Kolmogorovkomplexität

Intuition und Definitionen

Intuition.

- Die Kolmogorov-Komplexität ist eine *Beschreibungskomplexität*. Sie besagt für jedes x, wie kurz man x beschreiben kann; $K(x)$ ist die kürzestmögliche Länge einer Beschreibung von x.
- Somit misst die Kolmogorovkomplexität in gewissem Sinne Information: Wenn man erfährt, was x ist, gewinnt man nur die Information, die schon in der kürzesten Beschreibung von x steckt; der Informationsgehalt von x ist also $K(x)$.
- Trivialerweise kann man x beschreiben, indem man x direkt so angibt. Wir können das die *triviale Beschreibung* nennen.
- Wenn x regelmässig/strukturiert ist, dann kann man es komprimieren.

  Zum Beispiel das Wort $w = 000000000000000000000111111111111111111$:

  Es hat Länge $|w| = 42$; daher kann man jemandem das w trivial mit 42 Zeichen beschreiben, man schreibt das Wort einfach direkt so hin. Die Beschreibungslänge ist dann 42.

  Aber w ist sehr strukturiert, es besteht aus 21 Nullen und dann 21 Einsen. Das kann man ausnützen, um es als $0^{21}1^{21}$ zu beschreiben. Das ist eine viel kompaktere, komprimiertere Angabe von w, hier ist die Beschreibungslänge etwa 6 Zeichen.
- Wenn sich ein x nicht komprimieren lässt, also völlig unstrukturiert ist, dann nennt man es *zufällig*. In anderen Worten: Ein x ist genau dann zufällig, wenn es keine kürzere Beschreibung als die triviale Beschreibung gibt.
- Konkret heisst ein Wort w zufällig, wenn $K(w) \geq |w|$.
- Damit ist nicht definiert, wann ein Zahl n (also kein Wort) zufällig ist. Man könnte definieren, dass n genau dann zufällig ist, wenn $\mathrm{Bin}(n)$ zufällig ist, also genau dann, wenn $K(n) \geq |\mathrm{Bin}(n)| = \lceil \log(n+1) \rceil$. Die offizielle Definition nennt n aber zufällig, wenn $K(n) \geq \lceil \log(n+1) \rceil - 1$. Das „$-1$“ ist da, weil jede Binärdarstellung einer natürlichen Zahl ausser 0 mit einer 1 beginnt. Wenn man weiss, dass eine Zahl beschrieben wird, muss diese 1 also gar nicht mitgeteilt werden.
- Der Begriff der Zufälligkeit hier hat nichts zu tun mit dem Begriff der Zufälligkeit, wie wir sie aus dem Alltag oder der Wahrscheinlichkeitstheorie kennen.

Formalisierung der Intuition.

Beschreibung Ein Programm P *generiert* („beschreibt“) ein w, wenn es auf dem leeren Wort die Ausgabe w erzeugt. Als Formel kann man dafür $P(\lambda) = w$ schreiben.

Beschreibungslänge Wir messen die Länge eines Programmes P immer im zugehörigen Maschinencode. Das Buch nennt die Länge des Maschinencodes auch die *binäre Länge* von P. Wir können diese Länge als $|P|$ notieren. Wenn ein P ein w generiert, ist das also eine Beschreibung von w der Länge $|P|$.

Beschreibungskomplexität Die Kolmogorov-Komplexität $K(w)$ eines Wortes w ist die Länge eines kürzesten (Pascal-)Programms, das w generiert.
Man kann auch schreiben $K(w) \coloneqq \min\{\, |P|;\ P \text{ gen. } w \,\}$.

Die Kolmogorov-Komplexität $K(n)$ einer Zahl n muss separat definiert werden, denn n ist kein Wort. Wir definieren $K(n)$ via die Binärdarstellung von n als $K(n) \coloneqq K(\mathrm{Bin}(n))$.

Bemerkungen.

- Dass die Definition Pascal als Programmiersprache nutzt, spielt für uns keine Rolle, wir können Programme immer in Pseuo-Code schreiben. Der Grund ist, dass man mit jeder vernünftigen Programmiersprache jede andere simulieren kann. Man kann beispielsweise in Python einen Emulator für Pascal schreiben, der eine konstante Länge hat. Somit kann man also die Programmiersprache wechseln und es kommt nur eine Konstante dazu. Wenn man sich nicht um die Konstante kümmert, können wir also Aussagen treffen, die für alle Programmiersprachen gelten.
- Die triviale Beschreibung von w in einem Programm ist `write(`w`)`. Dieses Programm hat Länge $|w| + c$ für eine Konstante c, die von w unabhängig ist.
- Man könnte die Zufälligkeit von w als $K(w) \geq |w| + c$ definieren, wobei c die konkrete Konstante aus obigem Punkt ist, also die Länge von `write(` und `)` im Maschinencode eines Pascalprogrammes. Das wäre durchaus sinnvoll und problemlos möglich.

  Die offizielle Definition der Zufälligkeit setzt aber $c = 0$. Der Grund ist, dass $c = 0$ näher an der intuitiven Idee liegt, was eine Beschreibung ist; also beispielweise, mit wie vielen Zeichen man w auf einem Blatt hinschreiben kann. Diese Wahl von $c = 0$ passt nicht ganz zu der Definition der Kolmogorovkomplexität via Pascal. Doch letztlich ist Wahl der Konstanten ohnehin unwichtig, weils sie sich beim Wechsel der Programmiersprache ändert, und man eigentlich Aussagen treffen will, die von der Programmiersprache unabhängig sind.
- Wenn ein Programm P ein x generiert, ist es für die Kolmogorovkomplexität völlig egal, wie viel Zeit und Speicher P dazu benötigt; nur die Länge von P ist relevant. Die Zeitkomplexität und die Speicherplatzkomplexität sind uns also völlig egal, es geht nur um die *Beschreibungskomplexität*.

Drei typische Aufgaben zur Kolmogorovkomplexität mit Musterlösung

Aufgabentyp 1, erster Teil

Sei $w_n = 0^{3^{5n^7}}$. Beweisen Sie eine möglichst gute obere Schranke für die Kolmogorov-Komplexität von w_n.

Lösung

Das Programm

$$\begin{aligned} P_n: \quad & \mathrm{v} \coloneqq n \\ & \mathrm{v} \coloneqq 3 \,\hat{}\, (5 * (\mathrm{v} \,\hat{}\, 7)) \\ & \text{for 1 to } v\text{:} \\ & \quad \mathrm{print}(0) \end{aligned}$$

generiert w_n und hat Länge $\underbrace{\lceil \log_2(n+1) \rceil}_{|\operatorname{Bin}(n)|} + \underbrace{c.}_{\substack{\text{Für alles} \\ \text{ausser } n.}}$ für eine Konstante c.

Nach Def. der KK gilt also $K(w_n) \underset{(*)}{\leq} \lceil \log_2(n+1) \rceil$. □

Optional etwas am Ende etwas ausführlicher: In der unendlichen Folge der Programme P_n verändert sich im Programm-Code nur das n, welches im Binärcode die Länge $|P_n| = |\operatorname{Bin}(n)| = \lceil \log_2(n+1) \rceil$ einnimmt, das Restprogramm hat eine konstante Länge c. Wir gehen davon aus, dass der obige Pseudo-Code als Pascal-Programm umgesetzt ist und messen den zugehörigen Maschinencode. Nach Def. von $K(w_n)$ als Länge eines *kürzesten* w_n erzeugenden Pascal-Programms gilt also $K(w_n) \underset{(*)}{\leq} \lceil \log_2(n+1) \rceil$. □.

Aufgabentyp 1, zweiter Teil

Drücken Sie die obige Schranke durch $|w_n|$ aus.

Lösung

Es gilt

$$\begin{aligned} |w_n| &= \left| 0^{3^{5n^7}} \right| = 3^{5n^7} \\ \Rightarrow \quad \log_3 |w_n| &= 5n^7 \\ \Rightarrow \quad \sqrt[7]{\frac{1}{5} \log_3 |w_n|} &= n \end{aligned}$$

Einsetzen in $(*)$ ergibt:

$$K(w_n) \leq \lceil \log_2(\sqrt[7]{\frac{1}{5} \log_3 |w_n|} + 1) \rceil$$

Aufgabentyp 2

Geben Sie eine unendliche streng monoton steigende Folge natürlicher Zahlen $y_1, y_2, \ldots$ an, so dass

$$\exists c \in \mathbb{N} : \forall n \in \mathbb{N} : \qquad K(y_n) \leq \lceil \log_2 \log_3 (\sqrt[5]{y_n}) \rceil + c\,.$$

Lösung

Sei $y_n \coloneqq 2^{x_n}$, also $\mathrm{Bin}(y_n) = 1\underbrace{0\ldots0}_{x_n \text{ Mal}}$, wobei $x_n \coloneqq \left(3^{n+1}\right)^5$.

Wir wählen y_n als Zweierpotenz, um eine einfache Binärdarstellung zu erhalten. (Eine implizite Konvertierung mit `write(`n`)` wäre wahrscheinlich erlaubt, ist so aber nicht notwendig.) Wir können x_n später so festlegen, dass die untenstehende Gleichung $(*)$ erfüllt ist.

Das Programm

$$\begin{aligned} P_n : \quad & \mathrm{v} \coloneqq n \\ & \mathrm{x} \coloneqq (3 \,\hat{}\, (\mathrm{v}+1)) \,\hat{}\, 5 \\ & \mathrm{print}(0) \\ & \text{for 1 to } x\text{:} \\ & \qquad \mathrm{print}(0) \end{aligned}$$

erzeugt $\mathrm{Bin}(y_n)$ und hat Länge $\underbrace{\lceil \log_2(n+1) \rceil}_{|\mathrm{Bin}(n)|} + \underbrace{c.}_{\substack{\text{Für alles} \\ \text{ausser } n.}}$ für eine Konstante c.

Nach Def. von $K(y_n)$ – nämlich als Länge eines kürzesten $\mathrm{Bin}(y_n)$ erzeugenden Pascal-Programms in Maschinencode, als was der obige Pseudo-Code angenommen wird – folgt

$$K(y_n) \coloneqq K(\mathrm{Bin}(y_n)) \leq \lceil \log_2(n+1) \rceil + c$$

Wir müssen jetzt noch n in Abhängigkeit von y_n ausdrücken. Man kann jetzt die geforderte obere Schranke erzeugen, indem man immer eine Funktion und die Umkehrfunktion anwendet. Am Ende entsteht ganz innen das x_n. Wir setzen $x_n = \log y_n$ ein, was die Schranke nur nochmals um ein „log“ verbessert, was nicht nötig, aber möglich ist.

$$\begin{aligned} &= \left\lceil \log_2 \log_3 3^{n+1} \right\rceil + c \\ &= \left\lceil \log_2 \log_3 \sqrt[5]{\left(3^{n+1}\right)^5} \right\rceil + c \\ &\overset{(*)}{=} \lceil \log_2 \log_3 \sqrt[5]{x_n} \rceil + c \\ &= \left\lceil \log_2 \log_3 \sqrt[5]{\log y_n} \right\rceil + c \end{aligned}$$

□

Aufgabentyp 3

Sei $M = \{ 7^i \mid i \in \mathbb{N}, i \leq 2^n - 1 \}$. Beweisen Sie, dass mindestens sieben Achtel der Zahlen in M Kolmogorovkomplexität mindestens $n - 3$ haben.

Lösung

Zu zeigen: M enthält mindestens $\frac{7}{8}|M|$ Zahlen x mit $K(x) \geq n - 3$.
Äquivalent dazu: M enthält höchstens $\frac{1}{8}|M|$ Zahlen x mit $K(x) \leq n - 4$.

Widerspruchsannahme:
M enthält mehr als $\frac{1}{8}|M|$ Zahlen x mit $K(x) \leq n - 4$.
Nach Def. der KK folgt: Mehr als $\frac{1}{8}|M|$ Zahlen werden von einem Programm der Länge höchstens $n - 4$ generiert.
Weil die Zahlen in M paarweise verschieden sind, müssen diese generierenden Programme paarweise verschieden sein.
Es gibt also mehr als $\frac{1}{8}|M| = \frac{7}{8} \cdot 2^n$ Programme der Länge höchstens $n - 4$.
Jedes Programm ist ein Binärstring.
Es gibt aber nur $\sum_{k=0}^{n-4} 2^k = 2^{n-3} - 1 < \frac{1}{8}2^n$ Binärstrings der Länge höchstens $n - 4$.
Widerspruch. □

1. Zwischenklausur 2014, Aufgabe 4 (b)

Sei $C : \{0,1\}^* \to \{0,1\}^*$ ein verlustfreies Komprimierungsverfahren. Beweisen Sie, dass für mindestens die Hälfte der Wörter $w \in \{0,1\}^l$ gilt, dass $|C(w)| \geq l - 1$.

Beweis

C ist verlustfrei, also invertierbar, also injektiv. Die 2^l Wörter w aus $\{0,1\}^l$ werden also auf 2^l verschiedene Wörter $C(w)$ aus $\{0,1\}^*$ abgebildet. Es gibt nur $\sum_{k=0}^{l-2} 2^k = 2^{l-1} - 1$ Binärstrings kürzer $l-1$. Für $2^l - (2^{l-1} - 1) = 2^{l-1} + 1$ der Wörter w in $\{0,1\}^l$ gilt also $|C(w)| \geq l - 1$. Das ist eines mehr als die Hälfte. □

Endklausur 2016, Aufgabe 4 (b)

Sei $f : \{0,1\}^* \to \{0,1\}^*$ eine beliebige injektive Funktion zur Codierung von Binärwörtern. Beweisen Sie, dass für mindestens die Hälfte der Wörter $w \in \{0,1\}^{\leq n}$ gilt, dass $|f(w)| \geq |w|$.

Beweis

Die $\sum_{k=0}^{n} = 2^{n+1} - 1$ Wörter w aus $\{0,1\}^{\leq n}$ werden von dem injektiven f auf $2^{n+1} - 1$ verschiedene Wörter $f(w)$ aus $\{0,1\}^*$ abgebildet. Höchstens die Hälfte dieser $f(w)$ ist kürzer als n, weil es nur $\sum_{k=0}^{n-1} 2^k = 2^n - 1$ Binärstrings kürzer n gibt. Also hat mindestens die Hälfte eine Länge von mindestens $n \geq |w|$. □

1. Zwischenklausur 2015, Aufgabe 3

Zeigen Sie, dass mindestens die Hälfte aller Wörter in $\{0,1\}^{\leq n}$ zufällig ist.

Beweis

Widerspruchsannahme:
Seien weniger als die Hälfte der Wörter in $\{0,1\}^{\leq n}$ zufällig.
Dann ist mehr als die Hälfte der Wörter in $\{0,1\}^{\leq n}$ nicht zufällig.
Dann gibt es also mindestens $\frac{1}{2} \sum_{k=0}^{n} 2^k = \frac{2^{n+1}-1}{2}$ Wörter mit $K(w) < |w|$.
Also gibt es $\frac{2^{n+1}-1}{2}$ Programme kürzer als $|w| \leq n$, die diese Wörter generieren.
Jedes Programm (betrachtet im Maschinencode) ist ein Binärstring.
Also gibt es $\frac{2^{n+1}-1}{2}$ Binärstrings kürzer als n.
Tatsächlich gibt es davon aber nur $\sum_{k=0}^{n-1} 2^k = 2^n - 1 < \frac{2^{n+1}-1}{2}$. Widerspruch. □

Endklausur 2013, Aufgabe 2

Zeigen Sie, dass für alle $i, n \in \mathbb{N} - \{0\}$ mit $i \leq n$ mindestens $2^n - 2^{n-i}$ unterschiedliche Wörter $x \in (\Sigma_{\text{bool}})^n$ existieren, so dass $K(x) \geq n - i$.

Beweis

Widerspruchsannahme:
Es existieren $i, n \in \mathbb{N} - \{0\}$ mit $i \leq n$ so, dass in Σ^n_{bool} weniger als $2^n - 2^{n-i}$ Wörter x mit $K(x) \geq n - i$ existieren.
Weil $\Sigma_{\text{bool}})^n$ genau 2^n Wörter enthält, existieren dann also mehr als 2^{n-i} Wörter x mit $K(x) < n - i$.
Also existieren mehr als 2^{n-i} Programme kürzer als $n - i$, die diese Wörter generieren.
Jedes Programm ist (als Maschinencode) ein Binärstring.
Aber es gibt nur $\sum_{k=0}^{n-i-1} 2^k = 2^{n-i} - 1$ Binärstrings kürzer als $n - i$. □

Endklausur 2014, Aufgabe 3

Bezeichne p_n die n-te Primzahl. Beweisen Sie

$$\exists n_0 \in \mathbb{N}\colon\ \forall n \geq n_0\colon \quad K(p_n) < \log_2(p_n) - 2.$$

Hinweis: Aus dem Primzahlsatz folgt $p_n \in \Theta(n \cdot \ln n)$.

Beweis

Man kann ein Programm E schreiben, dass ein beliebiges $n \in \mathbb{N}$ als Eingabe liest und p_n ausgibt.
(E betrachtet jede natürliche Zahl in aufsteigender Reihenfolge, testet jeweils, ob es eine Primzahl ist, und zählt mit, wie viele Primzahlen bereits gefunden wurden. Die n-te gefundene Primzahl wird dann ausgegeben.)
Das Programm

$$P_n: \qquad \text{write}(E(n))$$

generiert dann $\text{Bin}(p_n)$ und hat Länge $|\text{Bin}(n)| + c = \lfloor \log n \rfloor + 1 + c$ für eine Konstante c, da in P_n alles bis auf das eine n konstant ist. Nach Definition der Kolmogorovkomplexität gilt also:

$$\exists c \in \mathbb{N}\colon\ \forall n \in \mathbb{N} \setminus \{0\}\colon \qquad K(p_n) = K(\text{Bin}(p_n)) \leq \lfloor \log n \rfloor + 1 + c \in \mathcal{O}(\log n)$$

Die Aussage $p_n \in \Theta(n \cdot \ln n)$ impliziert $p_n \in \Omega(n \cdot \ln n)$, und dies wiederum $n \in O(p_n / \ln n) \subseteq o(p_n)$. Somit gilt $K(p_n) \in o(\log p_n)$, woraus die Aussage folgt.
Etwas expliziter:
$p_n \in \Omega(n \cdot \ln n)$ bedeutet
$\exists C \in \mathbb{N}\colon\ \forall n \in \mathbb{N}\colon \quad C \cdot p_n \geq n \ln n.$
Umformen zu $n \leq C \cdot p_n / \ln n$ und einsetzen ergibt

$$\exists c, c', C \in \mathbb{N}\colon\ \forall n \in \mathbb{N}\colon \quad K(p_n) \leq \lfloor \log(C \cdot p_n / \log n) \rfloor + 1 + c \leq \log(C) + \log(p_n) - \log(\ln n) + c'.$$

Für beliebige Konstanten c' und C gilt

$$\exists n_0 \in \mathbb{N}\colon\ \forall n > n_0\colon \quad \log(C) - \log(\ln n) + c' < -2.$$

Damit folgt die verlangte Aussage.

1. Zwischenklausur 2018, Aufgabe 4

Auch 1. Zwischenklausur 2013, Aufgabe 3, und 1. Zwischenklausur 2011, Aufgabe 3. Entspricht Lemma 2.6 im Buch.

Sei $0 < n_1 < n_2 < n_3 < \dots$ eine streng monoton steigende Folge positiver natürlicher Zahlen mit

$$\frac{\lceil \log_2 n_i \rceil}{2} \leq K(n_i).$$

Sei für $i \in \mathbb{N} \setminus \{0\}$ die Zahl q_i die grösste Primzahl, die n_i teilt.
Zeigen Sie, dass die Menge $Q = \{q_i \mid i \in \mathbb{N} \setminus \{0\}\}$ unendlich ist.

Beweis

Widerspruchsannahme: Sei Q endlich. Dann ist $\max(Q)$ der grösste Primfaktor über alle Zahlen n_i wohldefiniert. Sei $\max(Q) = p_m$ die m-te Primzahl. Dann lässt sich jede Zahl n_i mit Hilfe der eindeutigen Primfaktorzerlegung als $n_i = \prod_{k=1}^{m} p_i^{r_{i,k}}$ mit eindeutigen $r_{i,k} \in \mathbb{N}$ darstellen. Diese Tatsache nutzen wir aus, um eine unendliche Folge von Programmen P_i anzugeben, die je n_i ausgeben – also $\text{Bin}(n_i)$ generieren – und Länge maximal $2m \lceil \log_2 (\log_2 n_i + 1) \rceil \in \mathcal{O}(\log_2 \log_2 n)$ haben, was eine entsprechende obere Schranke für die Kolmogorov-Komplexität $K(n_i)$ und damit einen Widerspruch zur obigen Ungleichung liefert.

Das simple Programm

$$P_i : \qquad \text{write}(p_1 \hat{} r_{i,1} * p_2 \hat{} r_{i,2} * \dots * p_m \hat{} r_{i,m})$$

generiert also $\text{Bin}(n_i)$, wir haben also $K(n_i) \leq |P_i|$.
Nun ist die Länge der Programme $|P_i|$ von oben zu beschränken.
Variabel sind nur $r_{i,1}$ bis und mit $r_{i,m}$. (Wohingegen die Menge Q und damit m zusammen mit der Folge $n_1, n_2, \dots$ fest vorgegeben sind.)
Ihre Binärdarstellungen beanspruchen

$$(\lfloor \log_2 r_{i,1} \rfloor + 1) + \dots + (\lfloor \log_2 r_{i,m} \rfloor + 1) \leq m(\lfloor \log_2 \log_2 n_i \rfloor + 1)$$

Bits.
Die Ungleichung folgt aus $r_{i,k} = \log_{p_k} p_k^{r_{i,k}} \leq \log_{p_k} n_i \leq \log_2 n_i$, was wegen $p_k \geq 2$ gilt.

Direkt in den Binärcode eines Programmes geschrieben nehmen diese m Zahlen $r_{i,k}$ insgesamt also höchstens die Länge $m \log_2 \log_2 n_i + m$ ein.
Bei genauer Betrachtung zeigt sicher aber, dass mehr Bits notwendig ist. Denn die Zahlen $r_{i,k}$ (und damit die Länge ihrer Binärdarstellungen) können beliebig gross werden, beim Ausführen des Programmcodes muss aber trotzdem immer klar sein, wo die Binärdarstellung einer Zahl beginnt und wo sie zu Ende geht. Irgendwie muss man also in die

Darstellung einer Zahl auch eincodieren, wie lange diese Darstellung ist. Eine Möglichkeit dafür, die Zahlen $r_{i,k}$ selbstbegrenzend abzuspeichern – also so, dass die Grenzen der Darstellung jeder Zahl im Gesamtcode aus der Darstellung selbst hervorgeht –, ist folgende: Wir fügen vor allen Bits, die zu der Binärdarstellung einer solchen Zahl gehören, eine Eins hinzu, und vor allen anderen eine Null. Dann ist jeder Block von Bitpaaren, die mit einer Eins beginnen, als einer Zahl erkennbar. Die richtige Decodierung des Programmcodes implementieren wir in einem konstanten, einleitenden Programmteil. Dadurch wird die Länge, welche die Codierung der Zahlen beansprucht, verdoppelt.
Wir verdoppeln die Bitzahl für eine einfache selbstbeschränkende Codierung.
Somit gilt:

$$\exists c, m \in \mathbb{N}\colon K(n_i) \leq |P_i| \leq 2m(\lfloor \log_2 \log_2 n_i \rfloor + 1) + c.$$

Die n_i werden beliebig gross, da die Folge streng monoton steigend ist. Die hergeleitete obere Schranken widerspricht aber der vorgegebenen unteren Schranke an $K(n_i)$ für ausreichend grosses n_i.

□

1. Zwischenklausur 2016, Aufgabe 4

Aufgabenstellung (in etwas verdichteter Form)

Sei $1 < n_1 < n_2 < n_3 < \ldots$ eine (streng monoton steigende) Folge natürlicher Zahlen.
Sei $P = \{p \in \text{PRIM} \mid \exists i \in \mathbb{N} - \{0\}\colon p | n_i\}$ die Menge aller Primfaktoren der Zahlen n_i.

a) Es gelte

$$\exists \varepsilon > 0\colon \forall i \in \mathbb{N} - \{0\}\colon (1+\varepsilon) \log_2 \log_2 n_i \leq K(n_i). \qquad (*)$$

Beweise $|P| > 1$.

b) Es gelte

$$\exists f \in \omega(1)\colon \forall i \in \mathbb{N} - \{0\}\colon f(n_i) \log_2 \log_2 n_i \leq K(n_i). \qquad (**)$$

Beweise $|P| = \infty$.

Beweis auf der Rückseite. Bitte zuerst selbst möglichst vollständig zu lösen versuchen. Vergleiche auch die originale Formulierung der Aufgabenstellung.

Beweis

a) Widerspruchsannahme: Sei $|P| \leq 1$.
Weil $n_1, n_2, \ldots$ streng monoton steigt, gilt $|P| = 1$.
Sei $p \in P$ der einzige Primfaktor aller Zahlen n_i.
Sei $r_i \coloneqq \log_p n_i$, also $p^{r_i} = n_i$.
Das Programm P_i:

$$\text{write}(p \hat{} r_i)$$

generiert (mit impliziter Konvertierung) $\text{Bin}(n_i)$.
Nur r_i ist variabel, somit gilt

$$\exists c \in \mathbb{N} \colon \forall i \in \mathbb{N} - \{0\} \colon K(n_i) \leq |P_i| = c + \lceil \log(r_i + 1) \rceil \leq \tilde{c} + \log_2 \log_2 n_i$$

Widerspruch zu $(*)$ für genügend grosses n_i.
(Ein solches n_i existiert, da $n_1, n_2, \ldots$ streng monoton steigt.) □

b) Widerspruchsannahme: Sei $|P| < \infty$.
Sei $m \coloneqq |P|$.
Seien $p_1, \ldots, p_m$ die m Primzahlen in P.
Jedes n_i lässt sich in diese Primfaktoren zerlegen: $n_i = p_1^{r_{1,i}} \cdots p_m^{r_{m,i}}$.
Das Programm P_i:

$$\text{write}(p_1 \hat{} r_{1,i} * \cdots p_m \hat{} r_{m,i})$$

generiert (mit impliziter Konvertierung) $\text{Bin}(n_i)$.
Nur die $r_{1,i}, \ldots, r_{m,i}$ sind variabel.
Ein $r_{k,i}$ benötigt in einfacher selbstbeschränkender Codierung $2|\,\text{Bin}(r_{k,i})|$ Bits.
(Abwechselnd ein Bit aus $\text{Bin}(r_{k,i})$ und eines, das angibt, ob $\text{Bin}(r_{k,i})$ damit endet.)[1]
Die benötigte Bitzahl für alle $r_{1,i}, \ldots, r_{m,k}$ zusammen ist dann

$$\sum_{k=1}^{m} 2(1 + \lfloor \log_2 r_{k,i} \rfloor) \leq 2m + 2m \log_2 \log_2 n_i,$$

wobei wir $r_{k,i} = \log_{p_k} p_k^{r_{k,i}} \leq \log_{p_k} n_i \leq \log_2 n_i$ verwendet haben.
Somit gilt:

$$\exists c \in \mathbb{N} \colon \forall i \in \mathbb{N} - \{0\} \colon \quad K(n_i) \leq |P_i| \leq c + 2m + 2m \log_2 \log_2 n_i.$$

Widerspruch zu $(**)$ für genügend grosses i.
(Denn c und m sind Konstanten, aber $f(n_i) \xrightarrow{n_i \to \infty} \infty$.
Und $n_i \xrightarrow{i \to \infty} \infty$, da $n_1, n_2, \ldots$ streng monoton steigend ist.) □

Nebenbemerkung: Die Aussage der ersten Teilaufgabe folgt auch aus dem Beweis der zweiten Teilaufgabe. Sie folgt aber nicht direkt aus der Aussage der zweiten Teilaufgabe.

[1] Man kommt auch ohne den Faktor aus, wenn man alle $\text{Bin}(r_{k,i})$ mit führenden Null füllt. Das ergibt dann einmal ein Maximum über k, das dann aber ohnehin mit n_i abgeschätzt wird. Es wird dann konstanter Teild es Programmes von vorne und hinten angegeben, der mittlere Teil gleichmässig in m Teile unterteilt.

1. Zwischenklausur 2017, Aufgabe 3

Aufgabenstellung (in etwas verdichteter Form)

a) Beweise $\exists c \in \mathbb{N}\colon \forall i \in \mathbb{N} - \{0\}\colon \quad K(x_i) \leq \lceil \log_2 \log_2 x_i \rceil + c$, wobei $x_i = 2^i \cdot 3^i \cdot 5^i \in \mathbb{N}$.

b) Beweise, dass es eine unendliche Folge $n_1, n_2, \ldots$ gibt, so dass

1. $\forall i \in \mathbb{N}\colon\ n_i < n_{i+1}$,
2. $|\{p_k \mid \exists i p_k | n_i\}| = \infty$, wobei p_k die k-te Primzahl bezeichnet,
   (n Worten also: Die Anzahl der Primfaktoren der n_i ist unendlich.) und
3. $\exists c \in \mathbb{N}\colon\ K(n_i) \leq c + \lceil \log_2 \log_2 n_i \rceil$.

Beweis

a) Das Programm

$$\begin{aligned} P_i\colon \quad & v \coloneqq i \\ & \text{write}(2^v * 3^v * 5^v) \end{aligned}$$

generiert $\text{Bin}(2^i 3^i 5^i)$. Eine Zahl zu generieren, bedeutet ihre Binärdarstellung auszugegen. Die Konvertierung der Zahl $2^v * 3^v * 5^v$ in ihre Binärdarstellung $\text{Bin}(2^v * 3^v * 5^v)$ nimmt die Funktion write implizit vor. In P_i ist nur das i variabel. Nach Definition der Kolmogorovkomplexität gilt also

$$\exists c \in \mathbb{N}\colon \forall i \in \mathbb{N} - \{0\}\colon \quad K(2^i 3^i 5^i) \leq |P_i| = \underbrace{\lceil \log_2(i+1) \rceil}_{\substack{|\text{Bin}(i)| \\ \text{Nur } i \text{ variabel in } P_i}} + c \leq \lceil \log_2 \log_2 2^i 3^i 5^i \rceil + 1 + c.$$

Die letzte Ungleichung gilt, da $\log_2(2^i 3^i 5^i) = \log_2(30^i) \geq \log_{30}(30^i) = i$. □

b) Wähle $n_i \coloneqq p_i^i$. Die ersten beiden Bedingungen sind offensichtlich erfüllt. Wir beweisen die dritte. Das Programm

$$\begin{aligned} P_i\colon \quad & v \coloneqq i \\ & \text{write}(E(v)\hat{}v) \end{aligned}$$

generiert $\text{Bin}(p_i^i)$. (Dabei ist E eine Unterprogramm, dass die i-te Primzahl berechnet, ein solches können wir beispielsweise mit Hilfe des Siebs des Eratosthenes berechnen.) In P_i ist nur das i variabel. Somit gilt aufgrund der Definition der Kolmogorovkomplexität:

$$\exists c \in \mathbb{N}\colon \forall i \in \mathbb{N} - \{0\}\colon \quad K(p_i^i) \leq |P_i| = \underbrace{\lceil \log_2(i+1) \rceil}_{\substack{|\text{Bin}(i)| \\ \text{Nur } i \text{ variabel in } P_i}} + c \leq \lceil \log_2 \log_2 p_i^i \rceil + 1 + c.$$

Die letzte Ungleichung folgt aus $\log_2 p_i^i \geq \log_{p_i} p_i^i = i$. □

Erklärung der Hauptargumentationslinie von Satz 2.4 (schwache Version des Primzahlsatzes) und Erweiterungen mithilfe von verbesserten selbstbeschränkenden Codierungen

Aussage des schwachen Primzahlsatzes
Es gibt eine Konstante $c \in \mathbb{N}$, sodass für unendliche viele $k \in \mathbb{N}$ gilt

$$\frac{k}{2^c \cdot \log k \cdot (\log\log k)^2} \leq \mathrm{Prim}(k)\,.$$

Kurzzusammenfassung des Beweises
Jedes $n \in \mathbb{N}$ hat einen grössten Primzahlfaktor, sei es die m-te Primzahl p_m. Wir schreiben für jedes n ein Programm, das n als Produkt von p_m und n/p_m berechnet und $\mathrm{Bin}(n)$ ausgibt. Dabei berechnen wir das erste p_m aus dem Index m; wir sparen so in der Darstellung $\log p_m$ Bits und verbrauchen höchstens $\log m + \log\log m + 2\log\log\log m + c$ zusätzliche Bits für eine Konstante $c \in \mathbb{N}$.
(Der erste Summand ist die Anzahl Bits zur Codierung von m als Bin(m), der zweite die Anzahl Bits zur Codierung der Länge der ersten Codierung und der dritte schliesslich die Anzahl Bits zur Codierung der Länge der zweiten Codierung, wobei der Faktor 2 eine selbstbeschränkende Codierung erlaubt.)
Falls n zufällig (d. h. unkomprimierbar) ist, darf man dadurch nichts sparen, es gilt also

$$\begin{aligned}
& & \log p_m &\leq \log m + \log\log m + 2\log\log\log m + c \\
&\xRightarrow{\exp(\cdot)} & \frac{p_m}{2^c \cdot \log m \cdot (\log\log m)^2} &\leq m \overset{\substack{\text{Def. v. } p_m \\ \text{u. Prim}(\cdot)}}{=\!=\!=} \mathrm{Prim}(p_m) \\
&\xRightarrow{m \leq p_m} & \frac{p_m}{2^c \cdot \log p_m \cdot (\log\log p_m)^2} &\leq \mathrm{Prim}(p_m)
\end{aligned}$$

Wir wissen, dass es unendlich viele zufällige Zahlen gibt. (Dies folgt aus Lemma 2.5 oder einem direkten Abzählargument.) Nach Lemma 2.6 haben unendlich viele zufällige Zahlen n auch unendlich vielee grössten Primzahlfaktoren p_m. Die letzte Ungleichung oben gilt also für unendliche viele $k \coloneqq p_m$.

Ausführlicher Beweis mit Programmangabe
Jede Zahl $n \in \mathbb{N} \setminus \{0, 1\}$ hat einen grössten Primzahlfaktor, sei dieser die m-te Primzahl p_m. Das m hängt also von n ab, $m = m(n)$. Es gibt ein Programm E, das aus dem Index m die Primzahl p_m berechnet – beispielsweise mittels einer einfachen While-Schleife, in der die natürlichen Zahle der Reihe nach auf Primalität getestet werden (beispielsweise mithilfe des Siebs des Eratosthenes), bis man die m-te Primzahl gefunden hat. Es gilt dann also $E(m) = p_m$. Folgendes Programm gibt somit n aus:

$$\begin{aligned}
P_n : \quad & \mathrm{v} \coloneqq m \\
& \mathrm{q} \coloneqq \frac{n}{p_m} \\
& \mathrm{write(q*E(v))}
\end{aligned}$$

Damit ist $K(n) \leq |P_n|$ bewiesen. Wir wollen nun $|P_n|$ möglichst klein halten. Dazu müssen wir über die Codierung der variablen Teile von von P_n nachdenken. Im Programm hängen nur m und $\frac{n}{p_m}$ von n ab, der Rest hat eine konstante Länge c. Kommt in einem Programm nur an einer Stelle eine variable Zahl vor, können wir diese direkt in Binärdarstellung eincodieren. Beim Lesen des Maschinencodes können die beiden konstanten Teile vor und hinter der Zahl einfach bestimmt werden, indem man vom Anfang bzw. vom Ende her den konstanten Teil abzählt. Alles Verbleibende ist dann die Binärdarstellung der variablen Zahl.
Sind in dem Programm hingegen zwei (oder mehr) variable Zahlen unterzubringen, ist dies Strategie nicht mehr möglich. Wir müssen daher mindestens eine der beiden variablen Zahlen (bzw. alle bis auf eine) selbstbeschränkend codieren. Wir tun dies für m. Für die andere Zahl $\frac{n}{p_m}$ kann dann die obige Methode verwendet werden, um weiterhin nur die reine Binärdarstellung in das Programm zu P_n schreiben. Damit haben wir

$$\exists c \in \mathbb{N}: \quad K(n) \leq |P_n| \approx c + \log_2 \frac{n}{p_m} + f(m) = c + \log_2 n - \log_2 p_m + f(m)\,,$$

wobei $f(m)$ angibt, wie viel Platz die selbstbeschränkende Codierung von m einnimmt. Wir werden später eine Codierung festlegen und dabei versuchen, f möglichst klein halten.
Wir haben also

$$\exists c \in \mathbb{N}\colon K(n) \leq c + \log_2 n - \log_2 p_m + f(m)\,.$$

Wenn wir zusätzlich noch annehmen, dass n zufällig ist – also $K(n) \geq \lceil \log_2(n+1) \rceil - 1 = \lfloor \log_2 n \rfloor$ gilt –, haben wir

$$\begin{aligned} \exists c \in \mathbb{N}: \quad & \log_2 n \leq c + \log_2 n - \log_2 p_m + f(m) \\ \implies \quad & \log(p_m) - c \leq f(m) \\ \xRightarrow{\exp(\cdot)} \quad & \frac{p_m}{2^c} \leq 2^{f(m)} \end{aligned}$$

Die Annahme, dass n zufällig ist, gilt für unendlich viele Zahlen, was sich mit dem gewöhnlichen Abzählargument einfach beweisen lässt. Lemma 2.6 gibt uns eine noch stärkere Aussage: Es gibt unendlich viele zufällige Zahlen, die zudem unendlich viele verschiedene grösste Primzahlfaktoren p_m haben.
Diese stärkere Aussage benötigen wir hier, damit die Ungleichungen, die wir nur für die zufälligen n herleiten können, auch tatsächlich unendlich viele Primzahlen p_m abdecken.

Einfache selbstbeschränkende Codierung
Sei

$$\mathrm{Bin}(m) = a_1 a_2 \dots a_{1+\lfloor \log_2 m \rfloor}$$

die Binärdarstellung von m. Ihre Länge ist $|\,\mathrm{Bin}(m)| \approx \log m$. Eine einfache selbstbeschränkende Codierung ist:

$$\mathrm{Bin}_{\mathrm{sb}}(m) = a_1 0 a_2 0 \dots 0 a_{1+\lfloor \log_2 m \rfloor} 1$$

(Auf den geraden Positionen besagt eine Null, dass die Codierung der Zahl weitergeht, eine 1 besagt, dass man das Ende erreicht hat.) Ihre Länge ist $|\operatorname{Bin}_{\text{sb}}(m)| \approx 2\log_2 m$.

Resultat mit einfachster selbstbeschränkenden Codierung
Wir benutzen vorerst diese beschriebene selbstbeschränkende Codierung für m. Wir setzen also $f(m) = |\operatorname{Bin}_{\text{sb}}(m)| = 2|\operatorname{Bin}(m)| = 2\lfloor\log(m)\rfloor + 2$ in die bereits für unendlich viele m bewiesene Ungeleichung $\frac{p_m}{2^c} \leq 2^{f(m)}$ ein und erhalten $\frac{p_m}{2^c} \leq 2^2 m^2$. Umgeschrieben mit der Konstanten $c' = (c+2)/2$ ist dies

$$\frac{\sqrt{p_m}}{2^{c'}} \leq m = \operatorname{Prim}(p_m)\,,$$

Die letzte Gleichung gilt einfach nach der Definition von p_m als die m-te Primzahl und Prim als die Anzahl Primzahlen bis und mit p_m.
Dies stellt eine erheblich schwächere Version von Satz 2.4, wo die untere Schranke anstelle von $\sqrt{p_m}$ das viel bessere $\frac{p_m}{\log(p_m)\cdot(\log\log p_m)^2}$ stehen hat. Wir leiten jetzt diese verbesserte Schranke her.

Verbesserung der selbstbeschränkenden Codierung und somit der Schranke
Wir verbessern unsere Schranke, indem wir eine effizientere selbstbeschränkende Codierung für m verwenden und damit eine kleinere Funktion $f(m)$ erhalten. Die Länge der Binärdarstellung von m ist $\lfloor\log_2 m\rfloor + 1$. Ist diese bekannt, benötigen wir keine selbstbeschränkende Codierung mehr. Die Länge der Binärdarstellung hat wiederum eine Binärdarstellung der Länge $\lfloor\log_2(\lfloor m\rfloor + 1)\rfloor + 1 \approx \log_2\log_2 m$. Diese können wir wiederum mit obigem Trick (also mit zusätzlichen Bits auf den geraden Positionen, die nur dazu da sind, die Länge anzugeben) selbstbeschränkend codieren, was einen Faktor 2 kostet. Insgesamt erhält man so $f(m) \approx \log_2 m + 2\log\log m + c$ für eine Konstante c. Einsetzen in $\frac{p_m}{2^c} \leq 2^{f(m)}$ ergibt jetzt die verbesserte Schranke

$$\frac{p_m}{2^{c'}(\log_2 m)^2} \leq \operatorname{Prim}(p_m)$$

für eine Konstante c'.

Weitere Verbesserung durch Iteration
In der ersten Verbesserung oben haben wir die simple selbstbeschränkenden Codierung, welche einen Faktor 2 verursacht, statt zur Angabe von m selbst nur zur Angabe der Länge von m verwendet. Insgesamt erhält man so eine verbesserte selbstbeschränkende Codierung von m. Diesen Trick kann man jetzt nochmals verwenden, sodass wir sogar nur noch auf die Länge der Länge von m die simple selbstbeschränkenden Codierung angeben müssen. Das ergibt dann

$$f(m) = \log_2 m + \log_2\log_2 m + 2\log_2\log_2\log_2 m + c \text{ und damit}$$

$$\frac{p_m}{2^{c'}\log_2 m(\log_2\log_2 m)^2} \leq \operatorname{Prim}(p_m)\,,$$

was die Schranke von Satz 2.4 ist. Eine weitere Iteration des Tricks ergäbe die Schranke

$$\frac{p_m}{2^{c'} \cdot \log_2 m \cdot \log_2 \log_2 m \cdot (\log_2 \log_2 \log_2 m)^2} \leq \mathrm{Prim}(p_m).$$

und weiteres Iterieren verbessert die Schranke immer weiter.

Alternative Verbesserungsmöglichkeit
Eine alternative – wenn auch asymptotisch viel schlechtere – Verbesserung erhält man indem man die simple selbstbeschränkende Codierung, die wir immer am Ende angewendet haben, ein wenig verbessert. Anstatt die Anzahl Bits zu verdoppeln und mit den Bits auf den geraden Positionen anzugeben, wo die selbstbeschränkende Codierung endet, benutzt nur noch jedes $(\alpha + 1)$-te Bit dafür. Die Bitzahl im letzten Schritt wird dadurch nur noch grob ver-$(1 + \frac{1}{\alpha})$-facht; womit man in $f(m)$ den Faktor 2 durch $1 + \frac{1}{\alpha}$ ersetzen kann. Dieser Faktor zeigt sich in der Schranke am Ende dann als Exponent.
Bei Anwendung mit $\alpha = 4$ auf die Variante von Satz 2.4. erhält man

$$f(m) \approx \log_2 m + \log_2 \log_2 m + \frac{5}{4} \log_2 \log_2 \log_2 m \text{ und}$$

$$\frac{1}{2^c p_m \log_2 m (\log_2 \log_2 m)^{\frac{5}{4}}} \lesssim \mathrm{Prim}(p_m).$$

Methode des Buchs zur Vermeidung der unbestimmten Konstante
Das Buch vermeidet die unbestimmte Konstante c, die vom konstanten Teil der Programmes P_n stammt, indem nicht mit einem Programm argumentiert wird, sondern mit einer konkreten Codierung $\mathrm{Wort}(m, n/p_m)$ ohne umgebendes Programm. Diese Codierung entspricht genau dem Programm P_n, einfach ohne die konstanten Programmteile.
Durch diesen Ansatz verliert man aber zunächst die Verbindung zur Kolmogorovkomplexität und damit zur Zufälligkeit der n. Diese Zufälligkeit hat man aber zweimal verwendet: Die erste Anwendung war, dass man beim Ersetzen der Binärdarstellung von p_m durch m nichts einsparen kann, weil sich n das nicht kürzer darstellen lässt. Die zweite Anwendung war die entsprechende Voraussetzung in Lemma 2.6.
Das Buch umgeht die erste Anwendung durch ein direktes Abzählarguement mit der konkreten Codierung von n als $\mathrm{Wort}(n) \coloneqq \mathrm{Wort}(m, n/p_m) \in \{0,1\}^*$ und beweist für die zweite Anwendung doch wieder eine untere Schranke für $K(n)$:
Für höchstens $\sum_{k=0}^{i-2} 2^k = 2^{i-1} - 1$ verschiedene n kann $|\mathrm{Wort}(n)| \leq i - 2$ gelten. Völlig analog kann $K(n) \leq i - 2$ für höchstens $\sum_{k=0}^{i-2} 2^k = 2^{i-1} - 1$ verschiedene n gelten.
Im Intervall $I = [2^i, 2^{i+1})$ gibt es aber 2^i verschiedene n. Für mehr als die Hälfte der n in I gilt also $|\mathrm{Wort}(n)| \geq i - 1$. Und für mehr als die Hälfte gilt $K(n) \geq i - 1$. Für mindestens ein n gilt also beides gleichzeitig. Für jedes $n \in I$ gilt $\mathrm{Bin}(n) = i + 1$ und gleichzeitig ist $\mathrm{Bin}(n) = \lfloor \log n \rfloor + 1$, woraus $i = \lfloor \log n \rfloor$ folgt.
Es gibt also ein $n \in I$ mit $|\mathrm{Wort}(n)| \geq \log(n) - 1$ und $K(n) \geq \log(n) - 1$. Da die Intervalle $I = I_i$ für $i = 1, 2, \ldots$ alle disjunkt sind, hat man also unendlich viele Zahlen, die beide Eigenschaften erfüllen.

Alternativausdruck für Länge der Binärdarstellung einer natürlichen Zahl
Wir wissen, dass die Binärdarstellung $\mathrm{Bin}(n)$ einer Zahl n ungefähr die Länge $\log n$ hat. Genauer gilt gemäss Vorlesung und Buch:

$$|\mathrm{Bin}(n)| = \lceil \log(n+1) \rceil$$

Nebenbemerkung: Für $n = 0$ ist die Gleichung falsch, weil $\mathrm{Bin}(0)$ im Buch als 0 definiert ist, mit 0 als Symbol aus dem Alphabet $\{0, 1\}$ und nicht der Zahl 0. Somit gilt $|\mathrm{Bin}(0)| = 1 \neq 0 = \lceil \log(0+1) \rceil$, dieser Spezialfall ist im Buch also nicht richtig berücksichtigt. Korrekt wäre es hingegen mit der natürlichen Definition $\mathrm{Bin}(0) := \lambda$.)
Ein nützlicher alternativer Ausdruck für die Länge der Binärdarstellung ist

$$|\mathrm{Bin}(n)| = \lfloor \log n \rfloor + 1.$$

(Auch diese Gleichung ist für $n = 0$ nicht korrekt. Man kann diesen Problemfall bei Anwendungen durch eine Fallunterscheidung umgehen. Natürlich könnte man beispielsweise auch $|\mathrm{Bin}(n)| = \max\{1, \lfloor \log n \rfloor + 1\}$ definieren.) Mit der Alternativdarstellung $|\mathrm{Bin}(n)| = \lfloor \log n \rfloor + 1$ erhält man insbesondere sofort $|\mathrm{Bin}(n)| \leq 1 + \log n$ als bestmögliche obere Schranke ohne Rundungsoperator, was in der Anwendung sehr oft nützlich ist.
Die folgenden Tabelle «beweist» die Korrektheit der beiden Ausdrücke für $n \neq 0$:

$\boldsymbol{n}$	$\mathbf{Bin}\,\boldsymbol{n}$	$\lvert\mathbf{Bin}\,\boldsymbol{n}\rvert$	$\mathbf{log_2}\boldsymbol{n}$	$\lfloor\mathbf{log_2}\boldsymbol{n}\rfloor$	$\lfloor\mathbf{log_2}\boldsymbol{n}\rfloor + \mathbf{1}$	$\mathbf{log_2}(\boldsymbol{n}+\mathbf{1})$	$\lceil\mathbf{log_2}(\boldsymbol{n}+\mathbf{1})\rceil$
0	0	1	$-\infty$	$-\infty$	$-\infty$	0	0
1	1	1	0	0	1	1	1
2	10	2	1	1	2	1,…	2
3	11	2	1,…	1	2	2	2
4	100	3	2	2	3	2,…	3
5	101	3	2,…	2	3	2,…	3
6	110	3	2,…	2	3	2,…	3
7	111	3	2,…	2	3	3	3
8	1000	4	3	3	4	3,…	4
9	1001	4	3,…	3	4	3,…	4
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Aussage und Beweis von Lemma 3.3 (Direkte Methode, S. 68)

Aussage

Sei A ein endlicher Automat, der eine Sprache $L \subseteq \Sigma^*$ erkennt.
Seien $x, y \in \Sigma^*$ mit $\hat{\delta}(q_0, x) = \hat{\delta}(q_0, y)$.[1]
Dann gilt $\forall z \in \Sigma^*\colon \hat{\delta}(q_0, xz) = \hat{\delta}(q_0, yz)$ und insbesondere $xz \in L \iff yz \in L$.

Beweis

Diese Aussage ist sehr transparent. Wenn das Einlesen von x und das Einlesen von y in denselben Zustand q führen – formal also: wenn $\hat{\delta}(q_0, x) = \hat{\delta}(q_0, y)$ gilt –, dann ist irrelevant, welches Wort man gelesen hat, das anschliessende Lesen von z führt immer in denselben Zustand $\hat{\delta}(q, z)$ – formal also: $\hat{\delta}(q_0, xz) = \hat{\delta}(\hat{\delta}(q_0, x), z) = \hat{\delta}(\hat{\delta}(q_0, y), yz) = \hat{\delta}(q_0, yz)$. Insbesondere folgt $\forall z \in \Sigma^*\colon\ xy \in L \Leftrightarrow xz \in L$.

Zusatzbemerkung

Von den dreien in dieser Vorlesung präsentierten Beweismethoden (Lemma 3.3, Pumping-Lemma und Kolmogorovkomplexitätsmethode) ist dies die einzige, die garantiert für jede nichtreguläre Sprache funktioniert – natürlich nur, sofern man die richtigen Wörter wählt. Für die anderen beiden Methoden gibt es Beispiel nichtregulärer Sprachen, deren Nichtregularität man mit dem Pumping-Lemma bzw. mit der Kolmogorovkomplexitätsmethode nicht direkt nachweisen kann; sie sind aber nicht leicht zu finden und der Nachweis, dass die beiden Methoden nicht funktionieren, ist recht aufwendig.

[1] In Worten sagt diese Gleichung: Die beiden Wörter („Wörter“ ist hier der korrekte Plural, nicht „Worte“) x und y führen in denselben Zustand. Man könnte formal auch schreiben

$$\exists q \in Q : (q_0, x) \overset{*}{\vdash} (q, \lambda) \wedge Q\colon (q_0, y) \overset{*}{\vdash} (q, \lambda).$$

Oder noch kürzer:

$$\exists q \in Q : x, y \in \mathrm{Kl}[q].$$

Das ist aber insofern kompliziert, als man einen Quantor benötigt, um das q explizit anzugeben, dass sonst in der Gleichung $\hat{\delta}(q_0, x) = \hat{\delta}(q_0, y)$ versteckt ist. Daher ist die erstgenannte Form wohl die einfachste.

Aussage und Beweis von Lemma 3.4 (Pumping-Lemma, S. 70)

Behauptung
Sei $L \subseteq \Sigma^*$ regulär. Dann gilt:

$$\exists n_0 \in \mathbb{N}\colon\ \forall w \in \Sigma^{\geq n_0}\colon\ \exists y, x, z \in \Sigma^*\colon\ (\text{ı}) \wedge (\text{i}) \wedge (\text{ii}) \wedge (\text{iii}),$$

wobei

(ı) $w = yxz$,

(i) $|yx| \leq n_0$,

(ii) $|x| > 0$ und

(iii) entweder $\forall k \in \mathbb{N}\colon yx^kz \in L$ oder $\forall k \in \mathbb{N}\colon yx^kz \notin L$.

Beweis
Sei $L \subseteq \Sigma^*$ eine reguläre Sprache. Dann existiert ein endlicher Automat $A = (Q, \Sigma, \delta, q_0, F)$ mit $L(A) = L$.
Sei $n_0 := |Q|$. Sei $w \in \Sigma^*$ ein Wort mit $|w| \geq n_0$. Dann ist $w = w_1w_2 \dots w_{n_0}u$ mit $w_1, w_2, \dots, w_{n_0} \in \Sigma, u \in \Sigma^*$. Die ersten n_0 Berechnungsschritte auf w sind:

$$(q_0, w_1w_2 \dots w_{n_0}u) \vdash_A (q_1, w_2 \dots w_{n_0}u) \vdash_A \dots \vdash_A (q_{n_0-1}w_{n_0}u) \vdash_A (q_{n_0}u)$$

Nach dem Schubfachprinzip sind mindestens zwei der $n_0 + 1$ Zustände $q_0, q_1, \dots, q_{n_0}$ identisch, es finden sich also $0 \leq i < j \leq n_0$ mit $q_i = q_j$.
Daher gilt mit

$$w = \underbrace{w_1 \dots w_i}_{y}\underbrace{w_{i+1}w_{i+2} \dots w_j}_{x}\underbrace{w_{j+1} \dots w_{n_0}u}_{z}$$

und für beliebiges $\ell \in \mathbb{N}$:

$$(q_i, x^{\ell+1}z) \vdash_A^* (q_j, x^\ell z) = (q_i, x^\ell z),$$

und durch k-malige Anwendung für beliebiges $k \in \mathbb{N}$:

$$(q_0, yx^kz) \vdash_A^* (q_i, x^kz) \vdash_A^* (q_i, x^{k-1}z) \vdash_A^* \dots \vdash_A^* (q_i, x^0z) = (q_i, z).$$

Falls $\hat{\delta}(q_i, z) \in L$, gilt also $\{yx^kz \mid k \in \mathbb{N}\} \subseteq L$, andernfalls $\{yx^kz \mid k \in \mathbb{N}\} \cap L = \varnothing$.
Dies ist Eigenschaft (iii).
Eigenschaft (i) folgt aus $|yx| = |w_1 \dots w_j| = j \leq n_0$,
Eigenschaft (ii) aus $|x| = |w_{i+1}w_{i+2} \dots w_j| = j - (i + 1) + 1 = j - i > 0$ □

Übung (Bonusaufgabe in 1. Zwischenklausur 2016
Beweise das Lemma mit der modifizierter Bedingung (i^R) $|xz| \leq n_0$.
Beweise das Lemma mit der modifizierten Längenmessung $|w|_a$ statt $|w|$ für ein fixes für ein $a \in \Sigma$. (Diese Veränderung wirkt sich auch auf $\Sigma^{\geq n_0}$ aus.)

Aussage und Beweis von Satz 3.1 (KK-Methode, S. 73)

Behauptung

Sei $L \subseteq \{0,1\}^*$ regulär. Für jedes $x \in \{0,1\}$ definieren wir die Sprache $L_x := \{y \mid xy \in L\}$. (Diese Sprache enthält zu jedem Wort in $w \in L$, das mit dem Präfix x beginnt, den verbleibenden Rest y.) Es sei $y_{x,n}$ das in kanonischer Reihenfolge n-te Wort in L_x, also $L_x = \{y_{x,1}, y_{x,2}, y_{x,3}, \ldots\}$.
Die Aussage ist nun: $\exists c \in \mathbb{N}\colon\ \forall x \in \Sigma^*\colon\ \forall n \in \mathbb{N}\colon\ K(y_{x,n}) \leq \lceil \log_2(n+1) \rceil + c$.

Beweis

Seil L regulär. Es existiert also ein endlicher Automat $A = (Q, \{0,1\}, \delta, q_0, F)$ für L. Jedes Präfix x führt vom Startzustand q_0 in einen Zustand $q_x \in Q$. Von q_x her führt das Einlesen von y zum selben Zustand wie das Einlesen von xy aus dem eigentlichen Startzustand q_0.
Wir können das alternativ auch folgendermassen betrachten: Modifizieren wir den Automaten A zu einem Automaten A_x, in dem neu q_x der Startzustand ist, so wird y von A_x genau dann akzeptiert, wenn xy von A akzeptiert wird. Also gilt $L(A_x) = L_x$.
Wir betrachten folgendes Programm, welches durch alle Wörter in Σ^* in kanonischer Reihenfolge durchzugehen beginnt, dabei zählt, wie viele schon gefunden wurden, die in L_x liegen, und das n-te ausgibt, sobald es gefunden wurde.

$P_{x,n}$:
i ← 0
Für alle $y \in \{0,1\}^*$ in kanonischer Reihenfolge:
Simuliere y auf A mit Startzustand q_x, simuliere also $A_{q_x}(y)$.
Falls y akzeptiert wird: i ← i+1
Falls i = n: Gib y aus.

Das oben Beschriebene zeigt: Für jedes $x \in \Sigma^*$ und jedes $n \in \mathbb{N}$ generiert $P_{x,n}$ das Wort $y_{x,n}$. Nach Definition der Kolmogorovkomplexität gilt daher $K(y_{q_x,n}) \leq |P_{x,n}|$ und wir müssen nur noch $|P_{x,n}|$, die Länge des Programmes berechnen. Beachte, dass es sich bei $\{P_{x,n}\}_{x \in \Sigma^*, n \in \mathbb{N}}$ um eine zweidimensionale Familie von Programmen handelt, das x und das n sind keine Eingabe für ein allgemeines Programm, sondern fix in den Maschinencode des spezifischen $P_{x,n}$ integriert. Genauer gesagt codieren wir auch nicht das x selbst in den Maschinencode ein, sondern nur den Zustand q_x, den man mit x erreicht. Dies ist essentiell, denn es gibt nur endlich viele Zustände $q_x \in Q$, aber unendlich viele verschiedene $x \in \Sigma^*$. Wir können daher den Zustand q_x mit einer konstanten Zahl von $c_1 := \lceil \log_2(|Q|) \rceil$ Bits angeben, während wir die Eincodierung von x beliebig viel Platz in Anspruch nehmen würde. Der Rest des Programmes hat eine konstante Länge c_2. Sei $c := c_1 + c_2$. Die Gesamtlänge der Programme lässt sich also angeben als $|P_{q_x,n}| = |\mathrm{Bin(x)}| + c = \lceil \log_2(n+1) \rceil + c$. □

Übung

Beweise das Lemma mit der modifizierten Definition $L_x := \{y \mid yx \in L\}$.
Tipp: Anstatt bei der Simulation von y auf A einen anderen Startzustand als q_0 zu wählen, modifiziert man die Menge F der akzeptierenden Zustände.

Beweismuster und Anwendungsbeispiele

Wir nutzen die drei Methoden zunächst, um die Nichtregularität von $L = \{0^n 1^n \mid n \in \mathbb{N}\}$ möglichst konzis nachzuweisen. Hierbei sind Stellen grau hinterlegt, die je nach vorgegebener Sprache angepasst werden müssen. Danach folgen zwei weitere Nichtregularitätsbeweisaufgaben aus alten Prüfungen, die in etwas detaillierter in allen drei Methoden ausgeführt sind. In der zweiten dieser beiden Aufgaben ist die Sprache über dem unären Alphabet $\Sigma = \{0\}$ defniert. Dies stellt einen wichtigen Spezialfall dar und ist auch ein Beispiel dafür, dass es sich oft lohnt, das jeweils zweite Wort der Sprachen L_x zu betrachten anstelle des ersten. Allgemein gilt, dass die Methode der Kolmogorovkomplexität für gewöhnlich die kompakteste Lösung bieten kann, wenn man sie richtig anwendet.

Direkte Methode über den Automaten (Lemma 3.3)

Wid.-Ann.: L regulär.
Dann ex. EA $A = (Q, \Sigma, \delta, q_0, F)$ mit $L(A) = L$.
Betrachte die $|Q| + 1$ Wörter[2] 0^i für $k \in \{0, \ldots, |Q|\}$.
$\overset{\text{SFP}^3}{\Longrightarrow}$ Zwei führen in denselben Zustand, seien es 0^i und 0^j, $i < j$.
Wid. zu L. 3.3 für $z = 1^i$, da $xz = 0^i 1^i \in L$, aber $yz = 0^j 1^i \notin L$. □

Pumping-Lemma (Lemma 3.4)

Wid.-Ann. L regulär.
Sei n_0 die Konstante aus PL.
Wähle $w = 0^{n_0} 1^{n_0}$.
Sei $w = yxz$ beliebige Zerlegung.
(i) $\Rightarrow$ $x = 0^m$ [4]
(ii) $\Rightarrow$ $m > 0$
$yxz = w \in L$, aber $yx^2z = 0^{n_0+m} 1^{n_0} \notin L$ [5]
Wid. zu (iii). □

Kolmogorovkomplexitätsmethode (Satz 3.1)

Wid.-Ann.: L reg.
$\forall i \in \mathbb{N}$: Das 1. Wort in L_{0^i} ist 1^i.
$\overset{\text{L.3.3}}{\Longrightarrow} \exists c \in \mathbb{N}: \forall i \in \mathbb{N}: \quad K(1^i) \leq \lceil \log_2(1 + 1) \rceil + c = c + 1$
Unendlich viele Wörter mit KK höchstens $c + 1$,
aber endlich viele Programme der Länge höchstens $c + 1$. Wid.[6]

---

[2] Man könnte auch $|Q| + 2$ oder sogar eine unendliche Reihe von Wörtern betrachten, aber $|Q| + 1$ ist das Minimum, um das Schubfachprinzip anwenden zu können.
[3] Schubfachprinzip (engl. pigeonhole principle)
[4] Hier kann je nach Sprache eine Fallunterscheidung notwendig sein.
[5] Auch hier sind je nach Sprache zusätzliche Begründungen notwendig, manchmal mit Fallunterscheidungen für verschiedene Arten der Zerlegung. Man muss für *jeden* Fall einen Widerspruch herleiten.
[6] Man beachte, dass man in diesem Bweis nicht n anstelle von i als Variable verwenden sollte, da bereits in der Aussage von Satz 3.1 ein n vorkommt, nämlich um vom $n$ten Wort in L_x sprechen zu können. In der Anwendung wählen wir n fast immer fest als $n = 1$ oder aber fest als $n = 2$.

1. Zwischenklausur 2014, Aufgabe 2 (a)

Zeigen Sie, dass $L := \{www \mid w \in \{0,1\}^*\}$ nicht regulär ist.

Beweis direkt über den Automaten – Lemma 3.3

Widerspruchsannahme: L sei regulär.
Dann existiert ein endlicher Automat $A = (Q, \Sigma, \delta, q_0, F)$ mit $L(A) = L$.
Betrachte die $|Q| + 1$ Wörter 0^n1 mit $n \in M := \{0, 1, \ldots, |Q|\}$.
Dies sind mehr Wörter als Zustände, daher gibt es gemäss Schubfachprinzip
$i, j \in M$ mit $i < j$ und $\hat{\delta}(q_0, 0^i1) = \hat{\delta}(q_0, 0^j1)$.
Nach Lemma 3.3 gilt insbesondere für $z := 0^i10^i1$:

$$0^i10^i10^i1 \in L \Leftrightarrow 0^j10^i10^i1 \in L$$

Zwar ist $0^i10^i10^i1 \in L$, aber $0^j10^i10^i1 \notin L$, da $i < j$. *Widerspruch.* □

Beweis mittels Pumping-Lemma – Lemma 3.4

Widerspruchsannahme: L sei regulär.
Das Pumping-Lemma gibt uns ein $n_0 \in \mathbb{N}$. Betrachte $w := 0^{n_0}10^{n_0}10^{n_0}1 \in L$. Da offensichtlich $|w| \geq n_0$, gibt es eine Zerlegung $w = yxz$, für die die drei Eigenschaften des Pumping-Lemmas gelten.

(i) besagt $|yx| \leq n_0$, es folgt $|x| \leq n_0$ und $x \in \{0\}^*$.
(ii) besagt $|x| > 0$.
Somit gilt $yz = 0^{n_0-|x|}10^{n_0}10^{n_0}1 \notin L$, obwohl $w = yxz \in L$. *Widerspruch* zu (iii). □

Beweis mit Kolmogorov-Methode – Satz 3.1

Widerspruchsannahme: Sei L regulär.
Betrachte die Sprachen L_{0^i1}. Das erste Wort ist jeweils $v_i := 0^i10^i1$. Nach Satz 3.1 gibt es eine Konstante c, sodass für alle v_i gilt:

$$K(v_i) \leq \lceil \log_2(1+1) \rceil + c = 2 + c$$

Es gibt offensichtlich unendlich viele v_i,
aber nur endlich viele Binärstrings mit Länge maximal $2 + c$,
also nur endlich viele Programme mit Länge maximal $2 + c$, die je höchstens ein Wort erzeugen können,
also nur endlich viele Wörter w mit $K(w) \leq 2 + c$. *Widerspruch.* □

1. Zwischenklausur 2015, Aufgabe 4 (a)

Zeigen Sie, dass $L := \{0^{n \cdot \lceil \sqrt{n} \rceil} | n \in \mathbb{N}\}$ nicht regulär ist.

Allgemeine Vorbemerkungen zu Sprachen über einem Einsymbolalphabet

Es handelt sich bei $L \subseteq \Sigma^*$ um eine Sprache über einem unären Alphabet, als einem Alphabet mit nur einem einzigen Symbol: $|\Sigma| = 1$. Daher gilt offensichtlich: $\forall w \in \Sigma^*$: $w = 0^{|w|}$
Die Folge von Wörtern $w_n := 0^{n \cdot \lceil \sqrt{n} \rceil}$ ist somit kanonisch geordnet, da wir nur ein einziges Symbol haben und die Länge $n \cdot \lceil \sqrt{n} \rceil$ eine monoton steigende Funktion ist.
(Im Falle von anderen Funktionen als $f(n) = n \cdot \lceil \sqrt{n} \rceil$ gibt es eventuell einzelne störende n, die aber einfach durch eine untere Grenze $n > N \in \mathbb{N}$ von der Betrachtung ausgeschlossen werden können.)
Insbesondere folgt daraus, dass ein $w \in \Sigma^*$, dessen Länge zwischen jener von zwei direkt aufeinanderfolgenden Wörtern w_n und w_{n+1} aus L liegt, nicht in L liegen kann. Als Formel ausgedrückt:

$$\forall w \in \Sigma^* : \forall k \in \mathbb{N} : \quad |w_k| < |w| < |w_{k+1}| \quad \Rightarrow w \notin L$$

Der Nachweis der Nichtregularität beruht nun darauf, dass die Längendifferenz $|w_{n+1}| - |w_n|$ von einem Wort in L zum nächsten in L beliebig gross werden. Wir werden nun v_n als dasjenige Wort definieren, das man an w_n anhängen kann, um zu w_{n+1} erhalten.
Mit der Definition

$$v_n := 0^{|w_{n+1}| - |w_n|}$$

gilt

$$w_{n+1} = w_n v_n,$$

weil trivialerweise

$$|w_{n+1}| = |w_n| + \underbrace{|w_{n+1}| - |w_n|}_{|v_n|}.$$

Wegen der Rundungen gibt es leider keine nahe beieinander liegende oberen und unteren Schranken für $|v_n|$ zu finden.
Eine einfache untere Schranke ist die folgende:

$$\begin{aligned}
|v_n| &= |w_{n+1}| - |w_n| \\
&= (n+1) \cdot \left\lceil \sqrt{n+1} \right\rceil \quad - n \cdot \lceil \sqrt{n} \rceil \\
&\geq (n+1) \cdot \lceil \sqrt{n} \rceil \qquad - n \cdot \lceil \sqrt{n} \rceil \\
&= \lceil \sqrt{n} \rceil \\
&\geq \sqrt{n}.
\end{aligned}$$

Für Quadratzahlen ist diese Schranke viel zu schwach, wie wir jetzt gleich sehen werden. Für Quadratzahlen n^2 ist gilt:

$$\begin{aligned}|v_{n^2}| &= |w_{n^2+1}| - |w_{n^2}| \\ &= f(n^2+1) - f(n^2) \\ &= (n^2+1) \cdot \left\lceil \sqrt{n^2+1} \right\rceil \quad - n^2 \cdot \left\lceil \sqrt{n^2} \right\rceil \\ &= (n^2+1) \cdot (n+1) \qquad\qquad - n^2 \cdot n \\ &= n^2 + n + 1.\end{aligned}$$

Im Kontrast dazu gilt für Fast-Quadratzahlen $n^2 - 1 > 0$, dass

$$\begin{aligned}|v_{n^2-1}| &= |w_{n^2}| - |w_{n^2-1}| \\ &= (n^2) \cdot \left\lceil \sqrt{n^2} \right\rceil \qquad - n^2 - 1 \cdot \left\lceil \sqrt{n^2-1} \right\rceil \\ &= (n^2) \cdot n \qquad\qquad\qquad -(n^2-1) \cdot n \\ &= n.\end{aligned}$$

Beachte, dass solche Rechnungen bei der Verwendung von Lemma 3.3. oder des Pumping-Lemma notwendig sind, nicht aber bei der Methode der Kolmogorov-Komplexität, die daher bei solchen unären Sprachen zur Verwendung zu empfehlen ist.

Beweis mit Kolmogorov-Methode – Satz 3.1

Widerspruchsannahme: Sei L regulär. Seien $w_1, w_2, \ldots$ die Wörter von L in kanonischer Reihenfolge. Betrachte die Sprachen L_{w_n}. Das erste Wort ist jeweils λ, das zweite $v_n := 0^{|w_{n+1}|-|w_n|}$. Nach Satz 3.1 gibt es eine Konstante c, sodass für alle v_n gilt:

$$K(v_n) \leq \lceil \log_2(2+1) \rceil + c = 2 + c$$

Es gibt unendlich viele v_n, weil

$$\begin{aligned}|v_n| &= |w_{n+1}| - |w_n| \\ &= (n+1) \left\lceil \sqrt{n+1} \right\rceil \quad - n \left\lceil \sqrt{n} \right\rceil \\ &\geq (n+1) \left\lceil \sqrt{n} \right\rceil \qquad - n \left\lceil \sqrt{n} \right\rceil \\ &= \left\lceil \sqrt{n} \right\rceil \xrightarrow{n\to\infty} \infty\end{aligned}$$

Es gibt aber nur endlich viele Programme mit Länge maximal $2 + c$.
Ausführlichere Variante:
Es gibt aber nur endlich viele Binärstrings mit Länge maximal $2+c$ (nämlich $2^{2+c+1}-1$), die als Maschinencode für ein Programm in Frage kommen,
also nur endlich viele Programme mit Länge maximal $2 + c$,
die je höchstens ein Wort erzeugen können,
also nur endlich viele Wörter mit Kolmogorovkomplexität höchstens $2 + c$.
Widerspruch. □

Beweis mittels Pumping-Lemma – Lemma 3.4

Widerspruchsannahme: L sei regulär.
Seien $w_1, w_2, \ldots$ die Wörter von L in kanonischer Reihenfolge.
Das Pumping-Lemma gibt uns ein $n_0 \in \mathbb{N}$.
Betrachte $w_{n_0^2} := 0^{n_0^2 \lceil \sqrt{n_0^2} \rceil} \in L$.
(Begründung: Wir betrachten $w_{n_0^2}$, um unten mittels der Ungleichung $\sqrt{n} < |w_{n+1}| - |w_n|$, welche für Quadratzahlen n strikt ist, eine Längendifferenz grösser als n_0 zum nächsten Wort nach $w_{n_0^2}$ zu erhalten. Man könnte z. B. auch $w_{(n_0')^2}$ mit $n_0' := n_0 + 1$ betrachten, damit eine ungenauere Abschätzung ausreicht, sobald sich herausstellt, dass dies vorteilhaft ist.)
Da $|w_{n_0^2}| \geq n_0$, gibt es eine Zerlegung $w_{n_0^2} = yxz$, für die die drei Eigenschaften des Pumping-Lemmas gelten.

(i) besagt $|yx| \leq n_0$, es folgt $|x| \leq n_0$.
(ii) besagt $|x| > 0$.

Wegen $|yx^2z| = |yxz| + |x|$ gilt also $|yxz| < |yx^2z| \leq |yxz| + n_0$.
Das erste Wort in L nach $w_{n_0^2} = yxz$ ist $w_{n_0^2+1}$. Es gilt

$$\begin{aligned}
|w_{n_0^2+1}| - |w_{n_0^2}| &= \left(n_0^2+1\right) \cdot \lceil \sqrt{n_0^2+1} \rceil \quad - n_0^2 \cdot \lceil \sqrt{n_0^2} \rceil \\
&= \left(n_0^2+1\right) \cdot \lceil \sqrt{n_0^2+1} \rceil \quad - n_0^2 \cdot n_0 \\
&= \left(n_0^2+1\right) \cdot (n_0+1) \quad - n_0^2 \cdot n_0 \\
&> \left(n_0^2+1\right) \cdot n_0 \quad - n_0^2 \cdot n_0 \\
&= n_0
\end{aligned}$$

Die dritte Gleichung gilt, da $n_0 \in \mathbb{N}$ und $n_0 = \sqrt{n_0^2} < \sqrt{n_0^2+1} < \sqrt{(n_0+1)^2} = n_0 + 1$.

Somit ist $|yxz| + n_0 < |w_{n_0^2+1}|$, daher ist yx^2z länger als $w_{n_0^2}$, aber kürzer als das nächste Wort $w_{n_0^2}$. Somit gilt $yx^2z \notin L$, während $yxz \in L$, in Widerspruch zu (iii). □

Beweis direkt über den Automaten – Lemma 3.3

Definiere $w_n := 0^{n\lceil\sqrt{n}\rceil}$ und $v_n := 0^{|w_{n+1}|-|w_n|}$.
Widerspruchsannahme: L sei regulär.
Dann existiert ein endlicher Automat $A = (Q, \Sigma, \delta, q_0, F)$ mit $L(A) = L$.
Betrachte die $|Q| + 1$ Wörter w_n mit $n \in M := \{k^2 \mid k \in \{2, \dots, |Q| + 2\}\}$.
(M wählen wir als Menge von Quadratzahlen, damit wir unten für $i^2, j^2 \in M$ die Ungleichung $|v_{i^2}| < |v_{j^2}|$ erhalten können.)
Dies sind mehr Wörter als Zustände, daher gibt es gemäss Schubfachprinzip
$i^2, j^2 \in M$ mit $i < j$ und $\delta(q_0, w_{i^2}) = \delta(q_0, w_{j^2})$.
Nach Lemma 3.3 gilt insbesondere für $z := v_{i^2}$:

$$w_{i^2} v_{i^2} \in L \Leftrightarrow w_{j^2} v_{i^2} \in L$$

Zwar ist $w_{i^2} v_{i^2} = w_{i^2+1} \in L$, aber $w_{j^2} v_{i^2} \notin L$, denn $w_{j^2} v_{i^2}$ ist länger als w_{j^2} und kürzer als $w_{(j+1)^2}$, die in L direkt aufeinander folgen:

$$|w_{j^2}| < |w_{j^2} v_{i^2}| < |w_{j^2} v_{j^2}| = |w_{(j+1)^2}|$$

Für die erste Ungleichung verwenden wir $0 < |v_{i^2}|$, was bereits für das kleinstmögliche $i = 2^2$ gilt.
Die zweite Ungleichung folgt aus $|v_{j^2}| - |v_{i^2}| = j^2 + j + 1 - (i^2 + i + 1) > 0$. Dies verwendet, dass i^2 und j^2 Quadratzahlen sind. Die vollständige Rechnung ist

$$\begin{aligned}
|v_{n^2}| &= |w_{n^2+1}| - |w_{n^2}| \\
&= f(n^2 + 1) - f(n^2) \\
&= (n^2 + 1) \cdot \left\lceil \sqrt{n^2 + 1} \right\rceil \quad - n^2 \cdot \left\lceil \sqrt{n^2} \right\rceil \\
&= (n^2 + 1) \cdot (n + 1) \qquad\qquad - n^2 \cdot n \\
&= n^2 + n + 1.
\end{aligned}$$

Damit haben wir einen Widerspruch, L ist also nicht regulär. □

Nichtregularitätsaufgaben aus Prüfungen

1. $\{0^{n^2}1^n \mid n \in \mathbb{N}\}$
   (EK, HS06, A1b)
2. $\{0^{n^2} \mid n \in \mathbb{N}\}$
   (1. ZK, HS07, A3a)
3. $\{s1t \in \{0,1\}^* \mid |s| = |t|\}$
   (1. ZK, HS07, A3b)
4. $\{w10w^{\mathrm{R}} \mid w \in \{0,1\}^*\}$
   (1. ZK, HS08, A2a)
5. $\{0^n1^{\lceil\sqrt{n}\rceil} \mid n \in \mathbb{N}\}$
   (1. ZK, HS08, A2b)
6. $\{0^n10^{2n} \mid n \in \mathbb{N}\}$
   (EK, HS09, A2a)
7. $\{0^n1^n0^n \mid n \in \mathbb{N}\}$
   (1. ZK, HS10, A4a)
8. $\{u1uv \mid u, v \in \{0,1\}^*\}$
   (1. ZK, HS10, A4b)
9. $\{0^n \mid n$ ist eine Primzahl$\}$
   (EK, HS10, A2; 2. EK, HS11, A2b)
10. $\{u11u \mid u \in \{0,1\}^+\}$
    (1. ZK, HS11, A2a)
11. $\{1^i0^{i^2} \mid i \in \mathbb{N}\}$
    (1. ZK, HS11, A2a)
12. $\{0^{2i^2-5} \mid i \in \mathbb{N}\}$
    (1. EK, HS11, A2a)
13. $\{1^i0^j1^k \mid i, j, k \in \mathbb{N}$ und $j < j < k\}$
    (1. EK, HS11, A2b)
14. $\{w \in \{0,1\}^* \mid w = w^{\mathrm{R}}\}$
    (2. EK, HS11, A2a)
15. $\{0^k1^l0^m \mid k, l, m \in \mathbb{N}$ und $k + m \leq l\}$
    (1. ZK, HS12, A3a)
16. $\{0^{\lfloor\sqrt{i}\rfloor}1^i \mid i \in \mathbb{N}\}$
    (1. ZK, HS12, A3a)

17. $\{0^n1^m0^m1^n \mid n, m \in \mathbb{N}\}$
    (1. ZK, HS13, A2a)

18. $\{0^{n!} \mid n \in \mathbb{N}\}$
    (1. ZK, HS13, A2a; 1. ZK, HS17, A2b)

19. $\{0^n1^m \mid n, m \in \mathbb{N} \setminus \{0\} \text{ und } n \leq m \leq 2n\}$
    (EK, HS13, A3a)

20. $\{www \mid w \in \{0,1\}^*\}$
    (1. ZK, HS14, A2a)

21. $\{0^{n^{\lceil \log_2 n \rceil}} \mid n \in \mathbb{N}\}$
    (1. ZK, HS14, A2b)

22. $\{ww^{\mathrm{R}} \mid w \in \{0,1\}^*\}$
    (EK, HS14, A2a)

23. $\{0^{3n^2+5} \mid n \in \mathbb{N}\}$
    (EK, HS14, A2b)

24. $\{0^{n \cdot \lceil \sqrt{n} \rceil} \mid n \in \mathbb{N}\}$
    (1. ZK, HS15, A4a)

25. $\{w \in \{0,1\}^* \mid w = w^{\mathrm{R}} \text{ oder } |w|_0 \text{ ist Quadratzahl}\}$
    (1. ZK, HS15, A4b)

26. $\{0^n1^m \mid m, n \in \mathbb{N} \text{ und } m > 2n\}$
    (EK, HS15, A2a)

27. $\{0^n \mid n \text{ ist keine Quadratzahl}\}$
    (EK, HS15, A2b)

28. $\{ww^{\mathrm{R}}w \mid w \in \{0,1\}^*\}$ (1. ZK, HS16, A3a)

29. $\{0^{n^{\lceil \log_2 n \rceil}} \mid n \in \mathbb{N}\}$
    (1. ZK, HS16, A3b)

30. $\{w \in \{0,1\}^* \mid \text{es gibt ein } k \in \mathbb{N} \text{ mit } |w|_0 = k^2 \text{ oder } |w|_1 = k^3\}$
    (1. ZK, HS17, A2a)

31. $\{0^{\binom{2n}{n}} \mid n \in \mathbb{N}\}$
    (EK, HS17, A2)

32. $\{u\#v \mid u, v \in \{0,1\}^* \text{ und Nummer}(v) = 2 \cdot \text{Nummer(u)}\}$
    (1. ZK, HS18, A2a)

33. $\{0^{n^3} \mid n \in \mathbb{N}\}$
    (1. ZK, HS18, A2b)

34. $\{1^{n^3}0^n \mid n \in \mathbb{N}\}$
    (EK, HS18, A2a)

Nichtregularitätsaufgaben aus Übungen

1. $\{a^i b^j c^k \mid i, j, k \geq 0,\ i = 2j\}$ mit Lemma 3.3
   (HS08, B3, A11a)
2. $\{w \in \{a, b\}^* \mid |w|_a = 2|w|_b\}$ mit Lemma 3.3
   (HS08, B3, A11b)
3. $\{ww^{\mathrm{R}} \mid w \in \{0, 1\}^+\}$ mit Pumping-Lemma und KK-Methode
   (HS08, B4, A12)
4. $\{x1y \mid x, y \in \{0, 1\}^* |x| = |y|\}$ mit Pumping-Lemma und KK-Methode
   (HS08, B4, A13b)
5. $\{0^n 1^{3n} 0^n \mid n \in \mathbb{N}\}$
   (HS09, B3, A10)
6. $\{0^{n^3} \mid n \in \mathbb{N}\}$ mit allen drei Methoden
   (HS09, B4, A11)
7. $\{wxw^{\mathrm{R}} \mid w, x \in \{0, 1\}^* \text{ und } |w| = |x|\}$
   (HS09, B4, A12a)
8. $\{w \in \{0, 1\}^* \mid |w|_0 = 2 \cdot |w|_1\}$
   (HS10, B3, A11a)
9. $\{w1w \mid w \in \{0, 1\}^*\}$
   (HS10, B3, A11b)
10. $\{0^{2^n} \mid n \in \mathbb{N}\}$ mit Pumping-Lemma und mit KK-Methode
    (HS10, B4, A12)
11. $\{x1y \mid x, y \in \{0, 1\}^*, |x|_0 = |y|_0\}$
    (HS10, B4, A13a)
12. $\{ww \mid w \in \{0, 1\}^*\}$
    (HS10, B4, A13b; HS11, B4, A12a; HS17, B5, A13a)
13. $\{0^i 1^j \mid i, j \in \mathbb{N} \text{ und } i \neq j\}$
    (HS10, B4, BA2)
14. $\{0^{n \cdot 2^n} \mid n \in \mathbb{N}\}$
    (HS11, B4, A12b)

15. $\{a^n b^n c^n \mid n \in \mathbb{N}\}$
    (HS12, B3, A10a)

16. $\{a^{n^2} \mid n \in \mathbb{N}\}$
    (HS12, B3, A10b)

17. $\{w \in \{a,b\}^* \mid |w|_a = 3|w|_b\}$
    (HS12, B4, A11a)

18. $\{wuw \mid u \in \{a,b\}^* \text{ und } w \in \{a,b\}^+\}$ mit Lemma 3.3 und Pumping-Lemma
    (HS12, B4, A11b)

19. $\{ww^{\mathrm{R}} \mid w \in \{0,1\}^*\}$
    (HS12, B4, A12a)

20. $\{0^{n^3} \mid n \in \mathbb{N}\}$ mit Lemma 3.3
    (HS12, B4, A12b)

21. $\{a^n b^{2n} c^n \mid n \in \mathbb{N}\}$
    (HS13, B4, A11a)

22. $\{ww \mid w \in \{a,b\}^*\}$ mit Pumping-Lemma
    (HS13, B4, A11b)

23. $\{0^{2n^3} \mid n \in \mathbb{N}\}$ mit KK-Methode
    (HS13, B4, A11c)

24. $\{w \in \{0,1\}^* \mid |w|_0 = 2 \cdot |w|_1 + 3\}$
    (HS13, B4, A12a)

25. $\{0^i 1^j \mid i \neq j\}$
    (HS13, B4, A12b)

26. $\{0^n 1^{2n} 0^n \mid n \in \mathbb{N}\}$
    (HS14, B3, A9a)

27. $\{0^{n^3} \mid n \in \mathbb{N}\}$
    (HS14, B3, A9b)

28. $\{0^{n!} \mid n \in \mathbb{N}\}$ mit KK-Methode
    (HS14, B4, A10a; HS19, B4, A11a)

29. $\{w \in \{a,b,c\}^* \mid |w|_a + |w|_b = 2 \cdot |w|_c\}$ mit Lemma 3.3
    (HS14, B4, A10b)

30. $\{ucv \mid u, v \in \{a,b\}^*, u \neq v\}$ mit Pumping-Lemma
    (HS14, B4, A11a)

31. $\{0^p \mid p \in \mathbb{N} \text{ ist eine Primzahl}\}$
    (HS14, B4, A11b)

32. $\{0^m1^n0^{m+n} \mid m, n \in \mathbb{N}\}$ (HS15, B3, A9a)

33. $\{0^{n(n+1)} \mid n \in \mathbb{N}\}$
(HS15, B3, A9b)

34. $\{0^{\binom{2n}{n}} \mid n \in \mathbb{N}\}$ mit KK-Methode
(HS15, B4, A10a)

35. $\{w \in \{a, b, c\}^* \mid |w|_a = 2|w|_b\}$ mit Lemma 3.3
(HS15, B4, A10b)

36. $\{w \in \{0, 1\}^* \mid |w|_0 \neq |w|_1\}$ mit Pumping-Lemma
(HS15, B4, A11a)

37. $\{a^ib^jc^k \mid i, j, k \in \mathbb{N}, j \geq i + k\}$ mit Lemma 3.3
(HS16, B4, A10a)

38. $\{waaw \mid w \in \{a, b\}^*\}$ mit Pumping-Lemma
(HS16, B4, A10b)

39. $\{0^i1^{2^j} \mid i, j \in \mathbb{N}\}$ KK-Methode
(HS16, B4, A10c)

40. $\{0^i1^j \mid i, j \in \mathbb{N}, i \neq j\}$
(HS16, B4, A11a)

41. $\{w \in \{0, 1\}^* \mid |u|_0 \leq |u|_1$ für alle Präfixe u von $w\}$
(HS16, B4, A11b)

42. $\{w_i \mid i \in \mathbb{N}\} \subseteq \{0\}^*$ mit $|w_{i+1}| - |w_i| \geq \log_2 i$ für alle $i \in \mathbb{N}$
(HS16, B4, A12)

43. $\{x \in \{0, 1\}^* \mid |x|_0 = 2|x|_1\}$
(HS17, B4, A12a)

44. $\{u\#v\#w \mid u, v, w \in \{0, 1\}^+$ und Nummer(u)+Nummer(v)=Nummer(w)$\}$
(HS17, B4, A12b)

45. $\{0^{n2^n} \mid n \in \mathbb{N}\}$
(HS17, B5, A13b)

46. $\{a^ib^jc^k \mid i, j, k \in \mathbb{N}$ und $j > i + 2k\}$ mit Lemma 3.3
(HS18, B4, A10a)

47. $\{wabw^{\mathrm{R}} \mid w \in \{a, b\}^*\}$ mit Pumping-Lemma
(HS18, B4, A10b)

48. $\{0^{n^2 2^n} \mid n \in N\}$
(HS18, B4, A11a)

49. $\{u\#v\#w \mid u, v, w \in \{0,1\}^+ \text{ und } \text{Nummer}(u) \cdot \text{Nummer}(v) = \text{Nummer}(w)\}$
(HS18, B4, A11b)

50. $\{w \in \{0,1\}^* \mid |w|_0 \neq |w|_1\}$
(HS18, B4, A12a)

51. $\{waw^{\mathrm{R}}bw \mid w \in \{a,b\}^+\}$ mit Lemma 3.3
(HS19, B4, A10a)

52. $\{a^i b^j c^k \mid i, j, k \in \mathbb{N} \text{ und } i + j \leq \sqrt{k}\}$ mit Pumping-Lemma
(HS19, B4, A10b)

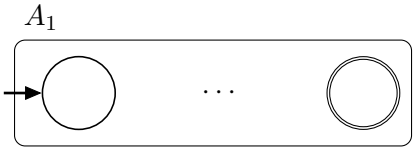
53. $\{u\#v \mid u, v \in \{0,1\}^+ \text{ und } \text{Nummer}(u) = |\text{v}|\}$
(HS19, B4, A11b)

54. $\{w_i \mid i \in \mathbb{N}\} \subseteq \{0\}^*$ unendlich mit $|w_{i+1}| - |w_i| \geq \log_2 \log_2 i$ für alle $i \in \mathbb{N}$
(HS19, B4, A12)

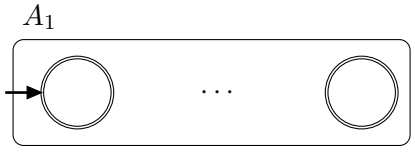
λ-NEA

In einem λ-NEA darf eine Transition nicht nur mit einem beliebigen Buchstaben $a \in \Sigma$, sondern auch mit einem λ bezeichnet sein. Jede Sprache, die von einem λ-NEAs erkannt wird, wird auch von einem NEA erkannt, es gilt also $\mathcal{L}_{\text{NEA}} = \mathcal{L}_{\lambda\text{-NEA}}$. Wir haben bereits gesehen, dass $\mathcal{L}_{\text{EA}} = \mathcal{L}_{\text{NEA}}$ gilt, es handelt sich also überall um Menge der regulären Sprachen.
Obwohl die erkannten Sprachen für EAs, NEAs und λ-NEAs dieselben sind, lassen sich Unterschiede zwischen den verschiedenen Modellen feststellen. Insbesondere gibt es Sprachen, für die sich die Mindestzustandszahl exponentiell vergrössert, wenn man keine NEAs, sondern nur EAs zulässt.
Die Vorteile von λ-NEAs gegenüber NEAs sind weniger entscheidend, man kommt immer mit derselben Anzahl Zustände aus. Dennoch werden viele Konstruktionen einfacher, insbesondere bei Sprachen mit "oder"-Bedingungen, welche sich aus der Vereinigung von mehreren Sprachen ergeben.
Seien zwei Sprachen $L_1, L_2 \in \mathcal{L}_{\text{EA}}$ über einem Alphabet Σ gegeben.
Seien A_1 und A_2 endliche Automaten mit $L(A_1) = L_1$ und $L(A_2) = L_2$.

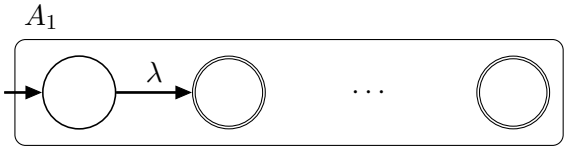
Einen λ-NEA kann man schematisch wie folgt darstellen:



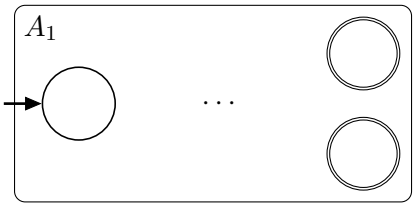
Dabei treffen wir implizit zwei Annahmen. Erstens, dass der Startzustand nicht akzeptierend ist, und zweitens, dass nur ein akzeptierender Zustand existiert, nämlich der ganz rechts. Im Schema sind alle weiteren Zustände nicht eingezeichnet und auch alle Transitionen fehlen. Die beiden Annahmen sind folgendermassen gerechtfertigt:
Falls die erste Bedingung nicht erfüllt ist, falls also der Startzustand akzeptierend ist,



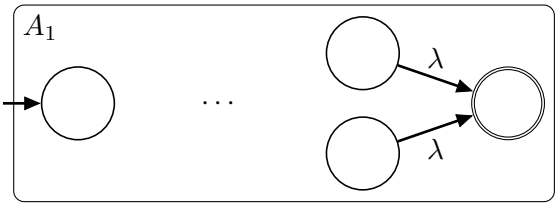
dann können wir das durch einen neuen Startzustand mit λ-Transition zum ehemaligen Startzustand beheben:



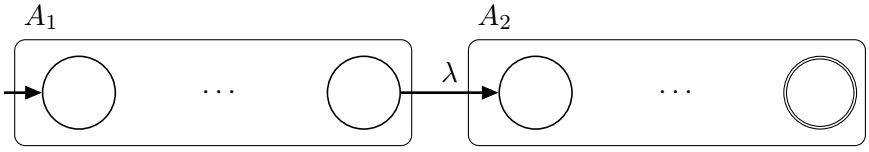
Und falls der Automat mehrere akzeptierende Zustände hat, etwa wie im folgenden Beispiel 2,



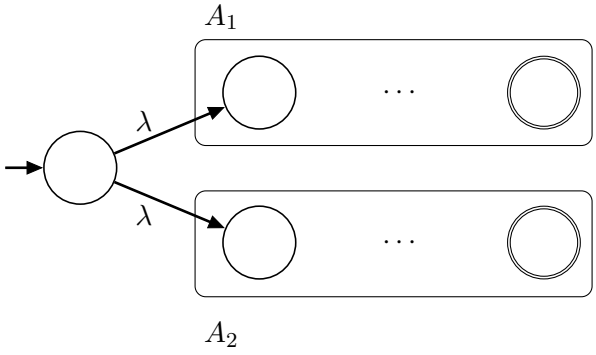
dann kann man diese in nicht-akzeptierende Zustände verwandeln und dafür mit einer λ-Transition in einen neuen, akezptierenden Zustand versehen. Somit hat man nur einen akzeptierenden Zustand, wie oben angenommen:



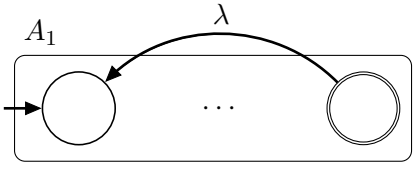
Kennt man für zwei Sprachen L_1 und L_2 bereits EAs (oder NEAs oder λ-NEAs) A_1 und A_2 mit $L(A_1) = L_1$ und $L(A_2) = L_2$, so kann man die neu erlaubten λ-Transitionen nutzen um sehr einfach Nun ist es relativ einfach λ-NEAs für die folgenden Sprachen zu erstellen: L_1L_2, L_1^+, L_1^*, $L_1 \cup L_2$, L^{R}.
Ein λ-NEA für L_1L_2:



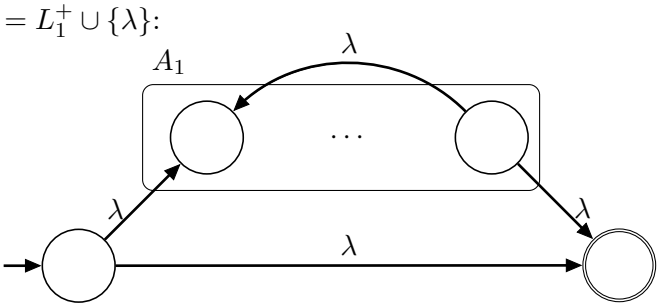
Ein λ-NEA für $L_1 \cup L_2$:



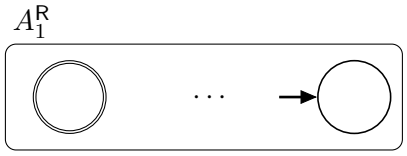
Ein λ-NEA für L_1^+:



Ein λ-NEA für $L_1^* = L_1^+ \cup \{\lambda\}$:



Und schliesslich erhält man durch Umkehren aller Transitionspfeile und Vertauschen der Funktion des Startzustands und des einzigen akzeptierenden Zustands einen λ-NEA für die Spiegelsprache $L_\mathsf{R} \coloneqq \{w^\mathsf{R} \mid w \in L\}$:



Weitere Abgeschlossenheitseigenschaften: Wenn L regulär ist, dann auch $L_x = \{y \mid xy \in L\}$. (Folgt durch Beweis von Satz 3.1.) Für $\cup$, $\cap$, $\setminus$ und $\triangle$ (symmetrische Differenz) kann man die Abgeschlossenheit einfach mit Hilfe von Produktautomaten zeigen, man muss nur die Menge der akzeptierenden Zustände jeweils richtig anpassen.

Wichtige Bemerkung zu Abgeschlossenheitseigenschaften

Im Allgemeinen gilt die Umkehrung der Abgeschlossenheitseigenschaften nicht. Beispielsweise gilt:

$$L_1 \in \mathcal{L}_\text{EA} \land L_2 \in \mathcal{L}_\text{EA} \implies L_1 \cup L_2 \in \mathcal{L}_\text{EA}$$

Es gilt auch die Kontraposition

$$L_1 \notin \mathcal{L}_\text{EA} \lor L_2 \notin \mathcal{L}_\text{EA} \impliedby L_1 \cup L_2 \notin \mathcal{L}_\text{EA}.$$

Es gilt aber nicht

$$L_1 \notin \mathcal{L}_\text{EA} \lor L_2 \notin \mathcal{L}_\text{EA} \implies L_1 \cup L_2 \notin \mathcal{L}_\text{EA}.$$

Ebenso ist auch folgende Implikation falsch:

$$L_1 \notin \mathcal{L}_\text{EA} \land L_2 \notin \mathcal{L}_\text{EA} \implies L_1 \cup L_2 \notin \mathcal{L}_\text{EA}.$$

Jede Sprache $L \notin \mathcal{L}_\text{EA}$ liefert ein einfaches Gegenbeispiel, wenn man $L_1 \coloneqq L$ und $L_2 \coloneqq L^\complement$ wählt, denn dann ist $L_1 \cup L_2 = \Sigma^* \in \mathcal{L}_\text{EA}$.

Für eine nicht-reguläre Sprache L lässt sich auch keine allgemeine Aussage über die Regularität bzw. Nicht-Regularität von Teilsprachen $L' \subseteq L$ und Übersprachen $L' \supseteq L$ machen. Das leere Sprache und Σ^* liefern wieder einfache Gegenbeispiele.
Wenn bewiesen werden muss, dass einen Sprache nichtregulär ist, muss man aufpassen, keine dieser falschen Schlüsse zu verwenden.

Mindestzustandszahlbeweis — Beschreibung und Muster

Der Beweis, dass jeder (deterministische) endliche Automat, der eine vorgegebene Sprache erkennt, mindestens eine vorgegebene Anzahl Zustände hat, ist eine häufige Prüfungsaufgabe.

Erklärung des Beweises

Sei eine Sprache L gegeben. Wir sollen beweisen, dass ein endlicher Automat A für L (also ein endlicher Automat A mit $L(A) = L$) mindestens n Zustände hat. Die Grundidee ist, dass wir n Wörter $w_1, \dots, w_n \in \Sigma^*$ angeben, für die wir beweisen können, dass sie nicht in denselben Zustand führen (also nicht in derselben Zustandsklasse liegen). Dies folgt aus Lemma 3.3, wenn wir für jedes Paar $w_i \neq w_j$ ein $z_{i,j} \in \Sigma^*$ angegeben können, so dass gilt

$$w_i z \in L \not\Leftrightarrow w_j z \in L.$$

(In anderen Worten: Man sucht n Wörter $w_1, \dots, w_n$, die sich paarweise durch Anhängen des angegebenen $z_{i,j}$ unterscheiden lassen.)
Die richtige Wahl dieser Wörter, welche natürlich von der Sprache abhängt, lässt sich gut in Form einer Tabelle angeben, die in der oberen rechten Hälfte gefüllt ist, siehe unten. (Die Hälfte links unten in der Tabelle könnte man einfach durch Transponieren bzw. Spiegeln von der anderen Hälfte ergänzen. Auf der Diagonale kann natürlich kein passendes z existieren.)
Man muss jetzt zeigen, dass die dadurch gegebene Wahl für z jeweils die Eigenschaft $w_i z \in L \not\Leftrightarrow w_j z$ erfüllt ist. Anders gesagt: Es gilt entweder $w_i z \in L$ und $w_j z \notin L$ oder es gilt $w_i z \notin L$ und $w_j z \in L$. Wenn dies offensichtlich ist, ist kein expliziter Beweis verlangt.

Beweismuster

Wid.-Ann.: Es gibt ein EA für L mit weniger als n Zuständen.
Wir haben n Wörter $w_1, \dots, w_n$.
Nach Schubfachprinzip gibt es $i < j$, so dass $\hat{\delta}(q_0, w_i) = \hat{\delta}(q_0, w_j)$.
Mit Lem. 3.3 folgt $\forall z \in \Sigma^*\colon\ w_i z \in L \Leftrightarrow w_j z \in L$.
Für $z = z_{i,j}$ gemäss Tabelle gilt aber $w_i z_{i,j} \in L \not\Leftrightarrow w_j z_{i,j} \in L$. Wid. □

z	w_2	w_3	$\dots$	w_{n-2}	w_{n-1}	w_n
w_1	$z_{1,2}$	$z_{1,2}$	$\dots$	$z_{1,n-2}$	$z_{1,n-1}$	$z_{1,n}$
w_2		$z_{2,2}$	$\dots$	$z_{2,n-2}$	$z_{2,n-1}$	$z_{2,n}$
$\vdots$			$\ddots$	$\vdots$	$\vdots$	$\vdots$
w_{n-3}				$z_{n-3,n-2}$	$z_{n-2,n-1}$	$z_{n-2,n}$
w_{n-2}					$z_{n-2,n-1}$	$z_{n-2,n}$
w_{n-1}						$z_{n-1,n}$

Bemerkung
Der Beweis besteht aus vier Teilen:

1. Angeben von $w_1, \ldots, w_n$.
2. Aufstellen der Tabelle. (Oder anderweitige Angabe der richtigen Wahl für z.)
3. Nachweis der geforderten Eigenschaft $w_i z \in L \not\Leftrightarrow w_j z \in L$. (Oft trivial.)
4. Führen des kurzen Widerspruchbeweises unter Verwendung von Lemma 3.3.

Bei dem Beweis sind alle vier Punkte wichtig. Insbesondere muss der Beweisrahmen für den Widerspruchsbeweis stimmen und das Lemma 3.3. an der richtigen Stelle angewendet werden.
Nota bene: Nirgends wird im Beweis ein konkreter endlicher Automat verwendet. Ein solcher ist also nicht notwendig, man kann die Tabelle auch konstruieren, wenn die Sprache nur in Mengenschreibweise angegeben ist.
Ein eventuell bereits vorhandener endlicher Automat für L, der so wenig Zustände hat wie gefordert, hilft aber sehr in drei Aspekten:

- Beim Finden der Wörter w_i.
- Beim Finden der $z_{i,j}$.
- Beim Beweis von $w_i z_{i,j} \in L \not\Leftrightarrow w_j z_{i,j} \in L$.

Wenn man nämlich einen minimalen EA für L hat (minimal bedeutet mit so wenig Zuständen wie möglich), dann kann man einfach zu jedem Zustand q_i ein Wort $w_i \in \mathrm{Kl}[q_i]$ wählen (also eines mit $\hat{\delta}(q_0, w_i) = q$). Der Einfachheit halber wählt man üblicherweise ein möglichst kurzes. Dann sucht man die passenden $z_{i,j}$ mit Hilfe der Diagrammdarstellung des Automaten und schreibt sie in eine Tabelle.
Falls der eventuell vorhandene Automat nicht minimal ist, muss man n geeignete Zustände auswählen, aus deren Klassen man dann die Wörter w_i wählt. Falls man die falschen Zustände wählt, existiert nicht für jedes Wortpaare (w_i, w_j) ein z mit der geforderten Eigenschaft. (Das bedeutet dann genau, dass diese Zustände verschmolzen werden können.) Es lohnt sich also, den vorhandenen endlichen Automaten zuerst möglichst klein zu machen.
Ein einfacher allgemeiner Trick bei der Wortwahl: Für alle Paare von einem akzeptierenden mit einem nichtakzeptierenden Zustand kann λ wählen.
Man kann die richtige Wahl der w_i und/oder $z_{i,j}$ manchmal auch als Formel angeben statt konkret, dann muss man aber meist auch mit einer Formel beweisen, dass sie die richtige Eigenschaft haben.
Wenn man die w_i und $z_{i,j}$ angegeben hat, reicht es meist auch zu sagen, dass die Eigenschaft $w_i z_{i,j} \in L \not\Leftrightarrow w_j z_{i,j} \in L$ anhand des Automaten sofot überprüft werden kann.

Rekursivitätstheorie: Definition einiger wichtiger Sprachen

Die grauen Anmerkungen sind notwendig, doch vermutlich kein Punktabzug bei Fehlen.

Im Buch definierte Sprachen:

$L_{\text{diag}} \coloneqq \{\ w_i \mid M_i \text{ akzeptiert } w_i \in \{0,1\}^* \text{ nicht } \}$, wobei w_i/M_i i. Wort/TM in kan. Ordnung.

$L_{\text{U}} \coloneqq \{\ \text{Kod}(M)\#w \mid M \text{ akzeptiert } w \in \{0,1\}^*\ \} \subseteq \{0,1,\#\}^*$

$L_{\text{H}} \coloneqq \{\ \text{Kod}(M)\#w \mid M \text{ hält auf } w \in \{0,1\}^*\ \} \subseteq \{0,1,\#\}^*$

$L_{\text{H},\lambda} \coloneqq \{\ \text{Kod}(M) \mid M \text{ hält auf } \lambda\ \} \subseteq \{0,1\}^*$

$L_{\text{empty}} \coloneqq \{\ \text{Kod}(M) \mid L(M) = \emptyset\ \} \subseteq \{0,1\}^*$

$L_{\text{EQ}} \coloneqq \{\ \text{Kod}(M_1)\#\text{Kod}(M_2) \mid L(M_1) = L(M_2)\ \} \subseteq \{0,1,\#\}^*$

Ihre Komplemente:

$L^{\complement}_{\text{diag}} = \{\ w_i \in \{0,1,\#\}^* \mid M_i \text{ akzeptiert } w_i \in \{0,1\}^*\ \}$, wobei w_i/M_i i. Wort/TM in kan. Ord.

$L^{\complement}_{\text{U}} = \{\ \text{Kod}(M)\#w \mid M \text{ akzeptiert } w \in \{0,1\}^* \text{ nicht } \}$
$\cup \{\ x \mid x \text{ nicht von Form Kod}(M)\#w.\ \} \subseteq \{0,1,\#\}^*$

$L^{\complement}_{\text{H}} = \{\ \text{Kod}(M)\#w \mid M \text{ hält nicht auf } w \in \{0,1\}^*\ \}$
$\cup \{\ x \mid x \text{ nicht von Form Kod}(M)\#w\ \} \subseteq \{0,1,\#\}^*$

$L^{\complement}_{\text{H},\lambda} = \{\ \text{Kod}(M)\#w \mid M \text{ hält nicht auf } \lambda\ \}$
$\cup \{\ x \mid x \text{ nicht von Form Kod}(M)\#w\ \} \subseteq \{0,1\}^*$

$L^{\complement}_{\text{empty}} = \{\ \text{Kod}(M) \mid L(M) \neq \emptyset\ \}$
$\cup \{\ x \mid x \text{ nicht von Form Kod}(M)\ \} \subseteq \{0,1\}^*$

$L^{\complement}_{\text{EQ}} = \{\ \text{Kod}(M_1)\#\text{Kod}(M_2) \mid L(M_1) \neq L(M_2)\ \}$
$\cup \{\ x \mid x \text{ nicht von Form Kod}(M_1)\#\text{Kod}(M_2)\#w\ \} \subseteq \{0,1,\#\}^*$

Drei weitere relevante Sprachen:

$L_{\text{acc}} = \{\ \text{Kod}(M) \mid M \text{ akzeptiert jede Eingabe}\ \} \subseteq \{0,1\}^*$

$L_{\text{rej}} = \{\ \text{Kod}(M) \mid M \text{ verwirft jede Eingabe}\ \} \subseteq \{0,1\}^*$

$L_{\text{inf}} = \{\ \text{Kod}(M) \mid M \text{ terminiert auf keiner Eingabe}\ \} \subseteq \{0,1\}^*$

$L_{\text{all}} \coloneqq L_{\text{acc}}$ und $L_{\text{nohalt}} \coloneqq L_{\text{inf}}$ sind häufige Alternativbezeichnungen. Beachte aber: $L_{\text{empty}} \neq L_{\text{rej}}$.

Rekursivitätstheorie: Sprachen bekannter Klassenzugehörigkeit

Einordnung der Sprachen

Die Platzierung der grauen Sprachen ist nicht explizit bewiesen, folgt aber leicht.

$\mathcal{L}_\mathrm{R}$	$\mathcal{L}_\mathrm{RE} \setminus \mathcal{L}_\mathrm{R}$	$\mathcal{L}_\mathrm{RE}^\complement$
	$L_\mathrm{diag}^\complement$	L_diag
	$L_\mathrm{empty}^\complement$	L_empty
	L_U	$L_\mathrm{U}^\complement$
	L_H	$L_\mathrm{H}^\complement$
	$L_{\mathrm{H},\lambda}$	$L_{\mathrm{H},\lambda}^\complement$
		$L_\mathrm{EQ}, L_\mathrm{EQ}^\complement$

Zugehörige Aussagen des Buches, 5. Auflage (Beweismethode jeweils in Klammern)

In Grau stehen Aussagen, deren Beweise im Buch nur als Aufgaben gestellt sind.

Satz 5.5: $L_\mathrm{diag} \notin \mathcal{L}_\mathrm{RE}$ (Diagonalisierung)

Aufgabe 5.15: EE-Reduzierbarkeit ist transitiv. (Konkatenation)

Lemma 5.4 $L \in \mathcal{L}_\mathrm{R} \Leftrightarrow L^\complement \in \mathcal{L}_\mathrm{R}$ (Triviale R-Reduktion)

Korollar 5.2 $L_\mathrm{diag}^\complement \notin \mathcal{L}_\mathrm{R}$ (Lemma 5.4)

Lemma 5.5: $L_\mathrm{diag}^\complement \in \mathcal{L}_\mathrm{RE}$ (Beschreibung von TM)

Satz 5.6: $L_\mathrm{U} \in \mathcal{L}_\mathrm{RE}$ (Konstruktion passender TM)

Satz 5.7: $L_\mathrm{U} \notin \mathcal{L}_\mathrm{R}$ ($L_\mathrm{diag} \leq_\mathrm{R} L_\mathrm{U}$)

Aufgabe 5.17: $L_\mathrm{H} \in \mathcal{L}_\mathrm{RE}$ (Konstruktion passender TM)

Satz 5.8: $L_\mathrm{H} \notin \mathcal{L}_\mathrm{R}$ ($L_\mathrm{U} \leq_\mathrm{R/EE} L_\mathrm{H}$)

Lemma 5.6: $L_\mathrm{empty}^\complement \in \mathcal{L}_\mathrm{RE}$ (Konstruktion passender NTM/TM)

Lemma 5.7: $L_\mathrm{empty}^\complement \notin \mathcal{L}_\mathrm{R}$ ($L_\mathrm{U} \leq_\mathrm{EE} L_\mathrm{empty}^\complement$)

Korollar 5.4: $L_\mathrm{empty} \notin \mathcal{L}_\mathrm{R}$ (Lemma 5.4)

Aufgabe 5.19: $L_\mathrm{empty}, L_\mathrm{U}^\complement, L_\mathrm{H}^\complement \notin \mathcal{L}_\mathrm{RE}$ (Aufgabe 5.22 mit $L_\mathrm{empty}^\complement, L_\mathrm{U}, L_\mathrm{H} \in \mathcal{L}_\mathrm{RE} \setminus \mathcal{L}_\mathrm{R}$)

Korollar 5.5: $L_\mathrm{EQ} \notin \mathcal{L}_\mathrm{R}$ ($L_\mathrm{empty} \leq_\mathrm{EE} L_\mathrm{EQ}$)

Lemma 5.8: $L_{\mathrm{H},\lambda} \notin \mathcal{L}_\mathrm{R}$ ($L_\mathrm{H} \leq_\mathrm{EE} L_{\mathrm{H},\lambda}$)

Aufgabe 5.22: $L, L^\complement \in \mathcal{L}_\mathrm{RE} \Leftrightarrow L \in \mathcal{L}_\mathrm{R}$.
(“$\Rightarrow$”: Parallele Simulation. “$\Leftarrow$”: Lemma 5.4 von oben und $\mathcal{L}_\mathrm{R} \subseteq \mathcal{L}_\mathrm{RE}$)

Rekursivitätstheorie: Beweismethoden allgemein

- In der *Rekursivitätstheorie* oder *Berechenbarkeitstheorie* geht um die Frage, ob eine gegebene Sprache rekursiv oder rekursiv aufzählbar oder keines von beidem ist.
- Für eine Sprache L gelten folgende Definitionen.

$$\begin{aligned}
L \text{ regulär} &\iff L \in \mathcal{L}_{\text{EA}} \iff \exists\,\text{EA } A\colon \quad L(A) = L \iff L \text{ mit EA erkennbar.}\\
L \text{ rekursiv} &\iff L \in \mathcal{L}_{\text{R}} \iff \exists\,\text{Alg. } A\colon \quad L(A) = L \iff L \text{ entscheidbar.}\\
L \text{ rek. aufz.} &\iff L \in \mathcal{L}_{\text{R}} \iff \exists\,\text{TM } M\colon L(M) = L \iff L \text{ erkennbar.}
\end{aligned}$$

- Das Wort „Algorithmus“ ist einfach eine kurze Bezeichnung für „Turingmaschine, die immer terminiert“. Ein „Programm“ ist eine Turingmaschine, die nicht zwingend terminieren muss.
- Wenn für eine Sprache L bewiesen werden soll, dass $L \in \mathcal{L}_{\text{R}}$, dann kann man das mittels einer Reduktion $L \leq_{\text{R}} L'$ für ein $L' \in \mathcal{L}_{\text{R}}$ tun (siehe Unterlagen zu den Reduktionen). Das ist meist aber viel umständlicher als die folgende direkte Methode.
- Die direkte Methode ist es, einen Algorithmus A anzugeben, der L entscheidet. Einen solchen Algorithmus kann man in Pseudocode beschreiben. Man muss $L(A) = L$ beweisen und dass A tatsächlich ein Algorithmus ist, also immer terminiert.
- Ebenso für $L \in \mathcal{L}_{\text{RE}}$, nur dass man jetzt eine TM M mit $L(M) = L$ angeben kann, die nicht zwingend immer terminiert. Man kann auch eine NTM angeben, weil jede NTM eine äquivalente TM hat. In beiden Fällen ist $L(M) = L$ zu beweisen.
- Ein wichtiger Trick ist die „parallele Simulation aller Wörter“, bei der man in irgendeinem Diagonalverfahren immer mehr Wörter für immer mehr Schritte simulieren. Damit verhindert man es, bei einem Wort in einer Endlosschleife stecken zu bleiben. (Oft hat man die Wahl zwischen einer parallelen Simulation und dem Verwenden von Nichtdeterminismus zum Erraten des richtigen Wortes.)
- Beweis von $L \notin \mathcal{L}_{\text{RE}}$. Hier kennen wir offiziell nur zwei Methoden, inoffiziell gibt es drei.
  1. Ein Diagonalargument, beispielsweise für $L_{\text{diag}} \notin \mathcal{L}_{\text{RE}}$, was die Verankerung ursprüngliche für alle Nichtrekursivitätsbeweise via Reduktionen darstellt.
  2. Anwendung des Satzes $L, L^{\complement} \in \mathcal{L}_{\text{RE}} \implies L \in \mathcal{L}_{\text{R}}$ in Widerspruchsbeweis.
  3. Inoffiziell: Eine EE-Reduktion. Dass dies funktioniert muss aber kurz bewiesen werden. (Siehe Unterlagen zu EE-Reduktion.)

- Beweis von $L \notin \mathcal{L}_{\mathrm{R}}$. Hier gibt es eine triviale und zwei weitere offizielle Methoden.
  1. Trivial: Falls $L \notin \mathcal{L}_{\mathrm{RE}}$ folgt es sofort.
  2. Eine R-Reduktion oder EE-Reduktion.
  3. Satz von Rice anwenden.
- Zum Satz von Rice:
  Er kann für gewisse Sprache L beweisen, dass $L \notin \mathcal{L}_{\mathrm{R}}$ (aber nicht $L \notin \mathcal{L}_{\mathrm{RE}}$. Für die Anwendung sind folgende vier Dinge zu verifzieren.
  1. $L \subseteq \mathrm{KodTM}$
  2. $\exists$ TM M: $\mathrm{Kod}(M) \in L$
  3. $\exists$ TM M: $\mathrm{Kod}(M) \notin L$
  4. $\forall$ TM M_1, M_2: $L(M_1) = L(M_2) \implies (\mathrm{Kod}(M_1) \in L \iff \mathrm{Kod}(M_2) \in L)$

  Konkret tut man dies am besten wie folgt.
  1. Prüfen, dass L die Form $\{\mathrm{Kod}(M) \mid M \text{ ist TM und } \dots\}$ hat.
  2. M_{acc}, M_{rej}, M_{inf} und ähnliches verwenden.
  3. M_{acc}, M_{rej}, M_{inf} und ähnliches verwenden.
  4. Prüfen, dass in den Bedingungen von $L = \{\mathrm{Kod}(M) \mid M \text{ ist TM und } \dots\}$ überall nur $L(M)$ auftaucht und nirgends M direkt. Es reicht, wenn man die Bedingung so umschreiben kann, dass dies gilt.

Spezifische Hinweise zum Lösen von Aufgaben mittels EE-Reduktionen und ein paar grundlegende Tricks

- Aus $L_1 \leq_{\text{EE}} L_2$ folgt sofort $L_1 \leq_{\text{R}} L_2$. Es reicht, das so hinzuschreiben.
- Ein Bild zu zeichnen ist optional. Ein Beweis für eine EE-Reduktion besteht nur aus der Beschreibung eines Algorithmus F, der $f(x)$ berechnet, und dem Korrektheitsbeweis, also dem Beweis von $x \in L_1 \iff f(x) \in L_2$.
- Der Algorithmus F zwingend ein Algorithmus sein, also immer terminieren.
- Die Prüfung auf ein x von falscher Form sollte von F immer zuerst durchgeführt werden.
- Ebenso dann beim Korrektheitsbeweis, der Fall einer falschen Form sollte zuerst separat abgehandelt werden.
- Es ist oft besser, die beiden Richtungen $x \in L_1 \implies f(x) \in L_2$ und $x \in L_1 \impliedby f(x) \in L_2$ separat zu beweisen. Wenn man den Fall der falschen Form zuerst abhandelt, geht es aber oft auch gut mit einer Äquivalenzkette.
- Transition nach $q_{\text{accept}}/q_{\text{reject}}$ umleiten nach $q_{\text{accept}}/q_{\text{reject}}$/Endlosschleife.
  Achtung, man kann nicht Endlosschleifen umleiten. Beispiele:
  - Für $L_{\text{diag}}^{\complement} \leq_{\text{EE}} L_{\text{U}}$ keine Umleitung notwendig.
  - Für $L_{\text{H}} \leq_{\text{EE}} L_{\text{U}}$ umleiten von q_{reject} nach q_{accept}.
  - Für $L_{\text{U}} \leq_{\text{EE}} L_{\text{H}}$ umleiten von q_{reject} nach Endlosschleife.
  - Für $L_{\text{diag}} \leq_{\text{EE}} L_{\text{U}}$ ist man versucht, von q_{accept} nach q_{reject} umleiten und von q_{reject} sowie Enlosschleifen nach q_{accept}. Letzteres ist aber leider unmöglich.
- Turingmaschine konstruieren, die unabhängig von ihrer eigenen Eingabe immer dasselbe tut. Eine solche Turingmaschine verwirft alles oder akzeptiert alles oder läuft auf allen Eingaben endlos.
- Falls die Sprache Wörter mit Codierungen mehrerer Turingmaschinen enthält, z. B. $\text{Kod}(M_1)\#\text{Kod}(M_2)$, dann entweder beide gleich wählen, $M_1 = M_2$, oder eine der beiden Maschinen fest als TM M_{acc}, M_{rej}, M_{inf}, die immer akzeptiert, verwirft, endlos läuft.
  (Dies ist analog zu Nichtregularitätsbeweisen mit einem oder/und in der Sprachbeschreibung.)
- Fortgeschrittener Trick, ziemlich kompliziert: Die Eingabe nutzen, um Endlosschleifen zu erkennen. Geht nur bei Reduktion auf eine Sprache, in der eine Bedingung für unendlich viele Wörter enthalten ist, z.B. M_{all}.
  Beispiel: $L_{\text{diag}} \leq_{\text{EE}} L_{\text{all}}$.

  (L_{all} ist eine rekursiv aufzählbare Sprache, deren Komplement auch nicht rekursiv aufzählbar ist.)

EE-Reduktionen, Zusammenhang von RE- und R-Reduktionen, alle Reduktionen zwischen L_{diag}, L_{U}, L_{H}, $L^{\complement}_{\mathsf{diag}}$, $L^{\complement}_{\mathsf{U}}$, $L^{\complement}_{\mathsf{H}}$

Definition von $\leq_{\mathsf{EE}}$

Für zwei Entscheidungsprobleme (Σ_1, L_1) und (Σ_2, L_2) sagen wir, dass L_1 auf L_2 EE-reduzierbar ist, wenn eine berechenbare Abbildung f existiert, die eine Probleminstanz $x \in \Sigma_1^*$ des ersten Problems auf eine Instanz $f(x) \in \Sigma_2^*$ des zweiten Problems abbildet, welche äquivalent ist, also dieselbe Antwort ("Ja, das Wort ist Element der Sprache." bzw. "Nein, das Wort ist nicht Element der Sprache.") liefert. Berechenbar ist eine Funktion nach Definition, wenn ein Programm (d.h., eine Turing-Maschine) F existiert, das auf der Eingabe x das $f(x)$ als Ausgabe liefert. Somit muss F ein Algorithmus sein (d.h., immer terminieren), sonst gäbe es eine Eingabe x zur der kein Funktionswert $f(x)$ ausgegeben wird.
Kurz und formal ist die Definition:

$$L_1 \leq_{\mathrm{EE}} L_2 \quad :\Longleftrightarrow \quad (\ \exists\, \text{Alg. } F\colon \forall x \in \Sigma_1^*\colon \quad x \in L_1 \Leftrightarrow f(x) \in L_2\ )$$

Zur formalen Schreibeweise $L_1 \leq_{\mathrm{R}} L_2$ gehört die Sprechweise "L_1 ist auf L_2 reduzierbar". Dies ist recht irreführend, denn "reduzieren" klingt nach verkleinern, die rechte Seite ist aber "grösser". Intuitiv verständlich wird das Symbol $\leq_{\mathrm{R}}$ erst, wenn man es bezüglich der Schwierigkeit interpretiert, also als "leichter-gleich". Man muss sehr gut aufpassen, nicht L_1 und L_2 zu vertauschen; das passiert sehr gerne.

Beweis von $L_1 \leq_{\mathsf{EE}} L_2$

Zum Nachweis von $L_1 \leq_{\mathrm{EE}} L_2$ ist ein Programm F zu beschreiben, sodass alle Anforderungen erfüllt sind:

- F terminiert immer. (Das heisst: F ist ein Algorithmus.)
- F berechnet eine Funktion $f : \Sigma_1^* \to \Sigma_2^*$.
- Diese Funktion erfüllt $x \in L_1 \iff f(x) \in L_2$ erfüllt,

Üblicherweise ist schematisch zu beschreiben, wie F aus der Eingabe x die Ausgabe $f(x)$ berechnet und dann $x \in L_1 \iff f(x) \in L_2$ in einer Äuqivalenzkette beweisen. Der erste Punkt ist meist offensichtlich erfüllt und erfordert daher keinen expliziten Beweis.

$\leq_{\mathsf{EE}}$ ist reflexiv

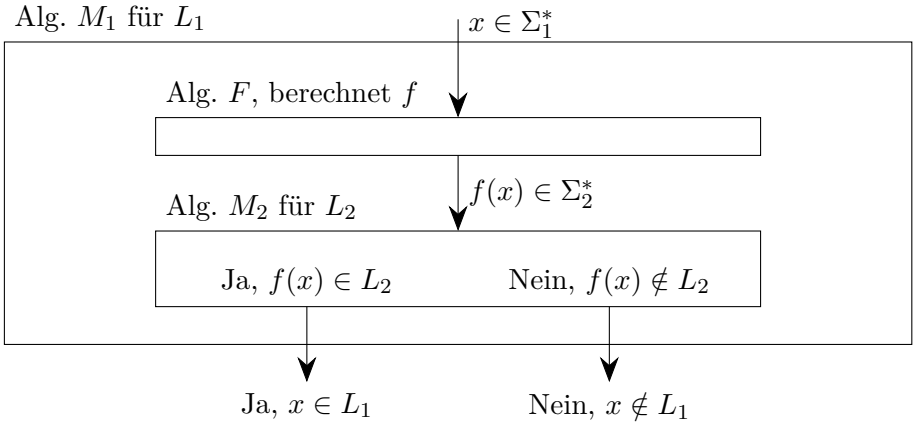
Es gilt offensichtlich $L \leq_{\mathrm{EE}} L$ mit $f(x) \coloneqq x$.

$\leq_{\mathsf{EE}}$ ist transitiv

Aus $L_1 \leq_{\mathrm{EE}} L_2$ und $L_2 \leq_{\mathrm{EE}} L_3$ folgt $L_1 \leq_{\mathrm{EE}} L_3$ via $f_3 \coloneqq f_2 \circ f_1$.

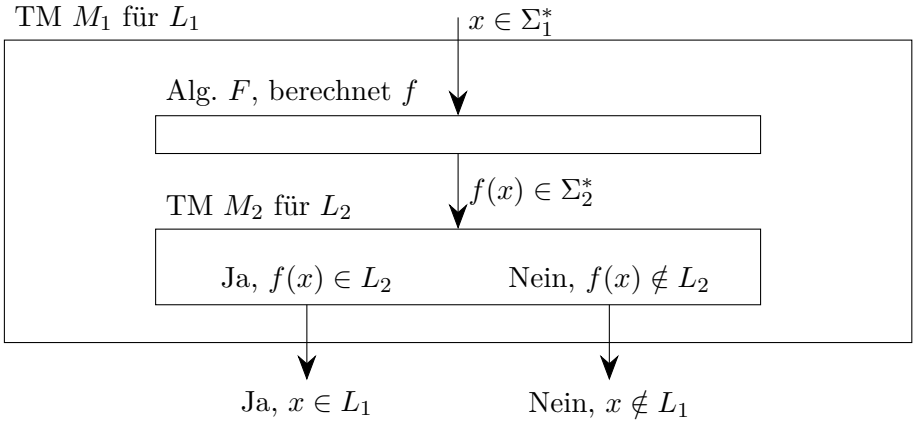
$\leq_{\mathsf{EE}} \implies \leq_{\mathsf{R}}$

Eine EE-Reduktion ist ein Spezialfall einer R-Reduktion, denn $L_1 \leq_{\mathrm{R}} L_2$ bedeutet per Definition $L_1 \in \mathcal{L}_{\mathrm{R}} \Longleftarrow L_2 \in \mathcal{L}_{\mathrm{R}}$. Setzt man $L_2 \in \mathcal{L}_{\mathrm{R}}$ voraus, dann existiert ein Algorithmus M_2 für L_2, dem man die Ausgabe des Algorithmus F übergeben kann. Damit hat man ein Algorithmus M_1 für L_1 konstruiert und die Behauptung bewiesen.



$\leq_{\mathsf{EE}} \implies \leq_{\mathsf{RE}}$

RE-Reduzierbarkeit ist analog zur R-Reduzierbarkeit definiert:
$L_1 \leq_{\mathrm{RE}} L_2$ bedeutet per Definition $L_1 \in \mathcal{L}_{\mathrm{RE}} \Longleftarrow L_2 \in \mathcal{L}_{\mathrm{RE}}$.
Der Begriff der RE-Reduzierbarkeit taucht weder in der Vorlesung noch im Buch auf.
EE-Reduzierbarkeit impliziert nicht nur R-, sondern auch RE-Reduzierbarkeit. (Somit ist die EE-Reduzierbarkeit eine stärkere Eigenschaft als R-Reduzierbarkeit, was die strengeren Restriktionen bei der Konstruktion rechtfertigt.) Man zeigt dies durch dieselbe Konstruktion wie oben; nun bloss ohne die Garantie, dass M_2 terminiert; dadurch ist vielleicht auch M_1 kein Algorithmus mehr, sondern nur noch eine Turing-Maschine, die auf gewissen Eingaben nicht terminiert.



$\leq_{\mathsf{R}} \not\Rightarrow \leq_{\mathsf{RE}}$

Ein Gegenbeispiel besteht aus zwei Sprachen L_1 und L_2, für die

$$\text{zwar } L_1 \leq_{\mathrm{R}} L_2, \text{ aber } L_1 \notin \mathcal{L}_{\mathrm{RE}} \text{ und } L_2 \in \mathcal{L}_{\mathrm{RE}}\,.$$

Die Wahl $L_1 = L_{\mathrm{diag}}$ und $L_2 = L_{\mathrm{diag}}^{\complement}$ erfüllt diese Bedingungen:

- Jede Sprache lässt sich trivialerweise auf ihr Komplement R-reduzieren, es sind dazu nur die beiden möglichen Ausgaben zu vertauschen.
- Es gilt $L_{\mathrm{diag}} \notin \mathcal{L}_{\mathrm{RE}}$ und $L_{\mathrm{diag}}^{\complement} \in \mathcal{L}_{\mathrm{RE}} \setminus \mathcal{L}_{\mathrm{R}}$.

Bemerkung: Es gilt sowohl $L_1 \leq_{\mathrm{R}} L_2$ als auch $L_1, L_2 \notin \mathcal{L}_{\mathrm{R}}$, dies ist kein Widerspruch.

$\leq_{\mathsf{RE}} \not\Rightarrow \leq_{\mathsf{R}}$

Ein Gegenbeispiel besteht aus zwei Sprachen L_1 und L_2, für die

$$\text{zwar } L_1 \leq_{\mathrm{RE}} L_2, \text{ aber } L_1 \notin \mathcal{L}_{\mathrm{R}} \text{ und } L_2 \in \mathcal{L}_{\mathrm{R}}.$$

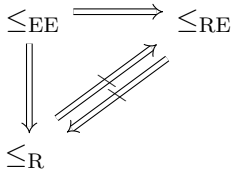
Wir wählen L_1 und L_2 so, dass $L_1 \in \mathcal{L}_{\mathrm{RE}} \setminus \mathcal{L}_{\mathrm{R}}$ und $L_2 \in \mathcal{L}_{\mathrm{R}}$.
Eine Möglichkeit dafür ist $L_1 = L_{\mathrm{U}}$ und $L_2 = \{0^i\#1^i \mid i \in \mathbb{N}\}$. Damit sind die Bedingungen erfüllt:

- $L_1 \leq_{\mathrm{RE}} L_2$ bedeutet $L_1 \in \mathcal{L}_{RE} \impliedby L_2 \in \mathcal{L}_{RE}$. Dies ist automatisch erfüllt, wenn $L_1 \in \mathcal{L}_{\mathrm{RE}}$.
- Aus $L_1 \in \mathcal{L}_{\mathrm{RE}} \setminus \mathcal{L}_{\mathrm{R}}$ folgt offensichtlich $L_1 \notin \mathcal{L}_{\mathrm{R}}$ gilt offensichtlich auch.

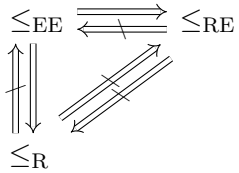
Bemerkung: Aus $L_1 \in \mathcal{L}_{\mathrm{RE}}$ folgt die Existenz einer Turingmaschine für L_1. Diese ist automatisch eine degenerierte RE-Reduktion auf L_2, welche die zusätzlich angenommene Turingmaschine für L_2 einfach gar nicht benutzt.

Beziehungen zwischen R-, RE- und EE-Reduzierbarkeit.

Wir haben mit den letzten vier Sätzen gezeigt, dass



gilt, woraus



folgt.

$\leq_{\mathsf{EE}}$ ist abgeschlossen unter Komplementbildung

Betrachten wir die Definition einer EE-Reduktionen, so sehen wir:

$$\begin{aligned} \text{Offensichtlich ist } x \in L_1 &\iff f(x) \in L_2 \\ \text{äquivalent zu } x \notin L_1 &\iff f(x) \notin L_2 \end{aligned}$$

und damit äquivalent zu $x \in L_1^{\mathsf{C}} \iff f(x) \in L_2^{\mathsf{C}}$.

Somit ist eine EE-Reduktion von L_1 auf L_2 zugleich eine EE-Reduktion von L_1^{C} auf L_2^{C}, formal gilt:

$$L_1 \leq_{\mathrm{EE}} L_2 \iff L_1^{\mathsf{C}} \leq_{\mathrm{EE}} L_2^{\mathsf{C}}$$

Aus $L \leq_{\mathsf{EE}} L^{\mathsf{C}}$ folgt $L^{\mathsf{C}} \leq_{\mathsf{EE}} L$.

Dies folgt aus direkt aus dem vorigen Resultat: $L^{\mathsf{C}} \leq_{\mathrm{EE}} \left(L^{\mathsf{C}}\right)^{\mathsf{C}} = L$. Formal kann man die Reduzierbarkeit in beide Richtungen auch kurz schreiben als $L =_{\mathrm{EE}} L^{\mathsf{C}}$.

L und L^{C} können nicht beide in $\mathcal{L}_{\mathsf{RE}} \setminus \mathcal{L}_{\mathsf{R}}$ liegen.

Angenommen, $L, L^{\mathsf{C}} \in \mathcal{L}_{\mathrm{RE}}$. Wir haben also eine TM M für L und eine TM M^{C} für L^{C}.
Wir erhalten einen Algorithmus A für L, der folgendermassen arbeitet:
Die Eingabe wird parallel auf M und M^{C} simuliert.
Falls M akzeptiert, so beendet A die Simulation und akzeptiert die Eingabe.
Falls M^{C} akzeptiert, so beendet A die Simulation und verwirft die Eingabe.

Somit gilt $L \in \mathcal{L}_{\mathrm{R}}$, also $L \notin \mathcal{L}_{\mathrm{RE}} \setminus \mathcal{L}_{\mathrm{R}}$.

Aus $L \leq_{\mathsf{EE}} L^{\mathsf{C}}$ oder $L^{\mathsf{C}} \leq_{\mathsf{EE}} L$ folgt $L, L^{\mathsf{C}} \notin \mathcal{L}_{\mathsf{RE}} \setminus \mathcal{L}_{\mathsf{R}}$.

Die beiden Voraussetzungen sind wie eben gezeigt äquivalent. Sei also $L =_{\mathrm{EE}} L^{\mathsf{C}}$.
Daraus folgt $L =_{\mathrm{R}} L^{\mathsf{C}}$ und $L =_{\mathrm{RE}} L^{\mathsf{C}}$.
Somit können L und L^{C} nur gemeinsam $\mathcal{L}_{\mathrm{RE}} \setminus \mathcal{L}_{\mathrm{R}}$ liegen. Dies ist gemäss dem letzten Resultat aber unmöglich.

Nichtexistenz von EE-Reduktionen zwischen universeller, Halte- und Diagonalsprache sowie Komplementen.

Wir wissen, dass $L_{\mathrm{diag}}^{\mathsf{C}} \in \mathcal{L}_{\mathrm{RE}} \setminus \mathcal{L}_{\mathrm{R}}$. Das letzte Resulat besagt, dass es keine EE-Reduktion auf oder vom Komplement gibt: Es gilt weder $L_{\mathrm{diag}}^{\mathsf{C}} \leq_{\mathrm{EE}} L_{\mathrm{diag}}$ noch $L_{\mathrm{diag}} \leq_{\mathrm{EE}} L_{\mathrm{diag}}^{\mathsf{C}}$. Formal kann man dies als $L_{\mathrm{diag}}^{\mathsf{C}} >_{\mathrm{EE}} L_{\mathrm{diag}}$ und $L_{\mathrm{diag}}^{\mathsf{C}} <_{\mathrm{EE}} L_{\mathrm{diag}}$ schreiben.

Mit der Transitivität von $\leq_{\mathrm{EE}}$ folgt, dass keine der drei Sprachen L_{U}, L_{H} und $L_{\mathrm{diag}}^{\complement}$ auf eines der drei Komplemente EE-reduziert werden kann. Formal ist das Argument kurz:

$$L_{\mathrm{U}} =_{\mathrm{EE}} L_{\mathrm{H}} =_{\mathrm{EE}} L_{\mathrm{diag}}^{\complement} <_{\mathrm{EE}} L_{\mathrm{diag}} =_{\mathrm{EE}} L_{\mathrm{U}}^{\complement} =_{\mathrm{EE}} L_{\mathrm{H}}^{\complement}$$
$$L_{\mathrm{U}} =_{\mathrm{EE}} L_{\mathrm{H}} =_{\mathrm{EE}} L_{\mathrm{diag}}^{\complement} >_{\mathrm{EE}} L_{\mathrm{diag}} =_{\mathrm{EE}} L_{\mathrm{U}}^{\complement} =_{\mathrm{EE}} L_{\mathrm{H}}^{\complement}$$

(Die verwendeten Gleichungen werden unten bewiesen.)
Dies zeigt für insgesamt $2 \cdot 3^2 = 18$ Paare (L_1, L_2) mit

$$L_1, L_2 \in \{L_{\mathrm{diag}}^{\complement}, L_{\mathrm{U}}, L_{\mathrm{H}}, L_{\mathrm{diag}}, L_{\mathrm{U}}^{\complement}, L_{\mathrm{H}}^{\complement}\},$$

dass keine EE-Reduktion existieren kann: $L_1 \not\leq_{\mathrm{EE}} L_2$.

Existierende EE-Reduktionen zwischen universeller, Halte- und Diagonalsprache sowie Komplementen.

Für die anderen 18 von den total $6^2 = 36$ Paaren (L_1, L_2) mit

$$L_1, L_2 \in \{L_{\mathrm{diag}}^{\complement}, L_{\mathrm{U}}, L_{\mathrm{H}}, L_{\mathrm{diag}}, L_{\mathrm{U}}^{\complement}, L_{\mathrm{H}}^{\complement}\},$$

gilt hingegen $L_1 \leq_{\mathrm{EE}} L_2$. Es handelt sich um die in untenstehender Tabelle angegebenen Reduktionen. Die letzten 6 sind trivial, die anderen 12 sind auf den nachfolgenden Seiten explizit ausgeführt. In der linken Spalte stehen die 6 Reduktionen, die $L_{\mathrm{diag}}^{\complement} =_{\mathrm{EE}} L_{\mathrm{U}} =_{\mathrm{EE}} L_{\mathrm{H}}$ beweisen, in der rechten Spalte jene, die $L_{\mathrm{diag}} =_{\mathrm{EE}} L_{\mathrm{U}}^{\complement} =_{\mathrm{EE}} L_{\mathrm{H}}^{\complement}$ beweisen.

(1)	$L_{\mathrm{diag}}^{\complement} \leq_{\mathrm{EE}} L_{\mathrm{U}}$	(2)	$L_{\mathrm{diag}} \leq_{\mathrm{EE}} L_{\mathrm{U}}^{\complement}$
(3)	$L_{\mathrm{diag}}^{\complement} \leq_{\mathrm{EE}} L_{\mathrm{H}}$	(4)	$L_{\mathrm{diag}} \leq_{\mathrm{EE}} L_{\mathrm{H}}^{\complement}$
(5)	$L_{\mathrm{U}} \leq_{\mathrm{EE}} L_{\mathrm{diag}}^{\complement}$	(6)	$L_{\mathrm{U}}^{\complement} \leq_{\mathrm{EE}} L_{\mathrm{diag}}$
(7)	$L_{\mathrm{H}} \leq_{\mathrm{EE}} L_{\mathrm{diag}}^{\complement}$	(8)	$L_{\mathrm{H}}^{\complement} \leq_{\mathrm{EE}} L_{\mathrm{diag}}$
(9)	$L_{\mathrm{H}} \leq_{\mathrm{EE}} L_{\mathrm{U}}$	(10)	$L_{\mathrm{H}}^{\complement} \leq_{\mathrm{EE}} L_{\mathrm{U}}^{\complement}$
(11)	$L_{\mathrm{U}} \leq_{\mathrm{EE}} L_{\mathrm{H}}$	(12)	$L_{\mathrm{U}}^{\complement} \leq_{\mathrm{EE}} L_{\mathrm{H}}^{\complement}$
(13)	$L_{\mathrm{diag}} \leq_{\mathrm{EE}} L_{\mathrm{diag}}$	(14)	$L_{\mathrm{diag}}^{\complement} \leq_{\mathrm{EE}} L_{\mathrm{diag}}^{\complement}$
(15)	$L_{\mathrm{U}} \leq_{\mathrm{EE}} L_{\mathrm{U}}$	(16)	$L_{\mathrm{U}}^{\complement} \leq_{\mathrm{EE}} L_{\mathrm{U}}^{\complement}$
(17)	$L_{\mathrm{H}} \leq_{\mathrm{EE}} L_{\mathrm{H}}$	(18)	$L_{\mathrm{H}}^{\complement} \leq_{\mathrm{EE}} L_{\mathrm{H}}^{\complement}$

Die eine Spalte ergibt sich aus der anderen durch die Abgeschlossenheit bezüglich Komplementbildung, also

$$L_1 \leq_{\mathrm{EE}} L_2 \iff L_1^{\complement} \leq_{\mathrm{EE}} L_2^{\complement}$$

Entsprechend sind die Reduktionen einer Zeile auch identisch ausgeführt, nur die Notation ändert sich minimal.

Wie schon beschrieben erhält man aus EE-Reduzierbarkeit sofort auch R-Reduzierbarkeit. Durch Anwendung des nachfolgenden Satzes erhält man daraus wiederum eine R-Reduktionen zu den 18 Paaren, die erwiesenermassen keine EE-Reduktion zulassen:

(19)	$L_{\text{diag}} \leq_{\text{R}} L_{\text{U}}$	(20)	$L_{\text{diag}}^{\complement} \leq_{\text{R}} L_{\text{U}}^{\complement}$
(21)	$L_{\text{diag}} \leq_{\text{R}} L_{\text{H}}$	(22)	$L_{\text{diag}}^{\complement} \leq_{\text{R}} L_{\text{H}}^{\complement}$
(23)	$L_{\text{U}}^{\complement} \leq_{\text{R}} L_{\text{diag}}^{\complement}$	(24)	$L_{\text{U}} \leq_{\text{R}} L_{\text{diag}}$
(25)	$L_{\text{H}}^{\complement} \leq_{\text{R}} L_{\text{diag}}^{\complement}$	(26)	$L_{\text{H}} \leq_{\text{R}} L_{\text{diag}}$
(27)	$L_{\text{H}}^{\complement} \leq_{\text{R}} L_{\text{U}}$	(28)	$L_{\text{H}} \leq_{\text{R}} L_{\text{U}}^{\complement}$
(29)	$L_{\text{U}}^{\complement} \leq_{\text{R}} L_{\text{H}}$	(30)	$L_{\text{U}} \leq_{\text{R}} L_{\text{H}}^{\complement}$
(31)	$L_{\text{diag}}^{\complement} \leq_{\text{R}} L_{\text{diag}}$	(32)	$L_{\text{diag}} \leq_{\text{R}} L_{\text{diag}}^{\complement}$
(33)	$L_{\text{U}}^{\complement} \leq_{\text{R}} L_{\text{U}}$	(34)	$L_{\text{U}} \leq_{\text{R}} L_{\text{U}}^{\complement}$
(35)	$L_{\text{H}}^{\complement} \leq_{\text{R}} L_{\text{H}}$	(36)	$L_{\text{H}} \leq_{\text{R}} L_{\text{H}}^{\complement}$

$L_1 \leq_{\text{R}} L_2 \implies L_1^{\complement} \leq_{\text{R}} L_2$.

Wir nehmen $L_1 \leq_{\text{R}} L_2$ an. Es existiert also ein Algorithmus A_1 für L_1, sofern ein Algorithmus A_2 für L_2 existiert.
Wenn wir in diesem Algorithmus die beiden möglichen Ausgaben vertauschen, haben wir einen Algorithmus $A_1^{\complement}$ für $L_1^{\complement}$, sofern ein Algorithmus A_2 für L_2 existiert. Dies ist die Definition von $L_1^{\complement} \leq_{\text{R}} L_2$.

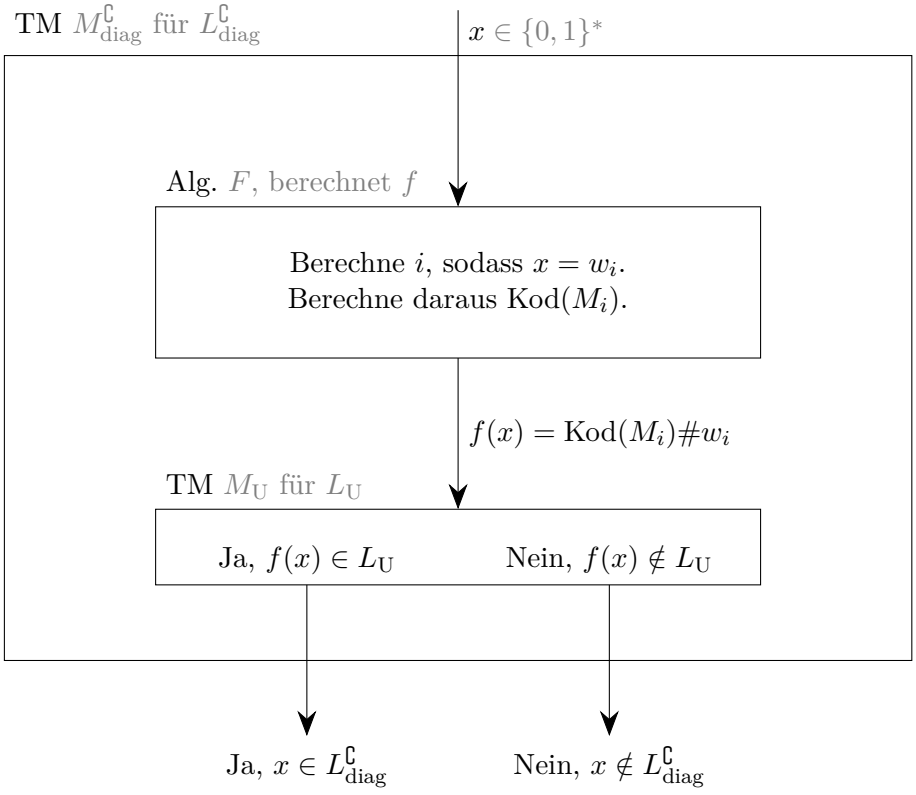
Behauptung
$L^{\complement}_{\text{diag}} \leq_{\text{EE}} L_{\text{U}}$

Beweis
Sei w_i das i-te Wort über $\{0,1\}$ und M_i die i-te TM (kanon. Ordnung).
Es gilt

$$L^{\complement}_{\text{diag}} \coloneqq \{w_i \in \{0,1\}^* \mid M_i \text{ akzeptiert } w_i\}$$

und

$$L_{\text{U}} \coloneqq \{\text{Kod}(M)\#w \in \{0,1,\#\}^* \mid M \text{ akzeptiert } w_i.\} \ .$$



Korrektheitsnachweis:

$$x \in L^{\complement}_{\text{diag}} \iff M_i \text{ akzeptiert } x = w_i. \iff f(x) = \text{Kod}(M_i)\#w_i \in L_{\text{U}}$$

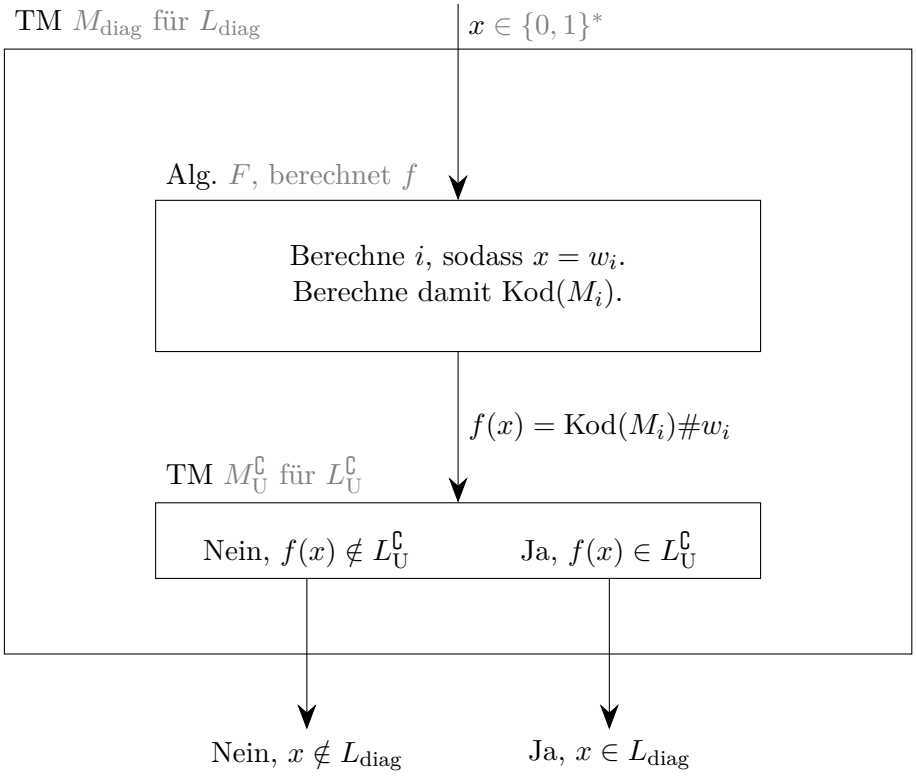
Behauptung
$L_{\text{diag}} \leq_{\text{EE}} L_{\text{U}}^{\complement}$.

Beweis
Sei w_i das i-te Wort über $\{0,1\}$ und M_i die i-te TM (kanon. Ordnung).
Es gilt

$$L_{\text{diag}} \coloneqq \{w_i \in \{0,1\}^* \mid M_i \text{ akzeptiert } w_i \text{ nicht.}\}$$

und

$$L_{\text{U}}^{\complement} \coloneqq \{\text{Kod}(M)\#w \in \{0,1,\#\}^* \mid M \text{ akzeptiert } w_i \text{ nicht.}\} \cup \{x \in \{0,1,\#\}^* \mid x \text{ nicht von Form Kod}(M)\#w.\} .$$



Korrektheitsnachweis:

$$x \in L_{\text{diag}} \iff M_i \text{ akzeptiert } x = w_i \text{ nicht.} \iff f(x) = \text{Kod}(M_i)\#w_i \in L_{\text{U}}^{\complement}$$

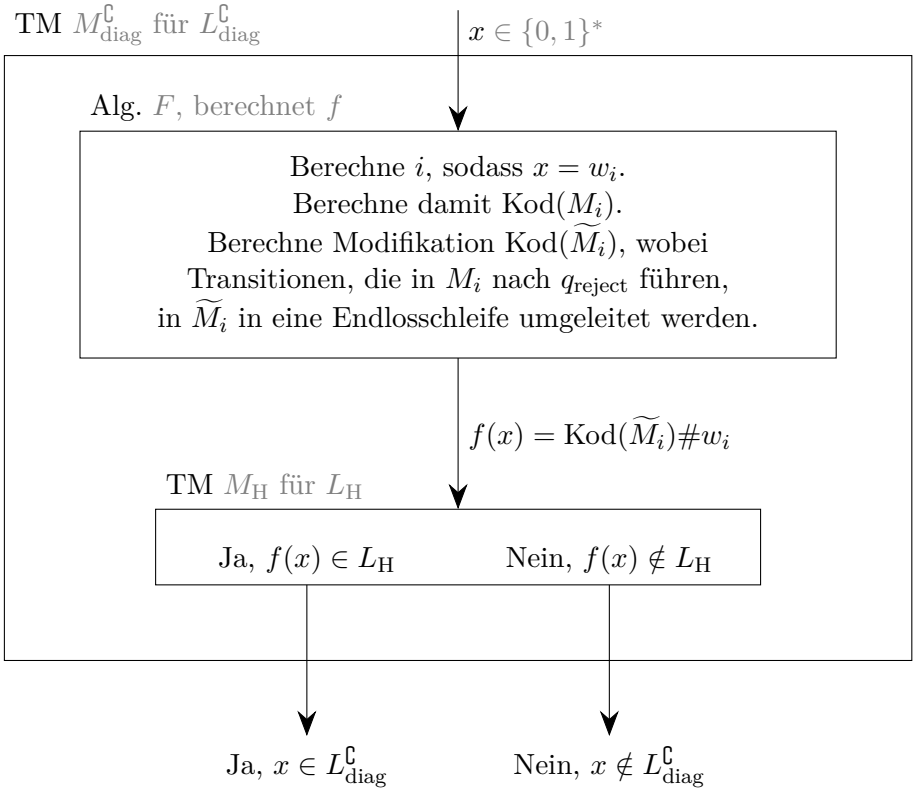
Behauptung
$L^{\complement}_{\text{diag}} \leq_{\text{EE}} L_{\text{H}}$

Beweis
Sei w_i das i-te Wort über $\{0,1\}$ und M_i die i-te TM (kanon. Ordnung).
Es gilt

$$L^{\complement}_{\text{diag}} \coloneqq \{w_i \in \{0,1\}^* \mid M_i \text{ akzeptiert } w_i.\}$$

und

$$L_{\text{H}} \coloneqq \{\text{Kod}(M)\#w \in \{0,1,\#\}^* \mid M \text{ hält auf } w.\} \ .$$



Korrektheitsnachweis:

$$\begin{aligned} x \in L^{\complement}_{\text{diag}} &\iff M_i \text{ akzeptiert } x = w_i. \\ &\iff \widetilde{M}_i \text{ hält auf } x = w_i. \\ &\iff f(x) = \text{Kod}(\widetilde{M}_i)\#w_i \in L_{\text{H}} \end{aligned}$$

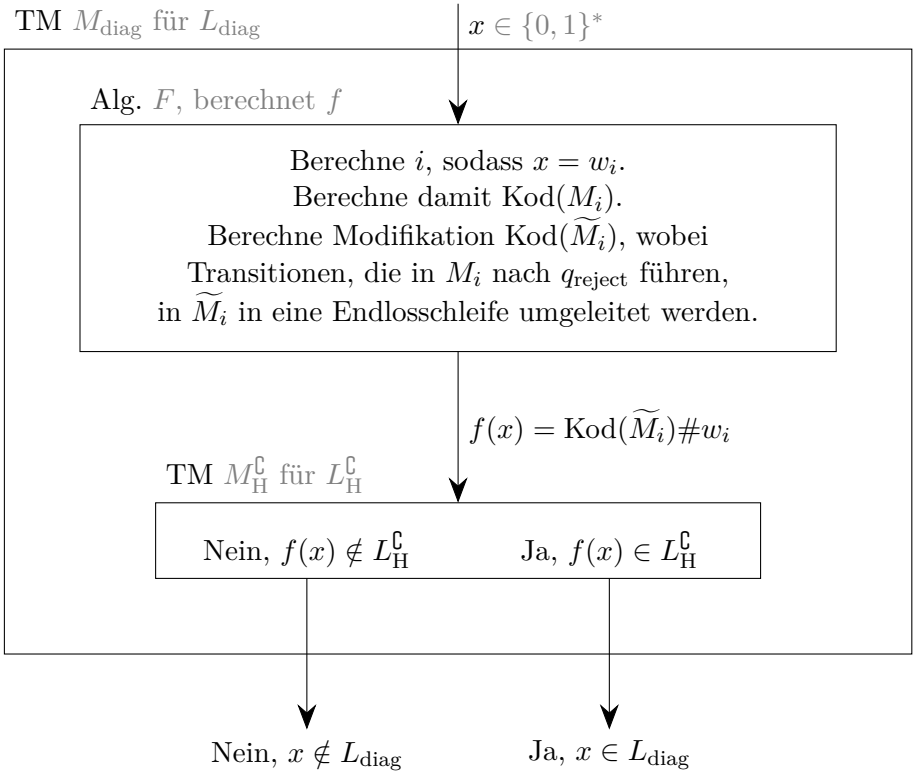
Behauptung
$L_{\text{diag}} \leq_{\text{EE}} L_{\text{H}}^{\complement}$

Beweis
Sei w_i das i-te Wort über $\{0,1\}$ und M_i die i-te TM (kanon. Ordnung).
Es gilt

$$L_{\text{diag}} \coloneqq \{w_i \in \{0,1\}^* \mid M_i \text{ akzeptiert } w_i \text{ nicht.}\}$$

und

$$\begin{aligned} L_{\text{H}}^{\complement} \coloneqq &\{\text{Kod}(M)\#w \in \{0,1,\#\}^* \mid M \text{ hält nicht auf } w.\} \\ &\cup \{x \in \{0,1,\#\}^* \mid x \text{ nicht von Form Kod}(M)\#w\} \ . \end{aligned}$$



Korrektheitsnachweis:

$$\begin{aligned} x \in L_{\text{diag}} &\iff M_i \text{ akzeptiert } x = w_i \text{ nicht.} \\ &\iff \widetilde{M}_i \text{ hält nicht auf } x = w_i. \\ &\iff f(x) = \text{Kod}(\widetilde{M}_i)\#w_i \in L_{\text{H}}^{\complement} \end{aligned}$$

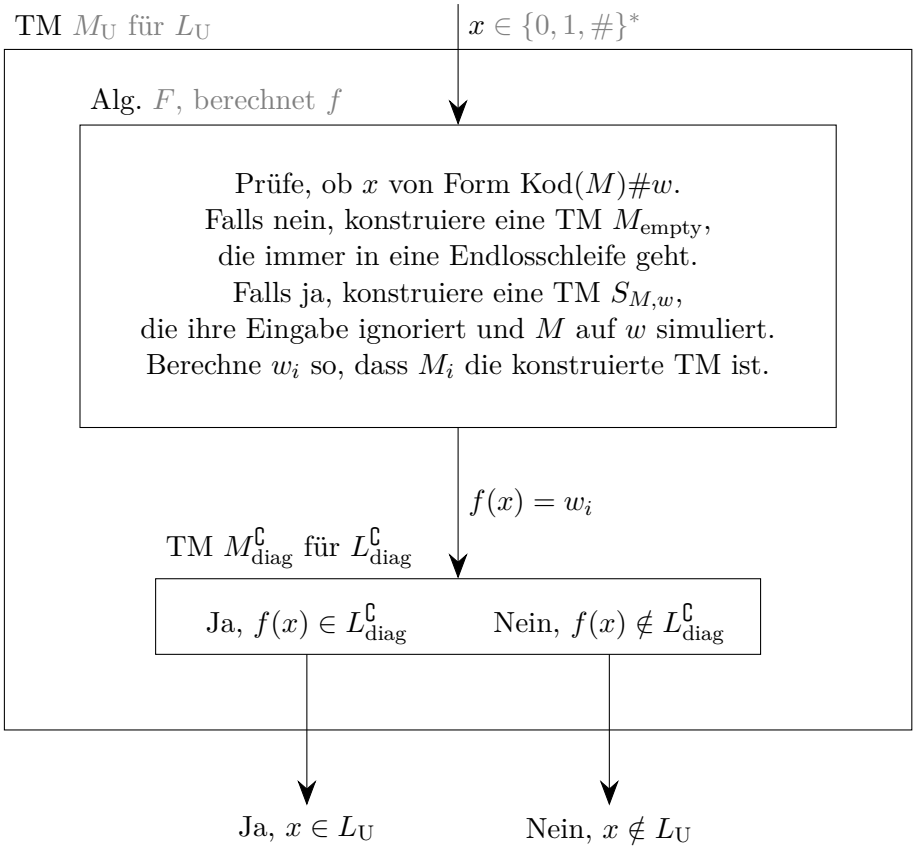
Behauptung
$L_{\mathrm{U}} \leq_{\mathrm{EE}} L^{\complement}_{\mathrm{diag}}$

Beweis
Sei w_i das i-te Wort über $\{0,1\}$ und M_i die i-te TM (kanon. Ordnung).
Es gilt

$$L^{\complement}_{\mathrm{diag}} \coloneqq \{w_i \in \{0,1\}^* \mid M_i \text{ akzeptiert } w_i\}$$

und

$$L_{\mathrm{U}} \coloneqq \{\mathrm{Kod}(M)\#w \in \{0,1,\#\}^* \mid M \text{ akzeptiert } w.\} \ .$$



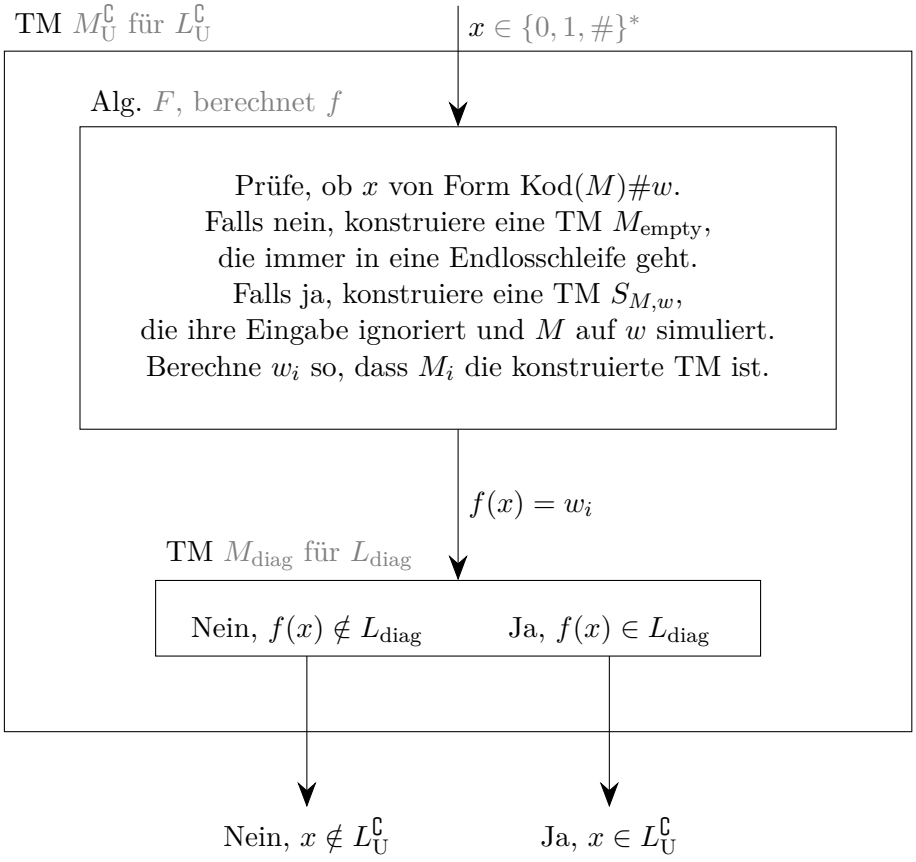
Korrektheitsnachweis:
Falls x nicht von Form $\mathrm{Kod}(M)\#w$, gilt $x \notin L_{\mathrm{U}}$ und
$M_i = M_{\mathrm{empty}}$ hält nie $\implies$ M_i akzeptiert w_i nicht. $\implies w_i \notin L^{\complement}_{\mathrm{diag}}$.

Sonst:
$$\begin{aligned} x \in L_{\mathrm{U}} &\iff M \text{ akzeptiert } w. \\ &\iff S_{M,w} = M_i \text{ akzeptiert alles.} \\ &\iff S_{M,w} = M_i \text{ akzeptiert } w_i. \\ &\iff f(x) = w_i \in L^{\complement}_{\mathrm{diag}} \end{aligned}$$

Behauptung
$L_{\mathrm{U}}^{\complement} \leq_{\mathrm{EE}} L_{\mathrm{diag}}$

Beweis
Sei w_i das i-te Wort über $\{0,1\}$ und M_i die i-te TM (kanon. Ordnung).
Es gilt:

$$L_{\mathrm{diag}} \coloneqq \{w_i \in \{0,1\}^* \mid M_i \text{ akzeptiert } w_i \text{ nicht.}\}$$
$$L_{\mathrm{U}}^{\complement} \coloneqq \{\mathrm{Kod}(M)\#w \in \{0,1,\#\}^* \mid M \text{ akzeptiert } w \text{ nicht.}\}$$
$$\cup \{x \in \{0,1,\#\}^* \mid x \text{ nicht von Form } \mathrm{Kod}(M)\#w.\}$$



Korrektheitsnachweis:
Falls x nicht von Form $\mathrm{Kod}(M)\#w$, gilt $x \in L_{\mathrm{U}}^{\complement}$ und
$M_i = M_{\mathrm{empty}}$ hält nie $\implies$ M_i akzeptiert w_i nicht. $\implies w_i \in L_{\mathrm{diag}}$.

Sonst:
$$\begin{aligned} x \notin L_{\mathrm{U}}^{\complement} &\iff M \text{ akzeptiert } w. \\ &\iff S_{M,w} = M_i \text{ akzeptiert alles.} \\ &\iff S_{M,w} = M_i \text{ akzeptiert } w_i. \\ &\iff f(x) = w_i \notin L_{\mathrm{diag}} \end{aligned}$$

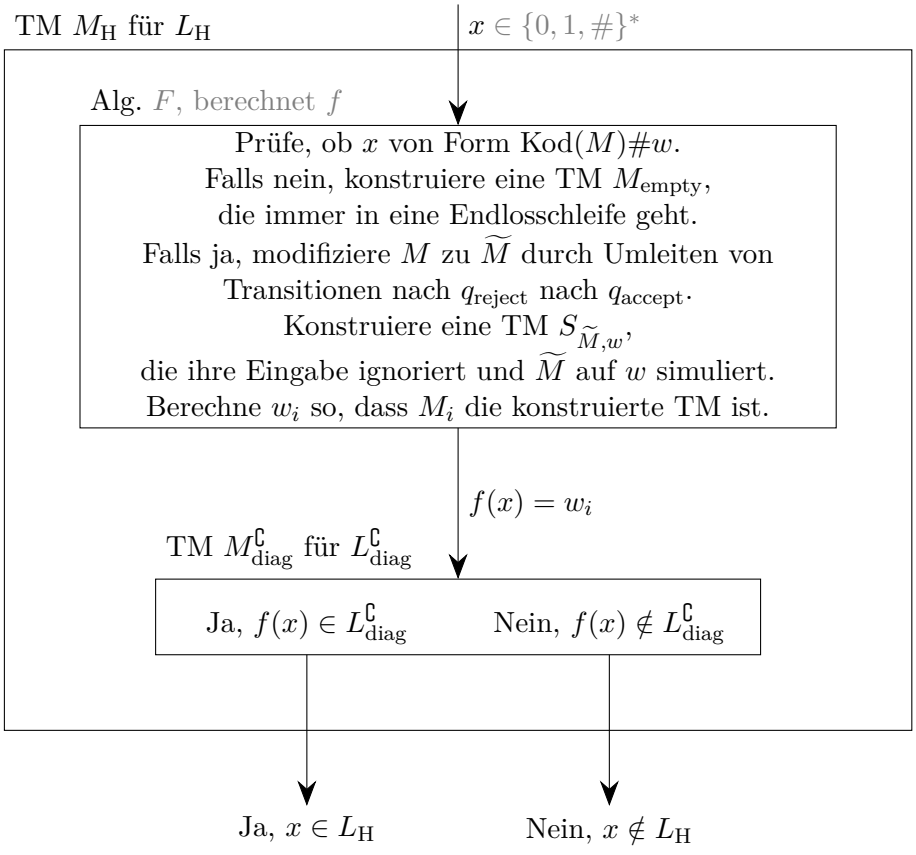
Behauptung
$L_{\mathrm{H}} \leq_{\mathrm{EE}} L^{\complement}_{\mathrm{diag}}$

Beweis
Sei w_i das i-te Wort über $\{0,1\}$ und M_i die i-te TM (kanon. Ordnung).
Es gilt

$$L^{\complement}_{\mathrm{diag}} \coloneqq \{w_i \in \{0,1\}^* \mid M_i \text{ akzeptiert } w_i\}$$

und

$$L_{\mathrm{H}} \coloneqq \{\mathrm{Kod}(M)\#w \in \{0,1,\#\}^* \mid M \text{ hält auf } w.\} \ .$$



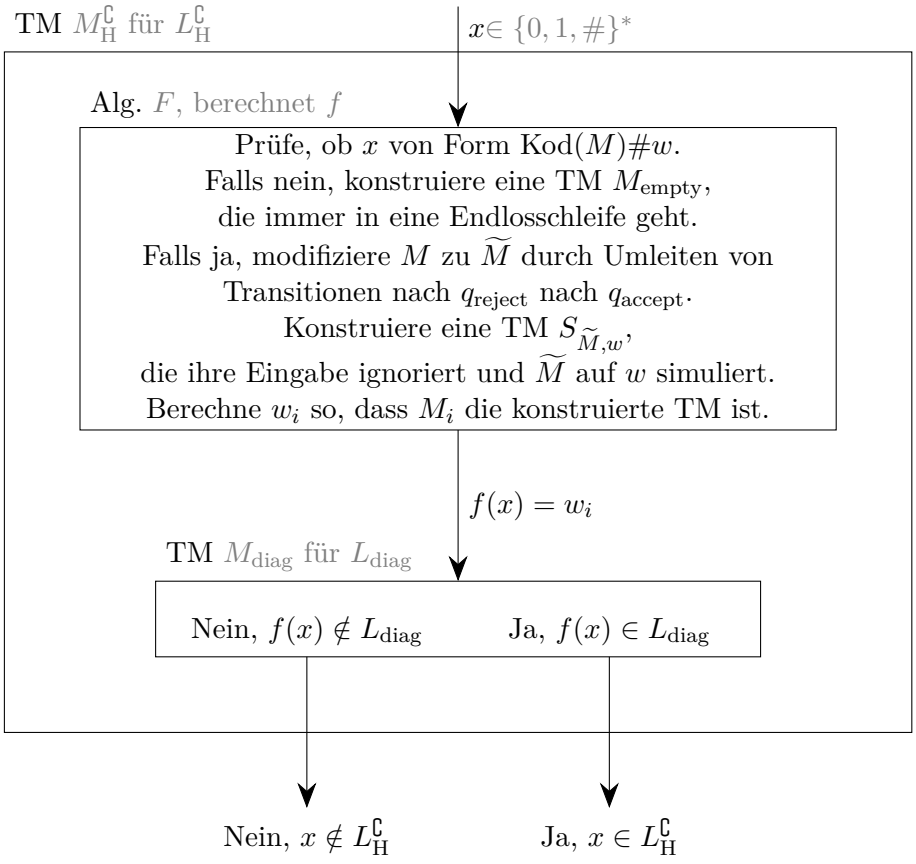
Korrektheitsnachweis:
Falls x nicht von Form $\mathrm{Kod}(M)\#w$, gilt $x \notin L_{\mathrm{H}}$ und
$M_i = M_{\mathrm{empty}}$ hält nie $\implies$ M_i akzeptiert w_i nicht. $\implies w_i \notin L^{\complement}_{\mathrm{diag}}$.

Sonst: $x \in L_{\mathrm{H}} \iff M$ hält auf $w. \iff \widetilde{M}$ akzeptiert $w.$
$\iff S_{\widetilde{M},w} = M_i$ akzeptiert alles.
$\iff S_{\widetilde{M},w} = M_i$ akzeptiert w_i.
$\iff f(x) = w_i \in L^{\complement}_{\mathrm{diag}}$

Behauptung
$L_{\mathrm{H}}^{\complement} \leq_{\mathrm{EE}} L_{\mathrm{diag}}$

Beweis
Sei w_i das i-te Wort über $\{0,1\}$ und M_i die i-te TM (kanon. Ordnung).

$$L_{\mathrm{diag}} \coloneqq \{w_i \in \{0,1\}^* \mid M_i \text{ akzeptiert } w_i \text{ nicht.}\}$$
$$L_{\mathrm{H}}^{\complement} \coloneqq \{\mathrm{Kod}(M)\#w \in \{0,1,\#\}^* \mid M \text{ hält nicht auf } w.\}$$
$$\cup \{x \in \{0,1,\#\}^* \mid x \text{ nicht von Form } \mathrm{Kod}(M)\#w.\}\ .$$



Korrektheitsnachweis:
Falls x nicht von Form $\mathrm{Kod}(M)\#w$, gilt $x \in L_{\mathrm{H}}^{\complement}$ und
$M_i = M_{\mathrm{empty}}$ hält nie $\implies$ M_i akzeptiert w_i nicht. $\implies w_i \in L_{\mathrm{diag}}$.

Sonst: $x \notin L_{\mathrm{H}}^{\complement} \iff M$ hält auf $w. \iff \widetilde{M}$ akzeptiert $w.$
$\iff S_{\widetilde{M},w} = M_i$ akzeptiert alles.
$\iff S_{\widetilde{M},w} = M_i$ akzeptiert w_i.
$\iff f(x) = w_i \notin L_{\mathrm{diag}}$

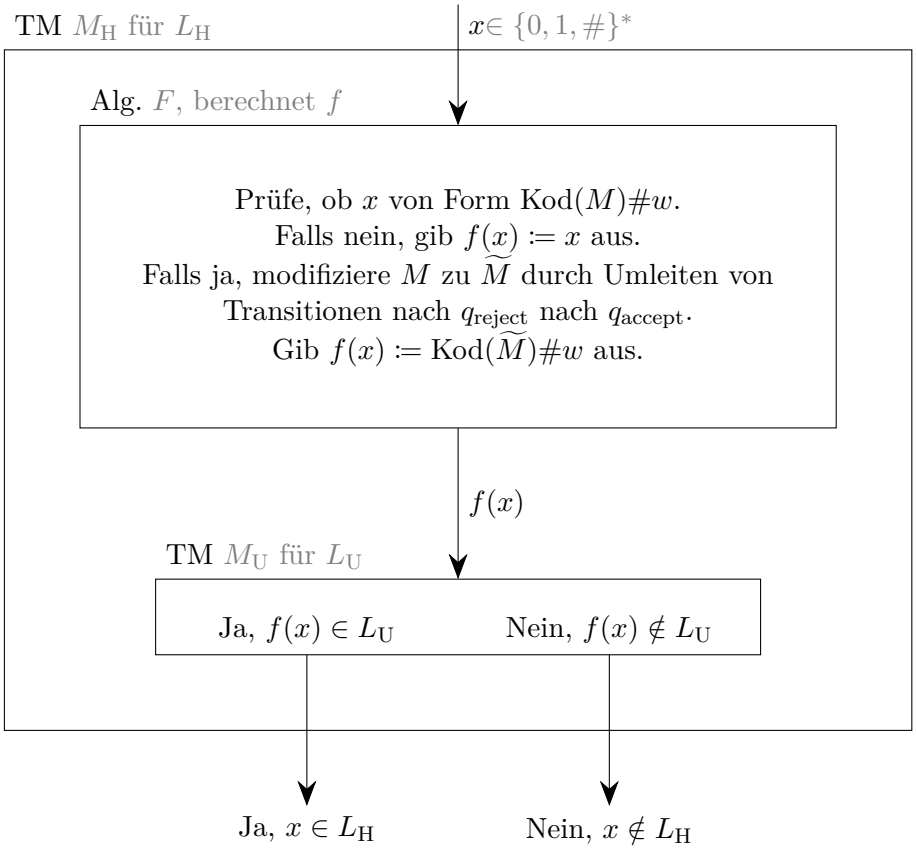
Behauptung
$L_{\mathrm{H}} \leq_{\mathrm{EE}} L_{\mathrm{U}}$

Beweis
Es gilt

$$L_{\mathrm{U}} \coloneqq \{\mathrm{Kod}(M)\#w \in \{0,1,\#\}^* \mid M \text{ akzeptiert } w.\}$$

und

$$L_{\mathrm{H}} \coloneqq \{\mathrm{Kod}(M)\#w \in \{0,1,\#\}^* \mid M \text{ hält auf } w.\} \ .$$



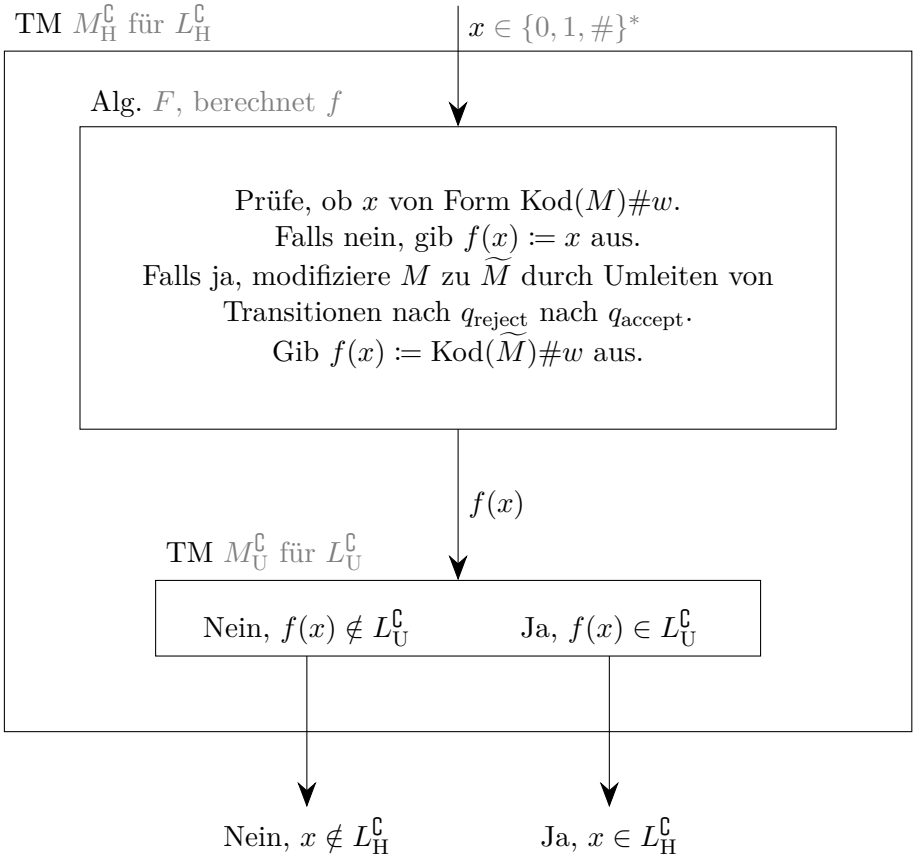
Korrektheitsnachweis:
Falls x nicht von Form $\mathrm{Kod}(M)\#w$, gilt $x \notin L_{\mathrm{H}}$ und $f(x) = x \notin L_{\mathrm{U}}$.

Sonst:
$$\begin{aligned} & x = \mathrm{Kod}(M)\#w \in L_{\mathrm{H}} \\ \iff & M \text{ hält auf } w. \\ \iff & \widetilde{M} \text{ akzeptiert } w. \\ \iff & f(x) = \mathrm{Kod}(\widetilde{M})\#w \in L_{\mathrm{U}} \end{aligned}$$

Behauptung
$L_{\mathrm{H}}^{\complement} \leq_{\mathrm{EE}} L_{\mathrm{U}}^{\complement}$

Beweis
Es gilt

$$\begin{aligned} L_{\mathrm{U}}^{\complement} &\coloneqq \{\mathrm{Kod}(M)\#w \in \{0,1,\#\}^* \mid M \text{ akzeptiert } w_i \text{ nicht.}\} \\ &\quad \cup \{x \in \{0,1,\#\}^* \mid x \text{ nicht von Form } \mathrm{Kod}(M)\#w.\} \\ L_{\mathrm{H}}^{\complement} &\coloneqq \{\mathrm{Kod}(M)\#w \in \{0,1,\#\}^* \mid M \text{ hält nicht auf } w_i.\} \\ &\quad \cup \{x \in \{0,1,\#\}^* \mid x \text{ nicht von Form } \mathrm{Kod}(M)\#w.\} \ . \end{aligned}$$



Korrektheitsnachweis:
Falls x nicht von Form $\mathrm{Kod}(M)\#w$, gilt $x \in L_{\mathrm{H}}^{\complement}$ und $x \in L_{\mathrm{U}}^{\complement}$.

$$\begin{aligned} \text{Sonst:} \qquad & x = \mathrm{Kod}(M)\#w \notin L_{\mathrm{H}}^{\complement} \\ \iff & M \text{ hält auf } w. \\ \iff & \widetilde{M} \text{ akzeptiert } w. \\ \iff & f(x) = \mathrm{Kod}(\widetilde{M})\#w \notin L_{\mathrm{U}}^{\complement} \end{aligned}$$

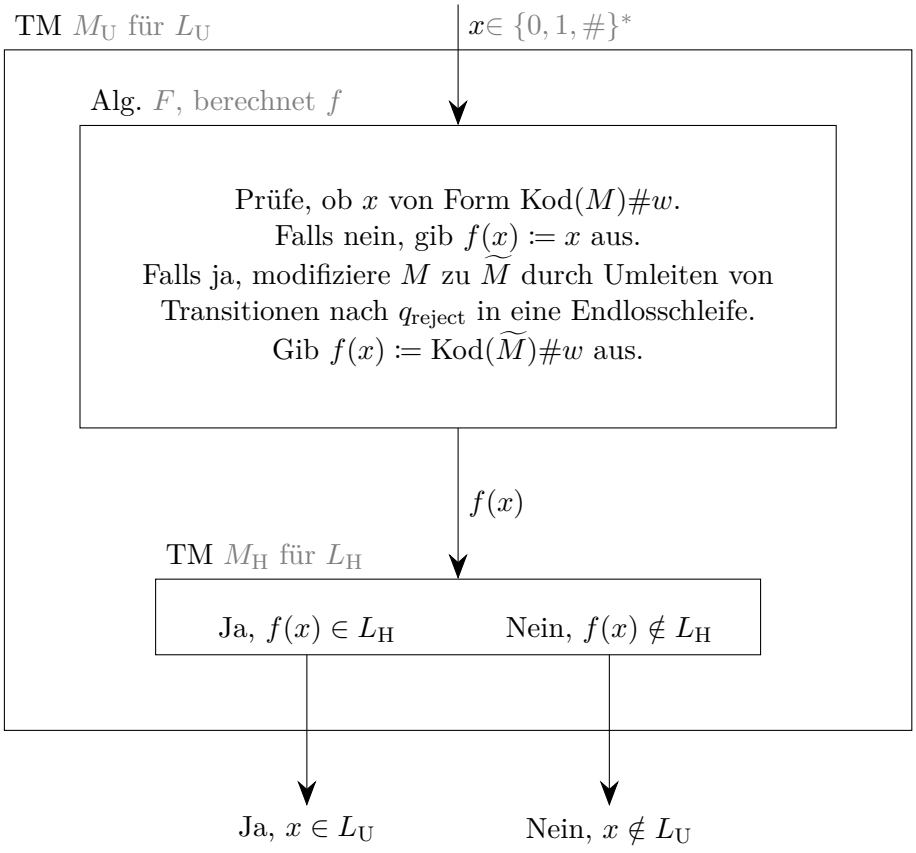
Behauptung
$L_{\mathrm{U}} \leq_{\mathrm{EE}} L_{\mathrm{H}}$

Beweis
Es gilt

$$L_{\mathrm{U}} \coloneqq \{\mathrm{Kod}(M)\#w \in \{0,1,\#\}^* \mid M \text{ akzeptiert } w.\}$$

und

$$L_{\mathrm{H}} \coloneqq \{\mathrm{Kod}(M)\#w \in \{0,1,\#\}^* \mid M \text{ hält auf } w.\} \ .$$



Korrektheitsnachweis:
Falls x nicht von Form $\mathrm{Kod}(M)\#w$, gilt $x \notin L_{\mathrm{U}}$ und $f(x) = x \notin L_{\mathrm{H}}$.

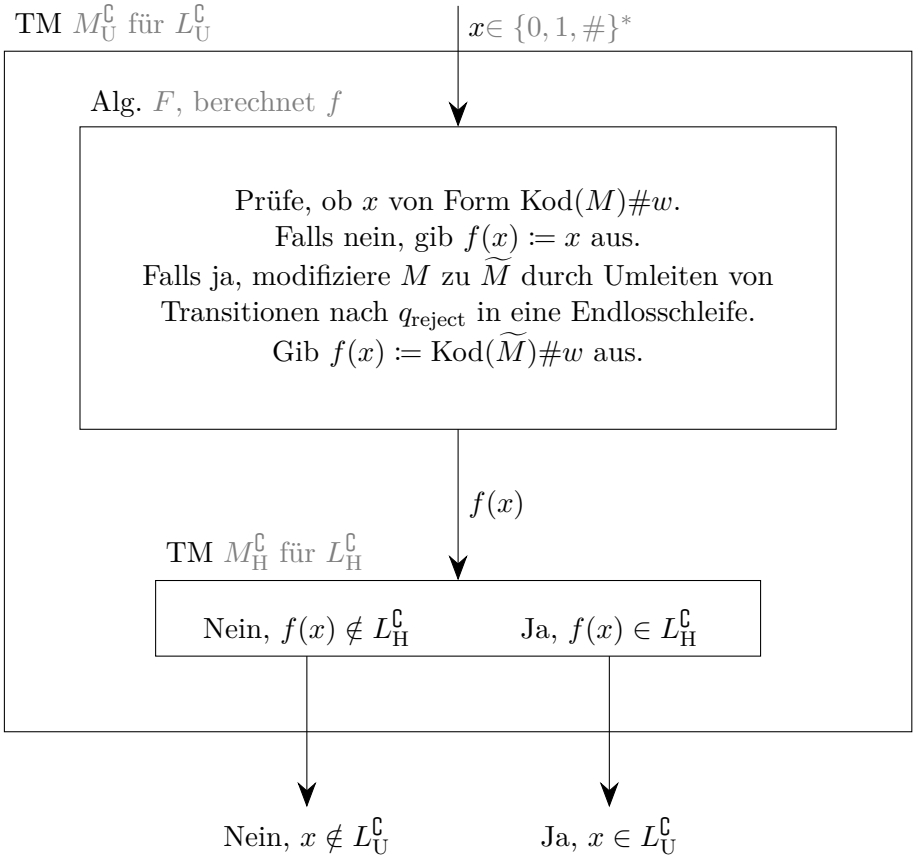
Sonst:

$$\begin{aligned} & x = \mathrm{Kod}(M)\#w \in \ L_{\mathrm{U}} \\ \iff & M \text{ akzeptiert } w. \\ \iff & \widetilde{M} \text{ hält auf } w. \\ \iff & f(x) = \mathrm{Kod}(\widetilde{M})\#w \in L_{\mathrm{H}} \end{aligned}$$

Behauptung
$L_{\mathrm{U}}^{\complement} \leq_{\mathrm{EE}} L_{\mathrm{H}}^{\complement}$

Beweis
Es gilt

$$\begin{aligned} L_{\mathrm{U}}^{\complement} \coloneqq &\{\mathrm{Kod}(M)\#w \in \{0,1,\#\}^* \mid M \text{ akzeptiert } w_i \text{ nicht.}\} \\ &\cup \{x \in \{0,1,\#\}^* \mid x \text{ nicht von Form } \mathrm{Kod}(M)\#w.\} \\ L_{\mathrm{H}}^{\complement} \coloneqq &\{\mathrm{Kod}(M)\#w \in \{0,1,\#\}^* \mid M \text{ hält nicht auf } w_i.\} \\ &\cup \{x \in \{0,1,\#\}^* \mid x \text{ nicht von Form } \mathrm{Kod}(M)\#w.\} \ . \end{aligned}$$



Korrektheitsnachweis:
Falls x nicht von Form $\mathrm{Kod}(M)\#w$, gilt $x \in L_{\mathrm{U}}^{\complement}$ und $x \in L_{\mathrm{H}}^{\complement}$.

$$\begin{aligned} \text{Sonst:} \qquad & x = \mathrm{Kod}(M)\#w \notin L_{\mathrm{H}}^{\complement} \\ \iff & M \text{ akzeptiert } w. \\ \iff & \widetilde{M} \text{ hält auf } w. \\ \iff & f(x) = \mathrm{Kod}(\widetilde{M})\#w \notin L_{\mathrm{U}}^{\complement} \end{aligned}$$

Rekursivitätsaufgaben in Zwischen- und Endklausuren (ZK und EK)

1. $L_{\text{empty}}^{\complement} \in \mathcal{L}_{\text{RE}}$ (1. ZK, HS14, A3a)
2. $L_{\text{U}} \leq_{\text{EE}} L_{\text{empty}}^{\complement}$ (1. ZK, HS14, A3b)
3. $L_{\text{U}} \leq_{\text{EE}} L_{\text{H}}$ (1. ZK, HS15, A5)
4. $L_{\text{U}} \leq_{\text{R}} L_{\text{H}}^{\complement}$ (1. ZK, HS17, A4)
5. $L_{\cap=\emptyset} \notin \mathcal{L}_{\text{R}}$ (2. ZK, HS17, A1a)
6. $L_{\text{empty}}^{\complement} in \mathcal{L}_{\text{RE}}$ (2. ZK, HS17, A1b; EK, HS09, 5a)
7. $L_{\text{U}} \leq_{\text{R}} L_{\text{H},\lambda}$ (2. ZK, HS17, A2)
8. $L_{\text{empty}}^{\complement} \in \mathcal{L}_{\text{RE}}$ (2. ZK, HS16, A1a; 2. ZK, HS10, A3a; EK, HS13, 5a)
9. $L_{\text{nohalt}} \notin \mathcal{L}_{\text{R}}$ (2. ZK, HS16, A1b)
10. $L_{\cap\neq\emptyset} \in \mathcal{L}_{\text{RE}}$ (2. ZK, HS16, A2a)
11. $L_{\text{diag}} \leq_{\text{R}} L_{\cap\neq\emptyset}$ (2. ZK, HS16, A2b)
12. $L_{\text{H},\lambda} \leq_{\text{EE}} L_{001}$ (2. ZK, HS16, BA5a)
13. $L_{\text{all}}^{\complement} \notin \mathcal{L}_{\text{RE}}$ (2. ZK, HS16, BA5b)
14. $L_{\text{U},\lambda} \leq_{\text{EE}} L_{\text{all}}$ (2. ZK, HS15, A2a)
15. $L_{\text{H},\lambda} \leq_{\text{R}} L_{\text{U},\lambda}$ (2. ZK, HS15, A2b)
16. $L_{\text{all}}^{\complement} \notin \mathcal{L}_{\text{RE}}$ (2. ZK, HS15, BA2c)
17. $L_{0,1\in} \notin \mathcal{L}_{\text{R}}$, mit EE-Reduktion, ohne Satz von Rice. (2. ZK, HS15, A3)
18. $L_{011\in} \leq_{\text{R}} L_{110\in}$ mit konkreter Reduktion (2. ZK, HS14, A3)
19. $L_{\text{U}} \leq_{\text{R}} L_{\text{ungleich}}$ (2. ZK, HS13, A2)
20. $L_{\text{H}} \leq_{\text{EE}} L_{\text{H},001}$ (2. ZK, HS11, A1a)
21. $L_{001\in} \leq_{\text{R}} L_{\text{H},001}$ (2. ZK, HS11, A1b)
22. Aus $L_{\text{H}} \in \mathcal{L}_{\text{R}}$ folgt “KK-Berechnen”$\in \mathcal{L}_{\text{R}}$ (2. ZK, HS11, A3; EK, HS12, A3)
23. $L_{\text{H}} \leq_{\text{R}} L_{\text{U}}$ mit konkreter Reduktion (2. ZK, HS10, A3b)
24. $L_{\text{U}} \leq_{\text{R}} L_{\cup=\Sigma^*}$ mit konkreter Reduktion (2. ZK, HS10, A3b)
25. $L_{\text{U}} \leq_{\text{R}} L_{001\in}$ (2. ZK, HS09, A3a)

26. $L_{\mathrm{H},\lambda} \leq_{\mathrm{R}} L_{001\in}$ (2. ZK, HS09, A3b)
27. Satz 5.11: "Berechnen der Kolmogorovkomplexität $K(x)$ ist nicht rekursiv." (2. ZK, HS09, A4; EK, HS11, A5; EK, HS10, A6; EK, HS08, A5; EK, HS07, A5)
28. $L_{\mathrm{U},\lambda} \notin \mathcal{L}_{\mathrm{R}}$, mit Reduktion von L_{U}, L_{H} oder $L_{\mathrm{H},\lambda}$. (2. ZK, HS08, A3)
29. $L_{\mathrm{H}} \in \mathcal{L}_{\mathrm{RE}} - \mathcal{L}_{\mathrm{R}}$. (2. ZK, HS07, A2)
30. $L_{\mathrm{all}} \notin \mathcal{L}_{\mathrm{R}}$ (2. ZK, HS07, A3)
31. $L_{\lambda\in\mathrm{U}} \notin \mathcal{L}_{\mathrm{R}}$ (EK, HS17, A4a)
32. $L_{\mathrm{U}} \leq_{\mathrm{R}} L_{\exists,H}$ (EK, HS17, A4b)
33. $L_{\exists,\mathrm{H}} \in \mathcal{L}_{\mathrm{RE}}$ (EK, HS17, A4c)
34. $L_{\cap\neq\emptyset} \leq_{\mathrm{R}} L_{\mathrm{U}}$ (EK, HS16, A5a)
35. $L_{\mathrm{U}} \leq_{\mathrm{EE}} L_{\cap\neq\emptyset}$ (EK, HS16, A5b)
36. L_{diag} und $L^{\mathtt{C}}_{\mathrm{empty}}$ definieren. (EK, HS15, A5a)
37. $L_{\mathrm{diag}} \leq_{\mathrm{R}} L^{\mathtt{C}}_{\mathrm{empty}}$ mit konkreter Reduktion. (EK, HS15, A5b)
38. $L_{\not\subseteq} \notin \mathcal{L}_{\mathrm{R}}$ (EK, HS14, A5)
39. $L_{\mathrm{empty}} \notin \mathcal{L}_{\mathrm{R}}$ (Reduktion von L_{U}, L_{H} oder L_{diag}). (EK, HS13, A5b; EK, HS08, A4)
40. $L_{\mathrm{U}} \leq_{\mathrm{EE}} L_{\mathrm{all}}$ (EK, HS12, A2)
41. $L_{\mathrm{all}} \notin \mathcal{L}_{\mathrm{R}}$, mit Reduktion von L_{U}, L_{H}, L_{diag} oder L_{empty}. (EK, HS11, A5)
42. L_{diag} definieren. (EK, HS10, A5a)
43. $L_{\mathrm{diag}} \notin \mathcal{L}_{\mathrm{RE}}$. (EK, HS10, A5b)
44. $L_{\mathrm{H}} \leq_{\mathrm{R}} L_{\mathrm{empty}}$ (EK, HS09, A5b)
45. $L_{\mathrm{H}} \leq_{\mathrm{R}} L_{\mathrm{U}}$ (EK, HS08, A4)

Behauptung
$L_{\text{empty}}^{\complement} \in \mathcal{L}_{\text{RE}}$

Beweis (ohne Nichtdeterminismus)
Es gilt

$$L_{\text{empty}} \coloneqq \{\text{Kod}(M) \mid \text{TM } M \text{ akzeptiert keine Eingabe}\}.$$

und damit

$$\begin{aligned} L_{\text{empty}}^{\complement} = \ & \{x \mid x \text{ ist nicht von Form Kod}(M)\} \\ & \cup \{\text{Kod}(M) \mid \text{TM } M \text{ akzeptiert mindestens eine Eingabe}\} \end{aligned}$$

Wir geben eine TM D mit $L(D) = L_{\text{empty}}$ an.

D arbeitet auf der Eingabe x wie folgt.
D prüft, ob x die Form Kod(M) für eine TM M hat.
Falls nein, wird x akzeptiert.
Falls ja, simuliert D der Reihe nach für $k = 0, 1, 2 \ldots$ jeweils (bis zu) k Schritte von M auf den kanonisch ersten k Wörtern.
Sobald in der Simulation irgendein ein Wort akzeptiert wird, akzeptiert auch D. In jedem anderen Fall fährt man mit der Simulation fort.

Korrektheitsnachweis:
Falls x nicht von der Form Kod(M) ist, gilt $x \in L_{\text{empty}}^{\complement}$ und $x \in L(D)$.

Sonst:

$$\begin{aligned} & x = \text{Kod}(M) \in L_{\text{empty}}^{\complement} \\ \iff & \text{Es gibt ein } w, \text{ das von } M \text{ akzeptiert wird.} \\ \iff & \exists w_i \in \{0,1\}^* \colon M \text{ akzeptiert } w_i. \\ \iff & \exists w_i \in \{0,1\}^* \, \exists j \in \mathbb{N} \colon M \text{ akzeptiert } w_i \text{ in } j \text{ Schritten.} \\ \iff & \exists i, j \in \mathbb{N} \colon \ M \text{ akzeptiert } w_i \text{ in } j \text{ Schritten.} \\ \iff & \exists i, j \in \mathbb{N} \colon D \text{ akzeptiert } x \text{ für } k \leq \max\{i, j\}. \\ \iff & \exists k_0 \in \mathbb{N} \colon \ D \text{ akzeptiert } x \text{ für } k = k_0. \\ \iff & D \text{ akzeptiert } x. \\ \iff & x = \text{Kod}(M) \in L(N). \end{aligned}$$

Behauptung
$L^{\complement}_{\text{empty}} \in \mathcal{L}_{\text{RE}}$

Beweis (mit Nichtdeterminismus)
Es gilt

$$L_{\text{empty}} \coloneqq \{\text{Kod}(M) \mid \text{TM } M \text{ akzeptiert keine Eingabe}\}\,.$$

und damit

$$\begin{aligned} L^{\complement}_{\text{empty}} = \ & \{x \mid x \text{ ist nicht von Form Kod}(M)\} \\ & \cup \{\text{Kod}(M) \mid \text{TM } M \text{ akzeptiert mindestens eine Eingabe}\} \end{aligned}$$

Zu jeder NTM existiert eine äquivalente TM. Es reicht also, eine NTM N mit $L(N) = L_{\text{empty}}$ anzugeben. Es gibt damit auch eine TM M mit $L(M) = L_{\text{empty}}$, womit $L^{\complement}_{\text{empty}} \in \mathcal{L}_{\text{RE}}$ bewiesen ist.

N arbeitet auf der Eingabe x wie folgt.
N prüft, ob x die Form Kod(M) für eine TM M hat.
Falls nein, wird x akzeptiert.
Falls ja, wählt N nichtdeterministisch ein Wort $w \in \Sigma^*$
und arbeitet genau wie M auf w.
Korrektheitsnachweis:
Falls x nicht von der Form Kod(M) ist, gilt $x \in L^{\complement}_{\text{empty}}$ und $x \in L(N)$.

Sonst: $x = \text{Kod}(M) \in L^{\complement}_{\text{empty}}$

$\iff M$ akzeptiert eine Eingabe w.

$\iff$ Es gibt ein w, auf dem M q_{accept} erreicht.

$\iff$ Es gibt eine Wortwahl w, für die N auf x q_{accept} erreicht.

$\iff N$ kann w erraten, mit dem q_{accept} erreicht wird.

$\iff N$ hat eine akzeptierende Berechnung auf x.

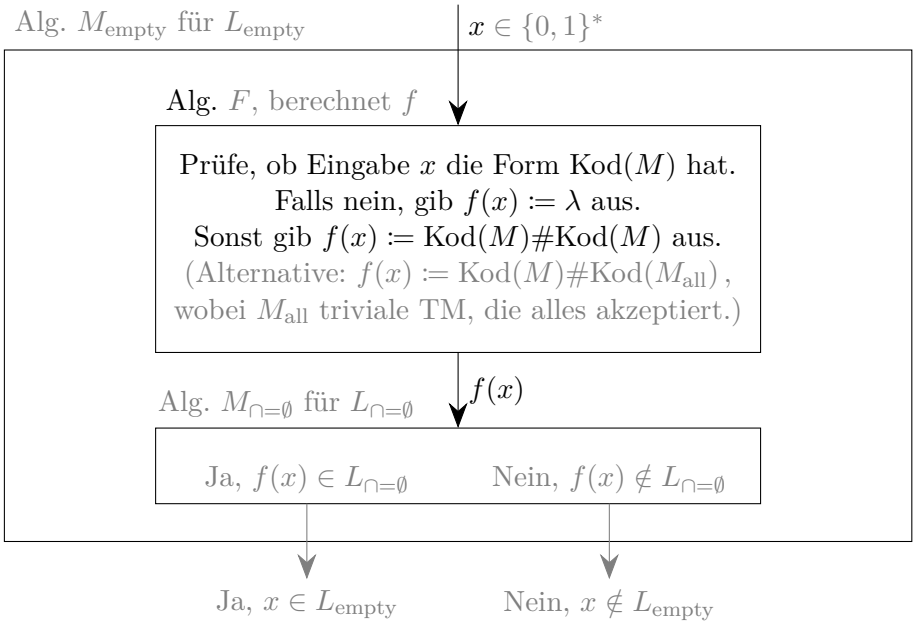
$\iff x = \text{Kod}(M) \in L(N)$.

Behauptung
Sei $L_{\cap=\emptyset} \coloneqq \{\text{Kod}(M)\#\text{Kod}(M') \mid M_1 \text{ und } M_2 \text{ sind TM und } L(M_1) \cap L(M_2) = \emptyset\}$.
Es gilt $L_{\cap=\emptyset} \notin \mathcal{L}_\text{R}$.

Beweis
Wir zeigen $L_\text{empty} \leq_\text{EE} L_{\cap=\emptyset}$. Daraus folgt $L_\text{empty} \leq_\text{R} L_{\cap=\emptyset}$. Wir wissen $L_\text{empty} \notin \mathcal{L}_\text{R}$, damit folgt die Behauptung.

Es gilt

$$L_\text{empty} \coloneqq \{\text{Kod}(M) \mid \text{TM } M \text{ akzeptiert keine Eingabe}\}.$$



Korrektheitsnachweis:
Falls x nicht von Form Kod(M) ist, gilt $x \notin L_\text{empty}$ und $f(x) = \lambda \notin L_{\cap=\emptyset}$.

Sonst:

$$\begin{aligned}
&\text{Kod}(M) = x \in L_\text{empty} \\
\iff\ &M \text{ akzeptiert kein Wort.} \\
\iff\ &L(M) = \emptyset \\
\iff\ &L(M) \cap L(M) = \emptyset \\
\iff\ &\text{Kod}(M)\#\text{Kod}(M) \in L_{\cap=\emptyset}. \\
(\iff\ &L(M) = \emptyset \\
\iff\ &L(M) \cap \Sigma^* = \emptyset \\
\iff\ &L(M) \cap L(M_\text{all}) = \emptyset \\
\iff\ &\text{Kod}(M)\#\text{Kod}(M_\text{all}) \in L_{\cap=\emptyset}\,)
\end{aligned}$$

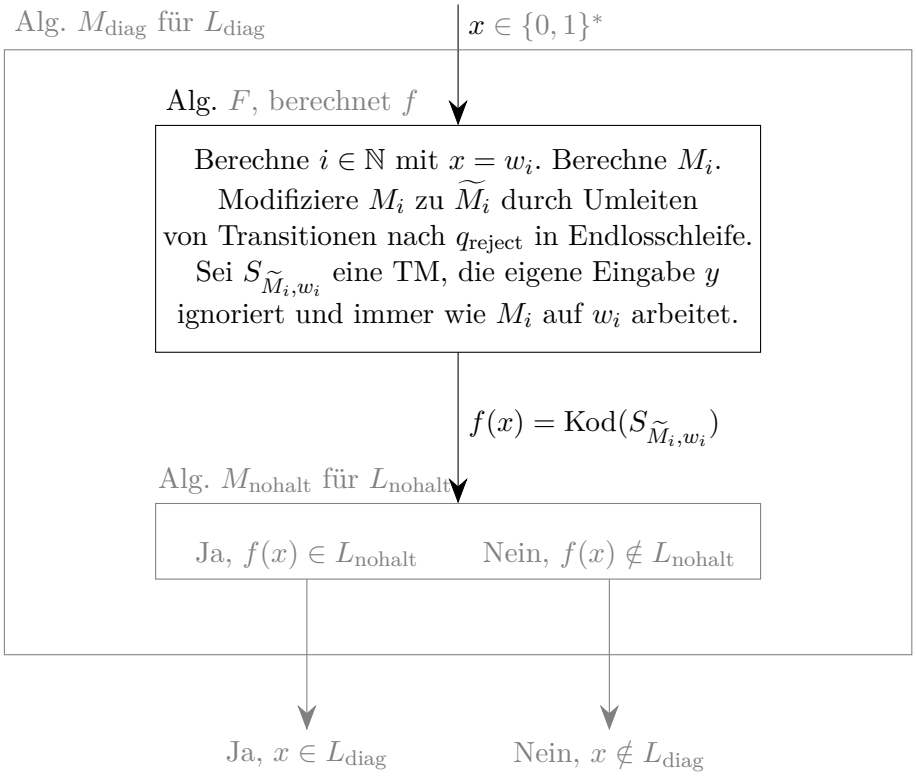
Behauptung
$L_{\text{diag}} \leq_{\text{EE}} L_{\text{nohalt}}$

Beweis
Sei w_i das i-te Wort über $\{0,1\}$ und M_i die i-te Turingmaschine (kanon. Ordnung).
Es gilt

$$L_{\text{diag}} \coloneqq \{w \mid w = w_i \notin L(M_i)\}$$

und

$$L_{\text{nohalt}} \coloneqq \{\text{Kod}(M) \mid \text{TM } M \text{ hält auf keiner Eingabe}\} .$$



Korrektheitsnachweis:

$$\begin{aligned} x \in L_{\text{diag}} &\iff M_i \text{ akzeptiert } x = w_i \text{ nicht.} \\ &\iff \widetilde{M_i} \text{ hält nicht auf } w_i \\ &\iff S_{\widetilde{M_i},w_i} \text{ hält nie.} \\ &\iff f(x) = \text{Kod}(S_{\widetilde{M_i},w_i}) \in L_{\text{nohalt}} \end{aligned}$$

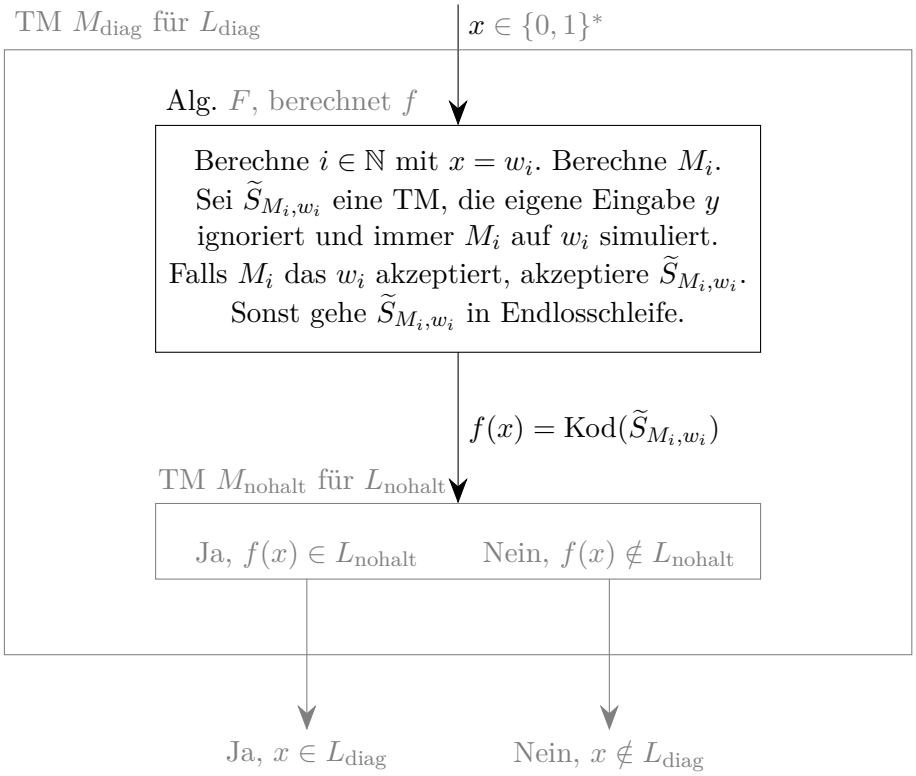
Behauptung
$L_{\text{diag}} \leq_{\text{EE}} L_{\text{nohalt}}$

Beweis
Sei w_i das i-te Wort über $\{0,1\}$ und M_i die i-te Turingmaschine (kanon. Ordnung).
Es gilt

$$L_{\text{diag}} \coloneqq \{w \mid w = w_i \notin L(M_i)\}$$

und

$$L_{\text{nohalt}} \coloneqq \{\text{Kod}(M) \mid \text{TM } M \text{ hält auf keiner Eingabe}\} .$$



Korrektheitsnachweis:

$$\begin{aligned} x \in L_{\text{diag}} &\iff M_i \text{ akzeptiert } x = w_i \text{ nicht.} \\ &\iff S_{M_i,w_i} \text{ hält nie.} \\ &\iff f(x) = \text{Kod}(S_{M_i,w_i}) \in L_{\text{nohalt}} \end{aligned}$$

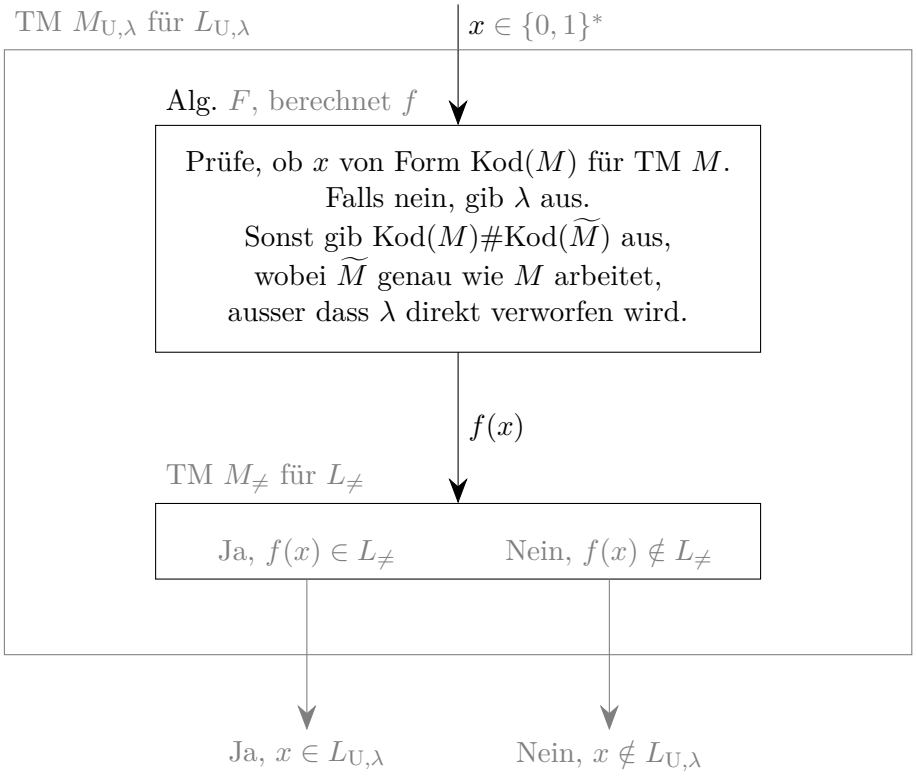
Behauptung
$L_{\mathrm{U},\lambda} \leq_{\mathrm{EE}} L_{\neq}$

Beweis
Es gilt

$$L_{\mathrm{U},\lambda} \coloneqq \{\mathrm{Kod}(M) \mid M \text{ akzeptiert } \lambda.\}$$

und

$$L_{\neq} \coloneqq \{\mathrm{Kod}(M)\#\mathrm{Kod}(M') \mid L(M_1) \neq L(M_2)\} \ .$$



Korrektheitsnachweis:
Falls x nicht von Form Kod(M)#w, gilt $x \notin L_{\mathrm{U},\lambda}$ und $f(x) = \lambda \notin L_{\neq}$.

Sonst:

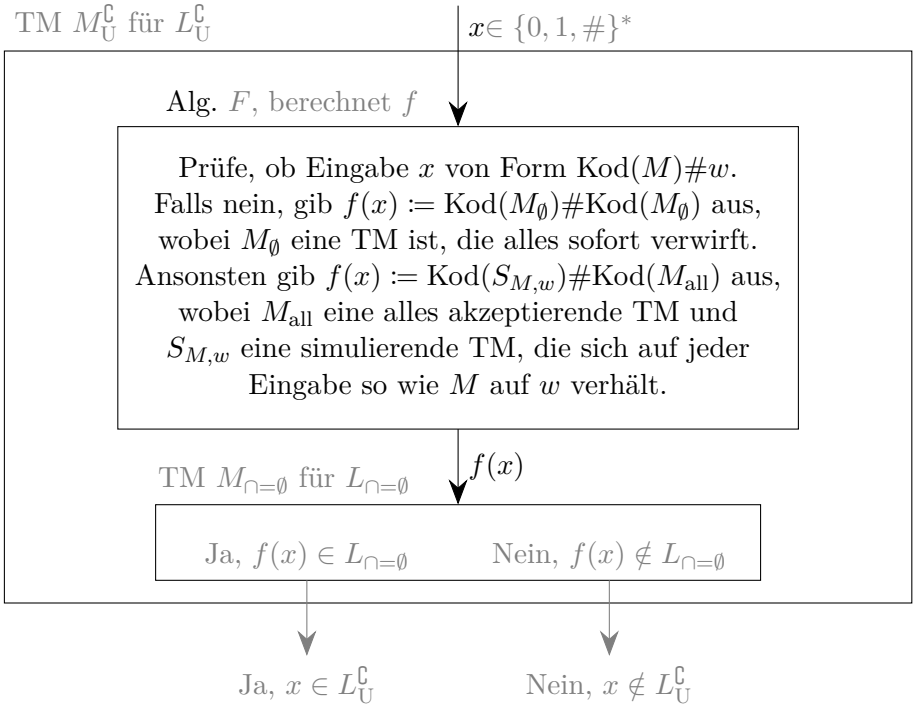
$$\begin{aligned}
&\mathrm{Kod}(M) = x \in L_{\mathrm{U},\lambda} \\
\iff\ &M \text{ akzeptiert } \lambda. \\
\iff\ &\lambda \in L(M). \\
\iff\ &\lambda \in L(M) \text{ und } \lambda \notin L(\widetilde{M}). \\
\iff\ &L(M) \neq L(\widetilde{M}). \\
\iff\ &\mathrm{Kod}(M)\#\mathrm{Kod}(\widetilde{M}) = f(x) \in L_{\mathrm{U},\lambda}
\end{aligned}$$

Behauptung
Sei $L_{\cap=\emptyset} \coloneqq \{\text{Kod}(M)\#\text{Kod}(M') \mid M_1 \text{ und } M_2 \text{ sind TM und } L(M_1) \cap L(M_2) = \emptyset\}$.
Es gilt $L_{\cap=\emptyset} \notin \mathcal{L}_\text{R}$.

Beweis
Wir zeigen $L_\text{U}^\complement \leq_\text{EE} L_{\cap=\emptyset}$. Daraus folgt $L_\text{U}^\complement \leq_\text{R} L_{\cap=\emptyset}$. Wir wissen $L_\text{U} \notin \mathcal{L}_\text{R}$, also auch $L_\text{U}^\complement \notin \mathcal{L}_\text{R}$, womit die Behauptung folgt.

$L_\text{U} \coloneqq \{\text{Kod}(M)\#w \mid \text{TM } M \text{ akzeptiert } w\}$

$L_\text{U}^\complement \coloneqq \{\text{Kod}(M)\#w \mid \text{TM } M \text{ akzeptiert } w \text{ nicht.}\} \cup \{x \mid x \text{ hat nicht Form Kod}(M)\#w\}$



Korrektheitsnachweis:
Falls x nicht Form Kod$(M)\#w$ hat, gilt $x \in L_\text{U}^\complement$ und $f(x) = \text{Kod}(M_\emptyset)\#\text{Kod}(M_\emptyset) \in L_{\cap=\emptyset}$.

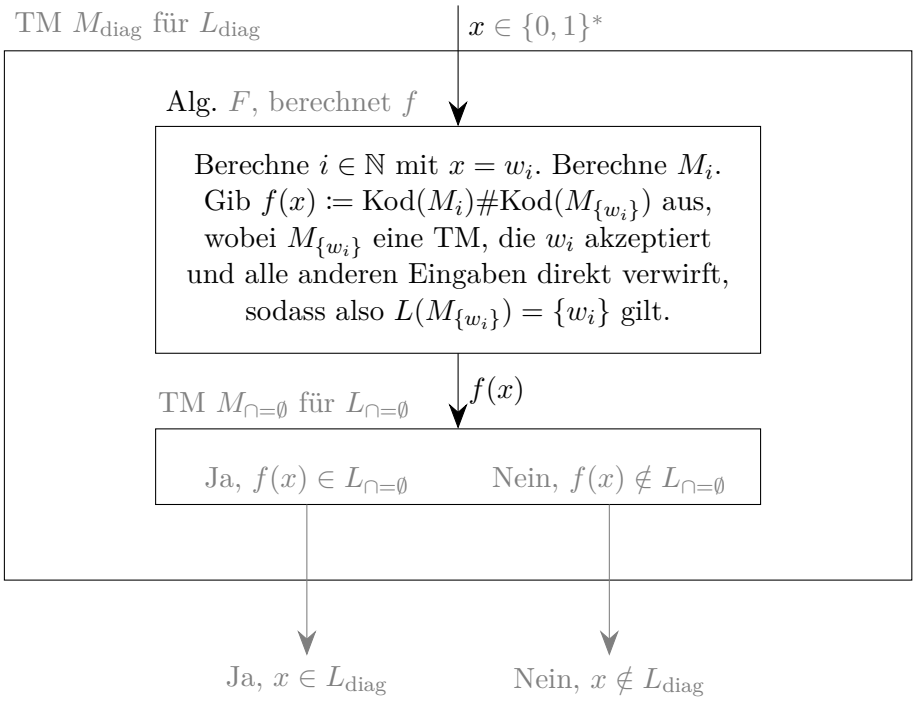
Sonst:
$$\begin{aligned}
&\text{Kod}(M) = x \in L_\text{U}^\complement \\
\iff &\text{Kod}(M) \notin L_\text{U} \\
\iff &w \notin L(M) \\
\iff &\forall y\colon y \notin L(S_{M,w}) \\
\iff &L(S_{M,w}) = \emptyset \\
\iff &L(S_{M,w}) \cap L(M_\text{all}) = \emptyset \\
\iff &\text{Kod}(S_{M,w})\#\text{Kod}(M_\text{all}) = f(x) \in L_{\cap=\emptyset}.
\end{aligned}$$

Behauptung
Sei $L_{\cap=\emptyset} \coloneqq \{\text{Kod}(M)\#\text{Kod}(M') \mid M_1 \text{ und } M_2 \text{ sind TM und } L(M_1) \cap L(M_2) = \emptyset\}$.
Es gilt $L_{\cap=\emptyset} \notin \mathcal{L}_\text{R}$.

Beweis
Wir zeigen $L_\text{diag} \leq_\text{EE} L_{\cap=\emptyset}$. Daraus folgt $L_\text{diag} \leq_\text{R} L_{\cap=\emptyset}$. Wir wissen $L_\text{diag} \notin \mathcal{L}_\text{RE}$, also auch $L_\text{diag} \notin \mathcal{L}_\text{R}$, womit die Behauptung folgt.

Sei w_i das i-te Wort über $\{0,1\}$ und M_i die i-te Turingmaschine (kanon. Ordnung). Es gilt

$$L_\text{diag} \coloneqq \{x \mid x = w_i \wedge w_i \notin M_i\}\,.$$

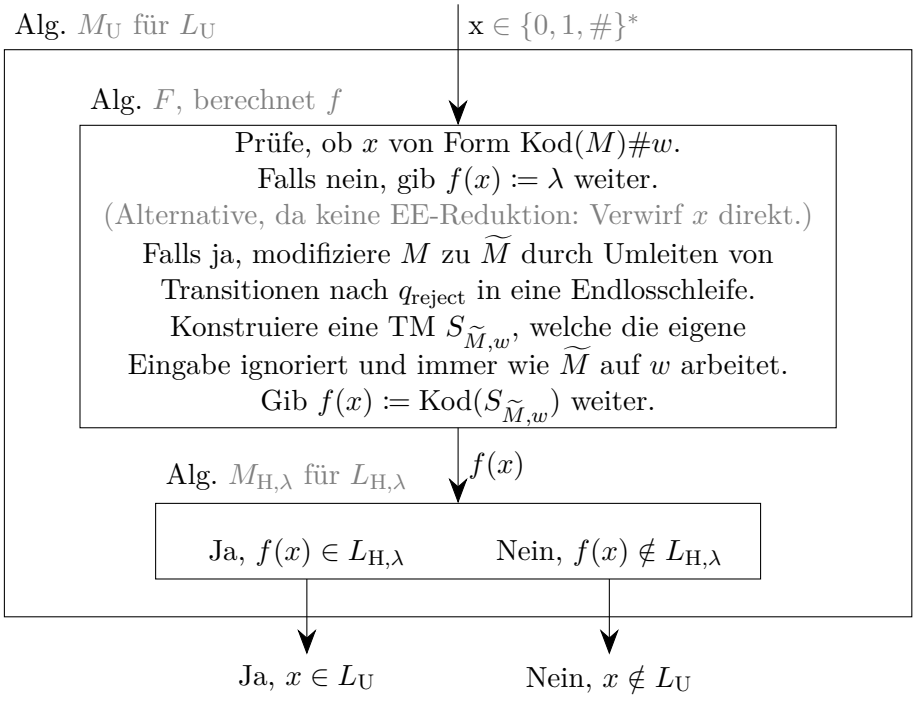


Korrektheitsnachweis:

$$\begin{aligned} x = w_i \in L_\text{diag} &\iff w_i \notin L(M_i) \\ &\iff w_i \notin L(M_i) \cap \{w_i\} \\ &\iff L(M_i) \cap \{w_i\} = \emptyset \\ &\iff \text{Kod}(M_i)\#\text{Kod}(M_{\{w_i\}}) = f(x) \in L_{\cap=\emptyset} \end{aligned}$$

Behauptung
$L_{\mathrm{U}} \leq_{\mathrm{R}} L_{\mathrm{H},\lambda}$

Beweis
Es gilt $L_{\mathrm{U}} \coloneqq \{\mathrm{Kod}(M)\#w \in \{0,1,\#\}^* \mid M \text{ akzeptiert } w.\}$
und $L_{\mathrm{H},\lambda} \coloneqq \{\mathrm{Kod}(M) \in \{0,1\}^* \mid M \text{ hält auf } \lambda.\}$.
Per Definition bedeutet $L_{\mathrm{U}} \leq_{\mathrm{R}} L_{\mathrm{H},\lambda}$, dass $L_{\mathrm{H},\lambda} \in \mathcal{L}_{\mathrm{R}} \implies L_{\mathrm{U}} \in \mathcal{L}_{\mathrm{R}}$.
Wir nehmen zum Beweis daher an, dass $L_{\mathrm{H},\lambda} \in \mathcal{L}_{\mathrm{R}}$, also ein Alg. $M_{\mathrm{H},\lambda}$ für $L_{\mathrm{H},\lambda}$ (also mit $L(M_{\mathrm{H},\lambda}) = L_{\mathrm{H},\lambda}$) existiert und konstruieren damit einen Alg. M_{U} für L_{U} (also mit $L(M_{\mathrm{U}})) = L_{\mathrm{U}}$), was $L_{\mathrm{U}} \in \mathcal{L}_{\mathrm{R}}$ beweist.
Das folgende Diagramm beschreibt die Arbeitsweise von M_{U}.

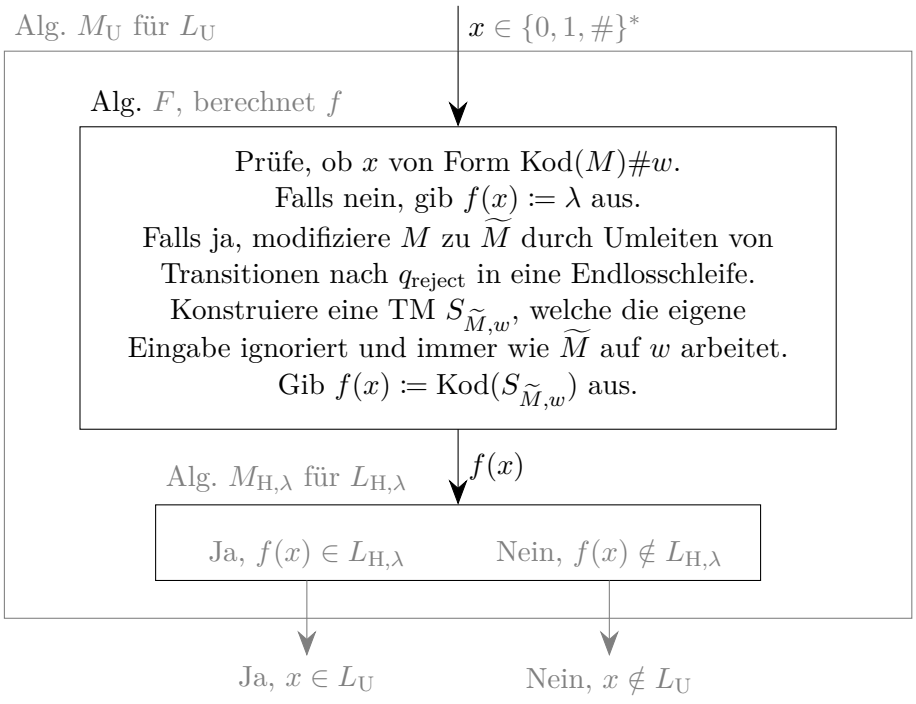


Korrektheitsnachweis:
Falls x nicht von Form $\mathrm{Kod}(M)\#w$ ist, gilt $x \notin L_{\mathrm{U}}$ und somit auch $f(x) = \lambda \notin L_{\mathrm{H},\lambda}$.
(Alternative: und damit wird x direkt verworfen.)

Sonst:
$$\begin{aligned} & x = \mathrm{Kod}(M)\#w \in L_{\mathrm{U}} \\ \iff & M \text{ akzeptiert } w. \\ \iff & \widetilde{M} \text{ hält auf } w. \\ \iff & S_{\widetilde{M},w} \text{ hält auf jeder Eingabe.} \\ \iff & S_{\widetilde{M},w} \text{ hält auf } \lambda \\ \iff & f(x) = \mathrm{Kod}(S) \in L_{\mathrm{H},\lambda} \end{aligned}$$

Behauptung
$L_{\mathrm{U}} \leq_{\mathrm{R}} L_{\mathrm{H},\lambda}$

Beweis
Es gilt $L_{\mathrm{U}} \coloneqq \{\mathrm{Kod}(M)\#w \in \{0,1,\#\}^* \mid M \text{ akzeptiert } w.\}$
und $L_{\mathrm{H},\lambda} \coloneqq \{\mathrm{Kod}(M) \in \{0,1\}^* \mid M \text{ hält auf } \lambda.\}$.
Wir zeigen $L_{\mathrm{U}} \leq_{\mathrm{EE}} L_{\mathrm{H},\lambda}$, denn $\leq_{\mathrm{EE}}$ impliziert $\leq_{\mathrm{R}}$.



Korrektheitsnachweis:

Falls x nicht von Form $\mathrm{Kod}(M)\#w$ ist, gilt $x \notin L_{\mathrm{U}}$ und $f(x) = \lambda \notin L_{\mathrm{H},\lambda}$.

Sonst:
$$\begin{aligned} & x = \mathrm{Kod}(M)\#w \in L_{\mathrm{U}} \\ \iff & M \text{ akzeptiert } w. \\ \iff & \widetilde{M} \text{ hält auf } w. \\ \iff & S_{\widetilde{M},w} \text{ hält auf jeder Eingabe.} \\ \iff & S_{\widetilde{M},w} \text{ hält auf } \lambda \\ \iff & f(x) = \mathrm{Kod}(S) \in L_{\mathrm{H},\lambda} \end{aligned}$$

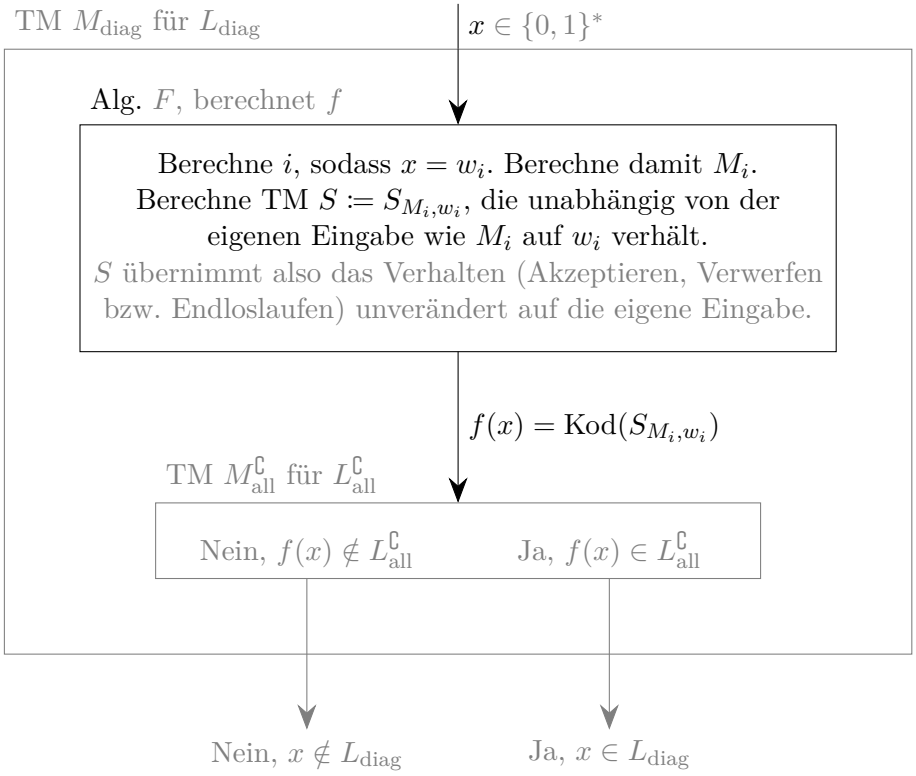
Behauptung
$L_{\text{diag}} \leq_{\text{EE}} L_{\text{all}}^{\complement}$

Beweis
Sei w_i das i-te Wort über $\{0,1\}$ und M_i die i-te Turingmaschine (kanon. Ordnung).
Es gilt

$$L_{\text{diag}} \coloneqq \{w_i \in \{0,1\}^* \mid M_i \text{ akzeptiert } w_i \text{ nicht}\}$$

und

$$\begin{aligned} L_{\text{all}}^{\complement} \coloneqq &\{\text{Kod}(M) \mid M \text{ TM mit } L(M) \neq \Sigma^*\} \\ &\cup \{x \in \{0,1\}^* \mid x \text{ nicht von Form Kod}(M)\} \ . \end{aligned}$$



Korrektheitsnachweis:

$$\begin{aligned} x \in L_{\text{diag}} &\iff M_i \text{ akzeptiert } x = w_i \text{ nicht.} \\ &\iff S_{M_i,w_i} \text{ akzeptiert kein Wort.} \\ &\iff S_{M_i,w_i} \text{ akzeptiert mindestens ein Wort nicht.} \\ &\iff L(S_{M_i,w_i}) \neq \Sigma^* \\ &\iff f(x) = \text{Kod}(S_{M_i,w_i}) \in L_{\text{all}}^{\complement} \end{aligned}$$

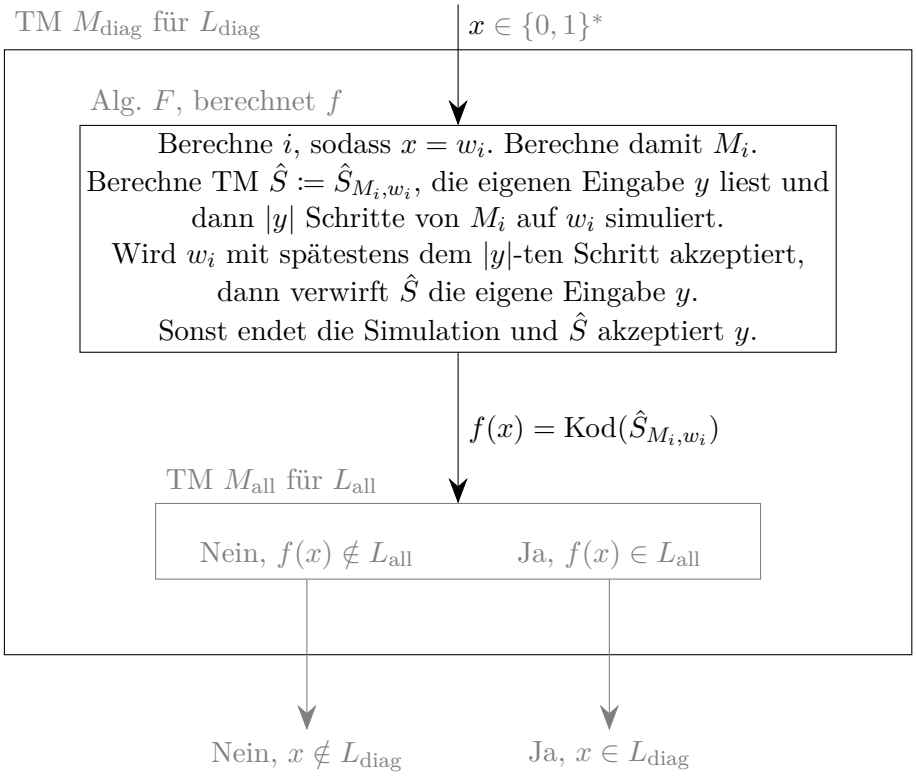
Behauptung
$L_{\text{diag}} \leq_{\text{EE}} L_{\text{all}}$

Beweis
Sei w_i das i-te Wort über $\{0,1\}$ und M_i die i-te Turingmaschine (kanon. Ordnung).
Es gilt

$$L_{\text{diag}} \coloneqq \{w_i \in \{0,1\}^* \mid M_i \text{ akzeptiert } w_i \text{ nicht}\}$$

und

$$L_{\text{all}} \coloneqq \{\text{Kod}(M) \mid M \text{ TM mit } L(M) \neq \Sigma^*\}.$$



Korrektheitsnachweis:

$$\begin{aligned}
x \in L_{\text{diag}} &\iff M_i \text{ akzeptiert } x = w_i \text{ nicht} \\
&\iff \forall k \in \mathbb{N}\colon\ M_i \text{ akzeptiert } w_i \text{ nicht in } k \text{ Schritten.} \\
&\iff \forall y \in \Sigma^*\colon\ M_i \text{ akzeptiert } w_i \text{ nicht in } |y| \text{ Schritten.} \\
&\iff \forall y \in \Sigma^*\colon\ \hat{S}_{M_i,w_i} \text{ akzeptiert die Eingabe } y. \\
&\iff L(\hat{S}_{M_i,w_i}) = \Sigma^* \\
&\iff f(x) = \text{Kod}(\hat{S}_{M_i,w_i}) \in L_{\text{all}}
\end{aligned}$$

Typische Aufgaben und Lösungsansätze zu Polynomzeitreduktionen

- Aufgabe: L ist NP-vollständig.
  - Zu zeigen: $L \in NP$ und L NP-schwer. (So hinschreiben.)
    (Siehe unten für beide Teile.)
- Aufgabe: PROB ist in NP.
  - Lösungsmethode:
    - **Beweisstruktur:** Wir beschreiben eine NTM N, die PROB erkennt.
    - **Konstruktion:** N errät (nichtdeterministisch) Zertifikat (z.B. erfüllende Belegung oder ein Vertex-Cover der verlangten Grösse) für die Eingabe und verifiziert es (deterministisch) in Polynomzeit. N akzeptiert, wenn die Verifikation erfolgreich.
    - **Korrektheitsbeweis:** Ein solches Zertifikat kann erraten werden. $\Leftrightarrow$ N hat eine akzeptierende Berechnung. $\Leftrightarrow$ N akzeptiert.
- Behauptung: PROB ist NP-schwer.
  - Lösungsmethode 1: Beweise $\text{PROB}' \leq_p \text{PROB}$ (siehe unten) für eine Sprache PROB′, die bekanntermassen NP-schwer ist. (Dies auch erwähnen.)
  - Lösungsmethode 2: Etwas à la Satz von Cook, also allgemeine Reduktion. (Sehr aufwendig, nicht realistisch für Prüfung.)
- Behauptung: $\text{PROB}_1 \leq_p \text{PROB}_2$.
  - Lösungsmethode:
    - Beschreibe Reduktion.
      - **Beweisrahmen:** Sei x eine Eingabe für PROB_1.
      - **Konstruktion:** Beschreibe Polynomzeitumwandlung von x in $f(x)$, sodass $x \in \text{PROB}_1 \iff f(x) \in \text{PROB}_2$.
      - **Korrektheitsbeweis:** $x \in \text{PROB}_1 \iff f(x) \in \text{PROB}_2$ (Für diesen Punkt oft auch eine nicht ganz triviale Beweisstruktur erforderlich.)
      - **Polynomzeit prüfen:** Beweise/erwähne das alles in Polynomlaufzeit möglich. (Meist trivial.)

Beweisstruktur illustriert anhand eines (fast) trivialen Beispiels

Aufgabe: Sei

DOPPEL-CLIQUE $= \{(G, k) \mid G$ ist ungerichteter Graph mit mindestens 2 Cliquen der Grösse $k\}$.

Beweise, dass DOPPEL-CLIQUE NP-vollständig ist.

Lösung: Zu zeigen: DOPPEL-CLIQUE $\in$ NP und DOPPEL-CLIQUE NP-schwer.

- DOPPEL-CLIQUE $\in$ NP: Errate nichtdet. Doppel-Clique, Prüfung in Polynomzeit möglich.
- Da CLIQUE NP-schwer, reicht es CLIQUE $\leq_p$ DOPPEL-CLIQUE zu zeigen:
  - **Konstruktion:** Wir bilden (G, k) auf $(G \cup G, k)$ ab, wobei $\cup$ hier die disjunkte Vereinigung bezeichnet.
  - **Polynomzeit:** Konstruktion offensichtlich in Polynomzeit möglich.
  - **Korrektheit:**

    "$\Rightarrow$": Wenn G eine Clique der Grösse k hat, hat $G \cup G$ offensichtlich zwei.

    "$\Leftarrow$": Angenommen, $G \cup G$ hat zwei Cliquen der Grösse k. Wähle eine von beiden. Da es eine Clique ist und die beiden Kopien von G disjunkt sind, befindet sie sich vollständig in einer Kopie. Also ist es auch eine Clique der Grösse k für G.

Muster für reine Polynomzeitreduktionsaufgabe

Definitionen:
Sei 4SAT analog zu 3SAT definiert, also:

$$\text{4SAT} \coloneqq \{\Phi \mid \Phi \text{ ist Formel in 4KNF und erfüllbar.}\}$$

Wir sagen dabei, $\Phi \in \Sigma^*_{\text{logic}}$ habe (bzw. sei in) 4KNF (also 4-Konjunktivnormalform), wenn Φ eine syntaktisch gültige Formel in KNF (konjunktiver Normalform) ist und jede disjunktive Klausel aus höchstens 4 Literalen besteht.

Behauptung:
Es gilt $\text{4SAT} \leq_{\text{p}} \text{3SAT}$.

Beweis:
Konstruktion:
Wir beschreiben Poly.-Alg. (also einen Algorithmus, der in Zeit läuft, die polynomiell in cer Eingabelänge ist), der aus einer Eingabe für 4SAT (also einer Formel Φ in 4KNF) eine äquialente Eingabe für 3SAT (also eine Formel Ψ in 3KNF) konstruiert. Äquivalent bedeutet hier $\Phi \in \text{4SAT} \Leftrightarrow \Psi \in \text{3SAT}$, also Φ erfüllbar. $\Leftrightarrow$ Ψ erfüllbar.
Falls Φ nicht 4KNF hat, sei $\Psi = \Phi$.
Sonst: Sei $\Phi = C_1 \wedge \ldots \wedge C_m$, wobei die C_i disjunktive Klauseln mit maximal 4 Literalen sind. Sei dann $\Psi \coloneqq \widetilde{C}_1 \wedge \ldots \wedge \widetilde{C}_m$, wobei

$$\widetilde{C}_i \coloneqq \begin{cases} C_i, & \text{falls } C_i \text{ aus höchstens 3 Literalen besteht} \\ (\ell_1 \vee \ell_2 \vee y_i) \wedge (\ell_3 \vee \ell_4 \vee \overline{y}_i), & \text{falls } C_i = (\ell_1 \vee \ell_2 \vee \ell_3 \vee \ell_4), \end{cases}$$

und y_i neue Variablen, die sonst also nirgends als Variablen in Φ auftauchen und untereinander verschieden sind.

Fortsetzung nächster Seite.

Korrektheit:
Beweise $\Phi \in 4\text{SAT} \iff \Psi \in 3\text{SAT}$.

Fall 1
Φ hat nicht 4KNF: Dann hat $\Psi = \Phi$ auch nicht 3KNF; es gilt also $\Phi \notin 4\text{SAT}$ und $\Psi \notin 3\text{SAT}$.

Fall 2
Φ hat 4KNF. Dann gilt: Φ erfüllbar $\iff$ Ψ erfüllbar.

- "$\Longrightarrow$":
  α erfülle Φ, also alle C_i. Sei $\beta = \alpha$ auf Variablen von Φ und
  $$\beta(y_i) \coloneqq \begin{cases} 0 & \text{falls } \beta(\ell_1) = 1 \text{ oder } \beta(\ell_2) = 1 \\ 1, & \text{sonst.} \end{cases}$$
  Dann erfüllt β alle $\widetilde{C}_i$. Trivial für $\widetilde{C}_i = C_i$. Sei also $C_i = (\ell_1 \vee \ell_2 \vee \ell_3 \vee \ell_4)$.
  α erfüllt ℓ_1, ℓ_2, ℓ_3 oder ℓ_4, also auch β.
  Ist ℓ_1 oder ℓ_2 erfüllt, so auch $(\ell_1 \vee \ell_2 \vee y_i)$. Zudem ist dann $\beta(y_i) = 0$, also ist auch $(\ell_3 \vee \ell_4 \vee \overline{y}_i)$ erfüllt und damit $\widetilde{C}_i$.
  Sonst ist ℓ_3 oder ℓ_4 erfüllt, also auch $(\ell_3 \vee \ell_4 \vee \overline{y}_i)$, und zudem ist $\beta(y_i) = 1$, also ist auch $(\ell_1 \vee \ell_2 \vee y_i)$ erfüllt und damit $\widetilde{C}_i$.

- "$\Longleftarrow$":
  β erfülle Ψ, also alle $\widetilde{C}_i$. Dann erfüllt β (oder genauer gesagt: die Einschränkung α von β auf die Variablenmenge von Φ) auch alle C_i, also Φ. Trivial für $\widetilde{C}_i = C_i$.
  Und sonst erfüllt β beide Klauseln von $\widetilde{C}_i = (\ell_1 \vee \ell_2 \vee y_i) \wedge (\ell_3 \vee \ell_4 \vee \overline{y}_i)$. Entweder y_i oder $\overline{y}_i$ ist nicht erfüllt, also muss ein ℓ_1, ℓ_2, ℓ_3 oder ℓ_4 erfüllt sein, also auch C_i.

Es gibt also einen Polynomzeitalgorithmus F, der eine Funktion $f\colon \Sigma^*_{\text{logic}} \to \Sigma^*_{\text{logic}}$ mit $f\colon \Phi \mapsto \Psi = F(\Phi)$ berechnet, sodass $\Phi \in 4\text{SAT} \iff \Psi \in 3\text{SAT}$ gilt. □

Kreativer Teil: Vorgehen, um die richtige Konstruktion zu finden.

Die beiden wichtigsten Typen von Sprachen für uns sind:

1. Typ "SAT": SAT und Derivate 3SAT, E3SAT, VIERFACHSAT etc.
2. Typ "Graph": Alles andere, insb. Graphenprobleme, insb. VC = Vertex Cover, Clique, und IS = IndependentSet. Zudem DS = DominatingSet und das Nicht-Graphen-Problem SCP = SetCoverProblem.

Wie findet man in den verschiedenen Fällen eine geeignete Konstruktion? Nachfolgend sind Methoden für die typischsten Fälle skizziert. (Achtung, der Korrektheitsbeweis ist im Folgenden nirgends ausgeführt, dieser muss aber immer ergänzt werden, und zwar immer in beide Richtungen.)

- "SAT"$\leq_p$"SAT". Drei Grundvarianten und eine allgemeine Methode.
  1. Anzahl Literale pro Klausel verringern. (Lemma 6.11, S. 209).
     Zum Beispiel für E5SAT $\leq_p$ E3SAT, klauselweise folgende Abbildung:
     $(x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5) \mapsto (x_1 \vee x_2 \vee y_2) \wedge (\overline{y}_2 \vee x_3 \vee y_3) \wedge (\overline{y}_3 \vee x_4 \vee x_5)$,
     wobei alle y_i neue Variablen sind.
  2. Anzahl Literale pro Klausel erhöhen.
     Zum Beispiel für 3SAT $\leq_p$ E3SAT, klauselweise folgende Abbildung:
     $(x_1 \vee x_2 \vee x_3) \mapsto (x_1 \vee x_2 \vee x_3)$,
     $(x_1 \vee x_2) \mapsto (x_1 \vee x_2 \vee y_1) \wedge (x_1 \vee x_2 \vee \overline{y}_1)$,
     $(x_1) \mapsto (x_1 \vee y_1 \vee y_2) \wedge (x_1 \vee \overline{y}_1 \vee y_2) \wedge (x_1 \vee y_1 \vee \overline{y}_2) \wedge (x_1 \vee \overline{y}_1 \vee \overline{y}_2)$,
     wobei alle y_i neue Variablen sind.
  3. Anzahl erfüllender Belegungen vervielfachen.
     Zum Beispiel SAT $\leq_p$ 5FACH-SAT:[1]
     Hier nicht klauselweise, sondern einmal eine einfache globale Abbildung:
     $\Phi \mapsto \Phi \wedge (y_1 \vee y_2 \vee y_3)$, wobei alle y_i neue Variablen sind.
  4. Wenn nichts vom Obigen hilft: Benutze neue Variablen und beliebige Implikationen $l_i \Rightarrow l_j$, beliebig verbunden mit $\wedge$, $\vee$ und Negationen $\neg$ bzw. $\overline{x}$, um die geforderten Bedingungen umzusetzen. Benutze dann die Gesetze von De Morgan und die Distributivität von $\vee$ über $\wedge$, um eine KNF zu erhalten. Siehe für eine etwas ausführlichere Beschreibung den letzten Punkt"Graph"$\leq_p$"SAT".

  Man beachte, dass man in den Fällen 1. und 2. und gegebenenfalls auch 4. nicht nur in der Konstruktion, sondern auch im Korrektheitsbeweis zunächst auf die Klauselebene wechseln muss. Das Schema sieht ungefähr wie folgt aus: Sei $\Phi = \mathbb{C}_1 \wedge \ldots \wedge C_m$ eine XYZ-KNF-Formel mit Klauseln C_i. Wir konstruieren $\Psi =$

[1] Das Entscheidungsproblem 5FACH-SAT ist die Menge der KNF mit mindestens fünf verschiedenen erfüllenden Belegungen.

$F_1 \wedge \ldots \wedge F_m$ per $C_i \mapsto F_i = \ldots$

$\Phi \in$ XYZ-SAT
$\iff$ Φ ist erfüllbar.
$\iff$ $\exists$ Belegung α: α erfüllt Φ.
$\iff$ $\exists$ Belegung α: $\forall i \alpha$ erfüllt C_i.
$\vdots$ *Argumentation auf Klausel-/Teilformel-Ebene mit konkretem α bzw. β*
$\iff$ $\exists$ Belegung β: $\forall i \alpha$ erfüllt F_i.
$\iff$ $\exists$ Belegung β: β erfüllt Ψ.
$\iff$ Ψ ist erfüllbar.
$\iff$ $\Psi \in$ ABC-SAT

Für konkrete Ausführung, siehe Beweismuster.

- "Graph"$\leq_p$"Graph". Beispiel Buch Clique $\leq_p VC$ (Lemma 6.10, S. 208).
  - Clique und IS sind aufeinander reduzierbar durch Abbildung auf Komplementgraph, nämlich $(G, k) \mapsto (\overline{G}, k)$. Dabei ist $G = (V, E)$, $n = |V|$, der Komplementgraph ist $\overline{G} = (V, \overline{E})$ mit $\overline{E} = \binom{V}{2} \setminus E$, wobei $\binom{V}{2} = \{\{u, v\} \mid u, v, \in V, u \neq v\}$ die Menge aller möglichen Kanten ist. (Korrektheitsbeweis sehr einfach.)
  - VC und IS sind aufeinander reduzierbar durch die Abbildung $(G, k) \mapsto (G, n-k)$. (Korrektheitsbeweis leicht schwieriger.)
  - VC und Clique sind aufeinander reduzierbar durch die Abbildung $(G, k) \mapsto (\overline{G}, n-k)$. (Verknüpfung der beiden obigen Abbildungen und Korrektheitsbeweise.)
  - VC $\leq_p$ DS mittels Ersetzen jeder Kante durch ein Dreieck. (Also einem neuen Knoten und zwei neuen Kanten pro bestehder Kante.) DS $\leq_p$ VC komplizierter, sehr unwahrscheinlich an Prüfung.[2]
- "SAT"$\leq_p$"Graph". Beispiel in Buch, $SAT \leq_p$ Clique (Lemma 6.9, S. 206). Reduktion auf IS und auf VC durch Verknüpfung oben gegebenen Abbildungen.
- "Graph"$\leq_p$"SAT".
  - Allgemein gelöst durch Satz von Cook (Satz 6.9, S. 199), der sämtliche Sprachen in NP auf SAT reduziert.
  - Konkrete Reduktionen aber häufig sehr komplex. Insbesondere zu komplex für Prüfung z.B. Reduktion von VC, IS und VC, benötigt Wissen über Addier-Schaltkreise polynomieller Grösse.

[2] Lösung: "Verdopple" alle Knoten, d.h., ersetze jeden originalen Knoten v durch zwei Knoten v_1 und v_2, die untereinander verbunden sind und beide mit allen Knoten verbunden sind, mit welchen v verbunden war. Ein Vertex-Cover muss mindestens einen der beiden Knoten v_1 und v_2 enthalten, der Rest verhält sich wie ein Dominating-Set. Man erhält ein Vertex-Cover der Grösse $n+k$ genau dann, wenn man ein Dominating-Set der Grösse k hat.

– Für Fälle, wo es mit vertretbarem Aufwand möglich ist, ist das Vorgehen zum Beweis von Prob $\leq_p$ SAT prinzipiell folgendes:
  * Finde geeignete Lösungsbestandteile, aus welchen sich alle Lösungkandidaten (Zertifikate) S für ein Instanz I von Prob zusammensetzen lassen. Zum Beispiel für VC die Knoten des Graphen G aus der Eingabe $I = (G, k)$.
  * Die Variablenmenge X enthalte eine Variable pro Bestandteil. Seien die Variablen o.B.d.A. $x_1, \ldots, x_n$.
  Für das Beispiel ist VC: $X = \{x_v \mid v \in V\}$.
  * Wir erhalten eine direkte Korrespondenz zwischen Belegungen $\alpha\colon X \to \{0, 1\}$ und Lösungskandidaten S für Prob.
  Im Beispiel: $S \subseteq V$.
  * Codiere die Bedingungen, die einen Lösungskandidaten (ein Zertifikat) zu einer Lösung (einem gültigem Zertifikat) machen, durch eine SAT-Formel Φ. (Siehe nächsten Punkt für Ansätze dafür.) Es gilt dann für alle korrespondierenden S und α:

    $$S \text{ ist gültiges Zertifikat für } I \iff \alpha \text{ erfüllt } \Phi.$$

    Es folgt:

    Es ex. gültiges Zertifikat S für $I \iff$ Es ex. erfüllende Belegung für Φ,

    was gleichbedeutend ist mit

    $$I \in \text{Prob} \iff \Phi \in \text{SAT}.$$

    Im Beispiel: S muss eine Knotenüberdeckung[3] sein, das heisst, jede Kante muss durch einen Knoten in S überdeckt sein. Wie codiert man die notwendigen Bedingungen in eine KNF-Formel Φ? Finde äquivalente Bedingungen der folgenden Form:

    1. Sei a die Anzahl Variablen, welche eine erfüllende Belegung auf 1 setzt, und seien $d, D \in \mathbb{N}$ zwei natürliche Zahlen. Dann soll $d \leq a \leq D$ gelten.
    2. Formuliere alle restlichen Bedingungen direkt durch Teilformeln oder durch Implikationen der Form $x_i \Rightarrow x_j$, beliebig kombiniert mit Klammern, $\wedge$, $\vee$ und Negationen $\neg$ bzw. $\overline{x}$. (Insbesondere kann man $x_i \Leftrightarrow x_j$ codieren als $(x_i \Rightarrow x_j) \wedge (x_j \Rightarrow x_i)$, und analog für $x_i \Leftrightarrow \overline{x}_j$.)
    Im Beispiel VC hat man für die Eingabe (G, k) die Bedingungen $d = D = k$ und für jede Kante $\{x_u, x_v\} \in E$ hat man $x_u \vee x_v$.

[3] Beachte hier, dass der Begriff Vertex-Cover/Knotenüberdeckung etwas kontraintuitiv ist und dadurch gerne falsch interpretiert wird. In einer Knotenüberdeckung sind nicht alle Knoten überdeckt—wie der Name es spontan suggerieren würde—, stattdessen werden *mit* (einer Teilmenge von) Knoten alle *Kanten* überdeckt.

Verwende nun folgendes Verfahren, um alles in eine KNF zu verpacken.

1. Eine Implikation $l_i \Rightarrow l_j$ ist äquivalent zu der disjunktiven Klausel $\bar{l}_i \vee l_j$.
2. Es sollen mindestens d der n Variablen erfüllt sein:
$$\bigwedge_{1 \leq i_1 < \ldots < i_d \leq n} (x_{i_1} \vee \ldots \vee x_{i_d}).$$
3. Es sollen höchstens D der n Variablen erfüllt sein:
$$\bigwedge_{1 \leq i_1 < \ldots < i_{D+1} \leq n} \overbrace{(\overline{z}_1 \vee \ldots \vee \overline{z}_{D+1})}^{\neg(z_1 \wedge \ldots \wedge z_{K+1})}$$
4. Verwende die Gesetze von De Morgan (nämlich $\neg(x_i \vee x_j) \equiv \overline{x}_i \wedge \overline{x}_j$ und $\neg(x_i \wedge x_j) \equiv \overline{x}_i \vee \overline{x}_j$, um alle Negationen auf die Ebene der einzelnen Variablen zu bringen. (Im obigen Punkt bereits angewendet.)
5. Verwende bei Bedarf die Distributivität von $\vee$ über $\wedge$, also $x \vee (y \wedge z) \equiv (x \vee y) \wedge (x \vee z)$, um die Formel definitiv in eine KNF umzuformen.

Weiteres gutes Beispiel, für das diese Methode (ausserhalb des Zusammenhangs von NP-Schwierigkeitsbeweisen) verwendet werden kann, ist das eines $n \times n$-Schachbretts, auf dem n Türme platziert werden sollen, die sich gegenseitig nicht bedrohen.

Bekannte Polynomzeitreduktionen

Aus Vorlesung/Buch

- "TM" $\leq_p$ SAT (Satz von Cook, Satz 6.9, S. 199)
- SAT $\leq_p$ CLIQUE (Lemma 6.9, S. 206)
- CLIQUE $\leq_p$ VC (Lemma 6.10, S. 208)
- SAT $\leq_p$ 3SAT (Lemma 6.11)

Übungsaufgaben des Buchs

- VC $\leq_p$ CLIQUE (Aufgabe 6.22a, S. 210)
- 3SAT $\leq_p$ VC (Aufgabe 6.22b, S. 210)
- SAT $\leq_p$ 4SAT (Kontrollaufgabe 6.18a, S. 215)
- 5SAT $\leq_p$ CLIQUE (Kontrollaufgabe 6.18b, S. 215)
- CLIQUE $\leq_p$ 4SAT (Kontrollaufgabe 6.18c, S. 215)
- 4SAT $\leq_p$ 3SAT (Kontrollaufgabe 6.18d, S. 215)
- 5SAT $\leq_p$ VC (Kontrollaufgabe 6.18e, S. 215)
- VC $\leq_p$ SAT (Kontrollaufgabe 6.18f, S. 215)

Alte Prüfungsaufgaben

- NON-3-MONOTONE-3SAT NP-vollständig. *Als NP-schwer bekannt:* SAT/3SAT/E3SAT/3-/4-FACH-SAT/CLIQUE/VC/SCP/DS/SUBSET-SUM. (2. ZK, HS17, A4; EK, HS13, A6)
- 4SAT $\leq_p$ 3SAT (2. ZK, HS16, A3)
- LARGE-CLIQUE (G mit $|V| = 3k$, hat k-Clique.) NP-vollständig. *Als NP-schwer bekannt:* SAT/3SAT/E3SAT/DOPPEL-SAT/CLIQUE/VC. (2. ZK, HS15, A4; leicht variiert zu $k \geq |V|/3$: EK, HS07, A6)
- SCHWELLENWERT-2SAT NP-schwer. Viele Hilfestellungen. (2. ZK, HS14, A4)
- LARGE-CLIQUE (G mit $|V| = 3k$, hat k-Clique.) NP-vollständig. *Als NP-schwer bekannt:* ((E)3)SAT/TRIPEL-SAT/CLIQUE/IS/SUBSET-SUM. (2. ZK, HS13, A4)
- DOPPEL-CLIQUE (G hat 2 disj. k-Cliquen) NP-vollständig. (2. ZK, HS12, A4)
- 5SAT $\leq_p$ 3SAT. (2. ZK, HS11, A2b; EK, HS15, A6)
- IS ist NP-vollständig, mittels Reduktion von CLIQUE. (2. ZK, HS11, A4)
- VC $\leq_p$ DS. (EK, HS17, A5; EK, HS12, A4)
- 3SAT $\leq_p$ VC. (EK, HS17, A5; EK, HS14, A6; EK, HS11, A6)
- SAT $\leq_p$ CLIQUE. (EK, HS10, A7; EK, HS09, A6)

Übersicht Hierarchiesätze

Aussagen des Buches, 5. Auflage. Überall gilt $s = s(n)$ und $t = t(n)$. Triviale Korollare und schwierige Beweise, die auch im Buch fehlen, in Grau.

Nur Determinismus

Lemma 6.1 Jede k-Band-TM A hat äquivalente 1-Band-TM B mit $\text{Space}_B(n) \leq \text{Space}_A(n)$.

Lemma 6.2 Zu MTM A hat äquivalente MTM B mit $\text{Space}_B(n) \leq \frac{\text{Space}_A(n)}{2} + 2$.

Aufgabe 6.1 Jede MTM A hat äquivalente MTM B mit $\text{Time}_B(n) \leq \frac{\text{Time}_A(n)}{2} + 2n$.

Satz 6.1 Es existiert eine Sprache, sodass zu jeder MTM A mit $L(M)$ eine äquivalente MTM B existiert mit $\text{Time}_B(n) \leq \log \text{Time}_A(n)$ für unendlich viele $n \in \mathbb{N}$.

Lemma 6.3 $\text{TIME}(t) \subseteq \text{SPACE}(t)$

Korollar 6.1 $\text{P} \subseteq \text{PSPACE}$

Lemma 6.4 Sei s platzkonstruierbar. Zu jeder MTM, welche $\text{Space}_M(w) \leq s(|w|)$ nur für alle $w \in L(M)$ erfüllt, existiert äquivalente M', welche dies für alle $w \in \Sigma^*$ erfüllt.

Lemma 6.5 Sei s zeitkonstruierbar. Zu jeder MTM, welche $\text{Time}_M(w) \leq t(|w|)$ nur für alle $w \in L(M)$ erfüllt, existiert äquivalente M', welche zumindest $\text{Time}_M(w) \leq 2t(|w|) \in O(t(|w|))$ für alle $w \in \Sigma^*$ erfüllt.

Satz 6.2 $\text{SPACE}(s) \subseteq \bigcup_{c \in \mathbb{N}} \text{TIME}(c^s)$ für $s \geq \log n$.

Korollar 6.2 $\text{DLOG} \subseteq \text{P}$ und $\text{PSPACE} \subseteq \text{EXPTIME}$

Satz 6.3* $\text{SPACE}(s_1) \subsetneq \text{SPACE}(s_2)$ für alle $s_1 \in o(s_2)$ mit $s_2(n) \geq \log n$ platzkonstruierbar.

Satz 6.4* $\text{TIME}(s_1 \log t_1) \subsetneq \text{TIME}(t_2)$ für alle $t_1 \in o(t_2)$ mit $s(n) \geq \log n$ zeitkonstruierbar.

Inklusive Nichtdeterminismus

Lemma 6.6.(i) $\mathrm{NTIME}(t) \subseteq \mathrm{NSPACE}(t)$

Lemma 6.6.(ii) $\mathrm{NSPACE}(s) \subseteq \bigcup_{c\in\mathbb{N}} \mathrm{NTIME}(c^s)$ mit $s(n) \geq \log n$.

Satz 6.5.(i) $\mathrm{TIME}(t) \subseteq \mathrm{NTIME}(t)$

Satz 6.5.(ii) $\mathrm{SPACE}(s) \subseteq \mathrm{NSPACE}(s)$

Satz 6.5.(iii) $\mathrm{NTIME}(t) \subseteq \mathrm{SPACE}(t)$ für platzkonstruierbares $t(n) \geq \log n$.

Korollar 6.3 $\mathrm{NP} \subseteq \mathrm{PSPACE}$

Satz 6.6* $\mathrm{NSPACE}(s) \subseteq \bigcup_{c\in\mathbb{N}} \mathrm{TIME}(c^s)$ für platzkonstruierbares $s(n) \geq \log n$.

Korollar 6.4 $\mathrm{NLOG} \subseteq \mathrm{P}$ und $\mathrm{NSPACE} \subseteq \mathrm{EXPTIME}$

Satz 6.7* $\mathrm{NSPACE}(s) \subseteq \mathrm{SPACE}(s^2)$ für platzkonstruierbares $s \geq \log n$

Korollar 6.5 $\mathrm{PSPACE} = \mathrm{NPSPACE}$

Übersicht Hierarchiesätze mit Beweisskizzen

Aussagen des Buches, 5. Auflage (Beweisidee in Klammern) Überall ist $s = s(n)$ und $t = t(n)$.

In Grau stehen Beweise, die nicht im Buch oder Übungen ausgeführt sind.

Lemma 6.1 Zu jeder k-Band-TM A existiert äquivalente 1-Band-TM B mit $\text{Space}_B(n) \leq \text{Space}_A(n)$.
Beweisskizze: Siehe Lemma 4.2: Ein Symbol in Γ_B ist $2(k+1)$ dimensionales Tupel mit Eingabesymbol aus Σ_A und k Arbeitsbandsymbolen aus Γ_A und dazwischen $k+1$ Symbolen (entweder Leerzeichen oder Pfeil) zur Markierung der Kopfpositionen. (Plus noch Leersymbol/Anfangs-/Endmarksymbole und Ursprungsalphabet aus definitionstechnischen Gründen, unwichtig.) Pro Schritt von A mit B dann einmal über das ganze Band hinweg, um sich die k Arbeitsbandsymbole bei den markierten Positionen zu merken. (Sich merken bedeutet hier in den Zuständen speichern). Dann simuliert δ_B das δ_A. Das Ergebnis merkt sich B wieder und schreibt es dann im Zurückgehen wieder hin. Insgesamt B also $2 \cdot \text{Space}_A(n) \leq 2 \cdot \text{Time}_A(n)$ Schritte für die Simulation eines Schrittes von A. Da höchstens $\text{Time}_A(n)$ Schritt zu simulieren sind, also $\text{Time}_B(n) \in O((\text{Time}_A(n))^2)$.

Lemma 6.2 Zu jeder MTM A existiert äquivalente MTM B mit $\text{Space}_B(n) \leq \frac{\text{Space}_A(n)}{2} + 2$.
Beweisskizze: Fasse zwei Felder zu einem zusammen. Plus 2 fürs Aufrunden an beiden Rändern

Aufgabe 6.1 Zu jeder MTM A existiert äquivalente MTM B mit $\text{Time}_B(n) \leq \frac{\text{Time}_A(n)}{2} + 2n$.
Beweisskizze: Fasse jede 12 Felder zu einem zusammen, dann 12 Schritt in 6 simulieren: 4 zum Lesen des aktuellen Felds und der beiden Nachbarfelder und zurückkehren, dann die 12 Schritte simulieren, dann nochmals 2 um Ergebnis hinzuschreiben. Plus $2n$ fürs Kopieren des Eingabebandes auf spezielles Arbeitsband in komprimierter Form und Zurücksetzen des Kopfes.

Satz 6.1 Es existiert ein Sprache, sodass zu jeder MTM A mit $L(M)$ eine äquivalente MTM B existiert mit $\text{Time}_B(n) \leq \log \text{Time}_A(n)$.
Beweisskizze: Erweitertes Diagonalargument, Filterung auf MTMs mit immer strengerer Zeitschranke mit wachsender Eingabelänge. Tabellieren für alle Wörter bis zu einer konstanten Länge ermöglicht dann beliebige Beschleunigung mittels Vergrössern der Konstante.

Lemma 6.3 $\text{TIME}(t) \subseteq \text{SPACE}(t)$
Beweisskizze: In t Schritten höchstens t Felder beschreibbar.

Korollar 6.1 $\text{P} \subseteq \text{PSPACE}$
Beweisskizze: Lemma 6.3 mit $t(n) = n^k$ für alle $k \in \mathbb{N}$, dann Vereinigung bilden.

Lemma 6.4 Sei s platzkonstruierbar. Zu jeder MTM, welche $\text{Space}_M(w) \leq s(|w|)$ nur für alle $w \in L(M)$ erfüllt, existiert äquivalente M', welche dies für alle $w \in \Sigma^*$ erfüllt.
Beweisskizze: Erzeuge $0^{s(n)}$ auf separatem Band, nutze das dann zur Platzüberwachung, Verwerfen bei Überschreitung.

Lemma 6.5 Sei t zeitkonstruierbar. Zu jeder MTM, welche $\text{Time}_M(w) \leq t(|w|)$ nur für alle $w \in L(M)$ erfüllt, existiert äquivalente M', welche zumindest $\text{Time}_M(w) \leq 2t(|w|) \in O(t(|w|))$ für alle $w \in \Sigma^*$ erfüllt.
Beweisskizze: Erzeuge $0^{t(n)}$ auf separatem Band in Zeit höchstens $t(|w|)$, nutze das zur Zeitzählung, Verwerfen bei Überschreitung.

Satz 6.2 $\text{SPACE}(s) \subseteq \bigcup_{c \in \mathbb{N}} \text{TIME}(c^s)$ für $s \geq \log n$.
Beweisskizze: Gegebene TM hat wegen Platzbeschränkung höchstens $O(n \cdot c^s)$ innerer Konfiguration, also alle endliche Berechnungen höchstens so lange. Denn sonst wäre Pumping-Lemma anwendbar und Berechnung doch unendlich lange, Widerspruch.
Somit alle Berechnungen so kurz, denn alle Berechnungen sind endlich gemäss versteckter Veraussetzung in der Definition von Platzkomplexität, siehe Beginn von Definition 6.2, S. 169.
Wegen $s \geq \log n$ kann man auch c^s statt $n \cdot c^s$ schreiben.

Korollar 6.2.(i) $\text{DLOG} \subseteq \text{P}$
Beweisskizze: Satz 6.2 mit $s(n) = \log n$.

Korollar 6.2.(ii) $\text{PSPACE} \subseteq \text{EXPTIME}$
Beweisskizze: Satz 6.2 mit $s(n) = n^k$ für alle $k \in \mathbb{N}$, Vereinigung.

Satz 6.3* $\text{SPACE}(s_1) \subsetneq \text{SPACE}(s_2)$ für alle $s_1 \in o(s_2)$ mit $s_2(n) \geq \log n$ platzkonstruierbar.
Beweisskizze: Schwierig. Diagonlisierung mit Filterung auf TM, die nur Platz s_2 benötigen. Dann ist Simulation der via Eingabe gegebenen TM in Platz s_2 möglich durch Abbruchbedingung mittels Platzkonstruierbarkeit, während die Striktheit für platzsparendere TM durch das Diagonalargument folgt. Die Simulation benötigt Platz $\Omega(|Q|) \supseteq \Omega(\log n)$ Platz fürs Speichern der inneren Konfiguration, daher $s_2(n) \geq \log n$ verlangt. Wegen O-Notation noch ein Padding mit $\#0^k$ notwendig.

Satz 6.4* $\mathrm{TIME}(s_1 \log t_1) \subsetneq \mathrm{TIME}(t_2)$ für alle $t_1 \in o(t_2)$ mit $s(n) \geq \log n$ zeitkonstruierbar.
Beweisskizze: Noch schwieriger. Analog zu Beweis oben, aber zusätzlich noch Filterung auf TM mit kurzer Codierung $|\mathrm{Kod}(M)| \leq \sqrt{k}$, sodass der zusätzliche Faktor $\log t_1(n) \geq \log t_1(k)$ bei der Simulation zum Finden und Ausführen der Transitionen ausreicht.

Lemma 6.6.(i) $\mathrm{NTIME}(t) \subseteq \mathrm{NSPACE}(t)$
Beweisskizze: Eine schnellste akzeptierende Berechnung betrachten. In den höchstens t Schritten beschreibt sie höchstens t Felder.

Lemma 6.6.(ii) $\mathrm{NSPACE}(s) \subseteq \bigcup_{c\in\mathbb{N}} \mathrm{NTIME}(c^s)$ mit $s(n) \geq \log n$.
Beweisskizze: Betrachte eine kürzeste akzeptierende Berechnung. Wegen Platzbeschränkung durchläuft sie höchstens $O(nc^s)$ (und wegen $s \geq \log n$ also $O(c^s)$) innere Konfiguration. Denn sonst wäre Pumping-Lemma anwendbar und mit Herauspumpen ergäbe sich noch kürzere Berechnung, Widerspruch.
(Man braucht die Zeitbeschränkung nach Definition von nichtdeterministischen Komplexitätsklassen nur für akzeptierte Wörter.)

Satz 6.5.(i) $\mathrm{TIME}(t) \subseteq \mathrm{NTIME}(t)$
Beweisskizze: Jede TM ist auch eine NTM.

Satz 6.5.(ii) $\mathrm{SPACE}(s) \subseteq \mathrm{NSPACE}(s)$
Beweisskizze: Jede TM ist auch eine NTM.

Satz 6.5.(iii) $\mathrm{NTIME}(t) \subseteq \mathrm{SPACE}(t)$ für platzkonstruierbares $t(n) \geq \log n$.
Beweisskizze: Anzahl nichtdeterministischer Transitionsmöglichkeiten pro Schritt in k-Band-TM ist durch Konstante $r := |\mathcal{P}(Q \times (\Gamma \times \{\mathrm{L}, \mathrm{R}, \mathrm{N}\})^k\}| = 2^{|Q|(3|\Gamma|)^k}$ beschränkt. In t Schritten also t Mal r Entscheidungsmöglichkeiten. In Platz $O(t)$ kann man einen Kette von t Entscheidungen, die man treffen muss, auflisten. (Hierzu Platzkonstruierbarkeit von t notwendig.) In demselben Platz kann man auch alle r^t Möglichkeiten für die Entscheidungskette durchiterieren. Für jede Entscheidungskette kann man dann jeweils deterministisch die Berechnung für bis zu t Schritte simulieren und so eine allfällige akzeptierende Konfigurationen finden. Bei der Simulation reicht es, immer die nur die Startkonfiguration und die aktuelle Konfiguration gespeichert zu halten, also nochmals Platzverbrauch höchstens $O(t)$.
Die innere Konfiguration enthält noch Eingabekopfposition, was pro Konfiguration auch noch Platz bis zu $|\mathrm{Bin}(n)| = \log n$ beansprucht. Das liegt wegen $t(n) \geq \log n$ aber immer noch in $O(t)$.
Aufgabe 6.15: Exponentielle Zeit; die r^t Iterationen dominieren.

Aufgabe 6.16: Die einfachere Determinisierung aus Satz 4.2. braucht auch solchen exponentiellen Speicherplatz, da jeweils alle Konfigurationen einer Ebene gleichzeitig auf einem Band zu stehen kommen.

Korollar 6.3 NP $\subseteq$ PSPACE
Beweisskizze: Satz 6.5.(iii) mit $t(n) = n^k$ für alle $k \in \mathbb{N}$, Vereinigung.

Satz 6.6* NSPACE$(s) \subseteq \bigcup_{c\in\mathbb{N}}$ TIME(c^s) für platzkonstruierbares $s(n) \geq \log n$.
Beweisskizze: Wegen Platzkonstruierbarkeit von s können wir annehmen, dass alle Berechnung einen Platzverbrauch von höchstens $O(s)$ haben; richten dazu Platzüberwachung ein, die bei Überschreitung der Schranke die Berechnung abbricht.
Betrachte für beliebiges Eingabewort eine kürzeste Berechnungen auf einem Wort. Da sie höchstens $O(s)$ Platz beschreibt, durchläuft sie höchstens $O(c^s)$ innere Konfigurationen.
(Eigentlich $O(n \cdot c^s)$, aber man benützt das gegebene $s \geq \log n$.)
Ansonsten wäre mit Pumping-Lemma ein Teil entfernbar, in Widerspruch zur Minimalität der Berechnung.
Zu gegebener TM gibt es äquivalente, die vor Abschluss einfach Arbeitsbänder löscht und damit genau eine akz. Konfiguration hat.
Erstelle dann Graphen mit den $O(c^s)$ inneren Konfigurationenen als Knoten und den Transitionen als gerichtete Kanten. Dann Weg von Startkonfiguration zur akzeptierenden Suchen, beispielsweise durch Darstellen mit Adjazenzmatrix und c^s-faches Potenzieren. Jede innere Konfiguration enthält noch die Eingabekopfposition, was pro Konfiguration zusätzlichen Platz von bis $|\text{Bin}(n)| = \log n$ beansprucht. Das ist wegen $t(n) \geq \log n$ aber immer noch in $O(s)$.

Korollar 6.4.(i) NLOG $\subseteq$ P

Beweisskizze: Aus Satz 6.6 mit $s(n) = \log n$.

Korollar 6.4.(ii) NSPACE $\subseteq$ EXPTIME
Beweisskizze: Satz 6.6 mit $s(n) = n^k$ für alle $k \in \mathbb{N}$, Vereinigung.

Satz 6.7* $\mathrm{NSPACE}(s) \subseteq \mathrm{SPACE}(s^2)$ für platzkonstruierbares $s \geq \log n$.
Beweisskizze: Beginn genau wie Satz 6.6.:
Wegen Platzkonstruierbarkeit von s können wir annehmen, dass alle Berechnung einen Platzverbrauch von höchstens $O(s)$ haben; richten dazu Platzüberwachung ein, die bei Überschreitung der Schranke die Berechnung abbricht.
Betrachte für beliebiges Eingabewort eine kürzeste Berechnungen auf einem Wort. Da sie höchstens $O(s)$ Platz beschreibt, durchläuft sie höchstens $O(c^s)$ innere Konfigurationen.
(Eigentlich $O(n \cdot c^s)$, aber man benützt das gegebene $s \geq \log n$.)
Ansonsten wäre mit Pumping-Lemma ein Teil entfernbar, in Widerspruch zur Minimalität der Berechnung.
Zu gegebener TM gibt es äquivalente, die vor Abschluss einfach Arbeitsbänder löscht und damit genau eine akz. Konfiguration hat.
Eine kürzeste Berechnung nie länger als Anzahl Konfigurationen, also $nd^s \leq c^s$ wegen $s \geq \log n$. Wegen Platzkonstruierbarkeit ein Zähler dafür umsetzbar in Platz s, nämlich ein c-ärer.
O.B.d.A. genau eine akzeptierende Konfiguration, dazu am Ende einfach alles löschen und Köpfe zurücksetzen.
Weg von C_{start} nach C_{accept} suchen. Bis hierhin wie Satz 6.6.
Jetzt beantwortet REACHABLE(C,D,m) (anstelle Potenzieren der Adjazenmatrix), ob man in m Schritten von Konfiguration C nach D kommt. Eine for-Schleife über jede mögliche Zwischenkonfigurationen Z mit je rekursivem Aufruf von $\mathrm{REACHABLE}(C, Z, \lceil m/2 \rceil)$ und $\mathrm{REACHABLE}(C, Z, \lceil m/2 \rceil)$.
(Die for-Schleife durchläuft immmer *alle* Konfigurationen, inklusive C und D; das ist wichtig für den Fall, dass beispielseise schon drei Konfigurationsschritte von C_{start} zu C_{accept} führen.)
Verankerung, wenn der letzte Parameter 1 ist (Blatt des Rekursionsbaumes): Dann einmal alle Nachfolgekonfigurationen von C in Platz $O(s)$ durchiterieren und schauen, ob so D erreicht wird.
Uns interessiert die Rückgabe von $\mathrm{REACHABLE}(C_{\text{start}}, C_{\text{accept}}, c^s)$.
Ergibt einen c^s-ären Rekursionsbaum der Tiefe höchstens $\log c^s$ wegen Halbierung des letzten Parameters in jedem Schritt. Für jede Ebene des Baumes reicht es eine Konfiguration gespeichert zu halten, also insgesamt $O(s) \cdot O(\log c^s) = O(s^2)$ Speicherplatz.
Hätte die betrachtete TM zusätzlich irgendeine Zeitbeschränkung von $O(t(n))$, dann wäre der Platzbedarf am Ende $O(s) \cdot O(\log t)$.

Korollar 6.5 $\mathrm{PSPACE} = \mathrm{NPSPACE}$
Beweisskizze:
„$\subseteq$“: Jede TM ist auch eine NTM mit je genau einer Transitionsmöglichkeit.
„$\supseteq$“: Satz 6.7 mit $s(n) = n^k$ für alle $k \in \mathbb{N}$, dann Vereinigung bilden.