Fakt 4.21

$f: \{0,1\}^* \to \{0,1\}^*$ ist genau dann implizit logspace-berechenbar, wenn f write-once logspace-berechenbar ist.

Beweis: Übung!

Definition 4.22 (logspace-Reduktionsbarkeit)

Seien $B, C \subseteq \{0,1\}^*$

B ist logspace-reduzierbar auf C (kurz: $B \leq_e C$), falls es eine implizit logspace-berechenbare Funktion $f: \{0,1\}^* \to \{0,1\}^*$ gibt, so dass f a. $x \in \{0,1\}^*$ gilt:

$$x \in B \iff f(x) \in C.$$

Lemma 4.23

(a) $\leq_e$ ist transitiv, d.h. f.a. $B, C, D \subseteq \{0,1\}^*$ gilt:
Falls $B \leq_e C$ und $C \leq_e D$, so $B \leq_e D$.

(b) L ist abgeschlossen unter logspace-Reduktionen, d.h. f.a. $B, C \subseteq \{0,1\}^*$ gilt:
Falls $B \leq_e C$ und $C \in L$, so auch $B \in L$.

(c) NL ist abgeschlossen unter logspace-Reduktionen, d.h.: f.a. $B, C \in \{0,1\}^*$ gilt: Falls $B \leq_e C$ und $C \in NL$, so auch $B \in NL$.

Beweis:

(a)+(b): Seien $f, g: \{0,1\}^* \to \{0,1\}^*$ implizit logspace-berechenbar durch TMen M_f, M'_f, M_g, M'_g (für L_f, L'_f, L_g, L'_g).

Wir zeigen im Folgenden, dass die Funktion
$h: \{0,1\}^* \to \{0,1\}^*$ mit $h(x) := g(f(x))$ f.a. $x \in \{0,1\}^*$
implizit logspace-berechenbar ist

Beachte: Daraus folgt dann direkt (a), und (b) folgt auch, indem wir annehmen, dass f die Funktion ist, die zeigt, dass $B \leq_L^c C$ ist und indem wir $g: \{0,1\}^* \to \{0,1\}$ so wählen, dass $g(x) := \begin{cases} 1 & \text{falls } x \in B \\ 0 & \text{sonst} \end{cases}$ (Hier: wegen $C \in L$ ist g implizit logspace-berechenbar). Die Funktion h hat dann die Eigenschaft, dass $h(x) = \begin{cases} 1 & \text{falls } x \in C \\ 0 & \text{sonst} \end{cases}$ — und die implizite logspace-Berechenbarkeit von h zeigt, dass C in L liegt.

Zum Nachweis der impliziten logspace-Berechenbarkeit von h:

(1) h ist polynoriell beschränkt, da $h(x) = g(f(x))$ (f.a. $x \in \{0,1\}^*$) und da f und g polynoriell beschränkt sind

(2) Sei $L_h := \{ \langle x, j \rangle : x \in \{0,1\}^*, j \in \mathbb{N}, h(x)_j = 1 \}$ und
$L'_h := \{ \langle x, j \rangle : x \in \{0,1\}^*, j \in \mathbb{N}, |h(x)| \leq j \}$

Zu zeigen: $L_h \in L$ und $L'_h \in L$

Wir zeigen im Folgenden, dass $L_h \in L$ ist
($L'_h \in L$ kann auf ähnliche Weise gezeigt werden; Details: Übung!)

Wir konstruieren eine DTM M_h, die L_h auf logarithmischem Platz entscheidet. Dabei stellen wir uns vor, dass M_h ein "virtuelles Band" besitzt, auf das sie

bei Eingabe $\langle x,j\rangle$ das Wort $f(x)$ schreibt. Danach wird das virtuelle Band als Eingabeband betrachtet, für das M_g simuliert wird, um zu entscheiden, ob $\langle f(x),j\rangle \in L_g$ liegt (d.h. ob $h(x)_j \overset{\text{Def}}{=} g(f(x))_j = 1$ ist)

Problem: M_h hat nicht genug Platz zur Verfügung, um das gesamte Wort $f(x)$ auf einem Band zu speichern.

Lösung: Stattdessen merkt sich M_h auf einem extra Band die aktuelle Position i, die auf dem virtuellen Band gelesen werden soll und nützt in jedem einzelnen Schritt von M_g die TMen M_f und M'_f, um das auf dem virtuellen Band zu lesende Symbol zu ermitteln: Dieses ist ▷ falls $i=0$; 1 falls $\langle x,i\rangle \in L_f$ (da dann $f(x)_i=1$); □ falls $\langle x,i\rangle \notin L'_f$ (da dann $i>|f(x)|$); und 0 falls $\langle x,i\rangle \in L'_f$ und $\langle x,i\rangle \notin L_f$ (da dann $i\leq |f(x)|$ und $f(x)_i \neq 1$ – also $f(x)_i=0$).

Platzbedarf von M_h:

- $\sigma(\log(|f(x)|)$ Platz zur Simulation von M_g bei Eingabe $f(x)$
  $\sigma(\log(\text{poly}(|x|)) = \sigma(\log|x|)$
- $\sigma(\log(|f(x)|)$ Platz zum Speichern der aktuellen Position i auf dem virtuellen Band
- $\sigma(\log(|\langle x,i\rangle|))$ Platz zur Simulation von M_f und M'_f bei Eingabe $\langle x,i\rangle$
  $\sigma(\log|x|)$

Insgesamt zeigt dies, dass $L_h \in \mathbf{L}$ ist.

Definition 4.24

Sei $C \subseteq \{0,1\}^*$.

(a) C ist NL-hart, falls f.a. $B \in NL$ gilt: $B \leq_e C$.

(b) C ist NL-vollständig, falls $C \in NL$ und C NL-hart ist.

Bemerkung 4.25:

Aus Lemma 4.23 (b) folgt:

Falls es ein NL-vollständiges Problem gibt, das in L liegt, so ist $L = NL$

Theorem 4.26

Das folgende Problem ist NL-vollständig:

$$PATH \overset{Def}{=} \{ \langle G,s,t \rangle : G \text{ ist ein gerichteter endlicher Graph, in dem es einen Weg von Knoten } s \text{ zu Knoten } t \text{ gibt} \}$$

Beweis:

PATH $\in NL$ wurde schon in Beispiel 4.8(c) gezeigt.

NL-Härte von PATH:

Sei $B \subseteq \{0,1\}^*$ ein beliebiges Problem in NL und sei M eine NDTM, die B auf Platz $O(\log n)$ entscheidet.

Zu zeigen: $B \leq_e PATH$.

Sei dazu $f : \{0,1\}^* \to \{0,1\}^*$ die Funktion mit

$f(x) := \langle G_{M,x}, c_{start}, c_{accept} \rangle$ (f.a. $x \in \{0,1\}^*$).

Dann: $x \in B \iff f(x) \in PATH$.

f ist write-once logspace-berechenbar durch einen Algorithmus, der nacheinander alle Paare C, C' von Konfigurationen von M bei Eingabe x betrachtet (d.h. alle Knoten von $G_{M,x}$) und testet, ob M bei Eingabe x in einem Schritt von Konfiguration C in Konfiguration C' kommt — und entsprechend eine 1 oder eine 00 auf's Ausgabeband schreibt.
Auf diese Weise schreibt der Algorithmus die Kodierung der Adjazenzmatrix von $G_{M,x}$ auf's Ausgabeband. Danach schreibt er noch die Kodierungen von C_{start} und C_{accept} und hält an.

Platzbedarf: In jedem Schritt muss dieser Algorithmus 2 Konfigurationen von M bei Eingabe x speichern — und das geht auf Platz $O(\log |x|)$.

Insgesamt ist f daher write-once logspace-berechenbar und wegen Fakt 4.21 daher auch implizit logspace-berechenbar. Somit ist $B \leq_L PATH$. □

4.4 Nichtdeterministische Platzklassen sind unter Komplementbildung abgeschlossen

Zur Erinnerung:

- Für eine Komplexitätsklasse K ist

$$coK := \{ L \subseteq \{0,1\}^* : \overline{L} \in K \} = \{ \overline{L} : L \in K \}$$

- Vermutung: $NP \neq coNP$

- Bekannt: $NPSPACE \underset{\text{satzh}}{=} PSPACE \underset{\text{deterministisch}}{=} coPSPACE = coNPSPACE$

Theorem 4.27 (Der Satz von Immerman und Szelepcsényi, 1987)

(a) $\overline{PATH} \in NL$

(b) $NL = coNL$

(c) Für jedes platzkonstruierbare $S: \mathbb{N} \to \mathbb{N}$ mit $S(n) \geq \log n$ gilt: $NSPACE(S) = coNSPACE(S)$

Beweis: (a) Gesucht ist ein nichtdeterministischer Algorithmus B, der auf Platz $O(\log n)$ arbeitet und der bei Eingabe von $\langle G, s, t \rangle$ folgendes Verhalten hat:

(I) falls es in G einen Weg von s nach t gibt, dann soll der Algorithmus bei *jeder* Folge von nichtdet. Entscheidungen ablehnen

(II) falls es in G keinen Weg von s nach t gibt, dann soll es [illegible] bei der Alg. akzeptiert

Behauptung 1:

Es gibt einen ndet. Algo. A, der auf Platz $O(\log n)$ arbeitet und der bei Eingabe von $\langle G, s\rangle$ folgendes Verhalten hat.

- für jede Folge nichtdeterministischer Entscheidungen gibt A entweder eine Fehlermeldung (kurz: $\perp$) aus, oder die Anzahl r der von s aus in G erreichbaren Knoten (d.h. $r = |\{v : v$ ist ein Knoten von G, und es gibt in G einen Weg von s nach $v\}|$),
- es gibt mindestens eine Folge nichtdeterministischer Entscheidungen, so dass A die Zahl r ausgibt.

Bevor wir Beh 1 beweisen, zeigen wir, wie Algo A genutzt werden kann, um den gesuchten Algo B zu erhalten, der zeigt, dass $\overline{PATH} \in NL$ ist.

B geht bei Eingabe $\langle G, s, t\rangle$ wie folgt vor:

1) Lass A mit Eingabe $\langle G, s\rangle$ laufen.
   Falls A eine Fehlermeldung ausgibt, so STOPP mit Ausgabe "nein"
   Sonst sei r die Ausgabe von A (d.h. r ist die Anzahl der von s aus in G erreichbaren Knoten)
2) Zähler := 0; Weg-zu-t-gefunden := FALSE
3) Für jeden Knoten v von G tue folgendes:
   Rate einen in s startenden Weg der Länge $\leq n$ ($= |V(G)|$)
   Falls der Weg mit Knoten v endet, so:
   Zähler := Zähler + 1
   Falls $v = t$, so Weg-zu-t-gefunden := TRUE
4) Falls Zähler $= r$ und Weg-zu-t-gefunden = FALSE, so
   STOPP mit Ausgabe "ja"

Man kann sich leicht davon überzeugen, dass B die geforderten Eigenschaften (I) und (II) hat, und dass B auf Platz $O(\log n)$ arbeitet.

Wir müssen also "nur noch" die Behauptung 1 beweisen.

Für jedes $i \leq n = |V(G)|$ sei r_i die Anzahl der von s aus über Wege der Länge $\leq i$ erreichbaren Knoten (insbes: $r_0 = 1$ (nämlich s selbst) und $r_n = r$).

Algo A geht bei Eingabe $\langle G, s \rangle$ wie folgt vor:

```
r_0 := 1 ;  Fehlermeldung := FALSE ;
for i := 1, 2, ..., n do
    Berechne r_i unter Verwendung von r_{i-1}        (1)
Falls Fehlermeldung = TRUE, so
    gib Fehlermeldung aus
sonst gib r := r_n aus.
```

(1) wird dabei wie folgt realisiert:

```
Zähler := 0
Für jeden Knoten v von G do:
    Teste, ob es in G einen Weg der Länge ≤ i von s nach v gibt   (2)
        Falls ja, so Zähler := Zähler + 1
r_i := Zähler
```

② wird dabei wie folgt realisiert:

```
Zähler2 := 0;
Weg-zu-v-gefunden := FALSE
Für jeden Knoten u von G do:
    Teste, ob es in G einen Weg der Länge ≤ i-1 von s nach u gibt (3)
    Falls ja, so
        Zähler2 := Zähler2 + 1
        Falls (u=v oder es in G eine Kante von u zu v gibt), so
            Weg-zu-v-gefunden := TRUE
Falls Zähler2 ≠ r_{i-1}, so
    Fehlermeldung := TRUE
Ausgabe: der Wert der Variablen "Weg-zu-v gefunden"
```

③ wird dabei wie folgt realisiert:

```
Falls u=s, so
    STOPP mit Ausgabe "ja"
Sonst:
    w := s
    For j := 1, ..., i-1 do
        Rate einen Knoten w' von G
        Falls es keine Kante von w nach w' gibt, so
            Fehlermeldung := TRUE
        Sonst
            w := w'
            Falls w=u, so
                STOPP mit Ausgabe "ja"
```

Per Induktion nach $i = 1, 2, \ldots, n$ lässt sich zeigen:
Ist r_{i-1} korrekt berechnet worden, so wird bei der Berechnung von r_i entweder der korrekte Wert von r_i berechnet oder die Variable "Fehlermeldung" auf TRUE gesetzt, und

Platzbedarf des Algorithmus:

Speichern von s, i, r_{i-1}, Zähler, v, Zähler2, u, w, j, w' sowie der Booleschen Variablen "Fehlermeldung" und "Weg-zu-v-gefunden".

Insgesamt geht das auf Platz $O(\log n)$.

Damit ist der Beweis von Behauptung 1 fertig, und wir haben gezeigt, dass $\overline{\text{PATH}} \in NL$ ist.

□(a)

(b): Zu zeigen: $NL = coNL$

"$\subseteq$":

Sei $B \in NL$ beliebig. Dann ist $B \leq_L \text{PATH}$, da PATH NL-vollständig ist (↗ Theorem 4.26).

Somit ist auch $\overline{B} \leq_L \overline{\text{PATH}}$.

Wegen $\overline{\text{PATH}} \in NL$ (↗(a)) folgt, dass $\overline{B} \in NL$, da NL unter logspace-Reduktionen abgeschlossen ist (↗Lemma 4.23(c))

Somit ist $B \in coNL$. Also: $NL \subseteq coNL$

"$\supseteq$": Sei $C \in coNL$, d.h. $\overline{C} \in NL$. Wegen $NL \subseteq coNL$ ist dann $\overline{C} \in coNL$, also $C = \overline{\overline{C}} \in NL$. Also $coNL \subseteq NL$.

□(b)

(c): Analog zu (a), wobei an Stelle von $\langle G, s, t \rangle$ betrachtet wird, für eine $S(n)$-platzbeschränkte NTM M und eine Eingabe x für M

4.5 Platzklassen und die Chomsky-Hierarchie

Zur Erinnerung: Die Chomsky-Hierarchie

- Typ 0-Grammatiken: Grammatiken ohne irgendeine Einschränkung
- Typ 1-Grammatiken: kontextsensitive Grammatiken, d.h. Grammatiken, deren Produktionen von der Form $u \to v$ mit $|u| \leq |v|$ sind
- Typ 2-Grammatiken: kontextfreie Grammatiken, d.h. Grammatiken, deren Produktionen von der Form $A \to v$ sind, wobei A ein Nichtterminal ist.
- Typ 3-Grammatiken: reguläre Grammatiken

Aus der Veranstaltung "GL-2: Formale Sprachen und Berechenbarkeit" ist bekannt:

- Typ 0-Grammatiken können genau die rekursiv aufzählbaren (d.h. semi-entscheidbaren) Sprachen erzeugen
- Typ 1-Grammatiken, d.h. kontextsensitive Grammatiken, können genau die Sprachen in NSPACE(n) erzeugen.
  Aus dem Satz von Immerman und Szelepcsényi wissen wir: NSPACE(n) = coNSPACE(n). Daher sind die kontextsensitiven Sprachen unter Komplementbildung abgeschlossen.

- Für Typ 2-Grammatiken, d.h. kontextfreie Grammatiken, lässt sich das Wortproblem (Eingabe: ein Wort w
Frage: Wird w von der Grammatik erzeugt?)
mittels des CYK-Algorithmus in Zeit $O(n^3)$ lösen. Somit gilt für die Menge KFG aller kontextfreien Sprachen:

$$KFG \subseteq DTIME(n^3) \subsetneq P$$

Außerdem ist bekannt (hier ohne Beweis), dass

$$KFG \subseteq SPACE((\log n)^2)$$

(Lewis, Stearns, Hartmanis, 1965)

Bzgl Typ 3-Grammatiken, d.h. reguläre Grammatiken, zeigen wir:

Satz 4.28
Für jedes $S: \mathbb{N} \to \mathbb{N}$ mit $S(n) = o(\log \log n)$ und jedes $L \in SPACE(S)$ gilt: L ist regulär.
Somit ist $SPACE(S) = SPACE(0) = \{L \subseteq \{0,1\}^* : L \text{ regulär}\}$.
(kurz: "$SPACE(o(\log\log n)) = REG$")

Beweis:
Sei $S: \mathbb{N} \to \mathbb{N}$ und sei
$L \in SPACE(S)$.
Sei M eine $S(n)$-platzbeschränkte DTM, die L entscheidet.
Wegen "lineare Kompression" (Satz 4.9) können wir OBdA annehmen, dass M eine 2-Band DTM ist.

Behauptung 1:
Falls es ein $k \in \mathbb{N}$ gibt mit $S(n) \leq k$ f.a. $n \in \mathbb{N}$, so ist L regulär.

Beweis:
Wegen $S(n) \leq k$ f.a. $n \in \mathbb{N}$, kann der gesamte Inhalt des Arbeitsbandes auch in einer konstanten Anzahl von Zuständen gespeichert werden.
Daher können wir OBdA annehmen, dass M eine 1-Band DTM mit read-only Eingabeband ist.
D.h. M ist ein sog. deterministischer 2-Wege-Automat.

Solche det. 2-Wege-Automaten lassen sich durch herkömmliche NFAs (ndet. endliche Automaten) simulieren. Details: Übung!

▷ Beh 1.

Sei nun $S(n) = o(\log \log n)$.

Angenommen, L ist nicht regulär.

Wegen Beh. 1 muss es dann für jedes $k \in \mathbb{N}$ ein Wort $x^{(k)} \in \{0,1\}^*$ geben, bei dessen Eingabe M mehr als k Zellen des Arbeitsbandes benutzt.
Sei $x^{(k)}$ ein Wort minimaler Länge, für das dies gilt, und sei $n_k := |x^{(k)}|$ die Länge von $x^{(k)}$.

Klar: $n_1 \leq n_2 \leq n_3 \leq \dots$ und $n_k \xrightarrow[k \to \infty]{} \infty$.

Wir betrachten im Folgenden für jedes $x \in \{0,1\}^*$ die Berechnung $B_M(x)$, die als die Folge von Konfigurationen definiert ist, die M bei Eingabe x durchläuft.

Hier:

Jede Konfiguration C besteht aus

- dem aktuellen Zustand von M
- der aktuellen Kopfposition i auf dem Eingabeband
- dem aktuell auf dem Eingabeband gelesenen Symbol $x_i \in \{0,1,\triangleright,\square\}$
- der aktuellen Kopfposition auf dem Arbeitsband
- der aktuellen Beschriftung des Arbeitsbandes

Bei einer Eingabe x der Länge n gibt es für jede feste Kopfposition i auf dem Eingabeband höchstens

$$N_n := |Q| \cdot 4 \cdot S(n) \cdot |\Gamma|^{S(n)}$$

verschiedene Konfigurationen, wobei Q und Γ die Zustandsmenge und das Arbeitsalphabet von M sind. Somit gibt es eine Zahl $d \in \mathbb{N}$ so dass

(*) $N_n \leq 2^{d \cdot S(n)}$ f.a. hinreichend großen $n \in \mathbb{N}$ gilt

Für jede Position i auf dem Eingabeband sei

$$C_{i,M}(x)$$

die Teilfolge der Berechnung $B_M(x)$, die aus allen Konfigurationen besteht, bei denen der Kopf des Eingabebands auf Position i steht.

$C_{i,M}(x)$ wird <u>Crossing-Sequenz für Position i</u> genannt.

<u>Klar:</u> In $C_{i,M}(x)$ kommt keine Konfiguration mehrfach vor, denn sonst würde die DTM M bei Eingabe x in eine Endlosschleife kommen – da M aber die Sprache L entscheidet muss M bei jeder Eingabe x irgendwann anhalten.

Es gilt für die Länge $|C_{i,M}(x)|$ von $C_{i,M}(x)$:

$$|C_{i,M}(x)| \leq N_n \quad \text{, für } n := |x|$$

Außerdem gilt für jedes $\ell \in \mathbb{N}$:
Die Anzahl der möglichen Crossing-Sequenzen (für Pos. i) der Länge ℓ ist $\leq N_n^\ell$.

Somit gilt:

(**) Die Gesamtzahl möglicher Crossing-Sequenzen für jedes feste i ist

$$\leq \sum_{\ell=0}^{N_n} N_n^\ell < N_n^{N_n+1} = 2^{(\log N_n)\cdot(N_n+1)} \overset{(*)}{\leq} 2^{d\cdot S(n)\cdot(2^{d\cdot S(n)}+1)}$$

$$\leq 2^{2^{d'\cdot S(n)}}$$

für ein geeignetes $d' \in \mathbb{N}$ und alle hinreichend großen $n \in \mathbb{N}$.

Sei $C'_{i,M}(x)$ die Folge, die aus $C_{i,M}(x)$ entsteht, indem in jeder Konfiguration die Information über die Kopfposition i gelöscht wird.

Behauptung 2:
Sei $x \in \{0,1\}^*$, $n := |x|$ und sei $1 \leq i_1 < i_2 \leq n$ so dass
$C'_{i_1,M}(x) = C'_{i_2,M}(x)$.
Dann gilt für $x' := x_1 \cdots x_{i_1} x_{i_2+1} \cdots x_n$ (d.h.: x' ist das Wort, das aus x entsteht, indem das Teilwort $x_{i_1+1} \cdots x_{i_2}$ gelöscht wird):
$B_M(x')$ ist die Konfigurationenfolge, die aus $B_M(x)$ entsteht, indem

- alle Konfigurationen, bei denen die Kopfposition auf dem Eingabeband eine Position aus $\{i_1+1, \ldots, i_2\}$ ist, gelöscht werden, und
- bei allen Konfigurationen, bei denen die Kopfposition auf dem Eingabeband eine Zahl der Form i_2+j, für $j \geq 1$ ist, diese Zahl ersetzt wird durch die Zahl i_1+j.

Insbes. gilt: M akzeptiert $x' \iff M$ akzeptiert x.

Beweis: Übung!

Um den Beweis von Satz 4.28 zu beenden sei nun für jedes $k \in \mathbb{N}$ $x^{(k)} \in \{0,1\}^*$ ein Wort minimaler Länge, bei dessen Eingabe M mehr als k Zellen des Arbeitsbandes benützt.

D.h. in $B_M(x^{(k)})$ gibt es mindestens eine Konfiguration C, für die die Beschriftung des Arbeitsbandes die Länge $> k$ hat. Sei j die Position des Kopfes auf dem Eingabeband bei C.

Dann gilt:

(1) Die Crossing-Sequenzen $C'_{i,M}(x^{(k)})$ für alle $i \leq j$ müssen alle paarweise verschieden sein, und

(2) die Crossing-Sequenzen $C'_{i,M}(x^{(k)})$ für alle $i \geq j$ müssen alle paarweise verschieden sein.

Denn: Angenommen $C'_{i_1,M}(x^{(k)}) = C'_{i_2,M}(x^{(k)})$ mit $i_1 < i_2 \leq j$ oder $j \leq i_1 < i_2$. Dann gilt gemäß Behauptung 2 für $x' := x_1^{(k)} \dots x_{i_1}^{(k)} x_{i_2+1}^{(k)} \dots x_n^{(k)}$, dass $B_M(x')$ die Konfiguration C enthält (bzw. die Variante von C, bei der j ersetzt wurde durch $j - i_2 + i_1$).

D.h. insbesondere, dass M bei Eingabe x' mehr als k Zellen des Arbeitsbandes nutzt. Wegen $|x'| < |x^{(k)}|$ ist dies ein Widerspruch zur Wahl von $x^{(k)}$ als Wort minimaler Länge, bei dessen Eingabe M mehr als k Zellen des Arbeitsbandes nutzt.

Somit muss (1) und (2) gelten.

Insbesondere gibt es daher in $\{ C^{\mathcal{A}}_{i,n}(x^{(k)}) : 1 \le i \le |x^{(k)}| = n_k \}$

mindestens $\frac{n_k}{2}$ verschiedene Elemente (für $j \ge \frac{n_k}{2}$ folgt dies aus (1); für $j \le \frac{n_k}{2}$ folgt dies aus (2))

Mit (**) folgt: $\frac{n_k}{2} \le 2^{2^{d' \cdot S(n_k)}}$

Somit: $n_k \le 2 \cdot 2^{2^{d' \cdot S(n_k)}} < 2^{2^{d'' S(n_k)}}$

für ein geeignetes $d'' \in \mathbb{N}$ und alle hinreichend großen n_k

und daher: $\log\log n_k \le d'' \cdot S(n_k)$,

also $S(n_k) \ge \frac{1}{d''} \cdot \log\log n_k$

↳ Widerspruch zu $S(n) = o(\log\log n)$

Somit muss L regulär sein, und Satz 4.28 ist bewiesen. □

Bemerkung 4.29

Aus Satz 4.28 wissen wir, dass $\mathrm{SPACE}(o(\log\log n))$ genau die regulären Sprachen enthält.

$\mathrm{SPACE}(\log\log n)$ enthält auch nicht-reguläre Sprachen – z.B. die Sprache

$$L_{BIN} := \{ \langle bin_k(0), bin_k(1), \dots bin_k(2^k - 1) \rangle : k \in \mathbb{N} \}$$

wobei $bin_k(j)$ die Binärdarstellung der Länge k von j bezeichnet, für j mit $0 \le j < 2^k$.

(Dass L_{BIN} nicht regulär ist, folgt leicht aus dem Pumping-Lemma für reguläre Sprachen. Dass $L_{BIN} \in \mathrm{SPACE}(\log\log n)$ ist,

Bemerkung 4.30

Durch Betrachtung von Crossing-Sequenzen kann man für einige konkrete Sprachen untere Schranken für den Platz- oder Zeitbedarf zum Entscheiden dieser Sprachen beweisen.

z.B. kann man für die Sprache

$$PAL := \{ w \in \{0,1\}^* : w \text{ ist ein Palindrom} \}$$

zeigen:

(1) $PAL \notin SPACE(o(\log n))$

(d.h. es gibt kein $S: \mathbb{N} \to \mathbb{N}$ mit $S(n) = o(\log n)$ so dass $PAL \in SPACE(S(n))$

Zum Vergleich: $PAL \in L = SPACE(\log n)$

(2) PAL wird von keiner 1-Band DTM (mit Schreib-/Lesekopf auf dem Eingabeband) in $o(n^2)$ Schritten entschieden

Zum Vergleich: PAL kann in $O(n^2)$ Schritten von einer 1-Band DTM (mit Schreib-/Lesekopf auf dem Eingabeband) entschieden werden.
Und PAL kann in $O(n)$ Schritten von einer 2-Band DTM entschieden werden.

Details: Übung!

Überblick über die bisher betrachteten Komplexitätsklassen: