Kapitel 4 Platzkomplexität.

Verbrauchter Speicherplatz ≙ Anzahl Zellen auf Band der TM.
Damit Platz < n Sinn ergibt, darf Eingabe nicht mitgezählt werden. Bei Funktionen auch Ausgabe nicht!

Platzbeschränkter Akzeptor: (det. oder nichtdet.)
- Eingabeband ist read-only, kann aber nicht überschrieben werden
- Arbeitsbänder sind die k≥2 übrigen Bänder.

Beim Übersetzer zusätzlich
- Ausgabeband kann nur geschrieben werden, d.h. Kopf kann nur nach rechts oder stehen bleiben.

Für det. Akzeptor oder Übersetzer T

$SPACE_T(x)$ = max. Anzahl der auf Arbeitsbändern von T bei Input x benutzten Zellen

Für nichtdet. Akzeptor T

$NSPACE_T(x) := \begin{cases} \text{min. Platzbedarf einer akz. Berechnung,} & x \in L(T) \\ \text{min. Platzbedarf aller Berechn.,} & \text{sonst.} \end{cases}$

Platzkomplexitätsklassen

$L := \{A \subseteq \Sigma^* ;$ ex. DTM T mit $L(T)=A$ und $SPACE_T(x) \leq O(\log n)\}$

$PSPACE := \{ \quad (\log n)^{O(1)}$

$NL := \{A \subseteq \Sigma^* ;$ ex. NTM — " — $O(\log n)$

$NPSPACE \quad \{A \subseteq \Sigma^* ;$ ex. NTM — " — $(\log n)^{O(1)}$

$FL := \{f : \Sigma^* \to \Sigma^* ,$ berechenbar mit $SPACE \leq O(\log n)\}$

Platzkomplexitätsklassen haben viel mit Erreichbarkeits-Problemen zu tun.

REACH Problem

Geg. gerichteter Graph $G = (V, E)$, $s, t \in V$.

Frage Ist t von s erreichbar?

Dh. gibt es Weg $s = v_1 \to v_2 \to \cdots \to v_n = t$?

Offenbar ist REACH $\in P$ z.B. mittels Tiefensuche!

Gegeben ein Problem $A \in NL$, so kann ich die Frage $x \in A$ als Instanz von REACH auffassen.

Sei T NTM, die A erkennt mit $SPACE_T(x) \leq k \log n$. $k \geq 1$

Betrachte Graphen $G(x)$ der globalen Zustände von T bei Eingabe x. Es ist $|G(x)| \leq O(n^{k+2})$

OBdA gibt es genau einen akzeptierenden Zustand $t(x)$.

Also ist $x \in A$ gdw $(G(x), s_0(x), t(x)) \in$ REACH

Es folgt: $NL \subseteq P$. Analog: $NPSPACE \subseteq EXP$.

Außerdem ist $NP \subseteq PSPACE$, da der Durchsuch. des Berechnungsbaumes (im Beweis $NP \subseteq EXP$) nur polynomiell viel Platz verbraucht.

Landkarte:

$$L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE \subseteq NPSPACE \subseteq EXP.$$

Echtheit aller Inklusionen offene Probleme, mit einer Ausnahme!

Satz von SAVITCH: REACH kann von DTM mit Platz $O(\log^2 n)$ entschieden werden.

Beweis: Sei Instanz $G=(V,E)$, s, t v.u. REACH wird entschieden durch Reach$(s,t,\lceil \log n \rceil)$, wobei Reach das folgende Programm mit der Spezifikation:

Reach(u,v,j) akzeptiert gdw. es gibt Weg von u nach v der Länge $\leq 2^j$.

```
Procedur Reach(u,v,j)
  if j = 0 then
    if (u = v) or (u,v) ∈ E then accept
      else reject.
  else for each w ∈ V
    IF REACH(u,w,j-1) and REACH(w,v,j-1)
      then accept
      else reject.
```

Korrektheit ist klar, da Weg von s nach t hat Länge $\leq n \leq 2^{\lceil \log n \rceil}$.

Platzbedarf: gespeichert wird der Aufrufstack. Jeder Stack Frame enthält u,v,j, also größe $\leq 3 \log n$.

Rekursionstiefe ist $\leq \log n$, also Speicherbedarf $\leq O(\log^2 n)$ □

Korollar NPSPACE = PSPACE

Sei $A \in$ NPSPACE, akzeptiert durch NTM mit Platz n^k.

Globaler Zustandsgraph hat Größe $2^{n^{k+2}}$,

also REACH darauf lösbar durch DTM mit Platz $O(\log^2(2^{n^{k+2}})) = O(n^{2k+4})$

□

Vollständige Probleme in NL, P und PSPACE

Trivial: jedes Problem in P (oder NL) ist vollständig für P unter $\leq_p$

Sei $A \in P$, $B \in P$ beliebig, $b \in B$, $\bar{b} \in \bar{B}$:

Reduktion $f: x \mapsto \begin{cases} b & x \in A \\ \bar{b} & x \notin A \end{cases}$ zeigt $A \leq_p B$.

Für sinnvollen Begriff von Vollständigkeit in P, NL braucht man feineren Reduktionsbegriff.

Definition A ist logspace-Reduzierbar auf B, $A \leq_{\log} B$, falls es $f \in FL$ gibt mit

$$\forall x \quad x \in A \iff f(x) \in B.$$

A ist P-schwer, wenn $B \leq_{\log} A$ f. alle $B \in P$

A ist P-vollständig, wenn P-schwer und $A \in P$.

Analog für NL.

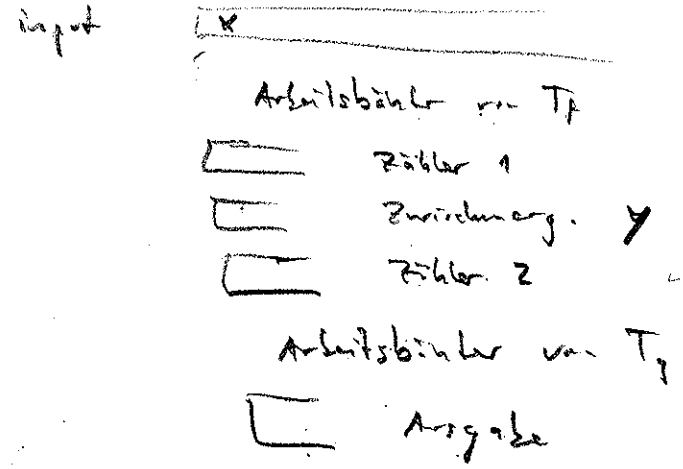
Fakt: Alle bisher betrachteten NP-vollständigen Probleme sind auch NP-vollständig unter $\leq_{\log}$.

Damit $\leq_{log}$-Reduktion vernünftig ist, brauchen wir, dass $\leq_{log}$ transitiv ist, also

SATZ: FL ist unter Komposition abgeschlossen:
Ist $f \in FL$ und $g \in FL$, so auch $g \circ f \in FL$,
mit $g \circ f : x \mapsto g(f(x))$.

Naiver Ansatz: berechne $y = f(x)$, dann $g(y)$
funktioniert nicht, da y möglicherweise zu lang ist.

Seien T_f, T_g DTM, die f, g berechnen.
Konstruiere Maschine die $g \circ f$ berechnet.



Idee: in y wird stets nur ein Symbol von $f(x)$ gespeichert.

Simuliere T_g.
Wenn T_g das i-te Symbol von $y = f(x)$ lesen will:
simuliere T_f, wobei nur eine Zelle von y benutzt wird.
Kopf auf Ausgabe von T_f rechts
erhöhe Zähler 1.
Simuliere, bis Zähler 1 $> i$
dann steht Symbol i von $f(x)$
in Zelle 1 auf y.
→ führe fort mit Simulation
von T_g.

Platzbedarf

$$\begin{aligned} & SPACE_{T_f}(x) \\ + \; & 2 \cdot \log|x| + 1 \qquad \text{Zähler \& } y \\ + \; & Space_{T_g}(x) \\ = \; & O(\log|x|) \end{aligned}$$

Also ist $g \circ f \in FL$. □

Konstruktion im Beweis $NL \subseteq P$ zeigt:

REACH ist NL-vollständig.

In der Übung: 2-SAT ist NL-vollständig.

Problem HORN:

Gegeben Menge P von Propositionen, Menge H von Klauseln der Form

$$p_1, \dots, p_k \to q \qquad p_i, q \in P$$

($k=0 \to q$ Fakt),

Ziel $p \in P$.

Frage Gilt $\vdash p$, wobei $\vdash$ induktiv definiert ist durch

Gilt $\vdash p_1$ und ... und $\vdash p_k$, und ist $p_1, \dots, p_k \to q$ Klausel, so gilt $\vdash q$.

Folgender Algorithmus zeigt HORN $\in P$.

```
M := ∅
done := false
while (¬done) do
    found := false
    for (p1,...,pk → q ∈ H) do
        if {p1,...,pk} ⊆ M then
            if q ∉ M then found := true
            M := M ∪ {q}
    if (¬found) then done := true
return (p ∈ M).
```

In jedem Schleifendurchlauf wird $M \subseteq P$ grösser, $|P| = n$ außer im letzten, maximal $n+1$ Durchläufe.

Innere Schleife hat m Durchläufe, also insgesamt Laufzeit $O(n \cdot m)$, polynomial in Inputgrösse. □

Algorithmus braucht linearen Platz, da M mit $|M| \leq n$ gespeichert werden muss. Geht wahrscheinlich auch nicht besser, denn:

Theorem HORN ist P-vollständig

Sei $A \in P$. z.Z.: $A \leq_{log}$ HORN.

Sei T eine 1-Band DTM, die A akzeptiert, mit $TIME_T(x) \leq p(|x|)$ für ein Polynom $p(\cdot)$, und der Eigenschaft:

(*) für alle x mit $|x| = n$ ist nach t Schritten die Kopfposition $h(t, n)$. (selbe für alle x mit)

Für jedes $t \leq p(n)$ und $i \leq p(n)$ Propositionen

$Z(t, q)$ "Zustand nach t Schritten ist q" $q \in Q$

$B(t, i, a)$ "Bandsymbol an der Stelle i nach t Schritten ist a" $a \in \Sigma$

$|Q| \cdot p(n) + |\Sigma| \cdot p^2(n)$ Propositionen!

Fakten: $\rightarrow B(0, 1, x_1)$ ... $\rightarrow B(0, n, x_n)$

$\rightarrow B(0, i, \square)$ für $n < i \leq p(n)$

$\rightarrow Z(0, q_0)$

Klauseln: für $i,t \in p(n)$, $(q,a,b,m,q') \in I$:

$$Z(t,q),\ B(t,i,a) \to Z(t+1,q')$$
$$Z(t,q),\ B(t,i,a) \to B(t+1,i,b)$$
} für $i = h(t,a)$

$$B(t,i,a) \to B(t+1,i,a) \quad \text{für } i \neq h(t,m)$$

$$Z(t,q) \to Z(t+1,q) \quad \text{für } q \in F.$$

Durch Induktion nach t zeigt man leicht:

$\vdash Z(t,q)$ gdw T nach t Schritten im Zustand q

$\vdash Z(t,i,a)$ gdw Bandzelle i nach t Schritten enthält a

Also gilt: $x \in A$ gdw $\vdash Z(p(n), Acc)$

Es ist leicht zu sehen, daß ein log-space-TM bei Input x diese Horn-Instanz ausgeben kann.

Also: $A \leq_{log}$ Horn. □

Horn kann auch als Spezialfall von SAT angesehen werden: Horn-SAT.

Klausel $p_1, \dots, p_k \to q \quad \hat{=} \quad \overline{p_1} \vee \dots \vee \overline{p_k} \vee q$

$\vdash p$ gdw Konjunktion dieser Klauseln mit $\overline{p}$ ist unerfüllbar

Quantifizierte Aussagenlogik

Formeln enthalten neben Variablen und den üblichen Konnektiven $\wedge, \vee, \neg$ auch Quantoren $\forall, \exists$:

$$F, G ::= 0 \mid 1 \mid x \mid \neg F \mid F \wedge G \mid F \vee G \mid \forall x.F \mid \exists x.G$$

Semantik der Quantoren ist:

$$\exists x.F \equiv F[x/0] \vee F[x/1]$$
$$\forall x.F \equiv F[x/0] \wedge F[x/1]$$

also eigentlich redundant, aber Elimination nur mit exponentiellem Blowup.

Problem QBF:

Gegeben: Formel F der quantifizierten Aussagenlogik ohne freie Variablen.

Frage: Ist F wahr?

Satz: QBF $\in$ PSPACE

Folgendes Programm entscheidet QBF:

```
input F
if eval(F) = 1
    then accept
    else reject

eval (F)
  case F of
    0: return 0
    1: return 1
    ¬F: return 1-eval(F)
```

```
G∧H   u := eval(G)
      v := eval(H)
      return u∧v

G∨H   u := eval(G)
      v := eval(H)
      return u∨v

∀x.G  u := eval(G[x/0])
      v := eval(G[x/1])
      return u∧v

∃x.G  u := eval(G[x/0])
      v := eval(G[x/1])
      return u∨v
```

Speicherplatzverbrauch:

Aufrufstacke: Bei jedem rek. Aufruf wird die Formel kürzer als Höhe $\leq n$

In jedem Stackframe wird gespeichert
u, v konstant viel
F $\leq n$

Insgesamt Speicherbedarf $\leq O(n^2)$ □

Tatsächlich geht es mit linearem Platz.
(Wortvercisung d. Variablen separat speichern, und statt der Teilformel nur Pointer darauf übergeben)

Satz: QBF ist PSPACE-schwer.

Sei $A \in$ PSPACE, akzeptiert von DTM T mit
Platz $= p(n)$ und Zeit $\leq 2^{p(n)}$.

Globaler Zustand S kann durch $O(p(n))$ viele boolesche Variablen repräsentiert werden, und es gibt Formeln

$I(S, x)$ wahr gdw. $S = S_0(x)$
$A(S)$ wahr gdw S akzeptierend
$T(S, S')$ wahr gdw $S \to_T S'$

(wie bei SAT NP-schwer)

Definiere nun Formeln T_j mit:

$T_j(S, S')$ wahr gdw Berechnung von T endet nach 2^j Schritten bei S'.

Dann ist

$$x \in A \text{ gdw } \exists S\, S'\; I(S,x) \wedge A(S') \wedge T_{p(n)}(S, S') \text{ wahr ist}$$

Bleibt T_j zu konstruieren so dass $|T_j| \leq O(q(n))$

Konstruktion der Formeln T_j induktiv nach j:

$$T_0(S, S') := T(S, S')$$

Man möchte eigentlich Iterieren (wie beim Algorithmus f. REACH)

$$T_{j+1}(S\ S') := \exists M\ (T_j(S, M) \wedge T_j(M, S'))$$

Da T_j zweimal verwendet wird, ist T_{j+n} exponentiell groß in j ☹.

Bessere Lösung

$$T_{j+n}(S, S') := \exists M\ \forall X, Y$$
$$(X = S \wedge Y = M) \vee (X = M \wedge Y = S')$$
$$\rightarrow T_j(X, Y)$$

Nun ist $|T_{j+1}| = |T_j| + O(p(n))$,

also $|T_{p(n)}| = O(p(n)^2)$ ∎

4.3 Nichtdeterministischer Platz und Komplementierung

Während NP = co-NP offen ist, ist für nichtdet. Platzklassen das Problem der Abgeschlossenheit unter Komplement gelöst:

Satz von Immerman - Szelepcsényi: (1987)

$$NL = co\text{-}NL$$

Es folgt (wie bei Savitch) dasselbe auch für größere Platzschranken, insbesondere nichtdet LINEAR SPACE = kontextsensitive Sprachen!

Da REACH NL-vollständig ist, reicht es zu zeigen:

Satz: $\overline{\text{REACH}} \in NL$

Problem $\overline{\text{REACH}}$

Gegeben: gerichteter Graph $G = (V, E)$, $s, t \in V$ $\quad |V| = n$

Frage: ist t nicht von s erreichbar?

Beweis:

Sei reachable(v, k) nichtdet. Algorithmus, der in $O(\log n)$ Platz entscheidet, ob v von s in $\leq k$ Schritten erreichbar ist.

Idee: Teste für jeden Knoten $v \in V$, ob v von s erreichbar ist, falls ja, teste ob $v = t$. Falls bis zum Ende t nicht gefunden, akzeptiere.

Programm:

```
found := false
z := 0
for v∈V do                                  ← nichtdet.!
    if reachable(v,n) then
        z := z+1
        if v=t then found := true
if (not found) and z = r(n)
    then accept
    else reject
```

Problem: wie stellt man sicher, dass man nur in den Zweigen akzeptiert, wo reachable(v,n) richtig gerechnet hat für alle v∈V ?

Sei $r(k) := \#\{v \in V ;$ v erreichbar von s in $\leq k$ Schritten $\}$

Lösung: zähle mit, dass r(n) erreichbare Knoten gefunden werden. Zähler z.

Bleibt zu zeigen, dass der Wert r(n) nichtdet. mit logarithmischem Platz berechenbar ist.
Dh. in jeder akzeptierenden Berechnung muss der gleiche Wert ausgegeben werden.

Dazu reicht es, induktiv r(k+1) zu berechnen, wenn man r(k) kennt.

Idee ist analog zum obigen Programm:
Schleife über alle v∈V,
bei v∈V mit Zähler erhöht,
falls ein u∈V gefunden wird, das in k Schritten erreichbar ist. Zur Überprüfung der Richtigkeit des nichtdet. Tests reachable(u,k) verwende den Zähler r(k).

Programm zur Berechnung von $r(k+1)$ aus $r(k)$:

```
r := 0
for v ∈ V do
    z := 0
    found := false
    for u ∈ V do
        if reachable(u,k) then
            z := z+1
            if (u,v) ∈ E then found := true
    if z = r(k)
        then if found then r := r+1
        else reject
return r
```

Korrektheit ist offensichtlich.

Analyse des Speicherplatzbedarfs:

Hauptprogramm speichert
z, v, $r(u)$ (s.unten)

Unterprogramm speichert:
alter Wert $r(k)$
neuer Wert r
v, z, u

} 7 Variablen, je $\log n$ Bits

$\rightsquigarrow$ $7 \cdot \log n$

Es folgt $\overline{\text{REACH}} \in NL$ □

Bemerke: Nichtdeterminismus stark ausgenutzt, es gibt sehr viele erfolglose Berechnungen, nur wenige erfolgreiche.