Speicherplatz-Komplexität

Sublogarithmischer Speicher (1/3)

- Für Eingaben der Länge n wird logarithmischer Speicher $O(\log_2 n)$ benötigt, um sich an eine Eingabeposition zu erinnern.
- Und wenn nur Speicher $o(\log_2 n)$ zur Verfügung steht?
  - $\mathrm{bin}(i)$ sei die Binärdarstellung der Zahl i ohne führende Nullen.
  - Für Eingabealphabet $\Sigma = \{0, 1, \$\}$ definiere

$$\mathrm{BIN} = \{\mathrm{bin}(1)\$\mathrm{bin}(2)\$ \cdots \$\mathrm{bin}(n) \mid n \in \mathbb{N}\}.$$

  - $\mathrm{BIN} \in \mathtt{DSPACE}(\log_2 \log_2 n)$, denn $\mathrm{bin}(i)$ besteht für jedes $i \leqslant n$ aus höchstens $\lceil \log_2 n \rceil$ Bits.
  - BIN ist nicht regulär.
- Reguläre Sprachen benötigen überhaupt keinen Speicher. Stimmt die Klasse regulärer Sprachen mit $\mathrm{DSPACE}(0)$ überein?
  - Achtung: Endliche Automaten durchqueren ihre Eingabe von links nach rechts, I-O TMs verhalten sich wie Zwei-Weg Automaten.
  - Gibt es nicht-reguläre Sprachen in $\mathrm{DSPACE}(o(\log_2 \log_2 n))$?

Sublogarithmischer Speicher (2/3)

Wenn $s = o(\log_2 \log_2 n)$, dann ist

$$\text{DSPACE}(s) = \text{DSPACE}(0) = \text{ die Klasse der regulären Sprachen.}$$

- Zeige die Behauptung für $s(n) = 0$.
  - Simuliere einen deterministischen Zwei-Weg Automaten Z durch einen NFA N mit ε-Übergängen.
  - Wenn Q die Zustandsmenge von Z und $|Q| = q$, dann wähle

$$Q' = \bigcup_{k=1}^{q} \Sigma \times (Q \times \{\text{links}, \text{rechts}\})^k$$

    als Zustandsmenge von N.
  - Im Zustand $(a, q_1, \text{richtung}_1, \ldots, q_r, \text{richtung}_r)$ spekuliert N, dass
    (a) a der gegenwärtig gelesene Buchstabe ist,
    (a) Z den Zustand q_i beim $i$ten Besuch der gegenwärtig besuchten Zelle annimmt,
    (b) und dass der $i$te Besuch von der linken Nachbarzelle ($\text{richtung}_i = \text{links}$), bzw. von der rechten Nachbarzelle ($\text{richtung}_i = \text{rechts}$) erfolgt.
- Spekulieren ist OK, aber wie verifiziert N? Welche Zustandsübergänge?

Sublogarithmischer Speicher (3/3)

N verifiziert mit Hilfe der Zustandsübergänge!

- der Zustandsübergang

$$(a, q_1, r_1, \ldots, q_k, r_k) \xrightarrow{N} (b, q'_1, r'_1, \ldots, q'_m, r'_m)$$

ist möglich, wenn *N*

(a) den Buchstaben a liest,
(b) die Vektoren $\vec{q}$ und $\vec{r}$ unter der Annahme verifizieren kann, dass die Vektoren $\vec{q'}$ und $\vec{r'}$ richtig geraten wurden und wenn
(c) für alle i mit „$r'_i = \text{links}$“ der Zustand q'_i richtig geraten wurde, bzw. konsistent mit der restlichen Information ist.

- Und wann wird akzeptiert?
  - Wenn $r_i = \text{links}$ für alle i
  - und wenn q_k akzeptierend ist.

N schiebt das Verifizieren von links nach rechts vor sich her.

Logarithmischer Speicherplatz

DL und NL gehören zu den wichtigsten Speicherplatz-Klassen.

- Die Berechnungskraft ist durchaus signifikant, da die Maschinen sich jetzt Positionen in der Eingabe merken können.
- Viele Eigenschaften, die für DL und NL gelten, verallgemeinern sich auf beliebige Speicherplatzklassen.

  Dieses Phänomen werden wir im Satz von Savitch und im Satz von Immerman-Szlepscenyi beobachten.

Wir beginnen mit deterministisch logarithmischem Platz.

Die Sprache der Palindrome

PALINDROM $\in$ DL. (PALINDROM ist die Menge aller Palindrome über dem Alphabet $\{0, 1\}$.)

1. Berechne $n - 1$ in Binärdarstellung, wenn n die Eingabelänge n ist.
   - Der Kopf des Arbeitsbands und der „Eingabekopf“ arbeiten zusammen.
   - Ein auf 0 initialisierter binärer Längenzähler wird solange inkrementiert bis der Eingabekopf das Ende der Eingabe erreicht hat.
   - Dann wird der Längenzähler um Eins herabgesetzt.
2. Vergleiche die Bits in den Positionen 1 und n, 2 und $n - 1$, etc.
   - Ein zweiter Zähler, der Distanzzähler, wird auf Null gesetzt. Der Lesekopf wandert nach rechts bis der Distanzzähler den Wert des Längenzählers erreicht hat.
   - Der Längenzähler wird um Eins herabgesetzt.
   - Der Distanzzähler wird auf Null gesetzt. Der Lesekopf wandert nach links, bis der Distanzzähler den Wert des Längenzählers erreicht hat...

Extrem mühselig, aber es funktioniert.

Wie mächtig ist DL? (1/3)

Zu DL gehören

- die Dyck-Sprache aller wohlgeformten Klammerausdrücke und
- die kontextsensitive Sprache $\{a^n b^n c^n \mid n \in \mathbb{N}\}$.
- Sogar U-REACHABILITY, die Menge aller **ungerichteten** Graphen G, die einen Weg von Knoten 1 nach Knoten 2 besitzen, gehört zu DL.
  - Der Graph G werde durch seine Adjazenzmatrix repräsentiert.
  - Der Nachweis ist sehr kompliziert.

D-REACHABILITY die Menge aller **gerichteten** Graphen G, die einen Weg von Knoten 1 nach Knoten 2 besitzen, gehört wahrscheinlich nicht zu DL.

Wie mächtig ist DL? (2/3)

(a) Die I-O Turingmaschine M arbeite mit Speicherplatz s. Dann ist die Laufzeit von M für Eingaben der Länge n durch $n \cdot 2^{O(s(n))}$ beschränkt.

(b) Es gelte $s(n) \geqslant \log_2 n$. Dann ist

$$\text{DSPACE}(s) \subseteq \bigcup_{k \in \mathbb{N}} \text{DTIME}(2^{k \cdot s}).$$

(c) Als Konsequenz folgt $\text{DL} \subseteq P$.

- Teil (b) folgt aus Teil (a), wenn wir die Annahme $s(n) \geqslant \log_2 n$ beachten.
- Teil (c) folgt aus Teil (b).

Idee für Teil (a)

Bei zu langer Laufzeit sollte eine platz-beschränkte Machine in eine Endlosschleife geraten!

Wie mächtig ist DL? (3/3)

Eine **Konfiguration** besteht aus

1. dem gegenwärtigen Zustand,
2. der Position des Lesekopfs,
3. der Position des Kopfs auf dem Arbeitsband und
4. dem Inhalt des Arbeitsbands.

- Wenn eine I-O Turingmaschine M eine Konfiguration zweimal annimmt, dann steckt M in einer Endlosschleife.

- Für Speicher s und Eingabelänge n gibt es höchstens $O(ns \cdot 2^{O(s)}) = n \cdot 2^{O(s)}$ Konfigurationen.

- Die Laufzeit von M ist durch die Anzahl der Konfigurationen beschränkt.

Wie mächtig ist NL?

- D-REACHABILITY gehört wahrscheinlich nicht zu DL, wohl aber zu NL.
  - Eine nichtdeterministische TM rät einen Weg von Knoten 1 nach Knoten 2.

  Gehört D-UNREACHABILITY zu NL? Ein gerichteter Graph gehört zu D-UNREACHABILITY, wenn G keinen Weg von Knoten 1 nach Knoten 2 besitzt.
- Das Wortproblem für NFAs (akzeptiert ein gegebener NFA eine gegebene Eingabe) gehört zu NL, wahrscheinlich aber nicht zu DL.
- 2-SAT, das Erfüllbarkeitsproblem für KNF-Formeln mit höchstens zwei Literalen pro Klausel gehört zu NL. (Warum? Begründung folgt später.)
- Das Entscheidungsproblem „Ist ein ungerichteter Graph bipartit“ gehört zu NL. (Warum? Die Begründung folgt später.)

Ist DL eine echte Teilmenge von NL?

LOGSPACE-Reduktionen

LOGSPACE-Reduktionen

Ist D-REACHABILITY deterministisch mit logarithmischem Speicherplatz erkennbar?

- Wir zeigen, dass eine positive Antwort die Gleichheit DL = NL erzwingt, und die wahrscheinliche Antwort ist also negativ.
- Insbesondere zeigen wir, dass D-REACHABILITY eine schwierigste Sprache in NL ist, wobei „Schwierigkeit“ durch **LOGSPACE-Reduktionen** gemessen wird.

L ist LOGSPACE-reduzierbar auf K (geschrieben $L \leqslant_{\mathrm{LOG}} K$), falls es eine (deterministische) I-O Turingmaschine M mit logarithmischem Speicherplatzbedarf gibt, so dass für alle Eingaben $w \in \Sigma_1^*$,

$$w \in L \;\Leftrightarrow\; M(w) \in K.$$

NL-Vollständigkeit

(a) Die Sprache K heißt NL-hart, falls $L \leqslant_{\mathrm{LOG}} K$ für alle Sprachen $L \in \mathrm{NL}$ gilt.

(b) Die Sprache K heißt genau dann NL-vollständig, wenn $K \in \mathrm{NL}$ und wenn K NL-hart ist.

Die wesentlichen Eigenschaften der LOGSPACE-Reduktion:

- Wenn $L \leqslant_{\mathrm{LOG}} K$ und wenn $K \in \mathrm{DL}$, dann ist auch $L \in \mathrm{DL}$.
  - Als Konsequenz: DL = NL, wenn irgendeine NL-vollständige Sprache in DL liegt.
    Die Sprache K sei NL-vollständig. Dann sind äquivalent

$$K \in \mathrm{DL} \Leftrightarrow \mathrm{DL} = \mathrm{NL}.$$

- Wenn $M \leqslant_{\mathrm{LOG}} K$ und $K \leqslant_{\mathrm{LOG}} L$, dann ist $M \leqslant_{\mathrm{LOG}} L$.
  - Als Konsequenz: Wie zeigt man neue NL-vollständige Sprachen?
    Wenn K NL-hart ist und wenn $K \leqslant_{\mathrm{LOG}} L$, dann ist auch L NL-hart.

D-REACHABILITY ist NL-vollständig

D-REACHABILITY ist NL-vollständig (1/2)

Wir führen das zentrale Konzept
des **Berechnungsgraphen** $G_M(w)$ von M auf Eingabe w ein.

- Die Knoten von $G_M(w)$ entsprechen den Konfigurationen.
- Wir fügen eine Kante von Konfiguration c nach Konfiguration d ein, wenn M (mit Eingabe w) in einem Schritt von c nach d gelangen kann.

$G_M(w)$ kann von einer deterministischen I-O Turingmaschine mit logarithmischem Speicherplatz berechnet werden.

- Wir müssen deterministisch, auf logarithmischem Platz, feststellen, ob ein 1-Schritt Übergang von einer Konfiguration c_1 zu einer Konfiguration c_2 möglich ist.
- Kein Problem, denn der Speicher in c_1 wie auch in c_2 ist logarithmisch.

D-REACHABILITY ist NL-vollständig (2/2)

- D-REACHABILITY liegt offensichtlich in NL.

- Zeige: $L \leqslant_{\mathrm{LOG}}$ D-REACHABILITY für eine beliebige Sprache $L \in$ NL.
  - $L = L(M)$ für eine nichtdeterministische I-O Turingmaschine M mit logarithmischem Speicher.
  - Der Berechnungsgraph $G_M(w)$ kann von einer deterministischen I-O TM berechnet werden.
  - Zusätzlich füge eine Kante von jeder akzeptierenden Haltekonfiguration zu einem neuen Knoten ein, dem wir den „Namen“ 2 geben. Der der Anfangskonfiguration entsprechende Knoten erhält den Namen 1.
  - $w \in L \Leftrightarrow G_M(w) \in$ D-REACHABILITY.

Weitere NL-vollständige Sprachen

1. Das Wortproblem für NFA
2. 2-SAT
3. Das Entscheidungsproblem „Ist G bipartit?“

Später zeigen wir: Wenn L NL-vollständig ist. dann ist auch die Komplementsprache NL-vollständig.

Wie mächtig ist NL?

$\mathrm{DL} \subseteq \mathrm{NL} \subseteq \mathrm{P} \subseteq \mathrm{NP}$.

- Die Beziehungen $\mathrm{DL} \subseteq \mathrm{NL}$ sowie $\mathrm{P} \subseteq \mathrm{NP}$ sind offensichtlich.
- Es genügt der Nachweis von $\mathrm{NL} \subseteq \mathrm{P}$.
  - Wenn $L \leqslant_{\mathrm{LOG}} K$ und $K \in \mathrm{P}$, dann auch $\in \mathrm{P}$.
    - Warum?
    - Deterministische I-O Turingmaschinen mit logarithmischem Speicher besitzen eine polynomielle Laufzeit.
  - Für eine beliebige Sprache L in NL gilt $L \leqslant_{\mathrm{LOG}}$ D-REACHABILITY.
  - Da D-REACHABILITY in P liegt, folgt somit auch $L \in \mathrm{P}$.

Der Satz von Savitch

Der Satz von Savitch

(a) D-REACHABILITY $\in$ DSPACE$(\log_2^2 n)$.

(b) Die Funktion s sei platz-konstruierbar. Dann ist

$$\text{NSPACE}(s) \subseteq \text{DSPACE}(s^2)$$

und insbesondere folgt

$$\text{NL} \subseteq \text{DSPACE}(\log_2^2 n) \quad \text{und} \quad \text{PSPACE} = \text{NPSPACE}.$$

- Leider können wir D-REACHABILITY weder mit Tiefensuche noch mit Breitensuche lösen:
  - Sowohl der Stack der Tiefensuche wie auch die Queue der Breitensuche verlangen bis zu linearem Speicherplatz.

  Wir erfinden ein neues speicher-effizientes Traversierungsverfahren.
- Teil (b) ist direkte Konsequenz von Teil (a):
  D-REACHABILITY nimmt eine Schlüsselrolle ein.

D-REACHABILITY $\in$ DSPACE$(\log_2^2 n)$ (1/3)

- Der **Algorithmus von Savich** überprüft ob der Eingabegraph G einen Weg von **Knoten** u nach **Knoten** v der **genauen Länge** m besitzt.
- Anfänglich ist $u = 1$ und $v = 2$.

1. Der Graph G habe n Knoten.
2. Für jedes $m \leqslant n - 1$ rufe den Algorithmus von Savitch mit den Parametern $u = 1$ und $v = 2$ auf.
3. Akzeptiere genau dann, wenn der Algorithmus von Savitch mindestens einmal akzeptiert.

D-REACHABILITY $\in$ DSPACE$(\log_2^2 n)$ (2/3)

Der Algorithmus von Savitch.

1. Die Eingaben seien der Graph G sowie die Knoten u und v und die Weglänge m.
2. Wenn $m = 1$, dann akzeptiere, falls (u, v) eine Kante von G ist und verwerfe ansonsten.
3. Für **alle** Knoten w führe zwei **rekursive** Aufrufe mit den jeweiligen Parametern u, w und $\lceil \frac{m}{2} \rceil$ beziehungsweise w, v und $\lfloor \frac{m}{2} \rfloor$ durch.
4. Akzeptiere, wenn es einen Knoten w gibt für den beide Aufrufe akzeptieren und verwerfe ansonsten.

D-REACHABILITY $\in$ DSPACE$(\log_2^2 n)$ (3/3)

Analyse von Speicherplatz und Laufzeit:

- Der Algorithmus von Savitch benötigt einen Stack der maximalen Höhe $\lceil \log_2 n \rceil$.
- Jedes Element des Stacks entspricht einem Knoten und nimmt ebenfalls logarithmischen Speicher in Anspruch.
- Insgesamt benötigen wir Speicherplatz $O(\log_2^2 n)$.
- Die Laufzeit ist höchstens exponentiell im Speicherplatz und deshalb durch $2^{O(\log_2^2 n)}$ beschränkt.

Warum gilt NSPACE$(s) \subseteq$ DSPACE(s^2), wenn s platzkonstruierbar ist?

$\mathrm{NSPACE}(s) \subseteq \mathrm{DSPACE}(s^2)$

Es sei $s(n) \geqslant \log_2 n$ und s sei platz-konstruierbar.

- M sei eine beliebige nichtdeterministische Turingmaschine mit Speicherplatzbedarf höchstens s und w sei eine Eingabe.
- Wir konstruieren eine deterministische Turingmaschine M^*, die M auf Speicherplatz $O(s^2)$ simuliert.
  - Da s platz-konstruierbar ist, kann M^* einen Speicherplatz von $s(n)$ Zellen abstecken und damit alle Konfigurationen von M systematisch erzeugen.
  - M^* ruft den Algorithmus von Savitch (für den Berechnungsgraphen $G_M(w)$ und Knoten 1,2 entsprechend gewählt) auf und akzeptiert genau dann, wenn ein Weg von Knoten 1 nach Knoten 2 gefunden wird.

M^* hat nur den Speicherplatzbedarf $O(s^2)$:

Der Algorithmus von Savitch benötigt Speicher $O(\log_2^2 K)$
für $K = 2^{O(s(n))}$ Konfigurationen von M, denn $G_M(w)$ hat K Knoten.

Der Satz von Immerman und Szlepscenyi

Der Satz von Immerman und Szelpscenyi (1/3)

(a) D-UNREACHABILITY, das Komplement von D-REACHABILITY, liegt ebenfalls in NL.

(b) Die Funktion s sei platz-konstruierbar. Dann ist

$$\text{NSPACE}(s) = \text{coNSPACE}(s),$$

wobei

$$\text{coNSPACE}(s) = \{\overline{L} \mid L \in \text{NSPACE}(s)\}$$

genau aus den Komplementen der Sprachen in NSPACE(s) besteht.

- **Nichtdeterministischer Speicherplatz ist abgeschlossen unter Komplementbildung.**
- Teil (b) wird sich als direkte Konsequenz von Teil (a) herausstellen.

Der Satz von Immerman und Szelpscenyi (1/3)

Der Graph G sei die Eingabe für D-UNREACHABILITY.

1. Angenommen, wir können das Anzahlproblem in NL lösen, also
   die Anzahl m der von Knoten 1 aus erreichbaren Knoten bestimmen.
   Wir konstruieren eine nichtdeterministische Maschine M, die D-UNREACHABILITY mit logarithmischem Speicher akzeptiert.
2. Im zweiten Schritt lösen wir das Anzahlproblem in NL.

- m Knoten seien vom Knoten 1 aus erreichbar.
- Die Idee und ihre Implementierung.
  - M rät **sukzessive** von 1 aus erreichbare Knoten $v_1 < v_2 < \cdots < v_m$. Wie?
    - Für v_i rät M einen Weg $1 \stackrel{*}{\rightarrow} v_i$.
    - Wenn dies erfolgreich war, rät M danach v_{i+1} mit $v_i < v_{i+1}$ und wiederholt ihr Vorgehen für v_{i+1}.
  - Wenn Knoten 2 von $v_1, \ldots, v_m$ verschieden ist, dann ist Knoten 2 **nicht** von Knoten 1 aus erreichbar und M akzeptiert.

Der Satz von Immerman und Szelpscenyi (2/3)

Wir müssen das **Anzahlproblem** für den Graphen G lösen.

- Knoten 1 erreiche m_i Knoten durch Wege der Länge höchstens i.
  - Offensichtlich ist $m_0 = 1$ und $m_{n-1} = m =$ Anzahl der von 1 aus erreichbaren Knoten.
- Zeige, dass m_{i+1} in NL berechnet werden kann, wenn m_i bekannt ist.
  - Setze zu Anfang $m_{i+1} = 1$, denn Knoten 1 erreicht sich selbst.
  - Verarbeite die Knoten k in **aufsteigender** Reihenfolge.
    - ⋆ Rate sukzessive Knoten $v_{i,1} < \cdots < v_{i,m_i}$ und verifiziere, dass jeder geratene Knoten vom Knoten 1 durch einen Weg der Länge höchstens i erreichbar ist.
    - ⋆ Wenn die Verifikation für $v_{i,r}$ erfolgreich war, und $(v_{i,r}, k)$ eine Kante ist, erhöhe m_{i+1} um Eins und beginne mit Knoten $k + 1$.
    - ⋆ Wenn die Verifikation für $v_{i,r}$ erfolgreich war, aber $(v_{i,r}, k)$ keine Kante ist, dann rate $v_{i,r+1}$ und fahre fort.
    - ⋆ Wenn die Verifikation für $v_{i,r}$ scheitert, dann verwerfe.

Der Satz von Immerman und Szelpscenyi (3/3)

- Die Sprache L werde von einer nichtdeterministischen Turingmaschine M mit Speicherplatzbedarf s erkannt.

  Es gelte $s \geqslant \log_2 n$ und s sei platzkonstruierbar.
- Wir bauen eine nichtdeterministische TM M^*, die das Komplement $\overline{L}$ mit Speicherplatzbedarf $O(s)$ erkennt.

- M^* muss überprüfen, ob M für eine Eingabe w einen Weg von der Startkonfiguration 1 zur akzeptierenden Haltekonfiguration 2 besitzt.

  M^* akzeptiert genau dann, wenn es keine akzeptierende Berechnung gibt, wenn also $G_M(w)$ zu D-UNREACHABILITY gehört.

- M^* wendet den Algorithmus von Immerman und Szelepscenyi für den Berechnungsgraphen $G_M(w)$ an.
  - M^* muss klären, welche Kanten in $G_M(w)$ einzusetzen sind.
  - Platz $O(s)$ reicht, da die Konfigurationen von M nur Platz $O(s)$ benötigen.
  - Also genügt insgesamt Platz $O(s)$ für M^*.

PSPACE-Vollständigkeit

PSPACE-Vollständigkeit

P $\stackrel{?}{=}$ PSPACE: Wir möchten die schwierigsten Sprachen in PSPACE in Bezug auf die polynomielle Reduktion bestimmen:

(a) Eine Sprache L heißt PSPACE-hart, falls $K \leqslant_P L$ für alle Sprachen $K \in$ PSPACE gilt.

(b) L heißt PSPACE-vollständig, falls L PSPACE-hart ist und falls $L \in$ PSPACE.

Die wesentlichen Eigenschaften der polynomiellen Reduktion:

- Wenn $L \leqslant_P K$ und wenn $K \in$ P, dann ist auch $L \in$ P.
  - Als Konsequenz: P = PSPACE, wenn irgendeine PSPACE-vollständige Sprache in P liegt.
    Die Sprache L sei PSPACE-vollständig. Dann sind äquivalent

$$L \in \mathrm{P} \Leftrightarrow \mathrm{P} = \mathrm{PSPACE}.$$

- Wenn $M \leqslant_P K$ und $K \leqslant_P L$, dann ist $M \leqslant_P L$.
  - Als Konsequenz: Wie zeigt man neue PSPACE-vollständige Sprachen?
    Wenn L PSPACE-hart ist und wenn $L \leqslant_P K$, dann ist auch K PSPACE-hart.

QBF: Quantifizierte Boolesche Formeln

Quantifizierte Boolesche Formeln

Eine quantifizierte Boolesche Formel ϕ besteht aus einem Quantorenteil

mit Existenz- und Allquantoren

gefolgt von einer aussagenlogischen Formel α. Jede in α vorkommende Variable wird von genau einem Quantor gebunden.

$$\text{QBF} = \{\langle\phi\rangle \mid \phi \text{ ist eine wahre quantifizierte Boolesche Formel }\}.$$

- Die Formel $\phi \equiv \exists p \forall q((p \vee \neg q) \wedge (\neg p \vee q))$ ist falsch, denn
  - sie drückt die Äquivalenz von p und q aus,
  - aber es gibt keinen Wahrheitswert für p, der mit 0 und 1 äquivalent ist.
- Die Formel $\psi \equiv \forall p \exists q((p \vee \neg q) \wedge (\neg p \vee q))$ ist hingegen wahr,
  - denn zu jedem Wahrheitswert für p gibt es einen äquivalenten Wahrheitswert für q.
  - Also ist $\psi \in$ QBF.

QBF ist PSPACE-vollständig (1/4)

QBF liegt in PSPACE

- Interpretiere QBF als ein Spiel des Existenzquantors $\exists$ gegen den Allquantor $\forall$.
- Bei n Variablen ist das Spiel nach höchstens n Zügen vorbei: Der Spielbaum hat Tiefe höchstens n.
- Werte den Spielbaum in DSPACE(n) aus.

***QBF* ist PSPACE-hart**:

- Die Sprache $L \in$ PSPACE werde von einer deterministischen Turingmaschine M mit Speicherplatzbedarf $O(n^k)$ berechnet.
- Zeige $L \leqslant_P QBF$: Baue für jede Eingabe w von L in polynomieller Zeit eine quantifizierte Boolesche Formel ϕ_w mit

$$w \in L \Leftrightarrow \phi_w \text{ ist wahr.}$$

QBF ist PSPACE-vollständig (2/4)

Wir müssen eine Formel ϕ_w bauen mit

$$w \in L \Leftrightarrow \phi_w \text{ ist wahr.}$$

$L = L(M)$ für eine deterministischen TM M mit Platzbedarf $s = O(n^k)$.

- Wie im NP-Vollständigkeitsbeweis von KNF-SAT führen wir aussagenlogischen Variablen wie
  - $Kopf^t(z)$ für die Kopfposition. $Kopf^t(z)$ *soll* genau dann wahr ist, wenn der Kopf zum Zeitpunkt t auf Zelle z steht,
  - $Zelle^t(z, a)$ für den Zelleninhalt. $Zelle^t(z, a)$ *soll* genau dann wahr ist, wenn die Zelle z zum Zeitpunkt t mit dem Buchstaben a beschriftet ist und
  - $Zustand^t(q)$ für den aktuellen Zustand. $Zustand^t(q)$ *soll* genau dann wahr ist, wenn q der Zustand zum Zeitpunkt ist.
- Aber M kann bis zu $\mathbf{2^{O(n^k)}}$ Schritte ausführen!!!

QBF ist PSPACE-vollständig (3/4)

(a) Für Konfigurationen c und d schreiben wir

$$c \xrightarrow{t} d,$$

wenn M nach höchstens t Schritten, von Konfiguration c aus startend, Konfiguration d erreicht.

(b) Baue eine höchstens polynomiell lange Formel $\psi_t(c, d)$, die genau dann wahr ist, wenn $c \xrightarrow{t} d$ gilt.

- Zur Erinnerung: M benötigt höchstens Speicher s.
- c_0 sei die Anfangskonfiguration und c_a sei die eindeutig bestimmte akzeptierende Haltekonfiguration von M. Dann

$$w \in L \Leftrightarrow \psi_{2^s}(c_0, c_a).$$

- Setze

$$\phi_w = \psi_{2^s}(c_0, c_a).$$

QBF ist PSPACE-vollständig (4/4)

Und jetzt der Clou: Drücke $\psi_{2^{t+1}}$ durch ψ_{2^t} aus.

$$\psi_{2^{t+1}}(c,d) \equiv \exists e \forall f \forall g(\ ((f = c \wedge g = e) \vee (f = e \wedge g = d)) \rightarrow \psi_{2^t}(f,g)\ ).$$

- $\exists e$ entspricht der **Folge** der Existenz-Quantoren zu den Variablen
  - Kopfposition, Zelleninhalt und Zustand der Konfiguration e.

  Ähnliches gilt für $\forall f$ und $\forall g$.
- $\psi_{2^{t+1}}(c,d)$: Eine Berechnung der Länge höchstens 2^{t+1} kann in zwei Berechnungen der Länge höchstens 2^t aufgespalten werden.
  - Der All-Quantor erlaubt eine **simultane** Überprüfung der beiden Berechnungen von c nach e und von e nach d.
- Dementsprechend wächst die Formellänge additiv um höchstens $O(n^k)$, also höchstens um den Speicherplatzbedarf von M.

$O(n^{2k})$ ist eine obere Schranke für die Länge der Formel $\phi_w = \psi_{2^s}(c_0, c_a)$.

Das GEOGRAPHIE-Spiel

Im Kinderspiel „Geographie“, müssen zwei Spieler abwechselnd noch nicht genannte Städtenamen wählen, wobei jede Stadt mit dem Endbuchstaben der zuvor genannten Stadt beginnen muß.

- Die Eingabe: Ein gerichteter Graph $G = (V, E)$ und ein ausgezeichneter Knoten $s \in V$.
- Die Spielregeln:
  - Zwei Spieler Alice und Bob wählen abwechselnd eine noch nicht benutzte Kante aus E.
  - Alice fängt an und wählt eine Kante mit Startknoten s.
  - Jede anschließend gewählte Kante muß im Endknoten der zuvor gewählten Kante beginnen.
  - Der Spieler, der als erster keine solche unbenutzte Kante mehr findet, verliert das Spiel.

Geographie ist PSPACE-vollständig.

PSPACE und die Komplexität von 2-Personen Spielen

- Wir haben schon beobachtet, dass sich QBF als ein Spiel zwischen dem Existenzquantor und dem Allquantor auffassen lässt.
- Wenn Spiele auf beliebige Spielgröße verallgemeinert werden, wie etwa auf $n \times n$ Bretter, sind viele weitere Vollständigkeits-Ergebnisse bekannt **www.ics.uci.edu/~eppstein/cgt/hard.html**:
  - Dame, Go, Othello , Schach, Sokoban sind Beispiele PSPACE-harter Spiele.
- PSPACE-Härte ist ein Gütesiegel, dass es sich um ein interessantes, weil sehr schwieriges Spiel handelt.

Leider kann die Komplexitätstheorie keine Aussagen über Spiele machen, deren Spielgröße fest ist wie etwa eine fixierte Brettgröße.

Das Universalitätsproblem für reguläre Ausdrücke und NFAs

Das Universalitätsproblem (1/4)

- Sei R ein regulärer Ausdruck. Entscheide, ob $L(R) \neq \Sigma^*$ gilt.
- Im Universalitätsproblem für NFA ist zu entscheiden, ob $L(N) \neq \Sigma^*$ für einen NFA N gilt.

Ist ein regulärer Ausdruck oder ein NFA trivial?

Diese Frage sollte einfach sein, schließlich wird für NFA doch nur gefragt, ob ein einziger Zustand ausreicht!

Die Universalität für reguläre Ausdrücke ist ebenso PSPACE-hart wie die Universalität für NFA.

Das Universalitätsproblem (2/4)

- Eine deterministische TM arbeitet „**in-place**“, wenn sie ihren Eingabebereich nie verlässt. Dann gibt es eine in-place TM M, so dass das Wortproblem für M PSPACE-vollständig ist.
  - Wähle für M eine in-place TM, die das QBF-Problem löst.
  - Wir können zusätzlich verlangen, dass M für jede Eingabe der Länge n mindestens 2^n Schritte benötigt.

- **Unser Ziel**: Konstruiere in polynomieller Zeit einen regulären Ausdruck R_w für w von M, so dass

  *R_w alle Worte akzeptiert, die **nicht** mit der Konfigurationenfolge einer akzeptierenden Berechnung von M für w übereinstimmen.*

Damit ist die Behauptung gezeigt, denn

$$w \in L(M) \quad \Leftrightarrow \quad \text{es gibt eine akzeptierende Berechnung von } M \text{ auf Eingabe } w$$
$$\Leftrightarrow \quad L(R_w) \neq \Sigma^*.$$

Das Universalitätsproblem (3/4)

Um Konfigurationen zu kodieren, benutzen wir die Symbole

- $\#$, um Konfigurationen voneinander zu trennen,
- $[q, a] \in Q \times \Sigma$, um anzugeben, dass a im Zustand q gelesen wird.

Nur die Konfigurationenfolge einer akzeptierenden Berechnung wird verworfen, wenn

1. die Anfangskonfiguration nicht von der Form

$$\#[q_0, w_1]w_2 \cdots w_n\#$$

   ist oder
2. keine Konfiguration der Konfigurationenfolge den Buchstaben $[q_f, \gamma]$ für irgendein $\gamma \in \Gamma$ und irgendeinen akzeptieren Zustand q_f enthält oder
3. die Folge nicht mit dem Trennsymbol $\#$ endet oder
4. wenn sich Bandinhalt oder Zustand für aufeinanderfolgende Konfigurationen auf nicht-legale Weise ändert.

Das Universalitätsproblem (4/4)

Konstruiere R_w als Vereinigung von vier regulären Ausdrücken der Länge $O(|w|)$, einen Ausdruck für jeden der vier Fälle.

Z.B. im vierten Fall, wenn sich Bandinhalt oder Zustand für aufeinanderfolgende Konfigurationen auf nicht-legale Weise ändert:

- M arbeitet in-place, die Länge des Bandes stimmt also mit $|w|$ überein.
- In einer **legalen** Folge y von Konfigurationen ist y_{i+n+1} eine Funktion von $y_{i-1}y_iy_{i+1}$.
- Insbesondere ist x genau dann **keine legale** Konfigurationenfolge, wenn es eine Position i gibt mit
  - $x_{i+n+1} \neq x_i$, obwohl der Kopf nicht auf Position i gestanden hat
  - oder x_{i+n+1} wird falsch aktualisiert.

Das Universalitätsproblem: Die Konsequenzen

(a) Das Äquivalenzproblem für reguläre Ausdrücke ist PSPACE-hart.

(b) Es gelte P $\neq$ PSPACE. Dann ist es nicht möglich, für einen gegebenen regulären Ausdruck der Größe m, die Größe eines minimalen regulären Ausdrucks innerhalb des Faktors $o(m)$ effizient zu approximieren.

- Behauptung (a) folgt, da schon die Frage, ob die Ausdrücke R und Σ^* äquivalent sind, PSPACE-hart ist.
- Zur Behauptung (b):
  - Wenn M die Eingabe w verwirft, dann ist $R_w = \Sigma^*$, und es gibt einen Ausdruck **konstanter Länge** der mit R_w äquivalent ist.
  - Wenn M die Eingabe w akzeptiert, dann ist $L(R_w) = \Sigma^* \setminus \{y\}$ für die Konfigurationenfolge y der akzeptierenden Berechnung.
    - Wie lang muss R_w in diesem Fall mindestens sein?
    - M benötigt Zeit $2^{|w|}$ nach Annahme.
    - Ein DFA hat deshalb die Länge mindestens $|y| \geqslant 2^{|w|}$ Zustände. Ein regulärer Ausdruck hat also die **Länge mindestens** $|w|$.
  - In polynomieller Zeit kann nicht zwischen konstanter Länge und der Länge mindestens w unterschieden werden.

Das Universalitätsproblem und NFAs

- Das Universalitätsproblem für NFAs ist aus denselben Gründen ebenfalls PSPACE-hart.
  - Deshalb ist auch das Äquivalenzproblem für NFA PSPACE-hart.
  - Eine effiziente Approximation der minimalen Zustandszahl, wenn ein NFA der Größe m gegeben ist, ist innerhalb eines Faktors $o(m)$ nicht möglich.

NFAs oder reguläre Ausdrücke sind im Hinblick auf

Universalität, Äquivalenz oder Minimierung

sehr viel schwieriger als DFA.

Die Chomsky-Hierarchie

Die Chomsky-Hierarchie

1. Grammatiken ohne jede Einschränkung heißen Typ-0-Grammatiken. Die entsprechende Sprachenfamilie ist

$$\mathcal{L}_0 = \Big\{ L(G) \mid G \text{ ist vom Typ } 0 \Big\}$$

2. Eine Grammatik G mit Produktionen der Form

$$u \to v \quad \text{mit } |u| \leqslant |v|$$

heißt Typ 1 oder kontextsensitiv. Die zugehörige Sprachenfamilie ist

$$\mathcal{L}_1 = \Big\{ L(G) \mid G \text{ ist vom Typ } 1 \Big\} \cup \Big\{ L(G) \cup \{\epsilon\} \mid G \text{ ist vom Typ } 1 \Big\}$$

3. Kontextfreie Grammatiken werden als Grammatiken vom Typ 2 bezeichnet. Die zugehörige Sprachenfamilie ist

$$\mathcal{L}_2 = \Big\{ L(G) \mid G \text{ hat Typ } 2 \Big\}$$

4. Eine reguläre Grammatik heißt auch Typ-3-Grammatik. Die zugehörige Sprachenfamilie ist

$$\mathcal{L}_3 = \Big\{ L(G) \mid G \text{ hat Typ } 3 \Big\}$$

Die Chomsky Hierarchie und Platzkomplexität

(a) $\mathcal{L}_0$ ist die Klasse aller rekursiv aufzählbaren Sprachen.

(b) Es gilt $\mathcal{L}_1 = \text{NSPACE}(n)$ und $\mathcal{L}_1$ ist die Klasse aller Sprachen, die von nichtdeterministischen Turingmaschinen auf linearem Platz erkannt werden.

(c) $\text{NL} \subseteq \text{LOGCFL} \subseteq \text{DSPACE}(\log_2^2 n)$. LOGCFL ist der Abschluß kontextfreier Sprachen unter LOGSPACE-Reduktionen. Insbesondere gilt also $\mathcal{L}_2 \subseteq \text{DSPACE}(\log_2^2 n)$.

(d) Die Klasse der regulären Sprachen stimmt mit der Klasse $\text{DSPACE}(0)$ überein, es gilt also $\mathcal{L}_3 = \text{DSPACE}(0)$.

(e) $\mathcal{L}_3 \subset \mathcal{L}_2 \subset \mathcal{L}_1 \subset \mathcal{L}_0$ und alle Inklusionen sind echt.

Typ-0 Grammatiken

- $L(G)$ ist für jede Grammatik G rekursiv aufzählbar.
  - $w \in L(G) \Leftrightarrow S \overset{*}{\to} w$: Zähle alle möglichen Ableitungen auf:
    - ★ Wenn $w \in L(G)$ werden wir eine Ableitung finden.
    - ★ Wenn $w \notin L(G)$, hält unser Programm nicht, aber was soll's?

- Andererseits sei $L = L(M)$:
  Wir müssen eine Grammatik G mit $L(M) = L(G)$ konstruieren.
  - Berechnungen von M beginnen mit der Eingabe w,
    Ableitungen von w enden mit w .
  - Also sollten wir die Grammatik G so konstruieren, dass
    die Berechnungen von M „rückwärts“ simuliert werden.
    - ★ Die Grammatik G wird die Konfigurationen

      $\alpha_1 \cdots \alpha_{i-1}\, q\, \alpha_i \cdots \alpha_N$ *(Die Maschine befindet sich im Zustand q, hat den Bandinhalt $\alpha_1 \cdots \alpha_N$ erstellt und liest das ite Symbol des Bands)*

      rückwärts konstruieren.
    - ★ Wir nehmen an, dass M nur einen akzeptierenden Zustand q_a besitzt und dass akzeptierende Berechnungen mit dem leeren Band enden.
  - G besitzt die Variablenmenge $(\Gamma \setminus \Sigma) \cup Q \cup \{\epsilon\}$ und q_a als Startsymbol.

Die Konstruktion der Typ-0 Grammatik (1/2)

- Zuerst wird der von M benutzte Bandbereich durch die Produktionen $q_a \to \mathtt{B} q_a \mid q_a \mathtt{B}$ erzeugt.
- Dann beginnt die Rückwärtsrechnung.
  - Wenn $\delta(q, a) = (q', b, \text{links})$,
    wählen wir die Produktion $q'cb \to cqa$ für alle $c \in \Gamma$.

    Wenn die Konfiguration $* \cdots * q'cb * \cdots *$ schon erzeugt wurde, können wir jetzt die mögliche Vorgänger-Konfiguration $* \cdots * cqa * \cdots *$ erzeugen.
  - Wenn $\delta(q, a) = (q', b, \text{bleib})$,
    fügen wir die Produktion $q'b \to qa$ zu G hinzu.

    Wenn die Konfiguration $* \cdots * q'b * \cdots *$ schon erzeugt wurde, können wir jetzt die mögliche Vorgänger-Konfiguration $* \cdots * qa * \cdots *$ erzeugen.
  - Wenn $\delta(q, a) = (q', b, \text{rechts})$,
    dann nehmen wir die Produktion $bq' \to qa$ auf.

    Wenn die Konfiguration $* \cdots * bq' * \cdots *$ schon erzeugt wurde, können wir jetzt die mögliche Vorgänger-Konfiguration $* \cdots * qa * \cdots *$ erzeugen.
- Am Ende der Ableitung haben wir eine Konfiguration $\mathtt{B}^k q_0 w \mathtt{B}^s$ erzeugt.

Die Konstruktion der Typ-0 Grammatik (2/2)

Am Ende der Ableitung haben wir eine Konfiguration $\mathtt{B}^k q_0 w \mathtt{B}^s$ erzeugt.

- Die zusätzlichen Produktionen

$$\begin{array}{rcll} q_0 & \to & \epsilon_1 & \\ \mathtt{B}\,\epsilon_1 & \to & \epsilon_1 & \\ \epsilon_1 & \to & \epsilon_2 & \\ \epsilon_2 a & \to & a\epsilon_2 & \text{für } a \in \Sigma \\ \epsilon_2 & \to & \epsilon_3 & \\ \epsilon_3\,\mathtt{B} & \to & \epsilon_3 & \\ \epsilon_3 & \to & \text{das leere Wort} & \end{array}$$

  garantieren jetzt, dass das Wort w abgeleitet wird.
- Wenn andererseits G eine Ableitung des Worts w besitzt, dann hat die Ableitung die Form

$$q_a \stackrel{*}{\to} \mathtt{B}^k q_0 w \mathtt{B}^s \stackrel{*}{\to} w$$

  und $w \in L(M)$ folgt.

Typ-1 Grammatiken

Eine kontextsensitive Grammatik ist längenerhaltend:
Die rechte Seite v einer kontextsensitiven Produktion $u \to v$
ist mindestens so lang wie die linke Seite u.

- Für jede Ableitung $S \overset{*}{\to} w$ sind alle zwischenzeitlich erzeugten Strings in ihrer Länge durch $|w|$ nach oben beschränkt.
- Eine mögliche Ableitungsfolge kann auf Platz $O(|w|)$ geraten und verifiziert werden.

- Jede kontextsensitive Sprache kann durch eine nichtdeterministische Turingmaschine auf linearem Platz erkannt werden.
- Gibt es zu jeder Sprache $L \in \text{NSPACE}(n)$ eine Typ-1 Grammatik G mit $L(G) = L$?

Die Konstruktion der Typ-1 Grammatik

Wir haben eine Typ-0 Grammatik G erzeugt, so dass

$$q_a \stackrel{*}{\to} \mathrm{B}^k q_0 w \mathrm{B}^s \Leftrightarrow w \in L(M). \tag{1}$$

gilt. Alle Produktionen in $q_a \stackrel{*}{\to} \mathrm{B}^k q_0 w \mathrm{B}^s$ sind längenerhaltend.

- Wenn die nicht-deterministische TM M mit linearem Platz arbeitet, dann gibt es eine äquivalente in-place TM M':

  Vergrößere das Bandalphabet entsprechend.

- Für M' erhalten wir somit aus (1)

$$q_a \stackrel{*}{\to} q_0 w \Leftrightarrow w \in L(M') = L(M)$$

  - Wenn $L(M) \in \mathrm{NSPACE}(n)$, dann ist $q_0 L(M)$ kontextsensitiv.
  - Übungsaufgabe:
    Wenn $q_0 L(M)$ kontextsensitiv ist, dann ist auch $L(M)$ kontextsensitiv.

NL $\subseteq$ LOGCFL

- Wir zeigen, dass D-REACHABILITY mit einer LOGSPACE-Reduktion auf eine kontextfreie Sprache L reduziert werden kann.
- Wir beschreiben L durch einen Kellerautomaten K, der L akzeptiert.

- K nimmt an, dass die Eingabe w ein Element von $(a^* b^*)^*$ ist und interpretiert ein Teilwort $a^r b^s$ als die Kante von Knoten r nach Knoten s.

  Ein mehrmaliges Auftreten von Kanten ist erlaubt.

- K rät einen Weg von Knoten 1 nach Knoten 2,
  - indem es eine erste Kante $(1, u)$ rät und u auf den Keller legt.
  - Der Knoten v liege gegenwärtig auf dem Keller. K rät eine Kante (v', w), die in der Eingabe nach den bisher geratenen Kanten erscheint.
    - ⋆ Mit Hilfe des Kellers verifiziert K, dass $v = v'$ gilt. Gilt $v \neq v'$, verwirft K.
    - ⋆ K legt w auf den Keller und akzeptiert, wenn $w = 2$.

Wir müssen eine LOGSPACE-Reduktion von D-REACHABILITY auf L angeben.

NL $\subseteq$ LOGCFL und LOGCFL $\subseteq$ DSPACE($\log_2^2 n$)

Eine LOGSPACE-Reduktion von D-REACHABILITY auf L

- Für einen gerichteten Graphen G zähle die Anzahl n der Kanten.
- Danach gib die Kanten, in jeweils beliebiger Reihenfolge, genau $n - 1$ Mal aus.
- Der Kellerautomat K, wenn auf die Ausgabe angesetzt, findet genau dann einen Weg von Knoten 1 nach Knoten 2, wenn ein solcher Weg in G existiert.

LOGCFL $\subseteq$ $DSPACE(\log_2^2 n)$.
Idee: Zeige, dass der Satz von Cocke, Younger und Kasami mit Speicherplatz $O(\log_2^2 n)$ implementiert werden kann.

Nicht-Standard Berechnungen und Speicherplatz

Probabilismus

Eine probabilistische Turingmaschine kann in jedem Schritt aus von bis zu zwei alternativen Anweisungen wählen. Bei zwei Alternativen ist die Wahrscheinlichkeit jeder Alternative genau $1/2$

- Eine probabilistische Turingmaschine M führt viele Berechnungen B aus.
- Wenn die Berechnung B die Wahrscheinlichkeit p_B besitzt, dann akzeptiert M mit Wahrscheinlichkeit

$$p = \sum_{B \text{ ist eine akzeptierende Berechnung}} p_B.$$

- p hat beschränkten Fehler, wenn $|p - \frac{1}{2}| > \varepsilon$ für eine Konstante $\varepsilon > 0$ gilt.
  - Berechnungen mit beschränktem Fehler sind vernünftig.
  - Berechnungen mit unbeschränktem Fehler hingegen sind sehr mächtig und können nichtdeterministische Berechnungen ohne Zeitverlust simulieren.

Probabilistische Zeit versus deterministischen Platz

Sei M eine probabilistische Turingmaschine (mit nicht notwendigerweise beschränktem Fehler). Wenn die worst-case Laufzeit von Berechnungen für Eingaben der Länge n durch $t(n)$ beschränkt ist, dann gilt

$$L(M) \in \text{DSPACE}(t).$$

- Die höchstens $2^{O(t(n))}$ Berechnungen für eine vorgegebene Eingabe x werden nacheinander simuliert.
- Ein Zähler summiert die Wahrscheinlichkeiten akzeptierender Berechnungen (auf $O(t(n))$ Zellen).
- Nachdem alle Berechnungen simuliert sind, wird geprüft, ob der Zähler einen Wert größer $\frac{1}{2}$ hat, und in diesem Fall wird akzeptiert.

Quantenberechnungen

- Einer Quantenberechnung B wird das Produkt $p_B \in \mathbb{C}$ der **Wahrscheinlichkeitsamplituden** der einzelnen Schritte, zugewiesen.
- Die Matrix A_Q der Übergänge zwischen Konfigurationen ist **unitär**.
  - $A_Q[C, C']$ ist die Wahrscheinlichkeitsamplitude des 1-Schritt Übergangs von C nach C'.
  - Eine Matrix A ist unitär, falls $\overline{A} \cdot A =$ Einheitsmatrix.
- Die **Wahrscheinlichkeitsamplitude einer Konfiguration C** ist

$$\tau_C = \sum_{B \text{ führt auf } C} p_B.$$

Die **Wahrscheinlichkeit von C** ist die quadrierte Länge von τ_C, also

$$|\tau_C|^2.$$

Quanten-Zeit versus deterministischem Platz (1/2)

Die von einem Quantenrechner Q akzeptierte Sprache ist

$$L(Q) = \left\{ x \middle| \sum_{C \text{ ist akzeptierende Konfiguration von } Q \text{ auf Eingabe } x} |\tau_C|^2 > \frac{1}{2}, \right\}.$$

- Wenn Q in Zeit t rechnet, bestimme die Einträge des Vektors

$$A_Q^t \cdot e_1$$

und summiere die Wahrscheinlichkeiten akzeptierender Konfigurationen.

e_1 hat nur Nullen, bis auf eine Eins für die der Startkonfiguration entsprechenden Komponente.

  - Der Vektor $A_Q^k \cdot e_1$ listet die Wahrscheinlichkeitsamplituden der k-Schritt Nachfolger der Startkonfiguration auf.

- Wie berechnet man $A_Q^t \cdot v$ platz-effizient? A_Q ist eine $2^{O(t)} \times 2^{O(t)}$ Matrix!
  - Berechne $A_Q^k \cdot v$ „ohne Nachzudenken“ auf Platz $O(k \cdot t)$.

Quanten-Zeit versus deterministischem Platz (1/2)

Wenn ein Quantenrechner Q die Sprache $L(Q)$ in Zeit $t(n)$ akzeptiert, dann ist

$$L(Q) \in \text{DSPACE}(t^2).$$

Hierzu ist die Forderung eines beschränkten Fehlers ebenso nicht notwendig wie die Forderung, dass die Konfigurationsmatrix A_Q unitär ist.

Probabilistische Berechnungen oder Quantenberechnungen in polynomieller Zeit akzeptieren selbst bei unbeschränktem Fehler „nur“ Sprachen in PSPACE.

Zusammenfassung (1/2)

- In $\mathrm{DSPACE}(o(\log_2 \log_2 n))$ berechenbare Sprachen stimmen mit den regulären Sprachen überein.
- Die erste nicht-triviale Klasse ist die Klasse DL aller auf logarithmischem Platz berechenbaren Sprachen.
  - U-REACHABILITY gehört zu DL.
  - D-REACHABILITY ist NL-vollständig,
    also ein für DL schwierigstes Problem in NL.
    - Insbesondere folgt, dass NL in P enthalten ist.
    - Weitere NL-vollständige Probleme sind 2-SAT, das Wortproblem für NFA und der Test auf Bipartitness.
- Der Satz von Savitch zeigt D-REACHABILITY $\in$ DL und allgemein $\mathrm{NSPACE}(s) \subseteq \mathrm{DSPACE}(s^2)$ für platz-konstruierbare Funktionen s.
- Das Komplementverhalten ist „nicht typisch für Nichtdeterminismus“, denn es ist überraschenderweise
  $\mathrm{NSPACE}(s) = \mathrm{coNSPACE}(s)$, falls s platz-konstruierbar ist.

Zusammenfassung (2/2)

- PSPACE lässt sich als die Komplexitätsklasse nicht-trivialer Zwei-Personen Spiele (wie QBF oder Geographie) auffassen.
- Entscheidungsprobleme für reguläre Ausdrücke oder NFA, wie die Universalität, das Äquivalenzproblem oder die Minimierung, haben sich als unanständig schwierig, nämlich als PSPACE-hart herausgestellt.
- Die Klasse PSPACE ist mächtig und enthält alle Entscheidungsprobleme, die durch randomisierte Algorithmen oder Quanten-Algorithmen in polynomieller Zeit lösbar sind.
- Wir haben dann die Chomsky-Hierarchie betrachtet.
  - Die von Typ-0 Grammatiken erzeugbaren Sprachen stimmen mit den rekursiv aufzählbaren Sprachen überein.
  - Die kontextsensitiven Sprachen sind noch zu komplex, da ihr Wortproblem PSPACE-vollständig sein kann.
  - Kontextfreie Sprachen können NL-hart sein, ihr Wortproblem ist aber in $\mathrm{DSPACE}(\log_2^2 n)$ lösbar.