zustand p in den Zustand q überzugehen.

Sei $A^* = (Q^*, \Sigma, \delta^*, q_B, q^E)$ ein variierter Automat. Sei $w \in \Sigma^*$. Wir sagen dann, dass A^* das Wort w akzeptiert gdw. eine Folge

$$q^{-1} = q^B, q^0, q^1, \ldots, q^l, q^E = q^{l+1}$$

und Worte $w^0, w^1, \ldots, w^{l+1}$ existieren, so dass für jedes $i \leq l+1$ das Wort w^i in der durch den regulären Ausdruck $\delta'((q^{i-1}, q^i))$ beschriebenen Sprache liegt und

$$w = w^0 w^1 \ldots w^{l+1}.$$

(Dabei werde das leere Wort durch ε beschrieben.) Für unseren oben konstruierten variierten Automaten A' gilt nun offensichtlich:

Sei $w \in \Sigma^*$. Dann akzeptiert A' das Wort w gdw. A das Wort w akzeptiert (gdw. $w \in B$). Wir zeigen nun:

Behauptung. Sei $A^* = (Q^*, \Sigma, \delta^*, q^B, q^E)$ ein variierter Automat, der genau die Worte aus B akzeptiert. Angenommen, $Q^* \setminus \{q^B, q^E\} \neq \emptyset$ (d.h. Q^* hat mehr als zwei Zustände). Sei $n > 2$ die Anzahl der Zustände von A^*. Dann existiert ein variierter Automat $A^{**} = (Q^{**}, \Sigma, \delta^{**}, q^B, q^E)$, der genau die Worte aus B akzeptiert und der $n-1$ Zustände hat.

Beweis der Behauptung. Sei $q^* \in Q^* \setminus \{q^B, q^E\}$ beliebig. Wir setzen $Q^{**} = Q^* \setminus \{q^*\}$. Die neue Funktion δ^{**} werde wie folgt definiert.

Sei $p \in Q^{**} \setminus \{q^E\}$ und $q \in Q^{**} \setminus \{q^B\}$. Sei $\delta^*((p, q^*)) = w_1, \delta^*((q^*, q^*)) = w_2, \delta^*((q^*, q)) = w_3$ und $\delta^*((p, q)) = w_4$. Dann sei

$$\delta^{**}((p, q)) = (((w_1 \circ w_2^*) \circ w_3) \cup w_4),$$

falls $\delta^*((p, q^*)) \neq \emptyset \neq \delta^*((q^*, q))$ und $\delta^*((q^*, q^*)) \neq \emptyset \neq \delta^*((p, q))$,

$$\delta^{**}((p, q)) = ((w_1 \circ w_2^*) \cup w_3),$$

falls $\delta^*((p,q^*)) \neq \emptyset \neq \delta^*((q^*,q))$ und $\delta^*((p,q)) = \emptyset$ (wobei wir dies so verstehen wollen, dass dann im zusätzlichen Falle von $\delta^*(q^*,q^*)) = \emptyset$ der Wert $(w_1 \circ w_3)$ sei), und

$$\delta^{**}((p,q)) = w_4,$$

falls $\delta^*((p,q^*)) = \emptyset$ oder $\delta^*((q^*,q)) = \emptyset$, aber $\delta((p,q)) \neq \emptyset$.

Während Q^* beschreibt, wie man entweder direkt oder auf dem Umweg über q^* von p nach q gelangen kann, erlaubt Q^{**} nur den direkten Weg von p nach q. Dieser direkte Weg ist entweder derselbe wie der durch Q^* gegeben oder er entsteht aus dem Umweg über q^*, indem erst von p nach q^* gegangen wird, sodann in beliebig vielen Schlaufen von q^* nach q^* gegangen wird, und schließlich von q^* nach q gegangen wird.

Es ist leicht zu verifizieren, dass $A^{**} = (Q^{**}, \Sigma, \delta^{**}, q^B, q^E)$ genau dieselben Worte akzeptiert wie A^*.

Sei k die Anzahl der Zustände von A, so dass A' $k+2$ Zustände besitzt. Durch k–fache Anwendung der obigen Behauptung erhalten wir einen variierten Automaten $A_0 = (\{q^B, q^E\}, \Sigma, \delta_0, q^B, q^E)$ mit den beiden Zuständen q^B und q^E, so dass A_0 genau die Worte aus B akzeptiert. Dann muss aber die Sprache B genau diejenige sein, die durch den regulären Ausdruck $\delta_0((q^B, q^E))$ beschrieben wird.

Dies beweist, dass jede reguläre Sprache durch einen regulären Ausdruck beschrieben wird. □

Wir wollen nun eine Methode kennen lernen, mit deren Hilfe sich zeigen lässt, dass eine gegebene Sprache $B \subset \Sigma^*$ *nicht* regulär ist.

Satz 1.7 (Aufpump–Lemma für reguläre Sprachen, "pumping lemma") *Sei Σ ein Alphabet, und sei $B \subset \Sigma^*$ eine reguläre Sprache. Dann existiert eine natürliche Zahl k, so dass sich jedes Wort $w \in B$ der Länge $\geq k+1$ schreiben lässt als*

$$w = w_0 w_1 w_2,$$

so dass weiterhin gilt:

(a) die Länge von $w_0 w_1$ ist $\leq k$, und

(b) für jedes $i \geq 0$ gilt

$$w_0 \underbrace{w_1 w_1 \ldots w_1}_{i \text{ viele}} w_2 \in B.$$

Beweis: Sei der Automat $A = (Q, \Sigma, \delta, q_0, F)$ Zeuge dafür, dass B regulär ist. Sei n die Anzahl der Elemente von Q, und sei m die Anzahl der Elemente von Σ. Setze $k = n \cdot m$. Sie Idee ist, dass A beim Lesen eines Wortes der Länge $> k$ eine "Schlaufe" durchlaufen muss.

Genauer gilt Folgendes. Sei $w = x_0 x_1 \ldots x_{l-1} \in \Sigma^*$, wobei $l > k$. Sei die Folge

$$q_1, q_2, \cdots, q_l$$

von Zuständen $q_i \in Q$ rekursiv definiert durch

$$q_{i+1} = \delta(q_i, x_i),$$

für $i < l$.

Da es nur $k = n{\cdot}m$ viele Paare $(q, x) \in Q \times \Sigma$ gibt, muss es $i < j < k{+}1 \leq l$ geben mit $(q_i, x_i) = (q_j, x_j)$. Wir setzen $w_0 = x_0 \ldots x_{i-1}, w_1 = x_i \ldots x_{j-1}$ und $w_2 = x_j \ldots x_{l-1}$. Es ist dann unschwer erkennbar, dass für jedes beliebige $i \geq 0$ A nach dem Einlesen von w in genau demselben Zustand ist wie nach dem Einlesen von

$$w_0 \underbrace{w_1 w_1 \ldots w_1}_{i \text{ viele}} w_2,$$

so dass aus $w \in B$ folgt, dass $w_0 w_1 \ldots w_1 w_2 \in B$. Außerdem gilt $j \leq k$, so dass die Länge von $w_1 w_2$ höchstens k ist.

Das Aufpump–Lemma kann benutzt werden zu zeigen, dass gewisse sehr einfache Sprachen nicht regulär sind, z.B.

$$B = \{0^n 1^n : n \in \mathbb{N}\},$$

wobei $0^n = \underbrace{0 \ldots 0}_{n \text{ viele}}, 1 = \underbrace{1 \ldots 1}_{n \text{ viele}}$.

Kapitel 2

Kontextfreie Sprachen und Kellerautomaten

Eine kontextfreie Grammatik sieht beispielsweise so aus:

$$\begin{array}{lcl} x & \mapsto & 0x1 \\ x & \mapsto & y \\ y & \mapsto & \# \end{array}$$

Diese Regeln sind so zu verstehen, dass mit ihrer Hilfe Symbolfolgen generiert werden. Ein einzelnes Symbol (x bzw. y) darf durch die angegebene Symbolfolge (also $0x1$ oder y bzw. $\#$) ersetzt werden. Dabei starten wir mit einem einzelnen Symbol. Wenn wir mit x starten, so können wir etwa $000\#111$ folgendermaßen generieren:

$$\begin{array}{l} x \Rightarrow 0x1 \Rightarrow 00x11 \Rightarrow \\ 000x111 \Rightarrow 000y111 \Rightarrow \\ 000\#111 \end{array}$$

Formal besteht eine *kontextfreie Grammatik* aus einer endlichen Menge V von Variablen, einer endlichen Menge Σ von Terminalen ($\Sigma \cap V = \emptyset$), einer endlichen Menge R von Regeln, die einzelnen Variablen Worte aus $(\Sigma \cup V)^*$ zuweisen, und einer Startvariablen $s \in V$. Im obigen Beispiel ist etwa $V = \{x, y\}$, $\Sigma = \{0, 1, \#\}$, R wird durch die obigen drei Zeilen angegeben, und $s = x$. Durch wiederholte Anwendung der Regeln können wir ausgehend von s verschiedene Symbolfolgen generieren.

Wenn wir wieder, für ein Symbol γ, γ^n für die Symbolfolge

$$\underbrace{\gamma\gamma\cdots\gamma}_{n \text{ viele}}$$

schreiben, dann können wir im obigen Beispiel die Folgen $0^n x 1^n$, $0^n y 1^n$ und $0^n \# 1^n$ für $n \in \mathbb{N}$ generieren. Ausgezeichnet seien dabei die Symbolfolgen aus Σ^*, da sie nur aus Terminalen bestehen.

Eine Sprache $L \subset \Sigma^*$ heißt *kontextfrei* gdw. es eine kontextfreie Grammatik gibt, die aus V, Σ, R, s besteht, so dass die Symbolfolgen aus Σ^*, die diese Grammatik generiert, genau diejenigen sind, die zu L gehören. Demnach ist also die Sprache $\{0^n \# 1^n : n \in \mathbb{N}\}$ kontextfrei.

Die leere Folge ϵ ist nach wie vor auch eine Folge.

Betrachten wir $V = \{x\}, \Sigma = \{0, 1\}, s = x$ und Regeln R, die wie folgt gegeben sind:

$$x \mapsto 0x1$$
$$x \mapsto \epsilon$$

Die Folgen aus Σ^*, die sich mit Hilfe dieser Grammatik erzeugen lassen, sind genau diejenigen der Form $0^n 1^n$ für $n \in \mathbb{N}$. Die Sprache $\{0^n 1^n : n \in \mathbb{N}\}$ ist also kontextfrei; dies sieht man leicht, indem man die obigen Regeln sanft variiert. Aufgrund des Aufpump-Lemmas für reguläre Sprachen ist aber $\{0^n 1^n : n \in \mathbb{N}\}$ nicht regulär.

Wie sieht es mit der Sprache $\{0^{2^n} : n \in \mathbb{N}\}$ aus? Wir wollen zeigen, dass diese *nicht* kontextfrei ist. Zunächst aber wollen wir den Begriff der kontextfreien Sprache formal definieren.

Definition 2.1 *Sei (V, Σ, R, s) eine kontextfreie Grammatik, und sei $w \in \Sigma^*$. Dann* generiert *(V, Σ, R, s) das Wort w gdw. eine Folge*

$$w_0, w_1, \ldots w_{N-1}$$

von Wörtern aus Σ^* (mit $N > 1$) existiert, so dass gilt: $w_0 = s$, $w_{N-1} = w$ und für alle $i + 1 < N$ können wir für geeignete $\gamma_0, \gamma_1, \ldots, \gamma_{k-1} \in V \cup \Sigma$ und $w^0, w^1, \ldots, w^{k-1} \in (V \cup \Sigma)^*$ schreiben

$$w_i = \gamma_0 \gamma_1 \cdots \gamma_{k-1},$$

$$w_{i+1} = w_0 w_1 \ldots w_{k-1},$$

und jedes w_j für $j < i$ geht aus γ_j durch Anwendung einer Regel aus R hervor.

Lemma 2.2 (Aufpump–Lemma für kontextfreie Sprachen, "pumping lemma"). *Sei $L \subset \Sigma^*$ eine kontextfreie Sprache. Dann existiert ein $p \in \mathbb{N}$, so dass für alle $w \in L$, deren Länge mindestens p ist, eine Zerlegung von w in fünf Teile $w \equiv acedb$ existiert, so dass gilt:*

(a) für alle $n \in \mathbb{N}$ ist $ac^n ed^n b$ in L,[1]

(b) cd hat positive Länge, und

(c) ced hat höchstens die Länge p.

Die Zahl p heißt die "Pump–Länge".

Aus diesem Lemma ergibt sich sofort, dass $\{0^{2^n} : n \in \mathbb{N}\}$ nicht kontextfrei ist. Andernfalls sei p die Pump–Länge. Sei $2^n \geq p$, und sei $w \equiv abcde$ eine Zerlegung des Wortes $w \equiv 0^{2^n}$ wie im Aufpump–Lemma. Wenn $m > 0$ die Länge von bd ist, dann wären alle Zahlen der Gestalt $2^n - m + km = 2^n + (k-1)m$ für $k \in \mathbb{N}$ Zweierpotenzen. Das ist aber Unsinn.

Beweis des Aufpump–Lemmas: Sei die durch V, Σ, R, s gegebene kontextfreie Grammatik Zeuge dafür, dass $L \subset \Sigma^*$ eine kontextfreie Sprache ist. Die Worte $w \in L$ sind also genau diejenigen $w \in \Sigma^*$, die sich aus s mit Hilfe von R generieren lassen, d.h. so dass eine Generierung $s = w_0 \Rightarrow w_1 \Rightarrow \ldots \Rightarrow w_{N-1} = w$ existiert, wobei in jedem Schritt für jedes $\gamma \in V$, das in w_i vorkommt, eine der entsprechenden Regeln aus R auf γ angewandt wird. Wenn $\gamma \in V$ und γ in w_i vorkommt (für ein $i < N-1$), dann können wir zusehen, was im weiteren Verlauf der Generierung (von w aus s) aus γ wird. Wir werden in einem nächsten Schritt eine Regel aus R auf γ anwenden (da γ kein Terminal ist), wodurch ein Wort entsteht, so dass auf die Variablen in diesem Wort wiederum später Regeln aus R angewandt werden, etc. Wenn also γ in w_i an der r^{ten} Stelle vorkommt, so gehört zu γ für jedes $j \geq i$ mit $j < N$ dasjenige Teilwort $w^r_{i,j}$ von w_j, das aus γ im weiteren Verlauf der Generierung aus γ hervorgeht:

$$\begin{gathered} s \\ \vdots \\ w_i \equiv a\gamma b \\ \vdots \\ w_j \equiv a' w^r_{i,j} b' \\ \vdots \\ w_{N-1} \equiv a'' w^r_{i,N-1} b''. \end{gathered}$$

Wir schreiben im Folgenden $w^\gamma_{i,j}$ anstelle von $w^r_{i,j}$, obwohl dieses Wort ja von der Selle des Vorkommens von γ in w_i (d.h. von r) abhängen kann.

[1] Sei f eine Folge. Wir schreiben dann f^n für das Resultat des n–fachen Hintereinanderschreibens der Folge f.

Wir dürfen annehmen, dass es beliebig lange Worte w aus L gibt, da sonst die zu beweisende Aussage trivial richtig ist.

Sei $c \geq 2$ so dass jede Anwendung einer Regel ein Symbol aus Σ durch eine Folge aus $(\Sigma \cup V)^*$ der Länge höchstens c ersetzt. Sei dann

$$s = w_0 \Rightarrow w_1 \Rightarrow \ldots \Rightarrow w_{N-1} = w$$

so, dass die Länge von w mindestens $p = c^{l+1}$ ist, wobei l die Zahl der Elemente von V ist. Für $i < N$ ist dann die Länge von w_i höchstens c^i, so dass sicherlich gilt: $N \geq l + 2$.

Wir wählen nun für jedes $w \in L$ der Länge $\geq p$ eine Art und Weise

$$s = w_0 \Rightarrow w_1 \Rightarrow \ldots \Rightarrow w_{N-1} = w$$

der Generierung, so dass für alle $i < N - 1$ gilt: w_i enthält Variablen, und auf jede Variable in w_i wird im Schritt von w_i nach w_{i+1} eine Regel aus R angewandt. Außerdem wollen wir voraussetzen, dass R keine "triviale" Regel der Form $\gamma \mapsto \gamma'$ für γ, $\gamma' \in V$ enthält. Darüberhinaus dürfen wir voraussetzen, dass R keine Regel der Gestalt $\gamma \mapsto \epsilon$ für $\gamma \in V \setminus \{s\}$ enthält; cf. den Beweis von Lemma 2.3. Für jede solche Generierung definieren wir $(i, \gamma) < (j, \gamma')$ durch: $i < j < N-1$, γ und γ' sind beide aus V, und γ' kommt in $w_{i,j}^{\gamma}$ vor. Da w_{N-2} Variablen enthält (!), existieren $\gamma_0, \gamma_1, \ldots, \gamma_{N-2}$ mit $(0, \gamma_0) < (1, \gamma_1) < \ldots < (N-2, \gamma_{N-2})$. Da $N - 1 \geq l + 1$, existiert ein $\gamma \in V$ mit $\gamma_i = \gamma = \gamma_j$ für geeignete $i < j < N - 1$ mit $i \geq N - l - 2$.

Aufgrund unserer Voraussetzungen an das Regelwerk R kann $w_{i,j}^{\gamma}$ nicht nur aus dem Symbol γ bestehen, hat also Länge ≥ 2. Wir erhalten damit folgendes Bild:

$$\begin{array}{c} s \\ \vdots \\ w_i \equiv a'\gamma b' \\ \vdots \\ w_j \equiv a''c'\gamma d'b'' \\ \vdots \\ w_{N-1} \equiv acw_{j,N-1}^{\gamma}db, \end{array}$$

wobei $c'\gamma d' \equiv w_{i,j}^{\gamma}$ und $c'd'$ *nicht* die leere Folge ϵ ist.

Durch Auslassen bzw. Ineinanderschachteln sieht man dann aber, dass auch

$$\begin{aligned}&a''\gamma b'',\\&a''c'w^\gamma_{i,j}d'b'' \equiv a''c'^2\gamma d'^2 b'',\\&a''c'^2w^\gamma_{i,j}d'^2b'' \equiv a''c'^3\gamma d'^3 b'', \text{ etc.,}\end{aligned}$$

also auch

$$\begin{aligned}&aw^\gamma_{j,N-1}b,\\&ac^2w^\gamma_{j,N-1}d^2b,\\&ac^3w^\gamma_{j,N-1}d^2b, \text{ etc.}\end{aligned}$$

generiert werden können. Mit $e \equiv w^\gamma_{j,N-1}$ zeigt dies also die Aussagen (a), (b) und (c) des Aufpump–Lemmas. Insbesondere gilt (c), da $i \geq N - l - 2$, so dass die Länge von *ced* höchstens $c^l < p$ sein kann. $\square$

Eine kontextfreie Grammatik ist in CHOMSKY*scher Normalform* gdw. jede Regel eine der drei folgenden Formen besitzt:

$$\begin{array}{ccc} s & \mapsto & \epsilon \\ v & \mapsto & uu' \\ v & \mapsto & t \end{array}$$

Hierbei ist s die Startvariable, v, u, u' sind Variablen und t ist ein Terminal.

Lemma 2.3 *Sei $L \subset \Sigma^*$ eine kontextfreie Sprache. Dann existiert eine kontextfreie Grammatik in Chomskyscher Normalform, so dass die Symbolfolgen aus Σ^*, die diese Grammatik generiert, genau diejenigen sind, die zu L gehören.*

Beweis: Werde L durch die kontextfreie Grammatik (V, Σ, R, s) generiert. Wir fügen zu V neue Variablen hinzu und zerbrechen einzelne Regelanwendungen in Anwendungen mehrerer Regeln.

Zunächst führen wir eine neue Variable s_0 als neues Startsymbol ein und wir nehmen

$$s_0 \mapsto s$$

als neue Regel hinzu.

Betrachten wir nun Regeln aus R der Form

$$v \mapsto \epsilon,$$

wobei $v \not\equiv s$. Für jede solche Regel betrachten wir alle Regeln

$$v' \mapsto w$$

aus R, wobei v' eine Variable und w ein Wort aus $(\Sigma \cup V)^*$ ist, in dem v vorkommt. Wir fügen dann für jede Menge M von Vorkommnissen der Variablen v in w die Regel

$$v' \mapsto \overline{w}$$

hinzu, wobei $\overline{w}$ aus w hervorgeht, indem alle Vorkommnisse von v in w aus dieser Menge M gelöscht werden. (Wenn also v in w n–fach vorkommt, dann fügen wir 2^n Regeln hinzu.) Anschließend streichen wir die Regel

$$v \mapsto \epsilon.$$

Wir wiederholen all dies solange, bis wir keine Regel der Form $v \mapsto \epsilon$ mit $v \not\equiv s_0$ mehr vorliegen haben.

Auf ähnliche Art und Weise können wir mit jeder Regel der Form

$$v \mapsto v',$$

wobei v und v' Variablen sind, umgehen.

Betrachten wir schließlich eine Regel der Form

$$v \mapsto w \equiv v_0 v_1 \ldots v_{i-1},$$

wobei $v, v_0, v_1, \ldots, v_{i-1}$ Variablen sind. Wir ersetzen diese eine Regel durch die folgende Liste von $i-1$ Regeln:

$$\begin{array}{rl} v \mapsto & v_0 x_0 \\ x_0 \mapsto & v_1 x_1 \\ \vdots & \\ x_{i-4} \mapsto & v_{i-3} x_{i-3} \\ x_{i-3} \mapsto & v_{i-2} v_{i-1} \end{array}$$

Hierbei sind $x_0, x_1, \ldots, x_{i-3}$ eigens für diesen Zweck neu eingeführte Variablen.

Es ist leicht zu sehen, dass die neue Grammatik dieselbe Sprache generiert wie die alte. Offensichtlich ist die neue Sprache in Chomskyscher Normalform. □

In Analogie zu den regulären Sprachen und den Automaten wollen wir nun eine Klasse von verallgemeinerten Automaten definieren, die genau die kontextfreien Sprachen realisieren. Diese Automaten sind die "Kellerautomaten", die es in "deterministischer" und "nichtdeterministischer" Variante gibt. Sie besitzen einen zusätzlichen Ablagestapel, wo Worte notiert werden dürfen; der Begriff "Ablagestapel" soll dabei suggerieren, dass in jedem "Rechenschritt" nur der *erste* ("oberste") Buchstabe aus dem Ablagestapel ersetzt, entfernt oder hinzugefügt werden darf und fr das weitere Vorgehen eine Rolle spielt.

Ein *(deterministischer) Kellerautomat* ist ein 6–Tupel $(Q, \Sigma, \Gamma, \delta, q_0, F)$, wobei Folgendes gilt:

(1) Q ist eine (nichtleere) Menge (die Menge der *Zustände*).

(2) Σ ist eine (nichtleere) Menge (das *Eingabealphabet*; die Elemente von Σ heißen *Eingabesymbole*).

(3) Γ ist eine (nichtleere) Menge (das *Stapelalphabet*; die Elemente von Γ heißen *Stapelsymbole*).

(4) $\delta : Q \times \Sigma \times \Gamma^* \to Q \times \Gamma^*$ ist eine Funktion (die *Übergangsfunktion*), so dass für alle $q, q' \in Q, x \in \Sigma$ und $w, w' \in \Gamma^*$ mit $\delta(q, x, w) = (q', w')$ gilt: entweder $w' = w$ oder $w' = yw$ für ein $y \in \Gamma$ oder $w = yw'$ für ein $y \in \Gamma$ oder $w = y\overline{w}, w' = y'\overline{w}$ für geeignete $y, y' \in \Gamma$ und $\overline{w} \in \Gamma^*$.[2] Dabei soll $\delta(q, x, w)$ nur von q, x und dem ersten Symbol von w abhängen, d.h. für alle $q \in Q$, $x \in \Sigma$ und w, $w' \in \Gamma^*$ gilt: wenn entweder $w = \epsilon = w'$ oder wenn $w = \gamma\bar{w}$ und $w' = \gamma\bar{w}'$ für geeignete $\gamma \in \Gamma$ und $\bar{w}$, $\bar{w}' \in \Gamma^*$, dann ist $\delta(q, x, w) = \delta(q, x, w')$.

(5) $q_0 \in Q$ ist der *Startzustand*.

(6) $F \subset Q$ ist die Menge der *Akzeptanzzustände*.

Sei $K = (Q, \Sigma, \Gamma, \delta, q_0, F)$ ein Kellerautomat. Sei

$$w = x_0 x_1 \ldots x_{l-1} \in \Sigma^*$$

ein Wort über dem Eingabealphabet, d.h. $x_i \in \Sigma$ für alle $i < l$. Wir sagen dann, dass K das Wort w akzeptiert gdw. eine Folge

$$q^0, q^1, \ldots, q^l$$

von Zuständen (d.h. $q^i \in a$ für alle $i \leq l$) und eine Folge

$$w^0, w^1, \ldots, w^l$$

[2] D.h. w' entsteht aus w durch Anfügen, Entfernen oder Ersetzen des vordersten Symbols.

von Worten über dem Stapelalphabet (d.h. $w^i \in \Gamma^*$ für alle $i \leq l$) existiert, so dass gilt:

(a) $q^0 = q_0$ und $w^0 = \varepsilon$,

(b) $(q^{i+1}, w^{i+1}) = \delta(q^i, x_i, w^i)$ für $i < l$, und

(c) $q^l \in F$.

Wir wollen die Wirkungsweise eines Kellerautomaten an einem Beispiel betrachten. Wir wollen einen Kellerautomaten $K = (Q, \Sigma, \Gamma, \delta, q_0, F)$ bauen, der genau die Worte

$$0^n 1^n = \underbrace{00\ldots0}_{n \text{ viele}}\underbrace{11\ldots1}_{n \text{ viele}}$$

für $n \in \mathbb{N}$ akzeptiert.

Hierzu sei $Q = \{q_0, q_1, q_2, q_3\}, \Sigma = \{0, 1\}$ und $\Gamma = \{0, \#\}$. Die Idee ist, den Ablagestapel zu benutzen um Buch darüber zu führen, wie viele Nullen bislang eingelesen wurden. Beim Einlesen der Einser wird der Ablagestapel dann geräumt. Das Symbol # dient zur Markierung. Die Funktion $\delta : Q \times \Sigma \times \Gamma^* \to Q \times \Gamma^*$ sei durch folgende Tafel gegeben. Hierbei sei w ein beliebiges Wort aus Γ^*.

$q_0, 0, w$	$q_1, \#w$
$q_0, 1, w$	q_4, w
$q_1, 0, w$	$q_1, 0w$
$q_1, 1, \#w$	q_3, w
$q_1, 1, 0w$	q_2, w
$q_2, 1, 0w$	q_2, w
$q_2, 1, \#w$	q_3, w
$q_3, 0, w$	q_4, w
$q_3, 1, w$	q_4, w

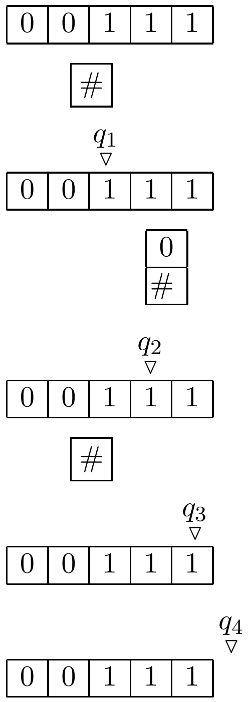
Außerdem sei immer $\delta(q_4, x, w) = (q_4, w)$. Wenn wir nun $F = \{q_0, q_3\}$ setzen, dann erkennt man, dass A ist wie gewünscht.

Wir wollen die Wirkungsweise dieses Automaten am Beispiel der Eingabe 00111 illustrieren.

q_0
$\triangledown$

0	0	1	1	1

q_1
$\triangledown$



Um Automaten zu erhalten, die beliebige kontextfreie Sprachen realisieren, benötigen wir nichtdeterministische Kellerautomaten.

Ein nichtdeterministischer Kellerautomat ist ein G–Tupel $(Q, \Sigma, \Gamma, \delta, q_0, F)$, wobei Folgendes gilt.

(1) Q ist eine (nichtleere) Menge (die Menge der *Zustände*).

(2) Σ ist eine (nichtleere) Menge (das *Eingabealphabet*; die Elemente von Σ heißen Eingabesymbole).

(3) Γ ist eine (nichtleere) Menge (das *Stapelalphabet*; die Elemente von Γ heißen *Stapelsymbole*).

(4) $\delta : Q \times \Sigma \cup \{\varepsilon\} \times \Gamma^* \to \mathcal{P}(Q \times \Gamma^*)$ ist eine Funktion (die *Übergangsfunktion*), so dass für alle $q, q' \in Q, x \in \Sigma \cup \{\varepsilon\}$ und $w, w' \in \Gamma^*$ mit $(q', w') \in \delta(q, x, w)$ gilt: entweder $w' = w$ oder $w' = yw$ für ein $y \in \Gamma$ oder $w = yw'$ für ein $y \in \Gamma$ oder $w = y\overline{w}, w' = y'\overline{w}$ für geeignete $y, y' \in \Gamma$ und $\overline{w} \in \Gamma^*$. Dabei soll $\delta(q, x, w)$ nur von q, x und dem ersten Symbol von w abhängen, d.h. für alle $q \in Q$, $x \in \Sigma$ und w, $w' \in \Gamma^*$ gilt: wenn entweder $w = \epsilon = w'$ oder wenn $w = \gamma\bar{w}$ und $w' = \gamma\bar{w}'$ für geeignete $\gamma \in \Gamma$ und $\bar{w}$, $\bar{w}' \in \Gamma^*$, dann ist $\delta(q, x, w) = \delta(q, x, w')$.

(5) $q_0 \in Q$ ist der *Startzustand.*

(6) $F \subset Q$ ist die Menge der *Akzeptanzzustände.*

Jeder (deterministische) Kellerautomat $K = (Q, \Sigma, \Gamma, \delta, q_0, F)$ kann als nichtdeterministischer Kellerautomat aufgefasst werden, indem wir δ durch $\delta' : Q \times \Sigma \cup \{\epsilon\} \times \Gamma^* \to \mathcal{P}(Q \times \Gamma^*)$ ersetzen, wobei $\delta'(q, x, w) = \{\delta(q, z, w)\}$ und $\delta'(q, x, \epsilon) = \{(q, x)\}$ für $(q, x, w) \in Q \times \Sigma \times \Gamma^*$. Dann ist $(Q, \Sigma, \Gamma, \delta, q_0, F)$ ein nichtdeterministischer Kellerautomat mit derselben Wirkungsweise wie K.

In der Regel erlauben aber nichtdeterministische Automaten "Wahlfreiheiten" in der Berechnung.

Sei $K = (Q, \Sigma, \Gamma, \delta, q_0, F)$ ein nichtdeterministischer Kellerautomat. Sei

$$w = x_0' x_1' \dots x_{S-1}' \in \Sigma^*$$

ein Wort über dem Eingabealphabet, d.h. $x_i' \in \Sigma$ für alle $i < S$. Wir sagen dann, dass K das Wort w *akzeptiert* gdw. w geschrieben werden kann als $w = x_0 x_1 \dots x_{l-1}$, wobei $x_i \in \Sigma \cup \{\varepsilon\}$ für alle $i < l$, und wenn eine Folge

$$q^0, q^1, \dots, q^l$$

von Zuständen und eine Folge

$$w^0, w^1, \dots, w^l$$

von Worten über dem Stapelalphabet existiert, so dass gilt:

(a) $q^0 = q_0$ und $w_0 = \varepsilon$,

(b) $(q^{i+1}, w^{i+1}) \in \delta(q^i, x_i, w^i)$ für $i < l$, und

(c) $q^l \in F$.

Jedes derartige Folgenpaar $(q^0, q^1, \dots, q^l), (w^0, w^1, \dots, w^l)$ wird als eine *Berechnung (von A) bei Eingabe von w* bezeichnet. Nichtdeterministische Kellerautomaten unterscheiden sich also von deterministischen durch zwei wesentliche Dinge. Erstens können nichtdeterministische Kellerautomaten das Eingabewort w so auffassen, dass an beliebigen Stellen beliebig viele leere Symbole ε eingefügt sind; anders gesagt bedeutet dies, dass im "Keller" (oder "Stapel") gearbeitet werden kann, ohne dass das Wort w tatsächlich weitergelesen wird. Zum Anderen "rät" der nichtdeterministische Kellerautomat im jeweiligen Zustand und bei Lesen des nächsten Symbols aus $\Sigma \cup \{\varepsilon\}$, wie er fortschreiten soll.

Es ist nicht schwierig, einen nichtdeterministischen Kellerautomaten zu bauen, der genau die Worte der Gestalt

$$0^n 1^m 2^p$$

mit $n = m$ oder $n = p$ akzeptiert. Nichtdeterminismus wird auch in der Theorie der Turing–Maschinen eine große Rolle spielen.

Sei Σ ein Alphabet, sei $B \subset \Sigma^*$ eine Sprache und sei K ein nichtdeterministischer Kellerautomat. Wir sagen dann, dass K die Sprache B *erkennt* gdw. für alle $w \in \Sigma^*$ gilt: $w \in B$ gdw. K das Wort w akzeptiert. Wir zeigen nun:

Satz 2.4 *Sei Σ ein Alphabet und sei $B \subset \Sigma^*$ eine Sprache. Dann ist B kontextfrei gdw. es einen nichtdeterministischen Kellerautomaten gibt, der B erkennt.*

Beweis: Wir zeigen zunächst, dass jede kontextfreie Sprache von einem nichtdeterministischen Kellerautomaten erkannt wird. Hierzu sei $B \subset \Sigma^*$ kontextfrei. B ist damit durch eine kontextfreie Grammatik (V, Σ, R, s), etwa in Chomskyscher Normalform, gegeben. Die Idee ist einfach, einen nichtdeterministischen Kellerautomaten $K = (Q, \Sigma, \Gamma, \delta, q_0, F)$ so zu bauen, dass K im Keller eine Generierung der Eingabe mit Hilfe der Regeln R "rät" und sodann nachrechnet, dass die Eingabe mit dem, was im Keller generiert wurde, übereinstimmt.

Wir beschreiben die Wirkungsweise von K informell und überlassen es der Leserin, die formale Definition von K niederzuschreiben.

Das Abarbeiten der Eingabe durch K erfolgt in mehreren Runden, wobei jede Runde aus mehreren Rechenschritten bestehen kann. Sei

$$w = x_0 x_1 \dots x_{l-1} \in \Sigma^*$$

die gegebene Eingabe.

Wenn $w \equiv \epsilon$, dann akzeptiere K dieses Wort gdw. $\epsilon \in B$. Wenn w nicht das leere Wort ist, dann gehe K weiter wie folgt vor.

In der ersten Runde schreibt K lediglich das Startsymbol $S \in V$, gefolgt von einem Markierungssymbol, etwa #, in den Stapel.

Werde eine neue Runde begonnen. Dann enthält der Keller de facto ein *Endsegment* einer Folge w^* von Variablen und Terminalen, die mit Hilfe von R generiert werden kann, gefolgt vom Markierungssymbol #. Weiterhin seien die Symbole $x_0, x_1, \dots x_{k-1}$ bereits eingelesen, wobei $k \leq l$. Dann geht A weiter wie folgt vor.

1. Fall. w^* beginnt mit einem Terminal, i.e. mit einem Symbol aus Σ, etwa mit einem x.
Dann verwirft A die Eingabe, es sei denn $k < l$ und $x_k = x$. Im letzteren Fall löscht A das x im Keller und sieht das $x = x_k$ der Eingabe als gelesen an.

2. Fall. w^* beginnt mit einer Variablen, i.e. mit einem Symbol aus V.
Dann "rät" A eine der Regeln aus R, die v als Argument besitzt, und ersetzt das v im Keller durch die Folge, die durch Anwendung dieser Regel aus v entsteht. Es gelten weiterhin lediglich die Symbole $x_0, x_1, \ldots, x_{k-1}$ der Eingabe als eingelesen.

3. Fall. w^* besteht nur (noch) aus dem Markierungssymbol $\#$.
Dann verwirft A die Eingabe, es sei denn $k = l$, so dass genau in diesem Moment die gesamte Eingabe eingelesen ist.

Der Prozess des Ratens in Fall 2 bedarf einer näheren Erklärung. Dieses "Raten" ist genauer so gemeint, dass die Übergangsfunktion δ von A so eingerichtet wird, dass für *jede* der Regeln aus R, die v als Argument besitzt, das Umräumen des Kellers, das dadurch entsteht, dass v durch die durch die Regelanwendung gegebene Folge ersetzt wird, als Wirkungsweise von δ erlaubt sein soll.

Es ist nun leicht einzusehen, dass ein solcher nichtdeterministischer Kellerautomat A die gegebene kontextfreie Sprache B erkennt.

Wir wollen nun umgekehrt zeigen, dass jede Sprache, die von einem nichtdeterministischen Kellerautomaten erkannt wird, kontextfrei ist.

Sei also $B \subset \Sigma^*$ eine Sprache, und sei $A = (Q, \Sigma, \Gamma, \delta, q_0, F)$ ein nichtdeterministischer Kellerautomat, so dass A die Sprache B erkennt. Es ist leicht zu sehen, dass A notfalls so umgebaut werden kann, dass Folgendes gilt:

(1) $F = \{q_\infty\}$ für ein (eindeutiges) $q_\infty \in Q$ (mit $q_\infty \neq q_0$).

(2) Wenn A ein Wort $w \in \Sigma^*$ akzeptiert, dann enthält der Keller am Ende jeder Berechnung, die w akzeptiert, das leere Wort.

(3) Wenn $(q', w') \in \delta(q, x, w)$, dann gilt *entweder* $w' = yw$ für ein $y \in \Gamma$ *oder* $w = yw'$ für ein $y \in \Gamma$.

Wir konstruieren nun eine kontextfreie Grammatik (V, Σ, R, s), die genau die Wrter aus B erzeugt.

Hierzu setzen wir zunächst

$$V = \{A_{p,q} : p, q \in Q\}$$

und $s = A_{p_0,p_\infty}$. Es bleibt, die Regel in R anzugeben.

Seien $p, q, r, s \in Q, t \in \Gamma$ und $a, b \in \Sigma \cup \{\varepsilon\}$ so, dass fr beliebige $w \in \Gamma^*$

$$(r, tw) \in \delta(p, a, w)$$

und

$$(q, w) \in \delta(s, b, tw),$$

dann enthalte R die Regel

$$A_{p,q} \mapsto aA_{r,s}b.$$

Seien $p, q, r \in Q$. Dann enthalte R die Regel

$$A_{p,q} \mapsto A_{p,r}A_{r,q}.$$

Schließlich enthalte R für jedes $p \in Q$ die Regel

$$A_{p,p} \mapsto \varepsilon.$$

Dies seien alle Regeln von R.

Es genügt nun offensichtlich, die folgenden beiden Behauptungen zu zeigen.

Behauptung 1. Seien $p, q \in Q$. Angenommen, ausgehend von der Variablen $A_{p,q}$ als "Startsymbol" kann mit Hilfe der Regeln von R das Wort $w \in \Sigma^*$ generiert werden. Dann gibt es eine "Berechnung" von A bei Eingabe von w, die im Zustand p beginnt und im Zustand q endet, so dass bei Beginn und Ende der Berechnung jeweils das leere Wort ε im Keller notiert ist und so dass durch die Berechnung das Wort w abgearbeitet wurde.

Behauptung 2. Seien $p, q \in Q$ und sei $w \in \Sigma^*$. Angenommen, es gibt eine "Berechnung" von A bei Eingabe von w, die im Zustand p beginnt und im Zustand q endet, so dass bei Beginn und Ende der Berechnung jeweils das leere Wort ε im Keller notiert ist und so dass durch die Berechnung das Wort w abgearbeitet wurde. Dann kann aus der Variablen $A_{p,q}$ als "Startsymbol" mit Hilfe der Regeln von R das Wort $w \in \Sigma^*$ generiert werden.

Wir zeigen beide Behauptungen durch Induktion.

Beweis der Behauptung 1: In diesem Falle führt eine Induktion nach der Länge der Ableitung von w aus $A_{p,q}$ zum Ziel.

Die Behauptung 1 gilt zunächst, falls die Ableitung die Länge 1 besitzt. Die einzigen derartigen Ableitungen entstehen durch einmalige Anwendung einer Regel der Gestalt

$$A_{p,p} \mapsto \varepsilon.$$

Sei nun Behauptung 1 gezeigt für Ableitungen der Länge $m \geq 1$.

Sei eine Ableitung von w aus $A_{p,q}$ der Länge $m + 1$ gegeben.

1. Fall. Der erste Schritt dieser Ableitung erfolgt durch Anwendung der Regel

$$A_{p,q} \mapsto aA_{r,s}b,$$

wobei $r, s \in Q$ und $a, b \in \Sigma \cup \{\varepsilon\}$. Damit gilt $w = a\overline{w}b$ für ein $\overline{w} \in \Sigma^*$, und nach Induktionsannahme gibt es eine Berechnung von A bei Eingabe von $\overline{w}$, die im Zustand r beginnt, im Zustand s endet, so das bei Beginn und Ende der Berechnung jeweils ε im Keller notiert ist und so dass durch die Berechnung das Wort $\overline{w}$ abgearbeitet wurde. Wir wissen auch, dass es ein $t \in \Gamma$ gibt, so dass

$$(r, t) \in \delta(p, a, \varepsilon)$$

und

$$(q, \varepsilon) \in \delta(s, b, t).$$

Natürlich gilt nun auch, dass es eine Berechnung von A bei Eingabe von $\overline{w}$ gibt, die im Zustand r beginnt, im Zustand s endet, so dass bei Beginn und Ende der Berechnung jeweils t (an Stelle von ε) im Keller notiert ist und so, dass durch die Berechnung das Wort $\overline{w}$ abgearbeitet wurde. Damit gibt es eine Berechnung von A bei Eingabe von $a\overline{w}b$, die im Zustand p beginnt, im Zustand q endet, so dass bei Beginn und Ende der Berechnung jeweils ε im Keller notiert ist und so dass durch die Berechnung w abgearbeitet wurde.

2. Fall. Der erste Schritt dieser Ableitung erfolgt durch Anwendung der Regel

$$A_{p,q} \mapsto A_{p,r}A_{r,q},$$

wobei $r \in Q$. Der weitere Verlauf der Ableitung generiere $\overline{w} \in \Sigma^*$ aus $A_{p,r}$ und $\overline{w}' \in \Sigma^*$ aus $A_{r,q}$, so dass $w = \overline{w}\ \overline{w}'$. Mit Hilfe der Induktionsvoraussetzung sieht man dann ähnlich wie im 1. Fall, dass die gewünschte Aussage zutrifft.

Beweis von Behauptung 2: In diesem Falle benutzen wir Induktion nach der Länge der Berechnung.

Behauptung 2 gilt zunächst offensichtlich für Berechnungen der Länge 0. Dann muss nämlich $p = q$ sein und $w = \varepsilon$. R besitzt aber die Regel

$$A_{p,p} \mapsto \varepsilon.$$

Sei nun Behauptung 2 gezeigt für alle Berechnungen der Länge m.

Sei eine Berechnung der Länge $m + 1$ gegeben.

1. Fall. Außer zu Beginn und Ende der Berechnung ist der Keller während der Berechnung niemals leer.

In diesem Falle existiert ein $t \in \Gamma$, so dass w geschrieben werden kann als $w = a\overline{w}b$, wobei $a, b \in \Sigma \cup \{\varepsilon\}$, und so dass

$$(r, t) \in \delta(p, a, \varepsilon)$$

und

$$(q, \varepsilon) \in \delta(s, b, t)$$

für geeignet $r, s \in Q$. (r ist der zweite Zustand, der von der fraglichen Berechnung durchlaufen wird, s der vorletzte; t ist das Symbol, das als erstes im Stapel eingetragen und als letztes herausgenommen wird.) Es ist leicht zu sehen, dass dann eine Berechnung von A bei Eingabe von $\overline{w}$ existiert, die im Zustand r beginnt und im Zustand s endet, so dass bei Beginn und Ende der Berechnung ε im Keller notiert ist und so dass durch die Berechnung das Wort $\overline{w}$ abgearbeitet wird. Nach Induktionsvoraussetzung kann dann aus der Variablen $A_{r,s}$ das Wort $\overline{w}$ generiert werden. Unter Zuhilfenahme der verfügbaren Regel

$$A_{p,q} \mapsto aA_{r,s}b$$

kann dann aber auch das Wort $w = a\overline{w}b$ generiert werden.

2. Fall. Es gibt einen Moment echt zwischen Beginn und Ende der Berechnung, in dem der Keller leer ist.

Wir können dann w schreiben als $w = \overline{w}\,\overline{w}'$, so dass ein $r \in Q$ existiert mit: Es gibt eine Berechnung von A bei Eingabe von $\overline{w}$, die im Zustand p beginnt und im Zustand r endet, so dass bei Beginn und Ende der Berechnung jeweils ε im Keller ist und die Berechnung $\overline{w}$ abarbeitet, und es gibt eine Berechnung von A bei Eingabe von $\overline{w}'$ die im Zustand r beginnt und im Zustand q endet, so dass bei Beginn und Ende der Berechnung jeweils ε im Keller ist die Berechnung $\overline{w}'$ abarbeitet. Nach Induktionsvoraussetzung ist dann ausgehend von $A_{p,r}$ das Wort $\overline{w}$ und ausgehend von $A_{r,q}$ das Wort $\overline{w}'$ generierbar. Mit Hilfe der in R verfügbaren Regel

$$A_{p,q} \mapsto A_{p,r}A_{r,q}$$

ist dann aber $w = \overline{w}\,\overline{w}'$ generierbar.

Kapitel 3

Aussagenlogik

Beweise, wie sie in der Mathematik und Informatik geliefert werden, können als "Berechnungen" aufgefaßt werden. Die einfachsten Beweise sind Beweise in der Aussagenlogik. Beispiel:

$$\begin{array}{l} \text{Die Werse ist ein Berg oder ein Fluss.} \\ \text{Die Werse ist kein Berg.} \\ \hline \text{Die Werse ist ein Fluss.} \end{array}$$

Mit $A_0 =$ "Die Werse ist ein Berg" und $A_1 =$ "Die Werse ist ein Fluss" sieht dieser "Beweis" (dieser logische Schluss) so aus:

$$\begin{array}{l} A_0 \text{ oder } A_1 \\ \text{nicht } A_0 \\ \hline A_1 \end{array}$$

Die logischen *Junktoren* sind Zeichen für: *nicht, und, oder, wenn* ... , *dann* ... , *genau dann, wenn.*

"$\neg$" steht für "nicht",
"$\wedge$" steht für "und",
"$\vee$" steht für "oder",
"$\rightarrow$" steht für "wenn ... , dann ... ",
"$\leftrightarrow$" steht für "genau dann, wenn".

Unser logischer Schluss sieht dann so aus:

$$\begin{array}{l} A_0 \vee A_1 \\ \neg A_0 \\ \hline A_1 \end{array}$$

Dieser Schluss ist logisch, da es auf die Bedeutung der darin vorkommenden Aussagenvariablen nicht ankommt.

Die *Symbole* der Aussagenlogik sind die folgenden:

Klammern: (und)
Junktoren: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$
Aussagenvariablen: $A_0, A_1, A_2 \ldots$

Ein *Ausdruck* ist eine endliche Folge von Symbolen. Z.B. ist $(\wedge A_3 \leftrightarrow)$ ein Ausdruck. Nicht alle Ausdrücke sind "Formeln", d.h. haben Bedeutung.

Welche Ausdrücke sind Formeln? Wir wollen:

(a) Jede Aussagenvariable ist eine Formel.

(b) Wenn φ und ψ Formeln sind, dann sind auch $\neg\varphi, (\varphi \wedge \psi), (\varphi \vee \psi), (\varphi \rightarrow \psi)$ und $(\varphi \leftrightarrow \psi)$ Formeln.

(c) Kein Ausdruck ist eine Formel, der es nicht auf Grund von (a) oder (b) sein muss.

Wie lässt sich dies mathematisch präziser fassen? Eine *Formel* der Aussagenlogik ist ein Ausdruck, der in jeder Menge M von Ausdrücken enthalten ist, so dass

(a)' jede Aussagenvariable in M enthalten ist, und

(b)' wenn φ und ψ in M enthalten sind, dann auch $\neg\varphi, (\varphi \wedge \psi), (\varphi \vee \psi), (\varphi \rightarrow \psi)$ und $(\varphi \leftrightarrow \psi)$.

Wenn wir (nur für diesen Zweck!) eine Menge M von Ausdrücken "gut" nennen genau dann, wenn M (a)' und (b)' erfüllt, dann gilt also: Ein Ausdruck ist eine Formel genau dann, wenn er im Durchschnitt aller "guten" Mengen von Ausdrücken liegt.

Die Aussagenvariablen heißen auch *atomare Formeln*, die übrigen Formeln *zusammengesetzte Formeln*. Eine zusammengesetzte Formel ist entweder Negation (d.h. von der Gestalt $\neg\varphi$), oder Konjunktion (von der Gestalt $(\varphi \wedge \psi)$), oder Disjunktion (von der Gestalt $(\varphi \vee \psi)$), oder Konditional (von der Gestalt $(\varphi \rightarrow \psi)$) oder Bikonditional (von der Gestalt $(\varphi \leftrightarrow \psi)$).

Die Definition des Formelbegriffs ist Beispiel für eine *rekursive Definition*. Es gilt das *Induktionsprinzip* für Formeln: Sei E eine beliebige Eigenschaft. Angenommen, E gilt für alle atomaren Formeln. Sei weiterhin Folgendes angenommen: wenn E auf die Ausdrücke φ und ψ zutrifft, dann gilt E auch für $\neg\varphi, (\varphi \wedge \psi), (\varphi \vee \psi), (\varphi \rightarrow \psi)$ und $(\varphi \leftrightarrow \psi)$. Dann gilt E für alle Formeln.

Hier ist ein Beispiel. Wir zeigen mit Hilfe des Induktionsprinzips, dass jede Formel die gleiche Anzahl von linken und rechten Klammern hat. Diese Aussage gilt nämlich offensichtlich für alle atomaren Formeln, und sie vererbt sich von φ und ψ weiter auf $\neg\varphi, (\varphi \wedge \psi), (\varphi \vee \psi), (\varphi \rightarrow \psi)$ und $(\varphi \leftrightarrow \psi)$. Also gilt die Aussage für alle Formeln.

Warum gilt das Induktionsprinzip? Sei M die Menge aller Ausdrücke, auf die E zutrifft. Dann gilt für M nach der im Induktionsprinzip für E formulierten Voraussetzung (a)' und (b)'. Also ist (nach Definition dessen, was es heißt, eine Formel zu sein) die Menge aller Formeln eine Teilmenge von M; d.h., auf jede Formel trifft E zu!

Rekursive Definitionen sind in der Mathematik sehr häufig. Für sie alle gibt es ein entsprechendes Induktionsprinzip. Das Induktionsprinzip für $\mathbb{N}$ ist ein weiteres Beispiel. Wir werden im Folgenden viele Beispiele zur Rekursion und Induktion sehen.

Wir wollen nun definieren, was es heißt, dass eine Formel logisch aus einer anderen Formel folgt. Hierzu definieren wir zunächst den Begriff der Belegung; eine Belegung β ordnet jeder Formel φ einen "Wahrheitswert" $\beta(\varphi) \in \{0, 1\}$ zu. Dabei lesen wir $\beta(\varphi) = 0$ auch als "φ ist falsch" und $\beta(\varphi) = 1$ als "φ ist wahr".

Sei F_0 die Menge aller atomaren Formeln (d.h. $F_0 = \{A_0, A_1, A_2, \dots\}$), und sei $F \supset F_0$ die Menge aller Formeln. Sei $\overline{\beta} : F_0 \rightarrow \{0, 1\}$ eine beliebige Funktion, die allen atomaren Formeln "Wahrheitswerte" zuordnet. Wir definieren dann die *zu $\overline{\beta}$ gehörige Belegung* $\beta : F \rightarrow \{0, 1\}$ rekursiv wie folgt:

(1) $\beta(\varphi) = \overline{\beta}(\varphi)$ für alle $\varphi \in F_0$

und für alle $\varphi, \psi \in F$:

(2) $\beta(\neg\varphi) = 1 - \beta(\varphi)$,

(3) $\beta((\varphi \wedge \psi)) = \beta(\varphi) \cdot \beta(\psi) = \min\{\beta(\varphi), \beta(\psi)\}$

(4) $\beta((\varphi \vee \psi)) = \max\{\beta(\varphi), \beta(\psi)\}$

(5) $\beta((\varphi \rightarrow \psi)) = \max\{1 - \beta(\varphi), \beta(\psi)\}$

(6) $\beta((\varphi \leftrightarrow \psi)) = \beta((\varphi \rightarrow \psi)) \cdot \beta((\psi \rightarrow \varphi))$

Es gilt also Folgendes: $\neg\varphi$ ist wahr gdw.[1] φ falsch ist; $(\varphi \wedge \psi)$ ist wahr gdw. φ und ψ beide wahr sind; $(\varphi \vee \psi)$ ist wahr gdw. φ oder ψ wahr ist (im nicht ausschließenden Sinne); $(\varphi \to \psi)$ ist wahr gdw. $\neg\varphi$ oder ψ wahr ist; $(\varphi \leftrightarrow \psi)$ ist wahr gdw. φ und ψ dieselben Wahrheitswerte besitzen.

Zu jedem $\overline{\beta} : F_0 \to \{0,1\}$ ist dadurch eindeutig eine zugehörige Belegung $\beta : F \to \{0,1\}$ definiert.

Wir nennen $\beta(\varphi)$ auch den *zu $\overline{\beta}$ gehörigen Wahrheitswert von φ*. Für eine gegebene Formel errechnet sich der Wahrheitswert am leichtesten mit Hilfe einer "Wahrheitstafel". Sei etwa die Formel $(A_0 \to (\neg A_1 \to A_0))$ gegeben. Sei $\overline{\beta}(A_0) = 0$, $\overline{\beta}(A_1) = 1$. Dann ergibt sich für den zugehörigen Wahrheitswert $\beta((A_0 \to (\neg A_1 \to A_0)))$:

A_0	A_1	$(A_0$	$\to$	$(\neg$	A_1	$\to$	$A_0))$
0	1		1	0		1	

D.h. $\beta((A_0 \to (\neg A_1 \to A_0))) = 1$. Offensichtlich hängt der Wahrheitwert $\beta(\varphi)$ einer Formel φ nur von den (endlich vielen) Werten $\overline{\beta}(A_n)$ ab, für die die atomare Formel A_n in φ vorkommt. In unserem Falle errechnen sich die zu verschiedenen $\overline{\beta}$ gehörigen Wahrheitswerte von $(A_0 \to (\neg A_1 \to A_0))$ wie folgt:

A_0	A_1	$(A_0$	$\to$	$(\neg$	A_1	$\to$	$A_0))$
0	0		1	1		0	
0	1		1	0		1	
1	0		1	1		1	
1	1		1	0		1	

Sei $\overline{\beta} : F_0 \to \{0,1\}$. Wir sagen, dass $\overline{\beta}$ die Formel φ *erfüllt* gdw. $\beta(\varphi) = 1$, wobei β die zu $\overline{\beta}$ gehörige Belegung ist. Sei Σ eine Menge von Formeln. (D.h. $\Sigma \subset F$.) Wir sagen, dass $\overline{\beta}$ die Formelmenge Σ *erfüllt* gdw. $\beta(\varphi) = 1$ für alle $\varphi \in \Sigma$, wobei β die zu $\overline{\beta}$ gehörige Belegung ist.

Definition 1.1 *Sei $\Sigma \cup \{\varphi\}$ eine Menge von Formeln. Σ* impliziert tautologisch *φ, in Zeichen $\Sigma \models \varphi$, gdw. für alle $\overline{\beta} : F_0 \to \{0,1\}$ Folgendes gilt: wenn $\overline{\beta}$ Σ erfüllt, dann erfüllt $\overline{\beta}$ auch φ. Anstelle von $\emptyset \models \varphi$ schreiben wir auch $\models \varphi$; in diesem Falle nennen wir φ eine* Tautologie.

[1] "gdw." steht für "genau dann wenn"

Die oben betrachtete Formel $(A_0 \to (\neg A_1 \to A_0))$ ist also eine Tautologie. Hier sind einige Beispiele für $\Sigma \models \varphi$:

$$\begin{aligned}
&\{\neg A_1\} \models A_1 \to A_0, \\
&\{A_3 \to A_5, A_5 \to A_3\} \models A_3 \leftrightarrow A_5, \\
&\{\neg A_0 \vee A_1\} \models A_0 \to A_1.
\end{aligned}$$

Die Menge Σ in der obigen Definition muss übrigens nicht endlich sein. Wir schreiben auch $\psi \models \varphi$ anstelle von $\{\psi\} \models \varphi$. ψ und φ heißen *tautologisch äquivalent* gdw. $\psi \models \varphi$ und $\varphi \models \psi$ gelten.

Wir haben oben ein Verfahren kennengelernt, nämlich die Verwendung von "Wahrheitstafeln", mit dessen Hilfe wir *entscheiden* können, ob eine gegebene aussagenlogische Formel eine Tautologie ist oder nicht. Wir werden später der Frage nachgehen, wie *schwierig* diese Entscheidung ist.

Sei $J \subset \{\neg, \wedge, \vee, \to, \leftrightarrow\}$ eine Menge von Junktoren. J heißt *vollständig* gdw. es zu jeder Formel φ eine tautologisch äquivalente Formel ψ gibt, in der nur Junktoren aus J vorkommen. Z.B. ist $\{\neg\}$ nicht vollständig. Es gilt aber:

Satz 1.2 *Die folgenden Mengen J von Junktoren sind vollständig.*

(1) $J = \{\neg, \wedge, \vee\}$,

(2) $J = \{\neg, \wedge\}$,

(3) $J = \{\neg, \vee\}$,

(4) $J = \{\neg, \to\}$.

Beweis von (1): Wir konstruieren rekursiv eine Funktion $\Phi : F \to F$, die jeder Formel φ eine tautologisch äquivalente Formel $\Phi(\varphi)$ zuordnet, in der höchstens die Junktoren $\neg, \wedge, \vee$ vorkommen.

Wir setzen $\Phi(\varphi) = \varphi$ für jedes atomare φ. Seien $\Phi(\varphi)$ und $\Phi(\varphi')$ bereits definiert. Dann setzen wir

$$\begin{aligned}
&\Phi(\neg\varphi) = \neg\Phi(\varphi), \\
&\Phi((\varphi \wedge \varphi')) = (\Phi(\varphi) \wedge \Phi(\varphi')), \\
&\Phi((\varphi \vee \varphi')) = (\Phi(\varphi) \vee \Phi(\varphi')), \\
&\Phi((\varphi \to \varphi')) = \neg\Phi(\varphi) \vee \Phi(\varphi'), \\
&\Phi((\varphi \leftrightarrow \varphi')) = \Phi((\varphi \to \varphi')) \wedge \Phi((\varphi' \to \varphi)).
\end{aligned}$$

Damit ist $\Phi : F \to F$ (durch "Rekursion längs der Formelkomplexität") wohldefiniert.

Man überlegt sich leicht induktiv, dass φ und $\Phi(\varphi)$ immer tautologisch äquivalent sind. Für atomares φ ist das trivial. Sei nun etwa φ ein Konditional, etwa φ gleich $(\psi \to \psi')$. Dann sind nach Induktionsvoraussetzung ψ und $\Phi(\psi)$ sowie ψ' und $\Phi(\psi')$ jeweils tautologisch äquivalent. Damit sind aber auch $\neg\psi$ und $\neg\Phi(\psi)$ sowie schließlich $(\neg\psi \vee \psi')$ und

$$(\neg\Phi(\psi) \vee \Phi(\varphi') = \Phi((\psi \to \psi')))$$

tautologisch äquivalent. Es sind aber $(\psi \to \psi')$ (d.h. φ) und $(\neg\psi \vee \psi')$ tautologisch äquivalent. $\square$

Im Beweis dieses Satzes sahen wir ein Beispiel für eine Definition durch "Rekursion längs der Formelkomplexität" und ein Beispiel für den Beweis einer Aussage durch "Induktion nach der Formelkomplexität".

Wir beweisen nun den Kompaktheitssatz der Aussagenlogik. Sei Σ eine Menge von Formeln. Σ heißt *erfüllbar* gdw. es ein $\overline{\beta} : F_0 \to \{0,1\}$ gibt, so dass $\overline{\beta}$ jedes φ aus Σ erfüllt. Σ heißt *endlich* erfüllbar gdw. jede endliche Teilmenge $\overline{\Sigma}$ von Σ erfüllbar ist. Offensichtlich ist jedes erfüllbare Σ auch endlich erfüllbar.

Satz 1.3 Kompaktheitssatz. *Sei Σ eine Menge von Formeln. Wenn Σ endlich erfüllbar ist, dann ist Σ auch erfüllbar.*

Beweis: Sei $(\varphi_n : n \in \mathbb{N})$ eine Aufzählung *aller* Formeln. Eine solche Aufzählung erhält man z.B., indem man im m^{ten} Schritt alle Formeln der Länge $\leq m$ aufzählt, in denen höchstens die Aussagenvariablen $A_0, \dots, A_{m-1}$ vorkommen.

Sei nun Σ endlich erfüllbar. Wir konstruieren nun rekursiv eine Folge $(\Sigma_n : n \in \mathbb{N})$ von Formelmengen Σ_n wie folgt. Setze $\Sigma_0 = \Sigma$. Sei nun Σ_n konstruiert. Setze dann $\Sigma_{n+1} = \Sigma_n \cup \{\varphi_n\}$, falls $\Sigma_n \cup \{\varphi_n\}$ endlich erfüllbar ist; ansonsten setze $\Sigma_{n+1} = \Sigma_n \cup \{\neg\varphi_n\}$. Schließlich sei Σ_∞ die Vereinigung aller Σ_n. Wir zeigen zunächst durch Induktion, dass alle Σ_n endlich erfüllbar sind. Dies gilt nach Annahme für $n = 0$. Sei nun Σ_n endlich erfüllbar. Falls $\Sigma_n \cup \{\varphi_n\}$ endlich erfüllbar ist, dann ist der Induktionsschritt trivial. Sei also o.B.d.A. $\Sigma_n \cup \{\varphi_n\}$ nicht endlich erfüllbar und $\Sigma_{n+1} = \Sigma_n \cup \{\neg\varphi_n\}$. Sei $\overline{\Sigma} \subset \Sigma_n \cup \{\neg\varphi_n\}$ endlich. Sei $\overline{\Sigma}' \subset \Sigma_n \cup \{\varphi_n\}$ endlich und nicht erfüllbar.

Nach Induktionsvoraussetzung existiert ein $\overline{\beta} : F_0 \to \{0,1\}$, das $(\overline{\Sigma} \cup \overline{\Sigma}')\setminus\{\varphi_n, \neg\varphi_n\}$ erfüllt. Sei β die zu $\overline{\beta}$ gehörige Belegung. Da Σ_n endlich erfüllbar und $\overline{\Sigma}'$ nicht erfüllbar ist, muss φ_n in $\overline{\Sigma}'$ enthalten sein und $\beta(\varphi_n) =$

0. Dann gilt aber $\beta(\neg\varphi_n) = 1$ und $\overline{\beta}$ erfüllt $(\overline{\Sigma} \cup \overline{\Sigma}' \cup \{\neg\varphi_n\}) \setminus \{\varphi_n\}$, also auch $\overline{\Sigma}$.

Damit ist nun sofort auch Σ_∞ endlich erfüllbar. Nach Konstruktion haben wir, dass für jede Formel φ entweder φ oder $\neg\varphi$ in Σ_∞ liegt.

Wir definieren nun ein $\overline{\beta} : F_0 \to \{0,1\}$ wie folgt. Wir setzen $\overline{\beta}(A_n) = 1$ gdw. A_n in Σ_∞ enthalten ist.

Sei $\beta : F \to \{0,1\}$ die zu $\overline{\beta}$ gehörige Belegung. Wir zeigen jetzt induktiv, dass $\beta(\varphi) = 1$ gdw. φ in Σ_∞ enthalten ist. Dies gilt zunächst für atomare Formeln φ. Sei die Aussage nun für φ gezeigt. Wir zeigen dann, dass sie auch für $\neg\varphi$ gilt. Es gilt aber $\beta(\neg\varphi) = 1$ gdw. $\beta(\varphi) = 0$ gdw. φ nicht in Σ_∞ enthalten ist gdw. $\neg\varphi$ in Σ_∞ enthalten ist. Sei die Aussage für φ und ψ gezeigt. Sie gilt dann auch für $(\varphi \wedge \psi)$: $\beta((\varphi \wedge \psi)) = 1$ gdw. $\beta(\varphi) = 1 = \beta(\psi)$ gdw. φ und ψ beide in Σ_∞ sind gdw. $(\varphi \wedge \psi)$ in Σ_∞ ist. Letztere Äquivalenz ergibt sich leicht aus der endlichen Erfüllbarkeit von Σ_∞. Ähnlich argumentiert man für die übrigen zusammengesetzten Formeln. $\square$

Aus Satz 1.3 ergibt sich unmittelbar:

Korollar 1.4 *Sei $\Sigma \cup \{\varphi\}$ eine Menge von Formeln. Wenn $\Sigma \models \varphi$, dann gibt es ein endliches $\overline{\Sigma} \subset \Sigma$ mit $\overline{\Sigma} \models \varphi$.*

Diese Aussage ist sogar äquivalent zu 1.3.

Eine Formel ist in *konjunktiver Normalform*, falls sie die Gestalt

$$(\varphi_{0,0} \vee \cdots \vee \varphi_{0,i_0}) \wedge \cdots \wedge (\varphi_{k-1,0} \vee \cdots \vee \varphi_{k-1,i_{k-1}})$$

besitzt, wobei jedes $\varphi_{l,m}$ entweder Aussagenvariable oder Negation einer Aussagenvariablen ist. Analog ist eine Formel in *disjunktiver Normalform*, falls sie die Gestalt

$$(\varphi_{0,0} \wedge \cdots \wedge \varphi_{0,i_0}) \vee \cdots \vee (\varphi_{k-1,0} \wedge \cdots \wedge \varphi_{k-1,i_{k-1}})$$

besitzt, wobei jedes $\varphi_{l,m}$ entweder Aussagenvariable oder Negation einer Aussagenvariablen ist. Man zeigt leicht, dass es zu jeder aussagenlogischen Formel φ aussagenlogische Formeln ψ und ψ' gibt, so dass sowohl φ und ψ als auch φ und ψ' tautologisch äquivalent sind und ψ ist in konjunktiver Normalform und ψ' ist in disjunktiver Normalform.

Nicht alle logisch gültigen Schlüsse sind tautologische Implikationen. Allgemeinere Schlüsse werden in der *Logik 1. Stufe* untersucht.

Die Aufpump-Lemmata der vorherigen Kapitel können sehr leicht benutzt werden, um die folgenden Aussagen zu beweisen. Hierbei wollen wir uns vorstellen, dass die Aussagenvariable A_n mit einer Folge von $n+1$ Sternen $*$ identifiziert werde, um sinnvoll über Regularität und Kontextfreiheit von Mengen aussagenlogischer Formeln sprechen zu können.

Lemma 1.5 *Die Menge der aussagenlogischen Formeln ist nicht regulär, jedoch kontextfrei.*

Beweis: Wäre die Menge der aussagenlogischen Formeln regulär, dann wäre auch die Menge

$$\{w \in \{(,)\}^* \ : w \text{ hat ebensoviele linke wie rechte Klammern } \}$$

ebenfalls regulär, was aber dem Aufpump-Lemma für reguläre Sprachen widerspricht. Darüberhinaus wird die Menge der aussagenlogischen Formeln offensichtlich durch eine kontextfreie Grammatik generiert. □

Lemma 1.6 *Die Menge der aussagenlogischen Tautologien ist nicht kontextfrei.*

Beweis: Andernfalls betrachte man die Formel $(A_0 \vee \underbrace{\neg \ldots \neg}_{n \text{ viele}} A_0)$ für ein hinreichend großes n. □

Kapitel 4

Turing–Maschinen

Wahrheitstafelmethode können Sie entscheiden, ob eine vorgelegte Formel der Aussagenlogik eine Tautologie ist oder nicht. Eine solche Entscheidung könnte Ihnen ein Computer abnehmen.

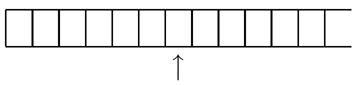
Eine (aussagenlogische) Formel φ heißt *erfüllbar* gdw. $\neg\varphi$ keine Tautologie ist, d.h. wenn es ein $\overline{\beta} : F_0 \to \{0,1\}$ gibt, das φ erfüllt. Wir definieren

$$SAT = \{\varphi : \varphi \text{ ist eine erfüllbare Formel}\}.$$

Wir wollen nun sehen, dass (in einem mathematisch präzisen Sinne) SAT (d.h. Elementschaft in SAT) entscheidbar ist. Später werden wir sehen, dass SAT "NP–vollständig" ist.

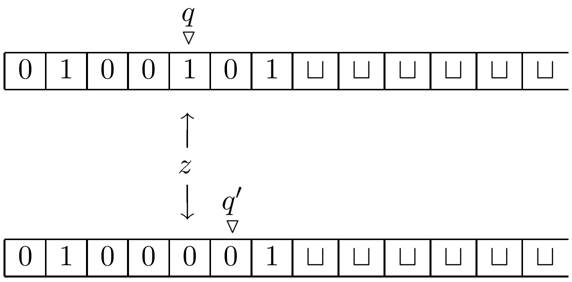
Im praktischen Leben verhalten sich Computer teilweise scheinbar chaotisch. In der Mathematik ist das anders: Computer folgen hier strengen Regeln. (Wir werden später auch "nicht–deterministische" Computer kennen lernen.) Dabei sind Computer bedeutend leistungsstärker als die bisher betrachteten Automaten.

Das mathematische Modell eines Computers ist die TURING–Maschine. Eine Turing–Maschine führt gemäßeines Programms Berechnungen auf einem Blatt Papier durch. Das "Blatt Papier" wird auch als *Schreibband* bezeichnet und sieht so aus:



n^{te} Zelle

Genau für jede natürliche Zahl n existiert eine n^{te} Zelle auf dem Schreibband; diese Zelle kann leer sein oder mit einem Symbol eines festen *Alphabets* Γ beschrieben sein. Die Turing–Maschine besitzt einen Kopf, der zu einem gegebenen Rechenzeitpunkt auf einer Zelle des Schreibbandes steht. Im nächsten Rechenschritt wird die Turing–Maschine abhängig von der Inschrift der Zelle z, auf der der Kopf steht, und abhängig vom gegenwärtigen *Zustand* der Maschine die Zelle z neu beschrieben, mit dem Kopf einen Schritt nach rechts oder links gehen und in einen neuen Zustand geraten:



Formal besteht für uns eine *Turing–Maschine* $\top$ aus den folgenden Objekten:

(1) Einer endlichen Menge Q von *Zuständen*, wobei es drei ausgezeichnete Zustände gibt: den *Anfangszustand* $q_0 \in Q$, den *positiven Endzustand* q_+ und den *negativen Endzustand* q_-.

(2) Ein endliches *Alphabet* Γ, d.h. eine Menge von Symbolen, die das Leerzeichen $\sqcup$ enthalten soll; eine nichtleere Teilmenge Σ von Γ, die nicht $\sqcup$ enthält, ist als *Eingabealphabet* ausgezeichnet.

(3) Eine *Übergangsfunktion*

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\},$$

die Paaren (q, a) mit $q \in Q$ und $a \in \Gamma$ jeweils ein Tripel (q', b, x) mit $q' \in Q, b \in \Gamma$ und $x \in \{L, R\}$ zuordnet.

Ein *Rechenvorgang* der Turing–Maschine $\top$ ist eine (endliche oder unendliche) Folge von *Konfigurationen* K_n. Die n^{te} Konfiguration ist gegeben durch:

(a) Einen Zustand $q \in Q$, in dem sich $\top$ nach n Rechenschritten befindet; dabei ist $q = q_0$ für $n = 0$, d.h. nach 0 Rechenschritten ist $\top$ im Anfangszustand.

(b) Eine Position des Kopfes, der auf einer Zelle des Rechenbandes steht; für $n = 0$ steht der Kopf am linken Ende des Bandes.

(c) Ein endliches *Wort*, das auf dem Rechenband niedergeschrieben ist; im Falle $n = 0$ ist dieses das *Eingabewort* (oder die *Eingabe*): für ein $l \in \mathbb{N}$ sind nach 0 Rechenschritten die ersten l Zellen mit jeweils einem Symbol aus Σ beschrieben.

Insbesondere liest der Kopf nach n Rechenschritten ein Symbol a (u.U. $\sqcup$), mit dem die Zelle, auf der er steht, beschrieben ist.

Der n^{te} Rechenschritt ist nun durch δ gegeben. Für $(q', b, x) = \delta(q, a)$ ist q' der Zustand, in dem $\top$ nach $n + 1$ Rechenschritten ist; das Wort, das nach $n + 1$ Rechenschritten auf dem Band steht, ergibt sich aus dem Wort, das nach n Rechenschritten auf dem Band steht, indem das Symbol a in der Zelle, auf der der Kopf nach n Schritten stand, durch das Symbol b ersetzt wird; der Kopf schließlich geht um einen Schritt nach links oder rechts, je nachdem, ob $x = L$ oder $x = R$ (falls der Kopf am linken Bandende steht, so bleibt er bei $x = L$ dort stehen).

Jeder Rechenvorgang von $\top$ ist offensichtlich durch die Eingabe determiniert. Ein Rechenvorgang bricht nach n Rechenschritten ab, falls $\top$ dann im Zustand q_+ oder im Zustand q_- ist. Im ersteren Fall sagen wir dann, dass $\top$ die Eingabe *akzeptiert*, im letzteren Fall sagen wir, dass $\top$ die Eingabe *verwirft*. (Es gibt dann keinen n^{ten} Rechenschritt.)

Falls $\top$ eine gegebene Eingabe weder verwirft, noch akzeptiert, dann ist der zugehörige Rechenvorgang ergebnislos und unendlich lange.

Sei w eine Eingabe. Wir schreiben

$$\top(w) \downarrow +, \text{ falls } \top w \text{ akzeptiert, und } \top(w) \downarrow -, \text{ falls } \top w \text{ verwirft.}$$

Falls der Rechenvorgang von $\top$ bei Eingabe von w nicht abbricht, so schreiben wir $\top(w) \uparrow$.

Wir wollen uns nun einige Beispiele ansehen. Es ist nicht auf Anhieb zu glauben, dass alles, was überhaupt "berechenbar" ist, mit Hilfe einer Turing–Maschine berechenbar ist. Dies wird aber die Aussage der These von Church sein.

Wir wollen zunächst eine Turing–Maschine $\top$ bauen, die entscheidet, ob die Eingabe eine gerade Anzahl von Nullen enthält.

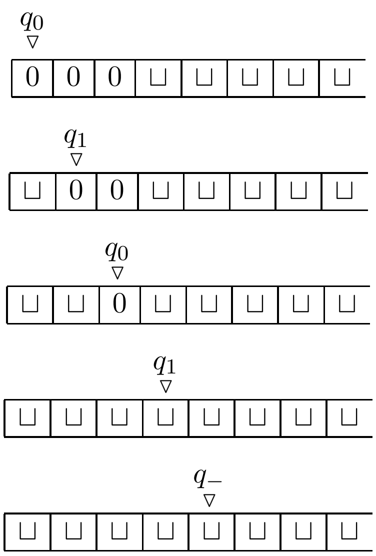
Sei $Q = \{q_0, q_1, q_+, q_-\}$. Sei etwa $\Sigma = \{0\}$ und $\Gamma = \{0, \sqcup\}$. Das Eingabealphabet hat also die Null als einziges Symbol. Die Übergangsfunktion δ

werde folgendermaßen beschrieben:

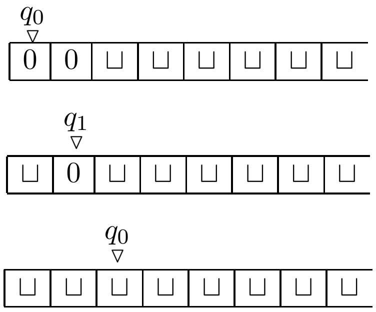
$$
\begin{array}{ll|lll}
q_0 & 0 & q_1 & \sqcup & R \\
q_1 & 0 & q_0 & \sqcup & R \\
q_0 & \sqcup & q_+ & \sqcup & R \\
q_1 & \sqcup & q_- & \sqcup & R
\end{array}
$$

$\top$ oszilliert zwischen den Zuständen q_0 und q_1 hin und her, wobei der Kopf in jedem Schritt nach rechts geht, solange, bis der Kopf das Eingabeende, d.h. $\sqcup$ erreicht. In diesem Fall wird akzeptiert/verworfen gdw. $\top$ eine gerade/ungerade Anzahl von Malen zwischen q_0 und q_1 oszilliert ist.

Ein typischer Rechenvorgang sieht wie folgt aus:



Die Eingabe 000 wird also verworfen. Die Eingabe 00 wird aber akzeptiert:



q_+
$\triangledown$

$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$

Es ist unschwer erkennbar, dass die Sprache, die aus Wörtern mit einer geraden Anzahl an Nullen besteht, auch von einem Automaten erkannt werden kann.

Etwas schwieriger ist es, eine Turing–Maschine zu konstruieren, die entscheidet, ob die Eingabe 2^n Nullen enthält, wobei $n \in \mathbb{N}$. (Das Aufpump–Lemma für kontextfreie Sprachen hatte gezeigt, dass $\{0^{2^n} : n \in \mathbb{N}\}$ nicht kontextfrei ist.) Die Idee ist die Folgende. Der Kopf liest wiederholt die Eingabe von links nach rechts, wobei jede zweite 0 durchgestrichen wird. Wenn dann eine ungerade Anzahl von Nullen auf dem Band bleibt, dann wird die Eingabe verworfen (es sei denn, es stand am Anfang genau eine 0 auf dem Band).

Es sei $Q = \{q_0, q_1, q_2, q_3, q_4, q_+, q_-\}, \Sigma = \{0\}$ und $\Gamma = \{0, x, \sqcup\}$. Die Übergangsfunktion δ werde wie folgt beschrieben.

q_0	$\sqcup$	q_-	$\sqcup$	R
q_0	0	q_1	$\sqcup$	R
q_1	$\sqcup$	q_+	$\sqcup$	R
q_1	x	q_1	x	R
q_1	0	q_2	x	R
q_2	x	q_2	x	R
q_2	$\sqcup$	q_4	$\sqcup$	L
q_4	0	q_4	0	L
q_4	x	q_4	x	L
q_4	$\sqcup$	q_1	$\sqcup$	R
q_2	0	q_3	0	R
q_3	x	q_3	x	R
q_3	0	q_2	x	R
q_3	$\sqcup$	q_-	$\sqcup$	R

Da die Maschine im Allgemeinen die Eingabe mehrfach lesen muss, ist es erforderlich, das linke Bandende beim Start zu *markieren*, so dass später beim Zurücksetzen des Kopfes erkannt werden kann, wann das linke Bandende erreicht ist. Dies geschieht, indem beim Start ein $\sqcup$ in die linkeste Zeile geschrieben wird.

Beim Nach–rechts–Gehen oszilliert die Maschine zwischen q_2 und q_3, wobei bereits durchgestrichene Nullen (d.h. x) ignoriert werden und jede zweite

Null neu durchgestrichen wird. Es dauert sicherlich einige Zeit, bis man sich in die Vorgehensweise dieser Turing–Maschine eingedacht hat!

Ein Wort im Alphabet Γ ist eine endliche Folge von Symbolen aus Γ. In jedem Rechenschritt ist (genau) ein Wort auf dem Schreibband niedergeschrieben. Das Wort endet mit dem ersten $\sqcup$, so dass nur noch weitere $\sqcup$ folgen.

Wir bezeichnen die Menge aller Worte im Alphabet Γ mit Γ^*. Eine Menge $L \subset \Gamma^*$ heißt auch *Sprache*. Die beiden grundlegenden Begriffe der Berechenbarkeitstheorie sind die folgenden:

Definition 4.1 *Sei $L \subset \Sigma^*$ eine Sprache, wobei Σ eine nichtleere endliche Menge von Symbolen ist. L heißt rekursiv aufzählbar (oder Turing–erkennbar) gdw. es eine Turing–Maschine $\top$ mit Eingabealphabet Σ gibt, so dass für alle $w \in \Sigma^*$:*

$$w \in L \Leftrightarrow \top(w) \downarrow +.$$

L heißt rekursiv (oder Turing–entscheidbar, oder einfach entscheidbar) gdw. es eine Turing–Maschine $\top$ mit Eingabealphabet Σ gibt, so dass für alle $w \in \Sigma^*$:

$$\begin{array}{l} w \in L \Leftrightarrow \top(w) \downarrow +, \textit{ und} \\ w \notin L \Leftrightarrow \top(w) \downarrow -. \end{array}$$

Offensichtlich ist jede entscheidbare Sprache rekursiv aufzählbar. Wir werden allerdings rekursiv aufzählbare Sprachen kennen lernen, die nicht entscheidbar sind (etwa das Halteproblem).

Der Beweis der folgenden Aussage ist nicht schwierig.

Lemma 4.2 *Eine Sprache $L \subset \Sigma^*$ ist Turing-entscheidbar gdw. sowohl L als auch $\Sigma^* \setminus L$ rekursiv aufzählbar ist.*

Beweisskizze: Sei sowohl L als auch $\Sigma^* \backslash L$ rekursiv aufzählbar. Sei $\top_0$ eine Turing–Maschine, die bezeugt, dass L rekursiv aufzählbar ist, und sei $\top_1$ eine Turing–Maschine, die bezeugt, dass $\Sigma^* \backslash L$ rekursiv aufzählbar ist. Wenn wir $\top_0$ und $\top_1$ gleichzeitig laufen lassen, dann erhalten wir bzgl. eines beliebigen $w \in \Sigma^*$ irgendwann eine Entscheidung, ob $w \in L$ oder $w \in \Sigma^* \backslash L$. Man kann nun recht einfach eine Turing–Maschine $\top$ bauen, die bei Eingabe von $w \in \Sigma^*$ in der n^{ten} Runde zunächst die ersten n Rechenschritte von $\top_0$ und sodann die ersten n Rechenschritte von $\top_1$ simuliert. □

Wir wollen nun zeigen, dass SAT entscheidbar ist. Offensichtlich können wir jede (aussagenlogische) Formel φ als Wort im Alphabet $\{(,), \neg, \wedge, \vee, \rightarrow, \leftrightarrow$

$,*\}$ auffassen; dabei identifizieren wir die Aussagenvariable A_n mit der Folge von $n+1$ vielen $*$. Aus $(A_3 \to (A_0 \vee A_1))$ wird also das Wort $(****\to (*\vee **))$. Teil des SAT–Problems ist es zu entscheiden, ob ein gegebener (aussagenlogischer) Ausdruck eine Formel ist. Hier ist eine Turing–Maschine $\top$ die entscheidet, ob im gegebenen Ausdruck eine Teilformel der Gestalt

$$(A_n \wedge A_m), \text{ d.h. } (*\ldots*\wedge*\ldots*)$$

vorkommt.

ein beliebiges q	(	q_1	(	R
q_1	$*$	q_2	$*$	R
q_2	$*$	q_2	$*$	R
q_2	$\wedge$	q_3	$\wedge$	R
q_3	$*$	q_4	$*$	R
q_4	$*$	q_4	$*$	R
q_4	)	q_+	)	R
ein beliebiges q	$\sqcup$	q_-	$\sqcup$	R
ein sonstiges Paar q	x	q_0	x	R

$\top$ hat also die 7 Zustände $q_0,\ldots,q_4,q_+,q_-$. Der Kopf läuft von links nach rechts, ohne die Zelleneintragungen zu ändern. Man erkennt unschwer, dass $\top$ die Eingabe akzeptiert (verwirft) gdw. eine Teilformel der Form $(*\ldots*\wedge*\ldots*)$ (nicht) vorkommt.

Wir können $\top$ leicht so variieren, dass $\top$ anstelle zu akzeptieren zunächst das gefundene Vorkommnis von $(*\ldots*\wedge*\ldots*)$ durch die "atomare Formel" $**\ldots***\ldots**$ ersetzt:

ein beliebiges q	(	q_1	(	R
q_1	$*$	q_2	$*$	R
q_2	$*$	q_2	$*$	R
q_2	$\wedge$	q_3	$\wedge$	R
q_3	$*$	q_4	$*$	R
q_4	$*$	q_4	$*$	R
q_4	)	q_5	$*$	L
q_5	$*$ oder $\wedge$	q_5	$*$	L
q_5	(	q_+	$*$	L
ein beliebiges q	$\sqcup$	q_-	$\sqcup$	R
ein sonstiges Paar q	x	q_0	x	R

Anstatt zu akzeptieren könnte diese neue Maschine auch mit dem Kopf zum linken Bandende zurücklaufen und von neuem starten und sehen, ob eine weitere Teilformel der Gestalt $(* \ldots * \wedge * \ldots *)$ im neu entstandenen Ausdruck auftritt und im positiven Falle diese Teilformel abermals durch $** \ldots *** \ldots **$ ersetzen.

Wir können schließlich sogar eine Turing–Maschine $\top_F$ bauen, die in jedem Lesedurchgang des Kopfes von links nach rechts nachsieht, ob eine Teilformel der Gestalt $\neg * \ldots *, (* \ldots * \wedge * \ldots *), (* \ldots * \vee * \ldots *), (* \ldots * \rightarrow * \ldots *)$ oder $(* \ldots * \leftrightarrow * \ldots *)$ im (jeweils neu entstandenen) Ausdruck auftritt und im positiven Falle diese Teilformel durch $** \ldots *$ bzw. $** \ldots *** \ldots **$ (d.h. durch eine "atomare Formel") ersetzt. Damit $\top_F$ keinen Unsinn produziert, sollte $\top_F$ allerdings in einem allerersten Lesedurchgang nachsehen, ob die Eingabe keine Teilfolgen der Form $*($ oder $)*$ enthält.[1] Falls nun dieser Prozess solange weiterläuft bis am Ende ein Ausdruck der Gestalt $* \ldots *$ (d.h. eine "atomare Formel") auf dem Band steht, dann akzeptiert $\top_F$ (und genau dann war die Eingabe eine (aussagenlogische) Formel); ansonsten verwirft $\top_F$ die Eingabe.

Ein kleines Problem hierbei ist wiederum: wie findet der Kopf von $\top_F$ jeweils das linke Bandende wieder? Eine einfache Lösung ist abermals, die linkeste Zelle sofort bei Rechenstart zu *markieren*, d.h. etwa

$$(,), \neg, \wedge, \vee, \rightarrow, \leftrightarrow, *$$

durch

$$(',)', \neg', \wedge', \vee', \rightarrow', \leftrightarrow', *'$$

zu ersetzen, wobei die Symbole $(', \ldots, *'$ nur für diesen Markierungszweck vorbehalten bleiben. Sobald der Kopf dann später ein Symbol der Form $(', \ldots, *'$ liest, erkennt $\top_F$, dass der Kopf am Bandanfang steht.

Wir haben also eine Turing–Maschine T_F mit folgender Eigenschaft gebaut: Für jeden (aussagenlogischen) Ausdruck w gilt $\top_F(w) \downarrow +$ gdw. w eine Formel ist, und $\top_F(w) \downarrow -$ gdw. w keine Formel ist. D.h. $\top_F$ bezeugt, dass die Menge aller Formeln entscheidbar ist.

Wenn wir nun entscheiden wollen, ob ein gegebenes φ eine erfüllbare Formel ist, so werden wir zunächst $\top_F$ entscheiden lassen, ob φ eine Formel ist; allerdings sollten wir in Wahrheit eine Variante von $\top_F$ rechnen lassen, die φ "intakt" lässt, d.h. nicht durch $* \ldots *$ ersetzt, da sich ansonsten φ nicht mehr rekonstruieren lässt. Es ist leicht, eine solche Variante von $\top_F$ zu bauen. Wenn dann der Formeltest positiv verlaufen ist, so wollen wir als nächstes entscheiden, ob φ erfüllbar ist. Dazu benutzen wir im Wesentlichen

[1] Andernfalls würde $\top_F$ z.B. den Ausdruck $*(** \wedge * **)$ akzeptieren.

die Methode der Wahrheitstafeln: für jedes $\overline{\beta} : F_0 \to \{0,1\}$ wollen wir ausrechnen, ob die zugehörige Belegung $\beta : F \to \{0,1\}$ das gegebene φ mit 0 (falsch) oder 1 (wahr) belegt. Sobald ein $\overline{\beta}$ mit $\beta(\varphi) = 1$ gefunden ist, akzeptieren wir φ. Falls niemals ein solches $\overline{\beta}$ gefunden wird, verwerfen wir φ. Da es bei $\overline{\beta}$ nur auf die Werte von Aussagenvariablen ankommt, die in φ tatsächlich vorkommen, kann tatsächlich nach endlich vielen Rechenschritten eine Entscheidung gefunden werden. Seien etwa $A_{i_0}, \ldots, A_{i_{n-1}}$ die in φ vorkommenden Aussagenvariablen. Es gibt dann 2^n Möglichkeiten, wie ein $\overline{\beta} : F_0 \to \{0,1\}$ diese Aussagenvariablen mit Wahrheitswerten belegt.

Die Lösung des SAT–Problems erfordert eine gute Buchhaltung. Wir bauen jetzt eine Turing–Maschine, die nach und nach die Zahlen von 0 bis 2^n in Dualdarstellung auf das Band schreibt und dann hält. Sei

$$\Sigma = \{\#, 0\}, \Gamma = \{\#, 0, 1, \sqcup\}.$$

dient zur Markierung des linken Bandendes. Wir starten mit der Eingabe $\#0 \ldots 0$, d.h. #, gefolgt von n Nullen. Die Maschine sei gegeben durch

q_0	$\#$	q_0	$\#$	R
q_0	0	q_0	0	R
q_0	1	q_0	1	R
q_0	$\sqcup$	q_1	$\sqcup$	L
q_1	0	q_2	1	L
q_1	1	q_3	0	L
q_2	0	q_2	0	L
q_2	1	q_2	1	L
q_3	0	q_2	1	L
q_3	1	q_3	0	L
q_1 oder q_2	$\#$	q_0	$\#$	R
q_3	$\#$	q_+	$\#$	L

Der Kopf geht zunächst nach rechts, wonach die Maschine schriftlich 1 addiert. Dabei entspricht der Zustand q_2 "Null gemerkt" und der Zustand q_3 "Eins gemerkt". Die Maschine hält, sobald der Kopf am linken Bandende bei "Eins gemerkt" steht. Die Bandinschrift lautet dann $\#1 \ldots 1$, d.h. #, gefolgt von n Einsen.

Wir beschreiben schließlich eine Turing–Maschine $\top_{SAT}$, die das SAT–Problem entscheidet.

Es werde φ eingegeben. Zunächst lässt $\top_{SAT}$ die Maschine $\top_F$ laufen und entscheidet, ob φ eine Formel ist. Im negativen Falle verwirft $\top_{SAT}$ die Eingabe. Sei nun diese erste Entscheidung positiv.

Zunächst markiert jetzt $\top_{SAT}$ das linke Bandende (dies ist wohl ohnehin durch $\top_F$ bereits geschehen). Sodann kopiert $\top_F$ die Formel φ nach rechts, vom Urbild etwa durch eine weitere Markierung # getrennt. Dann schreibt $\top_{SAT}$ weiter rechts davon, etwa nochmals durch # (oder besser ##) getrennt, eine Folge $0 \ldots 0$ von n Nullen, wobei etwa einfach n die Länge von φ sei. (Wenn die Aussagenvariablen A_i, d.h. $i{+}1$ viele $*$, in φ vorkommen, dann ist $i < n$.) Wir benutzen diesen hintersten Teil zur Buchführung bzgl. der abgehandelten Belegungen der Aussagenvariablen $A_0, \ldots, A_{n-1}$ mit Wahrheitswerten 0, bzw. 1.

Auf dem Band sieht es nun so aus:

$$\varphi \# \varphi \# \# 0 \ldots 0.$$

Zu einem späteren analogen Zeitpunkt der Berechnung sieht es auf dem Band so aus:

$$\varphi \# \varphi \# \# d,$$

wobei d die Dualdarstellung einer Zahl zwischen 0 und $2^n - 1$ ist.

Wir ersetzen nun gemäßd jedes Vorkommnis von $A_i, i < n$, (d.h. jedes weder nach links, noch nach rechts verlängerbare Vorkommnis einer $*$–Folge der Länge $i + 1$) in der Kopie von φ zwischen # und ## durch 0, bzw. 1. Sodann berechnen wir den sich ergebenden Wahrheitswert. Nach und nach ersetzen wir

$$\begin{array}{rcc} \neg 0 & \text{durch} & 1 \\ \neg 1 & & 0 \\ (0 \wedge 0) & & 0 \\ (0 \wedge 1) & & 0 \\ (1 \wedge 0) & & 0 \\ (1 \wedge 1) & & 1 \\ & \text{usw.} & \end{array}$$

Falls sich am Ende der Wert 1 ergibt, so akzeptieren wir die ursprüngliche Eingabe φ; wir haben dann eine Belegung gefunden, die φ erfüllt.

Falls sich 0 ergibt, so ersetzen wir diese 0 wieder durch φ und erhöhen den "Zähler" d um 1. Falls der Zählerstand gleich 2^n ist und sich auch in diesem letzten Falle als Wahrheitswert von φ die 0 ergibt, so verwerfen wird die ursprüngliche Eingabe; es gibt dann keine Belegung, die φ erfüllt.

Wir haben also gesehen, wie sich eine Turing–Maschine $\top_{SAT}$ bauen lässt, so dass $\top_{SAT}$ bezeugt, dass SAT entscheidbar ist.

Das Problem SAT lässt sich also mit relativ primitiven Mitteln lösen. Die *These von Church* sagt: wenn sich ein Problem im intuitiven Sinne durch

Rechnen (d.h. durch einen Algorithmus irgendwelcher Art) lösen lässt, dann lässt es sich mit Hilfe einer Turing–Maschine lösen.

Kapitel 5

Das Halteproblem

Das *Halteproblem* lautet: lässt sich entscheiden, ob eine gegebene Turing–Maschine $\top$ bei einer gegebenen Eingabe w hält oder nicht, d.h. ob $\top(w)\downarrow$ oder $\top(w)\uparrow$?[1]

Wir wollen zunächst eine *universelle Turing–Maschine* bauen. Eine solche ist eine Turing–Maschine, die so programmiert ist, dass sie alle Berechnungen beliebiger Turing–Maschinen simulieren kann. Wir erinnern uns daran, dass eine Turing–Maschine $\top$ durch endlich viele Zustände Q, ein endliches Alphabet Γ (wovon ein nichtleeres $\Sigma \subsetneqq \Gamma$ das Eingabealphabet ist) und eine (dann ebenfalls endliche) Übergangsfunktion δ gegeben ist. Es sei etwa

$$Q = \{q_0, q_+, q_-, q_1, q_2, \dots, q_{n-1}\}$$

und

$$\Gamma = \{\sqcup, x_0, x_1, \dots, x_{m-1}\}.$$

Wir wollen $\top$ einen *Namen* $\#\top$ zuordnen. Die universelle Turing–Maschine U wird dann bei Eingabe von $\#\top, w$ den Rechenvorgang von $\top$ bei Eingabe von w simulieren. Dies wird zeigen, dass $\{w : \top(w)\downarrow\}$ zumindest rekursiv aufzählbar ist.

Wir wollen den Zustand q_+ von $\top$ mit $*, q_-$ mit $**$ und q_i, für $i < n$, mit der Folge $*\dots*$ von $i+3$ Sternen identifizieren. Sodann wollen wir das Leerzeichen $\sqcup$ von $\top$ mit $|$ und das Symbol x_i, für $i < m$, mit der Folge $|\dots|$ von $i+2$ Strichen identifizieren. Für "links" und "rechts" reservieren wir uns die Symbole L und R. Die Übergangsfunktion δ von $\top$ können wir nun durch Auflistung ihres Graphen, d.h. aller 5–Tupel $(q, \gamma, q', \gamma', x)$ mit

[1] Hierbei schreiben wir $\top(w)\downarrow$ für $(\top(w)\downarrow +$ oder $(\top(w)\downarrow -)$.

$\delta(q,\gamma) = (q', \gamma', x)$, mitteilen. Es gibt $l = (n+2)\cdot(m+1)$ derartige Tupel. Diese Auflistung können wir einfach in der Form

$$(*, |, q', \gamma', x)(**, |, q'', \gamma'', x') \dots (* \dots *, | \dots |, q^{(l)}, \gamma^{(l)}, x^{(l-1)})$$

mitteilen. Diese Auflistung verwendet die 7 Symbole (,), das Komma, $*, |, L$ und R; wir können sie einer Turing–Maschine eingeben. Wir wollen sie den *Namen von* $\top$, in Zeichen $\#\top$, nennen.

Unsere universelle Turing–Maschine U besitzt das Eingabealphabet $\Sigma_U = \{(,),$ das Komma, $*, |, L, R, \sqcup', ; \}$. U arbeitet sinnvoll bei einer Eingabe der Gestalt

$$\#\top(* * * \sqcup' \dots \sqcup')w.$$

Hierbei ist $\#\top$ der Name einer Turing–Maschine $\top$, $\sqcup' \dots \sqcup'$ ist eine Folge von $n-1$ "Leerzeichen" $\sqcup'$, und w ist (ein Code für) eine Eingabe, so dass U bei obiger Eingabe die Berechnung von $\top$ bei Eingabe von w simulieren wird. Hierbei können wir eine solche Eingabe w etwa in folgender Art und Weise kodieren. Das Eingabealphabet von $\top$ ist eine Teilmenge von $\{x_0, \dots, x_{m-1}\}$, so dass w eine Folge von $x_i, i < m$, ist. Da wir $x_i, i < m$, mit einer Folge $| \dots |$ von $i+2$ Strichen identifizieren, können wir w als Folge derartiger Strichfolgen, voneinander etwa durch das Komma, abgetrennt, ansehen. w sei also von der Gestalt[2]

$$; | \dots |, | \dots |, \dots, | \dots |.$$

Der Teil $(***\sqcup' \dots \sqcup')$ der Eingabe dient als Platzhalter, so dass U festhalten kann, in welchem Zustand die simulierte Turing–Maschine $\top$ sich jeweils befindet. Beim Start von U befindet sich $\top$ im Anfangszustand q_0 von $\top$, d.h. $* * *$.

U kann unschwer so gestaltet werden, dass die drei Teile der Eingabe, $\#\top$ (zur Mitteilung des "Programms" von $\top$), $(***\sqcup' \dots \sqcup')$ (als Platzhalter für den Zustand von $\top$) und w (die Eingabe für $\top$) voneinander getrennt werden können.[3] Das Alphabet Γ_U von U sei Σ_U, zusammen mit dem Zeichen $\sqcup$.

Werde nun U mit der Eingabe

$$\#\top(* * * \sqcup' \dots \sqcup')w$$

[2] Die Rolle des ; am Anfang wird später erläutert.

[3] Eine Markierung des Bandanfangs ist übrigens nicht nötig. Wenn U beim Nach–Links–Gehen zweimal hintereinander (liest, dann ist das linke Bandende erreicht.

gefüttert, die so aussieht, wie soeben beschrieben. Zu einem späteren Zeitpunkt der Berechnung von U wird auf dem Band Folgendes niedergeschrieben sein:

$$\#\top(* \ldots * \sqcup' \ldots \sqcup')w'.$$

Hierbei ist $* \ldots *$ eine Folge von i Sternen, $1 \leq i < n+3$, und $\sqcup' \ldots \sqcup'$ ist eine Folge von $n+2-i$ "Leerzeichen"; dies teilt mit, dass die simulierte Turing–Maschine $\top$ gerade im Zustand q_+ (für $i=1$), q_- (für $i=z$), bzw. q_{i-3} (für $2 < i < n+3$) ist. w' ist von der Gestalt

$$, | \ldots |, | \ldots |, \ldots ; | \ldots |, \ldots , | \ldots |.$$

w' steht für einen Bandinhalt, den die simulierte Turing–Maschine $\top$ gerade auf ihrem Band vorfindet. Ähnlich wie w oben stellt w' eine Folge von Symbolen aus $\{\sqcup, x_0, \ldots, x_{m-1}\}$ dar, die voneinander durch das Komma und ; abgetrennt sind. Wie in w so soll auch in w' das ; genau an einer Stelle vorkommen; dadurch wird markiert, dass der Kopf der simulierten Maschine $\top$ gerade auf dem Zeichen steht, das durch die dahinter stehende Strichfolge $| \ldots |$ kodiert wird.

Die Turing–Maschine U wird nun das durch ; markierte Zeichen, sowie den aktuellen Zustand von $\top$ (an Hand von $(* \ldots * \sqcup' \ldots \sqcup')$) einlesen, und sodann mit Hilfe von $\#\top$ (d.h. mit Hilfe der eingelesenen Übergangsfunktion δ von $\top$) das markierte Zeichen durch ein neues Zeichen ersetzen, das Markierungszeichen ; nach links oder rechts versetzen (bzw. an derselben Stelle belassen, falls der Kopf von $\top$ als am linken Bandende stehend erkannt wird) und den neuen Zustand notieren. Dieser Vorgang, der einem Rechenschritt von $\top$ entspricht, wird einige Rechenschritte von U erfordern.

Schließlich soll U die ursprüngliche Eingabe $\#\top(* * * \sqcup' \ldots \sqcup')w$ akzeptieren/verwerfen, gdw. U als Zustand der simulierten Maschine $\top$ jemals $*$ (d.h. q_+ von $\top$)/$**$ (d.h. q_- von $\top$) vorfindet. Falls dies niemals der Fall ist, so wird U unendlich lange weiterrechnen.

Wir können schließlich U auch so bauen, dass diese Maschine, falls sie nicht mit einer Eingabe der Form

$$\#\top(* * * \sqcup' \ldots \sqcup')w$$

wie beschrieben gefüttert wird, ebenfalls unendlich lange rechnet (U kann entscheiden, ob die Eingabe "vernünftig" ist und im gegenseitigen Fall von da ab einfach mit dem Kopf immer einen Schritt nach rechts gehen).

Unsere Turing–Maschine U hat also die folgende Eigenschaft: U akzeptiert/verwirft eine Eingabe gdw. diese von der Gestalt $\#\top(* * * \sqcup' \ldots \sqcup')w$ ist, wobei $\top(w) \downarrow +$ bzw. $\top(w) \downarrow -$. Wir haben damit folgenden Satz bewiesen.

Satz 5.1 *Die Mengen* $\{(\#\top, w) : \top(w) \downarrow +\}$, $\{(\#\top, w) : \top(w) \downarrow -\}$ *und* $\{(\#\top, w) : \top(w) \downarrow\}$ *sind rekursiv aufzählbar.*

Wir hatten $\{(\#\top, w) : \top(w) \downarrow\}$ das Halteproblem genannt. Wir zeigen nun mit Hilfe eines "Diagonalargumentes":

Satz 5.2 *Das Halteproblem* $\{(\#\top, w) : \top(w) \downarrow\}$ *ist nicht entscheidbar.*

Beweis: Sei H eine Turing–Maschine, für die Folgendes gilt: $H(\overline{w}) \downarrow +$ gdw. $\overline{w}$ von der Gestalt $(\#\top, w)$ ist, wobei $\top$ eine Turing–Maschine ist mit $\top(w) \downarrow$; $H(\overline{(}w) \downarrow -$ gdw. $\overline{w}$ von der Gestalt $(\#\top, w)$ ist, wobei $\top$ eine Turing–Maschine ist mit $\top(w) \uparrow$, oder $\overline{w}$ ist nicht von der Gestalt $(\#\top, w)$. Wir können dann sehr einfach eine Turing–Maschine D bauen, für die gilt:

$$D(\#\top) \downarrow + \text{ gdw. } H((\#\top, \#\top)) \downarrow -$$

und[4]

$$D(\#\top) \downarrow - \text{ gdw. } H((\#\top, \#\top)) \downarrow +.$$

Dann gilt aber

$$D(\#D) \downarrow + \text{ gdw. } H((\#D, \#D)) \downarrow - \text{ gdw. } D(\#D) \uparrow .$$

Dies ist ein Widerspruch! □

[4]Wir benötigen dies nur für Maschinen $\top$ in die $\#\top$ eingegeben werden kann.

Kapitel 6

Reduktionen und das Postsche Korrespondenzproblem

Eine Reduktion ist eine effiziente Art und Weise, ein gegebenes Problem auf ein anderes zurückzuführen, so dass eine Lösung des zweiten Problems eine Lösung des ursprünglichen Problems liefert (falls das zweite Problem lösbar ist).

Wir wollen nun diese Methode an einem Beispiel betrachten, dem POSTschen Korrespondenzproblem.

Wir beginnen mit einer Menge von Dominosteinen. Ein typischer Dominostein hat die Gestalt

$$\boxed{\begin{array}{c} a \\ \hline b\,a \end{array}}$$

und eine typische Menge von Dominosteinen ist etwa

$$\left\{ \boxed{\begin{array}{c} b \\ \hline c\,a \end{array}},\ \boxed{\begin{array}{c} a \\ \hline a\,b \end{array}},\ \boxed{\begin{array}{c} c\,a \\ \hline a \end{array}},\ \boxed{\begin{array}{c} a\,b\,c \\ \hline c \end{array}} \right\}$$

Die Aufgabe ist es, aus einer solchen Menge von Dominosteinen ein *Match* zu gestalten. Dabei wird jeder der Menge entnommene Stein sofort durch einen gleichen Stein ersetzt, so dass also im Match Wiederholungen erlaubt sind. Ein Match ist eine Folge von Dominosteinen, bei der das in der oberen Zeile entstehende Wort mit dem in der unteren Zeile entstehenden Wort übereinstimmt.

Beispielsweise lässt sich aus der obigen Menge wie folgt ein Match bilden:

$$\begin{array}{|c|}\hline a\\\hline a\,b\\\hline\end{array},\ \begin{array}{|c|}\hline b\\\hline c\,a\\\hline\end{array},\ \begin{array}{|c|}\hline c\,a\\\hline a\\\hline\end{array},\ \begin{array}{|c|}\hline a\\\hline a\,b\\\hline\end{array},\ \begin{array}{|c|}\hline a\,b\,c\\\hline c\\\hline\end{array}$$

Aus anderen Mengen lässt sich kein Match bilden, z.B. nicht aus

$$\left\{\begin{array}{|c|}\hline a\\\hline b\,c\\\hline\end{array},\ \begin{array}{|c|}\hline a\,b\\\hline c\,b\\\hline\end{array},\ \begin{array}{|c|}\hline c\,a\\\hline b\\\hline\end{array}\right\}$$

da jeder Dominostein aus dieser Menge oben ein a und unten kein a enthält.

Das *Postsche Korrespondenzproblem* lautet:

Gegeben sei eine (endliche!) Menge M von Dominosteinen. Lässt sich aus M ein Match gestalten?

Satz 6.1 *Das Postsche Korrespondenzproblem ist nicht entscheidbar.*

Beweis: Wir liefern eine Reduktion des Halteproblems auf das Korrespondenzproblem. Wäre das Korrespondenzproblem entscheidbar, dann wäre auch das Halteproblem entscheidbar.

Unsere Aufgabe ist es also, eine Anfrage an das Halteproblem mit Hilfe von Dominosteinen umzuformulieren.

Das *modifizierte Korrespondenzproblem* lautet:

Gegeben sei eine (endliche!) Menge M von Dominosteinen, wobei ein Stein aus M als "erster" Stein ausgezeichnet sei. Lässt sich aus M ein Match gestalten, welches mit dem ersten Stein von M beginnt?

Wir zeigen zunächst, dass das Halteproblem entscheidbar wäre, wenn das modifizierte Korrespondenzproblem entscheidbar wäre.

Sei $\top$ eine Turingmaschine, die durch Q, Γ (und Σ), sowie δ gegeben ist. Wir wollen eine Menge $M = M_\top^W$ von Dominosteinen konstruieren, so dass einem Match ein Rechenvorgang von $\top$ bei Eingabe von $w \in \Sigma^*$ einspricht.

Sei $w = \gamma_0 \dots \gamma_{n-1}$, wobei $\gamma_0, \dots, \gamma_{n-1} \in \Sigma$. Dann sei

$$\begin{array}{|l|}\hline \#\\\hline \# q_0 \gamma_0 \cdots \gamma_{n-1} \#\\\hline\end{array}$$

ein Dominostein von M, der als "erster" Stein ausgezeichnet werden soll.[1] Diesem Stein entspricht in offensichtlicher Weise die nullte Konfiguration des Rechenvorgangs von $\top$ bei Eingabe von w. Die nächsten Dominosteine stehen für Anwendungen der Übergangsfunktion δ.

[1]Hierbei sei # ein Symbol, das weder in Q noch in Γ enthalten ist.

Seien $q, q' \in Q$ und $\gamma_0, \gamma_1 \in \Gamma$. Wenn $\delta(q, \gamma_0) = (q', \gamma_1, R)$, dann sei

$$\begin{array}{|c|}\hline q\ \gamma_0 \\ \hline \gamma_1\ q' \\ \hline\end{array}$$

ein Dominostein aus M. Wenn $\delta(q, \gamma_0) = (q', \gamma_1, L)$, dann sei für jedes $\gamma_2 \in \Gamma$

$$\begin{array}{|c|}\hline \gamma_2\ q\ \gamma_0 \\ \hline q'\ \gamma_2\ \gamma_1 \\ \hline\end{array}$$

ein Dominostein aus M.

Weiterhin sei für jedes $\gamma \in \Gamma \cup \{\#\}$

$$\begin{array}{|c|}\hline \gamma \\ \hline \gamma \\ \hline\end{array}$$

ein Dominostein von M, und es sei

$$\begin{array}{|r|}\hline \# \\ \hline \sqcup\ \# \\ \hline\end{array}$$

ein Dominostein von M.

Betrachten wir, bevor wir weitergehen, ein Beispiel. Sei etwa $\top$ die Maschine aus Kapitel 2, die entscheidet, ob die Eingabe eine gerade Anzahl von Nullen enthält. Den Rechenvorgang von $\top$ bei Eingabe von 000 können wir dann wie folgt durch eine Folge von Dominosteinen aus M wiedergeben (vgl. Seite 12):

$$\begin{array}{|l|}\hline \# \\ \hline \#q_0000\# \\ \hline\end{array}\ \begin{array}{|c|}\hline q_0\ 0 \\ \hline \sqcup\ q_1 \\ \hline\end{array}\ \begin{array}{|c|}\hline 0 \\ \hline 0 \\ \hline\end{array}\ \begin{array}{|c|}\hline 0 \\ \hline 0 \\ \hline\end{array}$$

$$\begin{array}{|c|}\hline \# \\ \hline \# \\ \hline\end{array}\ \begin{array}{|c|}\hline \sqcup \\ \hline \sqcup \\ \hline\end{array}\ \begin{array}{|c|}\hline q_1\ 0 \\ \hline \sqcup\ q_0 \\ \hline\end{array}\ \begin{array}{|c|}\hline 0 \\ \hline 0 \\ \hline\end{array}$$

$$\begin{array}{|c|}\hline \# \\ \hline \# \\ \hline\end{array}\ \begin{array}{|c|}\hline \sqcup \\ \hline \sqcup \\ \hline\end{array}\ \begin{array}{|c|}\hline \sqcup \\ \hline \sqcup \\ \hline\end{array}\ \begin{array}{|c|}\hline q_0\ 0 \\ \hline \sqcup\ q_1 \\ \hline\end{array}$$

$$\begin{array}{|r|}\hline \# \\ \hline \sqcup\ \# \\ \hline\end{array}\ \begin{array}{|c|}\hline \sqcup \\ \hline \sqcup \\ \hline\end{array}\ \begin{array}{|c|}\hline \sqcup \\ \hline \sqcup \\ \hline\end{array}\ \begin{array}{|c|}\hline \sqcup \\ \hline \sqcup \\ \hline\end{array}\ \begin{array}{|c|}\hline q_1\ \sqcup \\ \hline \sqcup\ q_- \\ \hline\end{array}$$

Dies ist natürlich kein Match. Die obere, bzw. untere Zeile lautet

$$\begin{array}{l} \#q_0000\# \sqcup q_100\# \sqcup \sqcup q_00\# \sqcup \sqcup \sqcup q_1\sqcup \\ \#q_0000\# \sqcup q_100\# \sqcup \sqcup q_00\# \sqcup \sqcup \sqcup q_1 \sqcup \# \sqcup \sqcup \sqcup \sqcup q_- \end{array}$$

Mit Hilfe von weiteren Dominosteinen, die das Halten von $\top$ wiederspiegeln, können wir aber die obige Folge zu einem Match fortsetzen.

Für jedes $\gamma \in \Gamma$ seien

$$\left[\frac{\gamma\, q_+}{q_+}\right], \quad \left[\frac{q_+\, \gamma}{q_+}\right], \quad \left[\frac{\gamma\, q_-}{q_-}\right] \quad \text{und} \quad \left[\frac{q_-\, \gamma}{q_-}\right]$$

Dominosteine von M. Schließlich seien auch

$$\left[\frac{q_+\, \#\, \#}{\#}\right] \quad \text{und} \quad \left[\frac{q_-\, \#\, \#}{\#}\right]$$

Dominosteine von M.

In unserem obigen Beispiel entsteht dann ein Match, wenn wir die Folge so fortsetzen:

$$\left[\frac{\#}{\#}\right]\left[\frac{\sqcup}{\sqcup}\right]\left[\frac{\sqcup}{\sqcup}\right]\left[\frac{\sqcup}{\sqcup}\right]\left[\frac{\sqcup\, q_-}{q_-}\right]$$
$$\left[\frac{\#}{\#}\right]\left[\frac{\sqcup}{\sqcup}\right]\left[\frac{\sqcup}{\sqcup}\right]\left[\frac{\sqcup\, q_-}{q_-}\right]$$
$$\left[\frac{\#}{\#}\right]\left[\frac{\sqcup}{\sqcup}\right]\left[\frac{\sqcup\, q_-}{q_-}\right]\left[\frac{\#}{\#}\right]\left[\frac{\sqcup\, q_-}{q_-}\right]$$
$$\left[\frac{\#}{\#}\right]\left[\frac{q_-\, \#\, \#}{\#}\right]$$

Die obere, bzw. untere Zeile dieser Fortsetzung lautet:

$$\begin{array}{r} \# \sqcup \sqcup \sqcup \sqcup q_- \# \sqcup \sqcup \sqcup\, q_- \# \sqcup \sqcup q_- \# \sqcup\, q_- \# q_- \#\# \\ \# \sqcup \sqcup \sqcup\, q_- \# \sqcup \sqcup q_- \# \sqcup\, q_- \# q_- \#\# \end{array}$$

Ein Dominostein möge nun zur Menge $M = M_\top^w$ gehören, gdw. er dies gemäßeiner der obigen Bestimmungen sein muss.

Nennen wir ad hoc eine Turingmaschine $\top$ *vernünftig*, falls bei keiner Eingabe von w Folgendes passiert: während der Berechnung steht der Kopf irgendwann am linken Ende des Bandes und δ sagt, dass der Kopf im nächsten Rechenschritt nach links gehen soll (so dass er also am linken Bandende stehen bleibt).

Falls $\top$ nicht vernünftig ist, dann haben wir Schwierigkeiten, mit Hilfe der obigen Dominosteine Berechnungen von $\top$ zu simulieren. Folgendes ist aber richtig: wenn $\top$ vernünftig ist, dann lässt sich aus dem oben definierten

$M^w_\top$ ein Match gestalten, welches mit dem "ersten" Stein von $M^w_\top$ beginnt, genau dann, wenn $\top(w)\downarrow$.

Mit dem Problem der Existenz nicht–vernünftiger Turingmaschinen gehen wir wie folgt um.

Zu jeder Turingmaschine $\top$ gibt es eine vernünftige Turingmaschine $\top^v$, so dass $\top(w)\downarrow$ gdw. $\top^v(w)\downarrow$ für alle Eingaben w gilt. Damit haben sir das Halteproblem auf das modifizierte Korrespondenzproblem reduziert: es gilt $\top(w)\downarrow$ gdw. sich aus $M^w_{\top^v}$ in Match gestalten lässt, welches mit dem "ersten" Stein beginnt.

Wir wollen nun $M = M^w_\top$ so zu einer Menge $\widetilde{M} = \widetilde{M}^w_\top$ umgestalten, dass jedes Match aus $\widetilde{M}$ einem Match aus M entspricht, welches mit dem ersten Stein beginnt.

Für eine Folge $w = \gamma_0 \dots \gamma_{k-1}$ schreiben wir

$*w$ für $*\gamma_0 \dots * \gamma_{k-1}$,

w für $\gamma_0 * \dots \gamma_{k-1}*$ und

$*w*$ für $*\gamma_0 * \dots * \gamma_{k-1}*$.

Bestehe nun $M = M^w_\top$ aus den Dominosteinen

$$\begin{array}{|c|}\hline w_i \\ \hline w'_i \\ \hline\end{array}$$

für $i < N$, wobei

$$\begin{array}{|c|}\hline w_0 \\ \hline w'_0 \\ \hline\end{array}$$

der erste Dominostein von M sei. Dann enthalte $\widetilde{M} = \widetilde{M}^w_\top$ die folgenden $N+2$ Dominosteine:

$$\begin{array}{|l|}\hline * \, w_0 \\ \hline * \, w'_0 \, * \\ \hline\end{array}\;, \quad \begin{array}{|r|}\hline * \, \diamond \\ \hline \diamond \\ \hline\end{array}\;,$$

sowie

$$\begin{array}{|l|}\hline * \, w_i \\ \hline w'_i \, * \\ \hline\end{array}$$

für alle $i < N$.[2] Es ist unschwer zu erkennen, dass jedes Match von Steinen aus $\widetilde{M}$ mit

$$\begin{array}{|l|}\hline * \, w_0 \\ \hline * \, w'_0 \, * \\ \hline\end{array}$$

[2]Hierbei seien $*$ und $\diamond$ neue Symbole.

beginnen und mit

$$\begin{array}{|r|}\hline * \diamond \\ \hline \diamond \\ \hline \end{array}$$

enden muss. Darüber hinaus entspricht jedes Match aus $\widetilde{M}$ genau einem Match aus M, das mit dem ersten Stein beginnt, und umgekehrt. Es gilt also: $\top(w)\downarrow$ gdw. sich aus $\widetilde{M}^{w}_{\top v}$ ein Match gestalten lässt. $\square$

Wir wollen schließlich den Begriff der Reduktion formal definieren.

Definition 6.2 *Eine Funktion $f : \Sigma^* \to \Sigma^*$ heißt* (Turing–) berechenbar *gdw. es eine Turing–Maschine $\top$ gibt, so dass für alle $w \in \Sigma^*$ $\top$ bei Eingabe von w hält, wobei am Ende $f(w)$ auf dem Rechenband geschrieben ist.*

Definition 6.3 *Seien $L_0, L_1 \subset \Sigma^*$ Sprachen. L_0 heißt* (Turing–) reduzierbar auf L_1, *kurz $L_0 \leq_\top L_1$, gdw. es eine berechenbare Funktion $f : \Sigma^* \to \Sigma^*$ gibt, so dass für alle $w \in \Sigma^*$ gilt:*

$$w \in L_0 \text{ gdw. } f(w) \in L_1.$$

Der obige Beweis zeigt, dass das Halteproblem auf das Postsche Korrespondenzproblem reduzierbar ist, im eben definierten Sinne. Allgemein gilt:

Lemma 6.4 *Seien $L_0, L_1 \subset \Sigma^*$ Sprachen mit $L_0, \leq_\top L_1$. Wenn L_1 rekursiv aufzählbar ist, dann ist auch L_0 rekursiv aufzählbar. Wenn L_1 entscheidbar ist, dann ist auch L_0 entscheidbar.*

Beweis: Sei $w \in L_0$ gdw. $f(w) \in L_1$ für alle $w \in \Sigma^*$, wobei f berechenbar ist. Wenn L_1 rekursiv aufzählbar ist, dann existiert eine Turing–Maschine, die bei eingegebenem $w \in \Sigma^*$ zunächst $f(w)$ berechnet und sodann $f(w)$ akzeptiert, gdw. $f(w) \in L_1$; diese Turing–Maschine bezeugt dann, dass L_0 rekursiv aufzählbar ist. Der zweite Teil ergibt sich dann mit Lemma 2.1. $\square$

Kapitel 7

Die Komplexitätsklassen P und NP

Entscheidbare Probleme müssen nicht "einfach" sein. Wir werden im nächsten Kapitel sehen, in welchem Sinne das in Kapitel 3 betrachtete SAT–Probleme sehr kompliziert ist. In diesem Kapitel wollen wir *einfache* Probleme studieren.

Probleme können schwierig sein, da ihre Lösung viel Zeit oder "Papier" (Speicherplatz, bzw. Platz auf dem Rechenband) in Anspruch nimmt. Wir wollen uns mit Zeitkomplexität beschäftigen.

Betrachten wir die drei Turing–Maschinen, die wir in Kapitel 2 gebaut haben. Die erste Maschine entscheidet die Menge aller Folgen der Form

$$\underbrace{00\ldots0}_{n \text{ viele}},$$

wobei n gerade ist. Bei einer Eingabe der Länge n benötigt die Maschine $n+1$ Rechenschritte, um eine Antwort zu finden.

Die zweite Maschine entscheidet

$$\underbrace{00\ldots0}_{2^n \text{ viele}},$$

wobei $n \in \mathbb{N}$. Bei einer Eingabe der Länge 2^n benötigt die Maschine

$$(2n+1)\cdot 2^n + 1$$

Rechenschritte; entsprechend benötigt sie bei einer Eingabe kürzerer Länge höchstens $(2n+1)2^n+1$ Rechenschritte. Da wir nicht allzu genau sein müssen, können wir also sagen, dass diese Maschine bei einer Eingabe der Länge n zirka

$$2 \cdot \log_2 n \cdot n$$

(also im Allgemeinen weniger als $2 \cdot n^2$) Rechenschritte benötigt.

Betrachten wir schließlich die Turing–Maschine, die SAT entscheidet. Hier dauert die Entscheidung wesentlich länger. Wenn die Eingabe die Länge n hat, dann kann die Eingabe "größenordnungsmäßig" n verschiedene Aussagenvariablen enthalten, so dass 2^n Belegungen dieser Aussagenvariablen betrachtet werden müssen.

Definition 7.1 *Seien $f : \mathbb{N} \to \mathbb{R}^+$, $g : \mathbb{N} \to \mathbb{R}^+$ Funktionen.*[1] *Wir schreiben $f \leq O(g)$ gdw. ein $c \in \mathbb{R}^+$ existiert mit*

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} \leq c.$$

Wenn $f \leq O(g)$, dann sagt man auch, dass f asymptotisch durch g beschränkt *wird. Beispielsweise wird $n \mapsto 2 \cdot \log_2 n \cdot n$ asymptotisch durch $n \mapsto n^2$ beschränkt.*

Definition 7.2 *Seien $f : \mathbb{N} \to \mathbb{R}^+$, $g : \mathbb{N} \to \mathbb{R}^+$ Funktionen. Wir schreiben $f \leq o(g)$ gdw.*

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0.$$

Offensichtlich gilt $f \leq O(f)$ für jedes f und $f \leq o(f)$ für kein f. Aus $f \leq o(g)$ folgt $f \leq O(g)$. Für $f(n) = 2 \cdot \log_2 n \cdot n$ und $g(n) = n^2$ gilt $f \leq o(g)$.

Mit Hilfe der O–Notation können wir nun die Laufzeitkomplexitätsklassen definieren.

Definition 7.3 *Sei $\top$ eine Turing–Maschine, die bei allen Eingaben hält. Die* Laufzeit von $\top$ bei Eingabe von w *ist die Zahl der Rechenschritte, die $\top$ bei Eingabe von w durchführt, bis sie hält. Die* Laufzeit von $\top$ *ist die Funktion $f : \mathbb{N} \to \mathbb{N}$, so dass für alle $n \in \mathbb{N}$ gilt: $f(n)$ ist die maximale Laufzeit von $\top$ bei Eingabe eines Wortes w der Länge n.*

[1]Hierbei ist $\mathbb{R}^+$ die Menge der positiven reellen Zahlen.

Definition 7.4 *Sei $f : \mathbb{N} \to \mathbb{N}$ eine Funktion. Die* zu f gehörige Laufzeitkomplexitätsklasse *ist die Menge aller entscheidbaren Sprachen L, für die Folgendes gilt: es gibt eine Turing–Maschine $\top$ mit Laufzeit $g : \mathbb{N} \to \mathbb{N}$, so dass $g \leq O(f)$ und $\top(w) \downarrow +$ gdw. $w \in L$ und $\top(w) \downarrow -$ gdw. $w \notin L$ für alle Worte w des Eingabealphabets.*

So gehört nun beispielsweise die Sprache

$$\{\underbrace{0\ldots0}_{n \text{ viele}} : n \text{ gerade}\}$$

zur zur Identität $n \mapsto n$ gehörigen Laufzeitkomplexitätsklasse. Die Sprache

$$\{\underbrace{0\ldots0}_{2^n \text{ viele}} : n \in \mathbb{N}\}$$

gehört zur zur Funktion $n \mapsto n^2$ gehörigen Laufzeitkomplexitätsklasse.

SAT gehört zur zur Funktion $n \mapsto 2^n$ gehörigen Laufzeitkomplexitätsklasse. Dies scheint zu besagen, dass SAT "schwieriger" ist als die beiden erstgenannten Probleme.

Für eine Funktion $f : \mathbb{N} \to \mathbb{N}$ bezeichnen wir im Folgenden die zu f gehörige Laufzeitkomplexitätsklasse mit $K(f)$.

Betrachten wir die Sprache

$$L = \{\underbrace{0\ldots0}_{n \text{ viele}}\underbrace{1\ldots1}_{n \text{ viele}} : n \in \mathbb{N}\}.$$

Es ist nicht schwer einzusehen, dass L zu $K(n \mapsto n^2)$ gehört. Etwas mehr Arbeit ist erforderlich, um zu zeigen, dass L zu $K(n \mapsto \log_2 n \cdot n)$ gehört. Dies wird durch die folgende Turing–Maschine $\top$ geleistet.

Sei $\Sigma = \{0, 1\}$, $\Gamma = \{0, 1, x, \sqcup\}$. Sei $w \in \Sigma^*$ gegeben. $\top$ sieht zunächst nach, ob in w eine 0 rechts von einer 1 vorkommt. In diesem Falle verwirft $\top$ die Eingabe w. Andernfalls geht der Kopf zum linken Bandende zurück und $\top$ durchläuft die folgende Schleife: Es werde nachgesehen, ob eine gerade Anzahl an Zeichen auf dem Band nicht durchgestrichen (d.h. durch ein x ersetzt) sind — andernfalls verwerfe man die Eingabe; sodann streiche man jede zweite 0 und sodann jede zweite 1. Falls am Ende nur noch Nullen oder nur noch Einsen auf dem Band übrig bleiben, verwerfe man die Eingabe. Falls am Ende Nichts übrig bleibt, akzeptiere man die Eingabe.

Wir definieren nun die Komplexitätsklasse P

Definition 7.5 *P ist die Menge aller Sprachen, die in einer zu einer Polynomfunktion gehörigen Laufzeitkomplexitätsklasse liegt.*

Für $f(n) = a_k n^k + a_{k-1} n^{k-1} + \ldots + a_0$ und $g(n) = n^k$ gilt $g \leq 0(f)$. Wir können also schreiben

$$P = \bigcup_{k \in \mathbb{N}} K(n \mapsto n^k).$$

Beispielsweise liegen also die Sprachen

$$\{\underbrace{0\ldots0}_{n \text{ viele}} : n \text{ gerade}\},$$
$$\{\underbrace{0\ldots0}_{2^n \text{ viele}} : n \in \mathbb{N}\} \text{ und}$$
$$\{\underbrace{0\ldots0}_{n \text{ viele}}\underbrace{1\ldots1}_{n \text{ viele}} : n \in \mathbb{N}\}$$

in P. Für die erste und die letzte dieser Sprachen ergibt sich dies auch aus der folgenden Aussage:

Lemma 7.6 *Sei $L \subset \Sigma^*$ eine kontextfreie Sprache. Dann ist L in P.*

Beweis: Wir zeigen zunächst, dass jede kontextfreie Sprache entscheidbar ist. Sei $L \subset \Sigma^*$ kontextfrei. Auf Grund von Lemma 2.3 existiert dann eine kontextfreie Grammatik in Chomskyscher Normalform, so dass die Symbolfolgen aus Σ^*, die diese Grammatik generiert, genau diejenigen sind, die zu L gehören.

Sei $w \in \Sigma^*$. Wir können dann wie folgt entscheiden, ob $w \in L$ oder nicht. Angenommen, $w \in L$. Sei dann

$$s \equiv w_0 \Rightarrow w_1 \Rightarrow \ldots \Rightarrow w_{N-1} \equiv w$$

eine Generierung von w, bei der in jedem Schritt genau eine Regel auf eine vorkommende Variable angewandt wird. Sei $w \not\equiv \emptyset$. Wenn dann $n > 0$ die Länge von w ist, dann muss bei obiger Generierung $n-1$–fach eine Regel der Form

$$v \mapsto uu'$$

(wobei v, u, u' Variablen sind) und n–fach eine Regel der Form

$$v \mapsto t$$

(wobei v Variable und t Terminal ist) angewandt werden. Es gilt also $N-1 = 2n-1$, d.h. $N = 2n$. Dies zeigt: wenn $w \in L, w \not\equiv \emptyset$, die Länge $n > 0$ hat, dann gilt für jede Generierung

$$s \equiv w_0 \Rightarrow w_1 \Rightarrow \ldots \Rightarrow w_{N-1} \equiv w$$

von w, bei der in jedem Schritt genau eine Regel auf eine vorkommende Variable angewandt wird, dass $N = 2n$.

Sei nun $w \in \Sigma^*$ beliebig. Sei $w \not\equiv \emptyset$, und sei $n > 0$ die Länge von w. Wenn dann k die Anzahl der Regeln der Grammatik ist, dann gibt es höchstens $(k \cdot \frac{N}{2})^{N-1}$ Generierungen

$$s \equiv w_0 \Rightarrow w_1 \Rightarrow \ldots \Rightarrow w_{N-1},$$

bei denen in jedem Schritt genau eine Regel auf eine vorkommende Variable angewandt wird. Wir können alle diese Generierungen für $N = 2n$ durchgehen und sehen, ob mittels einer von diesen das vorgelegte w generieren wird.

Dies zeigt, dass L entscheidbar ist, aber es zeigt offensichtlich noch nicht, dass L in P ist. Hierzu brauchen wir "dynamisches Programmieren". Wir sagen, dass $w \in \Sigma^*$ aus der Variablen $v \in V$ (wobei v nicht notwendigerweise die Startvariable sein muss) generiert werden kann gdw. eine Generierung

$$v \equiv w_0 \Rightarrow w_1 \Rightarrow \ldots \Rightarrow w_{N-1} \equiv w$$

mit Hilfe der vorliegenden kontextfreien Grammatik existiert.

Wir können die Frage, ob ein gegebenes Wort $w \in \Sigma^*$ der Länge $n \geq 2$ aus $v \in V$ generiert werden kann, wie folgt auf die Frage, ob gewisse Worte der Länge $< n$ aus Variablen v_0, v_1 generiert werden können, zurückführen. Es gibt $n-1$ Möglichkeiten, w in zwei nichttriviale Teile w_0 und w_1 aufzuspalten, d.h. zu schreiben

$$w \equiv w_0 w_1,$$

wobei $w_0, w_1 \in \Sigma^*$ und w_0 und w_1 beide positive Längen haben. Für jede solche Aufspaltung und für jede Regel

$$v \mapsto v_0 v_1$$

der Grammatik gilt offensichtlich: wenn w_0 aus v_0 und wenn w_1 aus v_1 generiert werden kann, dann kann w aus v generiert werden.

Wir können nun die Frage, ob ein gegebenes $w \in \Sigma^*$ in L ist, schnell folgendermaßen entscheiden. Sei $n \in \mathbb{N}$ die Länge von w,

$$w \equiv t_0 t_1 \ldots t_{n-1},$$

wobei $t_0, \ldots, t_{n-1}$ Terminale sind. Wir konstruieren dann eine $n \times n$ Matrix

$$M = \begin{pmatrix} a_{00} & \cdots & a_{0n-1} \\ \vdots & & \vdots \\ a_{n-1\,0} & \cdots & a_{n-1n-1} \end{pmatrix}$$

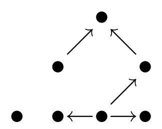
wie folgt: für jedes $i \leq j < n$ sei a_{ij} die Menge aller Variablen $v \in V$, so dass das Teilwort

$$t_i t_{i+1} \ldots t_{j-1} t_j$$

von w aus v generiert werden kann. Für $i > j$ sei $a_{ij} = \emptyset$. Mit Hilfe des oben dargestellten Verfahrens können wir sehr leicht durch Rekursion nach $j - i$ die Matrixeintragungen a_{ij} bestimmen. w kann dann (aus der Startvariablen s) generiert werden gdw. $s \in s_{0n-1}$. Man überzeugt sich, dass dieses Verfahren beweist, dass L in P ist. □

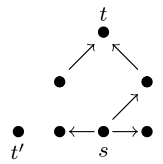
Wir werden nun einige andere Beispiele von Sprachen, die in P liegen, kennen lernen. Wenn eine Sprache L in P liegt, dann ist in gewisser Weise "schnell" entscheidbar, ob ein gegebenes w in L liegt oder nicht. Unsere Turing–Maschine, die das SAT–Problem entscheidet, ist *nicht* schnell in diesem Sinne. Wir werden uns später der Frage zuwenden, ob $P \in SAT$ oder nicht. Die Klasse P entspricht einer gängigen Auffassung nach der Klasse aller Probleme, die mit Hilfe von Computern im wahren Leben tatsächlich gelöst werden können.

Betrachten wir das *Pfadproblem.* Ein *gerichteter Graph* ist eine endliche Menge von Punkten zusammen mit einer Menge von Pfeilen, die von Punkten zu anderen Punkten zeigen. Beispiel:



Formal ist ein gerichteter Graph einfach eine Teilmenge G von $M \times M$, wobei M eine endliche Menge (von Punkten) ist. Dabei bedeutet $(p, q) \in G$, dass ein Pfeil von p nach q zeigt.

Sei nun ein gerichteter Graph $G \subset M \times M$ gegeben, und seien zwei Punkte s und t aus M als *Start–* und *Zielpunkt* ausgezeichnet. Das Pfadproblem lautet: gibt es durch G einen Pfad von s nach t? Beispiel:



Hier gibt es einen Pfad von s nach t, nicht aber von s nach t'.

Formal ist ein Pfad durch $G \subset M \times M$ von s nach t eine Folge $s_0, s_1, \ldots, s_{N-1}$ (für ein $N \in \mathbb{N}$), so dass $s_0 = s$, $s_{N-1} = t$ und $(s_i, s_{i+1}) \in G$ für alle $i < N-2$.

Die Frage, ob es durch G einen Pfad von s nach t gibt, lässt sich folgendermaßen beantworten: Offensichtlich kann es nicht sinnvoll, bzw. nötig sein, auf einen Pfad von s nach t ein und denselben Punkt mehrmals aufzusuchen. Wenn M nun n Elemente hat, dann gibt es $(n-1)!$ "potentielle" Pfade der Länge $n-1$, die s als Startpunkt besitzen und die keinen Punkt mehrmals aufsuchen, d.h. $(n-1)!$ Folgen der Form

$$s_0 = s, s_1, \ldots, s_{n-1},$$

wobei $s_i \neq s_j$ für $i \neq j$. Wenn $t = s_k$ (für $k < n$), dann können wir nachsehen, ob für alle $i < k$ gilt, dass $(s_i, s_{i-1}) \in G$.

Wenn wir das Pfadproblem mit diesem Ansatz lösen, dann ist die Laufzeit der entsprechenden Turing–Maschine sehr groß, nämlich "größenordnungsmäßig" $n \to (n-1)!$.

Es gibt aber eine schnellere Methode. Seien G, s, t gegeben. Im nullten Durchgang markieren wir den Punkt s. Im $(k-1)^{\text{ten}}$ Durchgang markieren wir alle Punkte, die durch einen Pfeil von bereits markierten Punkten aus erreichbar sind; falls allerdings keine neuen Punkte markiert würden, brechen wir dieses Verfahren ab. Offensichtlich kann es, wenn n die Zahl der Punkte aus M ist, höchstens n derartige Durchgänge geben. Nun gibt es offenbar einen Pfad durch G von s nach t gdw. am Ende der Punkt t markiert ist.

Es ist nicht schwer zu sehen, dass mit Hilfe von dieser Idee gezeigt werden kann, dass das Pfadproblem in $K(n \mapsto n^2)$, also insbesondere in P liegt.

Ein weiteres Problem, das nicht auf den ersten Blick in P liegt, lautet: sind zwei gegebene natürliche Zahlen n und m relativ prim?[2] Ein nahe liegender Lösungsansatz untersucht zu jeder natürlichen Zahl $q \leq \min(n, m)$

[2] n, m sind relativ prim gdw. 1 die größte Zahl ist, die sowohl n als auch m teilt.

(oder $q \leq \sqrt{\min(n,m)}$) ob q sowohl n als auch m teilt. Wenn nun die Zahlen n und m etwa in Dualdarstellung in eine Turing–Maschine eingegeben werden, dann ist die Länge der Eingabe zirka $\log_2(n) + \log_2(m)$; die Rechenzeit aber ist größenordnungsmäßig gleich $\min(n,m)$ (oder $\sqrt{\min(n,m)}$). Bei einer Eingabe der Länge k ist die Rechenzeit dann größenordnungsmäßig gleich 2^k.

Das Problem, ob zwei gegebene natürliche Zahlen relativ prim sind, ist dennoch in P. Man zeigt dies mit Hilfe des Euklidschen Algorithmus.

Wir wollen uns nun Problemen zuwenden, die scheinbar schwierig sind. Wir führen zunächst eine Variante des Begriffs "Turing–Maschine" ein, nämlich den der "nichtdeterministischen Turing–Maschine".

Während eine Turing–Maschine stur gemäßihres Programms (d.h. gemäßihrer Übergangsfunktion δ) rechnet, also deterministisch arbeitet, können nichtdeterministische Turing–Maschinen im Rahmen ihrer Möglichkeiten Entscheidungen treffen.

Eine *nichtdeterministische Turing–Maschine* $\top$ besteht formal aus den folgenden Objekten:

(1) Einer endlichen Menge Q von *Zuständen*, wobei es drei ausgezeichnete Zustände gibt: den *Anfangszustand* $q_0 \in Q$, den *positiven Endzustand* q_+ und den *negativen Endzustand* q_-.

(2) Ein endliches *Alphabet* Γ, d.h. eine Menge von Symbolen, die das Leerzeichen $\sqcup$ enthalten soll; eine nichtleere Teilmenge Σ von Γ, die nicht $\sqcup$ enthält, ist als *Eingabealphabet* ausgezeichnet.

(3) Eine *nichtdeterministische Übergangsfunktion* $\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\}) \setminus \emptyset$, die Paaren (q, γ) mit $q \in Q$ und $\gamma \in \Gamma$ eine nichtleere Menge von Tripeln (q', b, x) mit $q' \in Q, b \in \Gamma$ und $x \in \{L, R\}$ zuordnet.

(1) und (2) sind also genau wie im Falle der (klassischen, deterministischen) Turing–Maschine. In (3) wurde $Q \times \Gamma \times \{L, R\}$ durch die Potenzmenge $\mathcal{P}(Q \times \Gamma \times \{L, R\})$ von $Q \times \Gamma \times \{L, R\}$ (d.h. der Menge aller Teilmengen von $Q \times \Gamma \times \{L, R\}$) als Bildbereich von δ ersetzt. Während jeder konkreten Berechnung wählt die nichtdeterministische Turing–Maschine ein Element aus $\delta(q, \gamma)$ (für das aktuelle Paar (q, γ)) aus, um nach ihm zu verfahren.

Ein *Rechenvorgang* der nichtdeterministische Turing–Maschine $\top$ ist eine (endliche oder unendliche) Folge von *Konfigurationen* K_n. Dabei sieht jede Konfiguration genau so aus wie im Falle (klassischer) Turing–Maschinen. Sei q der aktuelle Zustand, und sei γ das Symbol, das aktuell gelesen wird.

Dann ergibt sich K_{n+1} aus K_n, indem $\top$ zunächst ein Tripel $(q', b, x) \in \delta(q, \gamma)$ auswählt und sodann verfährt wie im Falle der klassischen Turing–Maschinen.

Offensichtlich ist jede (klassische) Turing–Maschine auch eine nichtdeterministische Turing–Maschine, aber nicht umgekehrt.

Wir wollen die Funktionsweise nichtdeterministischer Turing–Maschinen zunächst an einem Beispiel illustrieren, nämlich einer Variante der Turing–Maschine $\top_{SAT}$, die im 2. Kapitel zur Lösung des SAT–Problems gebaut wurde. Diese Maschine produziert nach der Eingabe einer aussagenlogischen Formel φ auf dem Rechenband zunächst das Wort

$$\varphi\#\varphi\#\#\underbrace{0\ldots0}_{n \text{ viele}},$$

wobei n die Länge von φ ist. Zu einem späteren Zeitpunkt sieht es auf dem Band so aus:

$$\varphi\#\varphi\#\#d,$$

wobei d die Dualdarstellung einer Zahl zwischen 0 und $2^n - 1$ ist. Die Zahl d wird zur Buchführung benutzt und diktiert eine Belegung der in φ vorkommenden Aussagenvariablen durch Nullen und Einsen. $\top_{SAT}$ testet für eine solche Belegung, ob sie φ erfüllt. (Dabei wird mit dem Vorkommnis von φ zwischen # und ## gearbeitet.)

Es ist nicht schwer zu sehen, dass die Laufzeit von $\top_{SAT}$ "größenordnungsmäßig" $n \mapsto 2^n \cdot n^2$ insbesondere also nicht polynomiell ist, so dass $\top_{SAT}$ bezeugt, dass $SAT \in K(2^n \cdot n^2)$.

Wir können nun $\top_{SAT}$ sehr leicht in eine nichtdeterministische Turing–Maschine $\top^n_{SAT}$ umbauen, die eine erheblich geringere Laufzeit besitzt. Die Maschine $\top^n_{SAT}$ soll dabei nach der Eingabe einer aussagenlogischen Formel φ der Länge n wie folgt vorgehen. $\top^n_{SAT}$ "rät" zunächst eine Folge d von Nullen und Einsen der Länge n und schreibt diese, abgetrennt von φ durch #, auf das Rechenband, so dass dort

$$\varphi\#d$$

geschrieben steht. Sodann wird d analog wie im Falle von $\top_{SAT}$ benutzt, um eine Belegung der in φ vorkommenden Aussagenvariablen zu diktieren. Für diese eine Belegung testet $\top^n_{SAT}$ schließlich, ob sie φ erfüllt.

Das "Raten" von d ist der einzige Teil der Berechnung, der nichtdeterministisch stattfindet. Formal geht dies etwa folgendermaßen vor sich. Bei

Eingabe von φ produziert $\top^n_{SAT}$ zuallererst das Wort

$$\varphi\#\varphi,$$

indem # hinter φ geschrieben wird und dahinter wiederum φ kopiert wird. Sodann begibt sich der Kopf zum ersten Symbol des rechten Vorkommnisses von φ und $\top^n_{SAT}$ begibt sich in einen ausgezeichneten "Rate"–Zustand q_R. Die Übergangsfunktion δ sei nun so beschaffen, dass

$$\delta(q_R, \gamma) = \{(q_R, 0, R), (q_R, 1, R)\}$$

für alle $\gamma \in \Sigma$ (dem Eingabealphabet). Dies bedeutet, dass der Kopf nun solange nach rechts läuft, bis $\sqcup$ erscheint, und in jedem Schritt das aktuell gelesene Symbol durch 0 oder 1 ersetzt wird.

Definition 7.7 *Sei $\top$ eine nichtdeterministische Turing–Maschine, so dass jeder Rechenvorgang bei jeder Eingabe endlich ist.*[3] *Die Laufzeit von $\top$ ist die Funktion $f : \mathbb{N} \to \mathbb{N}$, so dass für alle $n \in \mathbb{N}$ gilt: $f(n)$ ist die maximale Laufzeit von $\top$ bei Eingabe eines Wortes w, welches höchstens die Länge n besitzt.*

Im obigen Beispiel ist dies nicht der Fall, aber im Allgemeinen kann natürlich die Länge der Berechnung bei Eingabe von w davon abhängen, welche Auswahlen im Verlauf der Berechnung getroffen werden. So existiert im Falle von nichtdeterministischen Turing–Maschinen $\top$ nicht notwendig "die" Laufzeit von $\top$ bei Eingabe von w.

Die Laufzeit von $\top^n_{SAT}$ ist "größenordnungsmäßig" $n \mapsto n^2$. Das SAT–Problem ist mit Hilfe von nichtdeterministischen Turing–Maschinen also wesentlich schneller lösbar als mit Hilfe von (klassischen, deterministischen) Turing–Maschinen.

Definition 7.8 *Sei $f : \mathbb{N} \to \mathbb{N}$ eine Funktion. Die* zu f gehörige nichtdeterministische Laufzeitkomplexitätsklasse *ist die Menge aller Sprachen L, für die Folgendes gilt: es gibt eine nichtdeterministische Turing–Maschine $\top$ mit Laufzeit $g : \mathbb{N} \to \mathbb{N}$, so dass $g \leq O(f), w \in L$ gdw. es einen Rechenvorgang von $\top$ bei Eingabe von w gibt, der w akzeptiert, und $w \notin L$ gdw. alle Rechenvorgänge von $\top$ bei Eingabe von w das Wort w verwerfen.*[4]

[3] Dies bedeutet, dass $\top$ bei jeder Eingabe hält, unabhängig davon, welche Auswahlen im Verlauf der Berechnung getroffen werden.

[4] Ein Rechenvorgang akzeptiert/verwirft w gdw. dieser Rechenvorgang mit einer Konfiguration endet, bei der die Maschine sich am Schluss im Zustand q_+/q_- befindet.

Beispielsweise bezeugt $\top^n_{SAT}$, dass SAT in der zu $n \mapsto n^2$ gehörigen nichtdeterministischen Komplexitätsklasse liegt.

Für eine Funktion $f : \mathbb{N} \to \mathbb{N}$ schreiben wir im Folgenden $K^n(f)$ für die zu f gehörige nichtdeterministische Komplexitätsklasse.

Definition 7.9 *NP ist die Menge aller Sprachen, die in einer zu einer Polynomfunktion gehörigen nichtdeterministischen Laufzeitkomplexitätsklasse liegen.*

Die obigen Überlegungen zeigen, dass SAT in NP liegt. Da jede (klassische) Turing–Maschine auch eine nichtdeterministische Turing–Maschine ist, gilt trivialerweise

$$P \subset NP.$$

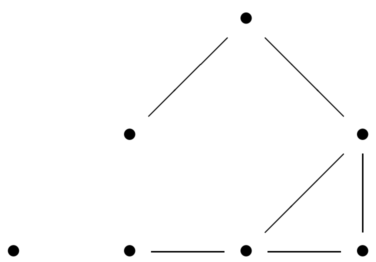
Es wird vermutet, dass NP verschieden ist von P. Hierfür gibt es aber keinen Beweis. Wir werden später sehen, dass $P \neq NP$ gdw. $SAT \notin P$.

Hier sind zwei weitere Beispiele für Probleme in NP.

Sei G ein gerichteter Graph. Ein *Hamiltonscher Weg* durch G ist ein Weg durch G, der jeden Knoten genau einmal besucht. Genauer können wir dies wie folgt definieren. Sei $G \subset M \times M$, wobei M endlich ist. Dann ist ein Hamiltonscher Weg durch G von $s \in M$ nach $z \in M$ eine Folge $(s_0, s_1), (s_1, s_2), \ldots , (s_{N-2}, s_{N-1})$, wobei $s_0 = s_1, s_{N-1} = z, (s_i, s_{i+1}) \in G$ für alle $i < N - 1$ und $s_i \neq s_j$ für alle $i < j < N$. (In diesem Falle muss offensichtlich N die Zahl der Elemente von M sein.)

Das *Problem Hamiltonscher Wege* lautet wie folgt. Gegeben seien ein gerichteter Graph $G \subset M \times M$ und $s, z \in M$. Existiert dann ein ein Hamiltonscher Weg durch G von s nach z? Man überzeugt sich leicht davon, dass dieses Problem in NP liegt. Wir können eine nichtdeterministische Turing–Maschine zunächst einen Weg durch G raten lassen und sie sodann entscheiden lassen, ob dieser Weg ein Hamiltonscher Weg von s nach z ist.

Unser nächstes Beispiel ist das Clique–Problem. Ein *ungerichteter Graph* ist eine endliche Menge von Punkten, zusammen mit einer Menge von Verbindungsstrecken, die gewisse Punkte mit anderen Punkten verbinden. Beispiel:



Formal ist ein ungerichteter Graph eine Teilmenge G von $[M]^2$, wobei M eine endliche Menge (von Punkten) ist. Hierbei ist $[M]^2$ die Menge aller zweielementigen Teilmengen von M und $\{s, t\} \in G$ bedeutet, dass eine Verbindungsstrecke von s mit t existiert.

Sei $k \in \mathbb{N}$. Eine *k–Clique in G* ist eine k–elementige Teilmenge X von M, so dass $[X]^2 \subset G$, d.h. so dass je zwei Punkte aus X durch eine Verbindungsstrecke verbunden sind.

Im obigen Beispiel gibt es etwa genau eine 3–Clique, aber keine 4–Clique.

Das Clique–Problem lautet wie folgt. Gegeben seien ein ungerichteter Graph G und eine natürliche Zahl k. Gibt es eine k–Clique in G?

Wir hätten NP auch mit Hilfe von "Verifikatoren" anstelle von nichtdeterministischen Turing–Maschinen definieren können. Sei $L \subset \Sigma^*$ eine Sprache. Dann heißt L *verifizierbar* gdw. eine (klassische) Turing–Maschine $\top$ existiert, so dass Folgendes gilt. Wenn $w \in L$, dann gibt es ein(en Zeugen) $z \in \Sigma^*$, so dass $\top(w\#z) \downarrow +$; wenn $w \notin L$, so gilt für alle $z \in \Sigma^*$, dass $\top(w\#z) \downarrow -$.[5]

Lemma 7.10 *Sei $L \subset \Sigma^*$ eine Sprache. Die folgenden Aussagen sind äquivalent.*

(1) L ist verifizierbar.

(2) Es gibt eine nichtdeterministische Turing–Maschine $\top$, so dass jeder Rechenvorgang bei jeder Eingabe endlich ist und so dass $w \in L$ gdw. es einen Rechenvorgang bei Eingabe von w gibt, der w akzeptiert.

Lemma 7.11 *Eine Sprache $L \subset \Sigma^*$ ist in NP gdw. L verifizierbar ist, wobei dies durch eine Turing–Maschine $\top$ bezeugt wird, deren Laufzeit polynomiell in der ersten Komponente der Eingabe ist.*

[5]Hierbei sei # ein neues (Trennungs–)Symbol.

Kapitel 8

Der Satz von Cook und Levin

Für Sprachen $L_0, L_1 \subset \Sigma^*$ hatten wir in Definition 6.3 den Begriff der "(Turing–)Reduzibilität" definiert. Wir interessieren uns nun für eine Verschärfung dieses Begriffs.

Definition 8.1 *Eine Funktion $f : \Sigma^* \to \Sigma^*$ heißt in polynomieller Zeit berechenbar gdw. es eine Turing–Maschine $\top$ mit polynomieller Laufzeit gibt, so dass für alle $w \in \Sigma^* \top$ bei Eingabe von w hält, wobei am Ende $f(w)$ auf dem Rechenband geschrieben ist.*

Lemma 8.2 *Seien $L_0, L_1 \subset \Sigma^*$ Sprachen mit $L_0 \leq_p L_1$. Dann folgt aus $L_1 \in P$, dass auch $L_0 \in P$.*

Beweis: Der Beweis ist analog zum Beweis von Lemma 6.4, wobei wir allerdings auf Laufzeiten zu achten haben. Sei $w \in L_0$ gdw. $f(w) \in L_1$ für alle $w \in \Sigma^*$, wobei f in polynomieller Zeit berechenbar ist. Werde dies etwa durch die Turing–Maschine $\top$ bezeugt, deren Laufzeit $\leq 0(n \mapsto n^k)$ sei. Sei $\top'$ eine Turing–Maschine mit Laufzeit $\leq 0(n \mapsto n^m)$, so dass $\top'$ ein Wort $w' \in \Sigma^*$ akzeptiert/verwirft gdw. $w' \in L_1/w' \notin L_1$. Sei U diejenige Turing–Maschine, die bei Eingabe von w zunächst das Programm von $\top$ laufen lässt und $f(w)$ berechnet und sodann das Programm von $\top'$ laufen lässt und entscheidet, ob $f(w) \in L_1$ oder nicht. Für hinreichend großes $n \in \mathbb{N}$ existieren dann Konstanten $c_0, c_1 \in \mathbb{R}^+$, so dass für alle w der Länge n Folgendes gilt. Da die Laufzeit von $\top$ bei Eingabe von w durch $c_0 \cdot n^k$ beschränkt ist, ist die Länge von $f(w)$ höchstens $c_0 \cdot n^k$, so dass die Laufzeit von $\top'$ bei Eingabe von $f(w)$ durch $c_1 \cdot (c_0 \cdot n^k)^m = c_1 \cdot c_0^m \cdot n^{km}$ beschränkt ist. Damit ist die Laufzeit von $U \leq 0(n \mapsto n^{km})$. Dies zeigt, dass L_0 in P ist.

Wir werden später Beispiele für $\leq_p$ kennen lernen. Wir wollen zunächst die allgemeine Theorie vorantreiben.

Definition 8.3 *Sei $L \subset \Sigma^*$ eine Sprache. L heißt NP–vollständig gdw. $L \in NP$ und für alle $L_0 \subset \Sigma^*$ mit $L_0 \in NP$ gilt $L_0 \leq_p L$.*

Satz 8.4 *Sei $L \subset \Sigma^*$ NP–vollständig. Dann gilt $P = NP$ gdw. $L \in P$.*

Beweis: Wenn $P = NP$, dann folgt $L \in P$ aus $L \in NP$. Sei umgekehrt $L \in P$. Sei $L_0 \in NP$. Dann folgt $L_0 \in P$ aus $L \in P$ mit Hilfe des obigen Lemmas.

Die Frage, ob $P = NP$ oder nicht, kann also entschieden werden, indem für ein konkretes NP–vollständiges Problem gezeigt wird, dass dieses in P bzw. nicht in P liegt. Wir werden sogleich mehrere NP–vollständige Probleme kennen lernen.

Lemma 8.5 *Sei $L \subset \Sigma^*$ NP–vollständig, und sei $L' \subset \Sigma^*$ in NP, wobei $L \leq_p L'$. Dann ist auch L' NP–vollständig.*

Beweis: Der Beweis von Lemma 8.2 zeigt auch, dass die Relation $\leq_p$ transitiv ist. Sei $L_0 \subset \Sigma^*$ in NP. Dann folgt aus $L_0 \leq_p L$ und $L \leq_p L'$, dass $L_0 \leq_p L'$. L' ist also NP–vollständig.

Satz 8.6 (Cook, Levin) *SAT ist NP–vollständig.*

Beweis: Wir haben im letzten Kapitel gesehen, dass $SAT \in NP$. Es bleibt also zu zeigen, dass jedes $L \in NP$ in polynomieller Zeit auf SAT reduzierbar ist.

Sei $L \subset \Sigma^*$ in NP. Sei $\top$ eine nichtdeterministische Turing–Maschine mit polynomieller Laufzeit, so dass gilt: $w \in L$ gdw. es einen Rechenvorgang von $\top$ bei Eingabe von w gibt, der w akzeptiert. Sei $k \in \mathbb{N}$ so, dass die Laufzeit von $\top \leq 0(n \mapsto n^k)$ ist. Wir nehmen im Folgenden tatsächlich an, dass die Laufzeit von $\top$ bei Eingabe eines Wortes w der Länge n höchstens $n^k - 3$ ist. (Andernfalls müssen im Folgenden nur die Notationen etwas geändert werden.)

Sei nun $w \in \Sigma^*$, und habe w die Länge n. Ein Rechenvorgang von $\top$ bei Eingabe von w ist dann eine Folge $(K_i : i < l)$ von Konfigurationen, wobei K_{l-1} den Zustand q_+ oder q_- beinhaltet. Es gilt $l \leq n^k$. Wir können uns einen solchen Rechenvorgang durch eine $n^k \times n^k$ Matrix (die wir jetzt auch "Tableau" nennen wollen) wie folgt veranschaulichen. Dabei sei $w \equiv \gamma_0 \dots \gamma_{n-1}$, wobei $\gamma_j \in \Sigma$ für alle $j < n$.

0	#	q_0	γ_0	γ_1	γ_2	$\cdots$	γ_{n-1}	$\sqcup$	$\cdots$	$\sqcup$	#
1	#										#
2	#										#
	#										#
	#										#
	#										#
	#										#
n^k-1	#										#
	0	1	2								

Dabei stellen wir uns vor, dass der Rechenvorgang ähnlich wie im Beweis von Satz 6.1 mitgeteilt wird. Sie i^{te} Zeile dieses Tableaus steht für die i^{te} Konfiguration der betrachteten Berechnung, wobei wir wieder etwa durch $\gamma'_0, \dots \gamma'_{r-1} q \gamma'_r \dots \gamma'_s \sqcup \dots \sqcup$ die Tatsache mitteilen, dass $\top$ aktuell im Zustand q ist, der Kopf auf der (mit γ'_r beschrifteten) r^{ten} Zeile steht und sich auf dem Rechenband die Inschrift $\gamma'_0 \dots \gamma'_{r-1} \gamma'_r \dots \gamma'_s \sqcup \dots$ befindet.

Wir werden sogleich den Übergang von K_i zu K_{i+1} ähnlich wie im Beweis von Satz 6.1 erfassen.

Unsere Aufgabe ist es, eine Turing–Maschine R mit polynomieller Laufzeit zu bauen, so dass R die Eingabe w in eine aussagenlogische Formel φ "übersetzt", so dass $w \in L$ gdw. φ erfüllbar ist. Die Tatsache, dass $w \in L$ gilt, ist aber äquivalent damit, dass es ein Tableau wie oben gibt, so dass q_+ eine der Tableaueintragungen ist und dieses Tableau für einen Rechenvorgang von $\top$ bei Eingabe von w steht.

Betrachten wir $Q \cup \Gamma \cup \{\#\}$. Hierbei ist Q die Menge der Zustände von $\top$, Γ (disjunkt von $\top$) ist das Schreibalphabet von $\top$ und # kommt nicht in $Q \cup \Gamma$ vor. Wir können sehr leicht in eindeutiger Art und Weise jeden Tripel $(i, j, s) \in \{0, \dots, n^k-1\} \times \{0, \dots, n^k-1\} \times (Q \cup \Gamma \cup \{\#\})$ eine Aussagenvariante aus $\{A_0, A_1, A_2, \dots\}$ zuordnen.[1] Die (i, j, s) zugeordnete Aussagenvariable wollen wir einfach mit $A_{i,j,s}$ bezeichnen. Sei $\overline{\beta} : \{A_0, A_1, \dots\} \to \{0, 1\}$. Dann soll $\overline{\beta}(A_{i,j,s}) = 1$ für die Tatsache stehen, dass die Tableauzeile mit Index (i, j) die Eintragung s enthält. Entsprechend steht $\overline{\beta}(A_{i,j,s}) = 0$ dafür, dass diese Tableauzeile nicht die Eintragung s enthält.

Wir konstruieren nun die aussagenlogische Formel φ. Die Maschine R wird dann ähnlich wie wir bei gegebenen w die Formel φ konstruieren. Es ist leicht sich zu überzeugen, dass R eine polynomielle Laufzeit hat.

Die Formel φ besitzt die Gestalt $\varphi_0 \wedge \varphi_1 \wedge \varphi_2 \wedge \varphi_3$.

[1]Wir könnten z.B. dem Tripel (i, j, s) die Aussagenvariable $A_{2^i.3^j.5^a}$ zuordnen, wobei s das a^{te} Element von $Q \cup \Gamma \cup \{\#\}$ gemäßeiner fixierten Aufzählung von $Q \cup \Gamma \cup \{\#\}$ ist.

Die Teilformel φ_0 sagt, dass jede Tableauzeile genau ein Symbol $s \in Q \cup \Gamma \cup \{\#\}$ enthält:

$$\varphi_0 \equiv \bigwedge_{0 \le i,j < n^k} \Big[\bigvee_s A_{i,j,s} \wedge \bigwedge_{s \neq t} (\neg A_{i,j,s} \vee \neg A_{i,j,t}) \Big].$$

Hierbei steht $\bigwedge$ für Konjunktionen und $\bigvee$ für Disjunktionen. Z.B. ist $\bigvee_s A_{i,j,s}$ die (endliche!) Disjunktion aller $A_{i,j,s}$, wobei $s \in Q \cup \Gamma \cup \{\#\}$.

Die Teilformel φ_1 beschreibt vollständig die oberste Tableauzeile:

$$\begin{aligned}\varphi_1 \equiv A_{0,0,\#} \wedge A_{0,1,q_0} \wedge A_{0,2,\gamma_0} \wedge \ldots \wedge A_{0,n+1,\gamma_{n-1}} \wedge \\ A_{0,n+2,\sqcup} \wedge \ldots \wedge A_{0,n^k-2,\sqcup} \wedge A_{0,n^k-1,\#}\end{aligned}$$

Die Teilformel φ_2 besagt, dass die Eingabe w akzeptiert wird:

$$\varphi_2 \equiv \bigvee_{0 \le i,j < n^k} A_{i,j,q_+}.$$

Die Teilformel φ_3 schließlich drückt aus, dass das Tableau einen Rechenvorgang von $\top$ bei Eingabe von w entspricht. Hierzu müssen wir mitteilen, dass die $i+1^{\text{te}}$ Tableauzeile aus der i^{ten} Tableauzeile gemäßder Übergangsfunktion δ hervorgeht. (Falls die i^{te} Tableauzeile bereits q_+ enthält, dann soll die $i+1^{\text{te}}$ Zeile einfach eine Wiederholung der i^{ten} Zeile sein.)

Für diesen Zweck betrachten wir 2×3–Fenster des Tableaus:

0	#	q_0	γ_0	γ_1	γ_2	$\cdots$	γ_{n-1}	$\sqcup$	$\cdots$	$\sqcup$	#
1	#										#
2	#										#
	#										#
	#										#
	#						s_0	s_1	s_2		#
	#						s_3	s_4	s_5		#
	#										#
n^k-1	#										#
	0	1	2								n^k-1

Ein solches Fenster besteht, für $i < n^k - 1$ und $j < n^k - 2$, aus den Tableauzeilen mit Index $(i,j), (i,j+1), (i,j+2), (i+1,j), (i+1,j+1)$ und $(i+1,j+2)$. Für die Tatsache, dass das Tableau einen Rechenvorgang wiedergibt, ist notwendig und hinreichend, dass jedes im Tableau vorkommende 2×3–Fenster im folgenden Sinne *legal* ist.

Sei $(q', b, L) \in \delta(q, a)$. Dann ist für alle $\gamma \in \Gamma$

$$\begin{array}{|c|c|c|}\hline \gamma & q & a \\ \hline q' & \gamma & b \\ \hline\end{array}$$

ein legales Fenster. Weiterhin seien für alle $\gamma, \gamma'' \in \Gamma$ und $\gamma' \in \Gamma \cup \{\#\}$

$$\begin{array}{|c|c|c|}\hline q & a & \gamma' \\ \hline \gamma & b & \gamma' \\ \hline\end{array} \quad ,$$

$$\begin{array}{|c|c|c|}\hline a & \gamma'' & \gamma' \\ \hline b & \gamma'' & \gamma' \\ \hline\end{array} \quad ,$$

$$\begin{array}{|c|c|c|}\hline \gamma' & \gamma & q \\ \hline \gamma' & q' & \gamma \\ \hline\end{array} \quad ,$$

und

$$\begin{array}{|c|c|c|}\hline \gamma' & \gamma' & \gamma \\ \hline \gamma' & \gamma' & q' \\ \hline\end{array}$$

legale Fenster. Darüber hinaus seien auch

$$\begin{array}{|c|c|c|}\hline \# & q & a \\ \hline \# & q' & b \\ \hline\end{array}$$

und

$$\begin{array}{|c|c|c|}\hline q & a & \gamma \\ \hline q' & b & \gamma \\ \hline\end{array}$$

legale Fenster. (Letztere werden für die Situation benötigt, dass der Kopf am linden Bandende steht und aufgefordert wird, nach links zu gehen.)

Sei nun $(q', b, R) \in \delta(q, a)$. Dann sei für alle $\gamma \in \Gamma$

$$\begin{array}{|c|c|c|}\hline q & a & \gamma \\ \hline b & q' & \gamma \\ \hline\end{array}$$

ein legales Fenster. Weiterhin seien für alle $\gamma \in \Gamma$ und $\gamma' \in \Gamma \cup \{\#\}$

$$\begin{array}{|c|c|c|}\hline \gamma' & q & a \\ \hline \gamma' & b & q' \\ \hline\end{array} \quad ,$$

$$\begin{array}{|c|c|c|}\hline \gamma' & \gamma & q \\ \hline \gamma' & \gamma & b \\ \hline\end{array}$$

und

a	γ	γ'
q'	γ	γ'

legale Fenster.

Schließlich seien alle Fenster der Form

x	y	z
x	y	z

mit $x, y, z \in \Gamma \cup \{\#, q_+\}$ legal. Alle übrigen Fenster gelten als illegal.

Wir setzen nun

$$\varphi_3 = \bigwedge_{\substack{i < n^k - 1 \\ j < n^k - 2}} \bigvee_{\substack{s_0, \ldots, s_5 \\ \text{ist legal}}} (A_{i,j,s_0} \wedge A_{i,j,s_1} \wedge A_{i,j+2,s_2} \wedge A_{i+1,j,s_3} \wedge A_{i+1,j+1,s_1} \wedge A_{i+1,j+2,s_5}).$$

Hierbei schreiben wir "$s_0, \ldots, s_5$ ist legal" für die Tatsache, dass da 2×3–Fenster

s_0	s_1	s_2
s_3	s_4	s_5

legal ist.

Die Formel $\varphi \equiv \varphi_0 \wedge \varphi_1 \wedge \varphi_2 \wedge \varphi_3$ ist erfüllbar gdw. $w \in L$. $\square$

Die folgende nur scheinbar einfachere Variante von SAT, die wir SAT_3 nennen, ist ebenfalls NP–vollständig.

Eine aussagenlogische Formel φ ist ein konjunktiver Normalform (vgl. Kapitel 1) gdw. φ die Gestalt

$$\bigwedge_i \bigvee_j \varphi_{i,j}$$

besitzt, wobei jedes $\varphi_{i,j}$ entweder atomar oder die Negation einer atomaren Formel ist. Eine aussagenlogische Formel ist in 3–*beschränkter konjunktiver Normalform* gdw. φ die Gestalt

$$\bigwedge_i (\varphi_{i,0} \vee \varphi_{i,1} \vee \varphi_{i,2})$$

besitzt, wobei jedes $\varphi_{i,j}$ entweder atomar oder die Negation einer atomaren Formel ist.

Das Problem SAT_3 lautet wie folgt. Sei φ eine aussagenlogische Formel in 3–beschränkter konjunktiver Normalform. Ist φ erfüllbar?

Satz 8.7 *SAT_3 ist NP–vollständig.*

Beweis: Diese Aussage ergibt sich aus dem Beweis des Satzes von Cook und Levin. Die zum vorgelegten w produzierte Formel φ kann in polynomieller Zeit in eine aussagenlogisch äquivalente Formel in 3–beschränkter konjunktiver Normalform umgewandelt werden.

Wir zeigen nun, dass da CLIQUE–Problem NP–vollständig ist. Dies ergibt sich jetzt sofort aus dem folgenden

Lemma 8.8 *$SAT_3 \leq_P$ CLIQUE.*

Beweis: Sei φ eine aussagenlogische Formel in 3–beschränkter konjunktiver Normalform. Sei etwa

$$\varphi \equiv \bigwedge_{i<n} (\varphi_{i,0} \vee \varphi_{i,1} \vee \varphi_{i,2}),$$

wobei jedes $\varphi_{i,j}$ entweder atomar oder Negation einer atomaren Aussage ist. Wir setzen $M = \{0, \dots, n-1\} \times \{0, 1, 2\}$ und ordnen φ den ungerichteten Graphen $G \subset [M]^2$ zu, wobei $\{(i,j), (i',j')\} \subset G$ gdw. $i \neq i'$ und es nicht der Fall ist, dass $\varphi_{i,j}$ die Negation von $\varphi_{i',j'}$ oder umgekehrt $\varphi_{i',j'}$ die Negation von $\varphi_{i,j}$ ist. Zwei Punkte $(i,j), (i',j')$ aus M sind also verbunden gdw. $i \neq i'$ und $\varphi_{i,j}$ nicht $\varphi_{i',j'}$ widerspricht. Jedes $\overline{\beta} : \{A_0, \dots\} \to \{0,1\}$, das die Erfüllbarkeit von φ bezeugt, entspricht einer n–CLIQUE in G und umgekehrt.

Auch das Problem Hamiltonscher Wege ist NP–vollständig:

Lemma 8.9 *SAT_3 ist in polynomieller Zeit auf das Problem Hamiltonscher Wege reduzierbar.*

Beweis: Sei wieder

$$\varphi \equiv \bigwedge_{i<n} (\varphi_{i,0} \vee \varphi_{i,1} \vee \varphi_{i,2})$$

gegeben, wobei jedes $\varphi_{i,j}$ entweder atomar oder Negation einer atomaren Formel ist.

Seien o.B.d.A. $A_0, A_1, \dots, A_{l-1}$ die Aussagenvariablen, die in φ vorkommen. (Es gibt also $l \leq 3n$.)

Wir setzen $M = \{s_0, s_1, \dots, s_l\} \cup \{0, 1, \dots, n-1\} \cup \{(i,j) : i < l \wedge j < 3n+1\}$ [2] Hierbei stehen $0, 1, \dots, n-1$ für die Konjunktionsglieder von φ, die Punkte (i,j) für $j < 3n+1$ stehen in gewisser Weise für die Aussagenvariable

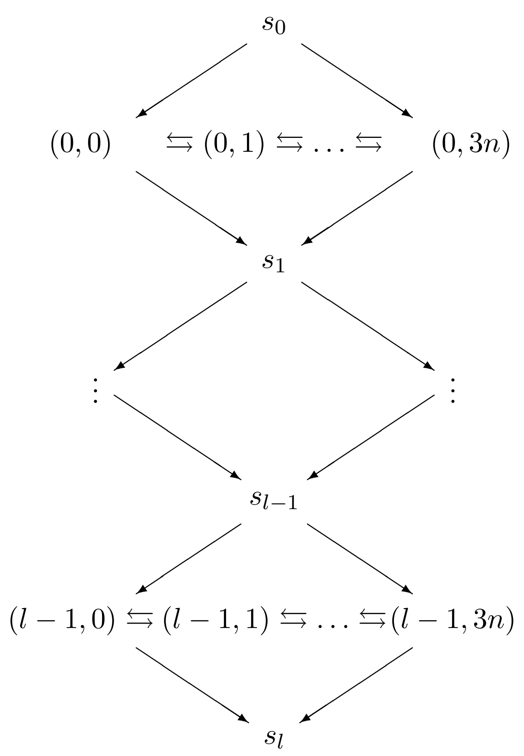
[2] Wir setzen voraus, dass die drei Mengen, deren Vereinigung M ist, paarweise disjunkt sind.

A_i, und die s_k sind “Übergangspunkte”. s_0 soll der Start und s_l der Zielpunkt werden.

Wir definieren nun $G \subset M \times M$ wie folgt.

Für alle $i < l$ und $j < 3n$ seien $((i,j),(i,j+1)) \in G$ und $((i,j+1),(i,j)) \in G$. Außerdem seien für alle $i < l$ sowohl $(s_i,(i,0)) \in G$ als auch $(s_i,(i,3n)) \in G$. Ferner seien für alle $i < l-1$ sowohl $((i,0),s_{i+1}) \in G$ als auch $((i,3n),s_{i+1}) \in G$.

Bislang ergibt sich damit das folgende Bild:



Die Punkte $0,1,\ldots,n-1$ sind noch nicht beteiligt. Im Moment gibt es zwei Hamiltonsche Wege durch $G \subset M \setminus \{0,1,\ldots,n-1\}$ von s_0 nach s_l.

Wir verbinden nun die Punkte $0,1,\ldots,n-1$ so mit Punkten (i,j), dass jeder Hamiltonsche Weg von s_0 nach s_l in konsistenter Weise für jedes Konjunktionsglied $\varphi_{i,0} \vee \varphi_{i,1} \vee \varphi_{i,2}$ von φ entscheidet, ob $\varphi_{i,0}, \varphi_{i,1}$ oder $\varphi_{i,2}$ als wahr bewertet werden soll oder nicht. Dabei ist jedes $\varphi_{i,j}$ eines der $A_0, A_1, \ldots, A_{l-1}$ oder Negation eines der $A_0, A_1, \ldots, A_{l-1}$.

Betrachten wir das i^{te} Konjunktionsglied

$$\varphi_{i,0} \vee \varphi_{i,1} \vee \varphi_{i,2}$$

von φ. Sei, für $j < 3, A_{f(i,j)}$ so, dass $\varphi_{i,j} \equiv A_{f(i,j)}$, oder $\varphi_{i,j} \equiv \neg A_{f(i,j)}$. Wir verbinden dann wie folgt den Punkt $i \in M$ mit anderen Punkten aus M.

Es seien $(i, 3 \cdot f(i,j) + 2) \in G$ und $(3 \cdot f(i,j) + 3, i) \in G$, falls $\varphi_{i,j} \equiv A_{f(i,j)}$. Und es seien $(i, 3 \cdot f(i,j) + 3) \in G$ und $(3 \cdot f(i,j) + 2, i) \in G$, falls $\varphi_{i,j} \equiv \neg A_{f(i,j)}$. Dies beendet die Beschreibung von G.

Jeder Hamiltonsche Weg durch G von s_0 nach s_l basiert auf einem der beiden Hamiltonschen Wege durch die Einschränkung von G auf $M \backslash \{0, 1, \dots, n-1\}$. Allerdings müssen Umwege so eingeflochten werden, dass auch die Punkte aus $\{0, 1, \dots, n-1\}$ längs des Weges besucht werden. Ein Hamiltonscher Weg durch G von s_0 nach s_l liefert dann ein $\overline{\beta} : \{A_0, \dots\} \to \{0, 1\}$, das die Erfüllbarkeit von φ zeigt, wie folgt: wir setzen $\overline{\beta}(A_i) = 0$ bzw. 1 gdw. wir auf dem Hamiltonschen Weg von s_i aus nach $(i, 0)$ bzw. nach $(i, 3n)$ gehen. Umgekehrt liefert auch jedes $\overline{\beta}$, das die Erfüllbarkeit von φ zeigt, einen Hamiltonschen Weg durch G von s_0 nach s_l.

Kapitel 9

Berechenbarkeit

Die folgenden Begriffe werfen ein neues Licht auf Fragen der Berechenbarkeitstheorie.

Definition 9.1 *Die Klasse der primitiv rekursiven (p.r.) Funktionen ist die kleinste Klasse K von Funktionen $f : A \to \mathbb{N}$, wobei $A \subset \mathbb{N}^k$ für ein $k > 0$, so dass gilt:*

(1) K enthält die Null– *und die* Nachfolgerfunktion 0 *und S, wobei $0 : \mathbb{N} \to \mathbb{N}, 0(n) = 0$ für alle n, und $S : \mathbb{N} \to \mathbb{N}, S(n) = n + 1$ für alle n.*

(2) Für $1 \leq i \leq n$ enthält K die Projektionsfunktion $U_i^n : \mathbb{N}^n \to \mathbb{N}$, *wobei $U_i^n(m_1, \dots, m_n) = m_i$ für alle $m_1, \dots, m_n \in \mathbb{N}$.*

(3) Wenn $f : A \to \mathbb{N}$ in K ist, wobei $A \subset \mathbb{N}^k$, und wenn $g_1, \dots, g_k$ in K sind, dann ist auch die Komposition von f und $g_1, \dots, g_k$, *nämlich die Funktion h mit $h(\vec{m}) = f(g_1(\vec{m}), \dots, g_k(\vec{m}))$ für alle $\vec{m}$,*[1] *in K enthalten. Hierbei ist h genau für diejenigen $\vec{m}$ definiert, so dass $g_i(\vec{m})$ für alle i und sodann $f(g_1(\vec{m}), \dots, g_k(\vec{m}))$ definiert ist.*

(4) K ist abgeschlossen bzgl. primitiver Rekursion, *d.h. wenn f und g in K sind, dann ist auch h in K, wobei für alle $\vec{m}$ gilt:*

$$h(\vec{m}, 0) = f(\vec{m})$$
$$h(\vec{m}, n + 1) = g(\vec{m}, n, h(\vec{m}, n)).$$

[1] Wir schreiben hier und im Folgenden $\vec{m}$ für $m_1, \dots, m_n$ (für ein n) und setzen hier voraus, dass der Urbildbereich von allen g_i Teilmenge von $\mathbb{N}^n$ ist.

Hierbei ist $h(\vec{m}, 0)$ definiert gdw. $f(\vec{m})$ definiert ist, und $h(\vec{m}, n+1)$ ist definiert gdw. $h(\vec{m}, n)$ und sodann auch $g(\vec{m}, n, h(\vec{m}, n))$ definiert ist.

Lemma 9.2 *Die folgenden Funktionen sind p.r.*

(a) *Die konstante Funktion.*

(b) $n + m$ [2]

(c) $n \cdot m$

(d) $n \dot{-} 1$, *wobei* $n \dot{-} 1 = n - 1$ *für* $n > 0$ *und* $n \dot{-} 1 = 0$ *für* $n = 0$

(e) $n \dot{-} m$, *wobei* $n \dot{-} m = n - m$ *für* $n \geq m$ *und* $n \dot{-} m = 0$ *für* $n < m$

(f) $\max(n, m)$

(g) $\min(n, m)$

(h) *Der Abstand von* n *und* m.

(i) $eq(n, m)$, *wobei* $eq(n, m) = 1$ *für* $n = m$ *und* $eq(n, m) = 0$ *für* $n \neq m$

(j) $neq(n, m)$, *wobei* $neq(n, m) = 1$ *für* $n \neq m$ *und* $neq(n, m) = 0$ *für* $n = m$.

(k) $leq(n, m)$, *wobei* $leq(n, m) = 1$ *für* $n \leq m$ *und* $leq(n, m) = 0$ *für* $n > m$.

(l) $le(n, m)$, *wobei* $le(n, m) = 1$ *für* $n < m$ *und* $lt(n, m) = 0$ *für* $n \geq m$.

Beweis: (a): Dies benutzt (1) und (4) von Definition 9.1 und die Beobachtung, dass die konstante Funktion $c\colon \mathbb{N} \to \mathbb{N}$ $c(0) = 0$ und $c(n+1) = S(c(n)) = c(n)+1$ erfüllt. Die Beweise von (b)–(l) sind analog einfach, bauen aufeinander auf und benutzen die folgenden Beobachtungen.

(b): $n + 0 = c(n) = n$ und $n + (m+1) = S(n+m) = (n+m)+1$.

(c): $n \cdot 0 = 0$ und $n \cdot (m+1) = (n \cdot m) + n$.

(d): $0 \dot{-} 1 = 0$ und $(n+1) \dot{-} 1 = c(n) = n$.

(e): $n \dot{-} 0 = n$ und $n \dot{-} (m+1) = (n \dot{-} m) \dot{-} 1$.

(f): $\max(n, m) = (m \dot{-} n) + n$.

(g): $\min(n, m) = (n+m) \dot{-} \max(n, m)$.

(h): Der Abstand $|n - m|$ von n und m ist $\max(n, m) \dot{-} \min(n, m)$.

(i): $eq(n, m) = 1 \dot{-} |n - m|$.

[2] Darunter verstehen wir die Funktion $n, m \mapsto n + m$. Analoges gilt für das Folgende.

(j): $neq(n,m) = 1 \dot{-} eq(n,m)$.
(k): $leq(n,m) = eq(|n-m|, m \dot{-} n)$.
(l): $le(n,m) = leq(n,m) \cdot neq(n,m)$. □

Definition 9.3 *Die Klasse der* partiell rekursiven Funktionen *ist die kleinste Klasse K von Funktionen $f : A \to \mathbb{N}$, wobei $A \subset \mathbb{N}^k$ für ein $k > 0$, so dass gilt:*

(1) – (4) wie in Definition 9.1, sowie

(5) K ist abgeschlossen unter Minimierung*, d.h. wenn f in K ist, dann ist auch g in K, wobei $g(\vec{m})$ = das kleinste n, so dass $f(\vec{m}, 0), \ldots, f(\vec{m}, n)$ alle definiert sind und $f(\vec{m}, n) = 0$.*

$g(\vec{m})$ ist nicht definiert, falls es kein solches n gibt.

Offensichtlich ist jede p.r. Funktion auch partiell rekursiv. Während jede p.r. Funktion total auf $\mathbb{N}$ ist (d.h. Definitionsbereich $\mathbb{N}$ besitzt), kann eine partiell rekursive Funktion *partiell* sein. Es gibt Beispiele totaler partiell rekursiver Funktionen, die nicht p.r. sind.

Wenn f und g wie in (5) von Definition 9.3 sind, dann schreibt man oft $\mu n.f(\vec{m}, n) = 0$ für $g(\vec{m})$. Das so definierte μ wird auch als "(unbeschränkter) μ–Operator" bezeichnet.

Lemma 9.4 *Sei K entweder die Klasse der p.r. Funktionen oder die Klasse der partiell rekursiven Funktionen. K besitzt die folgenden Abschlusseigenschaften:*

(a) Wenn $g, f_0, \ldots, f_k$ alle in K sind, dann ist auch h in K, wobei

$$h(\vec{m}) = \begin{cases} f_0(\vec{m}), \textit{ falls } g(\vec{m}) = 0 \\ \vdots \\ f_{k-1}(\vec{m}), \textit{ falls } g(\vec{m}) = k-1 \\ f_k(\vec{m}), \textit{ falls } g(\vec{m}) \geq k \end{cases}$$

$h(\vec{m})$ ist genau dann definiert, wenn sowohl $g(\vec{m})$ als auch das entsprechende $f_i(\vec{m})$ definiert ist.

(b) Wenn f in K ist, dann ist auch g in K, wobei

$$g(\vec{m}, n) = \begin{cases} \text{das kleinste } q \leq n, \text{ so dass} \\ f(\vec{m}, 0), \ldots, f(\vec{m}, q) \text{ alle definiert} \\ \text{sind und } f(\vec{m}, q) = 0, \text{ falls} \\ \text{ein solches } q \text{ existiert} \\ \\ n + 1, \text{ falls alle } f(\vec{m}, 0), \ldots, f(\vec{m}, n) \\ \text{definiert aber ungleich } 0 \text{ sind.} \end{cases}$$

$g(\vec{m}, n)$ ist nicht definiert, falls keiner dieser beiden Fälle zutrifft.

Beweis: (a): Die Funktion h lässt sich offenbar wie folgt darstellen. $h(\vec{m}) =$

$$f_0(\vec{m}) \cdot eq(g(\vec{m}), 0) + \cdots + f_{k-1}(\vec{m}) \cdot eq(g(\vec{m}), k - 1) + f_k(\vec{m}) \cdot leq(k, g(\vec{m})).$$

Damit ergibt sich die Behauptung sofort mit Hilfe von Lemma 9.2.

(b): Die Funktion g lässt sich wie folgt darstellen. $g(\vec{m}, n) =$

$$\sum_{i=0}^{n} neq(\prod_{j=0}^{i} f(\vec{m}, j), 0).$$

Mit Hilfe von Lemma 9.2 und Definition 9.1 (3) ergibt sich dann sofort die gewünschte Aussage. □

Wenn f und g wie in (2) von Lemma 9.4 sind, dann schreibt man oft $\mu q \leq n.g(\vec{m}, q) = 0$ für $g(\vec{m}, n)$. Das so definierte μ wird auch als "beschränkter μ–Operator bezeichnet.

Wenn wir wieder die natürliche Zahl n mit einer Folge von $n+1$ Sternen

$$\underbrace{* \cdots *}_{n+1 \text{ viele}}$$

identifizieren, dann können wir versuchen, partiell rekursive Funktionen mit Turing–berechenbaren Funktionen im Sinne von Definition 4.1 in Beziehung zu setzen. Jede partiell rekursive Funktion ist offenbar Turing–berechenbar. Wir wollen nun die Umkehrung hiervon zeigen, siehe Satz 9.9. Dies erfordert ein wenig Vorbereitung.

Definition 9.5 *Eine Menge (bzw. Relation) $A \subset \mathbb{N}^k$ heißt* primitiv rekursiv (p.r.) *gdw. die charakteristische Funktion χ_A von A p.r. ist, wobei*

$$\chi_A(\vec{m}) = \begin{cases} 1 \text{ gdw. } \vec{m} \in A \\ 0 \text{ gdw. } \vec{m} \notin A. \end{cases}$$

Die Lemmata 9.2 und 9.4 liefern sofort die folgende Aussage.

Lemma 9.6 *Die folgenden Relationen sind p.r.*

(a) $\{0\}$

(b) $\{1\}$

(c) $\{(n,m) \ : \ n = m\}$

(d) $\{(n,m) \ : \ n \leq m\}$

(e) $\{(n,m,s) \ : \ s = n + m\}$

(f) $\{(n,m,p) \ : \ p = n \cdot m\}$

Darüberhinaus gilt:

(g) Wenn A, $B \subset \mathbb{N}^k$ p.r. sind, dann ist auch $A \cap B$ p.r.

(h) Wenn $A \subset \mathbb{N}^k$ p.r. ist, dann ist auch $\mathbb{N}^k \setminus A$ p.r.

(i) Wenn $A \subset \mathbb{N}^{k+1}$ p.r. ist, dann ist auch $B \subset \mathbb{N}^{k+1}$ p.r., wobei

$$B = \{(\vec{m}, n) \ : \ \exists q \leq n (\vec{m}, q) \in A\}.$$

Beweis: (a)–(f) sind klar. (g): Mit χ_A und χ_B ist auch $\chi_{A \cap B} = \chi_A \cdot \chi_B$ p.r., wobei $(\chi_A \cdot \chi_B)(\vec{m}) = \chi_A(\vec{m}) \cdot \chi_B(\vec{m})$ für alle $\vec{m}$). (h): Mit χ_A ist auch $\chi_{\mathbb{N}^k \setminus A} = 1 \dot{-} \chi_A$ p.r., wobei $(1 \dot{-} \chi_A)(\vec{m}) = 1 \dot{-} \chi_A(\vec{m})$ für alle $\vec{m}$. (i): Nach Lemma 9.4 ist $g(\vec{m}, n)$ p.r., wobei $g(\vec{m}, n) = \mu q \leq n.(\vec{m}, q) \in A$ das kleineste $q \leq n$ mit $(\vec{m}, q) \in A$ sei, falls ein solches existiert, und $g(\vec{m}, n) = n + 1$ sonst. Dann ist offenbar $\chi_B(\vec{m}, n) = leq(g(\vec{m}, n), n)$. □

Aus Lemma 9.6 (g) und (h) ergibt sich sofort, dass die Klasse aller p.r. Relationen unter beliebigen Booleschen Kombinationen abgeschlossen ist, d.h. wenn A, B p.r. sind, dann auch $A \cup B$, $A \setminus B$, etc.

Lemma 9.7 *Sei $A \subset \mathbb{N}^{k+1}$ p.r. Dann gibt es eine partiell rekursive Funktion $g \colon B \to \mathbb{N}$, für deren Definitionsbereich B gilt:*

$$B = \{\vec{m} \ : \ \exists q (\vec{m}, q) \in A\}.$$

Beweis: Sei $f(\vec{m}, q) = 1 \dot{-} 1\chi_A$, und sei $g(\vec{m}) = \mu q.f(\vec{m}, n) = 0 = \mu q.(\vec{m}, q) \in A$ wie in Definition 9.3 (5). Offensichtlich ist dann g wie gewünscht. $\square$

Lemma 9.6 besagt, dass einfach definierbare Relationen p.r. sind. Hierbei bedeutet "einfach definierbar", dass sie mit Hilfe von Σ_0–Formeln in der Sprache der elementaren Zahlentheorie definierbar sind. Die Klasse der Σ_0–Formeln wird dabei wie folgt definiert.

Die Menge der Terme der Sprache der elementaren Zahlentheorie werde wie folgt definiert. Ein *Term* ist ein Ausdruck, der in jeder Menge K mit folgenden Eigenschaften liegt:

(1) Sowohl 0 als auch 1 (die Konstanten für die Null bzw. die Eins) sind in K.

(2) Jede der Variablen $v_0, v_1, \ldots$ ist in K.

(3) Wenn τ und σ in K sind, dann auch $(\tau + \sigma)$ und $(\tau \cdot \sigma)$.

Eine *atomare Formel* der Sprache der elementaren Zahlentheorie ist ein Ausdruck der Gestalt $\tau = \sigma$ oder $\tau \leq \sigma$, wobei τ und σ Terme sind.

Ein Ausdruck heißt eine Σ_0–*Formel* gdw. er zu allen Mengen S von Ausdrücken gehört, die Folgendes erfüllen:

(F1) Jede atomare Formel gehört zu S.

(F2) Mit φ und ψ gehören auch $\neg\varphi$, $(\varphi \wedge \psi)$, $(\varphi \vee \pi)$, $(\varphi \to \psi)$ und $(\varphi \leftrightarrow \psi)$ zu S.

(F3) Wenn φ zu S gehört, wenn $i \in \mathbb{N}$, und wenn τ ein Term ist, in dem die Variable v_i nicht vorkommt, dann gehören auch $\forall v_i(v_i < \tau \to \varphi)$ und $\exists v_i(v_i < \tau \wedge \varphi)$ zu S.

Ein Ausdruck heißt eine Formel gdw. er zu allen Mengen S von Ausdrücken gehört, die die letzten beiden Bedingungen (F1) und (F2) sowie Folgendes erfüllen.

(F3') Wenn φ zu S gehört, wenn $i \in \mathbb{N}$, und wenn τ ein Term ist, dann gehören auch $\forall v_i\ \varphi$ und $\exists v_i\ \varphi$ zu S.

Wir schreiben $\forall v_i(v_i < \tau \to \varphi)$ auch als $\forall v_i < \tau\varphi$ und $\exists v_i(v_i < \tau \wedge \varphi)$ auch als $\exists v_i < \tau\varphi$. Eine beschränkte Formel enthält keine "unbeschränkten" Quantoren. Eine Σ_0–Formel heißt auch "beschränkt".

Eine Formel φ heißt Σ_1 gdw. $i_0, \ldots, i_{n-1} \in \mathbb{N}$ und eine beschränkte Formel ψ existieren, so dass

$$\varphi \equiv \exists v_{i_0} \ldots \exists v_{i_{n-1}} \psi.$$

Wir wollen jetzt sehen, dass es für jede Turing–Maschine $\top$ eine Σ_1–Formel $\chi_\top$ gibt, so dass $\chi_\top(w)$ gilt gdw. $\top(w)\downarrow$. Nun, "$\chi_\top(w)$" ist zunächst nicht sinnvoll, da w als Eingabe für $\top$ eine Symbolfolge, jedoch keine natürliche Zahl ist. Wir können aber Symbolfolgen mit Zahlen "identifizieren"; eine solche Identifizierung heißt *"Gödelisierung"*.

Sei $\widetilde{\Gamma}$ ein endliches Alphabet, etwa $\widetilde{\Gamma} = \{\gamma_0, \ldots, \gamma_{n-1}\}$. Wir können dann zuerst, für $i < n$, γ_i mit der natürlichen Zahl $i+1$ identifizieren. Sei $p_0 = 2, p_1 = 3, p_2 = 5, \ldots$ die natürliche Aufzählung aller Primzahlen. Wir können dann ein Wort $w \in \widetilde{\Gamma}^*$, d.h. eine endliche Folge $\gamma_{i_0}\gamma_{i_1}\ldots\gamma_{i_{m-1}}$ von Symbolen aus $\widetilde{\Gamma}$ der Länge $m \in \mathbb{N}$ mit der natürlichen Zahl

$$p_0^{i_0} \cdot p_1^{i_1} \cdot \ldots \cdot p_{m-1}^{i_{m-1}}$$

identifizieren. Jedes Wort entspricht damit einer Zahl und jeder Zahl der Gestalt

$$p_0^{i_0} \cdot p_1^{i_1} \cdot \ldots \cdot p_{m-1}^{i_{m-1}},$$

wobei $m \in \mathbb{N}$ und $i_k < n$ für alle $k < m$, entspricht ein Wort. Wir bezeichnen die dem Wort $w \in \widetilde{\Gamma}^*$ zugeordnete Zahl als die *Gödelnummer von w*, in Zeichen $\ulcorner w \urcorner$. Es gilt sodann:

Satz 9.8 *Für jede Turing–Maschine $\top$ existiert eine Σ_1–Formel $\chi_\top$ der Sprache der elementaren Zahlentheorie, so dass*

$$\chi_\top(\ulcorner w \urcorner) \text{ gilt gdw. } \top(w)\downarrow$$

für alle Worte w über dem Eingabealphabet von $\top$.

Beweis: Sei $\top$ gegeben durch Q, Γ (und Σ), sowie δ. Sei etwa

$$Q = \{q_0, q_+, q_-, q_1, q_2, \ldots, q_{n-1}\}$$

und

$$\Gamma = \{\sqcup, x_0, x_1, \ldots, x_{m-1}\}.$$

Wir setzen $\widetilde{\Gamma} = Q \cup \Gamma$.[3] Eine Konfiguration im Rahmen eines Rechenvorgangs von $\top$ wollen wir durch das aktuell auf dem Band niedergeschriebene Wort kodieren, wobei wir vor die Stelle, auf der der Kopf steht, den aktuellen Maschinenzustand einfügen. Beispielsweise kodieren wir die Konfiguration

				q $\triangledown$								
0	1	0	0	1	0	1	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$

durch die Folge

$$0\,1\,0\,0\,q\,1\,0\,1,$$

oder auch durch die Folge der Form

$$0\,1\,0\,0\,q\,1\,0\,1\,\sqcup \ldots \sqcup\,.$$

Ein Rechenvorgang von $\top$, der ja eine Folge von Konfigurationen ist, kann dann als Folge derartiger Folgen, oder mit Hilfe von einfacher Buchhaltung nur als Folge von Symbolen aus $\widetilde{\Gamma}$ angesehen werden.

Sei w eine Eingabe mit $\top(w)\downarrow$. Sei $l \in \mathbb{N}$ die Laufzeit von $\top$ bei Eingabe von w, d.h. der Rechenvorgang von $\top$ bei Eingabe von w besteht aus l Konfigurationen $K_r, r < l$, und bricht dann ab, da $\top$ in den Zustand q_+ oder q_- geraten ist. Sei $k \in \mathbb{N}$ minimal, so dass die ersten k Zellen bei diesem Rechenvorgang benötigt werden, d.h. so dass die $(k+1)^{\text{te}}$, $(k+2)^{\text{te}}$ etc. Zelle in keinem Rechenschritt besucht werden. Wir können dann den Rechenvorgang von $\top$ bei Eingabe von w sehr leicht durch eine Folge $s_0, s_1 \ldots s_{l(k+1)-1}$ der Länge $l(k+1)$ von Symbolen aus $\widetilde{\Gamma}$ kodieren: für $r < l$ kodiert der r^{te} Block $s_{r(k+1)}s_{r(k+1)+1} \ldots s_{r(k+1)+k}$ die r^{te} Konfiguration K_r. Offensichtlich können wir jetzt $\top(w)\downarrow$ wie folgt ausdrücken:

Es gibt $l, k \in \mathbb{N}$ und eine Folge $s_0, s_1 \ldots s_{l(k+1)-1}$ der Länge $l(k+1)$ von Symbolen aus $\widetilde{\Gamma}$, wobei gilt:

(a) Der nullte Block $s_0 s_1 \ldots s_{k-1}$ ist von der Gestalt q_0, gefolgt von w, gefolgt von Leerzeichen $\sqcup$.

(b) Für $r + 1 < l$ ergibt sich der $(r+1)^{\text{te}}$ Block $s_{(r+1)(k+1)} \ldots s_{(r+1)(k+1)+k}$ aus dem r^{ten} Block $s_{r(k+1)} \ldots s_{r(k+1)+k}$ durch Anwendung der Übergangsfunktion δ.

[3] Wir nehmen hierbei o.B.d.A. an, dass Q und Γ disjunkt sind.

(c) Der $(l-1)^{\text{te}}$ Block $s_{(l-1)(k+1)} \cdots s_{(l-1)(k+1)+k}$ enthält eines der Symbole q_+ oder q_-.

Mit Hilfe der oben vorgenommenen Gödelisierung können wir dies wiederum wie folgt formulieren:

Es gibt $l, k, f \in \mathbb{N}$, wobei f von der Gestalt

$$p_0^{i_0} \cdot p_1^{i_1} \cdot \ldots \cdot p_{l(k+1)-1}^{i_{l(k+1)-1}}$$

ist mit[4] $i_j < n + m + 3$ und es gilt:

(a) $i_0 = q_0$, für ein $t < k + 1$ ist $i_1 i_2 \ldots i_t$ das Wort w (des Eingabealphabets), wobei $v_0 = \ulcorner w \urcorner$, und $i_{t+1} = \ldots = i_k = \sqcup$.

(b) Für $r + 1 < l$ ergibt sich der $(r+1)^{\text{te}}$ Block $i_{(r+1)(k+1)} \cdots i_{(r+1)(k+1)+k}$ aus dem r^{ten} Block $i_{r(k+1)} \ldots i_{r(k+1)+k}$ durch Anwendung von δ.

(c) $i_u = q_+$ oder $i_u = q_-$ für ein u mit $(l-1)(k+1) \leq u < l(k+1)$.

Eine nähere Inspektion, die im Detail langwierig aber nicht schwierig zu bewerkstelligen ist, zeigt, dass wir schließlich eine Σ_1–Formel $\chi_\top$ finden, die das Gewünschte leistet. $\square$

Wir zeigen nun:

Satz 9.9 *Sei $f \colon B \to \mathbb{N}$ eine Funktion mit Definitionsbereich $B \subset \mathbb{N}$. Dann ist f Turing–berechenbar gdw. f partiell rekursiv ist.*

Beweis: Wir zeigen "$\Longrightarrow$". Sei f eine (womöglich partielle) Turing–berechenbare Funktion von $\mathbb{N}$ nach $\mathbb{N}$. Es gibt also eine Turing–Maschine $\top$, so dass $f(n) = m$ gdw. $\top$ bei Eingabe von $\ulcorner n \urcorner$ hält, wobei am Ende der Berechnung $\ulcorner m \urcorner$ auf dem Band steht. Nach dem Beweis von Satz 9.8 gibt es dann eine Σ_1–Formel φ, so dass

$$f(n) = m \text{ gdw. } \varphi(n, m),$$

und nach Lemma 9.7 gibt es eine partiell rekursive Funktion $g \colon B \to \mathbb{N}$ mit Definitionsbereich

$$B = \{(n, m) \ : \ \varphi(n, m)\}.$$

Es ist leicht zu sehen, dass dann auch f partiell rekursiv ist. $\square$

[4] $\widetilde{\Gamma}$ enthält $n + m + 3$ viele Symbole, die wir mit $0, 1, \ldots, n + m + 2$ identifizieren.

Die hier geführten Diskussionen führen zur *These von Church*, wonach der informelle Begriff der "Berechenbarkeit" mit dem der Turing–Berechenbarkeit zusammenfällt.

Definition 9.10 *Sei f partiell rekursiv, wobei der Urbildbereich von f gleich $\mathbb{N}^k$ für ein $k > 0$ ist. Dann heißt f* rekursiv*.*

Der Begriff der rekursiven Menge (Sprache) wurde in Kapitel 4 definiert (siehe Definition 4.1). Es gilt nun:

Lemma 9.11 *Eine Menge $A \subset \mathbb{N}^k$ ist rekursiv gdw. die charakteristische Funktion χ_A von A rekursiv ist.*