Komplexitätstheorie

Kapitel 5: Platzkomplexität

Einleitung

Platzverbrauch:

- der temporäre Zwischenspeicher, der während der Berechnung verwendet wird (Datenstrukturen, Rekursionsstack, etc.)
- Im Fall von TMs: die verwendeten Bandzellen außer denen für die Eingabe

In der Praxis kann Platzverbrauch so kritisch sein wie Zeitverbrauch:

- geringer Zeitverbrauch nicht hilfreich wenn Speicher vorher erschöpft
- dies passiert nicht selten bei Algorithmen, die schwierige Probleme lösen

Wesentlicher Unterschied zur Zeitkomplexität:

Platz kann man wiederverwenden, Zeit nicht

Einleitung

Überblick:

- Polynomieller Platzverbrauch
- Quantifizierte Boolesche Formeln (QBFs)
- Savitch's Theorem
- Logarithmischer Platzverbrauch
- DTMs vs NTMs, Erreichbarkeit in Graphen
- Komplementklassen /
  Theorem von Immerman und Szelepcsenyi

Kapitel 5

Platzkomplexitätsklassen

Platzkomplexität

Um Eingabe und "Zwischenspeicher" zu trennen, nehmen wir dediziertes Eingabeband an:

- Eingabeband wird nur gelesen, nicht beschrieben

  (TM muss per Konvention das auf diesem Band gelesene Symbol wieder zurückschreiben)

- Auf dem Eingabeband kann man sich nach links und rechts bewegen, also die Eingabe auch mehrfach lesen

- Im Zusammenhang mit Platzkomplexität ist k-Band TM also TM mit k Arbeitsbändern + 1 Eingabeband

Dies ermöglicht es uns, auch Platzbeschränkungen s mit $s(n) \leq n$ zu betrachten!

Aufgrund Ihrer Bedeutung für NP betrachten wir auch NTMs!

Platzkomplexität

Definition Platzbeschränkt

Für k-Band DTM oder NTM M und $w \in \Sigma^*$ schreiben wir $\text{space}_M(w) = n$ wenn alle Berechnungen von M auf w höchstens n Bandzellen verwenden. (alle Bänder aufsummiert)

Sei $s : \mathbb{N} \to \mathbb{N}$ monoton wachsende Funktion.
M ist s-*platzbeschränkt* wenn $\text{space}_M(w) \leq s(n)$ für alle w der Länge n

Nun ist:

$$\text{DSpace}_k(s) := \{L \subseteq \Sigma^* \mid \exists \mathcal{O}(s)\text{-platzbeschränkte terminierende } k\text{-Band DTM } M \text{ mit } L(M) = L\}$$

$$\text{NSpace}_k(s) := \{L \subseteq \Sigma^* \mid \exists \mathcal{O}(s)\text{-platzbeschränkte terminierende } k\text{-Band NTM } M \text{ mit } L(M) = L\}$$

Mehrband TMs

Auch bei Platzkomplexität werden wir uns um die Anzahl Bänder keine Gedanken machen

Bandreduktion (Platzkomplexität)

Für alle $k \geq 1$ und s: $\mathsf{DSpace}_k(s) \subseteq \mathsf{DSpace}_1(s)$.

Beweis: exakt derselbe wie für Zeitkomplexität
(verwende ein Band mit mehreren Spuren)

Beachte: im Gegensatz zu Zeitkomplexität kein quadratischer Blowup!

Definition DSpace, NSpace

Wir setzen $\mathsf{DSpace}(s) := \bigcup_{k \geq 1} \mathsf{DSpace}_k(s)$, analog für $\mathsf{NSpace}(s)$

Kapitel 5

Polynomieller Platzverbrauch

PSPACE

Analog zu polynomieller Zeit kann man polynomiellen Platz betrachten

Mehr als polynomiell viel Platz anzunehmen ist nicht realistisch

Aber auch polynomieller Platz ist schon “recht viel”:

- Betrachte Eingabe der Größe 10.000
- kubischer Zeitverbrauch (n^3): 5 Minuten auf 3Ghz Prozessor
- kubischer Platzverbrauch: 1 Terabyte Speicher nötig!

Entsprechende Komplexitätsklasse:

$$\text{PSPACE} := \bigcup_{i \geq 1} \text{DSpace}(n^i)$$

QBF

Ein "typisches" Problem in PSPACE

Gültigkeit von quantifizierten Booleschen Formeln (QBFs)

So zu verstehen:

- Boolesche Formeln = AL-Formeln
- quantifiziert: zusätzliche Quantoren für Wahrheitswerte, als Präfix

Definition QBF

Eine *quantifizierte Boolesche Formel (QBF)* hat die Form

$$Q_1 p_1 \cdots Q_n p_n . \varphi$$

wobei $Q_i \in \{\forall, \exists\}$ und φ eine AL-Formel mit Variablen aus der Menge $\{p_1, \ldots, p_n\}$.

AL-Formeln dürfen hier auch die Konstanten 0,1 enthalten

QBF

Definition Gültigkeit einer QBF

QBF $Q_1 p_1 \cdots Q_n p_n \varphi$ ist *gültig* wenn

- $Q_1 = \exists$ und $Q_2 p_2 \cdots Q_n p_n \varphi[p_1/0]$ oder $Q_2 p_2 \cdots Q_n p_n \varphi[p_1/1]$ gültig
- $Q_1 = \forall$ und $Q_2 p_2 \cdots Q_n p_n \varphi[p_1/0]$ und $Q_2 p_2 \cdots Q_n p_n \varphi[p_1/1]$ gültig
- $n = 0$ und φ (welches dann variablenfrei ist) zu 1 auswertet.

Wobei: $\varphi[p/0]$ ist φ mit p ersetzt durch 0, analog für $\varphi[p/1]$

Z.B.: $(p_1 \vee p_2) \rightarrow (p_2 \vee p_3)[p_2/0] \quad = \quad (p_1 \vee 0) \rightarrow (0 \vee p_3)$

Gültigkeit überprüfen durch Auswertungsbaum!

Auswertungsbäume

Auswertungsbaum für QBF $\psi = Q_1 p_1 \cdots Q_n p_n \varphi$:

- binärer Baum der Tiefe n
- Wurzel korrespondiert zu ψ
- Übergang zu Nachfolger entspricht Elimination eines Quantors durch Festlegen des Variablenwertes
- Blätter entsprechen also AL-Formeln ohne Variablen

Erfolgreicher Auswertungsbaum:

- “Hochpropagieren” der Auswertung:
- existentielle Quantoren = or; universelle = and (and/or-Baum)
- Wurzel muss zu 1 evaluieren

Erfolgreicher Auswertungsbaum ist “Beweis” für Gültigkeit
wie in der Definition von NP, aber exponentiell groß!

QBF

Theorem

QBF ist in PSPACE

Idee:

- verwende rekursive Prozedur, die sich aus Definition von Gültigkeit ergibt
- Rekursionstiefe ist linear, für jeden Rekursionsschritt wird linear viel Speicher gebraucht
- das ergibt quadratischen Platzbedarf, da man den Speicher wiederverwenden kann, wenn man in anderen Ast des (exponentiell großen) Rekursionsbaumes absteigt.

Kapitel 5

Zeitkomplexität vs. Platzkomplexität

Zeit vs. Platz

Natürliche Frage:

- wie verhält sich PSPACE zu P, NP und EXPTIME?
- allgemeiner: wie verhält sich Platzkomplexität zu Zeitkomplexität?

"Was man in Zeit f berechnen kann, kann man auch in Platz f berechnen, sogar für nicht-deterministische Zeit und deterministischen Platz"

Theorem

Für alle (monoton wachsenden) $f : \mathbb{N} \to \mathbb{N}$:

$$\text{NTime}(f) \subseteq \text{DSpace}(f)$$

Es folgt $\text{P} \subseteq \text{NP} \subseteq \text{PSPACE}$

Echtheit vermutet, aber unbewiesen (also: NP-vs.-PSPACE-Problem)

Zeit vs. Platz

Grundlage für verschiedene weitere Resultate zur Platzkomplexität:

Definition Konfigurationsgraph

Sei M eine (1-Band) NTM und $s \in \mathbb{N}$.

Dann ist $\mathsf{Conf}_{M,s}$ die Menge aller Konfigurationen von M der Länge $\leq s$.

Der *s-Konfigurationsgraph* für M ist der gerichtete Graph $G_{M,s} = (V, E)$ mit

- $V = \mathsf{Conf}_{M,s}$
- $E = \{(C, C') \mid C \vdash_M C'\}$

Wir können o.B.d.A. genau eine akzeptierende Konfiguration annehmen:

- es gibt sowieso nur einen akzeptierenden Zustand
- vor Anhalten löscht TM alle Bänder und fährt Köpfe ganz nach links

Zeit vs. Platz

Nun: Akzeptanz als Erreichbarkeit (GAP):

Lemma

Sei M s-platzbeschränkte NTM mit akzeptierender Konfiguration C_{acc} und w Eingabe der Länge n.
Es gilt $w \in L(M)$ gdw. C_{acc} von $q_0 w$ aus in $G_{M,s(n)}$ erreichbar

Der Konfigurationsgraph ist exponentiell groß:

$G_{M,s}$ hat Größe $|Q| \cdot |\Gamma|^s \cdot s \quad \in 2^{\mathcal{O}(s)}$

mögliche Zustände, Bandinhalte, Kopfpositionen

Zeit vs. Platz

Wir verwenden Konfigurationsgraphen für Beweis von
PSPACE $\subseteq$ EXPTIME (und mehr)

Definition Platzkonstruierbar

Funktion $s : \mathbb{N} \to \mathbb{N}$ heißt *platzkonstruierbar* wenn es terminierende DTM M gibt mit $\text{space}_M(w) = s(|w|)$ für alle $w \in \Sigma^*$.

Betrachte Platzkonstruierbarkeit als "technische Annahme", die von allen natürlichen Funktionen erfüllt wird.

Z.B.:

- n^i ist platzkonstruierbar, für alle $i \geq 1$
- 2^n ist platzkonstruierbar

Zeit vs. Platz

"Was man in Platz f berechnen kann, kann man auch in Zeit $2^{\mathcal{O}(f)}$ berechnen, sogar für nicht-deterministischen Platz und deterministische Zeit"

Theorem

Für alle platzkonstruierbaren Funktionen $f : \mathbb{N} \to \mathbb{N}$:

$$\mathsf{NSpace}(f) \subseteq \mathsf{DTime}(2^{\mathcal{O}(f)})$$

Es folgt $\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXPTIME}$

Echtheit vermutet, aber unbewiesen (also: PSPACE-vs.-EXPTIME-Problem)

Auch für deterministischen Platz ist keine bessere Schranke bekannt.

Kapitel 5

Savitch's Theorem

Savitch's Theorem

Def. PSPACE ähnelt P; was ist Platz-Klasse mit Def. ähnlich NP?

Nach alternativer Charakterisierung von NP folgendes:

$$\text{NPSPACE} := \bigcup_{i \geq 1} \text{NSpace}(n^i)$$

Interessanter Gegensatz:

- P vs. NP ist wichtigstes offenes Problem der Informatik
- PSPACE vs. NPSPACE wurde 1970 von W. Savitch gelöst:
  es gilt PSPACE = NPSPACE!

Savitch's Theorem

Zu zeigen: NPSPACE $\subseteq$ PSPACE.

Naiver Versuch mittels Konfigurationsgraphen:

- Wenn $L \in$ NPSPACE, dann gibt es p-platzbeschränkte NTM M mit $L(M) = L$, p Polynom
- Gesucht: polyplatzbeschränkte DTM, die entscheidet, ob $w \in L(M)$
- Konstruiere deterministisch den Konfigurationsgraphen $G_{M,s}$ mit $s = p(|w|)$, verwende Erreichbarkeit

Problem: $G_{M,s}$ hat Größe $2^{\mathcal{O}(p(|w|))}$, also exponentiell

Zentrale Idee von Savitch: Entscheide Erreichbarkeit in $G_{M,s}$,

ohne den ganzen Graphen auf einmal zu berechnen (nur poly große Teilstücke)

Savitch's Theorem

Theorem (Savitch)

Wenn s platzkonstruierbar ist, dann NSpace$(s) \subseteq$ DSpace(s^2).

Idee:

- Prädikat Pfad(C, C', i) ist wahr gdw. es Pfad in $G_{M,s(n)}$ gibt von C nach C' und mit Länge max. 2^i
- Wir interessieren uns also für Pfad$(q_0 w, C_{\text{acc}}, \log(|\text{Conf}_{M,s(n)}|))$
- Jeder Pfad der Länge max. 2^i von C nach C' hat "Mittelpunkt" C_m: C erreicht C_m erreicht C', beides in max. 2^{i-1} Schritten
- Benutze "Teile & Herrsche":
  Um Pfad(C, C', i) zu entscheiden, betrachte alle möglichen Mittelpunkte C_m, entscheide rekursiv Pfad$(C, C_m, i-1) \wedge$ Pfad$(C_m, C', i-1)$
- Rekursionstiefe $\log(|\text{Conf}_{M,s(n)}|) \in \mathcal{O}(s(n))$, für jeden rekursiven Abstieg Speicherbedarf $\mathcal{O}(s(n))$ auf Stack

Konsequenzen aus Savitch's Theorem

Korollar

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

(Anm.: det. Platzkomplexitätsklassen sind wie det. Zeitkomplexitäts-klassen unter Komplement abgeschlossen)

Aus diesem Grund werden NPSPACE und co-NPSPACE nicht betrachtet.

Man kann Savitch auch auf größere Klassen anwenden:

$$\text{EXPSPACE} := \bigcup_{i \geq 1} \text{DSpace}(2^{n^i}) \qquad \text{NEXPSPACE} := \bigcup_{i \geq 1} \text{NSpace}(2^{n^i})$$

Korollar

$\text{EXPSPACE} = \text{NEXPSPACE} = \text{co-NEXPSPACE}$

Konsequenzen aus Savitch's Theorem

Andere Sicht auf Beweis:

Savitch zeigt eigentlich nur ein Komplexitätsresultat für das konkrete Problem GAP und wendet dieses dann geschickt an

Korollar

GAP $\in$ DSpace$(\log(n)^2)$

Nutzen von Savitch's Theorem

Savitch ist auch nützlich, um PSPACE-Algorithmen zu finden:

Wir können o.B.d.A. Nicht-Determinismus benutzen!

Z.B. Universalität von nicht-deterministischen endlichen Automaten: gegeben nicht-det. endlicher Automat $\mathcal{A}$, ist $\mathcal{A} = \Sigma^*$?

Lemma

Wenn $L(\mathcal{A}) \neq \Sigma^*$, dann gibt es $w \in \Sigma^* \setminus L(\mathcal{A})$ der Länge max. $2^{|Q|}$ mit Q Zustandsmenge von $\mathcal{A}$.

Theorem

Das Universalitätsproblem für endliche Automaten ist in PSPACE.

Kapitel 5

PSpace-Härte und -Vollständigkeit

PSPACE-Vollständigkeit

Zur Erinnerung:

Theorem

$P \subseteq NP \subseteq PSPACE \subseteq EXPTIME$

Echtheit unbekannt!

Wie bei P vs NP: man behilft sich mit dem Begriff der Härte und Vollständigkeit

Härte gründet sich wieder auf Polynomialzeit-Reduktionen

Lemma

PSPACE ist abgeschlossen unter Polynomialzeit-Reduktionen:

Wenn $L' \leq_p L$ und $L \in PSPACE$, dann $L' \in PSPACE$

PSPACE-Vollständigkeit

Definition PSPACE-Härte, PSPACE-Vollständigkeit

Problem L ist

- PSPACE-*hart* wenn $L' \leq_p L$ für alle $L' \in$ PSPACE;
- PSPACE-*vollständig* wenn L PSPACE-hart und in PSPACE.

Beobachtung

Wenn L PSPACE-hart und

- $L \in$ P, dann P $=$ PSPACE;
- $L \in$ NP, dann NP $=$ PSPACE.

Nächstes Ziel: zeigen, dass QBF PSPACE-vollständig ist.

QBF

Wir verwenden QBFs, bei denen Quantoren nicht unbedingt Präfix sind

Definition generelle QBF

Sei AV abzählbar unendliche Menge von *Aussagenvariablen*.

Menge der *generellen QBFs* ist kleinste Menge so dass:

- 0,1 sind generelle QBFs
- jedes $p \in$ AV ist generelle QBF
- wenn φ, ψ generelle QBFs und $p \in$ AV, so sind auch $\neg\varphi$, $\varphi \vee \psi$, $\varphi \wedge \psi$, $\exists p.\varphi$ und $\forall p.\varphi$ generelle QBFs

Freie Variable ist Variable, die nicht durch Quantor gebunden ist.

Generelle QBF ohne freie Variable heißt *genereller QBF-Satz*.

QBF

Definition generelle QBF Semantik

Wertzuweisung (WZ) π : AV $\to \{0, 1\}$ *erfüllt* generelle QBF

- 0 niemals und 1 immer;
- p wenn $\pi(p) = 1$;
- $\neg\varphi$, wenn π nicht φ erfüllt;
- $\varphi \wedge \psi$ wenn π sowohl φ als auch ψ erfüllt;
- $\varphi \vee \psi$ wenn π φ oder ψ erfüllt (oder beides);
- $\exists p.\varphi$ wenn $\pi[p/0]$ φ erfüllt oder $\pi[p/1]$ φ erfüllt;
- $\forall p.\varphi$ wenn $\pi[p/0]$ φ erfüllt und $\pi[p/1]$ φ erfüllt.

Genereller QBF-Satz φ ist *gültig*, wenn er von jeder (äquivalent: einer) WZ erfüllt wird.

QBF

Zur Unterscheidung nennen wir die ursprünglichen definierten *Präfix-QBF*

Nicht schwer zu zeigen:

Lemma

Jeder generelle QBF-Satz φ kann in polynomieller Zeit in eine Präfix-QBF φ' umgewandelt werden, so dass φ gültig gdw. φ' gültig.

Theorem

QBF ist PSPACE-hart, also PSPACE-vollständig.

Strategie: Zeigen, dass $L \leq_p$ QBF für alle $L \in$ PSPACE.

QBF

Sei $L \in$ PSPACE und M eine $p(n)$-platzbeschränkte DTM mit $L(M) = L$.

Ziel: gegeben w, finde generellen QBF-Satz φ_w so dass

1. φ_w gültig gdw. M akzeptiert w und
2. φ_w in Polyzeit aus w konstruierbar

Wir können nicht den Matrix-Ansatz von Cook verwenden, denn M hat u.U. exponentielle Laufzeit

Stattdessen: Reformulierung des Beweises von Savitch's Theorem "in der Sprache von QBF"

QBF

Reformulierung des Beweises von Savitch's Theorem
"in der Sprache von QBF"

Sei $|w| = n$

M ist $p(n)$-platzbeschränkt

$$\Rightarrow \quad M \text{ ist } T(n) = 2^{q(n)}\text{-zeitbeschränkt für Polynom } q$$

Konkreteres Ziel: φ_w beschreibt Prädikat $\mathsf{Pfad}\big(q_0 w, C_{\mathsf{acc}}, q(|w|)\big)$
und dessen rekursive Berechnung

Zur Erinnerung: $\mathsf{Pfad}(C, C', i)$ ist wahr, wenn es Pfad mit Länge $\leq 2^i$
von C nach C' in $G_{M,p(n)}$ gibt.

QBF

Wir repräsentieren Konfiguration durch folgende Variablen:

- Z_q für jedes $q \in Q$ beschreibt Zustand;
- $B_{a,i}$ für jedes $a \in \Gamma$ und $i \leq p(n)$ beschreibt Symbol auf i-ter Bandzelle (Nummerierung beginnt mit 0);
- K_i für jedes $i \leq p(n)$ beschreibt Kopfposition.

Tupel aller dieser Variablen: $\overline{C}$

Um über mehrere Konfigurationen zu sprechen: zusätzliche Tupel $\overline{C}'$, $\overline{C}''$
(alle Variablen mit $'$ bzw. $''$)

Diese Formel garantiert, dass die Variablen "legale" Konfiguration beschreiben:

$$\psi_{\mathsf{conf}}(\overline{C}) := \bigvee_{q \in Q} Z_q \wedge \bigwedge_{q,q' \in Q, q \neq q'} \neg(Z_q \wedge Z_{q'}) \wedge \bigvee_{i \leq p(n)} K_i \wedge \bigwedge_{i,i' \leq p(n), i \neq i'} \neg(K_i \wedge K_{i'}) \wedge$$
$$\bigwedge_{i \leq p(n)} \bigvee_{a \in \Gamma} B_{a,i} \wedge \bigwedge_{i \leq p(n), a, a' \in \Gamma, a \neq a'} \neg(B_{a,i} \wedge B_{a',i})$$

QBF

Sei $R(i) = i + 1$ und $L(i) = i - 1$ wenn $i > 0$ und $L(0) = 0$.

Folgende Formel sagt, dass $\overline{C}'$ sich in einem Schritt aus $\overline{C}$ ergibt

$$\psi_{\mathsf{next}}(\overline{C}, \overline{C}') := \psi_{\mathsf{conf}}(\overline{C}) \wedge \psi_{\mathsf{conf}}(\overline{C}') \wedge$$

$$\bigwedge_{i \leq p(n)} \Big(K_i \rightarrow \Big(\bigwedge_{j \leq p(n), j \neq i, a \in \Gamma} (B_{a,j} \leftrightarrow B'_{a,j}) \wedge \bigwedge_{\delta(q,a)=(q',a',M), M(i) \leq p(n)} (Z_q \wedge B_{a,i}) \rightarrow (Z'_{q'} \wedge B'_{a',i} \wedge K'_{M(i)}) \Big) \Big)$$

Wir definieren nun $\mathsf{Pfad}(C, C', i)$ durch Formeln $\psi^i_{\mathsf{reach}}(\overline{C}, \overline{C}')$

Für den Fall $i = 0$:

$$\psi^0_{\mathsf{reach}}(\overline{C}, \overline{C}') := \psi_{\mathsf{eq}}(\overline{C}, \overline{C}') \vee \psi_{\mathsf{next}}(\overline{C}, \overline{C}')$$

wobei $\psi_{\mathsf{eq}}(\overline{C}, \overline{C}')$ sagt, dass $\overline{C}$ und $\overline{C}'$ identisch

QBF

Für $i > 0$ ist es verlockend, zu schreiben

$$\psi^i_{\mathsf{reach}}(\overline{C}, \overline{C}') := \exists \overline{C}''.\psi^{i-1}_{\mathsf{reach}}(\overline{C}, \overline{C}'') \wedge \psi^{i-1}_{\mathsf{reach}}(\overline{C}'', \overline{C}')$$

aber das führt zu QBF der Größe min. 2^i (wiederholte Verdopplung!)

Universelle Quantoren helfen:

$$\begin{aligned}\psi^i_{\mathsf{reach}}(\overline{C}, \overline{C}') := \exists \overline{C}'' \forall \overline{K} \forall \overline{K}'.\\ \big((\psi_{\mathsf{eq}}(\overline{C}, \overline{K}) \wedge \psi_{\mathsf{eq}}(\overline{C}'', \overline{K}')) \vee \\ (\psi_{\mathsf{eq}}(\overline{C}'', \overline{K}) \wedge \psi_{\mathsf{eq}}(\overline{C}', \overline{K}')) \big) \rightarrow \psi^{i-1}_{\mathsf{reach}}(\overline{K}, \overline{K}')\end{aligned}$$

Leicht zu finden:

- Formel $\psi^w_{\mathsf{input}}(\overline{C})$ die sagt, dass $\overline{C}$ initiale Konfiguration für Eingabe w ist
- Formel $\psi_{\mathsf{acc}}(\overline{C})$ die sagt, dass $\overline{C}$ akzeptierend ist.

QBF

Zur Erinnerung: M ist $p(n)$-platzbeschränkt

$$\Rightarrow \quad M \text{ ist } T(n) = 2^{q(n)}\text{-zeitbeschränkt für Polynom } q.$$

Die Reduktions-QBF ist nun:

$$\psi_w := \exists \overline{C} \exists \overline{C'}.(\ \psi^w_{\mathsf{input}}(\overline{C}) \wedge \psi_{\mathsf{acc}}(\overline{C}') \wedge \psi^{q(|w|)}_{\mathsf{reach}}(\overline{C}, \overline{C}')\ )$$

Lemma

M akzeptiert w gdw. φ_w gültig.

QBF ist ein bisschen wie "SAT für PSPACE"

PSPACE-Vollständigkeit

Natürlichere Probleme, die PSPACE-vollständig sind:

- Universalität von endlichen Automaten
- Schnittproblem für endliche Automaten:
  gegeben nicht-det. endliche Automaten $\mathcal{A}$, $\mathcal{A}'$, ist $L(\mathcal{A}) \cap L(\mathcal{A}') = \emptyset$?
- SQL Anfrageproblem
  gegeben Instanz von relationaler Datenbank D, SQL-Anfrage q,
  Tupel t: ist t in der Antwort von q auf D enthalten?
- Viele Spielprobleme, z. B.
  Brettspiele mit 2 Personen wie Dame (auf Feldern beliebiger Größe)

Kapitel 5

Logarithmischer Platz

LogSpace

Die Definition und Analyse von PSpace hat Struktur im Raum der nicht effizient lösbaren Probleme offengelegt

Hier: weitere Analyse des Raumes der effizient lösbaren Probleme

$$\text{LogSpace} := \text{DSpace}(\log(n))$$

Aus $\log(n^i) = i \cdot \log(n)$ folgt $\text{LogSpace} = \bigcup_{i \geq 1} \text{DSpace}(\log(n^i))$

Aus $\text{NSpace}(f) \subseteq \text{DTime}(2^{\mathcal{O}(f)})$ folgt:

Theorem

$\text{LogSpace} \subseteq \text{P}$

LogSpace

Schon gesehen: $L = \{u\overleftarrow{u} \mid u \in \{a, b\}^*\} \in$ LogSpace

Ideen:

- Zähle Länge der Eingabe in binär mittels Zähler $Z1$
- Verwerfe wenn ungerade
- Halbiere $Z1$
- Vergleiche 1. Symbol mit letztem, 2. mit 2.-letztem, etc.
- Zähler $Z2$ speichert zu vergleichende Position, zählt von 1 bis $Z1$
- Zähler $Z3$ wird benutzt, um Position zu finden, zählt von 1 bis $Z2$

Intuitiv (und auch formalisierbar):
LogSpace beschreibt, was man mit einer fixen Zahl binärer Zähler erreichen kann.

LogSpace: Kompositionalität

Wollen zeigen: LogSpace ist abgeschlossen unter

- Ausführen zweier Algorithmen, triviale Kombination der Ergebnisse
- Ausführen eines Algorithmus in jedem Schritt eines anderen
- Anwenden eines Algorithmus auf Ergebnis eines anderen

Komposition erfordert

(1) TM mit Ausgabe und
(2) Zwischenspeichern von Ergebnissen

Aber:

- LogSpace-Algorithmus kann polynomiell große Ausgabe generieren, für Ausgabe benötigter Platz darf also nicht mitgezählt werden
- Speicherung der Ausgabe als “Zwischenergebnis” bei Komposition in LogSpace also auch unmöglich!

Wir definieren uns erstmal ein entsprechendes Maschinenmodell

LogSpace-Transduktor

Definition LogSpace-Transduktor

Ein LogSpace-*Transduktor* ist DTM M mit

- einem Eingabeband, von dem nur gelesen wird;
- einer festen Zahl von logarithmisch in der Eingabelänge beschränkten Arbeitsbändern
- einem Ausgabeband, von dem nicht gelesen wird und so dass in jedem Schritt:
  - entweder ein Symbol auf Ausgabeband geschrieben und Kopf einen Schritt nach rechts bewegt wird
  - oder nichts auf Ausgabeband geschrieben wird und der Kopf seine Position behält.

Abbildung $f : \Sigma^* \to \Gamma^*$ ist LogSpace-*berechenbar*, wenn es LogSpace-Transduktor gibt, der bei jeder Eingabe $w \in \Sigma^*$ anhält und $f(w)$ auf Ausgabeband schreibt.

LogSpace: Kompositionalität

LogSpace ist abgeschlossen unter:

- Ausführen zweier Algorithmen, triviale Kombination der Ergebnisse
- Ausführen eines Algorithmus in jedem Schritt eines anderen
- Anwenden eines Algorithmus auf Ergebnis eines anderen

Der Beweis ist in allen Fällen ähnlich, wir zeigen nur letzteres

Theorem

Wenn $f : \Sigma^* \to \Gamma^*$ und $g : \Gamma^* \to \Delta^*$ LogSpace-berechenbar, so auch $g(f) : \Sigma^* \to \Delta^*$.

ACYC

Ein natürliches Problem in LOGSPACE:

ACYC ist die Menge der ungerichteten Graphen G, die azyklisch sind

Zur Erinnerung: Sei $G = (V, E)$ ungerichteter Graph. Dann ist

- ein *Kreis* in G eine Knotenfolge $v_1, \ldots, v_n$ mit $n \geq 3$ und $v_1 = v_n$, so dass $v_1, \ldots, v_{n-1}$ paarw. verschieden und $\forall i < n : \{v_i, v_{i+1}\} \in E$
- G *azyklisch*, wenn es keinen Kreis in G gibt.

Erste Analyse:

- naiver Ansatz: Graph nach Zyklen absuchen
- das ist im Prinzip leicht, z.B. mittels Tiefensuche
- allerdings muss man sich für das Backtracking der Tiefensuche den gesamten bisher abgelaufenen Pfad merken $\Rightarrow$ kein LOGSPACE

ACYC

Ideen für Lösung:

- Nicht den ganzen Pfad merken – nur den aktuellen Knoten
- Das geht in LOGSPACE: Knoten als binäre Zahlen repräsentieren
  Eingabe der Größe n enthält maximal n Knoten $\Rightarrow$ $\log(n)$ bits
- Die “Nummerierung” der Knoten ist o.B.d.A. durch Eingabe gegeben
  (Repräsentation als Wort)
- Kein Backtracking mehr möglich
  $\Rightarrow$ Durchlaufen des Graphen in fester Reihenfolge ohne Backtracking
- Reihenfolge gegeben durch lokale Kantenordnung
  $=$ Ordnung der adjazenten Kanten an jedem Knoten:
  ebenfalls implizit durch Eingabe gegeben
- Zu zeigen: durch die fixierte Ordnung entgeht einem kein Kreis!

ACYC

Definition: geordneter Pfad/Zyklus

Geordneter Pfad in $G = (V, E)$ ist Knotenfolge $v_1, \ldots, v_n$, so dass:

- wenn v_i auf Kante j erreicht wird (bzgl. lokaler Kantenordnung bei v_i), dann wird v_i verlassen auf
  - Kante Nummer $j + 1$, wenn es eine solche gibt
  - Kante Nummer 1 sonst

Geordneter Zyklus in $G = (V, E)$ ist geordneter Pfad $v_1, \ldots, v_n$, so dass

- $v_1 = v_n$
- $v_1 \notin \{v_2, \ldots, v_{n-1}\}$ (d. h. Startknoten wird nicht schon eher wieder erreicht)
- $v_2 \neq v_{n-1}$ (d. h. Start und Ende mit unterschiedlichen Kanten)

Beachte:

Knoten auf geordnetem Pfad/Zyklus müssen nicht alle verschieden sein.

ACYC

Lemma

Sei G zusammenhängend und ohne geordneten Zyklus. Dann gilt:

Folgt man einem geordneten Pfad lange genug,
so wird jede Kante von G in beiden Richtungen beliebig oft besucht.

Lemma

G ist azyklisch gdw. G keinen geordneten Zyklus enthält.

Beachte:

- das ist vollkommen unabhängig von den gewählten Ordnungen
- einen geordneten Zyklus kann eine DTM Schritt für Schritt ablaufen

ACYC

Wir können nun zeigen:

Theorem

ACYC $\in$ LogSpace

Was wir gesehen haben, ist typisch für LogSpace:

- der eigentliche Algorithmus ist recht einfach
- seine Korrektheit zu beweisen erfordert eine sehr vorsichtige Analyse des Problems.

LOGSPACE

Weitere Probleme in LOGSPACE z.B.:

- Isomorphie von Bäumen (gerichtet und ungerichtet)
- Entscheiden, ob ein Graph zusammenhängend ist (gerichtet und ungerichtet)
- Pattern matching: gegeben Wort *w* und Pattern (Wort) *p*, entscheide ob *p* Teilwort von *w* ist
- Addition, Subtraktion, Multiplikation, Division von natürlichen Zahlen
- Auswertung von AL-Formeln (gegeben Formel und Belegung, erfüllt Belegung die Formel?)

Kapitel 5

Nicht-Determinismus und LogSpace

NLogSpace

Auch von LogSpace gibt es eine nicht-deterministische Variante:

$$\text{NLogSpace} := \text{NSpace}(\log(n))$$

Aus $\text{NSpace}(s) \subseteq \text{DTime}(2^{\mathcal{O}(s)})$ folgt

Theorem

LogSpace $\subseteq$ NLogSpace $\subseteq$ P

Aus Savitch's Theorem folgt nicht, dass LogSpace = NLogSpace, nur

$$\text{NLogSpace} \subseteq \text{DSpace}(\log(n)^2)$$

In der Tat ist nicht bekannt, ob LogSpace = NLogSpace

Es macht also Sinn, NLogSpace-Härte und -Vollständigkeit zu betrachten

LogSpace-Reduktionen

Polynomialzeit-Reduktionen sind hier nicht sinnvoll, da

für alle $L, L' \in$ NLogSpace mit L' nicht-trivial: $L \leq_p L'$

(*nicht-trivial*: es gibt positive Instanzen und negative Instanzen)

Definition LogSpace-Reduktion

Reduktion f von L auf L' ist LogSpace-*Reduktion*, wenn sie LogSpace-berechenbar ist. Wir schreiben $L \leq_{\log} L'$.

Beachte:

- $\leq_p$ bezieht sich immer auf Polynomialzeit
  (Polynomialplatz wäre zu mächtig für Reduktionen)
- $\leq_{\log}$ bezieht sich immer auf Logplatz
  (Logzeit würde nicht mal das Lesen der Eingabe erlauben)

LogSpace-Reduktionen

Lemma

1. "LogSpace-reduzierbar" ist transitive Relation:
   $L_1 \leq_{\log} L_2$ und $L_2 \leq_{\log} L_3$ impliziert $L_1 \leq_{\log} L_3$
2. LogSpace, NLogSpace, P, NP, PSpace sind abgeschlossen unter LogSpace-Reduktionen: $L' \in \mathcal{C}$ und $L \leq_{\log} L'$ impliziert $L \in \mathcal{C}$;

Beweis:

1. Folgt aus: wenn f und g LogSpace-berechenbar, dann auch $f(g)$
   (Kompositionalität, siehe vorn)
2. für P, NP, PSpace: wegen $\leq_{\log} \subseteq \leq_p$
   und bekannter Abgeschlossenheit unter $\leq_p$

   für (N)LogSpace: wieder wegen Kompositionalität

LogSpace-Reduktionen

Bemerkungen:

- alle hier gezeigten Polyzeit-Reduktionen für NP-Vollständigkeit lassen sich auf LogSpace-Reduktionen verbessern
- es ist unbekannt, ob LogSpace-Reduzierbarkeit und Polyzeit-Reduzierbarkeit dasselbe sind;

  keine einfache Frage, denn

$$\leq_{\log} = \leq_p \text{ gdw. } \text{LogSpace} = \text{P}$$

NLogSpace-Vollständigkeit

Definition NLogSpace-Härte, NLogSpace-Vollständigkeit

Problem L ist

- NLogSpace-*hart*, wenn $L' \leq_{\log} L$ für alle $L' \in$ NLogSpace;
- NLogSpace-*vollständig*, wenn L NLogSpace-hart & in NLogSpace.

Theorem

GAP ist NLogSpace-vollständig.

Beweis zeigt sehr engen Zusammenhang zwischen NLogSpace und GAP, schon fast: NLogSpace ist GAP!

NLogSpace-Vollständigkeit

Bemerkungen:

- Die LogSpace-vs.-NLogSpace-Frage ist:
  ist GAP für gerichtete Graphen in LogSpace?

- Omer Reingold hat 2004 in einem vielbeachteten Resultat gezeigt, dass GAP für *ungerichtete* Graphen in LogSpace ist

NLogSpace-Vollständigkeit

Weitere NLogSpace-vollständige Probleme z.B.:

- 2SAT
- Leerheitsproblem und Wortproblem für endliche Automaten
- Leerheitsproblem und Wortproblem für rechtslineare Grammatiken
- Entscheiden, ob ein Graph durch 2 Cliquen überdeckt werden kann
- Entscheiden, ob ein gerichteter Graph stark zusammenhängend ist, ob also jeder Knoten von jedem Knoten erreichbar ist

Kapitel 5

co-NLogSpace

co-NLogSpace

Da NLogSpace nicht-deterministische Klasse, macht es Sinn,

co-NLogSpace zu betrachten

co-NLogSpace $:= \{\overline{L} \mid L \in$ NLogSpace$\}$

Es war lange Zeit unbekannt, ob NLogSpace = co-NLogSpace

Theorem von Immerman und Szelepcsényi (1988)

NLogSpace = co-NLogSpace

Das Resultat lautet sogar:

für alle $s \in \Omega(\log(n))$ gilt NSpace(s) = co-NSpace(s)

Immerman/Szelepcsényi

Da GAP NLogSpace-vollständig, reicht es zu zeigen:

$$\overline{\text{GAP}} \in \text{NLogSpace}$$

Wir brauchen also eine LogSpace-NTM für Nicht-Erreichbarkeit in gerichteten Graphen!

Wir gehen in zwei Schritten vor:

1. NLogSpace-Algorithmus für: gegeben G, u_0, v_0, c mit c die Anzahl der von u_0 aus erreichbaren Knoten, ist v_0 nicht erreichbar von u_0?
2. Zeigen, dass c in NLogSpace berechnet werden kann.

Immerman/Szelepcsényi – Schritt 1

Gegeben G, u_0, v_0, c mit c die Anzahl der von u_0 aus erreichbaren Knoten, ist v_0 nicht erreichbar von u_0?

Maschine M im Überblick:

- Iteriere über alle Knoten v
  - Wenn $v \neq v_0$, dann rate, ob v von u_0 erreichbar ist
  - Wurde "erreichbar" geraten, so überprüfe dies durch sukzessives Raten eines Pfades (Länge $\leq |V|$) von u_0 nach v
  - Wenn Überprüfung fehlschlägt, verwirf *(keine "false positives")*
- Zähle in binär die Anzahl x der als erreichbar geratenen Knoten
- Wenn $x = c$, akzeptiere, sonst verwirf *(keine "false negatives")*

Beachte: wenn M akzeptiert, dann $x = c$, also sind alle als nicht erreichbar geratenen Knoten (inkl. v_0) wirklich nicht erreichbar

Erläuterungen zu “false positives/negatives”

“false positives”

Knoten v, für die der Algorithmus fälschlicherweise schließen würde:
“v von u_0 aus erreichbar”

$\Rightarrow$ werden durch Überprüfen der geratenen Erreichbarkeit identifiziert

“false negatives”

Knoten v, für die der Algorithmus fälschlicherweise schließen würde:
“v nicht von u_0 aus erreichbar”

$\Rightarrow$ werden identifiziert, indem nach der Iteration die Anzahl der als erreichbar *geratenen* Knoten mit der *bekannten* Anzahl erreichbarer Knoten verglichen wird:

wenn $x < c$, dann gab es “false negatives”, also verwirf

Immerman/Szelepcsényi – Schritt 2

Zeigen, dass c in NLogSpace berechnet werden kann.

Überblick:

- wir entwerfen LogSpace-NTM M, bei der eine Berechung den korrekten Wert für c ausrechnet und alle anderen verwerfen
- diese kann dann der vorigen NTM “vorgeschaltet” werden
- Sei $V_i \subseteq V$ die Menge der Knoten, die von u_0 aus in $\leq i$ Schritten erreicht werden und $|V_i| = c_i$ für $0 \leq i \leq |V|$
- Wir benötigen $c_{|V|}$, bestimmen es durch Berechnen von $c_0, \cdots, c_{|V|}$
- Berechnung von c_i: Iteriere über alle $v \in V$, bestimme ob $v \in V_i$, zähle binär
- Bestimmen ob $v \in V_i$ ähnlich Schritt 1: rate sukzessive alle Knoten in V_{i-1}, identifiziere “false positives” mit Erreichbarkeitstest und “false negatives” durch Vergleich mit dem schon berechneten c_{i-1}
  Es bleibt zu prüfen, ob v von Knoten in V_{i-1} in einem Schritt erreichbar

Zurück zu Immerman/Szelepcsényi

Das Resultat lautet sogar:
für alle $s \in \Omega(\log(n))$ gilt NSpace(s) = co-NSpace(s)

Erweiterung des Beweises auf diesen allgemeinen Fall:

- Gegeben s-platzbeschränkte NTM M und Eingabe w, $|w| = n$
- Konfigurationsgraph $G_{M,s(n)}$ ist $2^{\mathcal{O}(s(n))}$ groß

$\rightsquigarrow$ Der genannte Algorithmus zum Testen von Nicht-Erreichbarkeit in $G_{M,s(n)}$ liefert also s-platzbeschränkte NTM

- $s \in \Omega(\log(n))$: sonst können wir die Zähler nicht verwenden

Weiterer Nutzen von Immerman/Szelepcsényi

Randbemerkung:

in seiner allgemeinen Formulierung löst das Theorem von Immerman und Szelepcsényi noch eine weitere wichtige offene Frage.

Theorem

Die Klasse der kontextsensitiven Sprachen (Typ-1-Sprachen) ist unter Komplement abgeschlossen.

Denn:

- die kontextsensitiven Sprachen sind genau die Sprachen, die von linear beschränkten Automaten (LBAs) erkannt werden
- LBAs sind nichts weiter als n-platzbeschränkte NTMs
- Immerman/Szelepcsényi zeigen, dass NSpace$(n) =$ co-NSpace(n)

Kapitel 5

Platzhierarchiesatz

Hierarchiesatz

Auch für Platzkomplexitätsklassen gibt es einen Hierarchiesatz:

Theorem (Platzhierarchie).

Für jede platzkonstruierbare Funktion s_2 und jede Funktion s_1 mit $s_2 \in \omega(s_1)$ gilt: $\mathsf{DSpace}(s_1) \subsetneq \mathsf{DSpace}(s_2)$.

Beweis analog zu Zeithierarchiesätzen, aber etwas einfacher

Konsequenzen z.B.:

- $\mathsf{DSpace}(n^i) \subsetneq \mathsf{DSpace}(n^{i+1})$ für alle $i \geq 0$
- $\textsc{LogSpace} \subsetneq \textsc{PSpace} \subsetneq \textsc{ExpSpace} := \bigcup_{i \geq 1} \mathsf{DSpace}(2^{\mathcal{O}(n^i)})$
- $\textsc{LogSpace} \subsetneq \mathsf{DSpace}(\log(n)^2)$

Übersicht Vorlesung

- Kapitel 1: Einführung
- Kapitel 2: Turingmaschinen
- Kapitel 3: P vs. NP
- Kapitel 4: Mehr Ressourcen, mehr Möglichkeiten?
- Kapitel 5: Platzkomplexität
- Kapitel 6: Schaltkreise
- Kapitel 7: Orakel