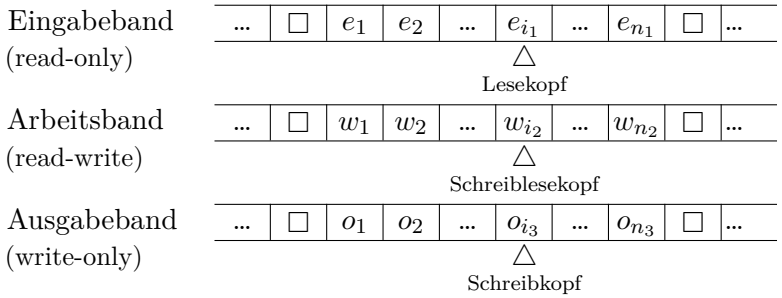
Proseminar Theoretische Informatik 26.01.2016

Komplexitätsklassen $\mathcal{L}$ und $\mathcal{NL}$

Anja Wolffgramm, Bianca George *Wolfgang Mulzer*

1 Einleitung

Die Ausführung eines Programms benötigt nicht nur Zeit, sondern auch Speicherplatz. Um den Platzbedarf von Algorithmen messen und vergleichen zu können, brauchen wir ein einheitliches Modell. Dazu nutzen wir weiterhin die Turingmaschine (TM). Der Platzverbrauch eines Algorithmus für eine konkrete Eingabe sei die Anzahl aller – während der Rechnung benutzten – Bandfelder (en. *tape cells*). Dabei wollen wir wissen, inwiefern der Platzbedarf in Abhängigkeit der Eingabegröße wächst. Um diesen zusätzlichen Platzverbrauch sinnvoll zu definieren, betrachten wir nicht die Ein- und Ausgabe und bedienen uns der 3-Band-Turingmaschine:



Definition 1. *3-Band-Turingmaschine*

Seien M_d eine deterministische *und M_n eine* nicht-deterministische *Turingmaschine mit M_d bzw. $M_n = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$, wobei*

- *Q eine endliche Zustandsmenge (set of conditions),*
- *Σ das Eingabealphabet,*
- *$\Gamma \supset \Sigma$ das Arbeitsalphabet,*
- *$\delta\colon Q \times \Gamma^3 \to Q \times \Gamma^3 \times \{L, R, N\}^3$ die Überführungsfunktion von M_d und*

  $\delta\colon Q \times \Gamma^3 \to \mathscr{P}\left(Q \times \Gamma^3 \times \{L, R, N\}^3\right)$ die von M_n,
- *q_0 der Startzustand,*
- *$\square \in \Gamma \setminus \Sigma$ das Blank-Zeichen und*
- *$F \subseteq Q$ die Menge der Endzustände (set of final conditions) ist.*

M_d und M_n halten, wenn $q \in F$ erreicht wurde.

Die Konfiguration einer TM beschreibt sowohl den aktuellen Zustand, in dem sie sich befindet, als auch die Position der (Schreib-)Leseköpfe des Eingabe- und Arbeitsbandes. Für unsere 3-Band-TM definieren wir die Notation der Konfiguration neu:

Definition 2. *Sei* $K = (q, e_{i_1}, w_1 w_2 \dots w_{i_2-1} \underset{\substack{\triangle \\ \text{Schreiblesekopf}}}{w_{i_2}} w_{i_2+1} \dots w_{n_2}) \in Q \times \Sigma \times \Gamma^*$ *eine Konfiguration, wobei*

- q *der aktuelle Zustand,*
- $e_{i_1} \in \Gamma$ *das Eingabeband-Symbol, auf welches der Lesekopf zeigt, und*
- $w_1 \in \Gamma$ *die am weitesten links und* $w_{n_2} \in \Gamma$ *die am weitesten rechts stehenden Arbeitsband-Symbole sind.*

2 Komplexitätsklasse $\mathcal{L}$

Wir messen den Speicherplatzbedarf von M_d in Abhängigkeit der Länge $|w|$ einer Eingabe w, indem wir die Anzahl der benutzten Zellen des Arbeitsbandes zählen, die zur Bearbeitung von w benötigt werden. Auf diese Weise ordnen wir jeder DTM mit Eingabealphabet Σ eine Speicherplatzfunktion

$$\text{space}_\text{D} : \Sigma^* \to \mathbb{N}_0$$

zu, die definiert ist durch

$$\text{space}_\text{D}(w) = \text{Anzahl der Bandspeicherplätze, die } M_d \text{ bei Eingabe } w \text{ beschreibt}$$

.

Wir wollen nun die Sprachen über Σ zu einer Komplexitätsklasse zusammenfassen, deren Wörter innerhalb einer bestimmten Platzbeschränkung von einer DTM akzeptiert werden.

Definition 3. *Sei* $f \colon \mathbb{N}_0 \to \mathbb{N}_0 \cup \{\infty\}$ *eine Funktion. Dann ist*

$$\text{SPACE}(f) = \{L \subseteq \Sigma^* \mid \exists\, M_d \; \forall w \in L : L(M_d) = L \wedge \text{space}_\text{D}(w) \leq f(|w|)\}$$

(syn. $\text{DSPACE}, \text{DTAPE}$*) die Menge aller Sprachen, deren Wörter* w *von einer DTM mit einem Speicherplatzbedarf von höchstens* $f(|w|)$ *Plätzen akzeptiert werden.*

Es gibt zwei bekannte Komplexitätsklassen für den Speicherplatz-Bedarf von DTM:

Definition 4.

$$\mathcal{L} = \text{SPACE}(\log n)$$

ist die Klasse der Sprachen, die von einer DTM mit logarithmisch beschränktem Speicherplatzbedarf akzeptiert werden.

Definition 5.

$$\text{PSPACE} = \bigcup_{k \geq 0} \text{SPACE}(n^k)$$

ist die Klasse der Sprachen, die von einer DTM mit polynomiell beschränktem Speicherplatzbedarf akzeptiert werden.

Offensichtlich gilt:

$$\mathcal{L} \subset \text{PSPACE}$$

Beispiel

Wir wollen zeigen, dass die Sprache $L := \{a^k b^k | k \in \mathbb{N}\} \in \mathcal{L}$.

Beweis. Wir verwenden eine DTM, welche die Sprache $\{a^k b^k | k \in \mathbb{N}\}$ akzeptiert.

Algorithmus:

1. Scanne die Eingabe w und überprüfe, ob alle a's vor den b's stehen.
2. Zähle die Anzahl der a's, indem auf dem Arbeitsband binär inkrementiert wird.
3. Dekrementiere die Binärzahl für jedes gelesene b.
4. Wenn der Wert des Arbeitsbandes Null ist und wir am Ende des Wortes w sind, schreibe eine Eins auf das Ausgabeband, sonst eine Null.

Speicherplatzbedarf: Die Anzahl an a's und b's ist höchstens $|w|$. Zum Speichern der Anzahl benötigen wir $\lceil \log |w| \rceil$ Bandzellen. Das Dekrementieren benötigt keinen zusätzlichen Speicher. Somit ist der Platzbedarf $\text{space}_\text{D}(|w|) \leq \log |w|$ und damit $L \in \mathcal{L}$. □

Eigenschaften

Abschlusseigenschaften: $\forall A, B \in \mathcal{L}$:

- Vereinigung: $A \cup B \in \mathcal{L}$
- Durchschnitt: $A \cap B \in \mathcal{L}$
- Komplement: $\bar{A} \in \mathcal{L}$

3 Komplexitätsklasse $\mathcal{NL}$

Wir messen den Speicherplatzbedarf einer NTM M_n in Abhängigkeit von der Länge $|w|$ einer Eingabe w, indem wir die Anzahl der benutzten Zellen des Arbeitsbandes zählen, die zur Bearbeitung von w benötigt werden. Bei nicht-deterministischen Rechnungen müssen alle Rechenwege betrachtet werden, da es bei den meisten Problemen notwendig sein kann, die gesamte Eingabe zu lesen.

Wir ordnen jeder NTM mit Eingabealphabet Σ eine Speicherplatzfunktion

$$\text{space}_\text{N} : \Sigma^* \to \mathbb{N}_0$$

zu, die definiert ist durch

$$\text{space}_\text{N}(w) = \text{Anzahl der Bandspeicherplätze, die } M_n \text{ bei Eingabe } w \text{ beschreibt}$$

.

Auch hier wollen wir nun die Sprachen über Σ zu einer Komplexitätsklasse zusammenfassen, deren Wörter innerhalb einer bestimmten Platzbeschränkung von einer NTM akzeptiert werden.

Definition 6. *Sei* $f\colon \mathbb{N}_0 \to \mathbb{N}_0 \cup \{\infty\}$ *eine Funktion. Dann ist*

$$\mathrm{NSPACE}(f) = \{L \subseteq \Sigma^* \mid \exists\, M_n\ \forall w \in L : L(M_n) = L \wedge \mathrm{space}_{\mathrm{N}}(w) \leq f(|w|)\}$$

(syn. NTAPE*) ist also die Menge aller Sprachen, deren Wörter* w *von einer nicht-deterministischen Turingmaschine mit einem Speicherplatzbedarf von höchstens* $f(|w|)$ *Plätzen akzeptiert werden.*

Es gibt zwei bekannte Komplexitätsklassen für den Speicherplatz-Bedarf von NTM:

Definition 7.

$$\mathcal{NL} = \mathrm{NSPACE}(\log n)$$

ist die Klasse der Sprachen, die von einer NTM mit logarithmisch beschränktem Speicherplatzbedarf akzeptiert werden.

Definition 8.

$$\mathrm{NPSPACE} = \bigcup_{k \geq 0} \mathrm{NSPACE}(n^k)$$

ist die Klasse der Sprachen, die von einer NTM mit polynomiell beschränktem Speicherplatz akzeptiert werden.

Offensichtlich gilt:

$$\mathcal{NL} \subset \mathrm{NPSPACE}$$

Beispiel: Das Pfadproblem (Path)

Gegeben sei ein Graph $G = (V, E)$ mit Startknoten $s \in V$ und Zielknoten $t \in V$. Frage: Existiert ein Weg von s nach t? Wir suchen also eine NTM, die dieses Problem mit Platzbedarf $\in \mathcal{NL}$ löst.

Algorithmus:

1. Nummeriere o. B. d. A. alle Knoten von 1 bis n, wobei Startknoten $s = 1$ und der Zielknoten $t = n$ sind.
2. Initialisiere einen Zähler mit Null. Beginne bei Knoten 1.
3. Schreibe aktuell betrachteten Knoten in Binärdarstellung auf das Arbeitsband.
4. Inkrementiere den Zähler entsprechend der gelesenen Knotenzahl.
5. Wähle nicht-deterministisch den nächsten Knoten, der möglicherweise auf einem Pfad zu n liegt.
6. Wiederhole solange, bis bei Knoten n angekommen oder der Zähler n ist. Falls ein eingeschlagener Weg nicht zu t führt, nimm einen anderen und überschreibe bereits genutzte Bandzellen.
7. Wenn aktueller Knoten n ist, schreiben wir auf das Ausgabeband eine Eins, sonst eine Null.

Platzbedarf: Sowohl die Knotennummern, als auch der Zähler nehmen höchstens den Wert n an. Folglich benötigt das Speichern beider höchstens $2 \cdot \lceil \log_2 n \rceil$ Bandzellen. Mit einem zusätzlichen Trennzeichen ist der gesamte Platzbedarf $\mathrm{space}_{\mathrm{N}}(G) \leq \log_2 n + 1$ und damit ist $\textsc{Path} \in \mathcal{NL}$.

Eigenschaften

Abschlusseigenschaften: $\forall A, B \in \mathcal{NL}$:

- Vereinigung: $A \cup B \in \mathcal{NL}$
- Durchschnitt: $A \cap B \in \mathcal{NL}$
- Komplement: $\bar{A} \in \mathcal{NL}$ (Beweis 1987 von N. Immermann und R. Szelepczényi)

Satz 9.

$$\mathcal{L} \subseteq \mathcal{NL}$$

Beweis. Jede DTM ist eine spezielle NTM, d. h. sie ist auch eine NTM, hat jedoch keine ϵ-Übergänge und sie hat eindeutige / deterministische Konfigurationsübergänge. □

Es konnte bisher nicht geklärt werden, ob $\mathcal{L} = \mathcal{NL}$ gilt.

4 $\mathcal{NL}$-vollständig

Seien $L_1 \in \Sigma_1^*$, $L_2 \in \Sigma_2^*$, $L_3 \in \Sigma_3^*$ Sprachen und $f\colon \Sigma_1^* \to \Sigma_2^*$ eine Funktion.

Definition 10. *Reduzierbarkeit von L_1 auf L_2:*

$$L_1 \leq_{log} L_2 \iff \exists f \text{ logarithmisch konstruierbar} : w \in L_1 \Leftrightarrow f(w) \in L_2$$

Definition 11. *L_2 heißt $\mathcal{NL}$-vollständig*

$$\begin{aligned} &\iff L_2 \in \mathcal{NL} \wedge \forall L_1 \in \mathcal{NL} : L_1 \leq_{log} L_2 \\ &\text{alternative Definition:} \\ &\iff L_2 \in \mathcal{NL} \wedge \exists L_3\, \mathcal{NP}\text{-vollständig} : L_3 \leq_{log} L_2 \end{aligned}$$

5 Zusammenhänge

Bemerkung: Bei der Zeitkomplexität haben wir gesehen, dass die Simulation eines NTM durch eine DTM auf Kosten exponentieller Laufzeit möglich ist. Wie ist das bei Speicherplatzkomplexität?

Satz 12. *(Satz von Savitch): Für jede Funktion $f\colon \mathbb{N}_0 \to \mathbb{N}_0$ mit $f(n) \geq \log n$ gilt:*

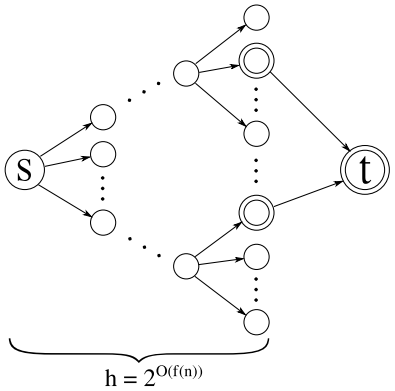
$$\mathrm{NSPACE}(f(n)) \subseteq \mathrm{SPACE}(f^2(n))$$

d. h. man kann Simulationen nicht-deterministischer Turing-Maschinen durch deterministische angeben, die höchstens quadratisches Anwachsen des benötigten Speicherplatzes zur Folge haben.

Beweis. Sei M_n eine NTM mit Platzbedarf $\mathcal{O}(f(n))$. M_n lässt sich als gerichteter Graph $G = (V, E)$ darstellen, wobei V die Menge aller Konfigurationen und E die Menge aller Konfigurationsübergänge ist. M_n akzeptiert ein Wort w genau dann, wenn es einen Pfad von der Startkonfiguration s zu einer akzeptierenden Konfiguration gibt.

Frage: Gibt es eine DTM M_d, die mit einem Platzbedarf von $\mathcal{O}(f^2(n))$ prüfen kann, ob es einen solchen Pfad gibt?
Dazu modifizieren wir G wie folgt:

- Füge einen neuen akzeptierenden Zielknoten t ein.
- Füge eine Kante für jeden akzeptierenden Knoten $v \in F$ nach t ein.



Mittels *dynamischem Programmieren* überprüft M_d, ob es einen Weg von s nach t mit $k = |V|$ Zwischenknoten gibt.

```
function reachMax(s,t,k):
   if   (s,t) in E
   then return true
   else
      for v in V\{s,t} do
         if   reachMax(s,v,floor(k/2))
         then reachMax(v,t,ceil(k/2))
         else return false
```

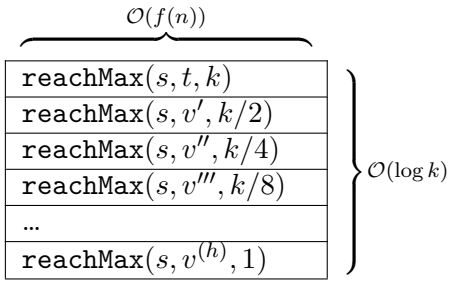
Frage: Wie viel Platz benötigt dies?

Seien $c_1 = |Q|, c_2 = |\Sigma|$ und $c_3 = |\Gamma|$ Konstanten.

- Eine Konfiguration $K = (q, e, w_1 \dots w_m)$ benötigt nach Voraussetzung $\mathcal{O}(f(n))$ Platz, somit ist $m \in \mathcal{O}(f(n))$.
- Es gibt höchstens
  - $|Q|$ verschiedene Belegungen für q,
  - $|\Sigma|$ verschiedene Belegungen für e,
  - $|\Gamma|^{\mathcal{O}(f(n))}$ viele Kombinationen der Zeichen auf dem Arbeitsband und
  - $\mathcal{O}(f(n))$ viele Positionen des Schreiblesekopfes (Arbeitsband)

  Folglich gibt es maximal $|Q| \cdot |\Sigma| \cdot \mathcal{O}(f(n)) \cdot |\Gamma|^{\mathcal{O}(f(n))} = 2^{\mathcal{O}(f(n))}$ verschiedene Konfigurationen.
- Folglich gibt es höchstens $|V| \leq 2^{\mathcal{O}(f(n))}$ viele Knoten, da es bei wiederholenden Konfigurationen einen Kreis gäbe.
- Die Länge eines Pfades kann höchstens $|V|$ betragen.
- Für jeden Rekursionsaufruf `reachMax(s,t,k)` werden sowohl die Konfigurationen s und t, als auch die Zahl k als Binärzahl gespeichert. Das benötigt $\mathcal{O}(f(n))$ viel Platz.

- Die Rekursionstiefe beträgt höchstens $\mathcal{O}(\log k) = \mathcal{O}(f(n))$. Folglich enthält der Aufrufstapel (siehe nachfolgende Grafik) höchstens $\mathcal{O}(f(n))$ viele Elemente.



Falls ein Rekursionsaufruf `false` zurück gibt, wird der begonnene Pfad verworfen und der benutzte Speicherplatz wieder verwendet. Sobald ein Pfad gefunden wurde, wird `true` zurück gegeben.

Somit beträgt der **Gesamtspeicherbedarf** der DTM $\mathcal{O}(f^2(n))$ und somit ist gezeigt, dass der Platzbedarf höchstens quadratisch Anwächst. □

Folgerung 13. *Aus* NSPACE$(f(n)) \subseteq$ SPACE$(f^2(n))$ *folgt mit* $f = \log n$*:*

$$\mathcal{L} \subseteq \mathcal{NL} \subseteq \mathcal{L}^2$$

Bei der Zeitkomplexität haben wir gesehen, dass die Frage $\mathcal{P} = \mathcal{NP}$ offen ist. Für die Speicherplatzkomplexität können wir die analoge Frage, ob PSPACE = NPSPACE ist, unmittelbar als Folgerung aus dem Satz von Savitch beantworten:

Folgerung 14.

$$\text{PSPACE} = \text{NPSPACE}$$

Satz 15.

$$\mathcal{P} \subseteq \text{PSPACE} \quad \wedge \quad \mathcal{NP} \subseteq \text{PSPACE}$$

Beweis. Denn jede Zeitbeschränkung stellt auch eine Platzbeschränkung dar: Bei n Konfigurationsübergängen kann der Schreib-Lesekopf höchstens $n + 1$ Speicherplätze besucht haben. □

Satz 16.

$$\mathcal{NL} \subseteq \mathcal{P}$$

Beweis. Das Arbeitsband einer NTM M_n benutzt höchstens logarithmisch viel Platz für jede Eingabe w mit $|w| = n$. Sei $K = (q, e, w_1 \dots w_m)$ mit $m \leq \log n$ eine Konfiguration von M_n, dann gibt es höchstens $2^{\mathcal{O}(\log n)}$ viele verschiedene Konfigurationen. Sei $G = (V, E)$ der Graph zu M_d. Dann gibt es $|V| = 2^{\mathcal{O}(\log n)} = 2^{c \log \mathcal{O}(n)} \in \mathcal{O}(n^c)$, c konstant, viele Knoten (Konfigurationen). Andernfalls gäbe es einen Kreis und M_n würde nicht terminieren. Folglich gibt es $|E| \leq |V|^2$ viele Kanten (Konfigurationsübergänge). Die Anzahl der durchgeführten Konfigurationsübergänge entspricht der Laufzeit. Da die Überführung einer NTM in eine DTM höchstens quadratisches Anwachsen des Speicherbedarfs nach sich zieht, liegt die Laufzeit in $\mathcal{O}(|V|^4) = \mathcal{O}(n^{2c})$ und somit in $\mathcal{P}$. □

Folgerung 17.

$$\mathcal{L} \subseteq \mathcal{NL} \subseteq \mathcal{P} \subseteq \mathcal{NP} \subseteq \text{PSPACE}$$

Vermutung: alle diese Inklusionen sind echt. Bekannt ist: $\mathcal{L} \subsetneq \text{PSPACE}$

Literatur

[1] Elias Drotleff. *Komplexitätsklassen*. 2006. URL: http://www.hs-weingarten.de/~ertel/vorlesungen/thinf/seminar-ws0607/EliasDrotleff/Komplexitaetsklassen.pdf.

[2] Peter Faymonville. *Der Satz von Savitch*. 2010. URL: http://wwwmayr.informatik.tu-muenchen.de/konferenzen/Sommerakademie2010/talks/faymonville_paper.pdf.

[3] Kurt-Ulrich Witt Gottfried Vossen. *Grundkurs Theoretische Informatik. Eine anwendungsbezogene Einführung – Für Studierende der Informatik, Wirtschaftsinformatik, Technik*. 3. Aufl. Vieweg, Apr. 2004, 335 ff. ISBN: 978-3-528-23147-7.

[4] Johannes Köbler. *Theoretische Informatik 2*. 2009. URL: https://www.informatik.hu-berlin.de/de/forschung/gebiete/algorithmenII/Lehre/ws09/theo2/skript/thi2-folien-kt.pdf.

[5] *Komplexitätstheorie*. 2010. URL: https://www.tu-ilmenau.de/fileadmin/public/iti/Lehre/BuK/WS10_11/buk_vorlesung_12.pdf.

[6] Markus Lohrey. *Komplexitätstheorie*. 2008. URL: http://www.informatik.uni-leipzig.de/alg/lehre/ws08_09/STU-KT/folien.pdf.

[7] *Multitape Turing machine*. URL: https://en.wikipedia.org/wiki/Multitape_Turing_machine.

[8] Arfst Nickelsen. *Komplexitätstheorie*. 2007. URL: http://www.tcs.uni-luebeck.de/downloads/lehre/2006-ws/komplexitaet/skript-08-feb.pdf.

[9] Christos H. Papadimitriou. *Compututational Complexity*. University of California – San Diego, 1994, 395 ff. ISBN: 0-201-53082-1.

[10] Michael Sipser. *Introduction to the Theory of Computation*. 2. Aufl. Thomson – Course Technology, 2006, 320 ff. ISBN: 0-534-95097-3. URL: http://ciencias.uis.edu.co/lenguajes/doc/sipser.pdf.

[11] *Speicherplatz-Komplexität*. 2013. URL: http://www.thi.informatik.uni-frankfurt.de/lehre/gl2/sose13/gl2_sose13_05_speicherplatz.pdf.