

# Komplexitätstheorie

SoSe 2018  
Thomas Schneider

## Kapitel 5: Platzkomplexität

Homepage der Vorlesung: <http://tinyurl.com/ss18-kt>

## Einleitung

Ab jetzt betrachten wir **Platzkomplexität**.

### Genauer:

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

### Platzverbrauch kann in der Praxis so kritisch sein wie Zeitverbrauch:

- Geringer Zeitverbrauch hilft nicht, wenn Speicher vorher erschöpft.
- Dies passiert nicht selten bei Algorithmen, die schwierige Probleme lösen.

### Wesentlicher Unterschied zur Zeitkomplexität:

Platz kann man wiederverwenden, Zeit nicht.

## Kapitel 5

**NEXT**



### 5.1 Platzkomplexitätsklassen

- 5.2 Polynomieller Platzverbrauch (PSPACE)
- 5.3 Zeitkomplexität versus Platzkomplexität
- 5.4 Der Satz von Savitch
- 5.5 PSPACE-Härte und -Vollständigkeit
- 5.6 Logarithmischer Platz (LogSpace)
- 5.7 ... und Nichtdeterminismus (NLogSpace)
- 5.8 ... und Komplemente (coNLogSpace)
- 5.9 Platzhierarchiesatz

## Platzkomplexität

Um Eingabe und „Zwischenspeicher“ zu trennen, nehmen wir für alle DTMs / NTMs ein **dediziertes Eingabeband** an:

- Eingabeband wird **nur gelesen**, nicht beschrieben. (d. h. TM muss per Konvention das auf diesem Band gelesene Symbol wieder zurückschreiben)
- TM kann sich auf dem Eingabeband nach links und rechts bewegen, also die Eingabe auch **mehrfach lesen**.
- Im Zusammenhang mit Platzkomplexität ist eine **k-Band-TM** also eine 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 5.1 (platzbeschränkte TM)

Für eine  $k$ -Band-DTM und  $w \in \Sigma^*$  schreiben wir  $\text{space}_M(w) = n$ , wenn die (eindeutige) Berechnung von  $M$  auf  $w$   $n$  **Bandzellen** verwendet. alle Bänder aufsummiert

Für eine  $k$ -Band-NTM und  $w \in \Sigma^*$  ist  $\text{space}_M(w)$  analog definiert, aber bezüglich der Berechnung von  $M$  auf  $w$ , die den meisten Platz verbraucht.

Sei  $s : \mathbb{N} \rightarrow \mathbb{N}$  monoton wachsende Funktion.

$M$  ist  $s$ -**platzbeschränkt**, wenn  $\text{space}_M(w) \leq s(n)$  für alle  $w$  der Länge  $n$ .

Nun ist:

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

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

T5.1

## Mehrband TMs

Auch bei Platzkomplexität brauchen wir uns um die **Anzahl Bänder** **keine Gedanken** zu machen:

### Satz 5.2 (Bandreduktion, Platzkomplexität)

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

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

**Beachte:** im Gegensatz zu Zeitkomplexität **kein quadratischer Blowup!**

### Definition 5.3 (DSpace, NSpace)

$$\text{DSpace}(s) := \bigcup_{k \geq 1} \text{DSpace}_k(s); \quad \text{NSpace}(s) := \bigcup_{k \geq 1} \text{NSpace}_k(s)$$

## Kapitel 5

5.1 Platzkomplexitätsklassen

**NEXT** 

**5.2 Polynomieller Platzverbrauch (PSpace)**

5.3 Zeitkomplexität versus Platzkomplexität

5.4 Der Satz von Savitch

5.5 PSpace-Härte und -Vollständigkeit

5.6 Logarithmischer Platz (LogSpace)

5.7 ... und Nichtdeterminismus (NLogSpace)

5.8 ... und Komplemente (coNLogSpace)

5.9 Platzhierarchiesatz

## 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 3-GHz-Prozessor
- kubischer Platzverbrauch: 1 Terabyte Speicher nötig!

**Entsprechende Komplexitätsklasse:**

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

## 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 5.4 (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  $\varphi$  eine AL-Formel mit Variablen aus der Menge  $\{p_1, \dots, p_n\}$ .AL-Formeln dürfen hier auch die **Konstanten 0,1** enthalten.

T5.2

## Definition 5.5 (Gültigkeit einer QBF)

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

- $n = 0$  und  $\varphi$  (welches dann variabelnfrei ist) zu 1 auswertet.
- $Q_1 = \exists$  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

**Dabei steht**  $\varphi[p/0]$  für „ $\varphi$  mit  $p$  durch 0 ersetzt“,  
 $\varphi[p/1]$  für „ $\varphi$  mit  $p$  durch 1 ersetzt“

z. B.  $(p_1 \vee p_2) \rightarrow (p_2 \vee p_3)[p_2/0] = (p_1 \vee 0) \rightarrow (0 \vee p_3)$ **Überprüfen von Gültigkeit:** durch **Auswertungsbaum**

T5.3

## Auswertungs bäume

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

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

**Erfolgreicher Auswertungsbaum:**

- „Hochpropagieren“ der Auswertung:
- $\exists$ -Quantoren  $\hat{=}$  **or**;  $\forall$ -Quantoren  $\hat{=}$  **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 5.6

QBF ist in PSpace.

**Idee:**

- Verwende rekursive Prozedur, die sich aus Def. von Gültigkeit ergibt.
- Rekursionstiefe ist **linear**, für jeden Rekursionsschritt wird linear viel Speicher gebraucht.
- Ergibt **quadratischen Platzbedarf**: man kann Speicher **wiederverwenden**, wenn man in anderen Ast des (exp. großen) Rekursionsbaums absteigt.

T5.4

5.1 Platzkomplexitätsklassen

5.2 Polynomieller Platzverbrauch (PSpace)

**NEXT**



**5.3 Zeitkomplexität versus Platzkomplexität**

5.4 Der Satz von Savitch

5.5 PSpace-Härte und -Vollständigkeit

5.6 Logarithmischer Platz (LogSpace)

5.7 ... und Nichtdeterminismus (NLogSpace)

5.8 ... und Komplemente (coNLogSpace)

5.9 Platzhierarchiesatz

**Natürliche Frage:**

- Wie verhält sich PSpace zu P, NP und ExpTime?
- (allgemeiner) Wie verhält sich Platzkomplexität zu Zeitkomplexität?

Wir geben im Folgenden **2 Antworten**.

## Zeit versus Platz (1)

„Was man in **Zeit**  $f$  berechnen kann, kann man auch in **Platz**  $f$  berechnen, sogar für **nichtdeterministische Zeit** und **deterministischen Platz**.“

### Theorem 5.7

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

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

**Beweis.**

Sei  $L \in \text{NTime}(f)$  und  $M$   $f$ -zeitbeschränkte NTM mit  $L(M) = L$ .

Für Konfiguration  $uqav$  von  $M$  und Übergang  $\alpha = (q, a, q', a', B) \in \Delta$  sei **Schritt**( $uqav, \alpha$ ) die Nachfolgekonfig. von  $uqav$  bei Anwendung von  $\alpha$ .

Folgender Algorithmus gibt **true** zurück  
gdw. es eine Berechnung von  $M$  gibt, die mit  $uqav$  startet und **akzeptiert**.

## Zeit versus Platz (1)

**Function**  $\text{acc}_M(uqav)$ :

```

if  $q = q_{\text{acc}}$  then return true
else if  $q = q_{\text{rej}}$  then return false
else
  forall  $\alpha = (q, a, q', a', B) \in \Delta$  do
    if  $\text{acc}_M(\text{Schritt}(uqav, \alpha))$  then return true
  return false

```

**Leicht zu sehen:**

$\text{acc}_M(q_0w)$  antwortet **true** gdw.  $M$  akzeptiert  $w$

**Platzverbrauch:**

- ist zunächst in  $\mathcal{O}(f(n)^2)$
- durch geschicktere Implementierung: wie gewünscht in  $\mathcal{O}(f(n))$

**T5.5**

## Zeit versus Platz (1)

### Theorem 5.7 (wiederholt)

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

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

### Korollar 5.8

$$\text{P} \subseteq \text{NP} \subseteq \text{PSpace}$$

Echtheit vermutet, aber unbewiesen (NP-versus-PSpace-Problem)

## Zeit versus Platz (2)

Für Antwort Nr. 2 verwenden wir Konfigurationsgraphen.

### Definition 5.9 (Konfigurationsgraph)

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

$\text{Conf}_{M,s}$  ist 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

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

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

- Es gibt laut Def. NTM sowieso nur einen akzeptierenden Zustand.
- Vor Anhalten kann TM alle Bänder löschen und Köpfe ganz nach links fahren.

## Zeit versus Platz (2)

Akzeptanz = Erreichbarkeit im Konfigurationsgraphen:

### Lemma 5.10

Sei  $M$   $s$ -platzbeschränkte NTM mit akzeptierender Konfiguration  $C_{\text{acc}}$  und  $w$  Eingabe der Länge  $n$ .

Dann gilt:  $w \in L(M)$  gdw.  $C_{\text{acc}}$  von  $q_0w$  aus in  $G_{M,s(n)}$  erreichbar

Der Konfigurationsgraph ist exponentiell groß:

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

mögliche Zustände, Bandinhalte, Kopfpositionen

## Zeit versus Platz (2)

Benötigen außerdem Platzkonstruierbarkeit:

### Definition 5.11 (Platzkonstruierbarkeit)

Eine Funktion  $s : \mathbb{N} \rightarrow \mathbb{N}$  heißt platzkonstruierbar, wenn es eine terminierende DTM  $M$  gibt, so dass für alle  $w \in \Sigma^*$  gilt:

$$\text{space}_M(w) = s(|w|)$$

Betrachte Platzkonstruierbarkeit wieder 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

T5.6

## Zeit versus Platz (2)

### Nun Antwort Nr. 2:

„Was man in Platz  $f$  berechnen kann, kann man auch in **Zeit**  $2^{O(f)}$  berechnen, sogar für **nichtdeterministischen Platz und deterministische Zeit**.“

### Theorem 5.12

Für alle **platzkonstruierbaren** Funktionen  $f : \mathbb{N} \rightarrow \mathbb{N}$  gilt:

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

### Beweis.

Sei  $L \in \text{NSpace}(f)$  und  $M$   $f$ -platzbeschränkte NTM mit  $L(M) = L$ .

Konstruieren  $2^{O(f)}$ -zeitbeschränkte DTM  $M'$  mit  $L(M') = L$  wie folgt:

0. Eingabe  $w$  mit  $|w| = n$
1. Generiere deterministisch den Konfigurationsgraphen  $G_{M,f(n)}$  **T5.7**
2. Entscheide, ob die (eindeutig bestimmte) akzeptierende Konfiguration von  $q_0w$  aus erreichbar ist

### Laufzeitanalyse:

**T5.8**

## Zeit versus Platz (2)

### Theorem 5.12 (wiederholt)

Für alle **platzkonstruierbaren** Funktionen  $f : \mathbb{N} \rightarrow \mathbb{N}$  gilt:

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

### Korollar 5.13

$$P \subseteq NP \subseteq \text{PSpace} \subseteq \text{ExpTime}$$

Echtheit vermutet, aber unbewiesen (**PSpace-versus-ExpTime-Problem**)

Auch für **deterministischen Platz** ist keine bessere Schranke bekannt.

## Kapitel 5

- 5.1 Platzkomplexitätsklassen
- 5.2 Polynomieller Platzverbrauch (PSpace)
- 5.3 Zeitkomplexität versus Platzkomplexität
- NEXT** → **5.4 Der Satz von Savitch**
- 5.5 PSpace-Härte und -Vollständigkeit
- 5.6 Logarithmischer Platz (LogSpace)
- 5.7 ... und Nichtdeterminismus (NLogSpace)
- 5.8 ... und Komplemente (coNLogSpace)
- 5.9 Platzhierarchiesatz

## Der Satz von Savitch

Definition von PSpace ähnelt der von P (Platz statt Zeit)

**Naheliegend:** nichtdeterministisches Pendant (ähnlich NP)

Analog zu alternativer Charakterisierung von NP:

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

### Interessanter Gegensatz:

- „P versus NP“ ist wichtigstes offenes Problem der Informatik
- „PSpace versus NPSpace“ wurde 1970 von Walter Savitch **gelöst**: es gilt **PSpace = NPSpace!**

## Der Satz von Savitch

**Zu zeigen:**  $\text{NPSpace} \subseteq \text{PSpace}$

**Naiver Versuch mittels Konfigurationsgraphen:**

- Wenn  $L \in \text{NPSpace}$ , dann gibt es  $p$ -platzbeschränkte NTM  $M$  mit  $L(M) = L$  und  $p$  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)

## Der Satz von Savitch

**Theorem 5.14 (Savitch 1970)**

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

**Idee:**

- Prädikat  $\text{Pfad}(C, C', i)$  ist wahr gdw. es Pfad in  $G_{M,s(n)}$  gibt von  $C$  nach  $C'$  mit Länge **max.  $2^i$** .
- Wir interessieren uns also für  $\text{Pfad}(q_0w, C_{\text{acc}}, \log(|\text{Conf}_{M,s(n)}|))$ .
- 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  $\text{Pfad}(C, C', i)$  zu entscheiden,
  - betrachte alle möglichen Mittelpunkte  $C_m$ ;
  - entscheide rekursiv  $\text{Pfad}(C, C_m, i-1) \wedge \text{Pfad}(C_m, C', i-1)$ .
- Rekursionstiefe  $\log(|\text{Conf}_{M,s(n)}|) \in \mathcal{O}(s(n))$ ;  
pro rekursiven Abstieg Speicherbedarf  $\mathcal{O}(s(n))$  auf Stack  
 $\leadsto$  **insgesamt  $\mathcal{O}(s(n)^2)$  Speicher**

## Der Satz von Savitch

**Beweis.**

Sei  $M$   $s$ -platzbeschränkte NTM mit  $s$  platzkonstruierbar.

Konstruiere DTM  $M'$  wie folgt.

0. Eingabe  $w$  mit  $|w| = n$  und  $s(n) = k$
1. Erzeuge  $k$  ( $s$  ist platzkonstruierbar!)
2. Berechne  $N := |\text{Conf}_{M,k}|$  in binär, basierend auf  $k$
3. Führe  $\text{path}(q_0w, C_{\text{acc}}, \lceil \log N \rceil)$  aus:

## Der Satz von Savitch

**Function**  $\text{path}(C, C', i)$ :

```
if  $i = 0$  then
  if  $C = C'$  or  $C \vdash_M C'$  then return true else return false
forall  $C_m \in \text{Conf}_{M,k}$  do
  if  $\text{path}(C, C_m, i-1)$  and  $\text{path}(C_m, C', i-1)$  then return true
return false
```

**Platzanalyse:**

**T5.9**

## Konsequenzen aus dem Satz von Savitch

... für PSpace:

Korollar 5.15

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

(Natürlich sind die **deterministischen** Platzkomplexitätsklassen wie die **det.** Zeitkomplexitätsklassen unter Komplement abgeschlossen.)

Wegen Korollar 5.15 werden NPSPACE und coNPSPACE **nicht als eigenständige Komplexitätsklassen betrachtet.**

... für größere Komplexitätsklassen:

$$\text{ExpSpace} := \bigcup_{i \geq 1} \text{DSpace}(2^{\mathcal{O}(n^i)}) \quad \text{NExpSpace} := \bigcup_{i \geq 1} \text{NSpace}(2^{\mathcal{O}(n^i)})$$

Korollar 5.16

$\text{ExpSpace} = \text{NExpSpace} = \text{coNExpSpace}$

## Konsequenzen aus dem Satz von Savitch

Andere Sicht auf den Beweis:

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

Korollar 5.17

$\text{GAP} \in \text{DSpace}((\log n)^2)$

T5.10

## Konsequenzen aus dem Satz von Savitch

Weiterer Nutzen des Satzes von Savitch:

Dürfen beim Finden von PSPACE-Algorithmen o. B. d. A. **Nichtdeterminismus** benutzen!

Beispiel:

Universalitätsproblem für nichtdeterministische endliche Automaten (NEAs)

Gegeben NEA  $\mathcal{A}$ , ist  $L(\mathcal{A}) = \Sigma^*$  ?

Lemma 5.18

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

T5.10

Theorem 5.19

Das Universalitätsproblem für NEAs ist in PSPACE.

## Konsequenzen aus dem Satz von Savitch

Theorem 5.19

Das Universalitätsproblem für NEAs ist in PSPACE.

Idee:

- Rate Wort der Länge  $\leq 2^{|Q|}$ ;  
  behalte zu jedem Zeitpunkt nur je 1 Symbol im Speicher.
- Verfolge währenddessen alle Läufe von  $\mathcal{A}$ ;  
  stelle sicher, dass **keiner** davon einen akzept. Zustand erreicht.

Algorithmus:

Function  $\text{univ}((Q, \Sigma, q_0, \Delta, F))$ :

```
Rate  $n \leq 2^{|Q|}$ 
 $P \leftarrow \{q_0\}$ 
for  $i = 1, \dots, n$  do
  Rate  $a \in \Sigma$ 
   $P \leftarrow \{q \in Q \mid \exists p \in P : (p, a, q) \in \Delta\}$ 
if  $P \cap F \neq \emptyset$  then reject else accept
```

... läuft in coNPSPACE.

Nach Satz von Savitch  
in coNPSPACE = PSPACE.



- 5.1 Platzkomplexitätsklassen
- 5.2 Polynomieller Platzverbrauch (PSpace)
- 5.3 Zeitkomplexität versus Platzkomplexität
- 5.4 Der Satz von Savitch
- NEXT** → **5.5 PSpace-Härte und -Vollständigkeit**
- 5.6 Logarithmischer Platz (LogSpace)
- 5.7 ... und Nichtdeterminismus (NLogSpace)
- 5.8 ... und Komplemente (coNLogSpace)
- 5.9 Platzhierarchiesatz

## Zur Erinnerung:

### Theorem 5.20

$P \subseteq NP \subseteq PSpace \subseteq ExpTime$

**Echtheit unbekannt!**

Wie bei P vs NP:  
man behilft sich mit dem Begriff der **Härte** und **Vollständigkeit**,  
basierend auf Polynomialzeit-Reduktionen.

## Einfache Beobachtung:

### Lemma 5.21

PSpace ist abgeschlossen unter Polynomialzeit-Reduktionen:  
Wenn  $L' \leq_p L$  und  $L \in PSpace$ , dann  $L' \in PSpace$ .

## Definition 5.22 (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 5.23

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.

Wir verwenden QBFs, bei denen Quantoren **nicht unbedingt Präfix** sind.

## Definition 5.24 (generelle QBF)

Sei  $AV$  abzählbar unendliche Menge von *Aussagenvariablen*.

Menge der *generellen QBFs* ist kleinste Menge, für die gilt:

- 0,1 sind generelle QBFs.
- Jedes  $p \in AV$  ist generelle QBF
- Wenn  $\varphi, \psi$  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*.

### Definition 5.25 (Semantik genereller QBF)

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

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

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

Dabei ist  $V[p/i](p') = \begin{cases} i & \text{falls } p' = p \\ V(p') & \text{sonst} \end{cases}$  („ $V[p/i]$  erhält man aus  $V$ ,  
indem man  $p$  auf  $i$  setzt.“)

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

**Nicht schwer zu zeigen:**

### Lemma 5.26

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

T5.12

Können ab jetzt also o. B. d. A. **generelle QBF-Sätze** verwenden.

### Theorem 5.27

QBF ist PSpace-vollständig.

#### Beweis.

Wegen Theorem 5.6 genügt es, PSpace-Härte zu zeigen.

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

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

**Ziel:** gegeben  $w$ , finde **generellen QBF-Satz**  $\varphi_w$ , so dass

1.  $\varphi_w$  gültig gdw.  $M$  akzeptiert  $w$  und
2.  $\varphi_w$  in Polyzeit aus  $w$  konstruierbar

**Ansatz 1:** kodiere „Matrix“ der Berechnung von  $M$  (siehe Satz von Cook)  
 $\rightsquigarrow$  **schlätzt fehl:**  $M$  hat u. U. **exponentielle** Laufzeit

**Ansatz 2:** reformuliere Beweis des Satzes von Savitch  
„in der Sprache von QBF“

### Reformulierung Beweis Satz von Savitch „in der Sprache von QBF“:

Sei  $|w| = n$ .

$M$  ist  $p(n)$ -platzbeschränkt

$\Rightarrow M$  ist  $2^{q(n)}$ -zeitbeschränkt für **Polynom**  $q$   
(Theorem 5.12)

**Konkreteres Ziel:**  $\varphi_w$  beschreibt Prädikat **Pfad** $(q_0w, C_{acc}, q(n))$   
und dessen rekursive Berechnung

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

## PSpace-Härte von QBF

Wir repräsentieren **Konfigurationen** von  $M$  durch folgende Variablen:

- $Z_q$  für jedes  $q \in Q$  beschreibt aktuellen 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:  $\bar{C}$

Um über mehrere Konfigurationen zu sprechen:

zusätzliche Tupel  $\bar{C}', \bar{C}''$  (alle Variablen mit ' bzw. '')

Diese Formel garantiert, dass die Variablen „legale“ **Konfiguration** beschreiben:

$$\begin{aligned} \psi_{\text{conf}}(\bar{C}) := & \bigvee_{q \in Q} \left( Z_q \wedge \bigwedge_{q' \neq q} \neg Z_{q'} \right) \quad \wedge \quad \bigvee_{i \leq p(n)} \left( K_i \wedge \bigwedge_{i' \neq i} \neg K_{i'} \right) \\ & \wedge \quad \bigwedge_{i \leq p(n)} \bigvee_{a \in \Gamma} \left( B_{a,i} \wedge \bigwedge_{a' \neq a} \neg B_{a',i} \right) \end{aligned}$$

## PSpace-Härte von QBF

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

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

$$\psi_{\text{next}}(\bar{C}, \bar{C}') := \psi_{\text{conf}}(\bar{C}) \wedge \psi_{\text{conf}}(\bar{C}') \quad \wedge$$

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

Wir definieren nun **Pfad**( $C, C', i$ ) durch Formeln  $\psi_{\text{reach}}^i(\bar{C}, \bar{C}')$

**Für den Fall  $i = 0$ :**

$$\psi_{\text{reach}}^0(\bar{C}, \bar{C}') := \psi_{\text{eq}}(\bar{C}, \bar{C}') \vee \psi_{\text{next}}(\bar{C}, \bar{C}')$$

wobei  $\psi_{\text{eq}}(\bar{C}, \bar{C}')$  sagt, dass  $\bar{C}$  und  $\bar{C}'$  identisch sind

## PSpace-Härte von QBF

**Für  $i > 0$**  ist es verlockend zu schreiben:

$$\psi_{\text{reach}}^i(\bar{C}, \bar{C}') := \exists \bar{C}'' \psi_{\text{reach}}^{i-1}(\bar{C}, \bar{C}'') \wedge \psi_{\text{reach}}^{i-1}(\bar{C}'', \bar{C}')$$

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

**Universelle Quantoren helfen:**

$$\begin{aligned} \psi_{\text{reach}}^i(\bar{C}, \bar{C}') := & \exists \bar{C}'' \forall \bar{K} \forall \bar{K}' \\ & ((\psi_{\text{eq}}(\bar{C}, \bar{K}) \wedge \psi_{\text{eq}}(\bar{C}'', \bar{K}')) \vee \\ & (\psi_{\text{eq}}(\bar{C}'', \bar{K}) \wedge \psi_{\text{eq}}(\bar{C}', \bar{K}')))) \rightarrow \psi_{\text{reach}}^{i-1}(\bar{K}, \bar{K}') \end{aligned}$$

**Leicht zu finden:**

- Formel  $\psi_{\text{input}}^w(\bar{C})$  – sagt, dass  $\bar{C}$  initiale Konfiguration für Eingabe  $w$  ist
- Formel  $\psi_{\text{acc}}(\bar{C})$  – sagt, dass  $\bar{C}$  akzeptierend ist.

## PSpace-Härte von QBF

**Zur Erinnerung:**

$M$  ist  $p(n)$ -platzbeschränkt

$\Rightarrow M$  ist  $2^{q(n)}$ -zeitbeschränkt für **Polynom  $q$**

**Die Reduktions-QBF ist nun:**

$$\psi_w := \exists \bar{C} \exists \bar{C}' \left( \psi_{\text{input}}^w(\bar{C}) \wedge \psi_{\text{acc}}(\bar{C}') \wedge \psi_{\text{reach}}^{q(|w|)}(\bar{C}, \bar{C}') \right)$$

**Lemma 5.28**

$M$  akzeptiert  $w$  gdw.  $\varphi_w$  gültig.

QBF ist ein bisschen wie „SAT für PSpace“.

## Natürlichere PSpace-vollständige Probleme:

- **Universalitätsproblem für NEAs**
- **Schnittleerheitsproblem für NEAs:**  
gegeben NEAs  $\mathcal{A}$ ,  $\mathcal{A}'$ , 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)

- 5.1 Platzkomplexitätsklassen
- 5.2 Polynomieller Platzverbrauch (PSpace)
- 5.3 Zeitkomplexität versus Platzkomplexität
- 5.4 Der Satz von Savitch
- 5.5 PSpace-Härte und -Vollständigkeit
- NEXT** → **5.6 Logarithmischer Platz (LogSpace)**
- 5.7 ... und Nichtdeterminismus (NLogSpace)
- 5.8 ... und Komplemente (coNLogSpace)
- 5.9 Platzhierarchiesatz

# 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 Theorem 5.12 ( $\text{NSpace}(f) \subseteq \text{DTime}(2^{\mathcal{O}(f)})$ ) folgt:

### Theorem 5.29

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

# LogSpace

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

### Ideen:

- Zähle Länge der Eingabe in binär mittels Zähler  $Z_1$ .
- Verwirf, wenn ungerade.
- Halbiere  $Z_1$ .
- Vergleiche 1. Symbol mit letztem, 2. mit zweitletztem etc.
- Zähler  $Z_2$  speichert zu vergleichende Position, zählt von 1 bis  $Z_1$ .
- Zähler  $Z_3$  wird benutzt, um diese Position zu finden, zählt von 1 bis  $Z_2$ .

### Intuitiv (und auch formalisierbar):

LogSpace beschreibt, was man mit einer **fixen Zahl binärer Zähler** berechnen/entscheiden 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

T5.13

**Komposition erfordert**

1. TM mit **Ausgabe** und
2. **Zwischenspeichern** von Ergebnissen

**Aber:**

- LogSpace-Algorithmus kann **polynomiell große Ausgabe** generieren.
- Ebenso für die Ausgabe des „Zwischenergebnisses“ bei Komposition.

→ Der für die Ausgabe benötigte Platz darf **nicht berücksichtigt werden**.

Entsprechendes Maschinenmodell: **LogSpace-Transduktoren**

## LogSpace-Transduktor

**Definition 5.30 (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  $f : \Sigma^* \rightarrow \Gamma^*$  ist *LogSpace-berechenbar*,

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

## LogSpace-Transduktor

**Zusammenhang LogSpace-Transduktoren und LogSpace-DTMs:**

**Definition 5.31**

Sei  $L \subseteq \Sigma^*$  eine Sprache. Die *charakteristische Funktion*  $\chi_L$  für  $L$  ist eine Abbildung  $\chi_L : \Sigma^* \rightarrow \{0, 1\}$  mit

$$\chi_L(w) = \begin{cases} 1 & \text{falls } w \in L \\ 0 & \text{sonst} \end{cases}$$

Man zeigt nun leicht:

**Theorem 5.32**

$L \in \text{LogSpace}$  gdw.  $\chi_L$  LogSpace-berechenbar ist.

## LogSpace: Kompositionalität

**Können nun 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

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

**Theorem 5.33**

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

T5.14

## Ein natürliches Problem in LogSpace:

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

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

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

T5.15

## 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. ➔ **kein LogSpace**

## 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 uns kein Kreis!

T5.16

T5.16

## Definition 5.34 (geordneter Pfad/Zyklus)

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

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

T5.17

## Beachte:

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

## Lemma 5.35

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.

T5.18

## Lemma 5.36

$G$  ist azyklisch gdw.  $G$  keinen **geordneten** Zyklus enthält.

T5.19

## Beachte:

- Das ist vollkommen unabhängig von den gewählten Ordnungen.
- Einen geordneten Zyklus kann eine **DTM** schrittweise ablaufen.

Wir können nun zeigen:

### Theorem 5.37

ACYC  $\in$  LogSpace

T5.20

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**.

Der Algorithmus aus dem Beweis von Lemma 5.37:

**forall** Kanten  $\{v_1, v_2\}$  **do**

```

   $u' \leftarrow v_1$ 
   $u \leftarrow v_2$ 
   $Z \leftarrow 0$ 
   $Z_{\max} \leftarrow 2|E| + 2$ 

```

**while**  $Z \leq Z_{\max}$  **do**


```

   $u_{\text{neu}} \leftarrow$  der eindeutig bestimmte Knoten, für den die Kante  $\{u, u_{\text{neu}}\}$ 
    bzgl. der lokalen Kantenordnung in  $u$  auf  $\{u', u\}$  folgt //  $u' \xrightarrow{k} u \xrightarrow{k+1} u_{\text{neu}}$ 
   $u' \leftarrow u$ 
   $u \leftarrow u_{\text{neu}}$ 
  if  $u = v_1$  then
    if  $u' \neq v_2$  then reject // geordneten Zyklus gefunden
    else  $Z \leftarrow Z_{\max}$  // abbrechen
   $Z \leftarrow Z + 1$ 
accept

```

Weitere Probleme in LogSpace z. B.:

- **Isomorphie von Bäumen** (gerichtet und ungerichtet)
- Entscheiden, ob ein Graph **zusammenhängend** ist (gerichtet / 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?)

- 5.1 Platzkomplexitätsklassen
- 5.2 Polynomieller Platzverbrauch (PSpace)
- 5.3 Zeitkomplexität versus Platzkomplexität
- 5.4 Der Satz von Savitch
- 5.5 PSpace-Härte und -Vollständigkeit
- 5.6 Logarithmischer Platz (LogSpace)
- NEXT**  **5.7 ... und Nichtdeterminismus (NLogSpace)**
- 5.8 ... und Komplemente (coNLogSpace)
- 5.9 Platzhierarchiesatz

## NLogSpace

Auch von LogSpace gibt es eine nichtdeterministische Variante:

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

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

### Theorem 5.38

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

Aus dem Satz von Savitch folgt **nicht**  $\text{LogSpace} = \text{NLogSpace}$ , sondern nur:

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

In der Tat ist nicht bekannt, ob  $\text{LogSpace} = \text{NLogSpace}$ .

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

## LogSpace-Reduktionen

**Polynomialzeit-Reduktionen sind hier nicht sinnvoll, denn:**

Für alle  $L, L' \in \text{NLogSpace}$  mit  $\underbrace{L'}_{L' \neq \emptyset \ \& \ L' \neq \Sigma^*}$  nicht-trivial gilt:  $L \leq_p L'$

(dieselbe Argumentation wie für P anstelle von NLogSpace)

### Definition 5.39 (LogSpace-Reduktion)

Eine Reduktion  $f$  von  $L$  auf  $L'$  heißt **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 5.40

1. „LogSpace-reduzierbar“ ist **transitive** Relation:

$$L_1 \leq_{\log} L_2 \text{ und } L_2 \leq_{\log} L_3 \text{ impliziert } L_1 \leq_{\log} L_3.$$

2. LogSpace, NLogSpace, P, NP und PSpace sind **abgeschlossen** unter LogSpace-Reduktionen:  $L' \in \mathcal{C}$  und  $L \leq_{\log} L'$  impliziert  $L \in \mathcal{C}$ .

### Beweis:

1. Folgt aus Kompositionalität (Theorem 5.33):  
Wenn  $f$  und  $g$  LogSpace-berechenbar, dann auch  $g(f)$ .
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.

Dies ist **keine einfache Frage**, denn

$$\leq_{\log} \stackrel{?}{=} \leq_p \text{ ist stark verwandt mit } \text{LogSpace} \stackrel{?}{=} \text{P}$$

**T5.21**



### Definition 5.41 (NLogSpace-Härte, NLogSpace-Vollständigkeit)

Problem  $L$  ist

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

### Theorem 5.42

GAP ist NLogSpace-vollständig.

T5.22

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

### 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.

### Weitere NLogSpace-vollständige Probleme z. B.:

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

- 5.1 Platzkomplexitätsklassen
- 5.2 Polynomieller Platzverbrauch (PSpace)
- 5.3 Zeitkomplexität versus Platzkomplexität
- 5.4 Der Satz von Savitch
- 5.5 PSpace-Härte und -Vollständigkeit
- 5.6 Logarithmischer Platz (LogSpace)
- 5.7 ... und Nichtdeterminismus (NLogSpace)

**NEXT**



**5.8 ... und Komplemente (coNLogSpace)**

5.9 Platzhierarchiesatz

## coNLogSpace

Da NLogSpace nichtdeterministische Klasse ist, macht es Sinn, coNLogSpace zu betrachten:

$$\text{coNLogSpace} := \{\bar{L} \mid L \in \text{NLogSpace}\}$$

Es war **lange Zeit unbekannt**, ob  $\text{NLogSpace} = \text{co-NLogSpace}$ .

**Theorem 5.43** (Immerman und Szelepcsényi, 1988)

$$\text{NLogSpace} = \text{coNLogSpace}$$

**Das Resultat lautet sogar:**

$$\text{Für alle } s \in \Omega(\log(n)) \text{ gilt } \text{NSpace}(s) = \text{coNSpace}(s)$$

## Satz von Immerman/Szelepcsényi

Es genügt zu zeigen:

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

Da GAP NLogSpace-vollständig, folgt  $\text{NLogSpace} = \text{coNLogSpace}$ .

**T5.23**

**Gesucht ist also:**

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  $c = \text{Anzahl der von } u_0 \text{ aus erreichbaren Knoten}$ , 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  $c = \text{Anzahl der von } u_0 \text{ aus erreichbaren Knoten}$ , ist  $v_0$  **nicht** erreichbar von  $u_0$ ?

**Maschine  $M_1$  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  $M_1$  akzeptiert, dann  $x = c$ ; also sind alle als **nicht** erreichbar geratenen Knoten (inkl.  $v_0$ ) wirklich **nicht** erreichbar.

**T5.24**

## Immerman/Szelepcsényi – Schritt 1

**Maschine  $M_1$  als Pseudocode-Algorithmus:**

**Function nonreach**( $G, u_0, v_0, c$ ):

```
x ← 0
forall v ∈ V \ {v_0} do
  rate b ∈ {0, 1}
  if b = 1 then // verifiziere
    x ← x + 1
    i ← 1
    u ← u_0
    while i < |V| and u ≠ v do
      rate u' mit (u, u') ∈ E
      u ← u'
      i ← i + 1
    if u ≠ v then reject
if x = c then accept else reject
```

**Platzanalyse:**

$v, u, i, x$  sind Binärzahlen mit Werten  $\leq |V|$   
 $\leadsto$  Platzbedarf  $\mathcal{O}(\log(n))$

### „false positives“:

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

⇒ 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“

⇒ werden identifiziert, indem **nach** der Iteration  
die Anzahl der als erreichbar **geratenen** Knoten  
mit der **bekannteren** Anzahl erreichbarer Knoten verglichen wird:

Wenn  $x < c$ , dann gab es „false negatives“, also verwirf.

Zeigen, dass  $c$  in NLogSpace berechnet werden kann.

### Maschine $M_2$ im Überblick:

- Eine Berechnung rechnet den korrekten Wert für  $c$  aus; alle anderen verwerfen (NTM).
- Dann kann man  $M_2$  der vorigen NTM  $M_1$  „vorschalten“.
- Sei  $V_i \subseteq V$  die Menge der Knoten, die von  $u_0$  aus **in  $\leq i$  Schritten** erreicht werden, und  $c_i = |V_i|$  für  $0 \leq i \leq |V|$ .
- **Wir benötigen  $c_{|V|}$**  — Bestimmung durch Berechnen von  $c_0, \dots, c_{|V|}$ .
- **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.

## Immerman/Szelepcsényi – Schritt 2

### Maschine $M_2$ als Pseudocode-Algorithmus:

```

c ← 1 // c = |V_{i-1}|   |V_0| = |\{u_0\}| = 1
for i = 1, ..., |V| do
  c' ← 0 // c' = |V_i|
  forall v ∈ V do
    x ← 0; f ← false // x: Anzahl Knoten in V_{i-1}
    // f: v erreichbar von V_{i-1} ?
    forall u ∈ V do
      rate e ∈ {0, 1} // e: u ∈ V_{i-1} ?
      if e = 1 then
        x → x + 1 // ↓ „false positive“ wie in Schritt 1
        if Raten des Pfades u ··· u_0 schlägt fehl then reject
        if (u, v) ∈ E then f ← true
    if x ≠ c then reject // „false negative“ wie in Schritt 1
    if f = true then c' ← c' + 1
  c ← c'
    
```

Am Ende enthält  $c$  das gewünschte Ergebnis. **Platzanalyse:** wie Schritt 1.

## Zurück zu Immerman/Szelepcsényi

### Das Resultat lautet sogar:

Für alle  $s \in \Omega(\log(n))$  gilt  $\text{NSpace}(s) = \text{coNSpace}(s)$

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
- ↔ 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

### Randbemerkung:

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

### Theorem 5.44

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  $\text{NSpace}(n) = \text{coNSpace}(n)$ .

- 5.1 Platzkomplexitätsklassen
- 5.2 Polynomieller Platzverbrauch (PSpace)
- 5.3 Zeitkomplexität versus Platzkomplexität
- 5.4 Der Satz von Savitch
- 5.5 PSpace-Härte und -Vollständigkeit
- 5.6 Logarithmischer Platz (LogSpace)
- 5.7 ... und Nichtdeterminismus (NLogSpace)
- 5.8 ... und Komplemente (coNLogSpace)

**NEXT**



**5.9 Platzhierarchiesatz**

## Platzhierarchiesatz

**Auch für Platzkomplexitätsklassen gibt es einen Hierarchiesatz:**

### Theorem 5.45 (Platzhierarchie)

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

Beweis analog zu Zeithierarchiesätzen, aber etwas einfacher.

### Konsequenzen z. B.:

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

## 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

